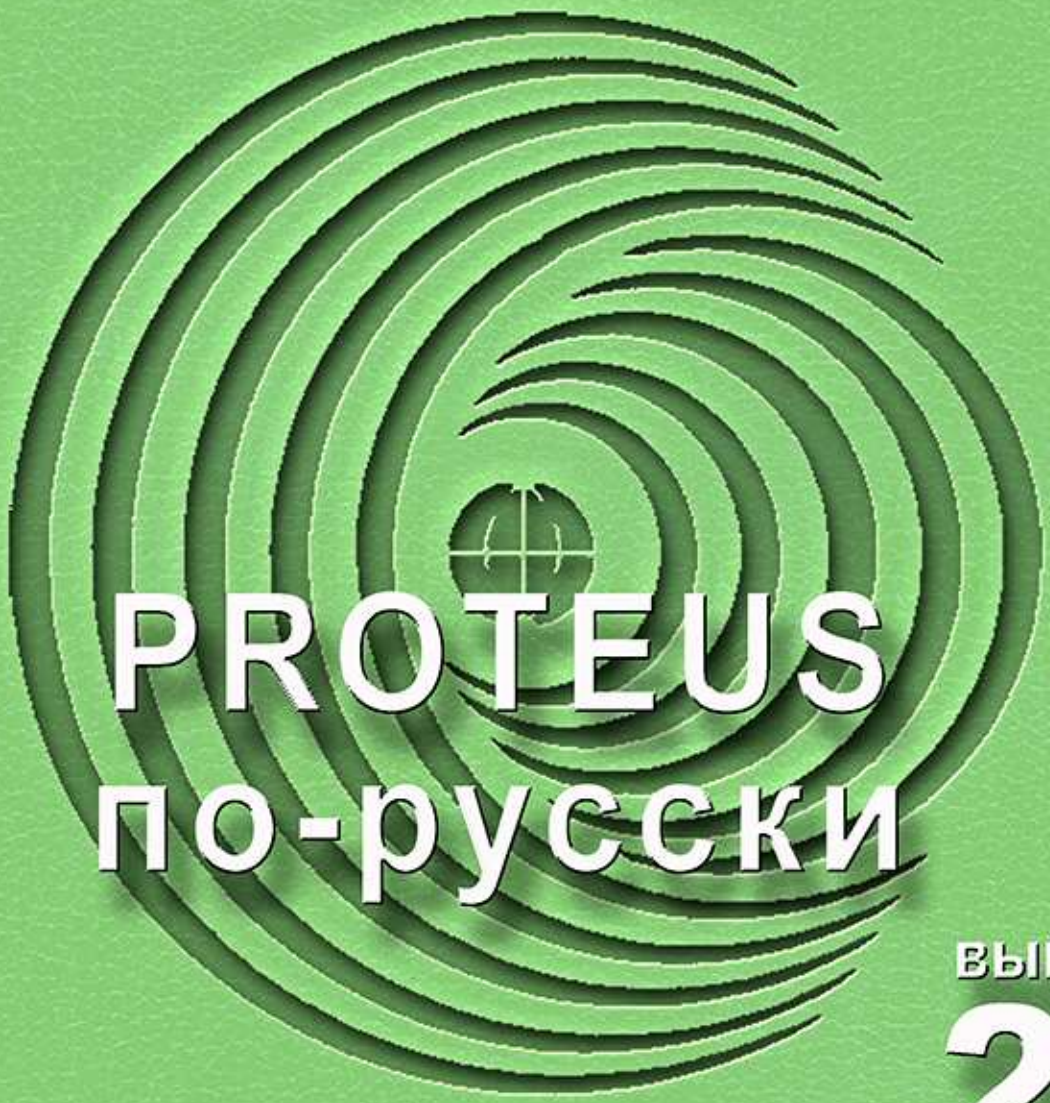
РАДИО-ЕЖЕГОДНИК



PROTEUS по-русски

ВЫПУСК

24

2013

РАДИО - ЕЖЕГОДНИК 2013 выпуск 24

ТЕМАТИЧЕСКИЙ ОБЗОР ПЕЧАТИ И ИНТЕРНЕТ-РЕСУРСОВ

ТЕМА НОМЕРА:

PROTEUS по-русски

В этом выпуске мы предлагаем познакомиться со всеми 4-я частями знаменитого «FAQ (ЧаВо) по PROTEUS для начинающих и не только» А. Христианчика, а также переводом на русский язык В.Н. Гололобова «Руководства пользователя программы ISIS Proteus VSM».

Выпускающий редактор: *С. Степанов*

Над выпуском работали: *С. Муратчаев*, *В. Гололобов*, *С. Скворцов*, *А. Максимов*, *В. Смирнов*, *А. Христианчик*

Художник: *О. Агафонов*

E-mail: ***radioyearbook@gmail.com***

Май 2013

Информационная поддержка: *Портал "РадиоЛоцман"* ***www.rlocman.ru***

Официальные версии журнала доступны для свободной загрузки:
www.rlocman.ru/radioyearbook

СОДЕРЖАНИЕ

PROTEUS по-русски

(в вопросах и ответах, примерах и пояснениях)

Рассказывая о программах моделирования, «Радиоежегодник» не мог оставить без внимания программу Proteus хотя бы по причине интереса, проявленного к ней на форуме портала KAZUS.RU.

Портал, собравший в своих рядах более полумиллиона пользователей, даёт им возможность обсудить все аспекты сегодняшней радиоэлектроники. Архив справочных данных на компоненты и статьи, книги и схемы, и просто разговор «по душам» - всё это привлекает и любителей, и профессионалов, словом всех, кого интересует электроника.

Обсуждение вопросов по работе с программой на форуме портала KAZUS.RU (http://kazus.ru/forums/forumdisplay.php?f=25) были собраны в единое целое модератором раздела Алексеем Христианчиком (Halex07), обрели ответы и появились в соответствующем разделе форума (http://kazus.ru/forums/showthread.php?t=13198). Затем появилась первая версия FAQ по Proteus в формате pdf. В этом выпуске мы предлагаем познакомиться со всеми 4-я частями знаменитого **FAQ (ЧаВо) по PROTEUS для начинающих и не только**.

Не потерял, в основном, актуальности и перевод В.Н. Гололобова **Руководства пользователя** к одной из ранних версий **программы ISIS Proteus VSM**. Он доступен для свободной загрузки с авторской страницы http://vgololobov.narod.ru/.

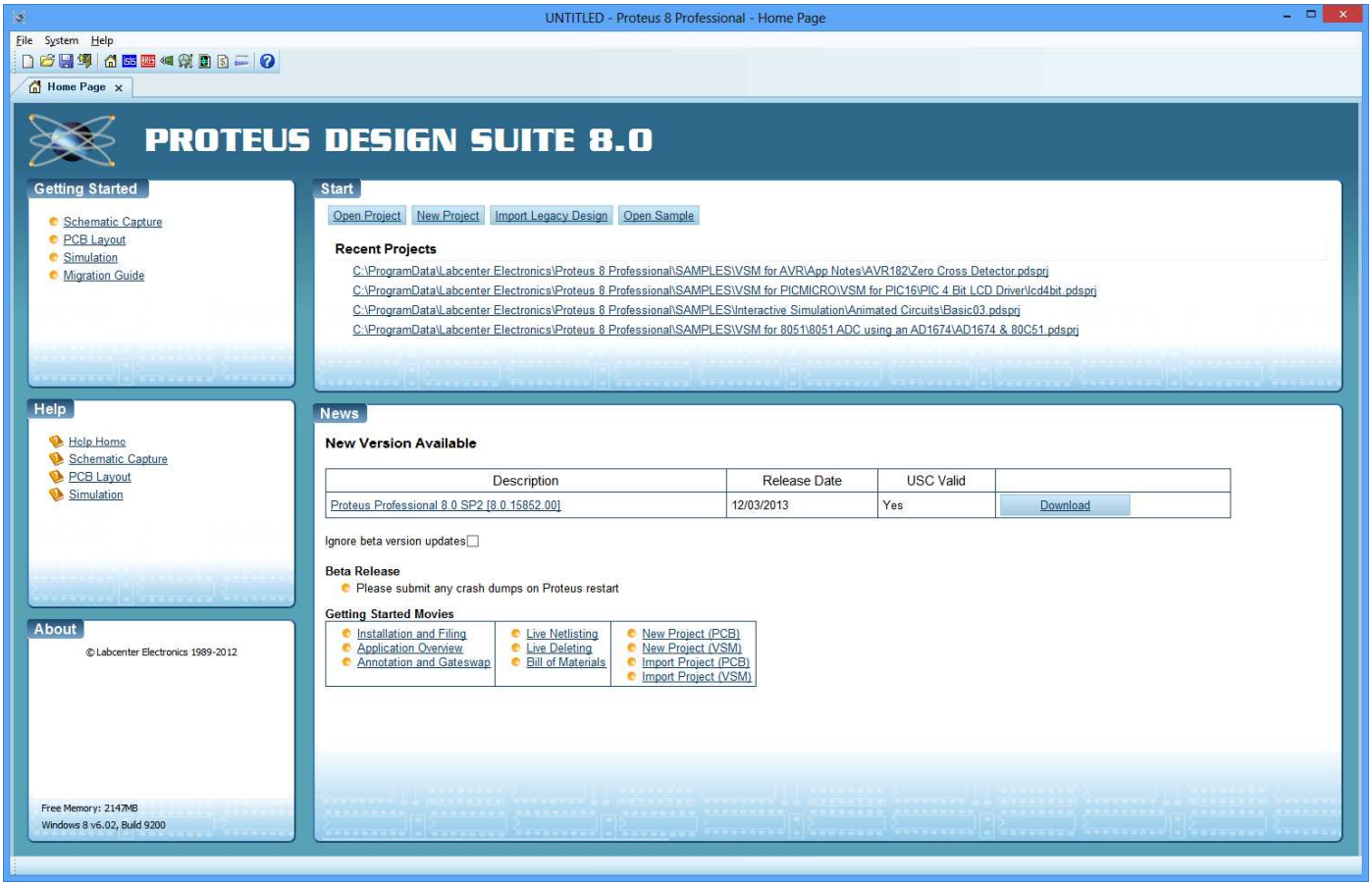
В журнале «Радио» за 2005 год №№4-6 была опубликована статья Алексея Максимова **Моделирование устройств на микроконтроллерах с помощью программы ISIS из пакета PROTEUS VSM**. В сети «витает» также и авторский вариант http://www.radioprog.ru/?p=91 **Симулятор-отладчик Proteus VSM на основе ядра SPICE3F5** (Максимов Алексей, ака Dosikus, ака Maksimus).

Благодаря поддержке ряда семейств микроконтроллеров и добавления в программу микросхем и компонентов, с которыми, как правило, микроконтроллеры работают, программа оказывается одной из немногих, в которых можно полностью проверить устройство на базе микроконтроллера. Этим она вызывает неподдельный интерес у радиолюбителей и специалистов. Программа коммерческая, однако, демонстрационная версия всегда доступна для скачивания на сайте производителя Labcenter Electronics:

http://www.labcenter.com/

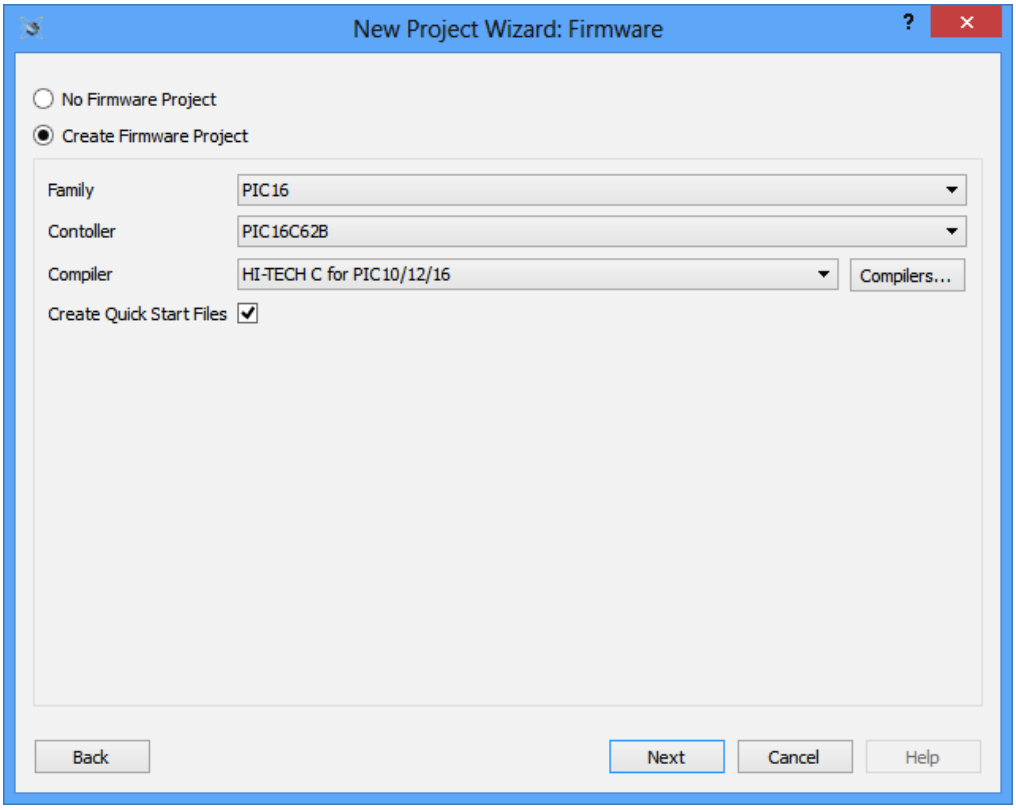
Но, как любая программа, Proteus развивается и совершенствуется. Сегодня доступна версия 8 этой программы. Вот, что можно прочитать о фирме, создавшей программу, на сайте:

LABCENTER Electronics Ltd. была основана в 1988 году, председатель правления и главный архитектор программного обеспечения Джон Джеймсон. С тех пор за 25 лет непрерывного развития, программа превратилась в один из наиболее экономически эффективных, полнофункциональных пакетов САПР на рынке программного обеспечения. С текущих продаж в более чем в 50 странах по всему миру LABCENTER постоянно расширяет и свой портфель продуктов, и клиентскую базу.

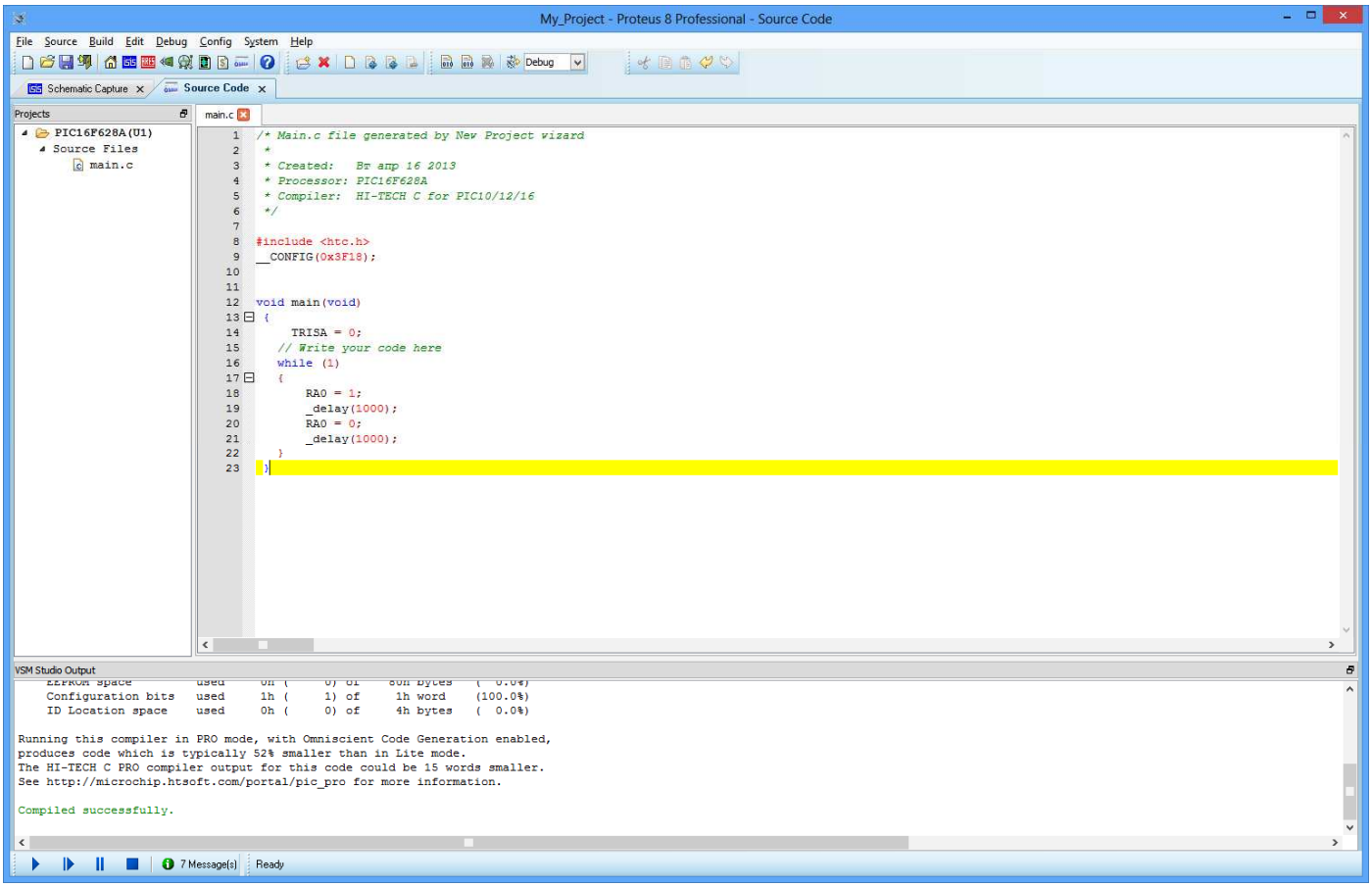


В восьмой версии программы весь пакет объединён в единое целое, добавлено много новых элементов, но, как это бывает обычно, основа интерфейса сохранилась. Так что пользователям предыдущих версий не придётся блуждать в меню, отыскивая нужные компоненты.

Удобнее стало разрабатывать схемы на основе микроконтроллеров. При создании проекта, а с него начинается работа, можно выбрать контроллер.

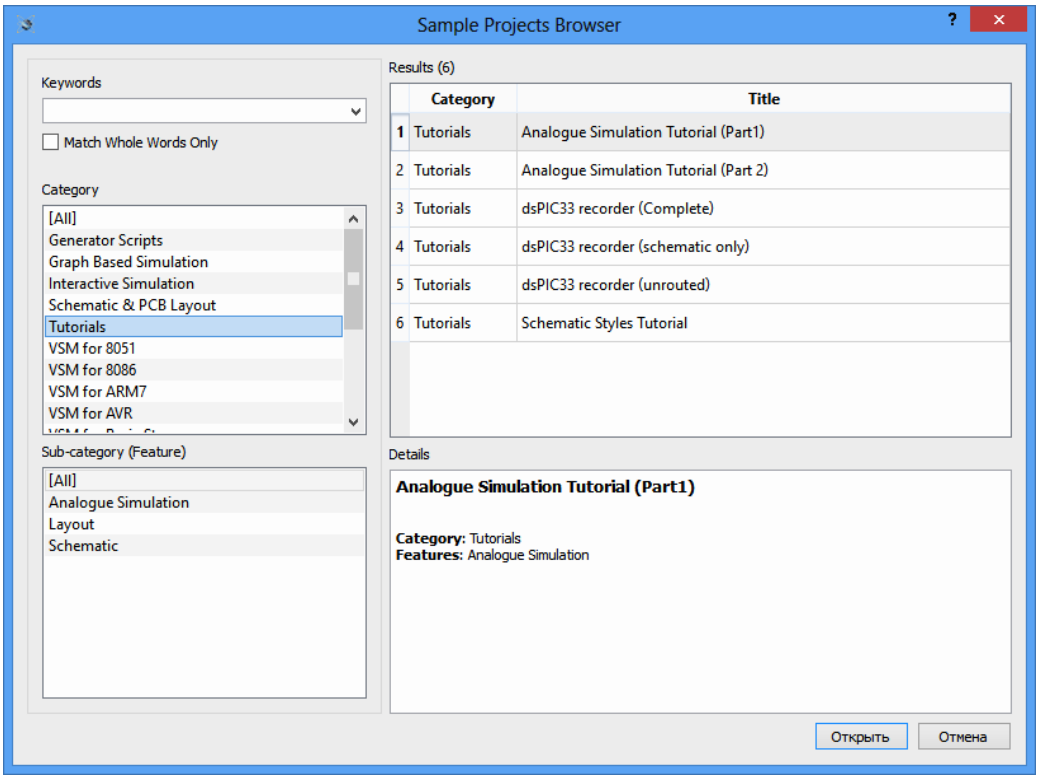


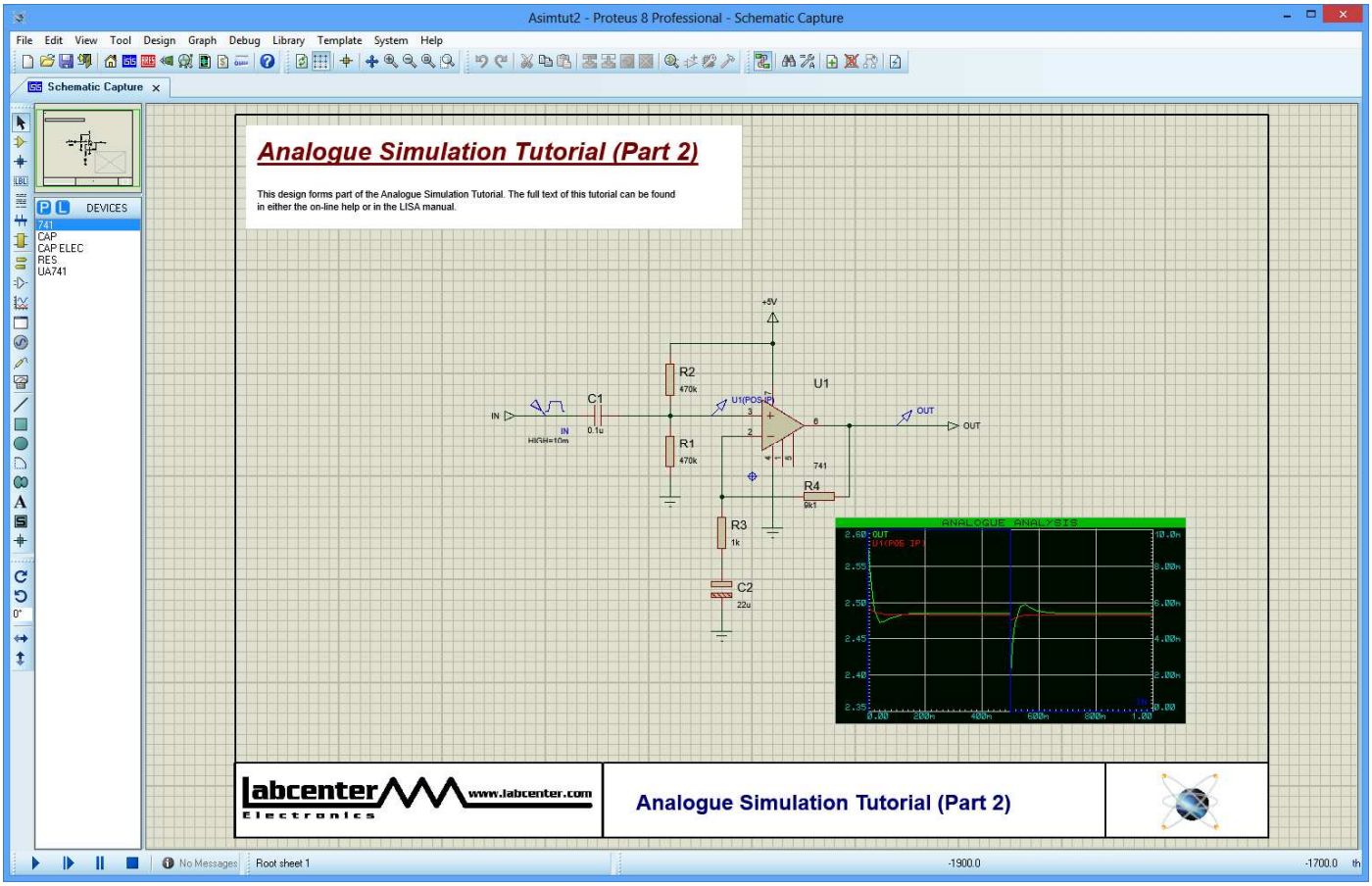
Но не это главное. После выбора компилятора, вы получите заготовку под текст программы.



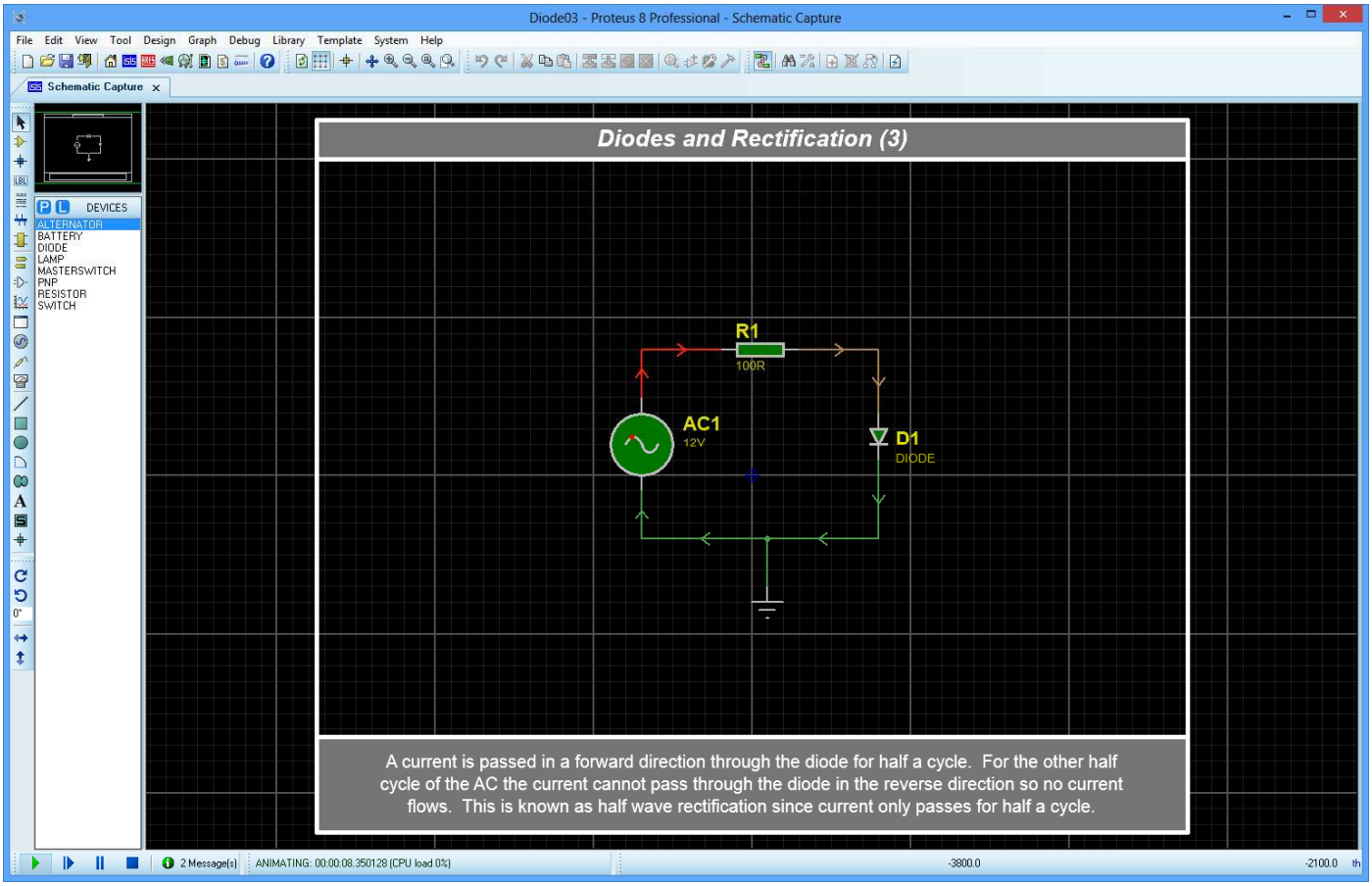
Код программы можно записать, откомпилировать и проверить работу в одной среде разработки.

Для тех, кто не знаком с программой, есть два пути это знакомство осуществить – использовать обучающую часть подсказки, нажав кнопку **Open Sample** основного окна (домашнего окна) программы:

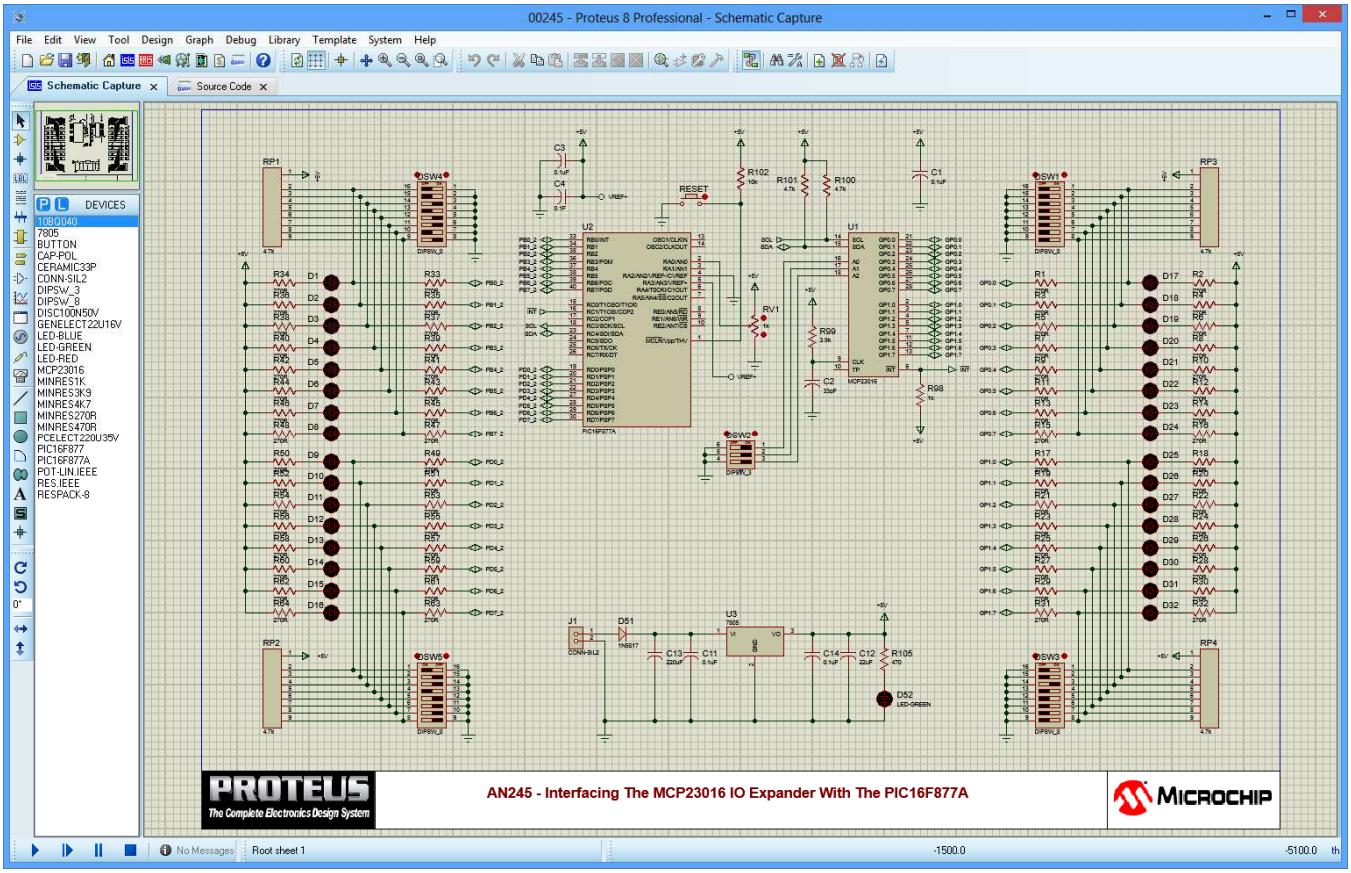




И внимательно просмотреть эти обучающие примеры. Или там же посмотреть примеры работающих схем. Есть схемы от самых простых, предназначенных для демонстрации на уроках физики:

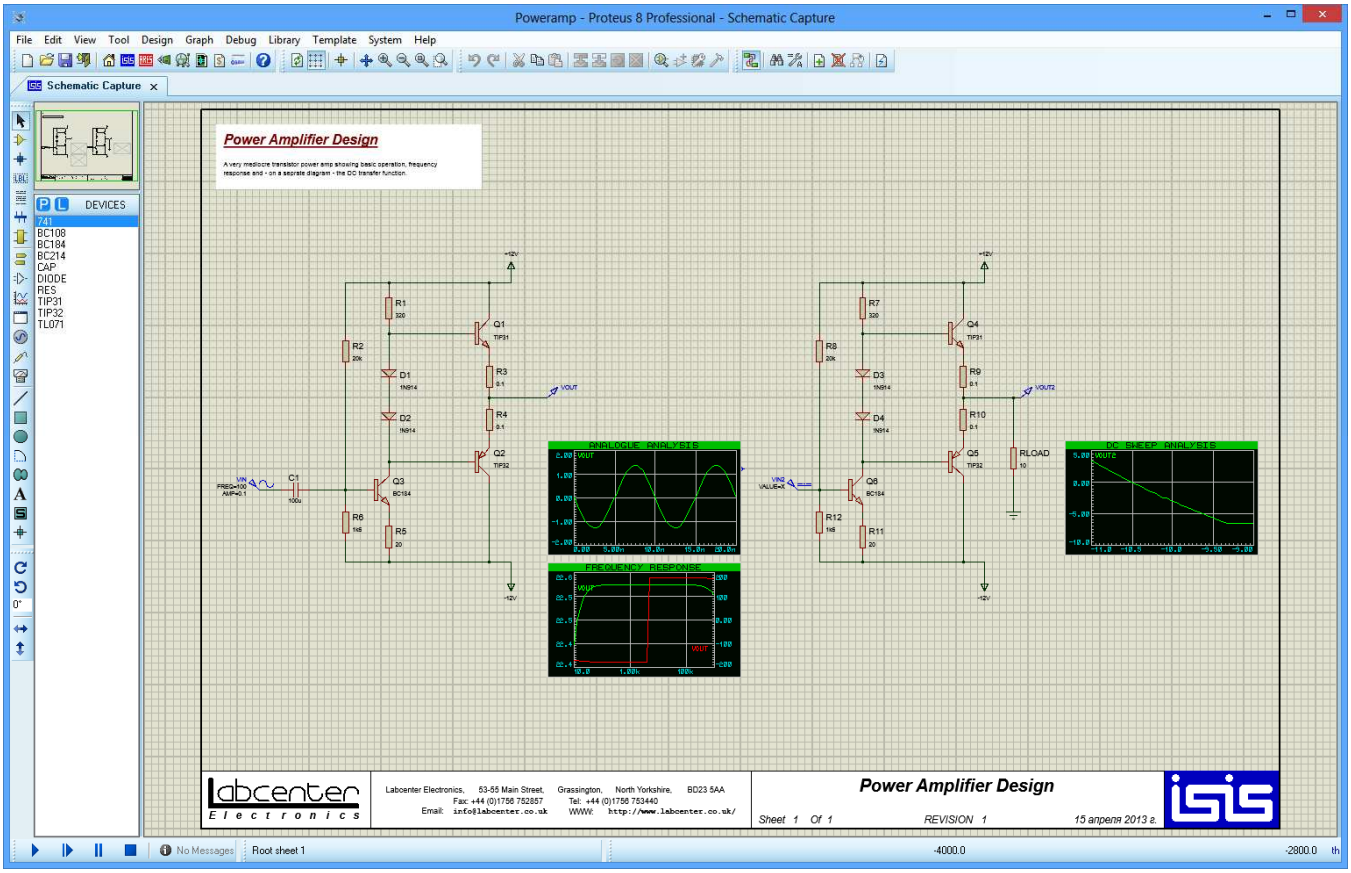


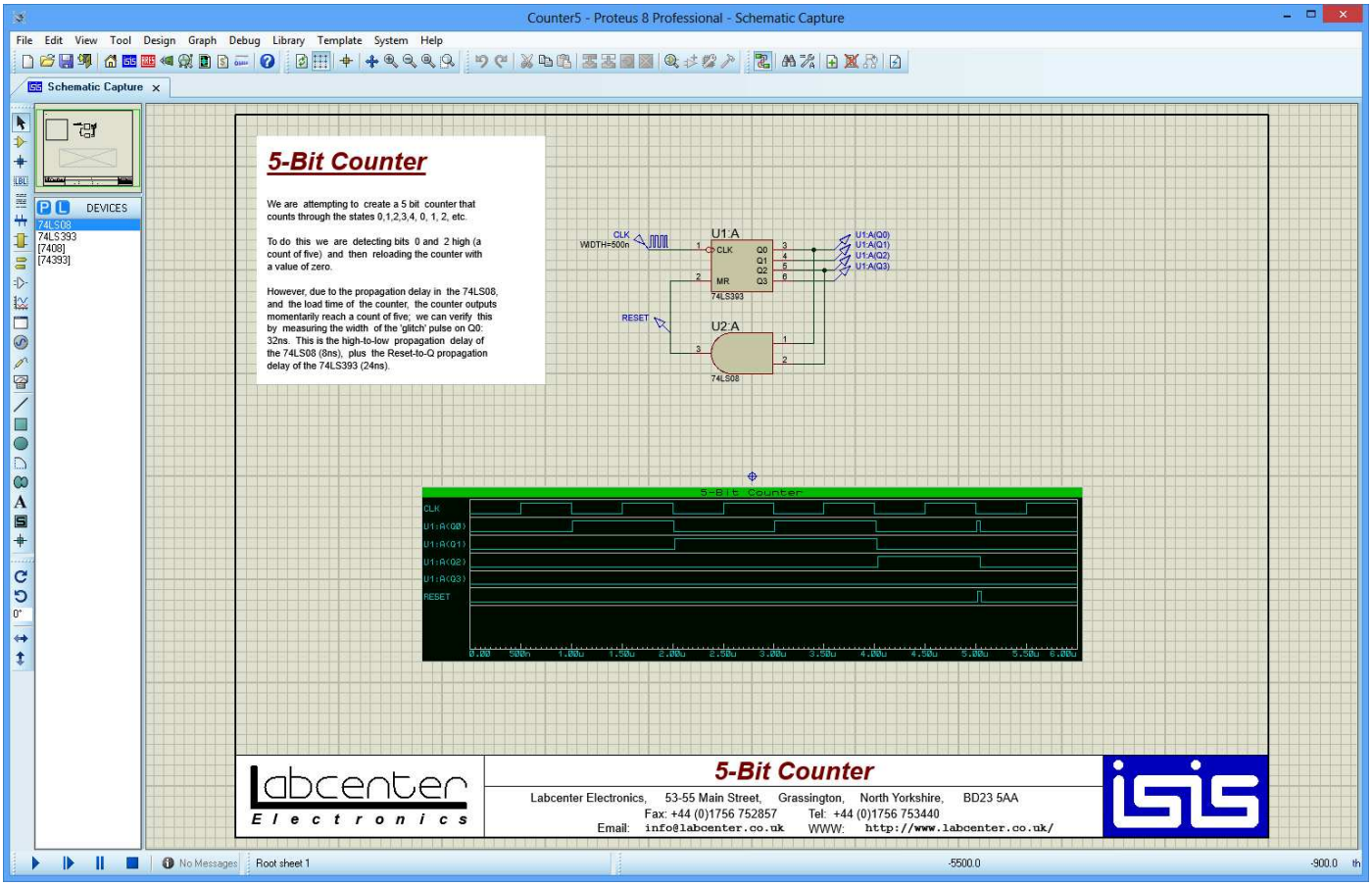
До достаточно сложных:



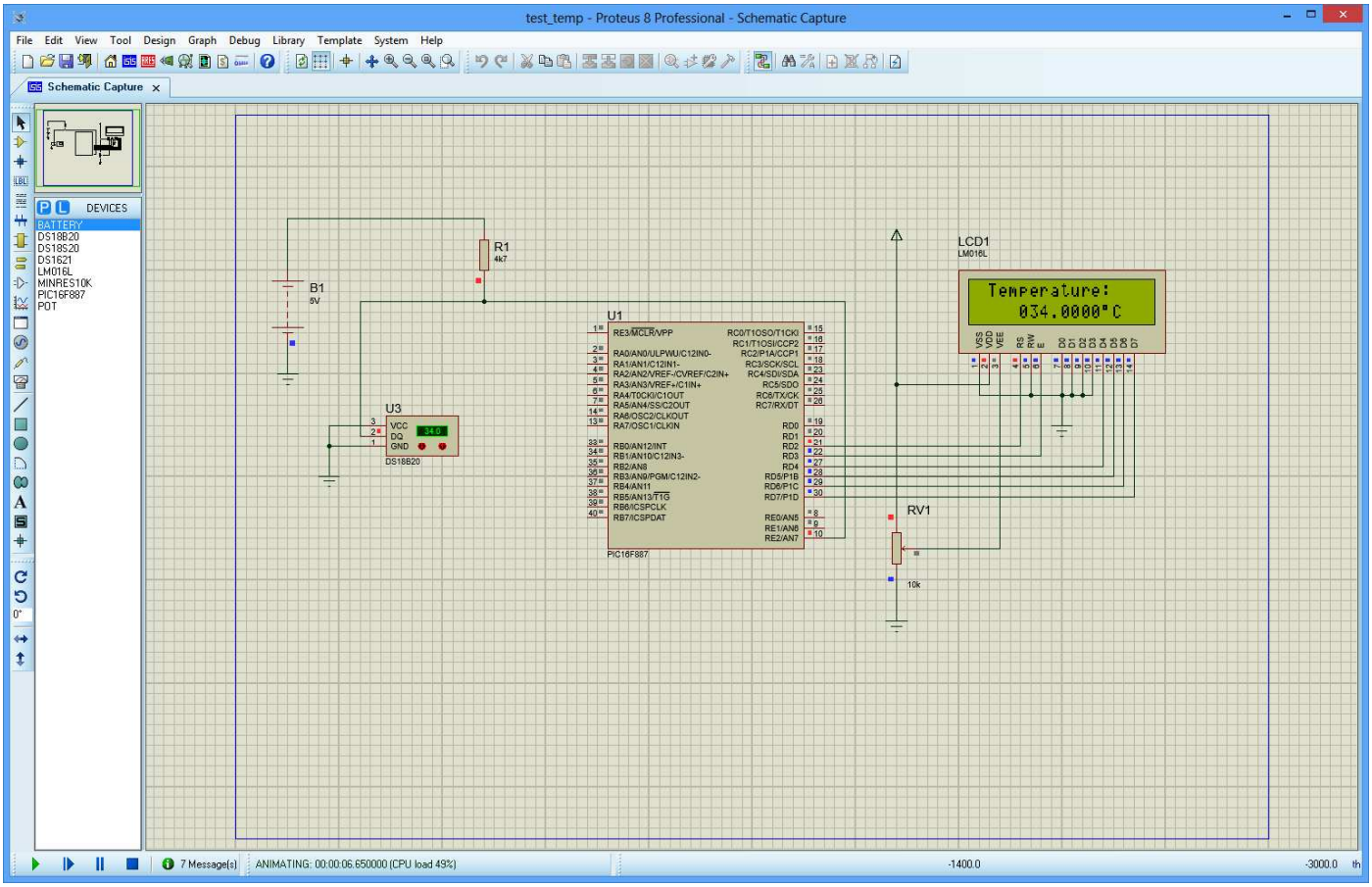
Приведённые примеры показывают тот широкий диапазон применения программы, от обучения до применения на производстве, который делает её уникальной в своём классе программ САПР.

Симулятор ISIS программы Proteus работает как с аналоговой, так и с цифровой техникой.



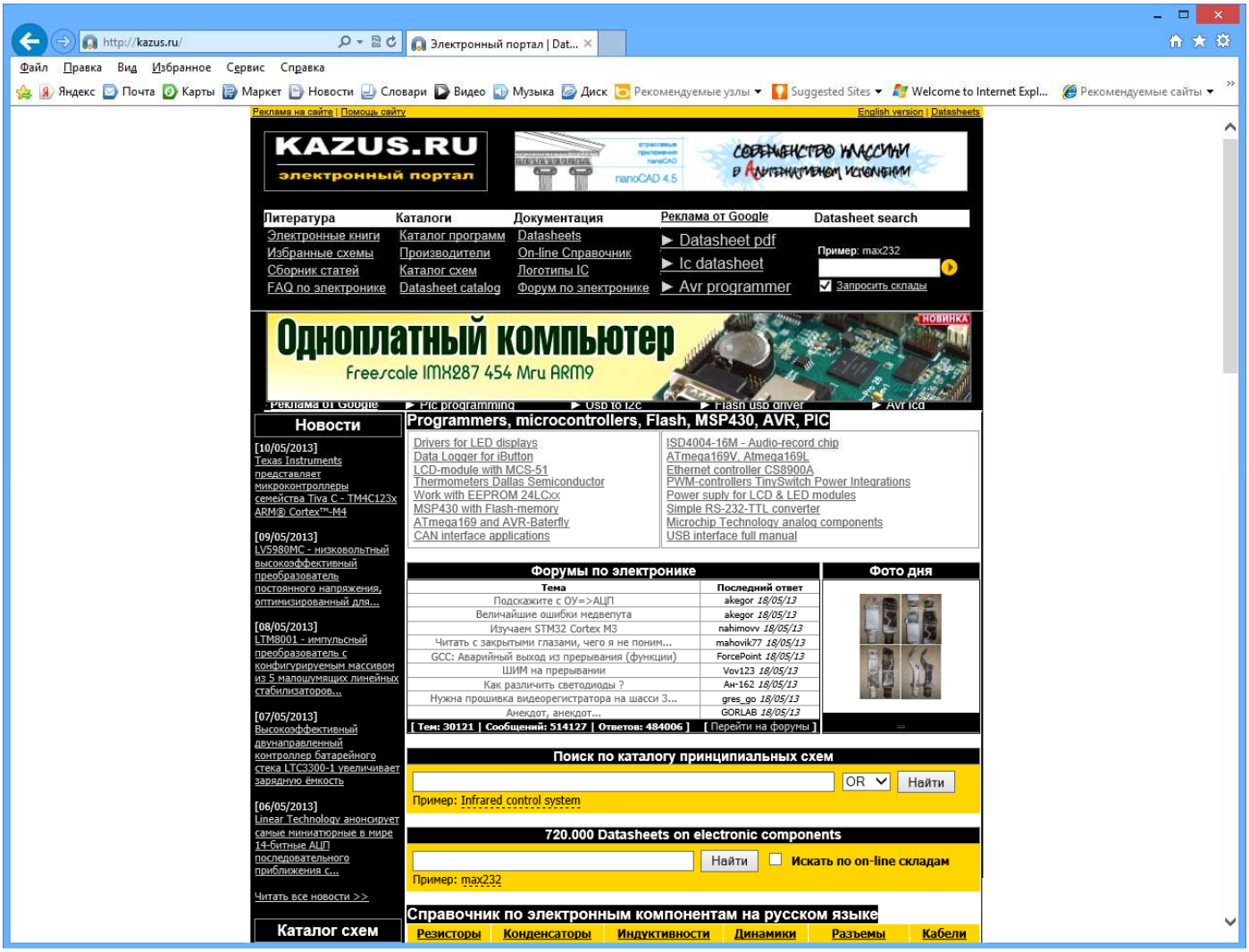


Но особенно важно то, что в ISIS можно проверить работу микроконтроллера с внешними элементами:



Программа Proteus – это система сквозного проектирования, поэтому есть вторая часть, ARES, позволяющая осуществить разводку печатной платы. Таким образом, можно разработать электронное устройство от идеи до опытного образца, не покидая программы, достаточно иметь опыт разработки и купить полную версию Proteus.

Современные программы достаточно сложны, требуют тщательной проверки, и обсуждение Proteus на форуме KAZUS.RU любителями и профессионалами может быть полезно не только им, но и разработчикам программы, особенно в том случае, если есть планы по продаже продукта в нашей стране.



FAQ (ЧаВО) по PROTEUS для начинающих и не только. ЧАСТЬ I

Содержание:

FAQ (ЧаВО) по PROTEUS для начинающих и не только.

*Это третья, надеюсь, последняя версия FAQ по Proteus. Учитывая опыт предыдущих, она построена несколько иначе. Первые три-четыре страницы предназначены в основном для начинающих. Здесь будут разобраны установка и настройка **ПО**, а также назначение основных функций меню и кнопок в программе **ISIS**, т.к. именно она вызывает повышенный интерес у Российских пользователей. Здесь же будут приведены советы по быстрому редактированию схем в **ISIS**, поскольку многие пользователи из-за отсутствия знаний английского языка не заглядывают в прилагаемый к программам HELP и просто не подозревают об этих возможностях.*

При подготовке данного материала использовалась информация от следующих участников форума Kazus.ru, посвященного теме «Микроконтроллеры и их применение»:
AndreiVV, Andronio, avr123-nm-ru, dosikus, Gordey, Kabron, Nemo78, retro55, TEHb (сотрудник Labcenter Electronics*), Um, vgololobov, Worker и многих других.*

С уважением, Halex07.

1. Краткие общие сведения о програмном продукте PROTEUS.

1.1. Протеус – что это такое?

Proteus — это коммерческий пакет программ класса САПР, объединяющий в себе две основных программы: ISIS – средство разработки и отладки в режиме реального времени электронных схем и ARES – средство разработки печатных плат. В качестве автоматического встроенного трассировщика в ARES, начиная с версии 7.4, используется программа ELECTRA Autorouter. До этого она являлась дополнительным и самостоятельным средством трассировки и устанавливалась в отдельную папку. Для создания собственных VSM (программных) моделей с версиями до 6.3 распространялась библиотека VSM SDK (папка INCLUDE), которая в более поздних версиях отсутствует, т.к. разработчик посчитал необходимым закрыть данную информацию с целью предотвращения «плагиата» моделей другими фирмами.

1.2. Сайт автора программы.

Разработчиком пакета Proteus является фирма Labcenter Electronics Великобритания. Сайт разработчика: http://www.labcenter.co.uk/ .

1.3. В чем отличие от других подобных программ.

Отличие от аналогичных по назначению пакетов программ, например, Electronics Workbench Multisim, MicroCap, Tina и т.п. в развитой системе симуляции (интерактивной отладки в режиме реального времени и пошаговой) для различных семейств микроконтроллеров: 8051, PIC (Microchip), AVR (Atmel), и др. Протеус имеет обширные библиотеки компонентов, в том числе и периферийных устройств: светодиодные и ЖК индикаторы, температурные датчики, часы реального времени - RTC, интерактивных элементов ввода-вывода: кнопок, переключателей, виртуальных портов и виртуальных измерительных приборов, интерактивных графиков, которые не всегда присутствуют в других подобных программах.

1.4. Системные требования. Различия в версиях программы.

Протеус устойчиво работает под управлением Windows 2k, XP, Vista. Имеются сведения об успешном запуске Proteus в Linux с помощью Windows эмуляторов (В частности автор этих строк успешно опробовал работу Proteus 7.5.SP3 в Ubuntu 7.10 под Wine). С «пиратскими» версиями операционных систем возможны проблемы устойчивой работы Протеуса.

Протеус активно развивается на протяжении 12 лет, начиная с ранних версий 4.xx и кончая последней на сегодняшний день версией 7.5SP3. Готовится к выходу версия 7.6. Если не рассматривать ранние четвертые версии, то наиболее распространенными являются версии 6 и 7. Главное отличие версий в постепенном увеличении количества компонентов в библиотеках и соответственно размера дистрибутива, а также в некоторых функциях кнопок мыши, которые в шестых версиях более напоминают настройку для «левши», что без некоторого навыка непривычно. Это напоминает езду на автомобилях с правым рулем и при левостороннем движении.

1.5. Что и где прочитать о Протеусе на русском языке.

Русскоязычные публикации на данную тему чрезвычайно скудно представлены в сети, а в печатном виде их и того меньше. Из известных печатных изданий можно порекомендовать серию статей А. Максимова в журналах «Радио» №№4-6 за 2005 г. и третью часть книги В. Гололобова «Экскурсия по электронике» - online публикация доступна по адресу:
http://vgololobov.narod.ru/content/proteus/Proteus.html
Вот еще некоторые онлайн ресурсы посвященные Протеусу:
http://kazus.ru/programs/viewdownload/kz_0/cid_190.html - учебник по Протеус на русском, правда к старой версии и довольно краткий и «кое-что» еще по Протеус.
http://www.proteus123.narod.ru – станичка AVR123-nm-ru
http://www.radiokot.ru/forum/viewtopic.php?t=3739 – страничка на сайте Радиокот
имеются также странички и на других сайтах посвященных радиоэлектронике:
pro-radio.ru, radioprog.ru, форумы на telesys.ru, caxapa.ru и др.

2. Установка и запуск Proteus. Интерфейс программы ISIS.

2.1. Где взять инсталляционный пакет Протеус.

На официальном сайте компании **Labcenter Electronics** доступна последняя демо-версия на данный момент v.7.5.SP3. Она имеет существенные ограничения: отсутствует опция сохранения проекта, симулируются в реальном времени в основном примеры из прилагаемой папки **Samples**. Учитывая географическую удаленность «туманного Альбиона» и приличный размер инсталлятора – более 60 Мбайт, я бы не рекомендовал скачивание в ознакомительных целях данного пакета тем, у кого медленное Интернет-соединение. Но мир не без «добрых» людей. Давать здесь конкретные ссылки на сайты файлообменники нет смысла, жизнь файлов там ограничена по времени. Поэтому воспользуйтесь поиском в Google или другом поисковике с параметрами Proteus v.7 (или 6), Proteus VSM или Proteus ISIS и вы легко найдете свежие ссылки. Только не стоит использовать поиск по одному слову «Протеус» или «Proteus», если вы не стремитесь приобрести одноименный силовой тренажер для накачки мускулатуры.

2.2. Установка программы на компьютер.

Для установки необходимо запустить инсталляционный пакет **Setup.exe**. В ходе установки **Proteus** (если это не демо версия) запросит путь к файлу лицензии. Если на этот момент файл лицензии отсутствует можно просто выбрать вариант наличия лицензии на сервере, а окно сервера оставить пустым, но перед первым запуском все равно необходимо будет установить лицензию файл **licence.lxk**, воспользовавшись менеджером лицензий. По умолчанию программа устанавливается в директорию: **Program Files\ Labcenter Electronics\ Proteus 7**, однако при желании можно изменить путь. Как уже отмечалось для профессиональной версии после установки необходимо установить лицензию. Для этого запускается программа менеджер лицензий *(рис.1):*

ПУСК=>Все программы=>Proteus x Professional=>Licence Manager

в левом окне через кнопки **Browse For Key File** (вручную) или **Find All Key File** (автопоиск) выбирается путь к файлу лицензий, затем нажимается кнопка **Install**, которая становится доступной при щелчке по нужной лицензии в левом окне, и выбранная информация должна появиться в правом окне. После чего менеджер можно закрыть. Обращаю Ваше внимание, что напротив изображения ключей перечисляются доступные для данной лицензии функции программы.

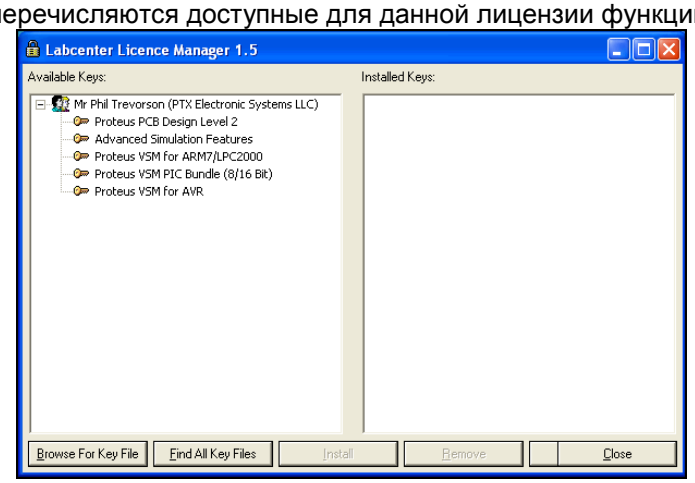


Рис.1

2.3. Первый запуск и первые проблемы.

I. При попытке запуска **ISIS** или **ARES** появляется окно с сообщением:

Cannot find a valid licence key for ISIS (ARES) on this computer.

Коментарий: *отсутствует лицензия, т.е. не выполнен или не до конца выполнен предыдущий пункт.*

II. При запуске симуляции (в том числе и прилагаемых примеров из папки **Samples**) она не функционирует, а в **Simulation log** *(Рис.2)* появляется сообщение:

Cannot open 'C:\DOCUME~1\=ТЕКПОЛЬЗ=\Local Setting\Temp\LISAxxx.SDF'
Simulation FAILED due to fatal simulator errors

где вместо **\=ТЕКПОЛЬЗ=** непонятные закорючки (крякозябры)

Коментарий: *Данная проблема не актуальна для версий начиная с 7.4 и выше. До этого Протеус категорически отвергал кириллицу в имени пользователя компьютера, а также и в пути к файлу проекта и в самом названии файла.*

Есть два пути решения этой проблемы:

1) *Изменить имя пользователя на английский вариант.*
2) *Зайти в Мой компьютер=>Свойства=>Дополнительно=>Переменные среды. В верхнем окне, выбрав переменную TEMP, нажать Изменить и вместо* ***%USERPROFILE%***

*набрать **%ALLUSERPROFILE%** (при этом необходимо, чтобы в папке **Document and Setting\All Users** имелись соответствующие папки **Local Settings** и **Temp** их можно просто перекопировать из текущего пользователя (папки СКРЫТЫЕ) или создать вручную). Можно по совету Nemo78 изменить путь на **%SYSTEMROOT%\Temp (именно так без Local Settings)**, тогда Протеус будет использовать папку **TEMP** в системном каталоге Windows.*

III. Симуляция запускается, но через несколько секунд (минут) программа закрывается. Симуляция работает только с некоторыми типами моделей. Примеры из Samples симулируются без проблем.

Коментарий: *Отсутствует лицензия на одну из используемых моделей. Вы используете «неофициальную» (крякнутую) версию и кряк либо не установлен, либо неправильно установлен. Протеус имеет многоступенчатую защиту от нелегального использования, которая многократно проверяется в процессе симуляции. Защищаются файлы как в основной папке программы \BIN (Isis.exe, Ares.exe, Licence.dll, Prospice.dll), так и в папке библиотек моделей \Models (Avr.dll, Lcdalfa.dll, Lcdpixel.dll, LedMPX.dll, Pic16.dll, Pic18.dll, Mcs8051.dll и некоторые другие модели). Поэтому симуляция будет работать только с теми библиотеками, на которые имеется лицензия, или к которым применялась «доработка».*

2.4. Интерфейс программы ISIS.

Ниже приведено основное окно программы ISIS с пояснениями по назначению основных элементов интерфейса. В дальнейшем я буду придерживаться именно такой терминологии в несколько сокращенной форме, т.е.: левое меню, верхнее меню команд, верхнее основное меню, кнопки симуляции, селектор объектов. Окно программы немного не соответствует полностью развернутому окну, поскольку при уменьшении размеров некоторые меню изменили положение. Так же как и во многих других программах для Windows меню можно перетаскивать в удобное для вас место внутри окна программы. Зацепив через левую кнопку мышки за стартовый элемент меню (прямоугольная серая полоска для горизонтальных меню слева, а для вертикальных – сверху) не отпуская кнопки перетаскиваете, например, меню ориентации (на картинке стартовый элемент виден над стрелокй вращения вправо) внутри окна к правой вертикальной границе окна и после отпускания кнопки оно «приклеится» вертикально справа. Аналогично можно поступить и с любым из верхних командных меню. Таким образом можно настроить удобное для себя расположение элементов программы. Другая приятная «фишка» программы: если щелкнуть внутри окна селектора правой кнопкой мышки и во всплывающем окне щелкнуть левой кнопкой по функции **Auto Hide**, то селектор будет автоматически сворачиваться, если на него не наведен курсор мышки. Это позволяет на мониторах с форматом 4:3 выиграть некоторое пространство для окна редактирования. Отмена этого режима повторными действиями.

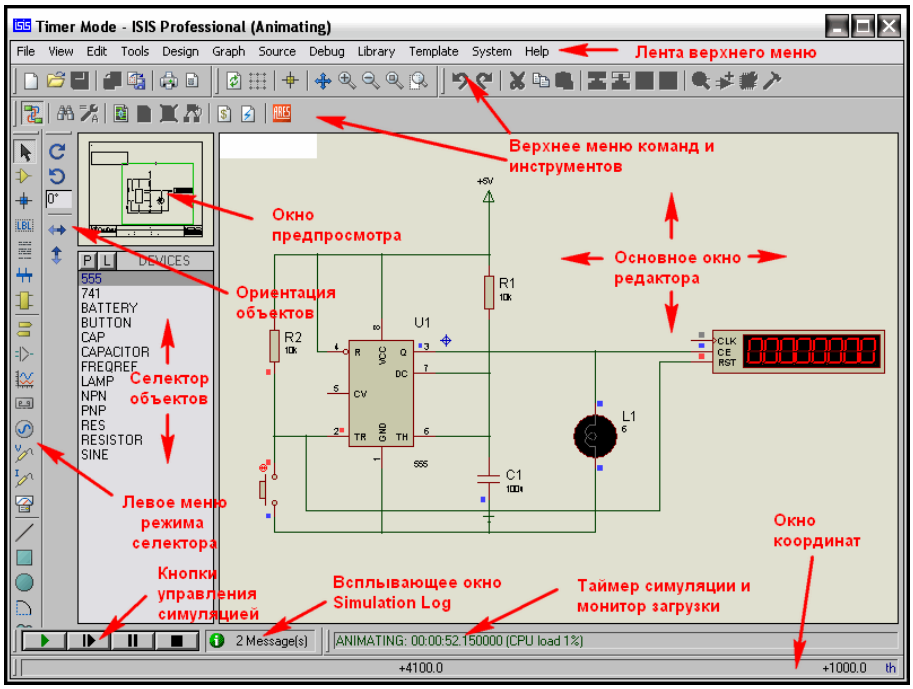


Рис.2

2.5. Папка Samples - кладезь примеров проектов для начинающих.

При первом запуске ISIS появляются два всплывающих окна. В одном из них будет предложено проверить обновления – здесь можно смело поставить галочку – «больше не показывать». Другое окно предлагает открыть многочисленные примеры **Sample Designs**, прилагаемые вместе с программой. Если Вы действительно начинающий пользователь, не торопитесь ставить аналогичную галку блокировки повторного показа. Ну а если уже заблокировали это окно – не отчаивайтесь. Быстрый доступ к примерам всегда возможен через верхнее меню **Help => Sample Designs**. Почему я так настойчиво рекомендую ознакомиться с примерами? Да потому-что третья часть вопросов приходящих на форум имеют готовые ответы в прилагаемых с программой

примерах. К сожалению, для того чтобы ознакомиться с содержимым того или иного примера приходиться его открывать, так как в большинстве случаев по имени файла невозможно понять - что там внутри. С шестыми версиями Протеуса прилагался **Help** по примерам, но в седьмых версиях разработчик почему-то тихо его умыкнул. Описать содержимое всех примеров здесь не представляется возможным из-за большого объема информации. Поэтому, я остановлюсь только на самых значимых для начинающих и приложу оригинальный файл **SAMPLES.HLP** от версии 6.9sp5. Конечно, в нем отсутствует описание примеров для новых МК добавленных в следующих версиях, а также примеров программных генераторов из версий 7.4 и 7.5, но для владеющих даже начальным английским этот **Help** большое подспорье. Тем более, что даже с установленными последними версиями при щелчке мышью по зеленому названию проекта в хелпе он открывается автоматически.

Schematic & PCB Layout - одна из самых интересных папок для начинающих. Все проекты, за исключением **Shiftpcb**, содержащиеся в ней не предназначены для симуляции в реальном времени но при этом имеют как законченный вариант схемы **xxx.DSN** в ISIS, так и проект платы **xxx.LYT** в ARES.

Обратите внимание на проекты **Cpu** с использованием МК Z80 и **Dbell** – дверной звонок. В этих проектах имеются промежуточные файлы PSB (плат) с именами **Cpuu.LYT** и **Dbellu.LYT** с не установленными на плату компонентами. Открыв эти проекты в ARES Вы можете самостоятельно опробовать функцию автоматического размещения компонентов. Достаточно выбрать в верхнем меню **Tools => Auto Placer** и в раскрывшемся окне просто щелкнуть **OK**. В проектах **Cpu.LYT** и **Dbell.LYT** компоненты уже размещены, но можно аналогично попробовать автотрассировку дорожек **Tools => Auto Router**. Проекты **Cpur.LYT** и **Dbellr.LYT** содержат уже оттрассированные платы. На любом этапе в ARES через верхнее меню **Output => 3D Visualization** можно вызвать трехмерное изображение платы и зацепив ее левой кнопкой мыши повращать и обследовать со всех сторон (Рис.3).

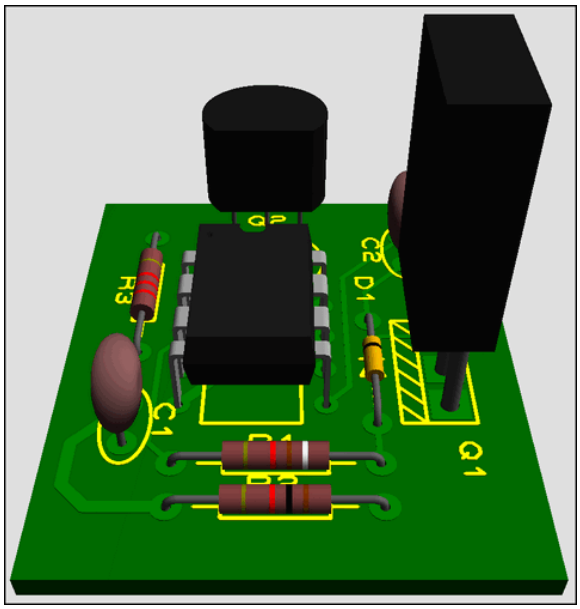


Рис.3

Отдельно остановлюсь на проекте **Shiftpcb.DSN** – 16-ти битный сдвиговый регистр на мелкой логике. Он заслуживает внимания по двум причинам. Во-первых в нем применена 4-х ступенчатая иерархическая структура, т.е. это сложный проект. На первом листе помещены четыре модуля четырехразрядных сдвиговых регистров. Чтобы посмотреть структуру каждого модуля необходимо щелкнуть по нему правой кнопкой мышки (элемент станет красным) и выбрать во всплывающем меню опцию **Goto Child Sheet**(Ctrl+C) – переход на дочерний лист. Аналогично можно попасть на следующий уровень и далее до конечного, содержащего обычный RS-триггер на элементах 2И-НЕ. Возврат на предыдущий уровень также щелчком правой кнопки щелчком только по свободному месту в окне и выбор опции **Exit to Parent Sheet** (возврат на родительский лист). Во-вторых здесь можно запустить симуляцию после некоторой коррекции проекта и посмотреть воочию работу сдвигового регистра. В исходном виде проект адаптирован под помещенный на первом листе график, поэтому при симуляции через кнопку управления симуляцией **Play** мы получим в логе предупреждение (желтый восклицательный знак) о загрузке ЦП компьютера 100% и невозможности симуляции в реальном времени:

Simulation is not running in real time due to excessive CPU load

Окно откроется, если щелкнуть по **Simulation Log** левой кнопкой мыши. Сразу же привыкайте к принципу светофора в **Simulation Log**: красный знак – грубая ошибка – симуляция невозможна; желтый («горчичник») – предупреждение – симуляция может и выполняться, но результат некорректен и зеленый – симуляция протекает нормально без ошибок. Поэтому, чтобы избежать предупреждения необходимо в свойствах генераторов **D** и **Clk** (доступны через правую кнопку мыши

опция **Edit Properties** **Ctrl+E**) установить соответственно **Pulse width** 200m и 100m (в данном случае миллисекунды). Запустив кнопкой **Play** симуляцию после этого можно на контактах разъема **J2** наблюдать состояние выходов сдвигового регистра.
В этой же папке содержатся другие примеры:
EPE.DSN – большой проект программатора EPROM на трех листах (переход между листами доступен через верхнее меню **Design** или щелчком правой кнопко мышки по свободному месту в окне редактирования и выборе соответствующего листа 1, 2 или 3). На некоторых листах содержатся субмодули. Вы уже усвоили, что они имеют темно-синюю обводку и соответственно доступные дочерние листы.
FEATURES.DSN – в проекте показаны различные способы выполнения схем в ISIS. Обратите внимание на правый верхний угол: вариант стереофонического предусилителя, оформленный в виде 2-х субмодулей с дочерними листами.
PPSU.DSN – очень простой проект стабилизатора напряжения. Имеет два варианта PSB: **PPSU.LYT** – для микросхемы в корпусе DIL8 (монтаж в отверстия) и **PSMT.LYT** - м/сх в планарном корпусе SO8. Обратите внимание, что DIL – Dual-In-Line почему-то у нас в России принято называть DIP. Если для PSB в Протеусе выбрать корпус DIP Dual-In-Plane – отверстий в плате вы не увидите! «Гробик» будет выведен в ARES как планарный с шагом 2,54 мм.
SIGGEN.DSN – проект генератора сигналов. В хелпе лихо заявлено, что симулируется – да, но после значительной правки.
STYLE1, 2, 3 – примеры различного оформления одного и того же проекта.
THERMO – термометр с термопарой в качкестве датчика и индикацией на семисегментных индикаторах. Здесь не симулируется, но в папке **VSM for PIC18\ MAX6675 Thermometer** есть работающий проект с программой на PICC18 и проектом для MPLAB.
dsPIC33_REC – проект устройства регистрации давления аналогично предыдущему имеет рабочий дубль в папке **VSM for dsPIC33.**
Interactive Simulation – папка содержит подпапку **Animated Circuits** с очень простыми анимированными примерами для начинающих.
Basic – примеры начинающиеся с этой аббревиатуры основаны на базовых познаниях электротехники: лампочка, батарейка, выключатель, потенциометр и показывают протекание тока в цепи.
MVCR – ряд примеров с использованием виртуальных приборов вольтметр/амперметр.
PCV – примеры с потенциометром ограничителем тока.
Intres – примеры на внутреннее сопротивление источника тока.
Cap – три примера работы конденсатора.
AC – примеры с переменным током.
Diode – примеры на применение диодов и диодных мостов.
Inrel – примеры спримененения индуктивностей и реле.
TRAN – семь примеров с транзисторами.
Opamp – шесть различных примеров с операционными усилителями. Заслуживают особого внимания. Там есть вариант включения ОУ, как компаратора (**Opamp1.DSN**). Все это анимировано, обвешано виртуальными приборами, можно покрутить и посмотреть на реакцию ОУ.
Osc – примеры генераторов. **Osc03.DSN** и **Osc04.DSN** на таймере 555, содержащем дочерний лист с внутренней структурой таймера на примитивах **Spice**. Это «стартовая площадка» для освоения создания собственных моделей.
Comb и **Seq** – примеры для освоения работы логических цифровых микросхем.
Ну и несколько познавательных примеров: **TRAFFIC.DSN** – светофор, **COUNTER.DSN** – четырехразрядный счетчик на 74LS390, **TTLCLOCK.DSN** – часы на TTL логике, **LISSAJOUS.DSN** – применение виртуального осциллографа для наблюдения фигур Лиссажу и **LM3914.DSN** – применение одноименного драйвера для управления линейной светодиодной шкалой.
Остальные подпапки из **Interactive Simulation** содержат примеры проектов на использование одноименных виртуальных инструментов из библиотек Протеуса: **Counter Timer** – применение виртуального таймера/счетчика в режимах таймера и частотомера. **Motor Examples** – примеры проектов с шаговыми двигателями. **Pattern Generator** – примеры применения виртуального генератора кодовой последовательности. **COMPIM Demo** – пример использования виртуального COM-порта и виртуального терминала в Протеусе. Последнему для выполнения симуляции необходимо наличие на компьютере двух реальных COM-портов, соединенных нуль-модемным кабелем, либо установки на компьютер программы виртуального COM-порта для имитации соединения с реальным. При этом в режиме симуляции можно организовать обмен данными через это соединение из программы ISIS с любой программой на компьютере, позволяющей работать с COM-портом (например, стандартной **Hyper Terminal**).
Остальные подпапки из папки **Samples** содержат примеры проектов с использованием соответствующих серий микроконтроллеров (например **VSM for PIC16** – примеры с МК Microchip PIC16). Я не буду их рассматривать подробно сейчас, так как к наиболее интересные будут рассматриваться позже, по мере освоения программы ISIS.
Здесь только перечислю, что Graph Based Simulation содержит примеры применения различных типов графиков для исследования схем, к папке Tutorials мы обратимся при создании собственных моделей. Особо отмечу две папки: **VSM MPLAB Viewer** и **VSM AVR Studio Viewer.** Эти папки содержат примеры совместного использования соответствующих инструментариев. При этом

Протеус ISIS выступает в качестве продвинутого отладчика, интегрированного в данные пакеты. Естественно при этом необходимо иметь установленные на компьютере **MPLAB IDE** версии не ниже 7.5 для микроконтроллеров PIC и **AVR Studio** версия 4.16 для микроконтроллеров AVR. Данные продукты абсолютно бесплатны и доступны для скачивания с соответствующих сайтов. Предвидя лишние вопросы «чайников», вот ссылки на страницы этих программ:
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725
Вызов Proteus, как отладчика осуществляется непосредственно из интерфейса этих программ.

2.6. Основное меню ISIS. Опции, необходимые на начальном этапе.

В классических учебниках и встроенных **Help** на следующем этапе принято подробно рассматривать назначение опций меню и кнопок интерфейса программы. Я немного отступлю от канонов. Сейчас мы рассмотрим только насущные на данный момент пункты меню и назначение самых необходимых кнопок. Это даст Вам возможность начать сразу же комфортно работать в ISIS. Остальные элементы интерфейса мы изучим по мере необходимости обращения к ним. Ну а принятое в таких случаях описание общераспространенных кнопок: Save, Print, Copy, Paste, Undo и т.д. я вообще опущу. Надеюсь, пользователь, решивший освоить Протеус, не первый раз сидит за компьютером и уже встречался с использованием аналогичных функций в других программах, хотя бы в тех же Notepad или MS Word. Итак, начинаем с верхней ленты стандартных меню.
В меню **File** остановимся на функциях **Export/Import**. **Import Bitmap…** позволяет поместить картинку в Ваш проект. Отмечу, что картинка должна быть в формате BMP с глубиной не более 256 цветов. Эта функция удобна при перерисовывании схем. Импортируете схему в окно редактирования, соответственно уменьшаете ее, потянув мышкой за угол, чтоб не занимала много места и затем составляете ее уже из элементов **ISIS** на свободном поле окна редактирования. **Export Graphics…** позволяет экспортировать нарисованный в окне **ISIS** проект, как графическое изображение различных форматов, в том числе и **DXF** (AutoCAD). **Import Section…** и **Export Section…** - сохраняют текущий лист проекта в файл с расширением **.SEC**. Внимание, это единственное средство позволяющее передать проект из Протеус последних версий в более ранние. Поясню, что в программе прекрасно соблюдается наследственность снизу вверх, т.е. проект из версии 6 всегда откроется в версии 7, но не наоборот. Здесь строгие ограничения. Проект, составленный в версии 7.5, Вы не сможете открыть даже в версии 7.4. Функции экспорта /импорта секций позволяют обойти это ограничение. В старшей версии вы экспортируете лист проекта, как секцию (отметьте, что операция проводиться с отдельными листами **Sheet**), а в ранней версии импортируете эту же секцию. Еще два замечания:
а) если в проекте использованы компоненты, отсутствующие в предыдущей версии, симуляция их невозможна;
б) касается на данный момент МК AVR, которые могут быть прописаны в библиотеках AVR2.DLL в поздних версиях и AVR.DLL в ранних. После импорта секции в старую версию модель МК придется также поменять.
В меню **View** сейчас нам важны следующие опции:
Grid (клавиша **G** здесь и далее я буду в скобках давать используемые по умолчанию клавиши) – включает/выключает изображение сетки. **SnapXX…** (F2…F4 и Ctrl+F1) переключает шаг сетки, где XX – десятые доли дюйма, т.е. 2,54 мм. По умолчанию при запуске ISIS всегда устанавливается 0,1 **Inch** (англ. дюйм). Думаю, многие догадались, что самый мелкий шаг 10th (0,01 дюйма) вызывается через Ctrl+F1, потому что просто F1 – это во всех программах вызов файла помощи.
Среди функций масштабирования остановлюсь только на **Zoom to Area**, позволяющей четко разместить в пределах окна редактирования выделенный перед этим участок схемы.
Пункт **Toolbars…** позволяет включить/выключить отображение одоименных верхних тулбаров. К сожалению, изменить отображаемый в них набор кнопок-инструментов невозможно.
В меню **Edit** отмечу опцию **Tidy**. Она позволяет удалить из окна селектора объектов все компоненты, не используемые в текущий момент в проекте. Т.е. если вы набрали в окно из библиотеки множество ненужных компонентов, этой опцией удалятся все, кроме тех, которые установлены в окне редактирования. Можно удалять и по одному через правую кнопку мыши опцией **Delete**.
В меню **Tools** разберу пока только две опции, остальные чуть позже. **Real Time Annotation** (Ctrl+N) – вкл/выкл автонумерации элементов при добавлении в окно редактирования. Когда функция активна (по умолчанию) кнопка **U1** в меню выглядит утопленной. **Wire Auto Router** (W) опция автоматического изменения трассы провода на схеме при его проведении. Эта кнопка (по умолчанию включена) доступна также в одном из верхних тулбаров. При активной кнопке линии проводятся только строго под прямым углом. Используйте при прокладке проводов щелчки левой кнопкой мышки в тех местах, где вам необходимо зафиксировать поворот, иначе ISIS автоматом изменит трассу по своему усмотрению и не всегда красиво.
Ну и здесь же рассмотрим различные виды курсора при редактировании проекта, поскольку одна из функций **Property Assigment Tools** (A) характерно при включении меняет вид курсора. Запомните название и клавиатурный вызов этой функции – она ключевая для быстрого редактирования дизайна. И к ней Вы будете обращаться очень часто, когда освоите все ее достоинства. Скоро мы ей воспользуемся, а пока на рисунке 4 различные виды курсора в зависимости от выполняемой функции. Я не нашел ничего лучшего, как перевести на русский этот раздел файла помощи ISIS.

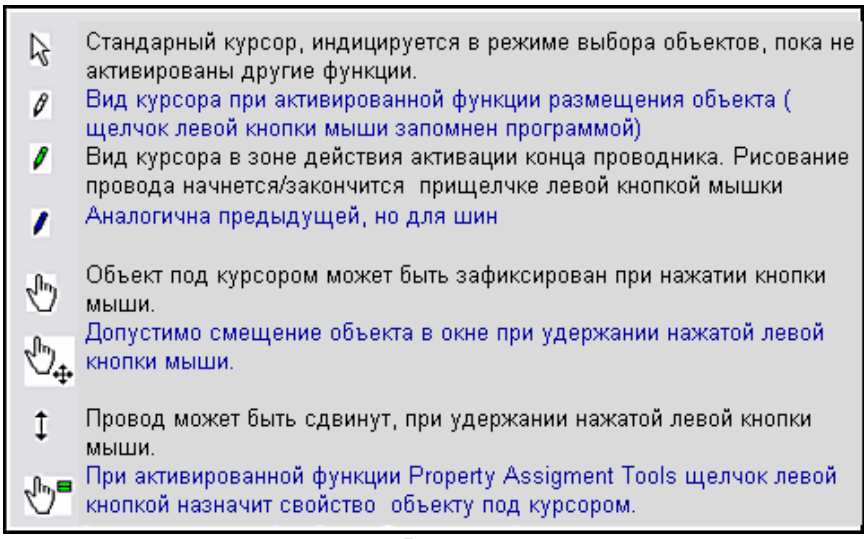


Рис.4

Меню **Design.** Верхние три пункта **Edit…** относятся соответственно к редактированию свойств проекта (**Design**), листа проекта (**Sheet**) и аннотации проекта (**Notes**). Здесь следует обратить внимание на окна с галочками для проекта и листа. Для проекта выбраны по умолчанию **Global Power Nets?** и **Cache Model Files**? Первый пункт означает будет ли глобальны цепи питания внутри всего проекта, например, если проект состоит из нескольких листов, а второй сохраняет файлы моделей внутри проекта, т.е. обеспечивает его переносимость. Поэтому этими галочками на первых порах не стоит экспериментировать. Для листа назначение подобных опций мы увидим при создании моделей, тем более, что пока окошки серые и не активны. О конфигурации шин питания - **Configure Power Rails** поговорим позже. Следующие далее опции меню **Design** касаются добавления, удаления листов (**Sheet**) в проекте и навигации между листами. Даже без перевода их назначение понятно из пиктограмм. Отмечу только, что при добалении листа Протеус автоматически присваивает ему имя **Root** на 10 больше предыдущего. Первый лист по умолчанию **Root10**, а внизу в трее программы отображается как **Root Sheet 1**. Навигация между листами доступна и непосредственно в нижней части меню **Design** и через меню правой кнопки мышки при щелчке по свободному полю листа. Ну и разберем оставшийся пункт **Design Explorer**, открывающий браузер проекта (Рис. 5).

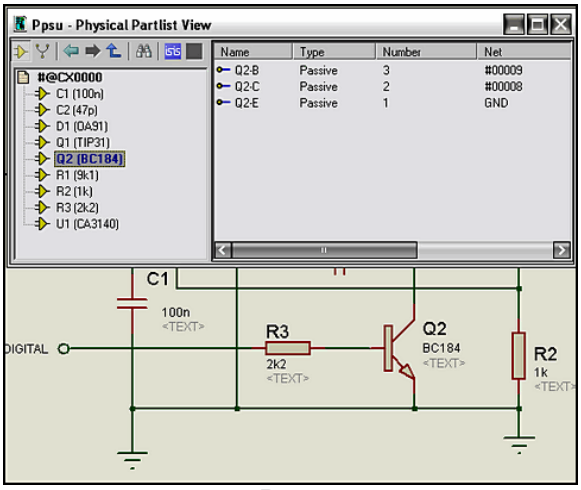


Рис.5

На рисунке слегка ужатое окно браузера, чтобы показать часть схемы примера **PPSU.DSN** из папки **Samples** Протеуса. **#@CX0000** в левом окне – это имя листа - то которое по умолчанию **Root** (см. выше). В древообразной структуре видны все элементы, размещенные на листе, а при выделении конкретного – транзистора **Q2** в правом окне видны все его выводы и номера цепей (**Net**) к которым они подключены.

Я пропущу часть пунктов верхнего меню, они будут подробно рассматриваться позже а здесь остановлюсь только на нескольких важных на данном этапе пунктах меню **Template** и **System**.

В меню **Template** обратите внимание на пункт **Set Design Default**. Если с назначением цветовой гаммы проекта можно разобраться почти интуитивно, то две опции: **Show Hidden Text** (Показать скрытый текст по умолчанию активна) и **Show Hidden Pins** (Показать скрытые выводы по умолчанию не активна) заслуживают пояснения. Какая на что влияет – показано на рисунке (Рис. 5) стрелками. Очень часто начинающие задают вопрос: как убрать серую надпись <TEXT> рядом с элементом? Очень просто – снять верхнюю галочку. Что это за надпись и чем это чревато? В этом месте появляются свойства, которые вы задаете в окне **Other Properties** при задании свойств конкретного элемента вручную. Если в этом окне пусто – индицируется серая надпись скрытого текста <TEXT>. Но иногда эта опция и полезна. Забегая вперед, покажу. Допустим мне необходимо,

чтобы счетчик стартовал не с нулевого состояния. В окне **Other Properties** я набираю **INIT0=1** (устанавливаю первый триггер счетчика – выход **Q0** в **1**). Если галочка снята, я этого на схеме визуально не увижу, если же нет, то эта надпись будет под счетчиком видна. Теперь о скрытых выводах. Почти все цифровые микросхемы, микроконтроллеры и некоторые другие элементы содержат скрытые выводы питания. По умолчанию им присвоены соответствующие цепи питания VCC/VDD и VSS. Установка галочки **Show Hidden Pins** позволяет увидеть их на схеме. Но это не означает, что я могу к ним теперь подключать терминалы питания или провода. Они по прежнему будут оставаться серыми и не активными (обратите внимание чуть ниже галочек для элементов **'Hidden'** назначен серый цвет). Как сделать их активными я расскажу в теме посвященной визуализации выводов питания.

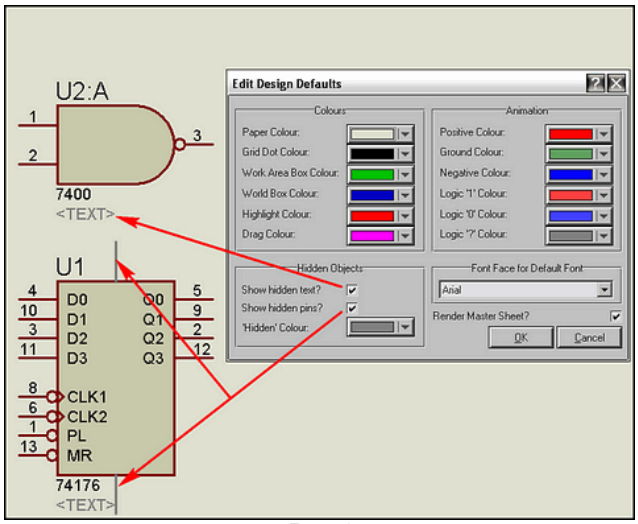


Рис.6

Теперь обратимся к вкладке **System**. Здесь тоже на первом этапе изучения ISIS желательно оставить все «as is», но есть несколько опций, на которых я остановлюсь подробнее. В пункте Set Paths устанавливаются пути к соответствующим директориям программы. Конечно, менять что-либо там себе дороже, но в верхней части раскрывающегося окна имеется переключатель **Initial Folder For Design**. По умолчанию стоит верхний флажок. При этом при сохранении нового проекта Вы неизбежно будете попадать в папку **Samples**, откуда потом придется выбираться кнопками стандартного проводника в нужное вам место на диске. У меня обычно стоит флажок во второй позиции **Initial folder is always the same one that was used last** (открыть папку, которая использовалась последней), потому что проекты я располагаю в разных местах. Но если вы создадите отдельную папку на диске для хранения своих проектов, то лучше поставить флажок в третью позицию и в ставшем при этом активном окне прописать или вручную или через раскрывающееся дерево дисков (щелчком по значку плюс справа в окне) путь к этой папке.

Еще одна полезная функция меню **System** – **Set Sheet Size…** (установить размер листа). Типичная ситуация: Вы рисуете свой проект, увлеклись и с ужасом обнаруживаете, что схема не помещается в синих границах листа (по умолчанию A4 альбомный). Через эту функцию вы выбираете больший, например A3. При этом то, что вы уже набросали на схеме, автоматически центрируется в рамках нового размера листа.

Ну и еще один полезный для начинающих пункт **Set Animation Option…** (установить опции анимации). Вы наверное уже пытались открывать примеры из папки **Interactive Simulation** и обратили внимание на то, как красиво оформлена симуляция: провода в зависимости от потенциала меняют свой цвет, направление тока указывается стрелкой. Вот в этой вкладке (Рисунок 7) и можно добавить такие «фишки» к своему проекту. По умолчанию стоят галочки показа напряжений и токов в пробниках (об этом чуть позже) и показ логических уровней на входа/выходах цифровых микросхем (меняющие цвет квадратики). Добавление галочки напротив **Show Wire Voltage by Color** – раскрасит ваши провода при симуляции, а галочка **Show Wire Current with Arrows** добавит указания направлений токов стрелками. Цвета, которые приняты для данных опций, выбираются в окне **Edit Design Default** в рамке **Animation** справа (рисунок 5 выше).

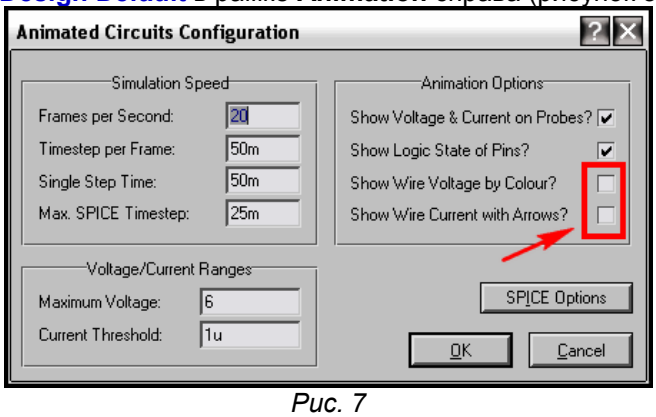


Рис. 7

2.7. Верхние (подключаемые) тулбары.

Под основным верхним меню находится полоса **Toolbars**. Она условно разделена на четыре секции (Рисунок 8), которые, как я уже упоминал, можно включать и выключать через опцию основного меню **View => Toolbars…** По умолчанию все они включены (во всплывающем окне **Show/Hide Toolbars…** отмечены галочками). Сразу оговорюсь, что термин «верхние» весьма относителен, их можно таскать по любым сторонам, как я указывал раньше. Если в случае уменьшения размера окна программы тулбары не помещаются в одной строке, они автоматически выстраиваются в две строки. Так же ведут себя и меню расположенные вертикально слева. При необходимости скрыть один из верхних тулбаров снимаем для него соответствующую галочку в **Show/Hide Toolbars…**. У меня обычно такой чести удостоен набор Файл/Печать, так как кнопки из него используются не так часто и всегда доступны через лишнее «телодвижение» в верхнее меню **File…** Все кнопки тулбаров при наведении на них курсора мышки имеют всплывающие текстовые подсказки, однако они не всегда адекватны аналогичным в основном меню. Чтобы окончательно покончить с тулбаром Файл/Печать приведу простой пример, упущенный мною ранее. Кнопка **Mark Output Area** (выделить выводимый на печать участок) – самая правая в верхнем ряду (Рис. 8) в меню **File…** носит название **Set Area**. Поэтому не удивляйтесь далее таким «сюрпризам» в интерфейсе программы. Здесь я бегло дам назначение остальных кнопок верхних тулбаров, а их использование будет подробно рассмотрено при описании редактирования проектов. Если где то и повторюсь, это не криминально, так как чаще я буду упоминать наиболее употребимые, и Вы их скорее запомните.

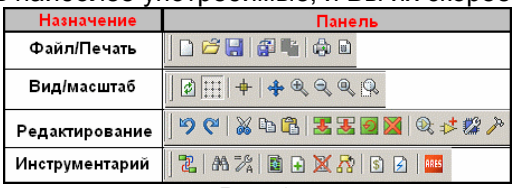


Рис. 8.

Вид/Масштаб (слева – направо):
Redraw Display (**R**- здесь и далее в скобках кнопки клавиатуры) – обновляет окно программы. *Круговые зеленые стрелки*
Toggle Grid (**G**) – включает/выключает показ сетки. *Точечная сетка.*
Toggle False Origin (**O**) – включает/выключает мнимую точку начала координат. Потребуется при создании графических моделей. *Прицел.*
Center At Cursor (**F5**) – центрирует изображение на экране по указателю курсора (щелчок левой кнопкой мышки). *Четыре голубые растягиваюшие стрелки.*
Zoom In (**F6**) – Увеличить масштаб. *Лупа с плюсом.*
Zoom Out (**F7**) – Уменьшить масштаб. *Лупа с минусом.*
Zoom To View Entire Sheet (**F8**) – разместить на экране лист целиком. В меню **View** опция **Zoom All…** *Лупа с мелким квадратом.*
Zoom To Area – разместить на экране выделенный регион (щелчок левой кнопки мыши первая точка, повторный вторая по диагонали выделяемого региона). *Лупа с белым прямоугольником.*
Редактирование (слева – направо):
Undo **(Ctrl+Z)** –отмена последнего действия. По умолчанию допустимо откатить до 20 шагов. Меню **System => Set Environment…** *Голубая стрелка против часовой.*
Redo **(Ctrl+Y**) – возврат последнего действия. Активна только после **Undo**. *Голубая стрелка по часовой.*
Три стандартные кнопки, начиная с ножниц, **Cut, Copy, Paste** - вырезать, копировать в буфер, вставить из буфера в пояснении не нуждаются. Как и четыре последующих кнопки операций с блоками, становятся активными только при выделении элемента или участка схемы левой кнопкой мыши. Однако, в отличие от того же **MS Word**, не имеют стандартных сочетаний клавиатуры типа **Ctrl+X** и т.д.
Block Copy – копировать блок (*два зеленых прямоугольника с вертикальной красной стрелкой вниз)*. **Block Move** – переместить блок *(кнопка аналогична предыдущей, только верхний прямоугольник прозрачный).*
Block Rotate – позволяет через всплывающее окно повернуть выделенный блок (элементы 2D графики) на заданный угол или отразить его (галочка **Mirror**) по оси **X** или **Y** *(зеленый прямоугольник с круговой стрелкой против часовой)*. Для этого проще использовать всплывающее меню правой кнопки мыши. Причем в нем автоматически изменяются опции для элемента и блока (участка схемы). Еще один нюанс, поворот на заданный угол доступен только для элементов 2D графики, а например резистор или диод установить можно только горизонтально или вертикально. Через меню правой кнопки отдельный элемент вертится только на 90 градусов. **Block Delete** **(Del)** – стирает выделенный блок/элемент *(зеленый прямоугольник с иксом).*
Pick Parts From Library – выбрать объект из библиотеки Протеуса *(лупа с мнемоникой ОУ – треугольник внутри)*. В отличие от аналогичной по действию кнопки вверху селектора объектов – **P**, всегда переносит нас в библиотеку электро-радиокомпонентов. Кнопка **P** – изменяет свое действие при выборе режима селектора. Мы это рассмотрим позже.
Make Device – создание нового устройства (компонента) из выделения (*мнемоника ОУ с символом плюс)*. Рассмотрим подробно при создании собственных моделей.

Packaging Tool – инструментарий назначения типа корпуса *(гаечный ключ на фоне синего корпуса микросхемы)*. Рассмотрим одновременно с предыдущим.
Decompose – разбить компонент на составляющие. (молоток – ну очень правильная мнемоника) Противоположен по действию **Make Device**. Пока пропустим.
Инструментарий (слева – направо):
Toggle Wire Autorouter (W) – включить/выключить автоизменение трассы провода *(два зеленых прямоугольника с красной и зеленой трассами)*.
Search Tag Components (New) (T) – поиск и выделение компонента *(бинокль)*. Почему **New** – непонятно, он во всех семерках **New**. Давайте рассмотрим сейчас его действие. При щелчке вызывает указанное окно (Рисунок 9). В принципе все должно быть понятно из красных комментариев на рисунке. Добавлю только, что в рамке **Search Mode** (Режим поиска) можно выбрать, например **Add to List** (добавить к списку внизу) и меняя **String** щелчками кнопки **Seach** набрать конкретный список для выделения (Пример: R2, C3, U10). Затем щелкаем **Done** и все эти элементы подсветятся красным. В правой рамке **Matching Mode** выбирается условие поиска: **Equals** – совпадает, **Begins** – начинается, **Contains** – содержит. Флажок **Case sensitive** – чуствительность к регистру букв.

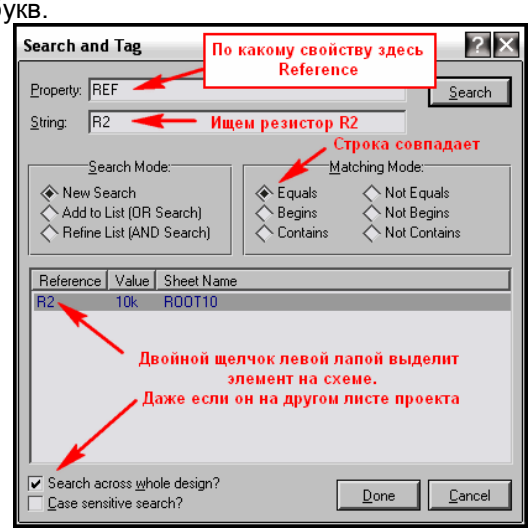


Рис. 9

Property Assignment Tools (A) – инструментарий назначения свойств *(гаечный ключ с символом равно и буквой А)*. Я уже упоминал его, и мы очень плотно займемся им в ближайшее время при создании проекта.
Design Explorer (Alt+X) – кнопка вызывает окно браузера проекта и была подробно рассмотрена раньше *(черный прямоугольник с бирюзовой мнемоникой)*.
Далее следуют две кнопки добавления **New (Root) Sheet** *(лист с плюсиком)* и удаления **Remove/Delete Sheet** листа *(лист перечеркнутый красным иксом)* проекта, назначение которых понятно из пояснений ранее.
Следующая кнопка **Exit To Parent Sheet** - возврат на родительский лист становится активной только когда Вы находитесь на дочернем листе модуля или проектируемого компонента и служит для выполнения означенного действия *(ветвящаяся желтая блок-схема)*.
View BOM Report – генерирует нечто аналогичное спецификации или перечня элементов проекта, который можно сохранить в виде HTM-файла *(лист со значком доллара)*. Через верхнее меню Tools… можно выбрать генерацию в другом формате, например ASCII – текстовом.
View Electrical Report – тоже отчет, но об проверке валидности электрических соединений (лист с голубой молнией).
Ну и наконец **Netlist Transfer To ARES** – передача созданной нами схемы в виде списка соединений в ARES для создания печатной платы *(красный квадрат с надписью ARES)*.

2.8. Набор кнопок левого тулбара. Связь их с селектором объектов и окном предпросмотра.

В отличие от верхних левый набор кнопок отключить нельзя. Для уменьшения размеров рисунка я перетащил его на горизонталь (Рисунок 10). Так он меньше места занимает, да и описывать мне его удобнее слева направо. Четкого разделения по функциональному назначению в этом тулбаре нет, поэтому я условно разделил его так, как проведены границы в Протеусе на три набора. Но это просто для удобства описания. Плюс к тому по умолчанию там же расположены кнопки поворота/отражения объекта. У меня на рис. 10 они получились внизу. Здесь я кратко, как и в предыдущем параграфе, приведу назначение кнопок, а подробнее мы столкнемся с ними при редактировании проектов и создании моделей. Чуть не забыл еще одно основное свойство левого тулбара – кнопки не имеют дублирующего вызова функций с клавиатуры. Так что здесь все действия возможны только мышкой. В скобках за названием кнопки размещено описание ее вида в меню. Итак:
Набор 1.
Selection Mode – *(жирная черная косая стрелка-указатель)* – режим выбора. В этом режиме в окне редактирования единичным щелчком левой кнопки мыши по объекту (компоненту, проводу,

шине, графическому элементу) вы можете выделить его – он становиться красным, а удерживая левую кнопку нажатой и обводя группу объектов можно выделить блок. Кроме того, из этого режима возможно проведение соединительных линий проводов между выводами компонентов или от выводов к шинам. В окне предпросмотра при этом виден уменьшенный лист проекта (синяя рамка) и положение текущего окна редактирования (зеленая рамка).

Component Mode – *(кнопка с изображением желтой мнемоники ОУ)* – режим выбора/размещения компонентов. В этом режиме компоненты из селектора объектов размещаются в окно редактирования. При выборе требуемого компонента в селекторе его вид отображается в окне предпросмотра. Вот здесь и вступают в действие кнопки предварительного поворота/отражения объекта. С помощью их можно выбрать в каком положении будет размещаться объект на поле в окне редактирования. В окне предпросмотра это положение будет отражено (Рис. 10). В режиме **Component Mode** первый щелчок левой лапой мыши по полю окна редактирования вызывает подсветку контура размещаемого объекта, а второй щелчок устанавливает его на выбранное место. Также как и в предыдущем режиме доступно проведение проводов между выводами компонентов.

Junction Dot Mode – *(прицел с синим квадратиком)* – режим расстановки точек соединения на проводах. Думаю в лишних комментариях не нуждается. Расстановка как и в предыдущем режиме на два щелчка мыши: подсветка, установка.

Wire Lebel mode – *(кнопка LBL)* – режим текстовой маркировки проводов в проекте (замечу, что и шин тоже). О нем подробнее будет в разделе редактирования проекта. При наведении курсора на маркируемый провод/шину под изображением карандаша появляется косое перекрестие, после чего щелчок мышью вызывает окно редактирования **Edit Wire Lebel**. В окне **String** проводнику присваивается уникальное в рамках проекта имя, либо выбирается из уже имеющихся через раскрывающийся список по стрелке справа от окна **String**.

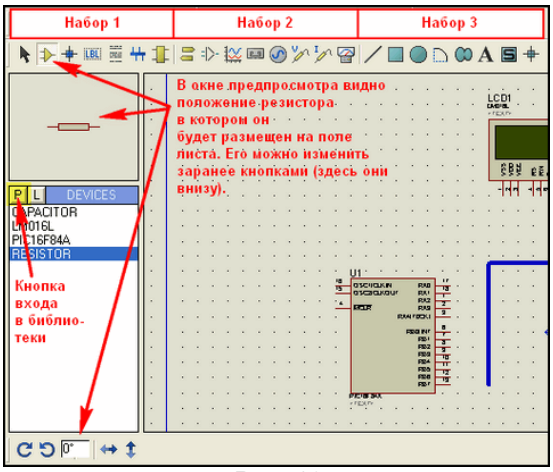


Рис. 10

Text Script Mode – *(горизонтальные пунктиры, изображающие текст)* – режим размещения текстовых скриптов (простых многострочных текстов). Щелчок по свободному полю в проекте вызывает всплывающее окно встроенного редактора текста **Edit Script Block** (Рисунок 11). В окне **Text** набираем текстовый блок. Допустим импорт текста из текстовых файлов или наоборот экспорт (очень удобная функция при создании собственных моделей) через соответствующие кнопки внизу справа. Переключателями **Rotation**, **Justification** выбирается расположение/ориентация текста в проекте (а не здесь в окне **Text** – не путайте). Если **Wire Autorouter** в верхнем меню (**Инструментарий** рис. 8) выключен, шины можно протягивать не только под прямым углом, но и наискось. Более добавить нечего.

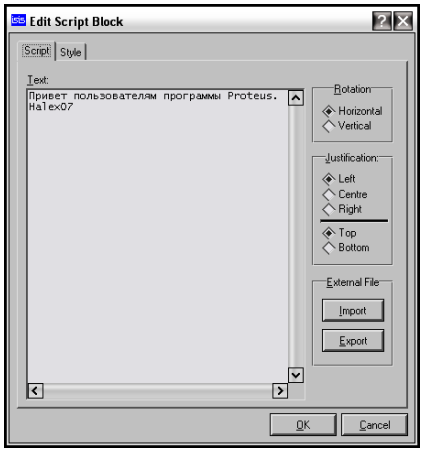


Рис. 11

На рисунке 12 цифрами показана последовательность превращения текста скрипта в жирный **(Bold)** красный на вкладке **Style** окна редактора **Edit Script Block**. Аналогичными вкладками **Style** обладают и другие объекты ISIS, например 2D графика (Набор 3 на рис. 10), но набор функций немного другой. Я это подчеркиваю к тому, чтобы в дальнейшем не останавливаться на том как поменять стиль: цвет, толщину линий, заливку и т.п. Последовательность действий везде одна и та же: сначала снять флажок, затем изменить параметр, который при этом становиться доступным для изменения.

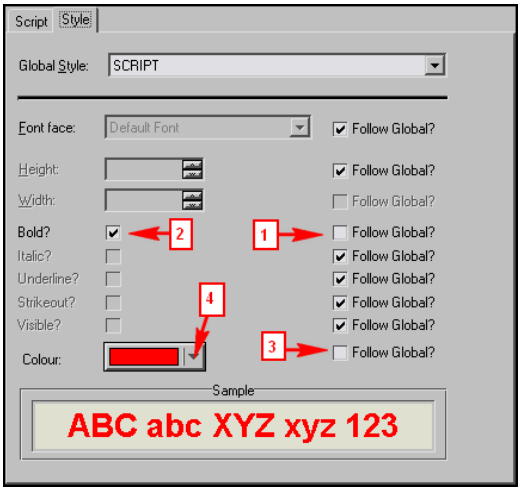


Рис. 12

Buses Mode – *(горизонтальная синяя шина с отводами вверх/вниз)* – режим рисования соединительных шин. Первым щелчком левой в нужной точке проекта стартуем начало шины, последующими одиночными ставим точки поворотов, двойным щелчком завершаем рисование.

Subcircuit Mode – *(желтый прямоугольник с выводами справа и слева)* – режим размещения субмодулей. Модули – прямоугольники с толстой синей окантовкой и заливкой цветом компонентов – позволяют в Протеусе вынести функционально законченные узлы на отдельные листы (**Child Sheet** - дочерний лист модуля). Кто смотрел примеры, прилагаемые к Протеусу, уже сталкивался с ними в сложных проектах. Их применение мы рассмотрим подробно в разделе иерархических структур. А здесь отмечу, что рисуется модуль удержанием нажатой левой кнопки мыши в окне проекта по диагонали с угла на угол. После чего через изменение свойств доступно присвоение ему индивидуального имени (по умолчанию подставляется **SUB?**). При установке **Subcircuit Mode** в окне селектора становятся доступными для выбора терминалы (порты ввода/вывода и питания) субмодуля. Расстановка выбранных терминалов возможна по левой и правой вертикальной синей окантовке модуля. По неписаным канонам принято входы (**Input)** ставить слева, а выходы (**Output)** справа. Изображение терминала появляется в окне предпросмотра при выборе его в селекторе.

Набор 2.

Terminal Mode – *(два желтых горизонтальных указателя вправо/влево)* – режим расстановки терминалов. Терминалы позволяют связать две или несколько точек схемы, расположенных как на одном листе, так и на разных листах, не проводя между ними соединительной линии. Для этого в свойствах (**Properties**) связанных между собой терминалов им указывают одинаковые имена. Имена указываются в окне String вручную или выбираются из уже назначенных через выпадающее меню при щелчке по стрелке справа в окне **String**. В окно свойств попадаем при двойном щелчке по установленному в схему терминалу, либо через правую кнопку мыши выбрав опцию **Edit Properties** (**Ctrl+E**). Выбранный в селекторе терминал доступен для предпросмотра перед установкой в окне **Preview**. Его положение можно изменять кнопками поворота/отражения. Еще одно важное замечание: выбор **Default**, **Output**, **Input** и **Bidir** влияет только на изображение терминала на схеме. Симулятору ISIS абсолютно безразлично какой из этих терминалов установлен – одноименные способны пропускать сигнал в любом направлении. Так что не уповайте на то, что если вы установили на выходе микросхемы терминал **Output**, то назад в микросхему он сигнал не пропустит – типичное заблуждение начинающих. Если терминалы **Power** и **Ground** после установки не поименованы особо, то они считаются подключенными к глобальным для проекта питанию и земле.

Отдельное замечание по терминалу **BUS** для шин, касающемуся также и шинных маркировок (режим **LBL**). Наменование терминала формируется так: ИМЯ**[**НАЧ_РАЗРЯД**..**КОН_РАЗРЯД**]**. Здесь ИМЯ – наименование шины латиницей без пробелов и спецсимволов, НАЧ_РАЗРЯД и КОН_РАЗРЯД два числа, из непрерывного возрастающего ряда, определяющие разрядность данного терминала. Обратите внимание, что скобки обязательно квадратные и между начальным и конечным разрядом ставится ДВЕ, а не три точки, как принято у нас в сокращениях. Как это выглядит на практике. Допустим, есть шина адреса с именем (лэйблом) **A[0..15]**. Поместив на ее конце, или отводе терминал с именем **A[0..15]** я получу на одноименном терминале шины в другом месте схемы все 16 разрядов сигнала, которые потом через отходящие от шины провода с именами A0, A1 и т.д. могу развести по компонентам. Если же я размещу на отводе **A[0..15]** терминал **A[8..11]**, то поместив терминал с таким же именем в другом месте, я получу на нем только четыре выделенных разряда A8, A9, A10 и A11, которые и могу растащить дальше одноименными проводами. Если кто-то не включился, я позже вернусь к этому вопросу наглядно в проекте.

Device Pins Mode – *(расчлененное изображение ОУ на сером фоне)* – режим расстановки выводов модели устройства (компонента) при его создании. Мы его рассмотрим подробно в соответствующем разделе.
Graph Mode – *(две оси координат с синусоидами)* – режим размещения графиков в проекте. Возможные варианты анализа с помощью графиков выбираются в окне селектора: ANALOGUE, DIGITAL и т. д. Окно соответствующего графика растягивается по полю проекта зажатой левой кнопкой мыши диагонально. Анализу с помощью графиков будет подробно посвящен целый раздел далее.
Tape Recorder Mode – (изображение магнитофонной кассеты) – режим установки магнитофона. Сигналы, генерируемые разработанным Вами устройством, в процессе симуляции можно записать в файл с последующим использованием их в другом проекте. Для этого и служит этот виртуальный магнитофон. Позже мы вернемся к вопросу его применения.
Generator Mode – синусоида в окружности – режим расстановки виртуальных генераторов. В этом режиме в окне селектора доступны для выбора и размещения в схеме виртуальные генераторы сигналов. Условно их можно разделить на цифровые (все начинающиеся с букы D, за исключением DC – постоянный потенциал) и аналоговые (все оставшиеся). Отдельно отмечу **SCRIPTABLE** – програмный генератор, для которого предварительно надо написать скрипт – программу на встроенном в Протеус языке **Easy HDL**. Два генератора **FILE** и **AUDIO** используют для генерации сигналов файлы, предварительно записанные на жесткий диск. Вопросам использования генераторов будет посвящена отдельная тема позже. Здесь же отмечу, что установка выбранного генератора осуществляется или на свободное место схемы двумя последовательными щелчками (подсветка-установка), или сразу на провод после чего в свойствах ему задаются требуемые параметры. В окне предпросмотра видно изображение генератора и доступны поворот и отражение. При последующем подключении к выводу компонента или проводу генератор автоматически изменит свое имя на имя соответствующего ближайшего вывода. Если оно Вам по каким-то причинам не нравится, через окно свойств генератора его можно переименовать по своему усмотрению.
Voltage Probe Mode (желтый щуп с буквой V) и **Current Probe Mode** (желтый щуп с буквой A) – два режима расстановки пробников соответственно напряжения и тока на провода схемы. Пробники ставятся именно на провода, а не на выводы компонентов и аналогично генераторам автоматически меняют свои имена. Если воткнуть пробник на пустое место он вместо имени высветит знак вопроса. Установка токовых пробников на цифровые цепи бессмысленна, пробники напряжения на этих цепях будут индицировать не значение напряжения, а логический уровень сигнала. Еще один нюанс для токовых пробников – стрелка в кружке должна быть ориентирована вдоль провода. Если направление тока в проводе совпадает, значение тока при симуляции будет положительным, если нет – отрицательным.
Virtual Instruments Mode – (*кнопка с изображением стредочного прибора)* – режим выбора и размещения виртуальных инструментов в проекте. В Протеусе, как и во многих других программах-симуляторах, имеется обширный инструментарий виртуальных приборов: вольтметры, амперметры, четырехканальный осциллограф, счетчик/частотомер, сигнал-генератор и др. специфичные приборы. В этом режиме осуществляется их выбор и размещение в проекте. Мы будем обращаться к ним по мере изучения ISIS, а пока отмечу, что большинство из них, за исключением вольтметров/амперметров имеют однополюсное подключение. Это означает, что измерение осуществляется ОТНОСИТЕЛЬНО земляного провода. Об этом не стоит забывать начинающим, чтобы не получить нереальные значения измерений. В примерах Протеуса, о которых шла речь выше есть проекты с использованием виртуальных приборов и всегда можно посмотреть: как оно действует в реальности. Единственное замечание – там использована двухканальная модель осциллографа из старых версий Протеуса. Примененная в ранних седьмых версиях четырехканалка «слегка» глючила, сейчас вроде постепенно ее довели до ума. И еще одно замечание – виртуальные приборы «съедают» определенное количество ресурсов компьютера и в сложных проектах могут стать причиной тормоза симуляции в режиме реального времени. Об этом тоже следует помнить.
Набор 3.
Все кнопки данного набора относятся к режиму 2D (двухмерной) графики. В общем те, кто имел дело с графическими редакторами, без труда догадаются что к чему. Но бегло поясню по рис. 10 слева направо: линии, прямоугольники, круги, дуги, замкнутые полигоны. Те из них, которые имеют в изображении зеленую заливку – могут заливаться. Немного про полигоны (восьмерка с заливкой на боку). Тут Лабцентр слегка «загнул» насчет мнемоники. Дело в том, что доступны для рисования только многоугольники, что обычно на кнопке изображается звездочкой или пятиугольником. Так что не надейтесь получить такую фигуру с плавными кривыми и заливкой, как на мнемонике кнопки – не выйдет. Далее следует кнопка размещения текста (мнемоника A), и на этом стандартные графические кнопки кончаются. По ним только одно существенное замечание. По умолчанию все они находятся в режиме **Global Style** – **Component**. Это означает, что нарисованные фигуры и линии по цвету будут соответствовать бордюру микросхем - коричневый, а заливка будет цвета хаки. Изменить цвета можно через окно свойств после прорисовки графики как я рассказывал ранее, либо заранее выбрав в селекторе объектов другой глобальный стиль. При этом в окне предпросмотра отображается как будет выглядеть нарисованный элемент. Редактирование же самих глобальных стилей возможно через верхнее главное меню **Template** => **Set Graphic Styles…**

или щелчком правой кнопкой по выбранному стилю в селекторе и выборе из всплывающего меню опции **Edit** (Редактировать). Но без особой надобности этого делать не советую. Проще создать свой новый стиль. Для этого, щелкнув в любом из режимов 2D графики в окне селектора объектов правой кнопкой мыши, выбираем опцию **Create** (Создать). Даем название стилю и жмем **OK**, а потом выбрав его в селекторе через правую кнопку редактируем как и выше.
Ну и остались нерассмотренными две кнопки, которые понадобятся при создании моделей:
2D Graphics Simbol Mode – *(буква S на зеленом квадрате)* – режим выбора и размещения графических символов.
2D Graphics Markers Mode – *(прицел с зеленым квадратом*) – режим выбора и размещения графических маркеров.
Пока по этим двум режимам поясню, что при нажатии кнопки P вверху селектора объектов из этих режимов вы попадете в библиотеки стандартных графических объектов ISIS, а не в библиотеки компонентов.
На этом наш экскурс по интерфейсу основного окна ISIS мы завершаем и переходим к более интересным темам – созданию и редактированию проектов.

2.9. «Как пройти в библиотеку? В три часа ночи?» — (к/ф «Операция Ы»).

Но прежде необходимо определиться с тренировочным проектом. Поскольку я преследую цель научить Вас быстро и красиво оформлять проект в ISIS, необходимо чтобы наш первый проект позволял показать все приемы и хитрости быстрой и качественной работы. Но и использовать бесполезные учебные проекты мне тоже не хочется. После недолгого раздумья я решил остановиться на популярной некогда цифровой шкале-частотомере на PIC16F84 А. Денисова (именно первая версия со светодиодным индикатором). Мы убъём не двух зайцев, а сразу целый табун тушканчиков. Во-первых, я почти уверен, что многие пытались запихнуть его в Протеус и убедились, что он просто так там не работает. Во-вторых, проект содержит и микроконтроллер и устройство индикации – есть что посмотреть при симуляции. Ну и в-третьих, это настолько классическая схема, что найти ее во всемирной паутине не составляет труда, а многие именно с нее начинают знакомство с микроконтроллерами. Я далеко не ходил, вот пара ссылок и одна из них прямо здесь на сайте:
http://www.cgham.ru/digi.htm
http://kazus.ru/shemes/showpage/0/33/1.html
Итак, начинаем с нуля, т.е. с пустого проекта в ISIS. Если под рукой нет принтера и у Вас подключен только один монитор для удобства вычерчивания я рекомендую втащить картинку схемы на лист проекта, иначе постоянно придется переключаться между окнами. Все просто: с любой из вышеупомянутых страниц прямо из браузера IE, Opera (у меня), FireFox клацаем по схеме правой кнопкой мыши и выбираем из менюшки **Сохранить изображение (рисунок) как…** и далее. В результате Вы сохраните его в формате **.gif**. Потом открываем его в любом графическом редакторе, например в стандартном виндозном **Paint** и сохраняем уже оттуда как **.bmp** (можно черно/белый, можно 256 цветов). Вот теперь можно из ISIS через **File=>Import Bitmap…** втащить его на поле проекта и на месте ужать, чтобы было компактно и читабельно.
Еще одно лирическое отступление, перед тем как мы пойдем в библиотеку. Сразу надо определиться для чего мы создаем проект. Если для реального устройства, то нам понадобятся все элементы схемы, причем в тех корпусах, которые у нас в наличии. Но в данном случае нам необходимо исследовать схему в симуляторе, поэтому корпуса элементов нас не интересуют. Кроме того, схему необходимо условно разделить на функционально законченные узлы, которые можно исследовать самостоятельно, поскольку вычислительная способность компьютера ограничена и нам ее просто может не хватить. В данном конкретном случае мы можем отсечь входной формирователь на транзисторе VT1 со всей его обвязкой – его желающие могут погонять самостоятельно и стабилизатор +5V на микросхеме 7805 – питание у нас виртуальное, поэтому данное излишество только помешает. Все модели микроконтроллеров в ISIS симулируются независимо от наличия внешних частотозадающих цепей (кварц или RC), так что и цепи кварцевого генератора тоже можно не рисовать, но я умышленно оставлю их в проекте, чтобы показать: как их исключить из процесса симуляции. Ну и еще одно упрощение – я не буду использовать токоограничивающие резисторы R2…R12 в цепи индикатора, а почему – поясню чуть позже.
В результате нам понадобятся следующие компоненты: PIC16F84A, восьмиразрядный семисегментный индикатор с ОК (аналог АЛС318), дешифратор 555ИД7 (которого конечно в библиотеках нет, но есть аналог – 74LS138), резисторы, конденсаторы и кварц. Пора идти в библиотеки Протеуса. «Снайперы» – заходят через одиночный щелчок левой лапкой мышки по кнопке P вверху окна селектора объектов, либо через кнопку верхнего тулбара, описанную раньше. Я поступаю проще – двойной щелчок левой в любом месте внутри селектора объектов – эффект тот же и снайпером быть не надо. В результате нам откроется окно браузера библиотек (Рисунок 13). Далее есть два пути поиска нужного нам компонента. Если мы хотим подобрать компонент, например биполярный NPN транзистор с подходящими характеристиками, тогда в окне **Category** – выбираем **Transistors**, в окне **Subcategory** – **Bipolar**, ну и я рекомендую при этом в окне ключа поиска **Keywords** ввести **npn**, чтобы отобрать только транзисторы с обратной проводимостью. После этого в списке доступных моделей отбираем подходящую (для многих в **Description** указаны краткие характеристики) и двойным щелчком по этой строчке отправляем ее в селектор объектов

ISIS. Некоторые модели имеют несколько исполнений корпусов, которые можно просмотреть через **PCB Preview**. Другой способ поиска более быстрый и прогрессивный представлен в анимации на рис. 14. В окне поиска вводим часть названия компонента и номинал, например, для резистора 47кОм вводим **res** (начало английского **resistor**) и номинал 47k. Список в центре при этом резко сужается, и отобрать нужный компонент не составляет труда.

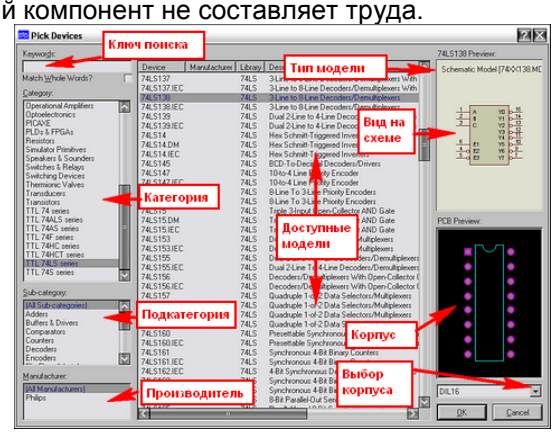


Рис. 13

Я рекомендую так поступить с микроконтроллером – введите **16f84** или **f84** и в списке компонентов окажется только нужная нам модель.

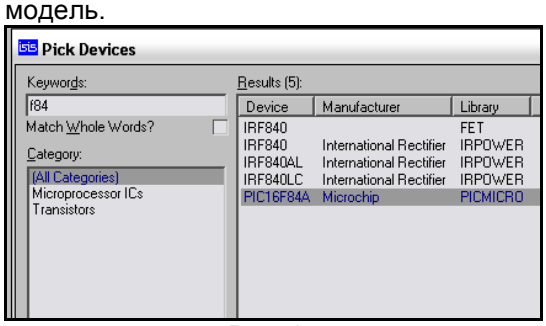


Рис. 14

2.10. Подбираем компоненты, расставляем их в проект.

Набираем нужные нам элементы из библиотеки двойным щелчком левой кнопкой мыши по найденным. В соответствии со схемой нам потребуются:
м/контроллер **PIC16F84A** – ключ быстрого поиска: **f84**;
7-сегментный индикатор с общим катодом – ключ: **7seg cc** – выбираем **8 digit**(цифр/разрядов);
Дешифратор **74LS138** – ключ: **ls138** – здесь обратите внимание будет присутствовать модель **74ALS138** для которой тип стоит **No Simulation** (не симулируется!!!) – для симуляции ее выбирать нельзя;
резисторы **10кОм** и **470 Ом** – ключи поиска соответственно: **res**(istor) **10k** и **res 470r**;
конденсаторы **15пФ** – ключ: **cap**(acitor) **15pf**;
триммер **4…15пФ** – ключ **cap pre** – опять **No Simulation** – но мы его и не будем симулировать;
кварц – ключ **cryst**(al);
кнопка – **butt**(on) – на схеме нет, но надо же нам получить элементы управления для исследования поведения схемы в соответствии с описанием в режиме цифровой шкалы.
Ключ быстрого поиска – это тот кусочек текста, который надо набрать в окошке **Keyword**, чтобы не просматривать весь список компонентов и не портить себе зрение. В скобках я указал, что **res** – это начало от английского resistor – резистор, **cap** – capacitor – конденсатор ну и т.д. Совсем не обязательно соблюдать регистр букв и последовательность расстановки отрезков текста в конечной фразе. Например поиск **7seg cc** даст тот же результат, что и **cc 7seg**, главное разделить их пробелом, а вот слитное написание **7segcc** не найдет ничего, т.к. будет искать целиком этот сочетание. Когда Вы познакомитесь с библиотеками поближе, такой поиск не будет доставлять Вам лишних хлопот. После того как мы разыщем все нужные компоненты и «прощелкаем» их, браузер библиотек можно закрыть щелчком по крестику вверху справа или по кнопке **Cancel** внизу справа. Все выбранные нами компоненты должны оказаться в селекторе объектов. Теперь можно разместить их на листе проекта. В режиме **Component Mode** встаем в селекторе на требуемый компонент – он появляется в верху в окне **Preview** – если надо заранее поворачиваем или отражаем его, щелкаем левой кнопкой в поле проекта – компонент подсвечивается – выбираем место установки и вторым щелчком фиксируем его. На рис. 15 я постарался показать - как это выглядит полностью с уже размещенными проводами и шинами. Обратите внимание, что по сравнению с исходной схемой произошли значительные изменения. Добавились кнопки **SB1...SB3** вместо перемычек **J3** и **J4** у автора. Особенно отмечу кнопку **SB3** состоящую из двух. ISIS при установке кнопок в проект не поддерживает для них автонумерацию и обозначение. Они введены вручную. Для чего мне понадобилась кнопка **SB3**, да еще со странным свойством **GANG=1** (рис. 16). По замыслу автора один из режимов должен включаться одновременным замыканием **J3** и **J4** на

землю. В ISIS мы управляем нажатием кнопок курсором мыши и левой кнопкой, но курсор то всего один, а замкнуть надо сразу две точки. Вот для этого и служит свойство **GANG (**произносится как в боксе, а не как индийская река), которое позволяет засинхронизировать нажатие нескольких кнопок или работу переключателей. Для другой синхронной группы нужно назначить **GANG=2** и т. д.

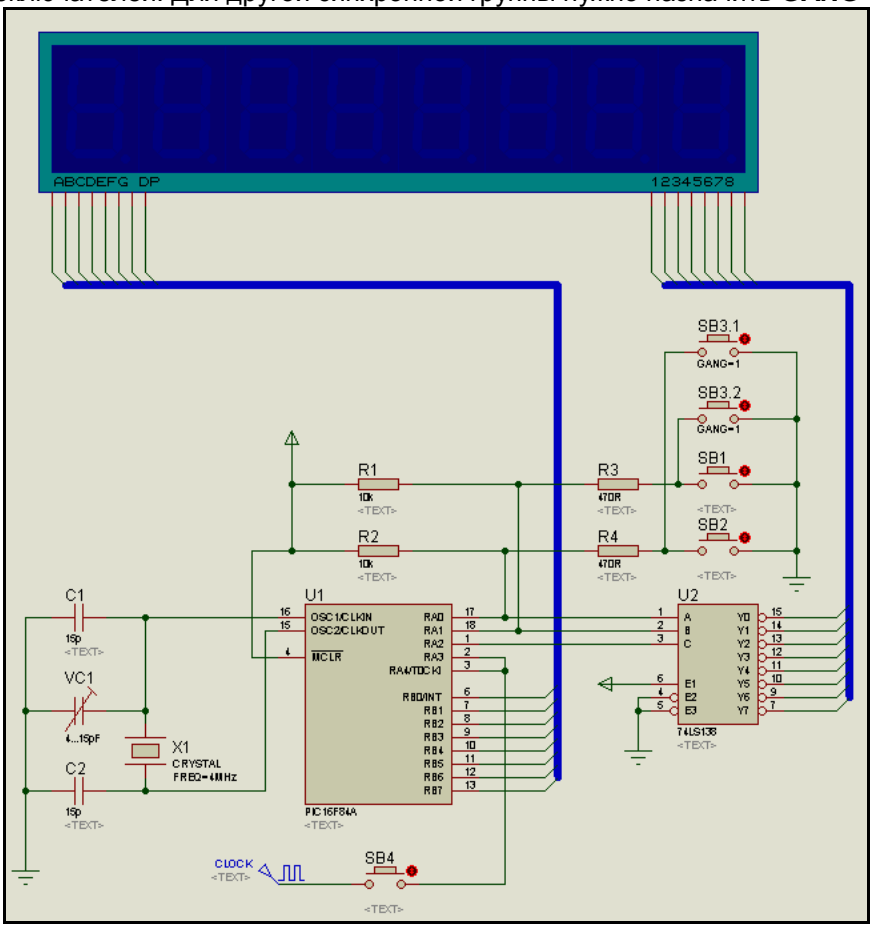


Рис. 15

Кроме того, для элементов кварцевого генератора: **C1, C2, VC1** и **X2** я заранее установил в свойствах галочку (Рис. 16) **Exclude From Simulation** (исключить из симуляции). Если этого не сделать, то ISIS выдал бы нам при запуске симуляции ошибку для подстроечного конденсатора – ведь он **No Simulation**. Да и модель кварца в Протеусе оставляет желать…, но об этом позже. Эти элементы вообще можно было не устанавливать, ведь для микроконтроллеров в Протеусе принята программная симуляция генератора и задается она в свойствах микроконтроллера в соответствующей строке: **Processor Clock Frequency**. Еще одна особенность – я торжественно «похоронил» входной формирователь, а вместо него подключил через кнопку **SB4** цифровой генератор **DCLOCK** из левой панели (режим **Generator Mode**).

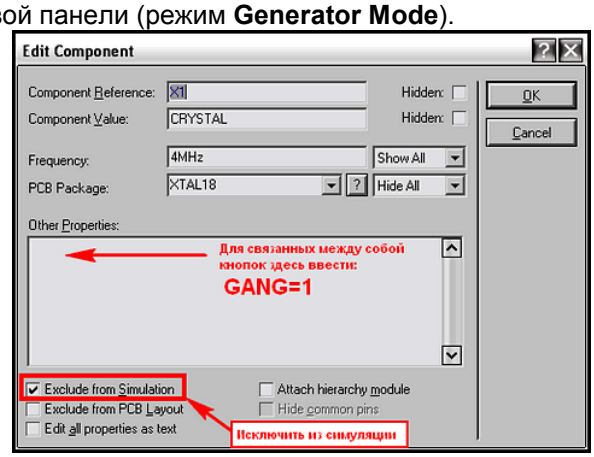


Рис. 16

Ну и последнее, что бросается в глаза на рисунке 15 – провода от шин не имеют обозначений. Да у Вас то они еще и не должны быть нарисованы – я слегка опередил события. В следующем разделе мы ими и займемся. А пока во вложении проект в том виде, как на рис. 15. Проект для версии Протеуса 7.4SP3, но там же лежит файл **Section_0.SEC** , который можно через **File=>Import** импортировать в более ранние версии. Там же помещен и текстовый файл описания схемы, картинка в формате BMP, а также ассемблерный файл исходной программы **DigiScal.asm** и прошивка **DIGISCAL.HEX** для микроконтроллера.

2.11. Приемы быстрого редактирования. Разводка проводов и шин.

Ну вот мы и подошли к тому материалу, из-за которого главным образом и затевался раздел для начинающих. Разводка одиночных проводов как правило не вызывает особых затруднений. Здесь важно подчеркнуть одну особенность – почти не важно в каком режиме находится меню селектора объектов: **Component Mode**, **2D Graphic** или другом. Если из этого режима возможно проведение одиночного провода, то при наведении курсора на начальную точку – он автоматически встанет в режим рисования проводов – курсор – зеленый карандаш (Рис. 17). Если начальная точка – вывод элемента, то она подсветится красным квадратиком – как на рисунке, если это шина или одиночный провод – середина подсветится тонким красным пунктиром (на анимации Рис. 18 это видно при проведении верхнего провода к шине). Первым щелчком левой кнопкой мыши стартуем начало рисования и ведем курсор к конечной точке, где фиксируем окончание повторным нажатием левой кнопки. Если включен режим **Wire Autorouter** (на Рис. 18 я его выключаю), то провод прокладывается строго под прямыми углами и автоматически обходит все «зарезервированные» места, т.е. компоненты, текстовые скрипты и т.п. Таким образом, достаточно щелкнуть начальную и конечную точки, и провод сам ляжет на схему, но при этом «как бог (точнее богиня ISIS) на душу положит». Я категорически против такого насилия над человеческими мозгами, поэтому рекомендую при проведении провода фиксировать привязку самих проводов и поворотов в нужных местах одиночными щелчками левой кнопкой. Это ограничивает действия автороутера, зато схема получается приличной на вид. Если же режим автороутера выключен, то провод можно прокладывать в любом направлении и под любым углом, даже поверх установленных элементов. Ну и еще один важный момент – в любом режиме провода прокладываются строго по установленной в данный момент сетке (по умолчанию 0,1 Inch = 2,54 мм), т.е. если необходимо провода расположить чаще – измените шаг сетки (меню **View=>Snap…** или соответствующими функциональными клавишами).

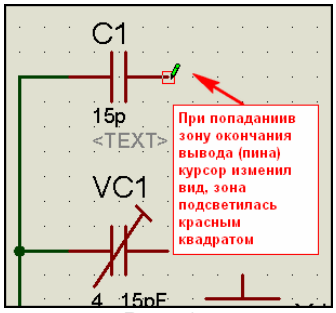


Рис. 17

Шины рисуются как и провода с той лишь особенностью, что в левом меню режима селектора необходимо выбрать **Buses Mode**.

Теперь еще об одной важной особенности ISIS, позволяющей существенно ускорить прокладку однотипных по виду проводов. После прокладки очередного провода (шины) двойной щелчок левой кнопкой мыши в любом месте поля проекта в точности повторит это действие. Что нам это сулит – ясно из анимированного рисунка 18. Здесь я применил эту особенность для прокладки проводов к шине от выходов дешифратора 74LS138. Для «чистоты эксперимента» я даже усложнил Протеусу задачу, заставив его рисовать столь любимые некоторыми подводки к шине с «загибулинами». Для этого сначала отключается режим **Wire Autorouter**, а на месте загиба при прокладке первого провода делается дополнительный щелчок. Остальные провода прокладываются просто двойным щелчком левой лапкой мышки при наведении на окончание вывода микросхемы. Ну и в завершение двойным щелчком уже правой лапкой удаляется ненужный хвост шины. Этот процесс я мог проделать и в обратном направлении, т.е. первый провод провести от шины к выводу и далее щелкать по ней с нужным шагом, но надо точно попадать курсором на пунктир внутри шины.

Вообще я как то упустил этот момент выше, поэтому отмечу здесь: двойной щелчок или два с паузой щелчка правой кнопкой мыши по любому элементу схемы в ISIS (будь то микросхема, резистор, надпись, шина, провод и т.п.) – удаляет этот элемент из проекта. Причем для данного конкретного случая – удалилась не вся шина, а только незадействованный ее отрезок.

Ну и завершая этот раздел, отмечу, что если стартовать провод Вы можете только с конкретных мест: выводы компонентов, другие провода или шины, то бросить его окончание можно в любом месте двойным щелчком левой кнопкой мыши. При этом он завершится соединительной точкой (**Junction Dot**), к которой впоследствии можно подтянуть провод от другого элемента.

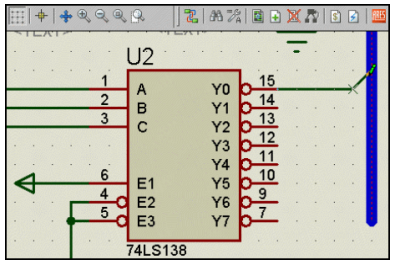


Рис. 18

2.12. Приемы быстрого редактирования. Маркировка проводов и шин. Перенумерация элементов и назначение им свойств с помощью Property Assigment Tools.

Для начала выясним – зачем нужна маркировка (лэйблы) проводов в ISIS и что в этом полезного. Дело в том, что все провода в Протеусе, которые имеют одинаковую маркировку, имеют как бы физическое соединение, даже если видимо не соединены. Маркировка должна быть уникальной и не содержать русских символов, только латинские буквы и цифры. Спецсимволы также лучше не использовать, поскольку многие из них имеют служебное значение и могут быть двояко приняты симулятором **PROSPICE**. Например, если текст наименования терминала, вывода или тот же лэйбл с двух сторон ограничить значком доллара **$**, то текст будет показываться с верхним надчеркиванием, как принято для инверсных сигналов. Это касается именно текста, а не сигнала в проводе, он как был, так и останется.

И еще одно важное замечание. В ряде случаев непрерывные провода можно заменить терминалами - из левого меню режим **Terminal Mode**, чтобы не загромождать схему. Одноименные терминалы считаются Протеусом физически соединенными, например RA2 на Рис. 19 . Причем вид терминала **Input**, **Output** и т. п. не влияет на направление проводимости. Об этом я уже упоминал. Но есть и еще одна особенность для проводов. Помните тот провод, сиротливо оканчивающийся **Junction Dot** из предыдущего параграфа? Даже если такому проводу присвоить **Label**, уже встречающийся в проекте, – ISIS воспримет его, как физически соединенный («припаянный») с одноименными проводами. Эту особенность можно использовать в своих целях, хотя и не очень корректно смотрится провод, идущий в никуда с присутствующим на нем сигналом.

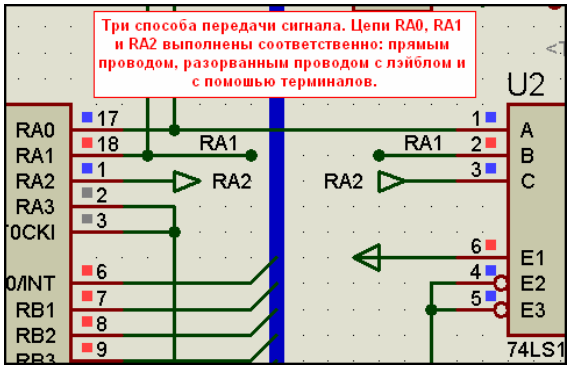


Рис. 19

Одиночные провода конечно проще маркировать вручную. Выбираем слева режим **Wire Label Mode** (кнопка **LBL**), наводим кусор на то место на проводе (шине) в котором надо поставить маркировку – под курсором появится белый крестик и щелкаем левой кнопкой. В результате попадаем в окно **Edit Wire Label** (Рис. 20). Здесь мы либо вручную вводим имя провода, либо выбираем из раскрывающегося списка, если провод в другом месте уже получил нужную нам маркировку. В списке даже для пустого проекта уже доступны стандартные питание и земля: **GND, VCC, VDD, VSS**. Остальные будут добавляться по мере присваивания маркировок проводам в ходе редактирования проекта. В этом же окне можно выбрать ориентировку текста и его положение, впрочем, для вертикальных и горизонтальных проводов ISIS предлагает это автоматически. На вкладке **Style** можно подкорректировать изображение маркировки (цвет, жирность, шрифт и т. п.).

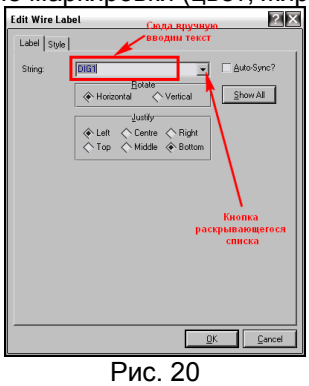


Рис. 20

Все вышесказанное хорошо, когда в схеме не больше десятка проводов, а если их много? Вот тут и пришел черед **Property Assigment Tools**, как было обещано. Это универсальное средство и мы им сейчас научимся пользоваться, поскольку даже в родном **HELP**-е Протеуса ему уделено весьма скромное внимание. Для начала применим его для маркировки проводов, отходящих от шины. Вызовем окно **Property Assigment Tools** (**PAT** далее в тексте и в родном хелпе Протеуса) (Рис. 21) через кнопку с изображением гаечного ключа и буквы A в верхнем меню, или просто нажав латинскую **A** на клавиатуре. Я решил провода, отходящие с выходов дешифратора, назвать в соответствии с индицируемым разрядом: **DIG1** – первый, **DIG2** – второй и т.д. (от английского Digit - цифра). Для проводов, как и для терминалов доступно свойство **NET** (от английского Network – цепь - в данном конкретном случае). В окне **String** вводим свойство, которое будем менять, и через

знак равенства новое его значение – **DIG**. Если необходимо обеспечить автонумерацию, то на то место, где будет располагаться номер ставится знак решетки: **#**, в окне **Count** вводится начальное значение автоотсчета (по умолчанию - 0), а в окне **Increment** – шаг приращения номера (по умолчанию - 1. В рамке **Action** (Действие) оставляем как есть **Assign** (Назначить). В рамке **Apply To** (Применить к …) оставляем **On Click** (На щелчок мышкой). Нажимаем **OK**, чтобы закрылось окно.

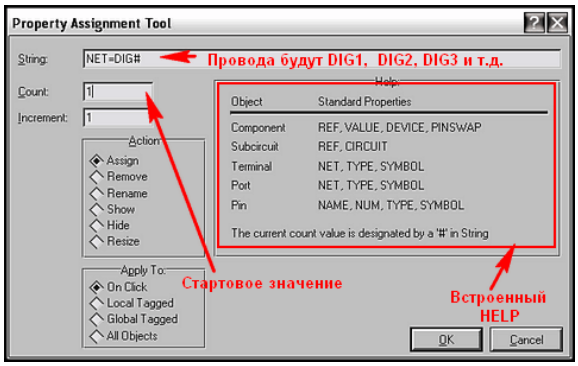


Рис. 21

После этого последовательно проводим курсор по тем выходам дешифратора, которые необходимо промаркировать и делаем одиночные щелчки левой кнопкой, когда курсор принимает соответствующий вид - рука-указатель с зеленым квадратом справа (Рис. 22). Маркировка восьми выходов займет не более восьми же секунд, даже с заторможенной принятой банкой пива реакцией. Проделав данную операцию с выходами дешифратора, снова нажимаем клавишу **A**, или кнопку в верхнем меню, чтобы вызвать окно **PAT**. В окне **String** все осталось по прежнему, а вот в окне **Count** придется поправить начальное значение на **1**, т.к. оно сбросилось в ноль. Снова давим **ОК**, чтобы закрыть окно и повторяем операцию с проводами к разрядам индикатора, маркируя их справа - налево. Чтобы завершить работу с функцией **PAT** придется еще раз вызвать окошко и нажать **Cancel**, иначе она будет действовать бесконечно. Аналогичным способом я промаркировал провода с выходов порта RB **SEG1**…**SEG8**. Конечно, логичнее было бы назвать из **SEGA**, **SEGB** и т.д., но тогда мне пришлось бы проделывать это вручную, а это лишняя трата времени при том же конечном результате. На все операции с маркировкой проводов от шин с использованием **PAT** я потратил менее двух минут – попробуйте проделать это вручную и засеките время.

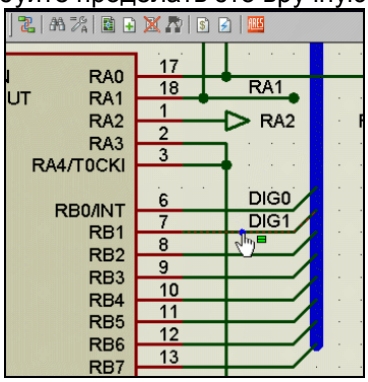


Рис. 22

Ну а дальше как в надоевшей рекламе телемагазина на диване: «Но и это еще не все…, купив у нас пылесос Вы получаете совершенно бесплатно замечательный подарок – огромный мешок пыли для его проверки». Обратите внимание на обведенный рамкой на Рис. 21 встроенный **Help** функции **PAT**. Там кратко перечислено какие свойства для каких объектов можно менять. Кстати, решетка автонумерации не обязательно должна быть в конце строки – где поставите там и будет вставляться номер, например, если так: #SEG, то получим 1SEG, 2SEG и т. д.. В рамке **Action** можно назначить другое действие, например **Rename** – переименовать, **Show** – показать, а в рамке **Apply To** – применительно к каким объектам. Чтобы привести примеры всех возможных сочетаний здесь не хватит места. В качестве закрепления материала попробуем применить функцию для других целей в нашем проекте. Забегая вперед, укажу, что мне необходимо сделать резисторы **R1**…**R4** цифровыми, чтобы снизить загрузку компьютера при симуляции. Для наглядности сначала «подсветим» это свойство резисторов, хотя этого можно было бы и не делать. Вызвав функцию **PAT**, в окошке String наберем **PRIMITIVE**, а в рамке **Action** выберем **Show** - показать. Оставляем по клику мышки и давим **ОК**. теперь кликаем левой лапкой по каждому резистору. В результате внизу под резисторами на месте серой надписи **<TEXT>** появится наше свойство **PRIMITIVE=ANALOG**. Снова вызываем **PAT** и набираем строчку **PRIMITIVE=DIGITAL**. Опять **ОК** и кликаем по всем резисторам. Мы видим, что строчка поменялась. Еще раз заходим в **PAT** и, оставив **String** только **PRIMITIVE**, выбираем действие **Hide** – скрыть. Далее **ОК**, и пробегаем кликами по резисторам. Еще раз в **PAT**, чтобы завершить функцию – **Cancel**. Вы наверное догадались, что первая и третья операции только для наглядности, можно было и не подсвечивать свойство. А можно было подсветить и немного по другому – выбрав в рамке **Apply To** – **All Object**, правда при этом подсветилось бы свойство у всех объектов, где оно представлено, т.е. еще и у кондесаторов, кнопок

и микроконтроллера. Подсвечивать свойство полезно тогда, когда меняем много объектов, чтобы не пропустить нужные и не изменить лишнего. Если выбирать опции **Local Tagged** (на листе) или **Global Tagged** (в проекте), то объекты перед вызовом **PAT** необходимо выделить. Ну и завершая тему **PAT**, еще два типичных частых применения функции: перенумерация компонентов по клику в окне **String** набираем, например, для резисторов: **REF=R#**, начальное значение ставим **1** и проходим по резисторам в том порядке в котором необходима нумерация в схеме. Изменение номинала резисторов: задаем в строке **VALUE=270R** – щелкаем по R3 и R4, вместо 470 Ом получили 270 Ом. Можно менять и пользовательские свойства. Задаем в **String** – **GANG=2**, щелкаем по кнопкам **SB3.1** и **SB3.2** и видим, что гонги изменились. Вот такой могучий инструмент заложен внутри программы ISIS. Надеюсь, после данной публикации на форум будут меньше выкладывать «кривых» проектов, а скорость разработки у прочитавших и освоивших этот материал существенно повысится.

2.13. Свойства моделей микроконтроллеров. Задание численных значений и размерности.

С оформлением схемы мы как будто закончили. Осталось разобраться со свойствами микроконтроллера, «зашить» его микропрограммой и поправить еще кое-что в свойствах. Для начала входим в свойства микроконтроллера двойным кликом по нему левой кнопкой или через правую и опцию **Edit Properties** (можно, выделив его, нажать на клавиатуре Ctrl+E). Получаем окно свойств (Рис. 23). Что здесь является обязательным хотя бы на первый момент? Я уже упоминал, но еще раз повторюсь, что во всех моделях микроконтроллеров реализовано программная установка частоты. Это означает, что в свойствах конкретного микроконтроллера всегда присутствует окно либо **Processor Clock Frequency** (как на рис. 23) – для PIC-контроллеров, либо просто Clock Frequency –для всех остальных (8051, AVR, ARM). Вот здесь и задается тактовая частота – будь то в реальности кварцевый резонатор, RC-цепочка или внешний источник тактовой частоты. И для простого моделирования (без разработки печатной платы в ARES) нет смысла навешивать на схеме какие-то еще частотозадающие элементы. Достаточно прописать здесь частоту, а для микроконтроллеров, имеющих в реальности внутренние задающие генераторы в соответствующем окне свойств модели дополнительно выбрать режим. В частности в данном случае он зависит от слова конфигурации, а в микроконтроллерах AVR, входящих в библиотеку **AVR2.DLL** необходимо установить фьюзы **CLKSEL**. В любом случае никогда не лишне воспользоваться кнопкой **HELP** (рис. 23). Конечно, необходимо познание хотя бы азов английского, чтобы сориентироваться там, но при этом Вы избавите себя от многих неожиданностей и потраченного впустую времени. Например, для PIC16F84A там доступны к просмотру следующие подразделы:

Model Properties (Alphabetical Index) – свойства модели с указанием принятых по умолчанию (**Default**) и их размерностью.

Explanation of Warning Messages – разъяснение предупреждающих сообщений симулятора.

Run Time Disassembler – пояснение по использованию встроенного дизассемблера.

High Level Language Support – перечень поддерживаемых компиляторов языков высокого уровня с указанием типов подключаемых файлов, позволяющих вести пошаговую отладку.

Кроме того, из всплывающего **HELP** при наличии подключенного канала Интернет возможно скачивание англоязычных даташитов (подробных описаний на компонент) при щелчке левой кнопкой по требуемому. Впрочем, для данного МК это возможно и без вызова **HELP** при нажатии кнопки **Data** (Рис. 23), но эта кнопка не всегда присутствует в окне **Properties**.

А для моделей МК AVR, например, через кнопку HELP доступны еще и такие разделы, как **General Model Limitations** – общие ограничения к использованию и отдельным пунктом - конкретные для группы МК. Знакомство с этими разделами избавит Вас от лишних хлопот. Простой пример для ABP есть общее ограничение: **JTAG interface is not supported** – интерфейс JTAG не поддерживается. Зная это, я никогда не буду пытаться выполнить симуляцию с использованием JTAG и терроризировать форумы бесполезными вопросами по этому поводу. Там же: **Power supply voltage changing is not supported**. И не надо зря искать варианты по запитке МК AVR отличным от +5V по умолчанию напряжением.

Но вернемся к «нашим баранам». Итак, в окне установки частоты я выставил **4MHz** в соответствии с частотой используемого кварца в исходной схеме. При желании я мог бы записать это и как **4000000** без указания размерности, что соответствовало бы четырем миллионам Герц, или так: **4000k** и ISIS приняла бы любую запись. Вообще давайте здесь разберем способы задания номиналов в ISIS, тем более, что сейчас я начну активно оперировать с ними. Разработчики Протеуса значительно облегчили пользователям задачу указания номиналов, переложив разборку написанной нами галиматьи на интерфейс программы. Так, в частности если Вы указываете номинал просто числом, то он в зависимости от типа компонента и задаваемого параметра будет соответствовать основной единице измерения данной величины.

Например, если указать для терминала питания значение **+5** – это 5 Вольт, для резистора – **510** = 510 Ом, конденсатора – **1** = 1 Фарада, для времени и длительности **10** = 10 секунд, частота **1000** =1000Гц. В тоже время Протеус прекрасно поймет, если вы используете буквенное обозначение величины в конце: **V** – Вольты, **A**- Амперы, **Ohm** (или **R**) – Омы, **F** – Фарады, **H** – Генри, **S** – секунды, **Hz** – Герцы.

Теперь разберемся с производными величин:
Вниз: нано – **n**, микро – **u**, мили – **m**. Вверх: кило – **k**, мега – **M**, гига – **G**.
ISIS поймет запись в любом исполнении. Если окно предназначено для задания времени, то вместо 1 (что означает 1 секунда) я могу указать и 1000mS или просто 1000m, опустив символ наименования. К примеру, частоту (рис. 23) можно было бы записать еще и как 4M.
Для разделения в десятичных дробях используется символ точки. Пример: **4.092kHz**.

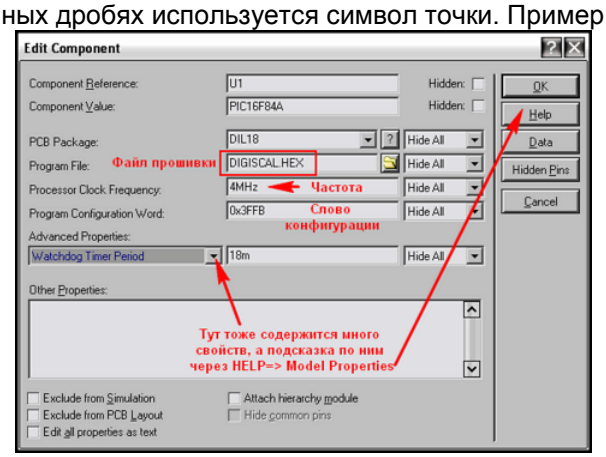


Рис. 23

2.14. «Прошивка» микроконтроллера в ISIS.

Настала пора «прошивать» микроконтроллер. Для задания в качестве прошивки для любой модели идеально подходит файл в формате **Intel Hex**. Это именно тот файл, который формируется практически всеми компиляторами и затем указывается программатору при прошивке реального контроллера. Файлы в формате **Hex** обычно прикладываются авторами разработок для самостоятельного повторения конструкции. Однако использование этого формата в ISIS исключает возможность пошаговой отладки программы, если только он не был сформирован с использованием встроенных компиляторов Протеуса, о чем чуть ниже в этом разделе. Поэтому, применяя чужие прошивки для тестирования в ISIS, не надейтесь сразу получить положительный результат в симуляторе. А как быть, если разработчик кроме hex-файла больше ничего не предоставил? Ответ прост, как яйца от МТС, - используйте программы дизассемблеры для восстановления исходного кода. Найти их бесплатные варианты в Интернет не составит большого труда через тот же **Google**.
Вот две ссылки для наиболее популярных МК:
для PIC - http://www.hagi-online.org/picmicro/picdisasm_en.html
для AVR - http://www.atmel.ru/Software/Software.htm - старенький дизассемблер **AVRDASM105**
Дизассембирование кода для большинства МК также возможно и в столь популярном дизассемблере IDA Pro: http://www.idapro.ru
Ну и, кроме того, многие компиляторы, например, столь популярный CCS PICC, содержат встроенные инструменты для дизассемблирования. Есть здесь только одно но… Если Вы воспользовались дизассемблированием hex-файла, то приготовьтесь к дальнейшей разборке полученного кода вручную. Потому что все авторские комментарии и названия переменных были утеряны при компиляции **Hex**, а после дизассемблирования они будут заменены на безликие ссылки автоматически генерируемые дизассемблером. Для сравнения приведу один и тот же кусок ассемблерного кода в авторском варианте и восстановленный дизассемблером из hex-файла:

; Проверка клавиатуры	
;=====================================	
Inkey	LADR_0x0001
clrf PortA ; RA0..RA3 = 0	CLRF PORTA ;
	Bank
	PORTA - TRISA
bsf Status,RP0	BSF STATUS,RP0 ;
	Bank Register-Bank(0/1)-Select
movlw b'00010011'	MOVLW 0x13 ; b'00010011' d'019'
movwf TrisA ; RA0,RA1,RA4 input	; Interrupt-Vector
	MOVWF PORTA ;
	Bank
	PORTA - TRISA
bcf Status,RP0 ;	BCF STATUS,RP0 ;
	Bank Register-Bank(0/1)-Select
movf PortA,w	MOVF PORTA,W ;
	Bank
	PORTA - TRISA
andlw b'00000011'	ANDLW 0x03 ; b'00000011' d'003'
return	RETURN

Как мы видим различия ассемблерного кода справа и слева налицо. Поэтому для программ полученных со стороны, всегда лучше иметь исходный код на том языке, на котором писалась программа: **Assembler, C, Basic** и т. д. . В нашем случае имеется файл **Digiscal.asm** и скоро он нам

понадобится, а пока я подключил авторский файл **DIGISCAL.HEX** (Рис. 23), чтобы как и многие до меня убедиться в том, что в Протеусе этот вариант работать как надо не будет. В дальнейшем в своих собственных проектах я рекомендую Вам следовать золотому правилу – файлы прошивки **hex**, а также исходники программы на ассемблере или языках высокого уровня и генерируемые ими промежуточные файлы складывать в ту же папку, где лежит проект для ISIS, иначе при пошаговой отладке Вы рискуете не увидеть исходного текста программы, поскольку Протеус их не будет искать по всему жесткому диску.

Еще нам надо до начала симуляции задать частоту генератора, с которого мы будем подавать частоту на вход нашего частотомера. Для этого надо войти в его свойства двойным кликом левой кнопкой мыши по нему или через правую **Edit Properties**. Я попутно переименовал его в **Clock**, чтобы помнить – какой тип выбран и задал ему для начала частоту 100 Гц (Рис. 24).

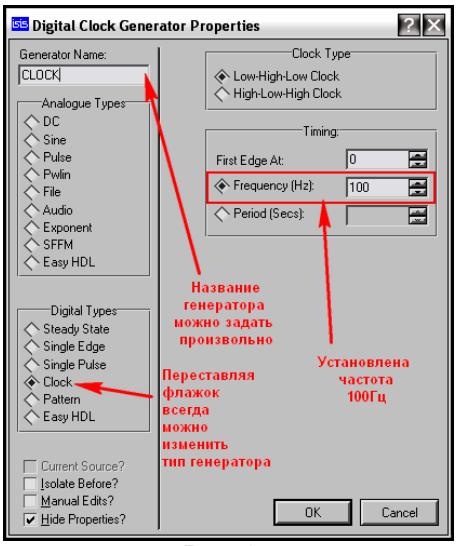


Рис. 24

Кроме того, для всех элементов кварцевого генератора в свойствах я установил галочку **Exclude From Simulation** (исключить из симуляции) по причинам о которых я уже упоминал (Рис. 25).

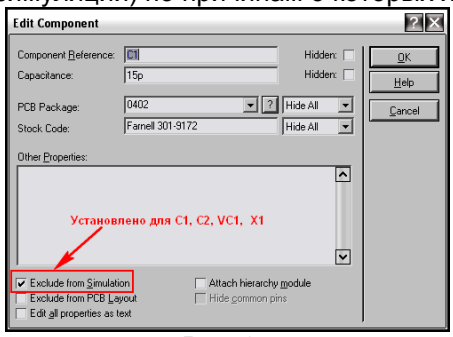


Рис. 25

Подготовленный таким образом проект во вложении. Я уже удалил из него схему, чтобы уменьшить размер, а для счастливых обладателей старых версий как всегда присутствует файл **Section**.

2.15. Первый неудачный запуск симуляции. Пляски с бубном или анализ возможных причин неработоспособности в симуляторе реально работающей схемы.

Для запуска симуляции созданного нами проекта осталось нажать кнопку **Play** внизу слева в трее окна ISIS. Результат первого запуска на рисунке 26. Я уменьшил окно программы, чтобы разместить его целиком на рисунке, поэтому расположение панелей внизу немного отличается от полноэкранного режима, где они расположены горизонтально в ряд.

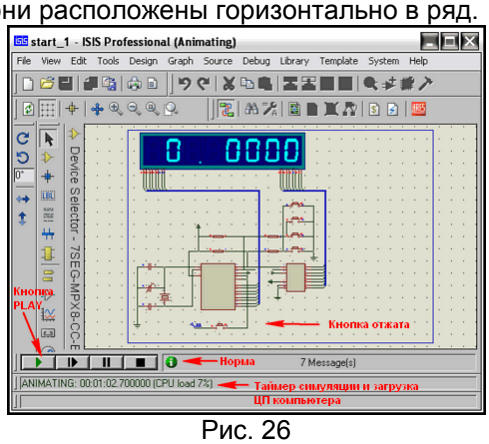


Рис. 26

О старте симуляции свидетельствует зеленая подсветка кнопки Play. Зеленый кружок с символом i свидетельствует о том, что запуск прошел без ошибок. Лог симулятора содержит 7 сообщений (Messages). При желании его можно открыть не останавливая симуляции щелкну по нему левой кнопкой мыши. Если бы были какие то отклонения при запуске симулятора, то вместо зеленого круга был бы желтый треугольник с восклицательным знаком, а при наличии критического сбоя симулятора – красный. В последнем случае окно лога открывается автоматически, т.к. симуляция не работает вообще. Но и наличие желтых «горчичников» тоже свидетельствует о том, что результат симуляции подлежит сомнению и дополнительной проверке. В таймере **ANIMATING:** происходит отсчет времени с начала запуска и показана загрузка процессора (**CPU**) компьютера. Еще раз хочу обратить внимание на этот показатель. Математика симулятора **PROSPICE**, на котором базируется **ISIS**, отнимает значительные ресурсы компьютера, особенно при расчете работы аналоговых схем, где требуется провести очень большое количество расчетов как по времени, так и по уровням сигналов. Поэтому при загрузке CPU 100% или близко к этому имитация работы схемы в реальном времени практически невозможна. Обычно Протеус предупреждает об этом наличием «горчичника» и сообщением в логе:

Simulation is not running in real time due to excessive CPU load

В этом случае необходимо либо прибегнуть к упрощению схемы за счет сокращения входящих в нее аналоговых компонентов, либо – попробовать проанализировать необходимые нам параметры с помощью графиков (Graph Mode), которые позволяют просчитать и получить результаты за счет более длительного но распределенного по времени вычисления, снимая при этом загрузку с ЦП компьютера. Этим мы чуть ниже и займемся, но совсем по другим причинам.

Итак, мы видим, что даже при отжатой кнопке на входе частотомера вместо нормального показания **00.00000** уже несоответствие показаний, а если кнопку нажать, то показания индикатора пропадают совсем и лишь изредка на нем мелькает непонятная информация. Немного отвлекусь на управление активным элементом **Button** (Кнопка) в процессе симуляции (Рис.27). Дело в том, что модели активных компонентов коммутации **Button** (Кнопка) и некоторые **Switch** (Переключатели), входящие в стандартные библиотеки Протеуса могут управляться двумя способами. В первом случае управление происходит щелчком левой кнопкой мыши по нужному элементу управления черный кружок со стрелкой, а во втором при наведении курсора на само изображение управляемой части компонента. Если у переключателей положение после воздействия фиксируется, то единственный в ISIS компонент **Button** ведет себя иначе. При щелчке по элементу управления – кнопка ведет себя как кнопка с фиксацией (выключатель), а при наведении курсора на саму кнопку – нажатие левой кнопки мышки вызывает нажатие кнопки, а отпускание мышки – возврат в исходное состояние – т.е. имеем нефиксируемую кнопку. Кнопку SB4 в нашей схеме мы будем использовать как фиксируемый выключатель входа. Еще один нюанс – все активные элементы управления в Протеусе функционируют даже при выключенной симуляции, что позволяет провести предустановку параметров схемы перед выполнением симуляции. Например, установив в проект активную модель термопары (**Thermocouple**) или One-Wire температурного датчика **DS18B20**, Вы можете до начала симуляции установить им температуру на нужное значение.

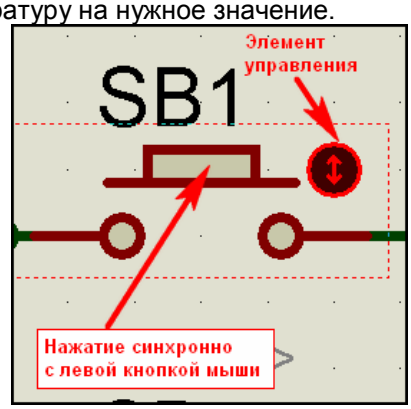


Рис. 27

На этом закончим лирическое отступление вернемся к нашей схеме и попробуем понять почему индикация погасла при подаче на вход измеряемого сигнала. Первая причина – низкая частота подаваемого сигнала – в нашем случае 100Гц. Я не буду полностью расписывать принцип работы частотомера Денисова, а только напомню, что такая структура прибора предусматривает измерение по переполнению счетчика/таймера микроконтроллера. Поскольку входную частоту мы выбрали очень низкой, переполнение происходит редко, а индикация жестко завязана с циклом измерения – отсюда и редкие мерцания. Тот же эффект будет и в реальном устройстве. Остановим симуляцию и увеличим измеряемую частоту генератора CLOCK до 10кГц, тем более, что низкая загрузка CPU (в моем случае 7%) позволяет это сделать. Снова запустим симуляцию. При подключенном генераторе картинка изменилась – вместо редко мерцающих хаотичных сегментов – появились постоянно горящие, но от этого нам не легче.

Исполним второе па «мармезонского балета» на этот раз с индикатором. Войдем в его свойства и найдем там параметр **Minimum Trigger Time**. По умолчанию там стоит значение **1ms** (милисекунда). Что он означает? Согласно Help на данный компонент – это минимальное время

присутствия сигнала на выводе индикатора при котором сегмент засвечивается. Правда в Help почему то указано по умолчанию **1us**, но оставим эту очепятку разработчикам, а сами действительно установим такое значение. Снова запустим симуляцию. Картинка опять изменилась. Теперь при подключенном генераторе горит непотребное значение, явно отличающееся от подаваемого, а при отключенном – все нули, но в обоих случаях висят лишние десятичные точки, хотя должна быть только одна. Это уже «теплее», но не соответствует реальности. Дальнейшее уменьшение **Trigger Time** желаемого эффекта уже не дает, так что пляски с бубном придется прекратить и перейти к более детальному исследованию поведения динамической индикации в симуляторе ISIS PROSPICE. Для этого нам придется прибегнуть к помощи графиков (**Graph**).

2.16. Зонды-пробники в Протеусе.

Прежде, чем мы начнем использовать графики для исследования нашей схемы, необходимо определиться: какие сигналы мы хотим поместить на график. В данном случае у нас возникла проблема с индикацией. Естественным образом напрашивается вывод: нужно рассмотреть сигналы, идущие на семисегментный индикатор. Кто-то из оппонентов тут же скажет: а на фига ему график – ведь есть же четырехканальный осциллограф. Да, есть. Но, во-первых мне необходимы 9 каналов: 8 на разряды индикатора и как минимум один на сегменты. Во-вторых, шину на сегменты я собираюсь рассматривать как единый сигнал, т. е. любое изменение данных на восьмиразрядной шине **SEG** – это смена информации. Вот и попробуйте отследить это с помощью осциллографа, а я постою в сторонке и посмотрю, что из этого выйдет. Единственный виртуальный прибор, который может дать нам реальную картинку в этом случае – логический анализатор. Но у меня со времен ЕС ЭВМ выработалась стойкая аллергия на этот агрегат. Кто копался с реальными приборами – знает, каково это – прицепиться к 8 точкам схемы с помощью анализатора, паутина получается еще та… Позже я покажу, как его использовать для данной цели, и Вы сможете сравнить – что лучше. В виртуальном мире все проще. Итак, мы будем использовать **Digital Graph** (цифровой график), а для того чтобы поместить в него сигналы – необходимо расставить зонды в нужных точках схемы.
В ISIS можно использовать два типа зондов: напряжения **V** и тока **I**. Соответствующий режим выбирается в левом вертикальном меню (Рис.27). На рисунке показан зонд, установленный на шину сегментов индикатора.

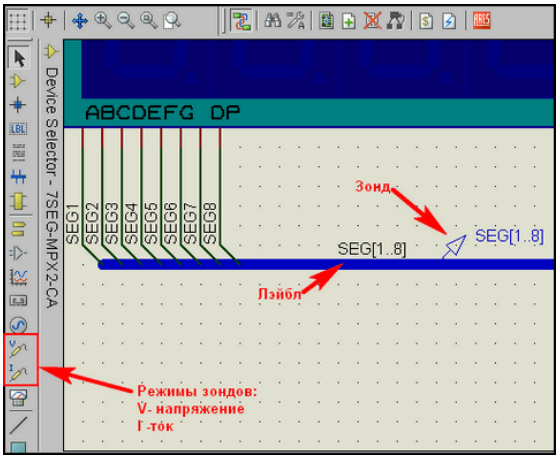


Рис. 28

Прежде чем его установить необходимо присвоить шине маркировку (лэйбл). В данном случае маркировка ставится вручную, т.е. выбираем режим **LBL**, наводим курсор в нужное место шины (при этом подсветится красный пунктир внутри ее а под карандашом курсора появиться белый X) и щелкаем левой кнопкой. Я присвоил шине имя **SEG[1..8]** (еще раз напомню- точек две!!!), потому что в данном пробнике мне необходимо контролировать все восемь сигналов сегментов. Совет на будущее – если сделать отвод от шины и ввести для лэйбла только сигналы сегментов **SEG[1..7]**, то код на графике этой шины будет совпадать с семисегментным кодом символа, записанным в программе микроконтроллера. Иногда такое полезно при отладке.
После этого переходим в режим расстановки пробников напряжения (кнопка в левом меню с логотипом щупа и буквой V) и в непосредственной близости от лэйбла ставим пробник. При этом курсор ведет себя также (карандаш с X). Пробник автоматически поймает имя ближайшей маркировки, т. е. **SEG[1..8]**.
Аналогично я установил пробник и на другую шину с именем **DIG[1..8]**. Но здесь нам понадобятся отдельные пробники по разрядам. Поскольку проект нарисован довольно компактно, если ставить их на провода, то получится «каша» из зондов и их названий, они будут наползать друг на друга. Поэтому здесь я применил другую тактику. На свободном месте разместил восемь зондов напряжения (неподключенные зонды получают имя: **?**), а затем подвел их к шине, используя функцию автоповторения провода. Если на шине уже стоит лэйбл **DIG[1..8]**, соотвествующее имя получат и все зонды. Потом, используя функцию **Property Assignmebt Tools** (**PAT**), назначаем проводам нужные лэйблы (пусть Вас не пугает, что пробники не переименовываются автоматически мгновенно – достаточно один раз толкнуть симуляцию и все встанет на свои места) . Весь процесс

продемонстрирован на анимированном Рисунке 28, и как можно видеть с использованием **PAT** занимает несколько секунд.

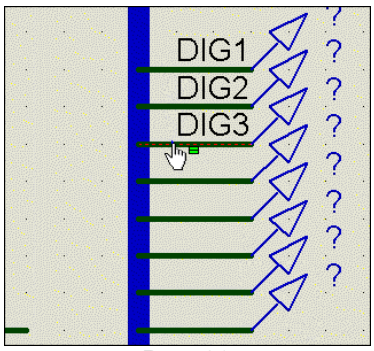


Рис. 29

Какие еще особенности установки зондов в ISIS существуют? Ну во первых при установке токовых зондов необходимо их сориентировать по направлению стрелки в круге вдоль провода, на который устанавливается пробник. Если ток в проводе совпадает по направлению со стрелкой, пробник покажет положительное значение, если нет – отрицательное. Напомню, что за положительное принято, как и обычно, направление от плюса к минусу. Важно отметить еще одно свойство, о котором многие «забывают» или просто не учитывают при установке зондов напряжения. Так как пробник является однополюсным элементом, напряжение на нем измеряется относительно заземленной шины питания. Поэтому, если вы устанавливаете зонд на провод, который имеет гальваническую развязку (например трансформаторами или конденсаторами) от земляного провода, показания будут некорректны.

2.17. Полезные свойства пробников.

На Рис. 30 показано окно свойств пробника напряжения, как наиболее «продвинутого» по наличию дополнительных возможностей. Установкой галочки **Load To Ground** (Нагрузить на землю) мы как бы иммитируем внутреннее сопротивление реального измерительного прибора. При этом в окне **Load(Ohm)**, которое становится доступным для редактирования необходимо ввести значение сопротивления. Установка галочки **Record To File** позволяет сохранить результаты измерений в файл для последующего использования, однако сразу оговорюсь, что данная функция не работает в интерактивном режиме и жестко связана с функцией **Tape** (Магнитофон) о которой пойдет речь позже.

Ну и наконец очень удобная встроенная функция аппаратного прерывания по сигналу пробника. При этом запущенная симуляция встает в состояние паузы. В чем польза данной функции? Допустим, Вам необходимо измерить время между появлениями сигнала логической единицы на каком либо проводе. Устанавливаем **Real Time Breakpoint** на **Digital**, а в окне **Trigger Value** вбиваем **1**. Запускаем симуляцию и ждем установки в паузу внизу считываем время появления сигнала. Повторно «толкаем» симуляцию кнопкой запуска (зеленый треугольник) и по второму останову считываем время появления второго прерывания. Далее используем кто свои мозги, а кто калькулятор для вычисления интервала между прерываниями. Аналогично можно поступить и с аналоговыми цепями, установив режим прерывания **Analog**, а в окне **Trigger Value** забив числовое значение напряжения. Окно **Am at Time** служит для установки времени «включения» режима прерываний. Например, если в предыдущем примере Вы поставите туда значение **2** (2 сек), то пауза возникнет при первом появлении логической единицы на пробнике после двух секунд выполнения симуляции.

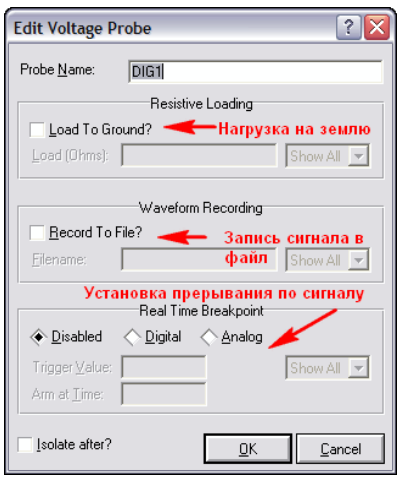


Рис. 30

И, забегая вперед, укажу еще одно полезное свойство зондов. Иногда сложно определить, каким образом симулятор **PROSPICE** интерпретирует ту или иную точку схемы (считает он там сигнал

цифровым или аналоговым). Все очень просто определить с помощью зонда. Если в этой точке просчитывается аналоговый сигнал, то зонд будет показывать реальное значение напряжения в числовой форме, а если сигнал цифровой, то одно из девяти значений, принятых для цифровой симуляции (Рис. 31). Эта таблица взята из **ProSPICE Help** версии 7.4, раздел **DIGITAL SIMULATION DARADISM => Nine State Model**. На деле несколько иначе: вместо **WUD** неопределенное состояние выхода индицируется **SUD** – имейте это ввиду.

State Type	Keyword	Description
Power High	PHI	Логическая 1 шина питания
Strong High	SHI	Логическая 1 активный выход
Weak High	WHI	Логическая 1 пассивный выход
Floating	FLT	Высокоимпедансное состояние
Undefined	WUD	Неопределенное между 1 и 0
Contention	CON	Неопред. конфликт сигналов
Weak Low	WLO	Логический 0 пассивный выход
Strong Low	SLO	Логический 0 активный выход
Power Low	PLO	Логический 0 шина питания

Рис. 31

2.18. Digital Graph – применяем на практике.

Зонды расставлены, и если мы запустим симуляцию в реальном времени, то можно увидеть мелькание логических значений из таблицы выше на единичных зондах и изменение значений в шестнадцатиричном формате на пробнике, установленном на шину. Пора задействовать график. Так как у нас только цифровые сигналы – логично применить **Digital Grath** (Цифровой график) для анализа наших сигналов. Его особенность в том, что он позволяет разместить отдельные сигналы на разнесенных по вертикали временных осях, что удобно для их просмотра и анализа. Итак, в левом меню задействуем кнопку **Graph Mode**, выбираем в селекторе **Digital** и на свободном месте листа проекта, удерживая нажатой левую кнопку мыши, растягиваем по диагонали наш будущий график (Рис. 32). Совсем необязательно тянуть как показано стрелкой, можно и снизу вверх и справа-налево, главное по диагонали. Впоследствии можно будет подкорректировать размеры окна графика, щелкнув по нему один раз мышкой и растянув или вертикально, или горизонтально, ну или за угол, зацепив мышью за появляющиеся при этом маркерные черные точки так, как мы делаем в любом графическом редакторе, или в офисных приложениях, например в Word с картинками, или прямоугольными объектами. График размещен и пора добавить на него наши исследуемые сигналы. В протеусе для этого предусмотрены два способа.

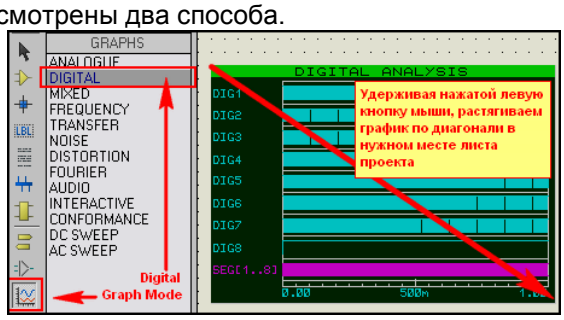


Рис.32

Для особо ленивых: щелкаем один раз по нужному зонду левой кнопкой мыши, чтобы он выделился (стал красного цвета). Затем зажимаем его левой кнопкой и тянем по экрану на черное поле графика, где бросаем кнопку. В результате слева от вертикальной оси координат появится название нашего пробника, а напротив него горизонтальная временная ось, показывающая абсолютный уровень нуля для этого сигнала.

Более продвинутый способ: щелкаем правой кнопкой мыши по черному полю графика и во всплывающем меню выбираем опцию **Add Traces…** В открывшемся окне через раскрывающееся меню выбираем нужный нам сигнал по имени зонда, например **Dig1** (Рис. 33). Жмем **OK** и получаем тот же результат.

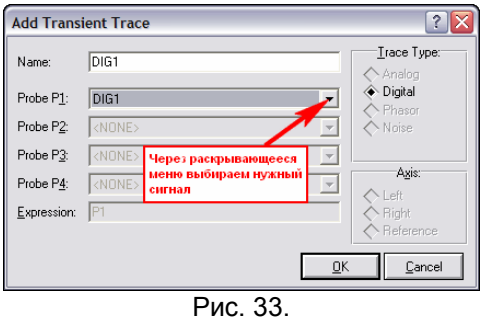


Рис. 33.

Чем хорош этот вариант. Сейчас у нас задействован самы простой тип графика, поэтому Trace Type (Тип трассы) и привязку к вертикальным осям – **Axis** изменить невозможно. Когда мы будем использовать другие типы графиков, эти опции станут активными.
Использовав любой из способов, я предпочитаю второй, помещаем все нужные нам сигналы на график в требуемом порядке. На рисунке 32 видно, что я разместил сначала сигналы разрядов в порядке справа-налево, а затем общий сигнал шины (он фиолетового цвета).
Чтобы запустить график на выполнение можно воспользоваться либо всплывающим по правому щелчку меню, как и для добавления сигналов, но выбрать опцию **Simulate Graph…** , либо просто «топнуть» по клавише пробела на клавиатуре. Сразу обращаю внимание, что если Вы разместите в проекте несколько графиков, то чтобы симулировать нужный – сначала надо его выделить одиночным щелчком левой кнопкой мыши по нему. Результат симуляции нашего графика виден все на том же рисунке 32, но конечно же он нас не устраивает и мы приступаем к изменению параметров графика, чтобы получить приемлемую картинку.

2.19. Свойства цифрового графика.

Для того, чтобы попасть в окно свойств (**Properties**) графика используем способ аналогичный работе с любыми объектами в ISIS – двойной щелчок левой по объекту или через контекстное меню правой –**Edit Properties** (**CTRL+E** с клавиатуры). Окно свойств показано на Рис. 34.

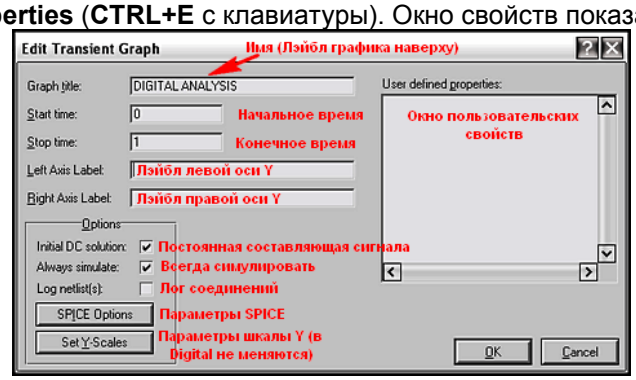


Рис. 34.

Большинство опций понятно из приведенных на рисунке комментариев. Цифровой график наиболее простой с точки зрения наличия свойств. Мы можем поменять ему название, присвоенное по умолчанию, например обозвать его вместо DIGITAL ANALYSIS - DINAMIC INDICATION. Поскольку на вертикальной оси мы располагаем множество цифровых сигналов, параметры ее в данном режиме и лэйблы не работают (их мы рассмотрим на примере других графиков). Флажок **Inicial DC Solution** (эквивалентный переключателю закрытый/открытый вход осциллографа) в данном случае также не активен. Отдельно остановлюсь на флажке **Always simulate**. Когда в проекте размещено несколько графиков одного и того же процесса в разных режимах, удаление этого флажка позволяет сохранить график в «неприкосновенности» при манипуляциях с другими. Кнопка SPICE Option обеспечивает быстрый доступ к параметрам симулятора, но менять что-то там без особой надобности начинающим не рекомендуется.
На данном этапе нас интересуют два конкретных окна – **Start Time** (начальное время графика) и **Stop Time** (конечное время графика) по шкале **X**. По умолчанию это **0** и **1** (напомню о том, что писалось выше – временной параметр без обозначения в секундах). При тактовой частоте микроконтроллера 4 МГц это явный перебор, поэтому мы и видим сигналы сжатой широкой лентой. Уменьшим параметр **Stop Time** раз в сто для начала – поставим **10m** (10милисек) и нажмем **OK**. На предложение Протеуса **Resimulate Graph?** ответим утвердительно. Результат на Рис. 35.
Будьте внимательны. Иногда Протеус «забывает» предложить повтор симуляции, и сам просто меняет масштаб оси X, сдвигая картинку. Поэтому лучше повторно «толкнуть» симуляцию самостоятельно, если предложения не последовало. Иначе достоверность картинки остается сомнительной.

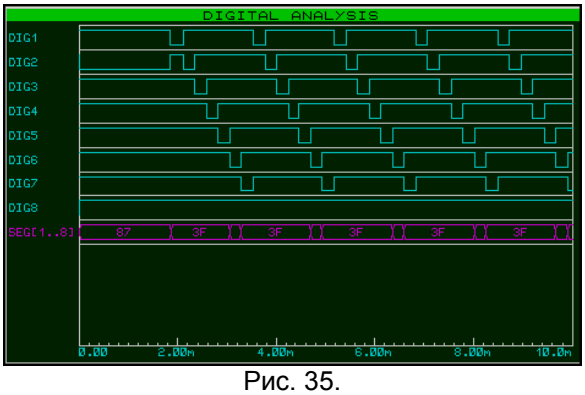


Рис. 35.

Ну вот, картинка при таком масштабе картинка приобрела более приемлемый вид, по которому уже можно анализировать сигналы. Можно и дальше растягивать график указанным способом. Например, вначале от 0 до почти 2 мсек присутствует «мертвая зона» - инициализация

микроконтроллера ее можно отсечь, установив **Start Time** **1.5m**. Для подробного анализа нам достаточно одного полного периода индикации (цикла перебора всех знакомест **DID1…DIG7**). Отмечу, что **DIG8** постоянно «висит» в единице, потому что мы находимся в режиме измерения, а этот разряд активизируется при установке параметров цифровой шкалы. Поэтому конечное время - **Stop Time** можно принять в районе **4m**. После таких трансформаций на графике останется только один полный период динамической индикации. С ним мы и продолжим наши изыскания.

2.20. Дополнительные возможности анализа графика при максимизации окна.

Дальнейший анализ графика лучше всего проводить в режиме максимизации окна. Для этого через меню правой кнопки при щелчке по графику выбираем опцию **Maximize (Show Window)**. Развернувшееся при этом окно показано на Рис. 36. Оно уже обладает самостоятельными меню и опциями, которые мы сейчас и рассмотрим подробнее.

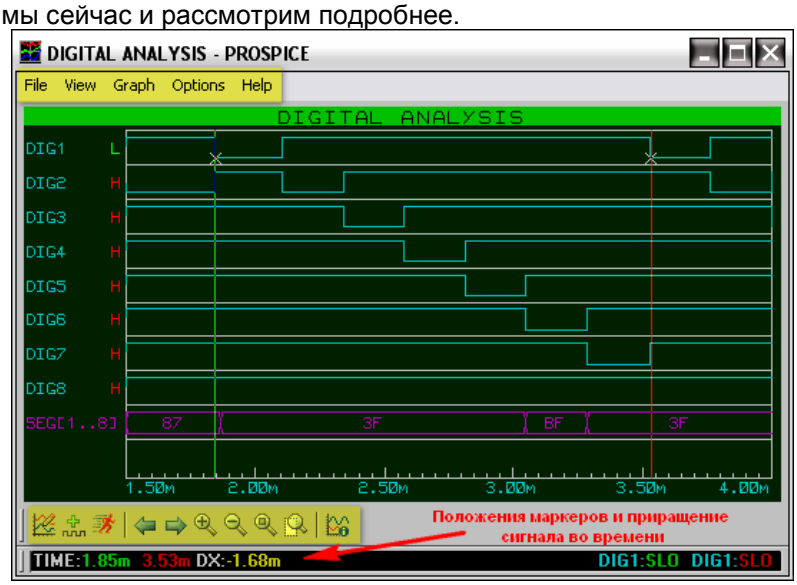


Рис. 36.

Начнем с того, что в этом режиме доступны два вертикальных маркера (на рисунке зеленая и красная вертикальные линии). Первый (зеленый) маркер устанавливается и сдвигается нажатой левой кнопкой мыши. Его положение по оси X отмечено зелеными цифрами в нижней строке статуса (на рисунке **1.85m**). Если при установке маркера навести курсор на конкретную трассу (в моем примере **DIG1**), то его положение на трассе отмечается дополнительно крестиком, а в нижней строке статуса указывается значение сигнала в данном месте зеленым цветом (**DIG1:SLO**) – низкий уровень. Все вышесказанное относится и ко второму (красному) маркеру, но для его установки надо удерживать нажатой клавишу **CTRL** на клавиатуре. Соответственно все значения, относящиеся к нему в строке статуса красные: **3.53m** и **DIG1:SLO**. Значение **DX:-1.68m** – это разность положений маркеров по оси X (зеленый минус красный - поэтому в данном случае оно отрицательное).

Что еще полезного мы можем извлечь из нашего графика? Обратите внимание на столбец букв напротив трасс, расположенный у левой вертикальной оси он показывает значения сигналов для зеленого маркера. В данном случае для **DIG1** – низкий **L**, для остальных разрядов высокий – **H**. Посмотрите на значения сигнала для шины (фиолетовая трасса). Для первых пяти разрядов он соответствует шестнадцатеричной $**3F** – это ноль на семисегментном индикаторе. Для **DIG6=SLO** (район 3.1m) этот сигнал – $**BF**, т.е. добавлена десятичная точка семисегментного индикатора.

Для особо непонятливых на Рис. 37 приведено окно от встроенного в компиляторы от **MikroElectronika** (в данном случае **microC**) декодера для семисегментных дисплеев с изображением нуля. Напомню, что у нас индикатор с общим катодом.

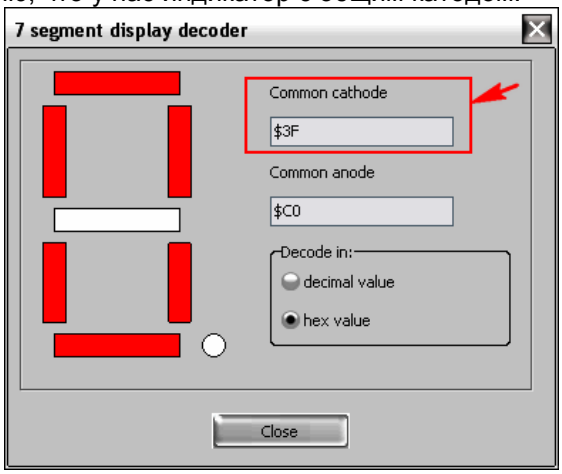


Рис. 37.

Ну что же, судя по нашему графику видимых причин для глючности индикации нет, и должно бы все работать, но… не работает.

2.21. Сравнение с работающим проектом динамической индикации. Находим причину глюка.

Отложим пока рассмотрение дальнейшее опций графика и обратимся к примеру, содержащему динамическую индикацию, прилагаемому с Протеусом - **SAMPLES\VSM for AVR\AVR Tiny15 Demo\t15demo.DSN**. Я слегка «модернизировал» его – убрал комментарии разработчика, собрал сигналы сегментов в шину и поместил в проект цифровой график с сигналами разрядов и сегментов четырехразрядного индикатора, примененного в нем.

Небольшой комментарий для тех, кто попытается самостоятельно разместить в этом проекте график. Если при попытке запустить график на исполнение ISIS ругнется на отсутствие лицензии на модель индикатора **7SEG-MPX4-CA**, просто обновите эту модель. Для этого зайдите в библиотеку, найдите эту же модель и дважды щелкните по ней, чтобы она поместилась в селектор объектов. На вопрос Протеуса произвести **Update** - ответьте утвердительно. Связано это с тем, что по каким-то причинам в этом примере затесалась модель индикатора с ограничением на использование графиков. Для тех, кому некогда этим заниматься я во вложение поместил уже подправленный проект, в котором попутно заменил модель микроконтроллера **Tiny15** на модель из библиотеки **AVR2.DLL**. На одном графике размещен полный период индикации четырех разрядов, а на другом (Рис. 38) растянут во времени момент смены первого и второго разрядов. Отметьте, что в данном случае применен индикатор с общим анодом, поэтому сигналы противоположны по уровням по отношению к анализируемому нами частотомеру.

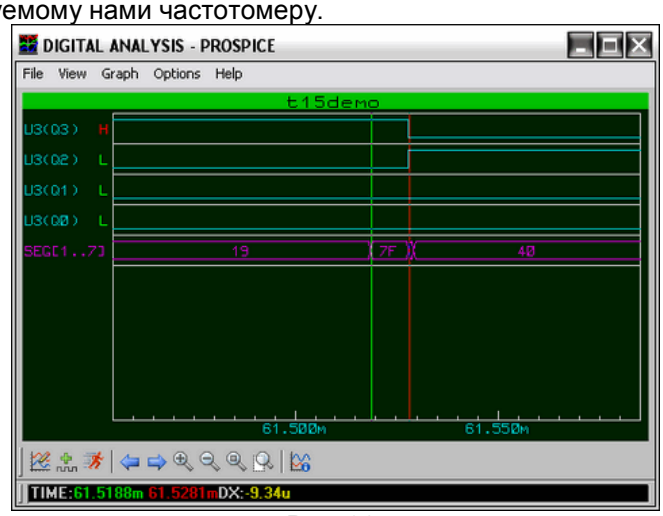


Рис. 38.

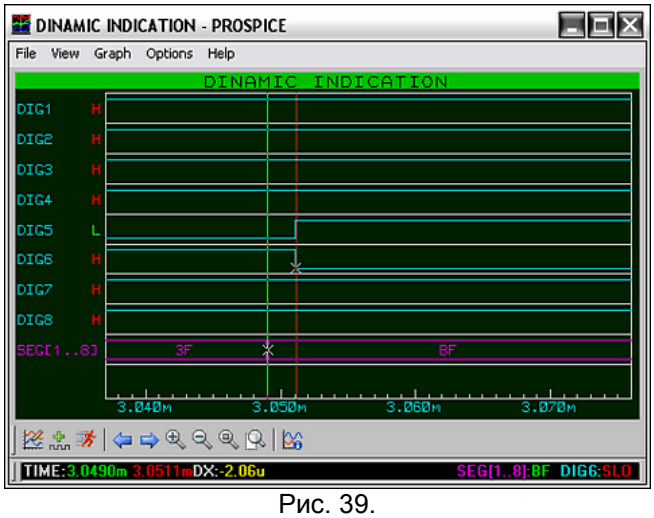


Рис. 39.

Для сравнения на Рис.39 помещен аналогично растянутый во времени момент смены пятого и шестого разрядов индикатора частотомера. Я нарочно выбрал этот момент, потому что в данном случае происходит добавление десятичной запятой (сигнал $**BF** на шине). Обратите внимание на график работающей индикации (Рис. 38). На момент смены горящих разрядов (в данном случае логических единиц) на шине сегментов на время около 9.4 микросек появляется сигнал $**7F** (я выделил его двумя маркерами), т.е. тоже фактически все единицы. Происходит кратковременное гашение разряда индикатора – и с анода единица и с сегментов все единицы. Теперь рассмотрим наш «тяжелый случай» (Рис. 39). Мало того, что никакого гашения и в помине нет, так еще и смена сигнала на шине сегментов (зеленый маркер) происходит на 2 микросек раньше, чем смена нулей на катодах.

Вот и обнаружился глюк индикации. И виноват в этом вовсе не ISIS, честно выполняющий то, что в него заложено создателями, а **…** ну в общем «стрелочник». Фактически же это выглядит так, как

показано на Рис. 40. При отсутствии входного сигнала должны индицироваться все нули и десятичная точка в шестом разряде, а у нас она присутствует еще и в пятом.
Чуть позже мы займемся «хирургическим вмешательством» в программу для устранения данного явления, а пока необходимо закончить рассмотрение опций графика, выделенных на Рис. 36 желтой подсветкой, иначе многим непонятно будет как я растянул графики на рисунках 38 и 39.

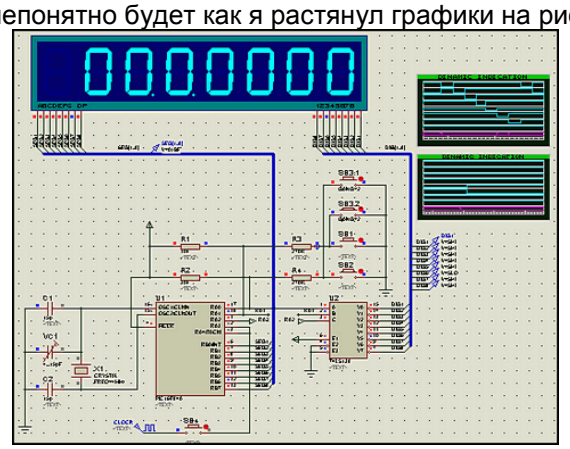


Рис. 40.

2.22. Меню и опции графиков в развернутом (Maximize) окне.

Для начала рассмотрим верхнее меню.
File – из этой вкладки график можно распечатать на принтере в цветном или черно-белом формате (**Print**) или экспортировать в один из предлагаемых форматов (Export Graphics…). При печати и сохранении черно-белого варианта фон белый, а трассы и координатные линии черные. Из графических форматов доступен bitmap (**BMP**). Можно также сохранить и в формате AutoCAD (**DXF**). В опциях выбирается разрешение картинки – чем выше разрешение, тем больше объем файла. Обращаю Ваше внимание, что маркеры при этом не сохраняются и не печатаются.
Опции вкладок **View** и **Graph** фактически повторяют кнопки содержащиеся в нижнем тулбаре графика, поэтому мы их рассмотрим подробно здесь же чуть ниже.
Вкладка **Options** позволяет обеспечивает прямой доступ к настройкам ISIS, расположенным в меню **Template** и **System** из основного верхнего меню ISIS. Наиболее интересна для нас на данном этапе опция **Set Graph Colours…** (Рис. 41), позволяющая изменять цветовую схему графиков.

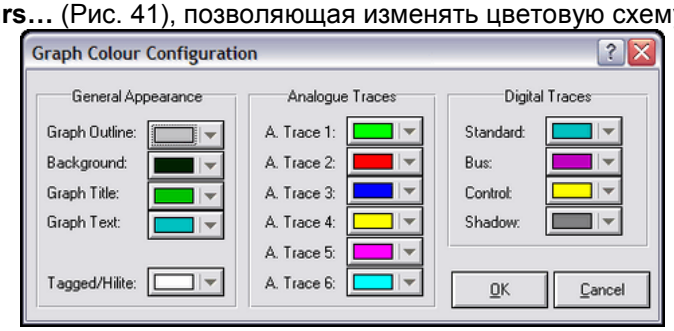


Рис. 41.

В левой колонке выбираются основные цвета оформления для всех графиков: Outline – линии сетки и координат, Background – фон, Title – заголовки, Text – текст, Tagged – контрольные точки. В средней колонке устанавливаются цвета для шести трасс аналоговых графиков, в правой для цифровых, т.е. для нашего случая. Мы и можем наблюдать на рис. 38,39 бирюзовые трассы сигналов и фиолетовую для шины, используемые по умолчанию. Напомню, что изменять цвета графиков можно и из основного меню ISIS **Template** -> **Set Graph Colours…** .
Вкладка **Help** обеспечивает доступ к файлу помощи ISIS.
Теперь рассмотрим кнопки расположенные внизу. Слева-направо в соответствии с всплывающими подсказками.
Edit Graph, **Add Traces** и **Simulate Graph** нам уже знакомы и просто обеспечивают доступ к данным функциям без свертывания окна графика.
Две кнопки со стрелками влево и вправо **Pan Graph…** позволяют горизонтальную прокрутку в указанных направлениях.
Кнопки **Zoom In** и **Zoom Out** (лупы с плюсом и минусом) позволяют увеличить/уменьшить горизонтальный масштаб. А вот следующие две более продвинутые.
Zoom To View Entire Sheet – сжимает график в тот временной масштаб, который задан первоначально позициями **Start Time** и **Stop Time**.
Zoom To Area – включает курсор в режим выделения площади (белый квадрат с прицелом), после чего можно, удерживая нажатой левую кнопку мыши, выделить участок для увеличения. Для Digital Graph масштабирование происходит только по оси X. Именно этим режимом я воспользовался несколько раз, чтобы растянуть нужные участки графика так, как показано на рис. 38,39.
И еще одно важное замечание. После того, как вы масштабировали график и свернули его – щелчком по кнопке **X** в правом верхнем углу, или через контекстное меню правой кнопки мыши –

опция **Restore (Close Window)** – в окне проекта оно будет показываться уже в новом, измененном масштабе.Вот и все, что хотелось бы отметить по работе с графиками в развернутом виде.

2.23. Подключаем файл микропрограммы для пошаговой отладки.

До сих пор мы использовали для симуляции прошивки микроконтроллера файл формата **Intel Hex**. Этобыло вполне приемлемо, пока нам не требовалось вносить изменений в программу. Но мы уже нашли причину нашей неудачи и пора попробовать избавиться от нее. Вот и настал черед подключения ассемблерного файла **DigiScal.asm**. Прежде, чем мы начнем заниматься «пластической хирургией» - сделайте резервную копию всей папки с проектом. В будущем этот шаг избавит Вас от поисков оригинала проекта, который к концу «операции» может быть изуродован до неузнаваемости. Итак мы вновь открыли наш проект и для подключения файла, написанного на ассемблере или «асме» на сленге эмбедеров нам необходимо войти в меню **Source** и выбрать опцию **Add/Remove Source Files…** . Откроется окно показанное на Рис. 42.

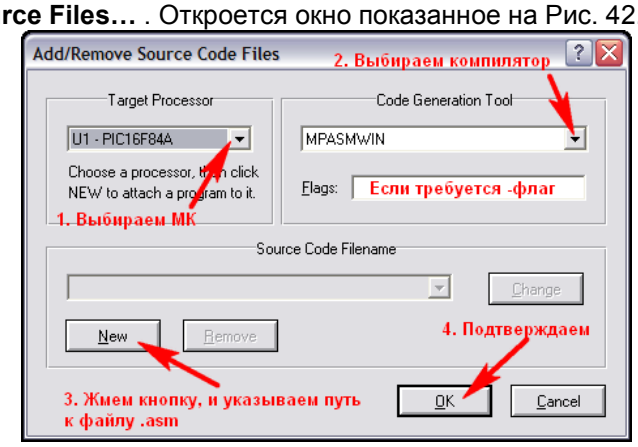


Рис. 42.

Последовательность присоединения файла программы к проекту и ясна из коментариев на рисунке. Остановлюсь на некоторых особенностях. Если в проекте используется более одного микроконтроллера на первом шаге надо выбрать нужный для конкретного файла.

На втором шаге выбирается компилятор, подходящий для данного микроконтроллера ну и конечно ассемблерного кода. В Протеусе 7.4 доступны следующие прилагаемые с программой компиляторы:

ASEM51 для МК серии MCS-51;
ASM11 для МК Motorola MC68HC11;
AVRASM2 для МК Atmel AVR;
MPASM и **MPASMWIN** для МК Microchip PIC.

Все они расположены в папке **Tools** установленного Протеуса и список их по мере выхода новых версий пополняется. Почему я выбрал именно виндозную версию MPASMWIN? Просто она мне кажется более дружелюбной в отношении интерфейса. Кроме того, некоторые компиляторы командной строки (в частности MPASM) требуют соблюдения DOSовского формата имени файла (семь плюс один символов имя и три расширение) и если имя файла будет, например, **DigiScal12.asm** выплюнут ошибку компиляции.

Хочу здесь остановится еще на одном нюансе. Не всегда Протеус содержит «свежие» версии компиляторов. Здесь можно прибегнуть к следующей операции, например для того же MPASM. Если у Вас установлена одна из последних версий MPLAB IDE, откройте через меню **Source** опцию **Define Code Generation Tools…** (Рис. 43). Укажите путь к новому компилятору через кнопку **Browse**. При необходимости измените опции командной строки (**Comand Line**) и путь **Debug Data Extraction**.

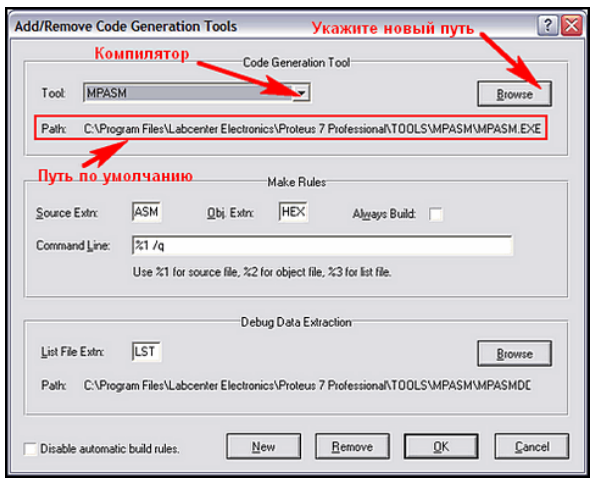


Рис. 43.

На третьем шаге нам необходимо указать наш файл **DigiScal.asm**. После чего жмем **OK**. Напомню, что идеально, когда файл с листингом ассемблера лежит в той же папке, что и проект ISIS.

Дальше можно было бы просто запустить симуляцию проекта и Протеус сам автоматически скомпилирует новый **HEX** перед началом симуляции. Но я все же рекомендую предварительно воспользоваться опцией **Build All…** из меню **Source**. При этом симуляция не запускается, а включается в работу только компилятор. Если все пройдет нормально, то откроется окно **BUILD.LOG**, содержащее отчет об успешной компиляции файла. Если в нем содержатся красные сообщения об ошибках, значит компиляция завершилась неудачно. Необходимо найти и исправить ошибки в листинге программы. Обычно содержится перечень строк, содержащих ошибки. Для поклонников PIC остановлюсь еще на одной особенности. Иногда попытка скомпилировать чужой листинг завершается неудачно из-за отсутствия в начале программы всего одной строки: **LIST p=<*тип МК*>**. Применительно к нашему случаю она должна выглядеть как: **LIST p=16F84**.
После удачного завершения компиляции в папке с проектом должен появится файл **DigiScal.SDI**, формируемый Протеусом на этапе компиляции. Именно это файл и позволяет пошаговую отладку ассемблерной программы.

2.24. Режим пошаговой отладки программы в ISIS.

После того как мы подключили ассемблерный файл к проекту, мы получили возможность пошаговой отладки со всеми вытекающими из этого возможностями. По счастью автор частотомера приложил ассемблерный листинг удобоваримый для компилятора и после команды **Build All…** мы получили подтверждение об удачной компиляции ассемблерого файла (Рис. 44).

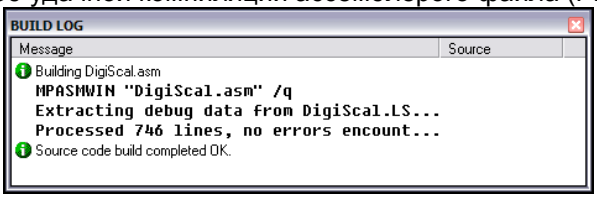


Рис. 44.

Чтобы войти в режим пошаговой отладки мы для запуска симуляции воспользуемся кнопкой **Pause** или кнопкой **Step** вместо обычной **Play**. При этом включится подсветка обеих кнопок: **Play** и **Pause**, а на экране появится окно **CPU Source Code** в котором представлен ассемблерный листинг программы, а курсор выполнения (красный треугольник слева) установлен на первой строке, расположенной по адресу $0000 в ПЗУ программ, соответственно подсвечивается и вся строка (Рис. 45). Колонка адресов ПЗУ программы микроконтроллера в данном случае бирюзового цвета слева в окне. Обратите внимание, что зеленые коментарии, находящиеся в листинге тоже отображены, но адресов не имеют – стоят прочерки, поскольку при компиляции в исполняемый код они выбрасываются. Однако, при переходах по номеру строки (**Goto Line**) они учитываются, так как занимают строку в листинге программы. Здесь надо четко представлять различия – треугольник это указатель выполняемой строки, а подсветка – это «курсор» выбранной строки.

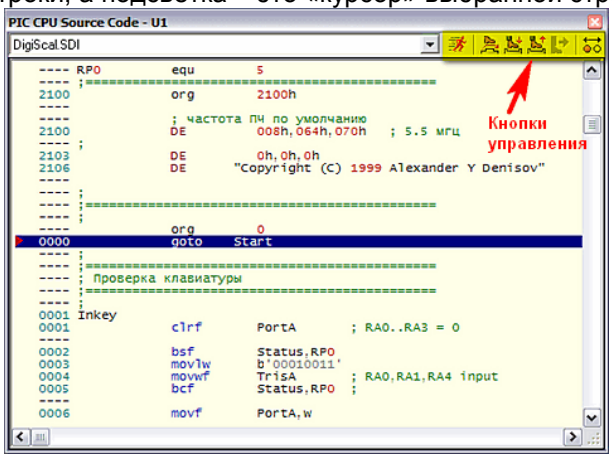


Рис. 45.

В правой верхней части окна расположены кнопки управления пошаговым режимом (на рисунке я подсветил их желтым цветом). Вот здесь и скрываются огромные возможности симулятора по сравнению с реальным устройством. Используя режим пошаговой отладки мы можем в любой момент исследовать что творится не только на внешних выводах микроконтроллера, но и в его «внутренностях». Давайте для начала познакомимся с назначением кнопок, а потом попробуем применить их в действии. Как обычно слева-направо, начиная с кнопки с изображением бегущей фигуры (**Run simulation**). Эта кнопка просто запускает продолжение симуляции, снимая ее с паузы, ну или с точки останова (**Breakpoint**).
Следующие три кнопки: **Step Over Source Line**, **Step Into Source Line**, **Step Out from Source Line** позволяют выполнить код пошагово соответственно до конца текущей строки, в текущей строке, с выходом из текущей строки.
Неактивная кнопка **Run To Source Line** (Выполнить код до текущей строки) станет доступной, если вы выделите щелкнув левой кнопкой мыши какую либо строку кода программы. При этом курсор текущей выполняемой строки остается на месте, а подсветка переместится на выбранную строку.

Запуск кнопкой **Run To Source Line** вызовет выполнение программы до достижения выделенной строки.
Ну и наконец **Toggle Breakpoint** – переключатель точек останова. Каждый щелчок по этой кнопке переключает точку останова в выбранной (подсвечиваемой) строке следующей последовательности:
«установить и активизировать» => «деактивировать» => «удалить».
Активная точка останова выглядит как закрашенный красный круг, неактивная - как красная окружность с белой серединой. Точки останова позволяют нам прерывать выполнение программы в строго указанных местах.

2.25. Контекстное меню окна пошаговой отладки.

Кроме верхних кнопок управления отладкой многие опции спрятаны в контекстное меню, вызываемое по правой кнопке мыши щелчком внутри окна **CPU Source Code** в режиме паузы. Вид меню представлен на Рис.46.

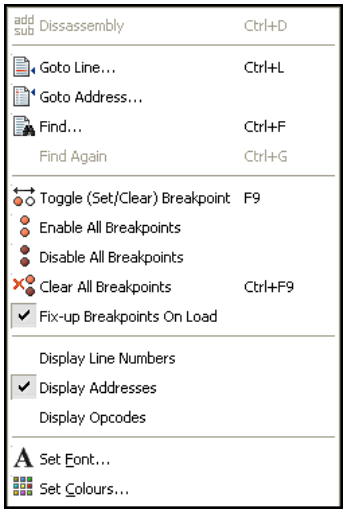


Рис. 46.

Окно поделено на пять секций. Верхняя опция **Dissasembly** в данном случае неактивна. Да и куда уж дальше разворачивать код, если мы и так в низкоуровневом ассемблере. Но когда позже мы будем подставлять в качестве исходного кода листинги на языках высокого уровня **Си**, а кто нибудь и **Basic** – эта функция нам здорово поможет, т.к. позволяет развернуть строку Си в коды ассемблера. Замечу, что именно таким образом и был обнаружен глюк библиотеки **AVR2.DLL** при выполнении переходов **RJMP** и **RCALL**. Сначала была найдена функция на Си, вызывающая ошибку, затем развернута в ассемблер и найдена конкретная команда – **RJMP**.
Следующие две команды **Goto Line…** и **Goto Addresss…** переведут текущее положение выбранной строки (подсветку) по указанному номеру (адресу в ПЗУ программ МК). Различия их в том, что в первом случае вы вводите десятичный номер строки, а во втором шестнадцатиричный адрес ПЗУ программ, например так: **0x007E**. Конечно, имея в голове «калькулятор» для перевода десятичных значений в шестнадцатиричные, можно и адрес вводить в десятичном виде, но Вам оно надо?
Опции **Find…** (Поиск) и **Find Again…** (Повторный поиск – становится активной после выполнения первого поиска) ничем не отличаются от аналогичных в других программах и ищут слово (фразу) и т.д. в соответствии с заданными условиями: регистр, слово целиком, вперед, назад…
Следующая группа команд управляет брекпойнтами (точками останова). Первая из них **Toogle…** – фактически повторяет кнопку в верхенм меню. Следующие три: **Enable** (активировать), **Disable** (дезактивировать) и **Clear** (очистить) выполняют данные действия сразу для всех установленных брекпойнтов. Установка флажка **Fix-Up Breakpoins On Load** – активирует все установленные точки при перезапуске симуляции.
В следующей группе установкой флажков активируется показ соответствующих колонок в окне **CPU Source Code**. **Display Line Numbers** – покажет номера строк, включая и комментарии. **Display Addresses** – уже установлен, показывает адреса по которым в ПЗУ программ МК расположены команды ассемблера. Установив флажок **Display Opcodes**, Вы увидите шестнадцатиричные коды записываемые в ПЗУ напротив соответствующих адресов.
Ну и последняя группа **Set Font…** и **Set Colours…** - позволяет настроить вид окна **CPU Source Code**: шрифт текста и цветовую гамму в соответствии с вашими вкусами и привычками.

2.26. Меню Debug в развернутом виде.

При первом знакомстве с интерфейсом ISIS я умышленно пропустил эту вкладку верхнего меню. Но теперь, когда мы вплотную занялись симуляцией проекта, настала пора познакомится с ним поближе. Тем более, что в режиме паузы при симуляции схем, содержащих микроконтроллеры там вылезают такие опции, которых в других случаях Вы не увидите. Итак, поставим наш проект в режим паузы и откроем эту вкладку (Рис. 47). Да, тут есть где порезвиться. Попробуйте открыть эту вкладку не запуская симуляции и, как гласит избитая телереклама, - «почувствуйте разницу». Давайте «осваивать» именно развернутую при симуляции версию. Здесь тоже схожие команды

сгруппированы по секциям. Верхняя секция дублирует кнопки управления симуляцией и в коментариях не нуждается. Первая опция второй секции – **Execute** нам также знакома по аналогичной кнопке окна **CPU Source Code**. А вот следующие две представлены только здесь. **Execute Without Breakpoints** (Alt+F12) – вызывает запуск симуляции с игнорированием установленных брекпойнтов, а **Execute For Specified Time** – вызывает окно в котором мы предварительно задаем время для перевода симуляции в режим паузы. По умолчанию предлагается 1, как вы уже догадались одна секунда. Тоже весьма полезная «фишка».

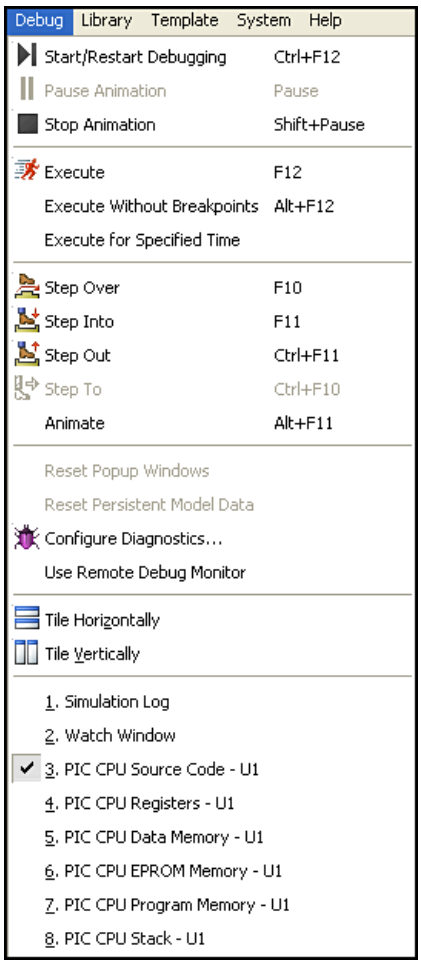


Рис. 47.

В следующей группе тоже знакомые нам по окну **CPU Source Code** команды. Хм, ошибочка вышла – прилепил рисуночек от версии 7.5 а в ней новая опция **Animate**, которой раньше не было. Сейчас мы ее… Ну в общем тоже полезная штука. Инициирует симуляцию в «заторможенном» режиме, при этом в окне **CPU Source Code** подсветка строки пошагово показывает процесс выполнения. Кто пользовался симуляцией в MPLAB, AVRStudio или других отладчиках уже имели удовольствие наблюдать такой режим.

Следующая группа представляет опции, которые представляют значительный интерес. **Reset Popup Windows** и **Reset Persistent Model Data** на картинке неактивны. Они позволяют сбросить в исходное состояние соответственно всплывающие окна и данные в ПЗУ (не путайте с памятью программ) моделей микроконтроллеров, флэш-памяти и других устройств, содержащих ПЗУ. Это возможно только когда симуляция не запущена. Состояние всплывающих окон по умолчанию представлено в последней группе. Галочкой отмечено только окно **PIC CPU Source Code** – которое мы и видим в режиме паузы. Если вы случайно или специально закроете это окно (щелчок по **X** в правом верхнем углу), то при следующей паузе симуляции оно не «всплывет». Добыть его обратно можно двумя способами: через опцию **Reset Popup Windows**, но при этом и все остальные окна встанут в исходное, или в режиме паузы восстановить галочку щелчком левой кнопкой мыши по **PIC CPU Source Code** в последней группе (Рис. 47). После закрытия окна **PIC CPU Source Code** в режиме симуляции галочка будет отсутствовать. Второй способ хорош тем, что вы не сбрасываете режимы остальных окон.

2.27. Конфигурация диагностических сообщений во вкладке Debug. Возможности окна SIMULATION LOG.

Опция **Configure Diagnostic** доступна как в режиме останова, так и при запущенной симуляции. Эта команда вызывает окно конфигурирования диагностических ссобщений ISIS (Рис. 48) и требует более детального знакомства. Здесь можно определить, каким образом Протеус будет формировать диагностические сообщения в окне **SIMULATION LOG**.

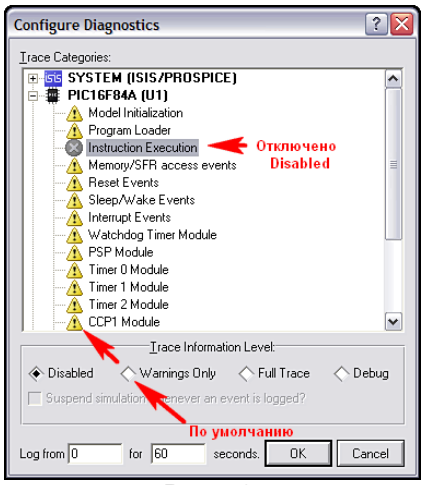


Рис. 48.

По умолчанию вся диагностика стоит в режиме **Warnings Only** (только предупреждения), что индицируется желтыми треугольниками с восклицательными знаками («горчичниками» в футбольных терминах). Для того, чтобы увидеть конкретный перечень формируемых диагностических сообщенийнеобходимо щелкнуть по плюсику слева от нужного компонента. На рисунке у меня свернуты сообщения самого симулятора PROSPICE и развернуты для микроконтроллера. По умолчанию лог ведется в течение одной минуты (время внизу от 0 до 60 сек), но при желании его можно изменить. Кроме того, для отдельных видов сообщений можно изменить способ оповещения. На приведенном рисунке отключена (**Disabled**) диагностика выполняемых инструкций для микроконтроллера. Обращаю внимание поклонников компилятора **CCS PICC** на такой режим. Дело в том, что для компиляции программ, написанных для МК серии **PIC16**, в данный компилятор использует **PCM 14 bit**, в котором изначально заложена устаревшая инструкция ассемблера **TRIS** (не путайте с Си-шной). «Умный» Протеус обрабатывает эту инструкцию, но в логе выдает «горчичник», например, для PIC16F84A: **TRISB Instruction is deprecated for PIC16F84** (Инструкция TRISB не рекомендована для PIC16F84). Причем таких сообщений в течение минуты может вылететь очень много. Чтобы они не раздражали глаз, ведь симуляция протекает нормально можно перевести опцию диагностики **Instruction Execution** (исполняемые инструкции) в положение **Disabled**, как показано на рисунке. Иногда наоборот полезно перевести какой либо вид сообщений в режим **Full Trace** (полная трассировка) или **Debug** (отладка). Например, если перевести **Instruction Execution** для микроконтроллера в один из этих режимов применительно к нашему проекту окно **SIMULATION LOG** после запуска симуляции будет выглядеть как на Рис. 49. Обратите внимание, что для МК ведется полный лог выполнения команд ассемблера, только ассемблерные метки заменены на абсолютные адреса в памяти программ (Например, вместо **Goto Start** в логе **GOTO 0x007E**). А как вам нравится колонка **Time**? Напротив каждой команды – время ее выполнения. Посредством обычного арифметического вычитания можно вычислить время выполнения любого участка программы, или даже отдельной команды. Одно неудобство - за минуту подробный лог разрастется до таких размеров… Но и здесь «все схвачено» - нам не надо минуту, мы возьмем да ограничим время в **Log from … for… seconds** (Рис. 48), установив его например 2m (2 милисек). Но и это еще не все. Находясь в режиме паузы, щелкните в **SIMULATION LOG** по подчеркнутому значению программного счетчика, например, **PC=0x0080** – откроется (даже если было закрыто) окно PIC CPU Source Code и указатель строки встанет по этому адресу, в котором находится следующая команда - в нашем случае: **movwf TrisA**. Что-то я увлекся подробностями диагностики, но тема того заслуживает. Добавлю еще, что не следует вообще пренебрегать подчеркнутыми значениями и терминами в окне **SIMULATION LOG**, но следует помнить, что дополнительные сведения при щелчке по ним доступны только при запущенной симуляции, например, в режиме паузы. Еще следует обратить внимание на появление синих вопросительных знаков. При щелчке мышью по ним доступны всплывающие подсказки соссылками на разделы **Help**.

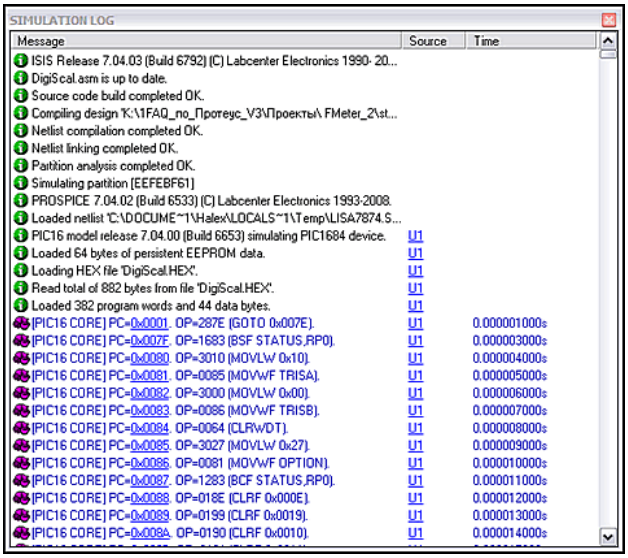


Рис. 49.

2.28. Меню Debug в развернутом виде (окончание) . Всплывающие окна. Суперполезное окно Watch Window.

У нас остались неразобранными до конца еще несколько опций меню **Debug**. По поводу **Use Remote Debug Monitor** пока ничего вразумительного написать не могу. Честно признаюсь, не нашел, что дает установка флажка в этой опции. Два пункта: **Tile Horizontally** и **Tile Vertically** выстраивают находящиеся на экране всплывающие окна соответственно горизонтально и вертикально. Ну и наконец последний раздел в котором содержится весь перечень всплывающих окон, доступных для просмотра. Установка флажка напротив любого из них вызывает его появление в режиме паузы (пошаговой отладки). Содержимое этого раздела меняется в зависимости от применяемого микроконтроллера или других программируемых компонентов. В нашем случае доступны восемь окон. С двумя из них мы уже познакомились достаточно подробно. Отмечу, что первые два: **Simulation Log** и **Watch Window** при установке соответствующих флажков находятся на экране даже в режиме симуляции в реальном времени, а не только в пошаговой отладке. Остальные доступны для просмотра только в паузе (пошаговой отладке). Большинство из них носят чисто информативный характер и по всплывающему меню правой кнопки мыши в них доступны для изменения только шрифт, цвета, ну иногда еще навигация, если все содержимое не умещается в окне. Вы можете самостоятельно установить флажки напротив нужных окон и посмотреть их содержимое.

Здесь же я подробно хочу остановиться на рассмотрении только **Watch Window**, поскольку оно обладает уникальными возможностями для отладки. При первом вызове **Watch Window** на экран оно абсолютно пустое. Ничего себе, смотровое окно – а именно так гласит дословный перевод. Да оно именно таковым и является, вот только надо выбрать – что мы хотим в нем смотреть. А выбор осуществляется по контекстному меню правой кнопки мышки. Давайте встанем в паузу, вытащим это окно на экран (установим флажок) и щелкнем внутри правой кнопкой. Появится контекстное меню, имеющее первоначально вид как на Рис. 50.

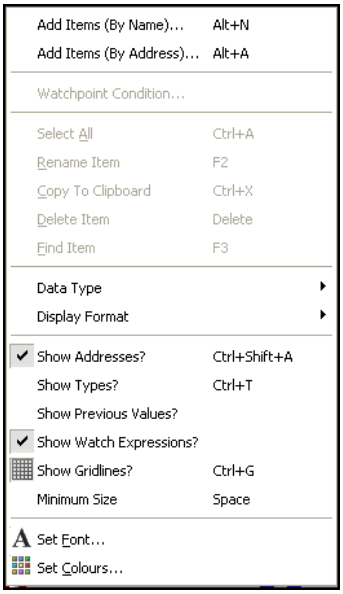


Рис. 50.

Первые две строчки его и позволяют наполнять **Watch Window** содержимым соответственно по имени (**By Name**) или по адресу в памяти микроконтроллера (**By Address**). Выбираем опцию «по имени» и получаем окно со списком названий регистров нашего микроконтроллера (Рис. 51).

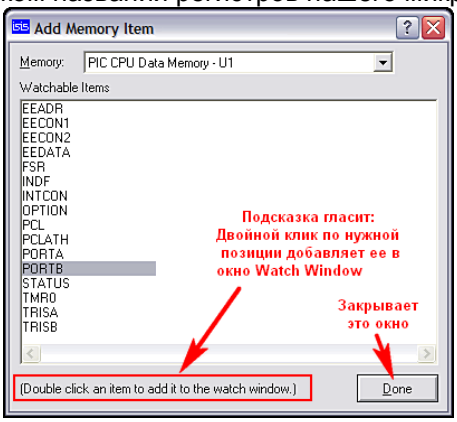


Рис. 51.

В качестве примера двойными щелчками левой, как гласит подсказка, я добавил в окно **Watch Window** два регистра: **PORTB** и **OPTION**. Эти регистры появились в списке **Watch Window**, причем у **OPTION** слева стоит плюсик, что означает наличие внутри его битов с различными назначениями. Кликнув по нему левой кнопкой можно развернуть содержимое.

Затем я вызвал другое окно опцией «по адресу» (Рис. 52). Тут уже придется задействовать клавиатуру. В качестве примера давайте добавим в окно регистр МК, в котором хранится указатель разрядов индикатора. Согласно ассемблерному листингу, этот указатель хранится по адресу **019h**. В строке **Address** вводим шестнадцатиричное значение адреса – **0x0019**, а в строке **Name** – его название. Чтобы не путаться – я ввел так как в ассемблерном листинге, хотя можно было назвать как кому нравится. Можно было вообще оставить это поле пустым, тогда **Isis** автоматически подставит введеный адрес и в это поле в качестве имени. Еще в этом окне имеется возможность сразу выбрать тип добавляемых данных (**Data Type**) и и формат, в котором будет представлено содержимое (**Display Format**). Я оставил все как есть по умолчанию, но иногда полезно поправить. Например, если бы у нас данные занимали 16 разрядов - лучше было бы применить **Word (2bytes)**, чтобы все значение помещалось в одной строке. В любом случае, исправить положение позже никогда не поздно. Обратите внимание – на Рис. 50 обе эти опции представлены в середине контекстного меню с раскрывающимися по черным стрелкам справа значениями. Поэтому всегда имеется возможность отредактировать и формат и вид.

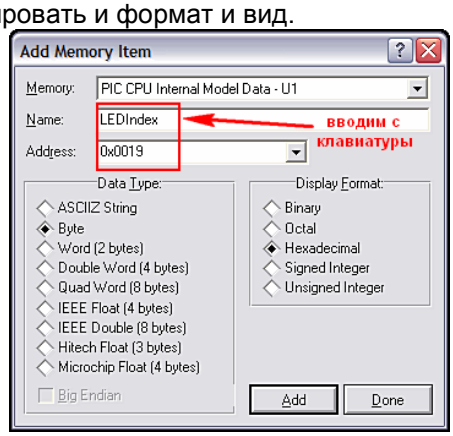


Рис. 52.

Конечный вид получившегося окна **Watch Window** представлен на Рис. 53. Обратите внимание, что я развернул регистр **OPTION** и теперь вижу его содержимое в соответствии с назначением битов по даташиту на PIC16F84A: биты пределителя **PSx** (0…3) в одной строке, а остальные побитно с именами в соответствии с даташитом.

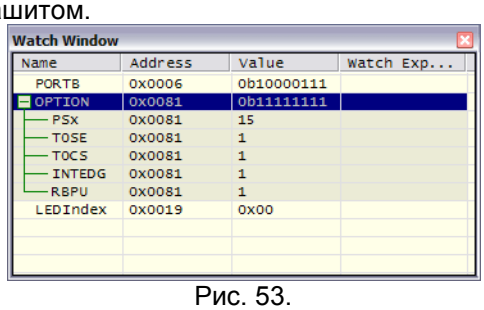


Рис. 53.

Вернемся к рассмотрению опций контекстного меню Рис. 50. Теперь, когда в окне **Watch Window** имеются выбранные переменные, все неактивные на рисунке функции меню стали доступными для использования. Здесь определенный интерес представляет функция **Watchpoint Condition…** (Условия прерываний для добавленных в **Watch Window** позиций), которая вызывает окно

представленное на Рис. 54. Ну вот, еще одна возможность останова симуляции. Вопрос: а нужна ли она нам? Надеюсь, вы оцените ее преимущества, когда познакомитесь поближе. Это единственный брекпойнт, который позволяет нам не просто прерваться по определенной команде в памяти программ, или по времени, как мы рассматривали раньше, а остановить симуляцию по достижению определенного значения внутренним регистром МК. При этом условием останова можно назначить изменение всего одного бита (!!!), например флага в регистре статуса. И не надо мудрить с выводом внутренних регистров на порты, установкой аппаратных брекпойнтов по совпадению и прочими «сексуальными извращениями». Все делается, как в знаменитой комедии: «Легким движением руки – брюки превращаются …». Для примера я назначил на Рис. 54 остановку симуляции при достижении регистром **LEDIndex** по адресу **0x019** значения **0x05**, и после таких манипуляций симуляция будет автоматом останавливаться при достижении в этой ячейке МК указанного значения.

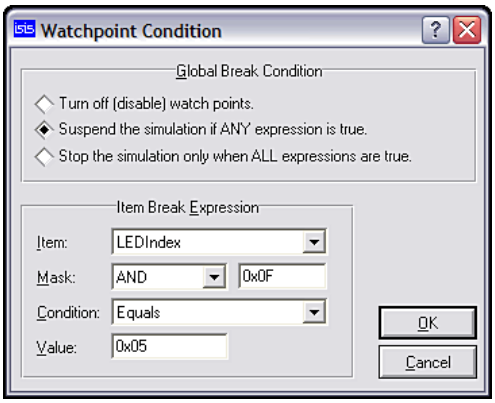


Рис. 54.

Рассмотрим, что для этого потребовалось. В **Global Break Condition** (глобальное условие прерывания) установлен переключатель **Suspend the simulation if ANY expression is true** (приостановить симуляцию, если ЛЮБОЕ из выражений истинно). Верхнее положение **Turn Off** соответственно отключает остановку по условию, а нижнее - **… only when ALL…** - останавливает по совпадению сразу всех условий. Обратите внимание, что для каждой позиции (строки) в окне **Watch Window** можно выбрать свое условие совпадения. Тогда нижнее положение переключателя сработает по совпадению сразу всех. В поле **Item** выбираем через раскрывающийся список наш пункт **Watch Window** – **LEDIndex**. В поле **Mask** аналогично выбираем логическое условие для маскирования, я выбрал **AND**, а в следующем поле вводим шестнадцатиричное значение маски – **0x0F** (в данном случае биты с 0 по 3). Если бы потребовалось контролировать только один бит, например четвертый, я бы ввел **0x10** – для «совсем чайников»: не забудьте, что счет байта идет с нулевого бита и заканчивается седьмым). Маска наложена, и в графе **Condition** (Условие) выбираем **Equals** (Равно), а в поле **Value** (Значение) набираем **0x05**. Жмем **OK**, после чего в **Watch Window** напротив **LEDIndex** колонка **Watch Expression** будет содержать наше выражение в компактном виде:

& 0x0F = 0x05 (маска : значение маски : условие : значение условия)

Чтобы отключить прерывание надо снова зайти в **Watchpoint Condition…** и поставить переключатель в **Turn Off**.

Давайте кратко завершим здесь с контектсным меню **Watch Window**. Пункты раздела начиная с **Select All** (Выделить все) и заканчивая **Find Item** (Найти пункт) и, надеюсь в особых комментариях не нуждаются, так как стандартны и для многих других программ. Про Data Type и Display Format я уже писал. Единственный нюанс – чтобы применить их отсюда надо встать на конкретный изменяемый пункт окна **Watch Window**.

В следующей группе устанавливаются флажки для выбора дополнительно индицируемых столбцов окна **Watch Window** (по умолчанию стоят два). Часто бывает полезен **Show Previous Values** (Показать предыдущие значения).

Show Gridlines – включает линии сетки таблицы окна. **Minimum Size** – сжимает размер окна по находящимся в нем значениям. Ну и две последние опции, как и раньше, служат для настройки шрифта и цветовой гаммы окна **Watch Window**.

Материал по отладке получился довольно объемным и растянутым, но, не зная вышеизложенного, – трудно понять преимущества ISIS перед другими отладчиками. Надеюсь, что и опытные пользователи Протеуса нашли здесь для себя полезные сведения.

Что-то я давно не прикладывал файл примера, - исправляюсь. Прикладываю здесь для практического освоения последнего материала. Просто запустите симуляцию, попробуйте отключить прерывание, или сделать свое. Здесь же присутствуют и графики из более раннего материала. Файл ассемблера уже подключен и откомпилирован через MPASMWIN. Как обычно версия 7.4 и секция для более ранних.

2.29. Исследуем исходник на ассемблере. Чем и как его открыть и редактировать.

В **п. 2.21** при анализе динамической индикации с помощью графиков мы определили, что причиной является пересечение во времени сигналов активных разрядов, вызывающее «наползание» индицируемых цифр одна на другую. Идеально было бы применить тот способ, который

использован в примере **t15demo.DSN**, а именно – стробирование (гашение) дешифратора короткими импульсами на время смены разрядов индикации. Легко сказать, но нелегко сделать, – ведь у нас задействованы все ноги МК, лишнюю никак не приделаешь. Значит, аппаратный способ исключается. Остается только правка программы. Конечно, поправить листинг ассемблера в самом Протеусе нам не удастся, он для этого и не предназначен. Потребуется какой-нибудь сторонний редактор. Самым тривиальным в данном случае является использовать родной виндоузный редактор Блокнот (он же Notepad в английской версии), поскольку листинг ассемблера с расширением **.asm** является обычным текстовым файлом и открывается любым редактором, поддерживающим кодировку DOS. Можно, при некотором навыке, воспользоваться даже MS Word, только придется поизощряться с сохранением файла в нужном формате. Но все это «крайние меры», поскольку ни дают никакой дополнительной информации по синтаксису текста – вы будете видеть обычный ровный черный текст, как на бумаге. Разбираться в таком тексте крайне затруднительно: где команды, где просто комментарии к программе – сходу не понять.
Конечно же, можно воспользоваться и родными редакторами, встроенными в компиляторы, например для PIC микроконтроллеров в том же MPLAB IDE или CCS PICC, но согласитесь – открывать для поправки пары строк кода навороченный компилятор как то душа не лежит.
Для таких целей идеально подходят редакторы с подсветкой синтаксиса языков программирования. Перечислять все доступные во «всемирной паутине» текстовые редакторы для программистов не хватит и нескольких листов. Почему то каждый начинающий программист спешит заявить о себе миру с создания именно текстового редактора, и каждый считает, что создал нечто уникальное, не имеющее аналогов. Но большинство из них поддерживают подсветку синтаксиса только для языков высокого уровня: Си, Бэйсик, Паскаль и т.п., ну в лучшем случае еще ассемблер x86. Правда, уважающие себя авторы, предусматривают возможность настраиваемой подсветки. Рекомендовать какой либо конкретный редактор, означает вызвать бурю негодования поклонников других программ, но я все же рискну. Вот уже на протяжении нескольких лет для быстрого просмотра и редактирования листингов программ я пользуюсь редактором **ConTEXT** (http://www.contexteditor.org). Кстати свежая версия **0.98.6** вышла 14 августа 2009 года. Почему именно он? Да потому что с 2007 года автор этой программы – Eden Kirin из Хорватии сделал его бесплатным и открыл доступ к исходникам. Теперь он развивается, как и Linux, что называется «всем миром». Ну а главные его достоинства – небольшой «вес» инсталятора – 1,6 Mb и огромное количество доступных для скачивания готовых подсветок языков программирования, в том числе и для ассемблеров всех популярных микроконтроллеров: AVR, PIC, ARM, 68000, 68HC11 и пр. Если кому не нравится готовый вариант подсветки, может создать свой или поправить его по своему вкусу. Ну и не последнее место среди «удобств» занимает встроенный русский интерфейс программы, а также функция **Compare** (сравнение двух файлов на идентичность текста). Не всякий навороченный редактор имеет такие возможности. Сайт программы англоязычный, поэтому для тех кто «плавает» в языке подскажу, что файлы подсветки синтаксиса доступны на страничке **Downloads** и распределены по группам **Highlighters** в алфавитном порядке. Например, чтобы скачать файлы для подсветки ассемблера PIC (их аж два варианта) заходим на страничку **Highlighters [M..P]** и скачиваем или **PIC Assembler.chl** или **PIC-MPASM.chl** ну или, как Винни-Пух: «и то и другое, и можно без хлеба». Эти файлы помещаются в папку **Highlighters** установленной программы, после чего подсветка доступна для выбора. Как это выглядит можно воочию убедиться по следующему скриншоту (Рис.55) с нужным нам участком программы. Да, еще для «не владеющих», при скачивании самой программы воспользуйтесь ссылкой: **Alternatively Download without making a Donation** на странице скачивания, если нужна бесплатная «халява», поскольку выше авторы предлагают «поддержать программу материально».

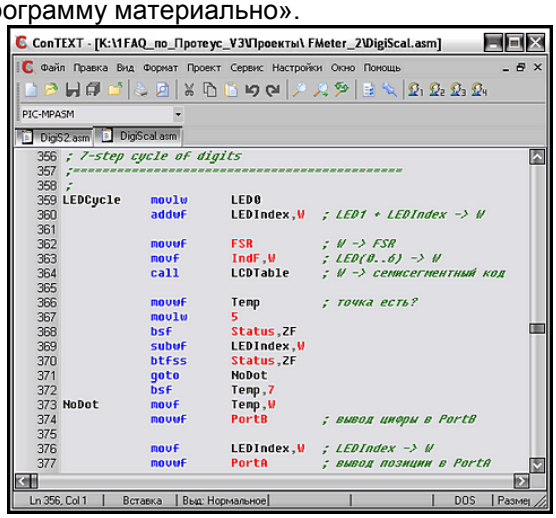


Рис. 55.

Еще раз подчеркиваю, чтоб в меня «не бросали камнями» - это не единственный текстовый редактор с подсветкой синтаксиса и, наверное, не самый лучший, но он прост и доступен, а для меня еще важно и то, что я им пользуюсь давно и привык, как к любому другому подручному

инструменту: паяльнику, пинцету и т.п. Желающие также могут попробовать другие бесплатные редакторы:
Notepad++ http://notepad-plus.sourceforge.net/ru/site.htm - 2.9 Mb (мое личное впечатление очень даже…);
PSPad - http://www.pspad.com/ru/compilers.htm - 4.2 Mb;
Notepad2 - http://flos-freeware.ch/notepad2.html - 251 Kb;
или совсем небесплатные:
MED - http://www.med-editor.com/indexus.html - 1.37 Mb;
SlickEdit - монстр, рекомендованый **dosikus**-ом, офф. сайт - http://www.slickedit.com – весит около 55 Mb, а вот здесь: http://megajohn.embedders.org/articles/?id=slickedit русскоязычное описание жутко навороченных его возможностей.
Я же на этом заканчиваю «рекламную паузу» и приступаю к этапу пластической хирургии. Итак, мы открыли файл **Digiscal.asm** в каком-нибудь редакторе и найдем в листинге две любопытных строки с комментарием:

```
356 ; 7-step cycle of digits
и
541 ; 8-step cycle of digits
```

Именно отсюда начинаются циклы динамической индикации, судя по переводу фраз. В одном случае для 7 разрядов, в другом – для 8. Давайте попробуем в ISIS воткнуть в этих местах брекпойнты и посмотреть – туда ли мы попали. Напомню, что точки останова можно поставить только в строчках с исполняемым кодом, либо метками – адрес которых совпадает со следующей исполняемой строкой. Ставим точки останова на строки:

```
359 LEDCycle    movlw     LED0
. . .
544             movlw     b'00000000'
```

Запускаем симуляцию обычной кнопкой **Play** и благополучно тормозимся на 359 строке (Рис. 56). Ну конечно, так и должно быть, ведь в исходном состоянии должны индицироваться 7 разрядов. Восьмой служит только для настройки. Остается только проверить это. Делаем брекпойнт 359-й строки неактивным, ну или совсем удаляем и снова запускаем симуляцию. Теперь она не тормозится, значит в строку 544 мы не попадаем. Давайте попробуем войти в режим установки нажав при симуляции на секунду кнопку **SB3.1** и мы благополучно «приземлились на наш брекпойнт. Статус кво восстановлен. Делаем вывод: со строки 356 начинается обычная индикация в рабочем режиме, со строки 541 – в режиме установки, когда присутствует 8-й разряд индикации.

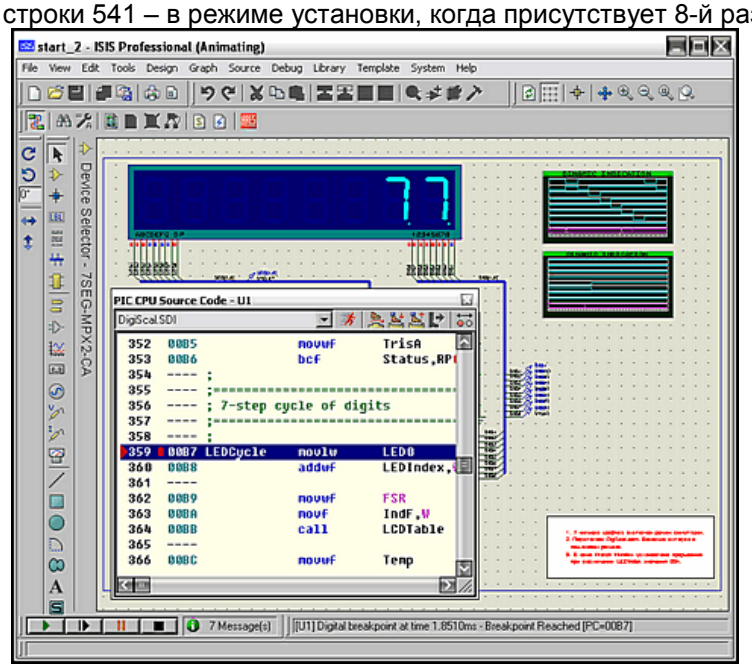


Рис. 56.

2.30. Реальные показания индикации в окне Watch Window.

Теперь нам необходимо определиться: а что же в дейтвительности должен показывать дисплей. Это тоже несложно сделать и мы это умеем. Обратите внимание вот на этом участок в начале листинга:

```
LED0    equ    010h    ;
LED1    equ    011h    ;
LED2    equ    012h    ;
LED3    equ    013h    ; ячейки
LED4    equ    014h    ; индикатора
LED5    equ    015h    ;
LED6    equ    016h    ;
LED7    equ    017h    ;
```

Ведь это ни что иное, как шестнадцатиричные адреса регистров памяти МК, в которые записываются отдельные разряды (цифры) значения текущей измеренной частоты. Давайте добавим их в окно **WATCH** по известным адресам, теперь мы умеем это делать. При этом дадим им имена те, что присвоены в листинге, чтобы не путаться в последствии. Ну и чтобы было «приятно глазу» для значений выберем **Display Format – Unsigned Integer** (беззнаковое целое). Запускаем симуляцию проекта и видим при отключенном генераторе на входе соответственно с **LED7** по **LED0 00000001**. Примем к сведению, что старший разряд **LED7** используется только при калибровке цифровой шкалы и не несет информации о частоте. Теперь подадим входную частоту **100k** (100кГц) соответственно в окне **Watch Window 00010026**. С учетом запятой должна индицироваться частота 00,10026 МГц. Проделаем те же манипуляции еще для пары частот, чтобы набрать статистику. Для генератора 1M (1МГц) получаем индикацию **00100091** или 01,00091 МГц. Для генератора 10M (10 МГц) – 01000701 или 10,00701 МГц, правда при этом загрузка ЦП составляет 100% о чем свидетельствует «горчичник» в логе симуляции. Для чего я это проделал? Да чтобы показать, что изначально существует погрешность в показаниях и при коррекции программы мы будем стремится получить именно эти значения для выбранных трех точек. И пусть Вас не смущает тот факт, что мы не получили точно соответствующие измеренные значения. Ведь и в реальной конструкции автор рекомендует произвести подстройку частоты кварцевого генератора. Мы же сейчас стремимся только воспроизвести в модели логику работы частотомера, ну и конечно по мере возможностей получить более-менее приемлемые результаты тестирования. Все изложенное в этом пункте с ведено в очередной вариации тестового проекта (вложение), а результат подачи тестовой частоты 100кГц приведен на рисунке 57.

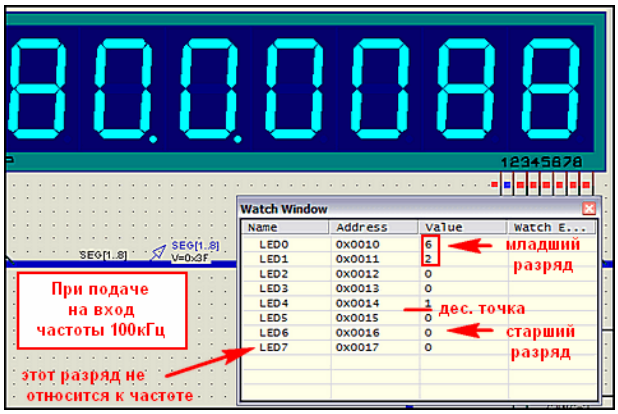


Рис. 57.

2.31. Корректируем ассемблерный файл. «И все таки она вертится».

Ну вот и подходит к концу наше затянувшееся исследование первого проекта. Настала пора внести исправления и убедиться, что все работает. Итак, нам необходимо получить гашение на короткий период индикатора при смене разрядов, а произойти оно может при подаче уровней логических нулей на сегменты индикатора, т.е. во все разряды порта PortB, к которым они подключены. Сделать это можно двумя командами:

```
movlw   b'00000000' ; записать все нули в регистр аккумулятор
movwf   PortB       ; записать содержимое аккумулятора в  PortB
```

Я не стал особо мудрствовать и добавил эти команды в начале цикла индикации **7-step cycle of digits**. Почему в начале? А какая собственно разница: в начале или в конце? Цикл все равно вращается по кругу, так что тут все относительно. Зато искать ничего не надо, мы уже определили начало цикла. Я бы покривил душой, если бы не коснулся еще одного момента. Для индикатора в свойствах я установил **Minimum Trigger Time 200us**, т.е. соизмеримым, ну или по крайней мере больше половины времени индикации одного разряда согласно графику, полученному нами в п. 2.20. Запускаем симуляцию без входного сигнала – все нули и десятичная точка только там где должна быть – заработало! Подаем сигнал 100кГц на вход получаем результат как на Рис. 58.

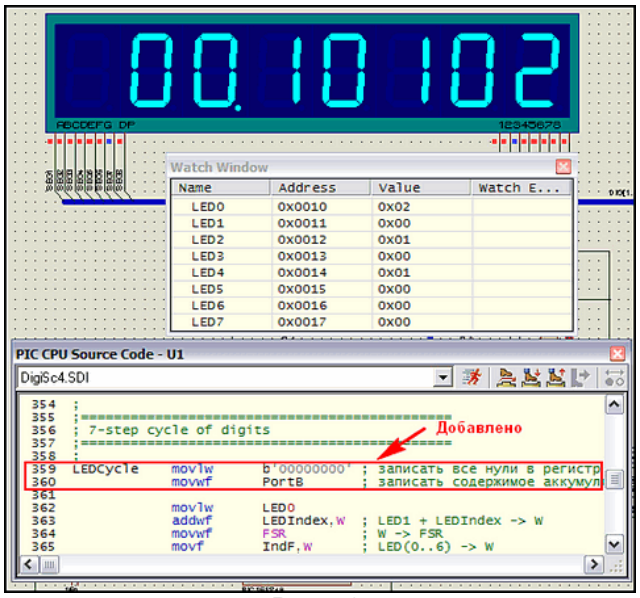


Рис. 58.

Ну что, – порядок значения совпадает, только погрешность увеличилась. Так и должно быть – ведь мы добавили в цикл индикации одного разряда две команды и общее время выполнения программы изменилось, а это для такого варианта измерения частоты существенно. Не зря же в начале листинга стоят два значения задержек **T1** и **T2**, а в коментарии сказано, что они подобраны для частоты кварца 4 МГц.

Теперь посмотрим на наши графики - не зря же мы их таскаем из проекта в проект. Запустим симуляцию первого из них и убедимся, что и здесь картинка изменилась (Рис. 59). На шине **SEG[1..8]** появились врезки изменений сигнала. Сравните с рисунком 36 или графиком в предыдущем примере.

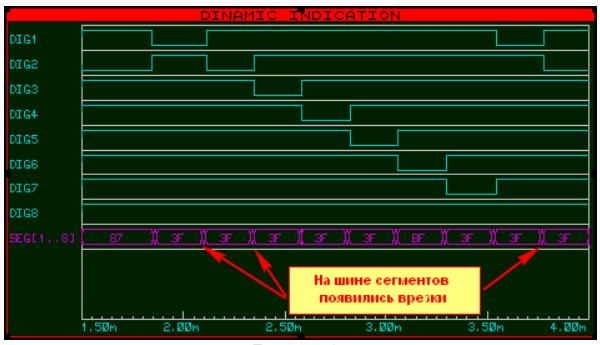


Рис. 59.

2.32. Финальный вариант проекта с рабочей индикацией.

Если растянуть место смены разрядов на индикаторе (второй график в проекте), то можно с помощью маркеров измерить время гашения индикатора (Рис. 60). У меня получилось около 19 мксек. Знатоки микрочиповских МК тут же возмутятся – откуда? Ведь мы добавили всего две команды, а каждая из них для частоты кварца 4 МГц длится 1 мксек (4 такта по 250 наносек). Согласен, но я не выбирал очень уж точно место, куда всунуть паузу, а следующая запись в **PortB** стоит на 13 строк ниже нашей вставки, да еще там присутствует вызов подпрограммы преобразования кода, пусть даже и очень короткой. Дотошные могут самостоятельно подсчитать по количеству исполняемых команд длительность паузы, ну или воспользоваться брекпойнтами в пошаговой отладке в нужных местах и вычислить по разнице времени между точками останова.

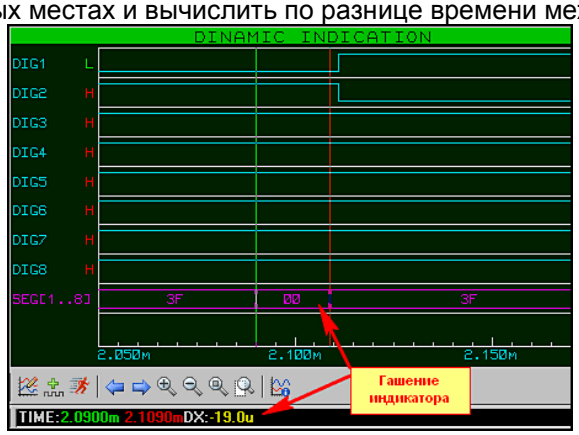


Рис. 60.

Я же сейчас постараюсь произвести коррекцию кода, чтобы подогнать показания индикатора к тем, что мы получили с исходным кодом. Опять буду действовать тривиально. Просматриваю ассемблерный листинг ниже нашей вставки и натыкаюсь на следующий участок кода:

```
; The first timing loop
;==========================================
O_K
        movlw     T1
        movwf     Temp
Pause
        decfsz    Temp,F
        goto      Pause
```

Типичная задержка, с использованием константы **T1** равной десятичному числу **67** (посмотрите в начале листинга присвоение ей значения). Еще чуть ниже находится проверка текущего разряда на достижение семи знаков и команда перехода к индикации следующего:

```
goto      LEDCycle    ; next 7xLED
```

Не буду вникать в тонкости грубого и точного подбора констант автором, для меня важно, что я добавил две команды внутри цикла индикации одного разряда, определяемого именно **T1**, поэтому уменьшать пробую ее. Уменьшение на 2 (добавлялось ведь 2 команды) слишком круто меняет результат измерения вниз – получаем для частоты **100k** результат 00.09850 МГц. Подставим значение **.66** и получаем 00.09965 МГц. Вот это уже лучше. Ну и, взяв за основу неписаное правило, – не знаешь что сделать – поставь **nop** (нет операции), я взял да и ткнул один лишний в код задержки приведенный выше перед циклом, начинающимся с метки **Pause**. Запускаю симуляцию при 100 кГц на входе и… 00,10026 МГц – как говорится: то, что доктор прописал. Проверяю с частотами 1 МГц и 10 МГц – индикация в соответствии с тем, что мы видели раньше в **Watch Window** для авторского кода. Дело сделано, но не совсем. Попробуйте нажать на кнопку **SB1** на 1-2 сек, чтобы перевести схему в режим установки промежуточной частоты и получите опять несоответствие того что в **Watch Window** и на индикаторе. Ну правильно, мы же скорректировали цикл только для индикации семи знаков. Теперь надо проделать аналогичную операцию с восьмизначной индикацией.

```
; 8-step cycle of digits
;==========================================
;

        movlw     b'00000000' ;
        movwf     PortA
;
        bsf       Status,RP0
        movlw     b'00010000' ; RA0..RA3 output,RA4 input
        movwf     TrisA       ;
        bcf       Status,RP0  ;
        clrf      LEDIndex    ; указатель цифры
EdtCycle

        movlw     b'00000000' ; записать все нули в регистр аккумулятор
        movwf     PortB       ; записать содержимое аккумулятора в  PortB

        movlw     LED0
        addwf     LEDIndex,W  ; LED1 + LEDIndex -> W
```

Обратите внимание, что я вставил код не в начале 8-step cycle of digits, а именно перед аналогичным для семизначной индикации участком кода. В финальном варианте при подаче на вход частоты 100 кГц в режиме частотомера показания соответствуют рисунку 61. Во вложении последний вариант со всеми исправлениями.

2.33. Выводы по применению динамической индикации в Протеусе и в реальности. Дополнительные ресурсы.

Хочу еще раз обратить внимание всех, кто использует проекты с семисегментными индикаторами в Протеусе, что программные модели многоразрядных сегментных индикаторов чисто цифровые. Что это означает, и чем это чревато для нас?

Во-первых: они не являются «потребителями тока», т.е. даже если Вы подключите напрямую к выводам микроконтроллера – никаких «перегрузок по току» не возникнет. Тут главное не нарушить полярности управляющих сигналов. Навешивание всевозможных мощных управляющих ключей на биполярных или полевых транзисторах в этом случае приводит только к увеличению нагрузки на процессор компьютера, и, как следствие, невозможности нормальной симуляции, поскольку все они являются аналоговыми элементами. С самого начала надо определиться – чего Вы добиваетесь. Если хотите симулировать схему – все аналоговые ключи долой! Когда очень уж надо проинвертировать выходной сигнал с выводов МК, например, когда индикаторы включены в коллекторную цепь транзисторов, – поставьте цифровые примитивы **INVERTER**, а для создания

печатной платы в ARES – сделайте отдельную копию дизайна с транзисторными ключами и прочей аналоговой бижутерией.

Во-вторых: все попытки изменить яркость многоразрядных сегментных индикаторов ISIS обернутся полной неудачей. Сегменты индикаторов имеют только два реальных «цифровых» состояния: включено и выключено. При небольшом количестве индикаторов можно попробовать применить схематичные (**Schematic**) одноразрядные модели индикаторов, но и они, если не переделать на свой лад (о чем поговорим позже) тоже имеют только два состояния. Единственное, что они Вам дадут – это реальную токовую нагрузку, поскольку для их создания были применены аналоговые примитивы.

В-третьих: для того, чтобы динамическая индикация при симуляции выглядела реально, нам необходимо предусмотреть кратковременные паузы при смене разрядов, исключающие наползание разрядов индикации. Паузы могут быть достаточно короткими (в районе 10 микросекунд), но их присутствие необходимо. Первым на эту особенность обратил внимание **dosikus** и материал по этой теме был в одной из веток форума Kazus:
http://kazus.ru/forum/topics/10434.html ,
а также размещен на сайте Каллиграфа:
http://www.kaligraf.narod.ru/nedodellki.html

Дополнительно реальность показаний корректируется свойством **Minimum Trigger Time** индикатора.

Нужны ли эти паузы в реальной индикации – решать Вам. Здесь хочу обратить ваше внимание только на один нюанс, который мы обнаружили и устраняли при изучении первого проекта. Вспомните про перекрывающиеся импульсы разрядов индикатора. А теперь представьте, что мы зажгли в двух соседних разрядах все сегменты – восьмерка. Может быть в реальности глазом подсветка и не будет заметна, но ведь сегменты подключены непосредственно к портам и на момент перекрытия импульсов разрядов нагрузка на порты подскочит вдвое!!! Порт может и выдержит, но опять-таки представим, что наш девайс питается от батарейки. Нужны ли нам эти абсолютно бесполезные броски по току? А если учесть то, что изложено чуть ниже в пунктах **b** и **c**, то можно запросто и порт МК спалить.

Теперь остановимся на некоторых особенностях реальной индикации:

a) для того чтобы человеческий глаз не замечал мерцаний картинка должна обновляться с частотой не менее 25 Гц (вспомним «эффект 25-го кадра»). Если мы имеем N индикаторов, для исключения мерцания необходима частота обновления 25 x N;
b) для того чтобы сохранить яркость свечения индикаторов в динамическом режиме – ведь средний ток через сегменты упадет – необходимо уменьшать токоограничивающие резисторы, а при большом количестве разрядов возможно и увеличивать напряжение питания индикаторов;
c) вот тут и потребуются в реальности мощные ключевые элементы и т.п, так как засветка, например, всех восьми сегментов (учитывая точку) знакоместа с током сегмента от 3 до 10 мА даст суммарный ток от 24 до 80 мА, но это в статике. Средний же ток через ключ, учитывая скважность импульсов для восьмиразрядного (N=8) для сохранения яркости свечения грубо необходимо увеличить в 8 раз (в реальности зависимость нелинейна) т.е. получим от 192 до 640 мА!!!

Ну и завершу я тему динамической индикации весьма неожиданным для многих заключением. Полученные для динамической индикации в результате симуляции в ISIS положительные результаты совсем не обязательно дадут аналогичную картину в реальности. Мы оперировали с моделями. У настоящего светодиодного индикатора присутствует куча параметров, которые не учтены в программной модели. Поэтому только проверка «в железе» может дать окончательный результат. В то же время воспроизведение даже реально работающей схемы в Протеусе позволяет выявить ее скрытые слабые стороны и позволяет исключить повторение ошибок при ее дальнейшем усовершенствовании. Ну и в заключение короткий список книг и он-лайн ресурсов по динамической индикации.

Книги:

- Вольфганг Трамперт «Измерение, управление и регулирование с помощью AVR-микроконтроллеров» *(глава 2 полностью посвящена динамической индикации, приведен расчет токоограничивающих резисторов)*.
- В. Н. Баранов «Применение микроконтроллеров AVR: схемы, алгоритмы, программы» (*глава 4 посвящена динамической индикации)* .Тот же материал опубликован в журнале «Схемотехника», № 5, 6 за 2006 г. Автор имеет свой сайт: http://bvn123.narod.ru/

Он-лайн ресурсы:

- А. В. Микушин Цикл лекций по теме «Цифровые устройства» на сайте СибГУТИ http://www.sibsutis.ru/~mavr/contCVT.htm. *(раздел 7.4 посвящен динамической индикации)*
- Динамическая индикация 9 разрядного индикатора по последовательной шине.(от DimAlt) у Радиокота http://radiokot.ru/lab/controller/08/ *(приложен рабочий проект для Протеуса с тестовой программой для Atmega8 на WinAVR).*
- Динамическая индикация и регулировка яркостиhttp://arv.radioliga.com/index.php?option=com_content&task=view&id=101&Itemid=49 *(интересная идея по ШИМ регулировке яркости индикаторов от Романа Абраша автора цикла статей по МК в журнале «Радиолюбитель»)*

2.34. Заключение к первой части.

На этом мы завершаем этап изучения Протеуса, который я условно для себя обозначил «Протеус для чайников». Излагая последующий материал, я буду считать, что Вы на примере разобранного выше проекта частотомера с динамической индикацией усвоили следующие материалы и приемы работы в программе ISIS:

- Программный интерфейс ISIS, назначение основных опций меню и кнопок;
- Создание и быстрое редактирование проекта в программе ISIS;
- Имеете навык по редактированию некоторых свойств применяемых компонентов;
- Ознакомились с начальными приемами размещения в проектах графиков и размещения на них трасс сигналов от зондов-пробников.
- Познакомились с возможностями применения моделей микроконтроллеров в Протеусе.

FAQ (ЧаВО) по PROTEUS для начинающих и не только. ЧАСТЬ II. PROTEUS для продвинутых пользователей.

Содержание.

3. Виды симуляции и типы моделей в ISIS.

3.1. Интерактивное и графическое моделирование. PROSpice - ядро симулятора ISIS.

Прежде чем мы продолжим изучение непосредственно программы **Proteus**, необходимо сделать небольшое лирическое отступление для того, чтобы следующий материал был более понятен. Имитация реального поведения электронных компонентов и схем, составленных из них, в программе ISIS основывается на применении SPICE-моделирования поведения компонентов. Язык **SPICE** (**Simulation Program with Integrated Circuit Emphasis**) был разработан в Калифорнийском университете Беркли в начале семидесятых годов прошлого века и де-факто стал основой для большинства программ, симулирующих поведение электронных компонентов. Это и P-CAD и MultiSim (Workbench) и даже такой монстр, как OrCAD. Отличие Протеуса от упомянутых заключается в том, что если там используется версия **PSPICE** (разработанная фирмой MicroSim Corporation, которую в 1998 году благополучно «проглотил» OrCAD), то ISIS **ProSPICE** базируется на другом клоне, именуемом **SPICE3F5**. Поскольку, начиная с версии PSPICE2, в ней появились некоторые отличия от стандарта SPICE, разработчики Протеуса предупреждают, что не всегда SPICE-модели сторонних фирм, большинство из которых предоставляет PSPICE-модели будут адекватно симулироваться в ISIS. Тем не менее, если заглянуть в библиотеки аналоговых компонентов ISIS, то можно заметить, что большинство моделей представлены именно SPICE-моделями.

ISIS поддерживает как интерактивную симуляцию в реальном времени, так и симуляцию с помощью графиков, воспроизводящих сигналы в определенных точках схемы с установленными там пробниками. В любом случае поведение схемы просчитывается с помощью встроенного в программу симулятора **ProSPICE**. С каждой новой версией Proteus ядро Prospice обновляется, например, в версии 7.6 SP0 поставляется и версия симулятора **ProSPICE 7.6.00**. Версию ядра вашей копии Proteus всегда можно посмотреть в логе симуляции после запуска на выполнение любого проекта или графика. **ProSPICE** различных версий не взаимозаменяемы и защищены лицензией на программу. Чем новее версия Proteus, тем более быстрое ядро ProSPICE встроено в программу.

Однако даже самые последние версии не в состоянии удовлетворить запросы наших амбициозных пользователей. Поэтому, если вы при симуляции схемы в реальном времени получаете в логе программы загрузку ЦП компьютера 100% и сообщение:

Simulation is not running in real time due to excessive CPU load

Симуляция не работает в реальном времени из-за чрезмерной загрузки центрального процессора

стоит подумать об использовании графиков для анализа поведения схемы.

Как правило, это сообщение вылетает из-за чрезмерной загруженности схемы аналоговыми компонентами и попыткой имитировать их работу на достаточно высоких частотах. Возможности ЦП компьютера небезграничны, а при интерактивной симуляции львиная доля ресурсов «съедается» на поддержку элементов, содержащих активную графику: индикация, анимированные провода и пробники и т.п. Не стоит забывать, что и виртуальные инструменты: осциллограф, таймер-частотомер, вольтметры/амперметры и пр. сами по себе являются активными моделями и тоже нагружают симулятор. В то же время при расчете графиков такой важный фактор, как быстродействие компьютера отходит на второй план. Протеус все равно рассчитает точки графика, только затратит на это больше времени. Именно поэтому применение графиков для анализа аналоговых схем предпочтительно. По этому поводу приведу дословную цитату из **ProSPICE HELP** - раздел **ADVANCED TOPICS => How to make interactive simulations run faster** (Как сделать интерактивную симуляцию быстрее):

«Моделирование цифровой схемы на два три порядка (т.е. вплоть до 1000 раз) быстрее, чем аналоговой. Это объясняется тем, что входящий в ProSPICE цифровой симулятор при обработке цифровых элементов опускает множество ненужных вычислений.

Например, компьютер с процессором Пентиум III 600 МГц может обрабатывать до 2 миллионов цифровых событий в секунду, но у того же компьютера при симуляции синусоидального генератора частотой 2 кГц загрузка ЦПУ может достигнуть 100%. Такая форма сигнала потребует расчета приблизительно 60 000 временных точек в секунду и для каждой из этих точек необходимо, чтобы было найдено решение уравнения сходимости – процесс намного более сложный, чем обработка простого цифрового сигнала.

Для многих компонентов интуитивно понятно: какое моделирование требуется – аналоговое или цифровое. Например, почти вся ТТЛ-логика и часть КМОП представлены цифровыми моделями, в то время как аналоговые ИС: операционные усилители, компараторы - аналоговыми. Все компоненты представленные (имеется ввиду в библиотеках) стандартными SPICE моделями требуют только аналогового моделирования.

Для некоторых целей строго аналоговые по своей природе компоненты диоды и резисторы могут быть представлены цифровой моделью. Это справедливо для монтажного ИЛИ, подтягивающих резисторов, элементов с открытым коллектором на выходе и диодно-резисторной логики».

Ну, вот мы постепенно и подошли к рассмотрению типов моделей в Протеусе.

3.2. Типы моделей в ISIS.

Если исключить из рассмотрения чисто графические модели, напротив которых в библиотеках стоит: **No Simulator Model** (*Не симулируемая модель),* а к ним относятся практически все разъемы и часть компонентов, к которым пока не разработаны модели, то все остальные можно поделить на четыре больших группы: различные **Primitive** (примитивы): **Analogue, Digital, PLD, Mixed** *(аналоговые, цифровые, программируемые логические матрицы, смешанные)*; **SPICE Model** *(Модели SPICE)*; **Schematic Model** *(Схематичная модель)* и **VSM DLL Model** *(Программные модели, объединенные в библиотеках DLL)*. На резонный вопрос – а зачем вообще представлены **No Simulator Model**, сразу отвечу – не забывайте, что существует еще и **ARES**, а как передать в него тот же разъем, если к нему нет ни схемного изображения, ни футпринта *(расположения выводов и занимаемого на печатной плате места)*. Вот для этих целей и находятся в библиотеках **ISIS** такие компоненты и если вы установили их в схему и попытаетесь запустить симуляцию, то получите сообщение об ошибке:

No model specified for (обозначение элемента)
Simulation FAILED due partition analysis error(s)

Об этом я уже говорил, но повториться не вредно – для таких моделей в свойствах установите галочку **Exclude from Simulation** *(Исключить из симуляции)*, чтобы избавиться от этой ошибки. Приступим к краткому рассмотрению остальных.

Сначала поговорим о примитивах. **Analogue Primitive** и **Digital Primitive**: К первым относятся большинство резисторов, конденсаторов, часть моделей транзисторов и диодов. Ко вторым - элементы цифровой логики из библиотеки **Simulator Primitives** и часть цифровых элементов из других библиотек. По своей сути это те же **SPICE**-модели, но встроенные непосредственно в симулятор **ProSPICE**. Особо хочу отметить примитивы из библиотеки **Simulator Primitives**, которые мы впоследствии будем использовать при создании схематичных моделей компонентов. Как и все SPICE-модели, они содержат ряд характерных свойств, которые заданы по умолчанию, но могут быть изменены пользователем по своему усмотрению. Описание **Properties** *(свойств)*, характерных для того или иного примитива всегда можно посмотреть в хелпе **ProSPICE Primitives** (Рис. 1).

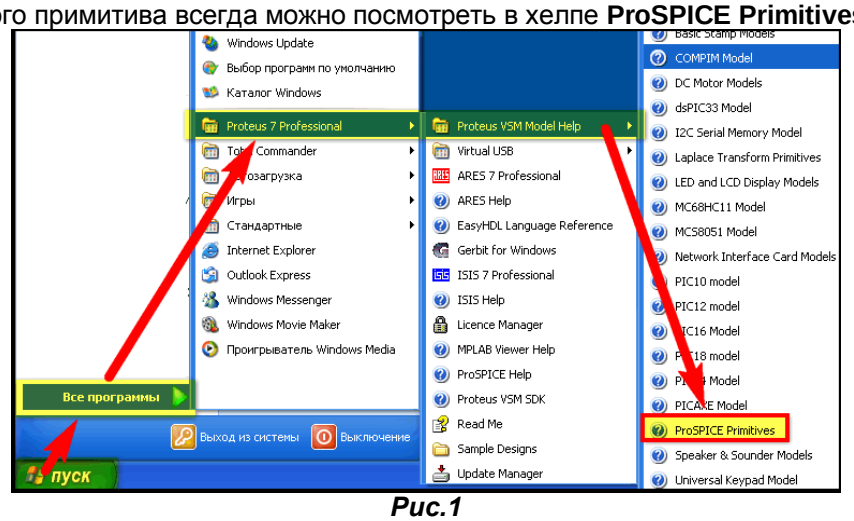


Рис.1

Кроме того, практически всегда доступна кнопка **HELP** при вызове свойств примитива, установленного в поле проекта (Рис. 2). Не стоит пренебрегать этими возможностями, особенно при моделировании аналоговых схем. Зачастую изменение всего одного из свойств компонента позволяют «оживить» молчавшую до этого схему. Ну и еще, что касается этих свойств. При создании более сложных моделей из примитивов они наследуют **Properties**, входящих в их состав примитивов и изменение этих свойств в окне **Other Properties** немедленно отзовется на поведении модели. Задание поведения и свойств **PLD Primitive** (логических матриц) осуществляется с помощью таблиц соответствия, содержащихся в JEDEC файлах. Подробно они рассмотрены в соответствующем разделе HELP: **PLD Modelling Primitives**. Особо хочу остановиться на **Mixed Model Primitives**, содержащих как аналоговые, так и цифровые свойства. Ряд свойств этих моделей применим в частности к КМОП логике и позволяет управлять уровнями переключения моделей, что важно при создании, например, мультивибраторов на КМОП микросхемах. Ну и особую группу в примитивах составляют **Real Time Primitives**. Эти примитивы мы будем применять для создания активных моделей, которые позволяют изменять и контролировать параметры схем в процессе симуляции.

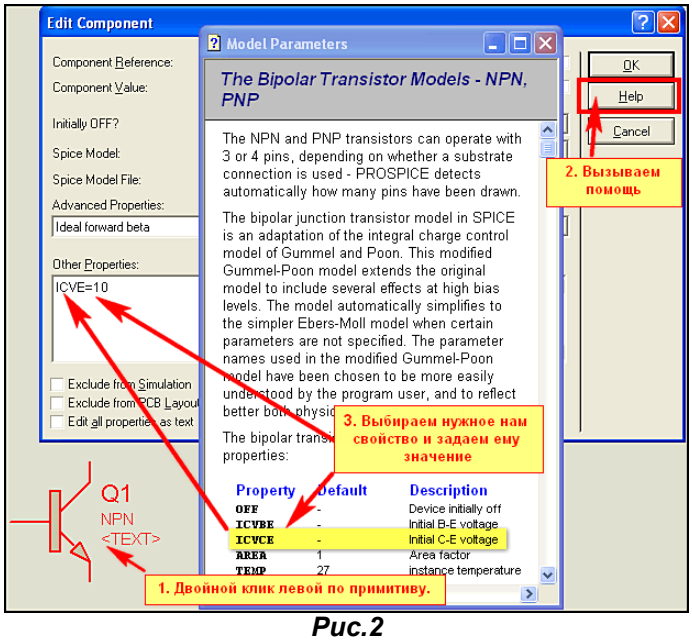


Рис.2

SPICE Model – это тоже модели **SPICE**, но содержащие текстовое описание на языке SPICE. Эти описания объединены в файлы-библиотеки с расширением **.SML** *(SPICE Model Library)* размещенные в папке **MODELS** программы Протеус. Чуть позже мы рассмотрим возможность применения описаний на языке **SPICE**, предоставляемых производителями компонентов на своих сайтах, для создания своих функционирующих моделей в ISIS.

Schematic Model – схематичные модели, построенные с применением аналоговых и цифровых примитивов. Схема создается на дочернем листе графической модели, затем преобразуется в файл описания модели с расширением **.MDF** *(Model Description File)* с помощью встроенного в ISIS компилятора (меню **Tools => Model Compiler…**). Далее MDF файл описания используется либо самостоятельно для соответствующей графической модели компонента, либо «родственные» по назначению или производителю файлы объединяются в библиотеки описаний компонентов с расширением **.LML**, которые также расположены в папке **MODELS** программы Протеус. Этой группе моделей мы чуть позже уделим особое внимание, поскольку **Schematic Model** наиболее приемлемы для быстрого создания собственных моделей.

Ну и последняя и самая «продвинутая» группа, «изюминка» Протеуса - это **VSM DLL Models** – программные модели. Это все модели микроконтроллеров, периферии, сложных индикаторов, датчиков и прочая, прочая, прочая. Все эти модели реализованы программным путем и используют для симуляции динамические библиотеки **DLL**, скомпилированные при их создании. Именно здесь кроются и их преимущества, и недостатки тоже. Любая ошибка программистов, допущенная, при создании модели выливается в многочисленные нарекания пользователей программы. Кроме того, при создании программных моделей, как правило, допускаются всевозможные упрощения, на некоторые из которых нет указаний в хелпах на эти модели. При использовании программных моделей в своих разработках не следует ожидать стопроцентного соответствия поведения реальному компоненту (микроконтроллеру, индикатору и т.п.). Почему то об этом напрочь забывают конечные пользователи Протеуса. Я по этому поводу привожу все время один и тот же пример с многоразрядными цифровыми индикаторами. Ну не реализованы там аналоговые свойства!!! Так зачем Вам в своих разработках, если они предназначены только для проверки работоспособности, цеплять к этим индикаторам всевозможную обвеску: резисторы, силовые ключи и пр. аналоговую мишуру, без которой реальная схема действительно работать не будет, а то и слегка «задымит». Конечно, если Вам предоставлен в аренду компьютер NASA, или даже нашего Российского ЦУП, то может и имеет смысл досконально воспроизводить схему. Но проделывать такое на домашней или офисной персоналке с весьма ограниченными возможностями – это уже из ряда «сексуальных извращений». Проявите творческую смекалку, которой всегда славился русский мужик. Ну и в заключение этого лирического отступления хочу отметить, что ни один симулятор не заменит полностью реальное моделирование устройства, каким бы наворочeнным он не был. Можно сколько угодно удачно гонять в ралли «Формула-1» на компьютере, но сев за руль реального авто благополучно въехать в ближайший столб. От этого не застрахован никто, недавно испытал на собственной шкуре. Это относится ко всем пакетам сквозного проектирования и даже к такому монстру как OrCAD. У каждой программы есть свои плюсы и минусы. А уж какой, и для чего пользоваться – выбирать Вам. Я не собираюсь здесь проводить сравнения и тесты, а перехожу к следующему разделу FAQ по Протеусу.

3.3. Взаимосвязи симулятора Proteus VSM.

Наш теоретический экскурс продолжается, и для пояснения работы симулятора я не нашел ничего лучшего, как привести картинку из старого описания **Proteus VSM SDK** с некоторыми моими комментариями (Рис. 3). Связи и направления взаимодействия различных составляющих симулятора показаны стрелками.

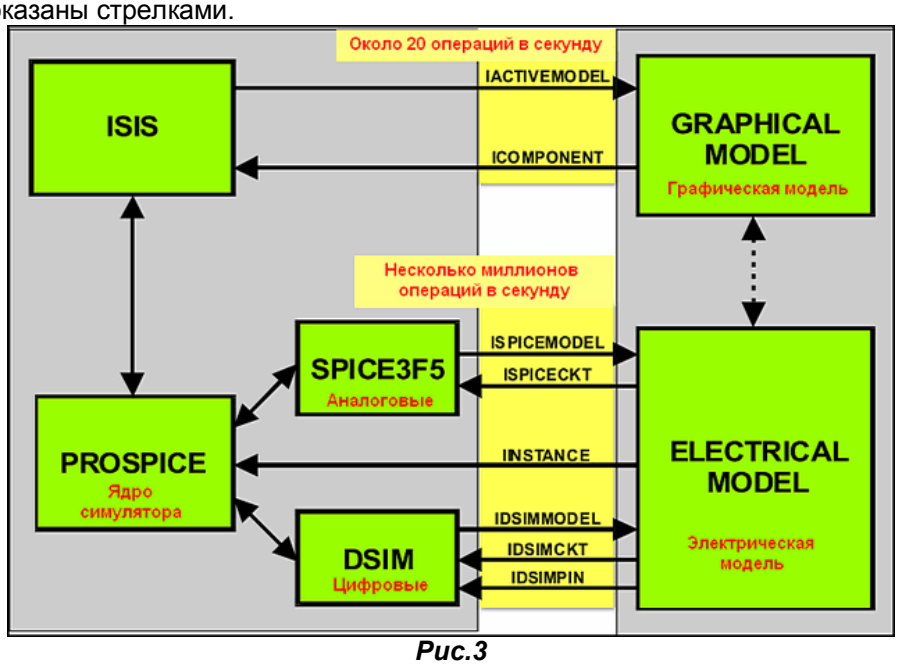


Рис.3

Из диаграммы видно, что ISIS непосредственно работает только с графическими моделями, размещенными в проекте и ядром симулятора **ProSPICE**. Как я уже упоминал выше, **ProSPICE** оперирует с аналоговыми свойствами моделей через **SPICE3F5** и цифровыми через **DSIM** – цифровой симулятор, который за счет того, что оперирует с меньшим количеством параметров, является наиболее быстрым. Обновление активной графики осуществляется по умолчанию с частотой около 20 кадров в секунду. Количество же операций по вычислению электрических параметров может достигать нескольких миллионов, и ограничено вычислительной мощью Вашего компьютера. Электрическая модель, в том числе и программная, может иметь и свой графический интерфейс, минующий управление ISIS. Поскольку операции по обслуживанию графики (а проще сказать мультфильма, который вы наблюдаете на экране) и расчета электрических параметров как бы разнесены в два разных интерфейса да еще с разными скоростями, между ними необходимо соблюдать некоторые соотношения, чтобы одно не противоречило другому.

3.4. Параметры анимации.

Возвращаемся к тем опциям меню **System**, которые я не стал описывать раньше. Итак, снова заглянем **System => Set Animations Options...** *(Рис.4)*. Ранее мы рассмотрели только назначение галочек в правой части этой панели, теперь пришла пора узнать – что скрывается за числами в левой части окна.

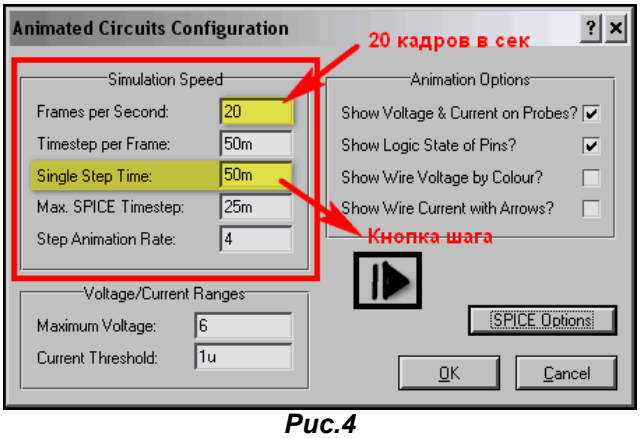


Рис.4

В первом же окне **Frames per Second** *(кадров в секунду)* и обнаружились наши 20 кадров. По рекомендации разработчиков это значение оптимально для большинства современных компьютеров, так как обеспечивает наилучшее соотношение между сглаженным воспроизведением графики и нагрузкой на графический процессор компьютера. Но если кому-то «шаловливые ручки» не дают покоя, то может поэкспериментировать с его изменением. Увеличение значения (max=50) увеличивает нагрузку на ЦПУ, уменьшение – уменьшает и замедляет анимацию.

Заранее хочу предупредить, и это будет подтверждено приложенным примером, что все изменения в **Animated Circuits Configuration** относятся только к открытому в данный момент проекту и сохраняются вместе с ним. Т.е. максимум, что вы сможете «испортить» это текущий проект и то при условии, что нажмете кнопку «Сохранить».

Куда больший интерес представляют **Timestep Per Frame** *(временной интервал на кадр анимации)* и **Step Animation Rate** *(частота кадров анимации в сек)*. По умолчанию они установлены соответственно **50m** (миллисекунд) и **4** (кадра в сек). Первый параметр определяет временной интервал, на который продвинется симуляция за один кадр, а второй – количество таких кадров в секунду. Здесь я должен сделать еще одно, ну очень лирическое отступление от темы, связанное с лингвистикой. Дело в том, что для аглицкого слова **Frame** *(кадр, фрагмент, фрейм…)*, я дважды использовал перевод «кадр», но именно в этом окне Протеуса все так замешано, что у нормально владеющего русским языком человека можно «сдвинуть крышу». Так вот эти два параметра больше связаны именно с симулятором **ProSPICE** и как бы взаимообратные. Чем меньше значение **Timestep Per Frame,** тем медленнее обновляется анимированная картинка, но зато можно разглядеть подробности быстропротекающих процессов. Те вычисления, которые **ProSPICE** не успел закончить к моменту окончания Timestep, урезаются и не входят в данный кадр. Максимальное **Step Animation** Rate – количество этих таймстепов в секунду, которое позволяет установить Протеус равно **10**. Еще одно немаловажная деталь – при уменьшении параметра **Timestep Per Frame**, его можно уменьшить даже до единиц и десятков микросекунд, но при этом сильно замедлится симуляция это то, что уменьшается нагрузка на ЦПУ компьютера. А теперь вспомните опять это противное сообщение в логе:

Simulation is not running in real time due to excessive CPU load

Если начать нахально на порядок увеличивать **Timestep Per Frame** даже на процессе, протекающем нормально, вы рискуете получить этот «горчичник». Для иллюстрации вышесказанного прилагается пример **Dinamic_point**, содержащий три варианта дизайна **Slow** (медленный), **Standard** (по умолчанию) и **Fast** (быстрый) с разными настройками **Timestep Per Frame**. По пожеланиям некоторых пользователей я в данном случае использовал AVR, хотя не уверен, что это корректно с моей стороны. Дело в том, что пример в версии 7.6, но я как обычно сделал файлы секций **.SEC** для импорта в старые версии. Но ведь использован процессор Mega8515 и в очень старых версиях он просто отсутствует. Вот и дилемма с приведением примеров на AVR – или использовать старую 90-ю серию, или лишить пользователей ранних версий возможности посмотреть примеры.

Обратите также внимание на параметр **Single Step Time** – именно он связан с кнопкой **Step** пошаговой анимации внизу слева в основном окне ISIS.. Вот именно с такими шагами симуляция будет продвигаться при каждом нажатии кнопки, а не по шагам микропрограммы контроллера, если таковой присутствует в схеме – для этого существует другая кнопка и в другом месте. Если Вам необходимо подробнее рассмотреть что то, то уменьшите это время. Для примера в Slow.DSN это время установлено **10mS**.

3.5. Параметры симуляции.

Перейдем к разбору параметров окна **System => Set Simulation Options...** Кстати, туда можно попасть и сразу из окна **Animated Circuit Configuration** *(Рис.4)*, нажав кнопку **SPICE Options**. Это окно предназначено для настройки параметров симулятора ProSPICE. Окно содержит пять вкладок: **Tolerances** *(допуски по точности)*, **MOSFET** *(параметры МОП транзисторов)*, **Iteration** *(количество шагов в вычислениях)*, **Temperature** *(температурные параметры)*, **Transient** *(параметры переходных процессов)*, **DSIM** *(параметры симуляции цифровых цепей).*

Здесь остановимся на тех значениях, на которых акцентировали внимание разработчики Лабцентра в стандартном **HELP** по **ProSPICE** в разделе **ADVANCED TOPICS => SIMULATOR CONTROL PROPERTIES.** Для остальных я просто приведу расшифровку назначения. Если кому то этого покажется мало, то рекомендую найти любую книжку по любому из пакетов программ: Electronic Workbench, Multisim, Microcap, OrCAD ну или наконец по самому PSpice. Дело в том, что параметры SPICE симуляции для всех этих программ обозначаются практически одинаково. В конце этого параграфа я размещу список книг, в которых они описаны точно.

- Вкладка **Tolerances** *(Рис.5).* На этой вкладке представлены параметры, определяющие с какой точностью **ProSPICE** производит вычисление решений. При очень высокой точности может потребоваться больше времени для моделирования и в ряде случаев решение может не иметь сходимости вообще.
  ABSTOL – абсолютная ошибка расчета токов;
  VNTOL – допустимая ошибка расчета напряжений;
  CHGTOL – допустимая ошибка расчета зарядов;
  RELTOL – допустимая относительная ошибка расчета напряжений и токов;
  PIVTOL и **PIVREL** – абсолютная и относительная величины элемента строки матрицы узловых проводимостей;
  GMIN – минимальная проводимость ветви цепи – определяет утечки обратно смещенных полупроводниковых переходов и других точек схемы с высоким импедансом. В ряде случаев уменьшение этого значения может помочь получить сходимость для схем, которые при стандартных параметрах по умолчанию не имеют решений, однако при этом понизится

точность вычислений. Проводимость меньше установленного здесь значения принимается равной нулю;
TRANGMIN – минимальная переходная проводимость;
RSHUNT – допустимое сопротивление утечки для всех узлов относительно общей шины.

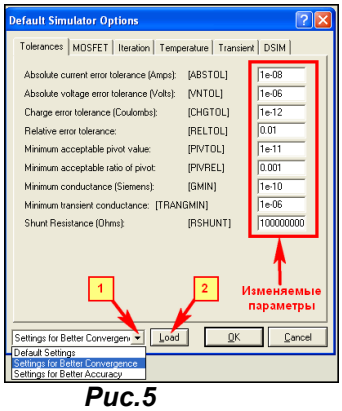


Рис.5

- Вкладка **MOSFET** *(Рис.6).* На этой вкладке сосредоточен ряд параметров, определяющих в **SPICE** геометрические размеры **MOSFET** элементов при условии, что они расположены на подложке интегральной микросхемы. Именно поэтому здесь даны значения в метрах. Здесь наиболее значимо то, что установкой галочек можно задавать значения для старых версий моделей **SPICE** или для новых **SPICE2**.
  DEFAD – площадь диффузионной области стока, м^2;
  DEFAS – площадь диффузионной области истока, м^2;
  DEFL – длина канала полевого транзистора, м;
  DEFW – ширина канала, м.

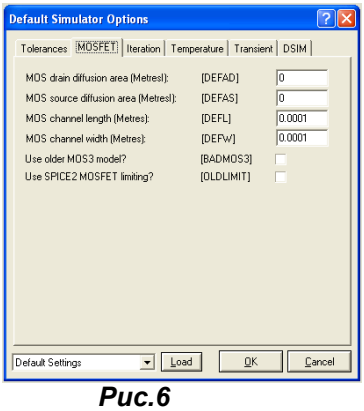


Рис.6

- Вкладка **Iteration** *(Рис. 7).*
  Integration Method – определяет каким методом **SPICE** вычисляет точки сходимости решений. Возможны два варианта: **GEAR** – метод Гира или **TRAPEZOIDAL** – трапецеидальный. Первый считается более новым и точным и применен по умолчанию, хотя и требует больше машинного времени. Здесь замечу, что в столь популярном сейчас по соседству **MultiSim 10** по умолчанию используется метод трапеций. Три последующие опции определяют количество шагов, которые **SPICE** применяет для вычисления каждой операционной точки.
  MAXORD – максимальный порядок (от 2 до 6) метода интегрирования дифференциального уравнения;
  SRCSTEPS – размер приращения напряжения питания в процентах от его номинального значения при вариации напряжения питания (используется при слабой сходимости итерационного процесса);
  GMINSTEPS – размер приращения проводимости в процентах от GMIN (используется при слабой сходимости итерационного процесса);
  ITL1, **ITL2** и **ITL4** – максимальное количество итераций (шагов) соответственно в режиме DC, передаточных функций при переходе к следующей точке в режиме DC, при переходе к следующему моменту времени в режиме Transient;
  Ну и наконец, три флажка, установка которых может в ряде случаев ускорить процесс вычислений:
  NOOPITER – перейти непосредственно к вычислению GMIN;
  COMPACT – сжатие вычислений LTRA. Применим только для линий с потерями LOSSYLINE, и заключается в том, что похожие по значениям точки не просчитываются, а принимаются идентичными, что позволяет сократить время вычислений.

BYPASS – включен по умолчанию и позволяет обойти при вычислениях неизменяемые элементы.

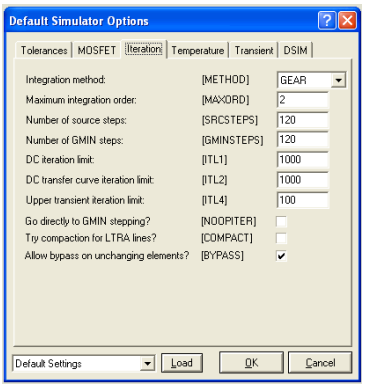


Рис.7

- Вкладка **Temperature** *(Рис. 8).* Здесь всего два параметра:
  TEMP - глобальная температура компонентов, применяемая ко всем элементам моделируемой схемы, которые имеют ее в своих параметрах, а это: резисторы, диоды, полевые и биполярные транзисторы. Будьте внимательны: параметр учитывается только для аналоговых компонентов. Если перевести те же резисторы и диоды в режим **DIGITAL**, он не просчитывается! Кроме того, для каждого из этих компонентов можно установить параметр **TEMP** индивидуально в его свойствах.
  TNOM – номинальная температура, при которой проводились измерения термозависимых параметров модели *(во загогулина, понимаишь!)*, ну или проще стандартная при которой снимались параметры. Этот параметр также можно устанавливать индивидуально.
  Ну и у Р. Хайнемана отмечается, что поскольку по умолчанию обе температуры равны +27 градусов по Цельсию - легко догадаться, что **SPICE** «родился» в штате Калифорния, а не в Сибири (*это так, к слову - для расширения кругозора)*.

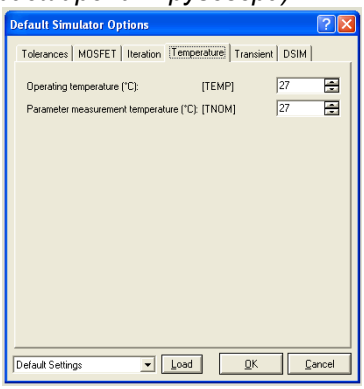


Рис.8

- Вкладка **Transient** *(Рис. 9).*
  NUMSTEPS – количество шагов, применяемое при анализе переходных процессов
  TRTOL – допуск на погрешность вычисления переменной – наиболее полезный на этой вкладке параметр. Если результаты моделирования имеют остроконечные всплески или наблюдается перерегулирование, то можно попробовать уменьшить этот параметр.
  TTOL – временной разброс при анализе смешанных (аналого-цифровых) схем;
  TMIN – минимальный шаг по времени при анализе аналоговых схем.

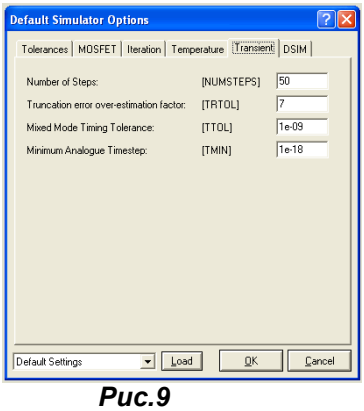


Рис.9

- Вкладка **DSIM** *(Рис. 10).* В седьмых версиях имеет всего два изменяемых параметра.
  Random Inicialisation Values *(случайные значения при инициализации)* определяет: каким образом устанавливаются значения для цифровых примитивов, которым в свойствах прописано значение, а это в частности модели микросхем памяти, установка для которых **INIT=RANDOM** заполняет их при старте в зависимости от установленного здесь переключателя либо **Fully Random** *(случайными)*, либо **Pseudo-random** *(псевдослучайными)* последовательностями из заданного диапазона от 1 до 32767. Если вы используете для микросхемы памяти загружаемый файл, но он заполняет ячейки не полностью, то при установленном **INIT=RANDOM** последовательностями заполнятся оставшиеся свободными ячейки.
  Propagation Delay Scaling *(масштабирование задержек распространения)* определяет установки для всех временных свойств цифрового моделирования, которые не заданы строго в свойствах конкретных моделей. По умолчанию для этих свойств установлен множитель **Scale all values by constant** равный **1**. Установка **Pseudo-random** или **Fully Random** позволяет задать диапазон от нижнего **Lower Scalling Limit** до верхнего **Upper Scalling Limit** значения, в котором будет изменяться этот множитель по псевдослучайной в том же диапазоне что и выше или полностью случайной последовательности. Это позволяет приблизить моделирование к реальности и исключить проектные недоработки, связанные с повторяющимися временными процессами, не имеющими отражения в реальных условиях.

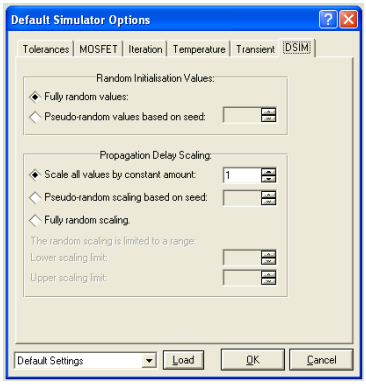


Рис.10

Ну, вот мы и прошлись по всем вкладкам параметров симуляции, и остался незатронутым один пункт, повторяющийся на всех вкладках. В последних версиях Протеуса внизу этого окна для удобства пользователей введен выбор трех предустановленных наборов параметров. По умолчанию всегда стоит **Default Setting** *(Рис. 5)*. В большинстве случаев для моделирования этого более чем достаточно, однако иногда имеет смысл изменить предустановки. В наличии имеются еще две:
Setting for Better Accuracy *(установки для наилучшей точности)* – но при этом увеличивается время вычислений и нагрузка на ЦП компьютера;
Setting for Better Convergence *(установки для наилучшей сходимости решений)* – здесь соответственно все наоборот.
Для того чтобы загрузить установки необходимо выполнить действия в последовательности по Рис. 5. Обращаю ваше внимание, на то, что загрузка (**Load**) действует сразу на все вкладки, относящиеся к аналоговой симуляции. Естественно температура и DSIM при этом не модифицируются. Для возврата к «заводским» установкам от Лабцентра снова загружаем **Default**. Ну и еще – эти настройки сохраняются в **ISIS**, а не в проекте! Поэтому, уйдя из дефолтовых настроек, при последующем закрытии/открытии **ISIS** не забудьте проверить – что Вы там назначили и не пора ли возвратить **Default** на место.
Ну и в заключение этого параграфа, как и обещал – список литературы с моими пояснениями. Ссылок не даю, поскольку на файлообменниках они часто меняются, но в сети эти книжки «гуляют».

Разевиг В.Д. «Схемотехническое моделирование с помощью программы Micro-Cap» 2003 г.
К сожалению, безвременно ушедший Виктор Данилович больше не порадует нас своими шедеврами. Если я когда-нибудь и напишу что-либо подобное, то буду считать, что жизнь прожита не зря. В любой его книге по косточкам разобран описываемый предмет. Например, из этой я частично «списал» описания параметров симуляции SPICE (стр. 62-65).
Короновский А.А., Храмов А.Е. «Применение Electronics Workbench для моделирования электронных схем» Учебно–методическое пособие. Саратовский Госуниверситет, 2004.
Как видите, даже старенький Electronics Workbench перекликается с Протеусом, а все благодаря SPICE. На стр. 17-18 те же описания параметров Spice, как впрочем, и Multisim:
Хернитер М. «Multisim 7 Современная система компьютерного моделирования и анализа схем электронных устройств». 2006 г.
Хайнеман. Р. PSPICE «Моделирование работы электронных схем». 2005 г.
Петраков. О М. Создание аналоговых PSPICE моделей радиоэлементов. 2004 г.
Последней книги нет в сети, я пользуюсь бумажным вариантом. Но аналогичный по названию цикл статей выходил в журналах «Схемотехника» в 2002 г. В них же в районе 2005 г. был и цикл по цифровым моделям. Для тех, кто решил

попробовать самостоятельно создавать SPICE модели последняя книга и статьи будут наилучшим подспорьем в работе.

4. Создание моделей компонентов в ISIS.

Чтобы больше не грузить Вас голой теорией, мы на время отложим теоретические основы в сторону, поскольку они так или иначе всплывут в этом разделе и будут рассмотрены, как говорится: «по ходу пьесы». Займемся более насущными проблемами, тем более что форум уже перегружен вопросами по теме создания моделей.

4.1. Создание графических моделей компонентов.

Итак, в разделе **3.2** мы кратко классифицировали типы моделей. Для начала создадим **No Simulation Model**. Почему именно ее, а не сразу микропроцессор Intel Core Duo? Да потому что любая модель компонента в ISIS должна иметь как минимум хоть одно графическое отображение, а иначе как ее разместить в проекте? Ну и потом любая, даже не симулируемая в ISIS модель при ее создании проходит все этапы, присущие даже самой сложной модели микроконтроллера и нам необходимо с ними познакомиться. Для начала необходимо создать графическое изображение модели с набором выводов (ножек, пинов – называйте, кто как хочет). Открываем новый проект и начинаем рисовать. В первую очередь необходимо нарисовать тело компонента. Рисование осуществляется с помощью элементов 2D графики. На рисунке показано: какие фигуры из какого режима создаются *(Рис.11)*. Обратите внимание, что я использовал режим рисования **COMPONENT** – чтобы не нарушать общий стиль графики компонентов в ISIS, но при желании можно воспользоваться любым другим из предустановленных или создать свой. Напомню, что стили графики представлены в меню **Template => Set Graphic Styles…** Если надо создать свой, жмем кнопку **New** и задаем название и параметры линий, цвета и т.п. *(Рис. 12)*.

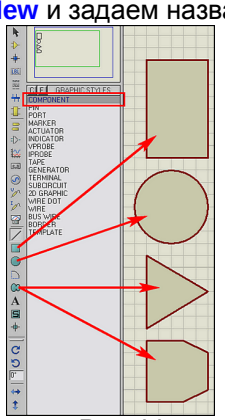


Рис.11

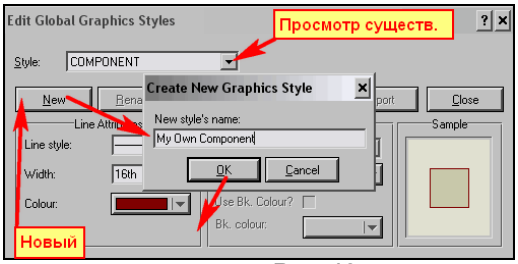


Рис.12

СОВЕТ 1. *Если необходим стандартный графический элемент, то зачастую нет необходимости создавать его с нуля. Например, тот же символ операционного усилителя можно взять из библиотеки символов. Процедура напоминает добавление компонентов, но в другом режиме. Выбираем в левом меню режим **S** и далее по пунктам практически, как и с компонентами (Рис. 13). Замечу только, что пока Вы не начали активно создавать собственные компоненты, пользовательская библиотека **USERSYM** пуста, и выбрать стандартные символы вы можете только из библиотеки **SYSTEM**. Поместив его в селектор символов на шаге 5 всегда можно добавить его в поле проекта оттуда, как и обычный компонент.*

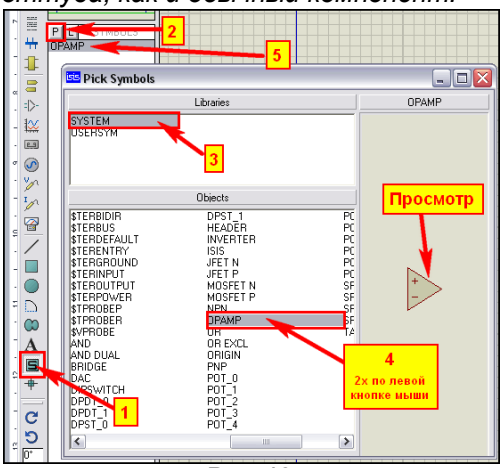


Рис.13

СОВЕТ 2. *Если в библиотеке компонентов уже имеется готовый, похожий на тот, что Вам необходим то воспользуйтесь функцией **Decompose** (пиктограмма с молотком в верхнем меню или через меню по правой кнопке мыши), чтобы «разобрать его на запчасти» и использовать*

необходимые графические элементы для своего компонента. Не бойтесь испортить существующие библиотеки. Если Вы не применяли «превентивных» мер к своей копии программы, то они по умолчанию защищены от записи. К тому же, «разобранный» компонент уже не является компонентом библиотеки, а представляет собой чистейший набор графики и текстовый скрипт с описанием его свойств. Для примера я «разобрал» ОУ 741 (Рис. 14).

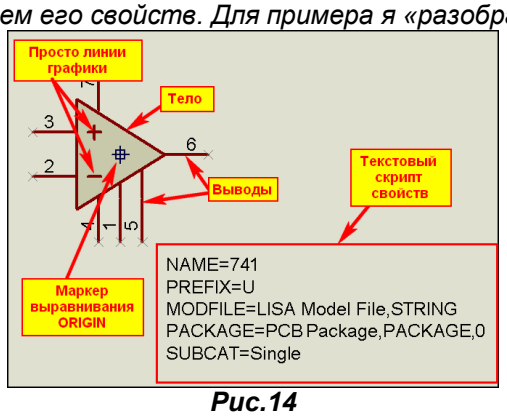


Рис.14

Внутри тела компонента вы вольны помещать любые элементы графики – линии, текст. Так например у того же ОУ 741 линиями начерчены символы прямого и инверсного входов. Здесь только одно примечание – не забывайте, что для того чтобы они были видимыми – само тело должно быть помещено на задний план (меню **Edit => Send To Back** или через **Ctrl+B**). Соответственно при размещении линий и текста желательно в селекторе выбирать стиль **COMPONENT**, чтобы все выглядело единообразно. Ну, уж если кому то нравится разрисовывать свои творения всеми цветами радуги, то и это не воспрещается.

Итак, после того как тело компонента со всеми прибамбасами нарисовано необходимо установить маркер выравнивания **ORIGIN**. По этому маркеру наш будущий компонент будет привязываться к сетке в проекте. На рисунке 14 он установлен в центре тела, хотя раньше разработчики программы рекомендовали ставить его по концу левого верхнего вывода, а для активных компонентов (например, индикаторов) левый верхний угол тела – потому что отсчет положения изменяющихся в процессе симуляции элементов (тех же светящихся сегментов индикатора будет вестись по нему). Но для обычных компонентов – микросхем и т.п. положение не так существенно, так что можно лепить куда хотите, но помните о привязке к сетке. Для установки маркера можно воспользоваться левым меню («прицел» ниже **S)**, либо через всплывающие меню правой кнопки мыши **PLACE => MARKER => ORIGIN**.

Ну и последним этапом создания графического изображения является размещение выводов компонента. Переходим в левое меню **Device Pins Mode** и выбираем в селекторе нужный нам тип вывода для размещения. Как правило, в большинстве случаев достаточно **DEFAULT** – т.е. простого прямого вывода, ну может еще **INVERT** – вывод с кружком. Замечу, что графическое изображение и назначение вывода в данном случае еще никак не связаны. И даже если вы воткнете на будущий вывод питания символ вывода **INVERT** – это не значит, что оно у вас изменит полярность, ну или что-то еще в том же духе. Пока речь идет ТОЛЬКО о графическом изображении. Если кому то покажется мало имеющихся в селекторе по умолчанию шести типов выводов, может добавить туда из имеющейся библиотеки **SYSTEM**. Процедура все та же. Находясь в режиме **Device Pins Mode**, двойным щелчком в поле селектора (или одинарным по буковке **P)** входим в библиотеку и добавляем из имеющихся двух десятков в селектор тот, который понравился. Когда он Вам там надоест, щелкаем правой кнопкой по нему и задаем **Delete**. При этом элемент будет убран из селектора, а не из библиотеки!!! Впоследствии вы снова можете его достать.

При размещении выводов обратите внимание на тонкое перекрестие в виде **X** – это внешний конец вывода, к которому будут позиционироваться провода, поэтому он должен быть направлен наружу. Если необходимо повернуть или отразить вывод воспользуйтесь соответствующими опциями в меню правой кнопки мыши. Когда выводы расставлены по местам, можно приступить к их нумерации и описанию назначения. Для нумерации при большом количестве выводов проще всего воспользоваться функцией меню **Property Assignment Tools**. В строке **Sting** задаем **PIN=NUM#** и задаем начальное **Count** равным **1**. Об этом режиме я уже рассказывал. Вот с наименованием, несколько сложнее, поскольку выводы, как правило, имеют уникальные имена. Хотя, для всевозможных многоразрядных портов и тут можно применить **NAME=(что-то там)#**. При небольшом количестве пинов их проще отредактировать вручную. Двойным щелчком по выводу входим в его свойства и задаем в соответствии с рисунком 15.

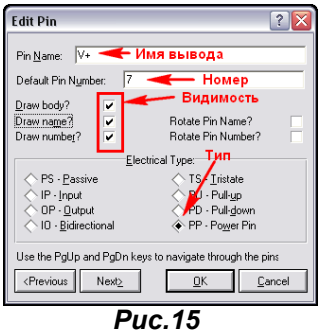


Рис.15

Здесь, пожалуй, остановимся чуть подробнее. В окне **Pin Name** вводится наименование вывода. Желательно, чтобы оно отражало физическую суть вывода, но в тоже время было не очень длинным и обязательно уникальным (не повторяющимся больше в этой модели). Если планируется, что это имя будет видимым, то оно должно вписываться в отведенном для этого месте и не наползать на другие. Ну про **Default Pin Number** и так ясно, что необходимо, чтоб он совпадал с номером вывода того корпуса, который будет впоследствии принят по умолчанию. Забегая вперед, замечу, что для одного компонента может быть назначено несколько корпусов.

Особо хочу остановиться на группе флажков, которые я обозначил как Видимость:

Draw Body? – вот она та пресловутая видимость вывода, которая позволяет присоединять к нему проводники. Она нам очень потребуется в дальнейшем, чтобы суметь подключиться к выводам компонентов скрытым по умолчанию, т.е. у которых отсутствует галочка в этом поле. Мы можем их увидеть в сером цвете, установив через меню **Template** => **Set Design Default** галочку **Show Hidden Pins**, но по-настоящему активными, т.е. доступными для использования они станут, когда в этом месте для них будет стоять галочка. Соответственно галки **Draw Name?** и **Draw Number?** Делают видимыми имя и номер вывода, а **Rotate…** поворачивают их на 90 градусов против часовой стрелки. Ниже расположено окно выбора электрического типа вывода. На рисунке у меня открыт вывод плюса питания ОУ, поэтому ему строго назначен тип **PP – Power Pin**. Если при создании компонента вы затрудняетесь в определении электрического типа, то просто оставьте **PS** – принятое по умолчанию. Поясню маленький нюанс: Для ISIS здешнее назначение в высшей степени безразлично, а вот ARES при разводке дорожек примет данное назначение, как руководство к действию и в зависимости от этого произведет трассировку, например для **Power Pin** более широкими дорожками. Ну и внизу данного окна имеются кнопки позволяющие ускорить редактирование **Previous** и **Next** – переключают окно на следующий по **номеру** **Pin Number** вывод, **OK** и **Cancel** – стандартные подтверждение и отмена

На рисунке 16 приведен вид селектора в режиме расстановки выводов, а также графическая модель ОУ 741 перерисованная в формат принятый в Российской технической документации. Я здесь просто разобрал стандартный 741, нарисовал свое **Body**, перетащил выводы с разобранного к новому, для выводов питания включил галку **Show Name**, а графику вывода 2 заменил на **INVERT**. Во вложении проект, в котором проделана эта работа, но еще не создана новая модель.

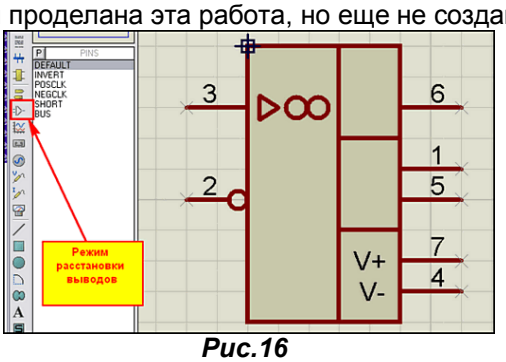


Рис.16

4.2. Пять этапов или пять составляющих окон создания модели функции Make Device.

Как и у В.И. Ленина «Три источника и три составляющие части марксизма», так и любая модель ISIS имеет правда не три, а пять составляющих этапов при создании. Мы очень часто будем проходить через них, поэтому настала пора знакомиться. Итак, я закончил тем, что нарисовал (именно нарисовал) графическую модель операционного усилителя в проекте, но если я даже захочу использовать ее для простой прорисовки схемы, то не смогу, она все еще состоит из отдельных элементов. Для того чтобы поиметь хоть какую-то пользу мне необходимо скомпоновать ее в единое целое и поместить в библиотеку, чтобы в нужный момент доставать оттуда. Чтобы создать модель служит функция **Make Device** в верхнем меню или в контекстном правой кнопкой мыши. Но прежде чем давить на нее необходимо выделить те графические компоненты, из которых будет создаваться модель. При выделении будьте внимательны: охвачено должно быть все, в том числе и маркер **ORIGIN**. В режиме **Selection mode** обводим, удерживая левую лапку мыши всю графику с небольшим запасом. Выделенные элементы примут красный цвет *(Рис. 17)*. Далее щелкаем по правой кнопке мышки и выбираем опцию **Make Device**.

После чего открывается первое из окон создания модели **Device Property** *(Свойства устройства, ну в данном случае правильнее перевести компонента) (Рис. 18)* В этом окне обязательными для заполнения являются два окна – **Device Name** (наименование компонента) и **Reference Prefix** (Префикс под которым устройство будет заноситься в список цепей). Я тут слегка погорячился ранее, заявив, что оригинальные библиотеки Протеуса защищены от записи, не учел «родной специфики», где используются взломанные программы. Поэтому, чтобы не испортить оригинал, давайте назовем наш девайс не просто **741**, а **741R**, по аналогии с ГОСТами России. Ну и префикс ему прилепим не стандартный для ISIS – **U**, а тоже в соответствии с нашими стандартами – **DA**. После чего считаем первое окно заполненным и переходим ко второму, нажав кнопку **Next**.

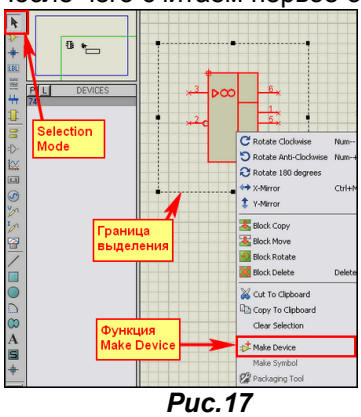


Рис.17

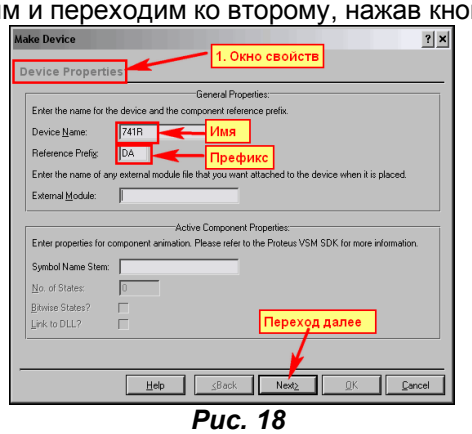


Рис. 18

Второе окно **Packaging** *(Рис. 19)* содержит информацию о назначенных нашему устройству корпусах. Конечно, можно бы назначить корпус и сейчас, но это никогда не поздно сделать, тем более что на этом примере я хочу показать назначение нестандартного корпуса и оставлю на потом. А пока идем в следующее окно.

Окно **Component Properties & Definition** *(Рис. 20)* скоро станет для нас одним из самых популярных, потому что именно в нем мы будем назначать характерные для того или иного компонента свойства, но на данном этапе оставим его пустым и перейдем к следующему.

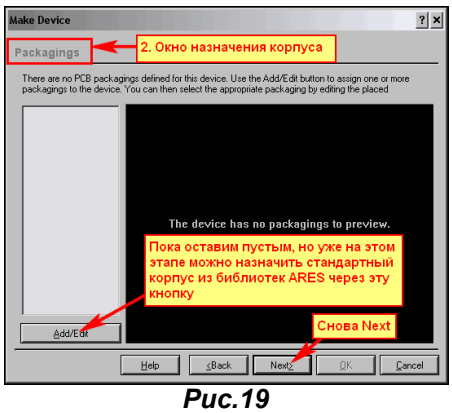


Рис.19

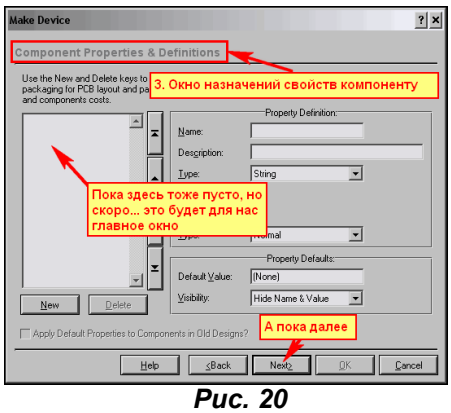


Рис. 20

Четвертое окно **Device Data Sheet & Help File** *(Рис. 21)* пожалуй, самое бесполезное для нас на ближайшее время, если конечно кто-то не собрался создавать модели на коммерческой основе с последующим распространением в качестве стандартных.

Ну и наконец, мы перешли к финальному окну создания модели, которое потребует от нас тоже некоторых завершающих «телодвижений». Во-первых, в строке **Device Category** через раскрывающееся меню я выбрал для устройства категорию **Operational Amplifiers** *(операционные усилители)*. Чуть ниже аналогичным способом субкатегорию **Single** *(одиночный).* Ну и в качестве **Device Manufacturer** *(производитель),* воспользовавшись кнопкой **New,** ввел нового производителя: **ExUSSR**, хотя и покривил душой (что-то не помню, чтоб в СССР производились именно 741, а не их «содранные» аналоги). Все эти сведения нужны Протеусу для того, чтобы поместить наш созданный девайс в определенный раздел библиотеки для быстрого поиска его в дальнейшем. Обращаю Ваше внимание, что все вновь созданные компоненты по умолчанию Протеус будет помещать в библиотеку **USERDVC,** и впоследствии мы их там найдем и удалим, как лишний и бесполезный мусор.

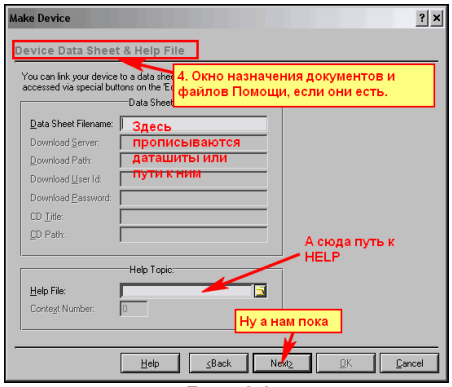


Рис.21

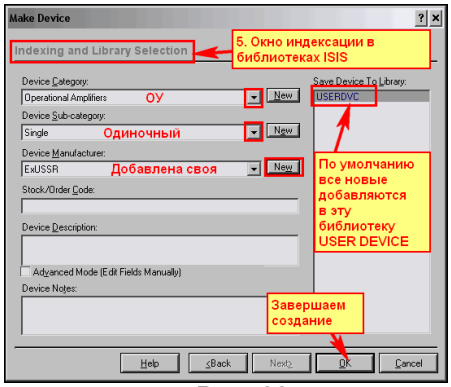


Рис. 22

Ну, вот собственно и вся процедура создания модели для данного этапа. После нажатия кнопки ОК на последней пятой вкладке мы помещаем наше творение в библиотеки ISIS.

4.3. Управление библиотеками Протеуса, или из простых «читателей» в «библиотекари».

Прошу обратить особое внимание! *Не зря я назвал этот раздел FAQ для «продвинутых». Со следующей информацией в этом параграфе надо обращаться осторожно, чтобы потом не клясть себя за кривые руки, а меня за то, что не предупреждал. Если вы повредите «родные» библиотеки программы, потом придется ее переустанавливать.*

Итак, мы создали модель 741R, пусть пока и бесполезную в работе, но необходимую, чтобы понять изложенный ниже материал. Если теперь войти в библиотеку компонентов и в окне поиска по ключевому слову набрать 741R – то вот нам и наша модель *(Рис. 23)*.

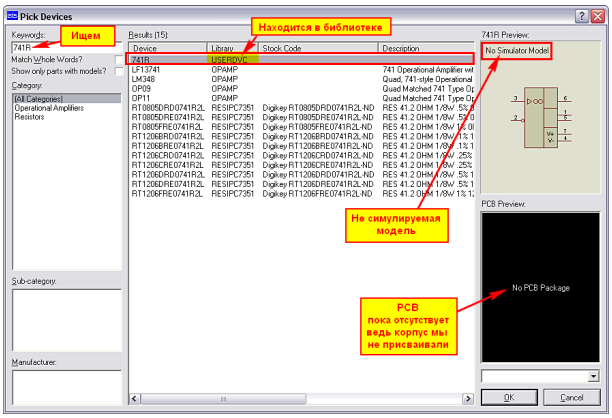


Рис.23

Как видим, мы получили то, что хотели – не симулируемую модель, для которой отсутствует корпус, и находится эта модель в библиотеке **USRDVC**, куда Протеус по умолчанию помещает все пользовательские модели. Теперь мы можем использовать ее в своих проектах в новом начертании, но работать она в симуляторе не будет и при попытке запустить симуляцию с ее участием, мы получим в логе сообщение об ошибке:

No model specified for DA1

Simulation FAILED due to partition analysis error(s)

Так что единственная польза от нашей модели в том, что можно использовать ее для рисования принципиальных схем. Давайте рассмотрим теперь - какие физические изменения внес Протеус при создании нашей модели, а для этого воспользуемся менеджером библиотек. Заходим в меню **Library** и выбираем **Library Manager…**

При первом открытии появится предупреждение ISIS о том, как размеры окна менеджера библиотек могут быть изменены и сохранены. Если кого-нибудь раздражают лишние советы – сразу поставьте галку **Don't display this massage again**. Для тех, кто не дружит с английским, поясню, что окно менеджера не содержит привычных кнопок развернуть/свернуть в верхнем правом углу. Поэтому, для изменения размеров необходимо растягивать его мышью за края. Чтобы сохранить измененные размеры надо, щелкнув правой кнопкой мыши в верхней полосе окна (в той, где находится заголовок) выбрать опцию **Save Window Size**. Итак, подтвердив кнопкой ОК, что мы ознакомились с сообщением, попадаем в окно менеджера (Рис. 24).

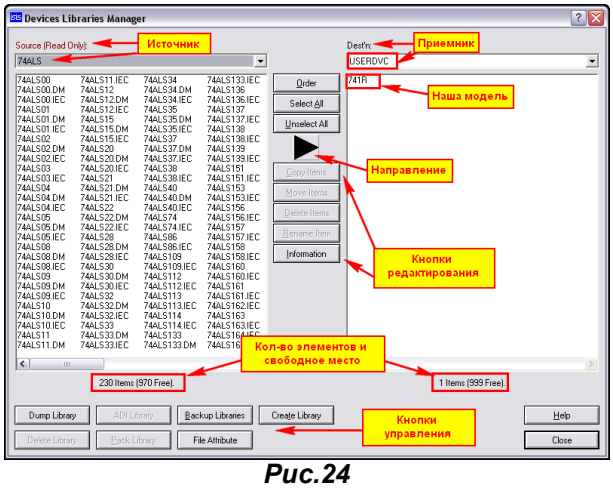


Рис.24

В левом окне менеджера по умолчанию открывается первая по алфавиту (а в данном случае начинающаяся с цифры) библиотека **74ALS**, а в правом наша библиотека **USRDVC** с сиротливо расположившимся в нем **741R**. Между двумя окнами расположены кнопки редактирования выбранных библиотек с большим черным треугольником-стрелкой, показывающим направление проводимых операций. Дополнительно **Source** (источник) и (**Dest'n**) приемник обозначены над окнами. Обратите внимание, что оригинальная **74ALS** обозначена как **Read Only** – только для чтения, о чем я и говорил раньше. Кроме того, внизу соответствующих окон отображается информация о количестве компонентов в библиотеке и наличии свободного места в ней. Например, в пользовательской **USRDVC** пока наш единственный ОУ и 999 свободных мест. Если выбрать элемент, – кликнуть мышкой по элементу в левой или правой библиотеке, то в зависимости от того, где вы щелкнули мышкой – направление и соответственно **Source/Dest'n** меняются местами. За этим тоже надо следить. Теперь познакомимся с назначением кнопок.

Кнопки редактирования между панелями:

- **Order** – порядок развертывания раскрывающегося списка библиотек при клике по треугольнику с направлением вниз. По умолчанию принят алфавитный от 0 до 9 и от A до Z. Нажатие этой кнопки вызывает окно, позволяющее изменить порядок перемещать одну выбранную библиотеку в списке, либо после нажатия **All** изменить порядок на обратный – **Reverce** или отсортировать по алфавиту – **Sort**.
- **Select All** и **Unselect All** – позволяют соответственно выделить или снять выделение со всех элементов в окне выбранной библиотеки.
- **Copy Items**, **Move Items** – соответственно копируют, перемещают выделенные элементы в направлении стрелки (из **Source** в **Dest'n**).
- **Delete Items** – удаляет выделенные элементы. Будьте осторожны с этой опцией!!! Обратный процесс невозможен.
- **Rename Item** – переименовывает выделенный элемент.
- **Information** – очень полезная кнопка, выводит информацию о выбранном элементе. Для примера на Рис. 25 показано выведенное окно для родного, Протеусного ОУ 741.

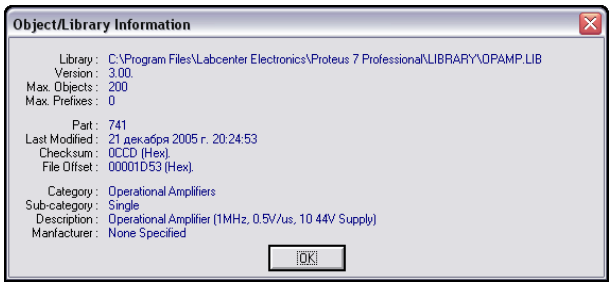


Рис.25

Теперь перейдем к рассмотрению кнопок управления библиотеками в нижней части:

- **Dump Library** – позволяет скопировать в буфер обмена или сохранить в текстовый файл информацию о выделенных элементах библиотеки. Если хотите сохранить информацию обо всех элементах, предварительно воспользуйтесь кнопкой **Select All**.
- **Delete Library** – удаляет выбранную библиотеку полностью. Также будьте внимательны с этой кнопкой, хотя в данном случае перед удалением Протеус вас предупредит, и вы можете отказаться от удаления.
- **ADI Library**, **Pack Library**, **Backup Library** – первая предлагает добавить информацию из файла **.ADI**, вторая создает для выбранной библиотеки файл **.TMP**, а третья - **.BAK**. Откровенно замечу, я и сам не знаю, для чего они нужны. Могу только предположить, что для «унутреннего потребления» сотрудниками Лабцентра, т.к. утилит, которые потом откроют эти файлы, я не знаю.

- **File Atribute** – устанавливает или снимает при повторном применении атрибут **Read Only** (только для чтения) для выбранной библиотеки.
- **Create Library** – ну вот и та опция, позволяющая создать собственную библиотеку, из-за которой и вклинен весь этот материал. После нажатия на эту кнопку открывается папка **Library** с предложением задать имя новой библиотеки. Зададим, например, произвольное имя **My_Lib**, после нажатия сохранить Протеус предложит еще одно окно с предложением задать количество элементов в этой библиотеке – по умолчанию 100, но вы можете изменить его по своему «вкусу». И уже после окончательного подтверждения будет создана новая библиотека.

Но естественный вопрос – а для чего она нужна? Ведь есть же для сохранения **USRDVC**. Да, есть и у каждого пользователя Протеуса. Создав свою библиотеку с оригинальным именем, вы получаете возможность сохранять вновь создаваемые элементы в ней *(Рис.26)*, ну или копировать/перемещать в нее элементы из **USRDVC**. Это позволяет передавать и переносить файлы библиотек на другой компьютер, не боясь при этом затереть файлы **USRDVC** на нем. Ведь у другого пользователя там могут храниться свои элементы. Физически это выглядит так:

При создании библиотеки **My_Lib** в папке **Library** Протеуса создаются два файла: **My_Lib.LIB** – непосредственно библиотека и индексный файл **My_Lib.IDX**. Вот они и подлежат переносу в соответствующую папку другой копии Протеуса. Кроме того, чуть позже при создании моделей мы будем создавать и файлы в папке **MODELS**, содержащие их описания для симулятора. Они также копируются в соответствующую папку другого компьютера для переноса моделей. Вот так возможна передача разработанных моделей другому пользователю.

Ну и в заключение данного материала хочу предостеречь наиболее рьяных пользователей от чрезмерной эйфории. Казалось бы, ну вот, теперь возьмем и начнем таскать библиотеки из копии в копию программы. Да, но только собственноручно созданные, а не «родные» Протеуса. Не забудьте, что большинство оригинальных библиотек программы обладают глубоко эшелонированной защитой. И простым копированием тут не обойдешься. Хотя с некоторыми моделями этот вариант и проходит. Так, автору этих строк в недалеком прошлом удалось перенести модель часиков DS1307 из демо-версии 7.3 в более раннюю, где она безнадежно глючила и тем самым благополучно закончить отладку незавершенного проекта. Но это частный случай, так что эксперименты – на свой страх и риск. И не забывайте делать резервные копии, а то…

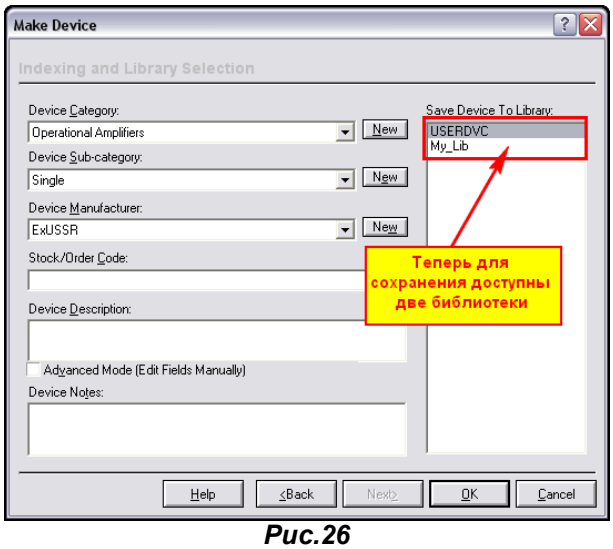


Рис.26

4.4. Netlist Compiler – список цепей. Основа для ARES и других приложений.

Настала пора рассмотреть – каким образом ISIS хранит информацию о нарисованной схеме и передает ее в ARES или другие сторонние приложения. Если кто-то считает, что в графическом – так как нарисовано, то глубоко заблуждается. Как и все другие CAD-ы Протеус для обмена информацией о схеме создает текстовый список цепей. К сожалению, единого общепризнанного стандарта на такие списки не существует *(у P-CAD свой, OrCAD - тоже, даже у столь популярного Eagle свой формат)*, и поэтому специалисты Лабцентра разработали свой собственный. Информация о разработанной схеме помещается в файл с расширением **.SDF** – **Schematic Description Format** *(формат описания схемы)* с помощью встроенного в ISIS компилятора цепей, ну или соединений в другой транскрипции перевода. Формат достаточно компактный и после небольшого навыка легко «расшифровываемый», что весьма пригодится нам в дальнейшем. Для начала «соберем» небольшую схемку *(я использовал типичный мультивибратор на таймере 555 Рис. 27)*.

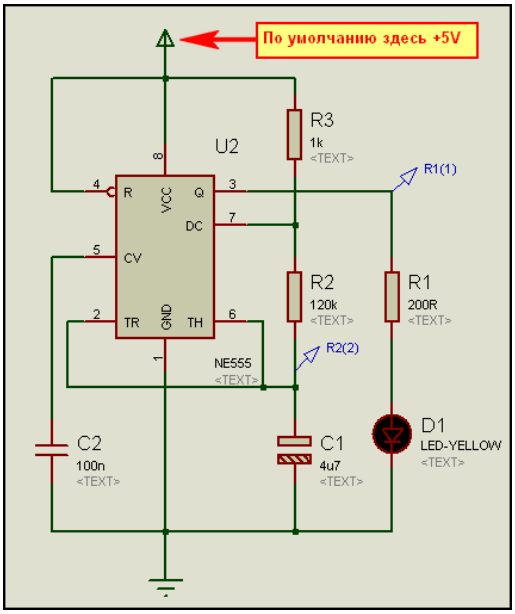


Рис.27

Все примеры будут во вложении к данной теме – это пример **Ex1**. Размещенный по умолчанию терминал питания принимает значение +5V по отношению к земле. Все элементы схемы, которые я применил здесь, имеют **PCB** (т.е. корпуса) за исключением светодиода. Для того чтобы назначить корпус ему воспользуемся следующим способом.

Двойным кликом по светодиоду или через правую кнопку мыши (**Edit Properties**) входим в окно **Edit Component**. В строке **PCB Package** *(шестая сверху для «любителей» русифицированных версий)* пока у нас стоит значение **(Not Specified)** т.е. не назначено. Справа от этой строчки щелкаем по маленькой кнопке со знаком вопроса и попадаем в библиотеку корпусов **ARES** (Рис. 28). Там мы ведем себя абсолютно так же, как и в библиотеках ISIS – вводим ключевое слово **LED** для поиска. Подходящий корпус только один – его и назначаем нашему светодиоду. Обратите внимание, что данное назначение действует только в пределах данного проекта. Если вы начнете новый проект в ISIS и добавите тот же **LED-YELLOW** из библиотеки, он опять будет иметь значение **(Not Specified)**.

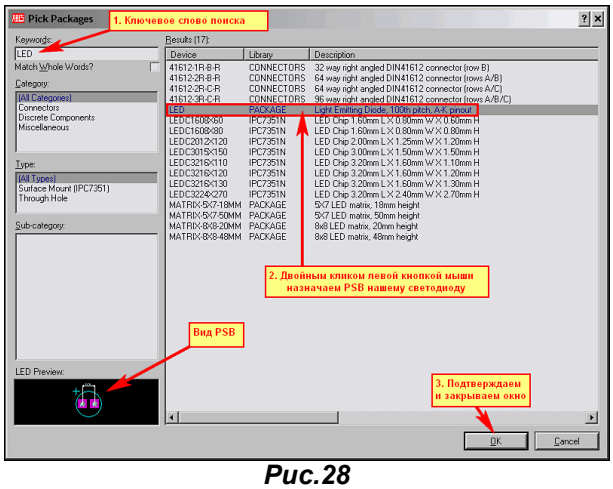


Рис.28

Я старался использовать все элементы с PCB, поскольку в этом и следующем параграфах нам так или иначе придется пересечься с разводкой платы в ARES для понимания материала. Итак, все компоненты у нас с PCB, хотя для списка цепей это и не требуется, давайте скомпилируем его. В верхнем меню выбираем **Tools => Netlist Compiler** и попадаем в окно выбора опций компиляции *(Рис. 29)*.

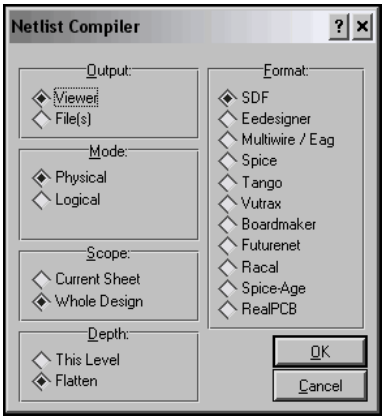


Рис.29

Пока оставляем все переключатели в нем как есть, чтобы сформировался стандартный **SDF** Протеуса нажимаем **OK**, после чего открывается окно просмотра скомпилированного **SDF** файла (Рис. 30). Из этого окна мы можем скопировать текст в буфер обмена - кнопка **Clipboard**, сохранить в стандартный ASCII текстовый файл – кнопка **Save As**, либо просто наплевать на все и закрыть окно **Close**. Вот именно такой файл каждый раз формируется автоматически, когда мы передаем информацию о «нарисованной» нами схеме в **ARES** для создания печатной платы.

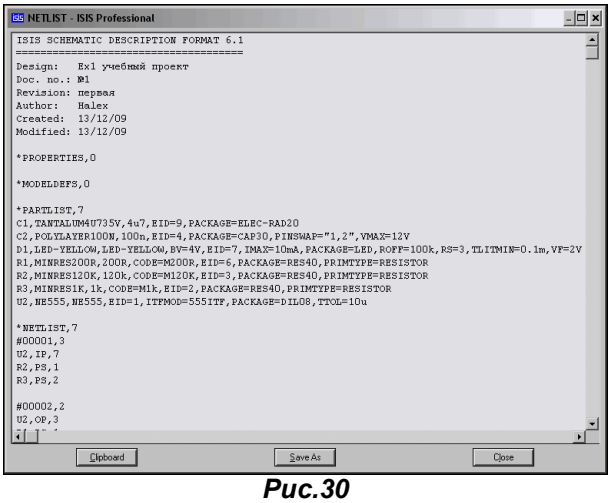


Рис.30

Рассмотрим структуру этого файла. Я приведу его полностью, а не так, как на рисунке ниже. В начале файла следует чисто описательная информация, заканчивающаяся датой создания **Created** и датой последней модификации **Modified**. Все, что выше берется из окна, вызываемого через меню Design => Edit Design Properties, если вы его, конечно, заполнили, как например я в данном случае. Иначе здесь кроме дат, вставляемых автоматом, информации не будет. Далее следует несколько разделов, начинающихся со знака звездочки - *. Разделы ***PROPERTIES** *(свойства)* и ***MODELDEFS** *(назначения для моделей)* пока пусты. Они получают информацию из одноименных текстовых скриптов, размещенных в проекте. Позже, при создании MDF-моделей мы познакомимся с ними поближе.

***PARTLIST** – список составляющих. В этом списке перечислены все компоненты, входящие в нашу схему с указанием их свойств. Каждая строка начинается с позиционного обозначения компонента, далее через запятые следует информация об его номиналах, корпусе, применяемой модели и т.п. Думаю, что не обязательно быть сотрудником спецслужб, чтобы без особого труда «расшифровать» эту информацию. Достаточно заглянуть в свойства любого из этих компонентов, чтобы обнаружить ее там.

***NETLIST,7** – ну вот это и есть собственно список цепей или соединений. В данном случае цифра **7** после запятой указывает на количество узлов (цепей) в списке. Каждая цепь начинается со знака решетки - #. Далее следует номер узла этой цепи в списке и количество подходящих к узлу проводов (подключенных выводов элементов). Ну и далее сам список подключенных выводов. Давайте для примера разберем первую цепь **#00001,3**. Число 3 указывает, что к узлу подключены три вывода компонентов, которые перечислены ниже. Это:

U2,IP,7 – расшифровывается как – микросхема **U2** – таймер555, вывод микросхемы номер **7**, тип вывода **IP** *(Input – про типы мы уже проходили при создании графической модели)*;

R2,PS,1 – **R2** –резистор, вывод **1** *(в данном случае у модели резистора выводы обозначены просто как 1 и 2, впрочем, как и у большинства моделей двухвыводных компонентов: резисторов, конденсаторов, катушек и т.п.)*, тип вывода **PS** *(пассивный).*

```
ISIS SCHEMATIC DESCRIPTION FORMAT 6.1
=====================================
Design:   Ex1 учебный проект
Doc. no.: №1
Revision: первая
Author:   Halex
Created:  13/12/09
Modified: 13/12/09

*PROPERTIES,0

*MODELDEFS,0

*PARTLIST,7
C1,TANTALUM4U735V,4u7,EID=9,PACKAGE=ELEC-RAD20
C2,POLYLAYER100N,100n,EID=4,PACKAGE=CAP30,PINSWAP="1,2",VMAX=12V
D1,LED-YELLOW,LED-YELLOW,BV=4V,EID=7,IMAX=10mA,PACKAGE=LED,ROFF=100k,RS=3,TLITMIN=0.1m,VF=2V
R1,MINRES200R,200R,CODE=M200R,EID=6,PACKAGE=RES40,PRIMTYPE=RESISTOR
R2,MINRES120K,120k,CODE=M120K,EID=3,PACKAGE=RES40,PRIMTYPE=RESISTOR
R3,MINRES1K,1k,CODE=M1k,EID=2,PACKAGE=RES40,PRIMTYPE=RESISTOR
U2,NE555,NE555,EID=1,ITFMOD=555ITF,PACKAGE=DIL08,TTOL=10u

*NETLIST,7
#00001,3
U2,IP,7
R2,PS,1
R3,PS,2

#00002,2
U2,OP,3
R1,PS,1

#00004,4
U2,IP,2
U2,IP,6
C1,PS,+
R2,PS,2

#00005,2
U2,IP,5
C2,PS,1

#00006,2
R1,PS,2
D1,PS,A

GND,5,CLASS=POWER
GND,PR
U2,PP,1
C2,PS,2
D1,PS,K
C1,PS,-

VCC/VDD,5,CLASS=POWER
VCC,PT
VCC/VDD,PR
U2,IP,4
U2,PP,8
R3,PS,1

*GATES,0
```

Как видите весьма простой и доходчивый способ записи списка. И при небольшом навыке его можно запросто читать, как букварь. Я же в данный момент хочу заострить ваше внимание на двух последних узлах, которые начинаются отнюдь не с решетки и имеют в своем описании типа **CLASS=POWER**. Это и есть наши цепи питания **GND** и **VCC/VDD**. В данном случае это терминалы питания, присвоенные проекту по умолчанию, и описанные в разделе меню **Design => Configure Power Rails**.

Следующий за ними раздел ***GATES** мы рассмотрим позднее, а пока сосредоточим свое внимание на шинах питания. Я в том же проекте *(Рис. 27)* присвою нашему верхнему терминалу питания лэйбл **+15V**. На Рис. 31 представлены два аналоговых графика для разных значений. Посмотрите внимательно на амплитуды сигналов таймера.

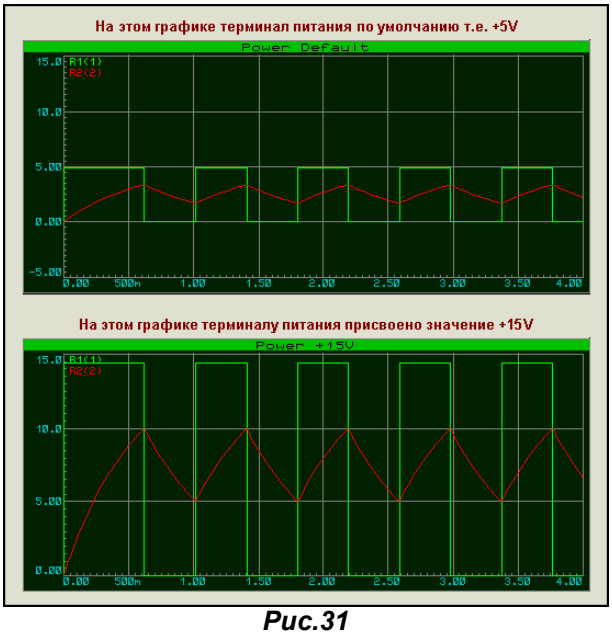


Рис.31

Это называется как в известной комедии - «легким движением руки». Но и это еще не все. Сформируем опять Netlist с помощью компилятора. Ниже приведено только окончание нового листа.

```
#00006,2
R1,PS,2
D1,PS,A

+15V,4,CLASS=POWER
+15V,PR
U2,IP,4
U2,PP,8
R3,PS,1

GND,5,CLASS=POWER
GND,PR
U2,PP,1
C2,PS,2
D1,PS,K
C1,PS,-

*GATES,0
```

Обратите внимание, у нас напрочь исчез узел **VCC/VDD**, зато появился аналогичный **+15V**. Наиболее сообразительные уже догадались, что именно он и будет передан через SDF в ARES для обработки. Для тех, кто еще не понял к чему я веду, – усложняем задачу (пример **Ex2**). В нем я вернул таймеру питание **VCC/VDD**, подключил светодиод через транзисторный ключ и запитал его от терминала **+15V** *(Рис. 32)*. Кроме того, я ввел в схему три единичных терминала из библиотеки **Connectors** ISIS, имеющих свои PSB и подключил их к соответствующим питающим терминалам: **GND**, **VCC/VDD** *(без обозначения)* и **+15V**. Они мне потребуются в ARES.
Снова сформировал список цепей и получил в конце его следующее:

```
+15V,3,CLASS=POWER
+15V,PR
J2,PS,1
R1,PS,1

GND,6,CLASS=POWER
GND,PR
J3,PS,1
U2,PP,1
C2,PS,2
C1,PS,-
Q1,PS,3

VCC/VDD,6,CLASS=POWER
VCC,PT
VCC/VDD,PR
J1,PS,1
U2,IP,4
U2,PP,8
R3,PS,1
```

Обратите внимание, что узлов питания в SDF стало три! И три отдельных цепи питания будут переданы в ARES для обработки. Соответственно они будут и разведены как три отдельные цепи на плате. Так можно продолжать и далее до бесконечности. Просто хочу отметить, что я не «рисую» цепи питания, а просто нахально добавляю их в схему с нужным мне вольтажом. Но я оставлю дальнейшие рассуждения на следующий параграф, а здесь необходимо закончить с **Netlist**, чтобы больше к нему не возвращаться.

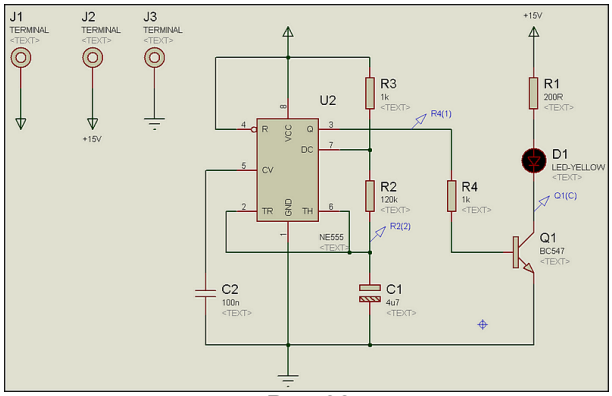


Рис.32

Вернемся к рисунку 29 параметров **Netlist Compiler…** До сих пор мы там ничего не трогали, чтобы формировался стандартный SDF, но давайте посмотрим – что там еще можно изменить и для чего.

В левой части окна расположены четыре переключателя.

- **Output** – вывод. По умолчанию стоит **Viewier** *(просмотрщик)*, поэтому и открывается окно. Если поставить в **File(s)** будет сразу формироваться файл без окна просмотра.
- **Mode** – режим. По умолчанию стоит **Phisical** *(физический)*, т.е. информация в списке содержит как имя вывода компонента, так и его номер. Этот режим считается обязательным для ARES и в большинстве случаев передачи в сторонние программы. Если поставить **Logical**, то информация о номере вывода формироваться не будет.
- **Scope** – область видимости. По умолчанию **Whole Design** – весь проект. Но иногда надо сформировать список только для текущего листа многолистового проекта, тогда выбирается опция **Current Sheet**.
- **Depth** – глубина захвата. По умолчанию и наиболее подходящая в большинстве случаев Flatten *(сглаженный)*. В этом случае объекты, содержащие дочерние листы будут заменены в списках их содержанием. Если выбрать **This Level**, то информация о содержании дочерних листов в списках соединений не появится.

В правой части окна расположен переключатель режима компилятора. По умолчанию формируется стандартный для Протеуса формат SDF-файла. Однако разработчики Лабцентра позаботились о том, чтобы информацию можно было передавать и в сторонние программы. Вопрос о том, насколько совместимыми окажутся форматы компилируемых файлов, я оставлю для самостоятельного исследования наиболее дотошным пользователям, а здесь только перечислю режимы, которые заложены в переключателе **Format**.

EEDESIGNER – создается файл формата EE Designer III. Информация о корпусах как в Boardmaker. При компиляции используется **Phisical** режим.

MULTIWIRE/EAG – создается файл формата Multiwire, также используется для передачи в небезызвестный Eagle. Поклонники «клювастого» могут поэкспериментировать. При компиляции используется **Phisical** режим.

SPICE – формат говорит сам за себя. Также может быть использован для передачи списка в PSpice. При необходимости выходной файл формата **.LXB** переименовывается в **.LIB**. При компиляции используется **Logical** режим. Для передачи в более старые версии Spice под DOS используется формат **SPICE AGE**, расположенный чуть ниже в списке форматов.

TANGO – формат используется для передачи например в Protel. При компиляции используется **Phisical** режим.

Ну и остальной ряд форматов для передачи в одноименные программы, о некоторых из которых я и сам ничего толком не могу сказать. Это Vutrax, Futurenet, RACAL и т.д. Я надеюсь, что те, кому понадобятся такие форматы, самостоятельно разберется, прочитав HELP. Я же на этом заканчиваю рассмотрение компилятора списка цепей и плавно перехожу в **Configure Power Rails**, как продолжение данной темы с развитием примеров приложенных к этой теме.

4.5. Configure Power Rails или питания «видимо, не видимо».

Итак, чуть выше мы разобрали, что терминалов питания можно навешать сколько угодно всяких и разных. И задумываться про это особенно не стоит. В первой части FAQ я уже упоминал про окно **Configure Power Rails** *(конфигурация шин питания)*, которое вызывается через верхнее меню **Design**, а заодно и разберемся со всеми видами электропитания в ISIS раз и навсегда. Как я уже отмечал, при создании нового проекта Протеус по умолчанию вводит три шины питания: **VCC/VDD**=+5V, **GND**=0V и **VEE**=-5V. Это при условии, что стоит флажок в соответствующем окне внизу (Рис. 33).

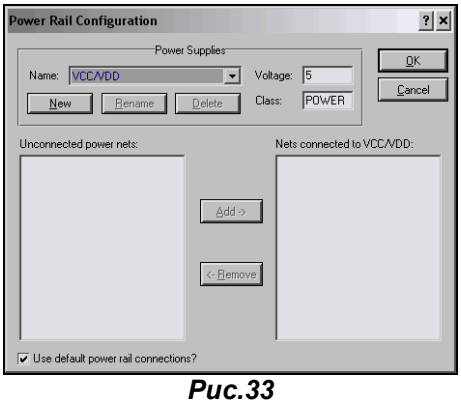


Рис.33

При этом первые две шины не надо даже именовать. Достаточно поставить в схеме терминал питания **POWER** из левого меню или терминал земли **GND** и присоединить к ним соответствующие выводы компонентов в проекте, и они автоматически присоединяются к соответствующим шинам питания. Это для тех компонентов, у которых выводы питания есть. А как быть с теми, у которых они скрыты? Почему-то наши пытливые умы упорно желают пририсовать видимые провода к этим выводам. Кроме как чисто физического удовлетворения от вождения мышью по ковру, по моему личному мнению здесь никакой пользы нет. Но специально для Вас – любой каприз. Давайте сначала обозначимся к чему можно «прорисовать» провода, а к чему нет. Все очень просто воспользуемся опцией меню **Template => Set Design Defaults…** и поставим в ней галочку **Show Hidden Pins?** *(показать скрытые выводы)*. Я это уже описывал в начале FAQ, но приходится повторяться. Теперь, с установленной галкой можно зайти в библиотеку компонентов ISIS и пошариться там. У всех компонентов, которые имеют скрытые выводы питания, они проявятся серым цветом в окне предпросмотра (Рис 34).

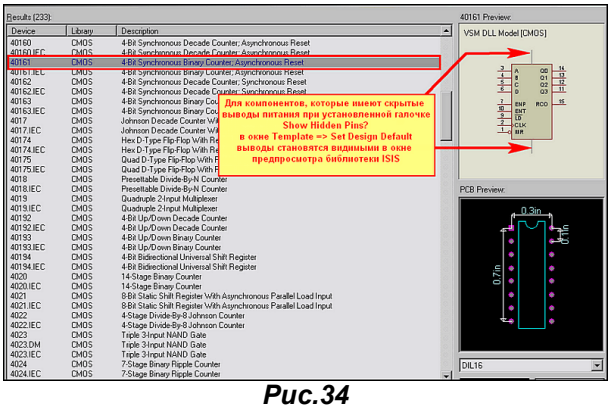


Рис.34

Таким образом, можно сразу определить – можно сделать их видимыми и активными или нет. А точнее сказать видимыми мы уже их сделали, но вот если поместить такие компоненты в проект, то подвести провода к этим серым выводам мы все-таки не можем, они к ним «не припаиваются». Этому горю легко помочь, вот только как бы эта услуга не оказалась «медвежьей».

Для начала три не очень «лирических» замечания.

Замечание первое. Это замечание касается микроконтроллеров AVR. Заходим в HELP по этим микроконтроллерам, заглядываем в раздел General Model Limitations (основные ограничения модели) и видим там фразу: **Power supply voltage changing is not supported**. Для англоязычной публики вопрос отпал сразу, для остальных - корявый машинный перевод: «Изменение напряжения источника питания не поддержано».

Замечание второе. Это замечание касается многоэлементных логических микросхем, столь популярных у начинающих: типа двухвходовых И, инверторов и т.п. Можете сразу пробежаться в библиотеках с включенным флажком **Show Hidden Pins** и убедиться, что у них нет ни видимых, ни невидимых ног питания (как у удава в мультике). Это дело поправимое, но об этом чуть позже и совсем не тем способом, который Вы ожидали лицезреть.

Замечание третье. Пожалуй, самое серьезное. Все, что я писал красным цветом в предупреждении к материалу о менеджере библиотек остается в силе и здесь. Сейчас мы начнем библиотеки «шерстить и в хвост и в гриву», поэтому позаботьтесь заранее о защите от записи оригинальных, если не хотите все испортить. Особенно это касается владельцев нелицензионных копий программы. О том, как поставить защиту писалось в материале двумя постами выше. Кто позабыл – бегом туда. За Ваши «кривые ручки» я ответственности не несу. Связано это с тем, что нам придется мэйкать (**Make Device**) модели под их оригинальными именами и желательно, чтобы дубли не заменили «родные» модели.

Ну а теперь перейдем конкретно к «оживлению» выводов питания. Берем навскидку любую подходящую для этих целей модель. Мне приглянулся счетчик **74HC161**. Тут я отстреливаю сразу

двух ушастых, поскольку это схематичная **MDF** модель, а именно из-за них и придется сохранять дубли с тем же именем. Обвешал я его тестовыми примочками из библиотеки ISIS **Debugging Tools**, прилепил тактовый генератор, и получилось то, что вы видите на Рис. 35 и в прилагаемом примере **Ex1_Pow**. Т.е. пока модель работает.

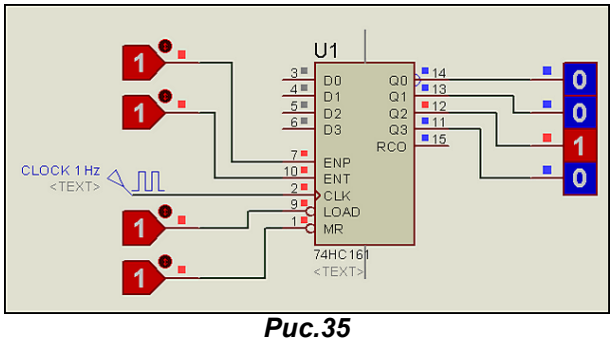


Рис.35

Затем, рядом я поставил точно такую же микросхему и применил к ней опцию **Decompose**: или через верхнее меню (кнопка с молотком), или через правую кнопку мыши. У разобранной на запчасти модели для верхней и нижней ног питания я зашел в свойства и поставил флажки - как на (*Рис. 36)*. Если кому-то хочется, чтобы подсвечивалось еще и имя – можете включить и третий флажок, не возбраняется.

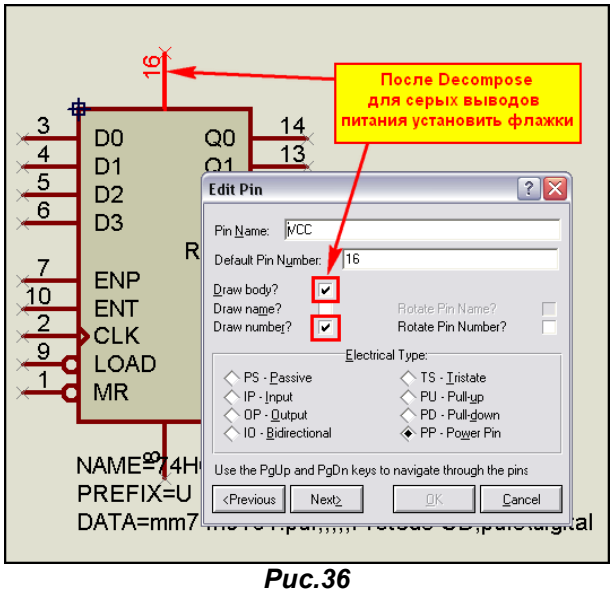


Рис.36

После этого с помощью левой кнопки мыши выделил область, обязательно (!!!) захватив текстовый скрипт *(Рис.37)*, и применил к ней функцию **Make Device** так, как мы делали при создании графической модели.

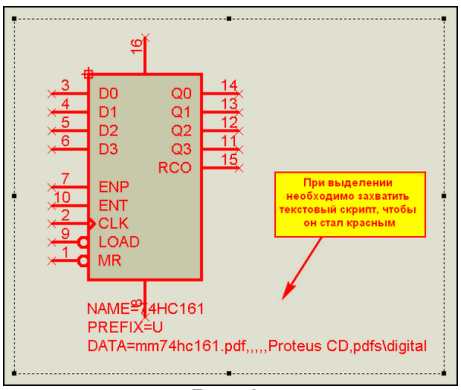


Рис.37

А теперь самый ответственный момент – все пять окон мы проходим, ничего не изменяя!!! Даже название в первом окне необходимо оставить то же. В последнем окне перед нажатием **ОК** необходимо убедиться, что наша модель будет сохраняться в **USRDVC**, ну или созданной Вами личной библиотеке. Вот собственно и вся процедура. Теперь у Вас в библиотеке ISIS будет две модели – одна в родной библиотеке **74HC**, а другая в **USRDVC** – вот она то и окажется с активированными выводами питания *(Рис. 38)*. Кстати, в селектор после **Make…** она сразу подставится автоматом. Ее я применил в примере **Ex2_Power**. Для тестирования приложен аналоговый график, чтобы увидеть уровни сигнала по напряжению.

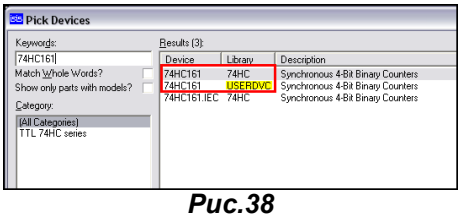


Рис.38

На Рис. 39 два графика для двух микросхем из **Ex2** при разных уровнях питания и разных частотах тактовой **Clock**. На графиках видно, что уровни по напряжению разные, что и требовалось доказать этим примером. Теперь объясню, почему потребовалось не изменять имя модели. Дело в том, что для большинства схематичных моделей часть свойств прописана в текстовых скриптах, находящихся в MDF-файлах, а там четко прописано имя модели. Вспомните предыдущий материал – скрипт ***PROPERTIES** – речь о нем. Если при создании модели мы изменим хотя бы букву в имени, то скрипт работать не будет, и мы получим ошибку при симуляции. Позже, при создании собственных **MDF**, мы к этому еще вернемся.

Рис.39

Теперь давайте заглянем в **Configure Power Rails** в тех проектах, где мы добавляли свое питание с помощью терминалов прямо на схеме, например в последнем **Ex2_Power**. Да ведь наши **+3V** сами там прописались в раскрывающемся списке. Вопрос – а зачем тогда нужно это окно? Ну, во-первых, если Вы делаете простой проект с одиночным питанием, то можно заранее изменить значение вольтажа например, для **VCC/VDD** на нужное и потом «не париться» с указанием вольтажа у терминалов питания, а просто ставить без наименования их в схему. Во-вторых, когда терминалов наставлена туча, да еще на нескольких листах то проще менять питание здесь, чем метаться по всем листам, разыскивая нужные терминалы. Ведь можно для терминалов в схеме назначить не конкретное напряжение, а буквенную аббревиатуру, например: **HV** (я сократил фразу Higth Voltage), а в окне **Configure Power Rails** для **HV** присвоить конкретное значение, допустим **+300V**. Теперь, для того чтобы изменить значение на **+330V** достаточно войти в окно и **Configure Power Rails** заменить значение. Все терминалы, с именем **HV** соответственно тоже поменяют значение напряжения.

Ну и в заключение данного материала хочется отметить, что на все, что здесь говорилось о визуализации можно «наплевать и забыть» в хорошем смысле этой фразы. Кроме морального удовлетворения от «изнасилования» моделей я не вижу в этом никакой пользы. Ну, увидели мы выводы питания на схеме – а «оно нам надо»? Обратимся к примеру **Ex3_Power**. В нем я вернул оригинальную модель счетчика, а на выход **Q0** прилепил инвертор **4049**. Счетчику я вообще все сместил и землю, и плюс питания, а инвертеру только плюс питания. Заметьте, инвертору, у которого нет видимых ног питания!!!

Теперь переходим к самой процедуре на примере инвертора. Входим в его свойства **Edit Properties** и нажимаем справа кнопку **Hidden Pins**. В открывшемся окне вводим для **Pin VDD** текст так, как на Рис. 40 и давим **OK**. Затем заходим в окно **Configure Power Rails** и в нем проделываем операции в соответствии с цифрами на Рис. 41.

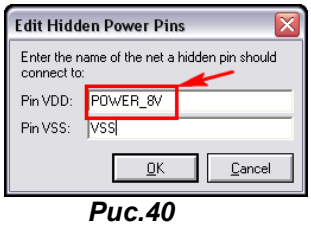


Рис.40

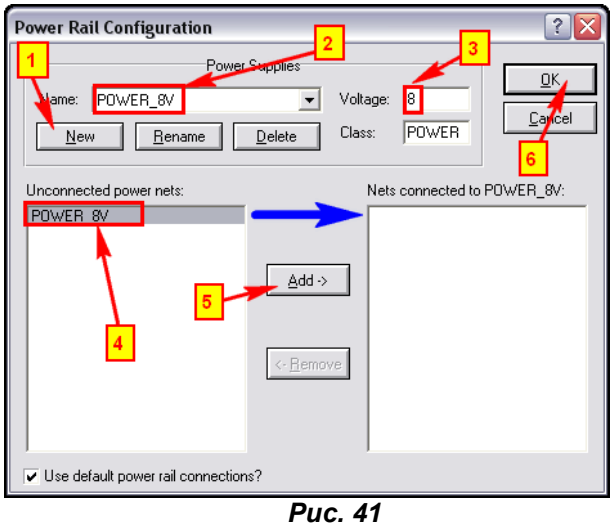


Рис. 41

Аналогичным образом я поступил и со счетчиком, только у него еще и для **GND** назначил **Power_3V**. После таких процедур внизу у моделей подсвечиваются внесенные изменения для питания, а на графиках видно, что изменения функционируют в полном объеме *(Рис. 42)*. При этом я не трогал сами модели и добился нужного результата. Ну, вот теперь, кажется, разжевал все до мелочей – глотайте. Все, о чем здесь говорилось, касается источников питания постоянного тока. О питании переменного тока поговорим ниже.

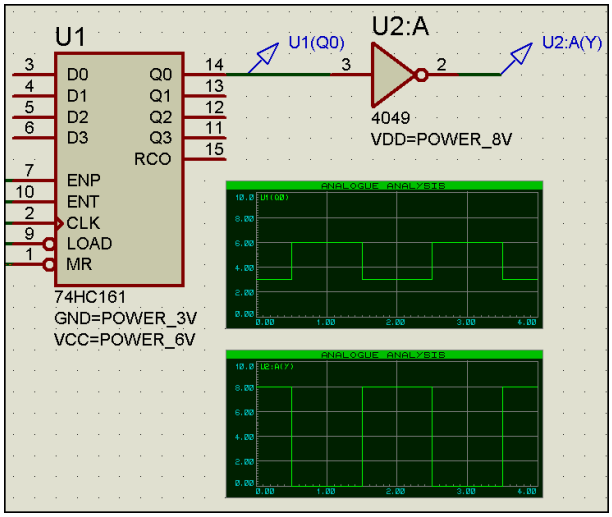


Рис.42

4.6. Источники переменного тока в ISIS. И опять об анимации.

Рассмотрение моделей, которые можно использовать в Протеусе в качестве источников переменного тока я хочу начать с типового примера, об который спотыкаются все начинающие (Рис. 43). Возьмем пару генераторов **SINE** из левого меню **Generator Mode** и анимированные модели Lamp из библиотеки **Optoelectronics => Lamps**.

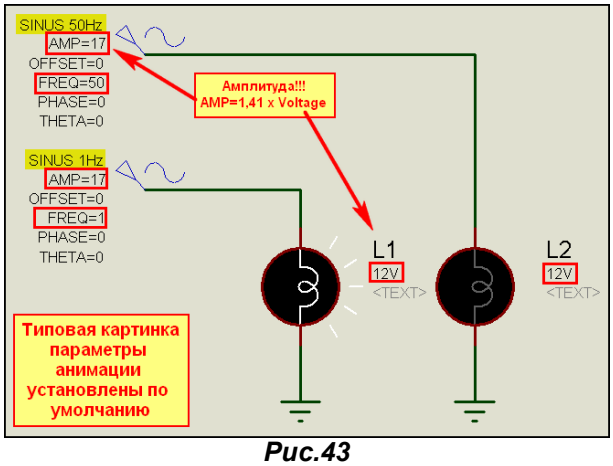


Рис.43

Я не стал сразу задирать напряжение до сетевого, но для поклонников сети 220V хочу сразу напомнить, что амплитуда напряжения сети составляет 310V. Почему то многие про это напрочь забывают при моделировании сетевых устройств. Именно поэтому амплитуда генераторов в учебном проекте выставлена не 12, а 17V *(в корень из 2 больше номинала для ламп)*. Зададим нашим генераторам частоты 1Гц и 50Гц и запустим симуляцию *(пример AC_1/Anim_Default.DSN из вложения)*. Ну, картинка обычная – при 50 Гц лампочка светиться отказывается. Ура!!! Даешь глюк!!! Вопрос только чей? А не вспомнить ли нам **п.3.4** этого FAQ. Заглянем в параметры анимации. А они у нас стоят по умолчанию, и куда не глянь – то 50 то 25. А частота то у нас тоже 50. Никаких светлых мыслей не навевает? А я возьму и поправлю **Timestep per Frame** всего лишь на **1** и поставлю **49** *(пример AC_1/Anim_49.DSN)*. Запускаем – моргает, однако. Ставим **Timestep** 25 *(Anim_25.DSN)* – тоже моргает, а при 20 *(Anim_20.DSN)* – опять перестала. Так что вовсе это не глюк, а просто по нашему недосмотру параметры анимации попадают в «резонанс» с нашим генератором 50 Гц и мы застаем все время нашу лампочку в неактивном (погашенном) состоянии. Этот фактор приходится всегда учитывать, если вы хотите в реальном времени наблюдать симуляцию с источниками переменного напряжения, да и с импульсами, впрочем, то же самое.

В этих примерах использованы модели генераторов синуса, у которых один вывод всегда соединен с терминалом **GND**. Но в ISIS есть и модели двухполюсников переменного напряжения, расположенные в библиотеке **Simulator Primitives => Sources**. Это **Alternator** – анимированный источник переменного напряжения, **VSINE** – двухполюсный генератор переменного напряжения (аналогичный тому, что мы использовали, но с двумя выводами), **ISINE** – генератор переменного тока и наконец, для любителей трехфазников – **V3PASE** – генератор трехфазного напряжения. На Рис. 44 приведен пример подключения альтернатора. Обратите внимание – амплитуда 17V, а вольтметр показывает 12. Это все к тому же корню из 2.

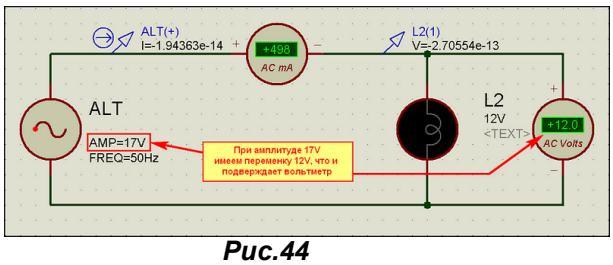


Рис.44

Примеры с этими генераторами собраны во вложении *(AC_Sources.DSN)*. Здесь я хотел бы только подчеркнуть несколько особенностей.

Ну, во-первых: для тех, кто пользуется первыми версиями FAQ, сразу приношу извинения за мою ошибку с трехфазником. Все там прекрасно работает, я сам тогда попался на параметрах анимации. Еще одно замечание, касающееся параметров трехфазника для начинающих. Обратите внимание на **Amplitude Mode** в его **Properties**. Там можно выбрать режим, который относится к параметру **Amplitude** (Volts): **Peak** – пиковый – разах амплитуды, **Peak to Peak** – удвоенный размах и **RMS** – среднее значение (фактически показания вашего вольтметра). Еще внизу окна **Properties** есть раскрывающийся список Advanced Properties с параметрами, относящимися непосредственно к генерации трехфазной сети. В том числе там стоит и **Insulation to earth** (сопротивление изоляции к земле) 2 МОм, поэтому то, о чем пойдет речь ниже здесь не так актуально.

А следующее замечание касается как раз использования измерительных приборов и зондов с двухполюсными источниками напряжений. Если вольтметры, амперметры, и токовые зонды как видно из рис. 44 проблем не вызывают, то иначе обстоит дело с зондами напряжения и, например, осциллографом. Для примера я приведу график *(Рис. 45)* из *AC_Sources.DSN*, на котором желтая трасса соответствует генератору, у которого один из выводов заземлен на терминал **GND**, а красная - с отсутствием связи на землю. Параметры генераторов одинаковы.

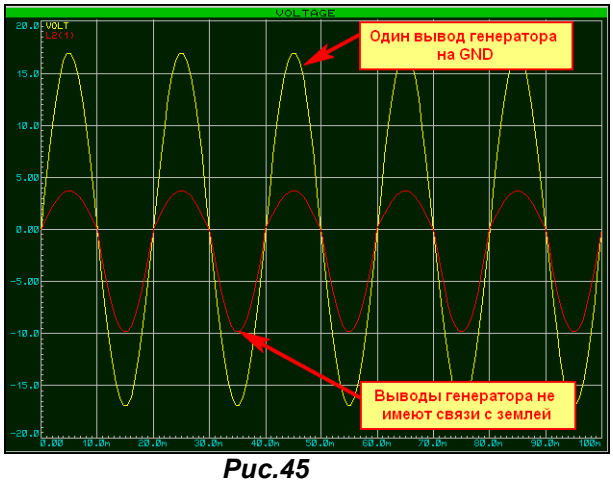


Рис.45

Надеюсь, из графика понятно – что Вас ожидает, если вы будете использовать виртуальный осциллограф для контроля источника, не имеющего связи с землей. И не надо при этом всенародно утверждать о глючности программы. В первую очередь всегда необходимо проверить – а все ли я сам сделал правильно. Вот на этой поучительной ноте я, пожалуй, и закончу рассмотрение источников переменного напряжения в программе ISIS.

4.7. Возвращаемся в SPICE моделирование. Начинаем знакомство с аналоговыми примитивами. Нелинейные управляемые источники сигналов.

Я все чаще начинаю приводить графики вместо интерактивного моделирования больше потому, что обычно это единственный метод исследования поведения аналоговых схем, особенно при использовании аналоговых генераторов с высокой частотой *(см. п.3.1 «зеленое выделение» этой части FAQ)*, да и в большинстве случаев цифрового моделирования можно обойтись ими. Это не означает, что я такой противник моделирования в реальном времени, просто старая добрая привычка: «Если на клетке слона прочтёшь надпись «буйвол», не верь глазам своим», а в интерактивном моделировании реального времени - это сплошь и рядом. Мы рассмотрим использование различных типов графиков в процессе моделирования аналоговых компонентов. И сейчас займемся этим вплотную.

Надеюсь, не стоит напоминать, что основным инструментом исследования аналоговых устройств в Протеусе является **ProSPICE**. Поскольку он базируется на **SPICE3F5** в отличие от **PSPICE**, принятого в **OrCAD**, **PCAD**, **Multisim** и т.п., имеет смысл остановиться на Протеусном варианте, и несколько расставить точки над и. Ну, в общем, особо приятного тут мало. Все дело в том, что **PSPICE** в некоторых отношениях развился дальше и от классического **SPICE** ушел намного вперед. В частности последнее время все чаще попадаются наши доблестные исследователи поведения индуктивных элементов (даешь трансформатор!!!) с сердечниками из ферромагнетиков. К сожалению, классический **SPICE** и **SPICE3F5** здесь мало помогут, в то время как фирма **Cadence** (основоположник **OrCAD** и «поглотитель» прародителя **PSPICE** – фирмы **MicroSim**) разработала свои продвижения по этой теме. Но, поскольку Лабцентр сделал основную ставку на имитацию микроконтроллеров – и в этом его главный конек, а поддержка аналоговых устройств является несколько второстепенной, то налицо некоторое несоответствие в этой области. Однако это не значит, что аналоговое моделирование в **ISIS** совсем «в загоне». Большинство из принятых в том же **OrCAD PSPICE** методов моделирования доступно и здесь и это будет доказано следующим материалом. Характерной особенностью **ISIS** в этом плане является то, что моделирование без наличия графической модели невозможно. Если в классическом **SPICE**, и **PSPICE** в частности, достаточно текстового описания моделируемой схемы, то здесь обязателен графический проект. В других пакетах такой вариант представлен как «advanced», т.е. дополнительный – например, в том же **OrCAD** для этого служит **Capture**, а в Протеусе этот вариант является основным. Именно поэтому первое, что я рассмотрел - было создание графической модели компонента. Но, от хотя бы первичного знакомства со **SPICE** (и его модификации **3F5**) нам и здесь никуда не уйти. К сожалению, у меня сейчас нет времени на перевод с английского руководства по **SPICE3F5**. Здесь я приложу вариант **HTM Manual**, найденный мной после долгих поисков на просторах Интернет, а что касается соответствия – то существует прилагаемый к Протеусу **ProSPICE HELP**. Но в данном случае нас выручит больше даже не он, а хелп **ProSPICE Primitives**. На ближайшее время он становится нашим основным файлом помощи, а почему – Вы поймете в этом разделе чуть ниже.

Что же такое **SPICE** и с чем его едят? Основными моделями **SPICE** главным образом являются – резисторы, идеальные источники тока и идеальные источники напряжения. На основе их построены всевозможные модификации (источник тока, управляемый током; источник тока, управляемый напряжением; источник напряжения, управляемый током и т.д.). А уже на основе этих модификаций строятся все остальные модели аналоговых компонентов. В библиотеках **ISIS** эти модели расположены в **Modelling Primitives => Analog (SPICE)**. Аббревиатура модели несет в себе и некоторую англоязычную информацию. Примеры: **CCCS** – *(Current Controlled Current Source)* – источник тока, управляемый током; **VCR** – (*Voltage Controlled Resistor)* – резистор, управляемый напряжением; **AVCVS** – *(Arbitrary Voltage Controlled Voltage Source)* – в данном случае нелинейный *(Arbitrary)* источник напряжения, управляемый напряжением. Надеюсь, аббревиатуру других не затруднит расшифровать самостоятельно на основе данной информации. В этом разделе мы рассмотрим так называемые нелинейные источники. К ним относятся *(Рис. 46)*:

AVCVS – *(Arbitrary Voltage Controlled Voltage Source)* нелинейный источник напряжения, управляемый напряжением;

AVCCS – *(Arbitrary Voltage Controlled Current Source)* нелинейный источник тока, управляемый напряжением;

ACCVS – *(Arbitrary Current Controlled Voltage Source)* нелинейный источник напряжения, управляемый током;

ACCCS – *(Arbitrary Current Controlled Current Source)* нелинейный источник тока, управляемый током;

SUMMER – *(Arbitrary Voltage Controlled Voltage Source)* аналоговый сумматор;

MULTIPLIER – *(Arbitrary Voltage Controlled Voltage Source)* аналоговый перемножитель.

На *Рис. 46* - синим цветом расставлены скрытые имена выводов примитивов.

Полезный совет: *пока вы не начали активно работать с примитивами и не запомнили: где какой вывод – «разбейте» (**Decompose**) нужную модель и включите для выводов флажки **Show Name** где-нибудь на свободном поле проекта, при этом она перестает быть моделью, а становится набором графики и текстовых скриптов и не мешает при симуляции. У Вас же при этом всегда будет перед глазами информация – что и где расположено. Это касается не только примитивов, но и других моделей, имеющих скрытые наименования выводов.*

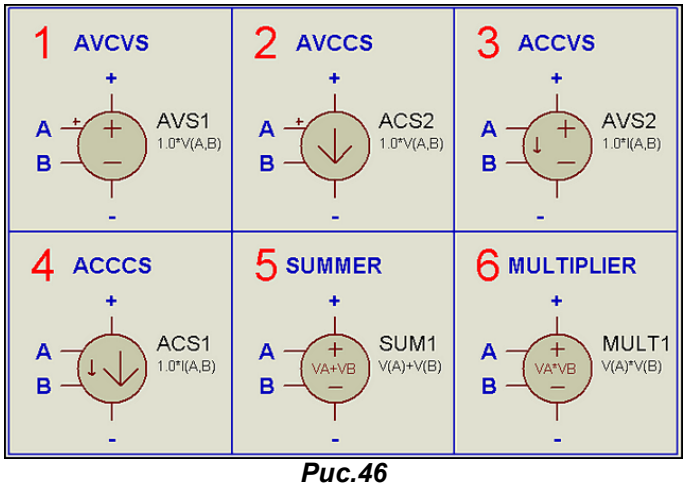


Рис.46

Рассмотрим подробно первый из них: **AVCVS** – источник напряжения, управляемый напряжением *(в русскоязычной литературе по SPICE сокращенно ИНУН)*. Его функцию можно просто описать как **Uвых=F(Uвх)**. По умолчанию она выглядит как **1.0*V(A, B)** – т.е. напряжение на выходе (между выводами **+** и **–**) равно единице умноженной на напряжение между входами **A** и **B**. Поддерживаются следующие варианты для входов: **V(A)**, **V(B)** и представленная по умолчанию дифференциальная разница напряжений между входами **V(A, B)**. Нужен линейный усилитель напряжения – в **Transfer Function** *(функция передачи)* достаточно изменить **1.0** на нужное усиление, например – в 10 раз *(Рис. 47)*, хотя проще это сделать с помощью описанных далее линейных источников.

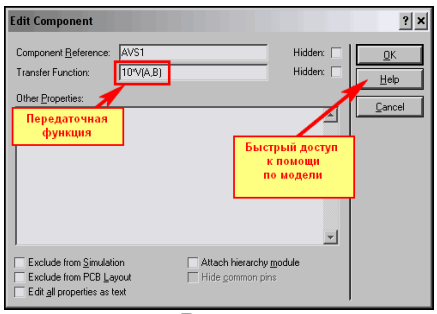


Рис.47

Что при этом мы увидим на аналоговом графике - показано на Рис. 48. В прилагаемом примере **Ex_AVS** представлен этот вариант и вариант «выпрямляющего» усилителя – функция записана, как **ABS(10*V(A,B))**.

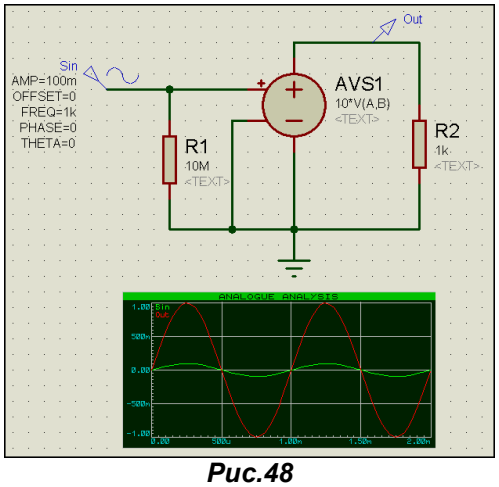


Рис.48

Основное достоинство нелинейных источников состоит в том, что мы можем применить достаточно сложные формулы, описывающие их поведение. Список возможных операторов и функций можно посмотреть, кликнув по кнопке **HELP** в окне редактирования свойств *(Рис. 47)*. Учитывая, что не все владеют английским и основами программирования, а также то, что там есть некоторые «экзотические» функции привожу их краткие характеристики на русском языке.

Допустимы следующие операторы математических действий:

+ - * / ^ – соответственно: сложение, вычитание, умножение, деление, возведение в степень. Например, для сумматора (**SUMMER)** по умолчанию: **V(A)+V(B)** – означает, что напряжение на выходе будет равно алгебраической сумме напряжений входа **A** относительно **GND** и входа **B** относительно **GND** *(Рис. 49)*. И совсем не обязательно, чтобы входные сигналы были постоянными потенциалами (как на рисунке) – это могут быть любые источники переменного, пульсирующего и т.п. напряжений. Операция **x^y** (**x** в степени **y**) фактически аналогична функции **PWR**, описанной ниже.

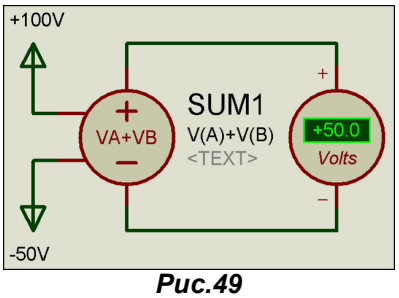


Рис.49

Поскольку мы рассматриваем подробно только источник с входным управляющим сигналом напряжения, где входное сопротивление выводов «ну очень велико», и им можно пренебречь, то хотелось бы здесь остановиться на особенностях применения источников с токовыми (**Current**) входами и выходами. В этих случаях сопротивление между соответствующими выводами входа или выхода модели равно нулю. Поэтому при применении таких источников будьте внимательны – Вам придется самостоятельно добавить последовательное сопротивление нагрузки, будь то вход или выход, чтобы ограничить соответствующий ток *(Рис. 50)*. Иначе получите ошибку, поскольку **ProSPICE** не сможет просчитать нулевое сопротивление.

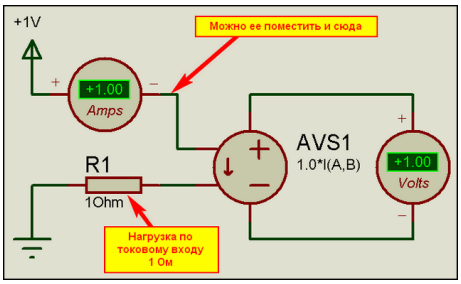


Рис.50

Теперь рассмотрим допустимые функции:

Тригонометрия: **SIN**, **SINH**, **ASIN**, **ASINH** *(синусы)*; **COS**, **COSH**, **ACOS**, **ACOSH** *(косинусы)*; **TAN, TANH, ATAN, ATANH** *(тангенсы)* – все это тригонометрические функции. Например: **SIN** – синус; **SINH** – гиперболический синус; **ASIN** – арксинус - функция, обратная синусу. Я не собираюсь здесь рассматривать тригонометрию, поэтому кто ее «прошел, но забыл» - открывайте любой справочник по математике и тригонометрическим функциям. Остальная математика:

ABS – абсолютное значение – или в другой трактации модуль числа (только положительное значение). Приведен в примере **Ex_AVS**, как возможность выпрямления синусоидального напряжения между выводами **A** и **B**.

EXP – функция обратная натуральному логарифму, т.е. это степень числа Эйлера **e = 2,712828**. Хотелось бы сразу обратить внимание, что и представленный в этом же списке функций **LN** – натуральный логарифм связан с этим основанием. А если Вам потребуется десятичный, то придется воспользоваться функцией **LOG** – логарифм по основанию 10. Соответственно, записав передаточную функцию, например, как **log(V(A, B))** при подаче напряжения **1000V** на вход (между **A** и **B**) Вы получите на выходе напряжение **3V** (степень числа 1000 по основанию 10).

SQRT – как многие, знакомые с программированием уже догадались, – это квадратный корень числа. Ну и одно существенное замечание – для функций **LN**, **LOG** и **SQRT** в случае отрицательного значения аргумента результат выражения принимает значение от модуля (т.е. от положительного значения аргумента), однако следует учитывать, что при наличии в выражении возможности деления на ноль Вы получите ошибку симуляции. Я хочу здесь заранее подчеркнуть, что если, например, у Вас в знаменателе дроби выражения стоит синусоидальная или другая функция, проходящая через ноль при некоторых значениях выражения (имеется в виду формула, введенная в **Transfer Function**) - вы и словите эту ошибку. За этим надо тоже внимательно следить, чтобы не напороть косяков.

Ну, это были все стандартные функции, а теперь переходим к «экзотике».

LIMIT – функция установленных пределов. Применительно к входам напряжения **AVCVS** для двух пределов: нижнего **10V** и верхнего **50V** описывается так: **limit(V(A,B), 10, 50)**. При этом возвращает на выходе (между выводами **+** и **–**) напряжение **10V** при входном меньше этого значения, **50V** – при входном больше этого значения или равное входному, если оно лежит в пределах от **10** до **50V**. Ну и конечно, для источника с входным токовым сигналом запись будет иная – например: **limit(I(A,B), 0.001, 0.1**). В данном случае пределы нижний – **1мА**, верхний – **100мА**. Обращаю Ваше внимание на то, что для описания пределов можно использовать только голые числа без расширений типа **m**, **u**, **M** и т.п. *(Рис. 51)*.

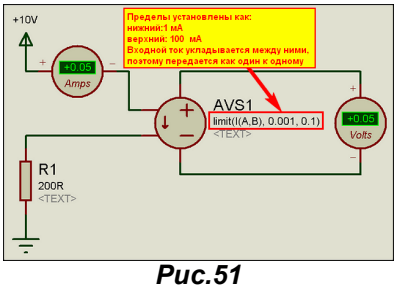


Рис.51

Функции **PWR** и **PWRS** оперируют с двумя аргументами и согласно **HELP** должны возвращать первая только положительное значение **|x|^y** (модуль **x** в степени **y**), а вторая аналогично, но со знаком при условии, если **x<0** значение функции отрицательное число, а если **x>=0** значение функции положительное. Однако, «что-то не так в доме Лабцентра». Действие этих функций показано на Рис. 52. В обоих случаях мы имеем знаковое значение на выходе. Оставим этот парадокс на совести разработчика, а сами примем к сведению, что если необходим модуль, то придется дополнительно воспользоваться функцией **ABS**.

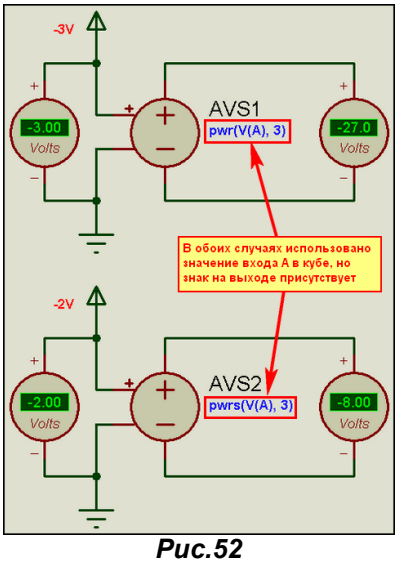


Рис.52

Функции **U** или **STP** возвращают единицу, если аргумент **x>1** или ноль в случае **x<0**, а функция **URAMP(x)** возвращает ноль при **x<0** или само значение **x** при **x>1**. Проверим их действие аналогично предыдущим *(Рис. 53)*. Здесь вроде все, так как и описано. Эти функции могут использоваться, чтобы синтезировать кусочно-нелинейные функции, хотя проблемы конвергенции могут возникнуть в точках излома.

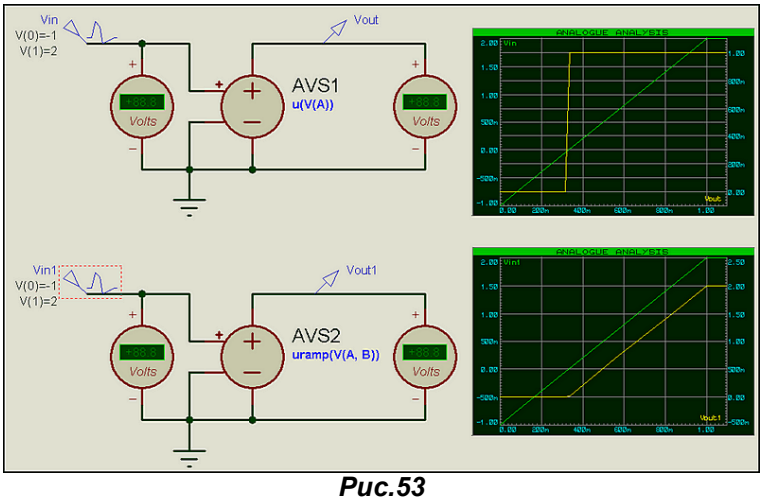


Рис.53

Ну и функция **SGN**, почему то не описанная в **HELP** говорит сама за себя **sign** – дословный перевод для математики знак. Функция принимает значение равное **-1** при **x<0** и **+1** при **x>0**. Картинку не привожу, но в прилагаемом архиве с примерами этот вариант есть – **Ex_SGN.DSN**.

Теперь к тому, почему я так подробно и в первую очередь разобрал эти источники и даже изменил первоначальный вариант этого раздела. Да потому, что это же основа основ для самостоятельного моделирования. Чуть позже мы рассмотрим варианты подробного схемотехнического и имитационного *(моделирование поведения)* моделирования, но забегая вперед, укажу, что фактически любой из управляемых источников почти готовая модель и операционного усилителя, и трансформатора (правда без учета магнитных свойств сердечника) и других линейных и нелинейных устройств. Тут главное проявить творческую смекалку, чего в России не занимать. Конечно, Протеус в смысле аналогового **SPICE** моделирования несколько отстает от такого монстра как **OrCAD** и ему подобных пакетов, базирующихся на **PSPICE**. Но, при умелом использовании, можно и здесь почерпнуть много нового и интересного в области схемотехнического моделирования электроники на компьютере.

В большинстве примеров, как я уже и отмечал, использована модель **AVCVS**, но все вышесказанное касается и остальных моделей данной группы примитивов. И еще такое мелкое замечание в помощь: как Вы наверно заметили из рисунка 46 – в графических изображениях ISIS принято обозначать токовые источники, пробники и т.п. стрелкой, а источники напряжения знаками **+** и **–** у выводов. Поэтому, надеюсь, что после более близкого знакомства с данными моделями Вы научитесь их легко и свободно различать по внешнему виду.

В дополнительном вложении **Arbitrary.rar** находятся в формате Proteus 7.6.SP0:
Ex_Arbitr_Devices.DSN – файл иллюстрация к Рис. 46;
Ex_AVS.DSN – файл проекта к рисунку 48;
Ex_Summ.DSN – файл проекта к рисунку 49 (SUMMER);
Ex_Cur_Res.DSN файл проекта к рисунку 50 (токоограничивающий резистор в ИНУТ);
Ex_Limit.DSN и **Ex_Cur_Limit.DSN** файлы к пояснению функции LIMIT для напряжения и тока;
Ex_Pwr.DSN – файл проекта к рисунку 52 – функции PWR и PWRS;
Ex_Uramp.DSN – файл проекта к рисунку 53 – функции U и URAMP;
Ex_Sign.DSN – файл проекта к пояснению действия функции SIGN.
Соответственно имеются и одноименные файлы секций для импорта в предыдущие версии.

Для того чтобы комфортно просматривать приложенный **Manual** по **SPICE3F5** начинайте просмотр с **index.html**. Из него возможна навигация по мануалу с помощью Internet Explorer или другого интернет браузера.

4.8. Аналоговые примитивы. Линейные управляемые источники сигналов.

Еще чуть-чуть остановимся на управляемых источниках. В примитивах, как я уже упоминал их несколько типов. Если в предыдущем случае мы рассмотрели **Arbitrary** источник, то здесь я хочу остановиться на управляемых линейных. Они более просты и по сущности относятся к классическим **SPICE** источникам токов и напряжений. Обозначения этих примитивов при помещении в схему совпадают с классическими, принятыми в **SPICE** и **PSPICE**. К ним относятся:

Modelling Primitives/ Analog(SPICE)	Символ обозначения в схеме и SPICE	Тип компонента: англ. (рус.)
VCVS	**E**	Voltage-Controlled Voltage Source *(Источник напряжения управляемый напряжением)*
CCCS	**F**	Current-Controlled Current Source *(Источник тока управляемый током)*
VCCS	**G**	Voltage-Controlled Current Source *(Источник тока управляемый напряжением)*
CCVS	**H**	Current-Controlled Voltage Source *(Источник напряжения управляемый током)*

Все это четырехполюсники *(Рис. 54)*и имеют всего два свойства:
GAIN – коэффициент передачи.
IC – inicial condition *(начальное состояние)* источника. По умолчанию оно не определено, однако в ряде случаев бывает полезно, например, если нам необходимо, чтобы источник напряжения стартовал не с нуля, а с 10V, то можно задать **IC=10**.

В примитивах есть и двухполюсные источники (на конце имеют цифру 2 – например, **CCCS2**). Отличие этих источников от своих четырехполюсных собратьев состоит в том, что в качестве входного сигнала в свойствах задается имя **Probe** – пробника (зонда), установленного в проекте. Однако толкового запуска двухполюсных примитивов мне так и не удалось добиться даже в лицензионной версии. Впрочем, всегда можно обойтись и четырехполюсниками.

Хотелось бы также обратить внимание на использование аналоговых резисторов на входах/выходах как в предыдущем разделе выше, так и в примерах из прилагаемого архива ***Lin_Sources.rar***. Дело в том, что входы и выходы примитивных идеальных источников не имеют собственного сопротивления, поэтому в случае применения токовых источников приходится включать нагрузочный резистор последовательно, а в случаях входов/выходов по напряжению подключать параллельно для создания сопротивления нагрузки. Этот факт тоже необходимо учитывать.

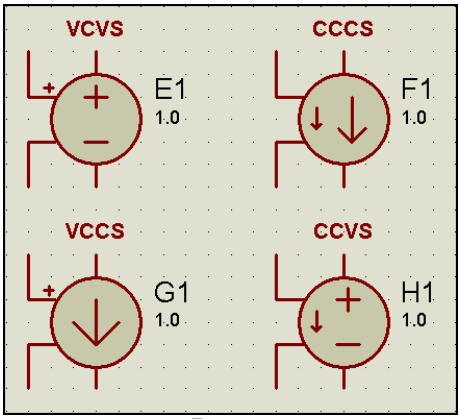


Рис.54

Несколько особняком выделяются линейные управляемые напряжением или током резисторы: **VCR** *(Voltage-Controlled Resistor)* и **CCR** *(Current-Controlled Resistor)* У этих линейных примитивов больше задаваемых свойств. Ведут они себя следующим образом *(рассмотрим на примере резистора управляемого напряжением Рис. 55)*:

При напряжении на входе ниже **Off Voltage** *(в примере VOFF=1V)* сопротивление резистора будет равно **Off Resistance** *(в примере ROFF=100 Ом)*. При напряжении на входе выше **On Voltage** *(в данном случае VON=10V)* сопротивление выходного резистора будет равно **On Resistance** *(в примере RON=1 Ом)*. При изменении напряжения в пределах от VOFF до VON согласно HELP на данную модель используется линейная интерполяция. Ну, линейностью, судя из графика на Рис. 55 тут не очень пахнет, хотя в HELP и оговаривается, что при этом модель ведет себя как усилитель. Примеры с данными моделями приведены во вложении VCR.DSN, упакованном в ***Lin_Sources.rar***.

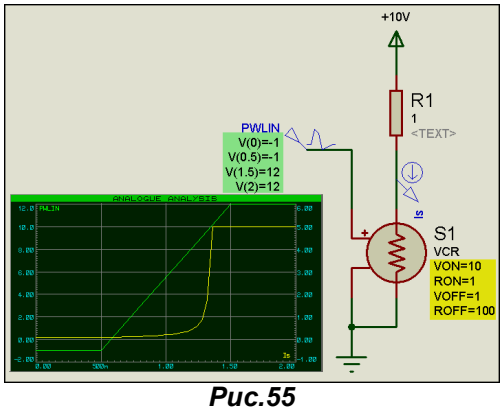


Рис.55

Ну и в заключение данного материала хочу привести один «нестандартный» прием использования линейных управляемых источников. Поскольку все они могут работать как усилители с единичным коэффициентом и являются четырехполюсниками их можно применить для измерения сигналов между двумя точками схемы для виртуальных приборов из набора Протеуса и для графиков тоже. Напомню, что тот же осциллограф при применении всех его четырех каналов меряет сигнал относительно шины **GND**. Установленные в схеме зонды напряжения – тоже. Чем это чревато видно из *(Рис 56)*. В качестве источника сигнала здесь применен двухполюсник **ALTERNATOR** – не имеющий связи с **GND**, поэтому результат, измеренный непосредственно с него E1(P) – зеленая трасса занижен относительно реальной амплитуды 10V. Применение в качестве «разделительного трансформатора» **VCVR** с единичным усилением позволяет и на графике и на осциллографе получить реальный результат (красная трасса). Пример в IZMER.DSN вложения.

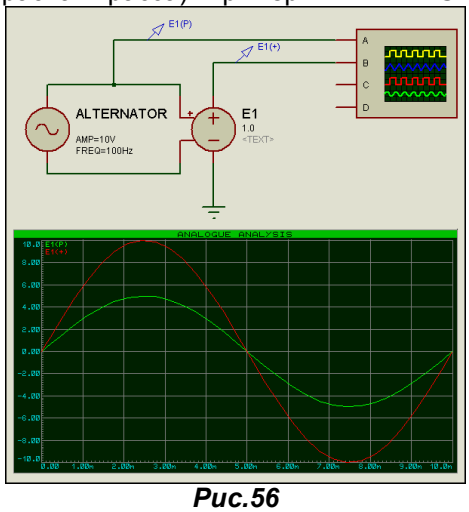


Рис.56

4.9. Применение Analogue и Mixed Graph для исследования сигналов.

В **п. 2.18-2.20** первой части FAQ мы рассмотрели применение **Digital Graph** (Цифрового графика) для исследования цифровых сигналов. Так как сейчас мы рассматриваем в основном аналоговые модели, пришла пора более подробно рассмотреть возможности аналогового графика – **Analogue Graph**. По сути, все сказанное в первой части применимо и к аналоговому графику, за исключением того, что в цифровом для каждой трассы (**Trace**) имеется своя горизонтальная ось, а в аналоговом трассы рисуются, накладываясь друг на друга разными цветами. Необходимые зонды или сигналы генераторов добавляются теми же способами: либо перетаскиванием мышкой, либо через клик правой кнопкой внутри графика через опцию **Add Traces** (**Ctrl+T**). При добавлении сигналов через правую кнопку можно видеть, что в правой верхней части Trace Type активным является только флажок **Analog** – ведь это аналоговый график, но зато стала возможным привязка сигнала, как к левой, так и к правой оси Y (Axis). Вещь чрезвычайно полезная, особенно при большой разнице в уровнях разных сигналов размещенных на одном графике. Например: размещая на одном графике сигнал с амплитудой **100V** – привязываем его к **Left** (по умолчанию), а с амплитудой **100mV** к правой оси **Y**. При этом если мы не трогаем масштабы осей *(об этом чуть ниже)*, сигналы на графике будут выглядеть по вертикали сопоставимо, т.к. ISIS автоматически установит разные масштабы. Кроме того, на графике мы можем отобразить и результирующие трассы от разных сигналов (до четырех **P1, P2, P3, P4**).

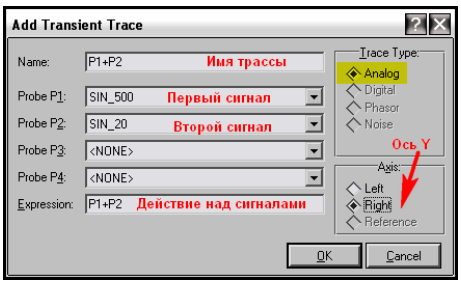


Рис.57

Например, на Рис. 57 задано сложение (оно встало по умолчанию) в строке **Expression** для сигналов генераторов **SIN_500** (500Гц 500mV) и **SIN_20** (20Гц 1V). Имя трассы P1+P2 также изменено (по умолчанию туда подставляется имя пробника P1), ну и в довершение всего привязал этот сигнал к правой оси. К левой оси графика ранее были привязаны исходные трассы генераторов. Результат действия можно посмотреть на (*Рис.58)* и в прилагаемом вложении **Analog_Graph.rar**. Желтая трасса является результирующей сложения для двух синусоид.

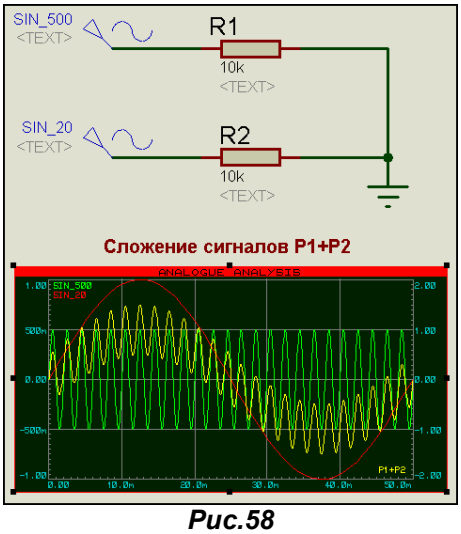


Рис.58

При этом совсем не обязательно в графе **Expression** применять именно сложение. Вы можете применить и все те функции, которые описаны в разделе нелинейных источников. Во вложении есть графики для произведения **P1*P2** и даже для такой экзотики как **URAMP(P1+P2)**. Созданную трассу всегда можно подкорректировать, если кликнуть по ее названию правой кнопкой мыши – будьте внимательны – щелкать надо именно по названию (например, для желтой трассы на Рис. 58 по P1+P2 в правом нижнем углу). При этом в контекстном всплывающем меню вверху будут доступны три опции именно для этой трассы *(Рис.59)*:

Drag Trace Label – позволяет передвинуть название трассы к другой оси Y (от правой к левой или наоборот по диагонали). Если у данной оси уже стоит несколько лейблов, то по этой опции можно переместить имя трассы по вертикали в списке, при этом автоматом изменится и цвет в соответствии с установленным для данного номера трассы в меню **TEMLATE => SET GRAPH COLOR**.

Edit Trace Properties – вызывает всплывающее окно, в котором можно изменить имя (**Label**) или действие (**Expression**), а также включить подсветку вычисленных симулятором точек графика флажок **Show Data Points?** Сразу отмечу, что поскольку точки расположены достаточно часто, то видны они только при максимизации графика и его отдельных участков.
Delete Trace – удаляет выбранную трассу с поля графика.

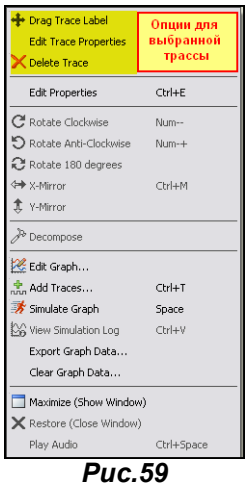


Рис.59

Парочка советов «для ленивых». Если Вы пользуетесь перетаскиванием генераторов и сигналов зондов на график левой кнопкой мыши, то отпустив кнопку, когда курсор в левом верхнем углу поля графика, вы назначите трассу к **Left Axis**, а в правом нижнем – соответственно к **Right Axis**. Зацепив лэйбл (имя трассы) сразу левой кнопкой в поле графика его можно перетягивать по диагонали, назначая или к правой или к левой оси Y.

Немного поигравшись с трассами теперь заглянем в свойства самого аналогового графика. Дважды кликнув левой кнопкой мыши по графику или через правую и опцию **Edit Graph**, попадаем в свойства графика *(Рис. 60)*. Здесь, как и в **Digital Graph** можно установить время старта графика, останова графика (ось X). Можно поменять название в титульной строке – по умолчанию там стоит **ANALOGUE ANALYSIS**. Не советую здесь применять кириллицу во избежание ошибок. Равно как и для названий левой и правой осей.

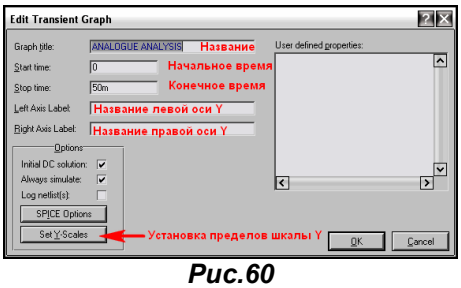


Рис.60

По умолчанию уже установлены флажки Inicial **DC Solution** – учитывать постоянную составляющую и **Always simulate** – всегда симулировать. При желании можно поставить галочку **Log Netlist**, чтобы в логе прописывался список цепей. Через кнопку **SPICE Option** осуществляется быстрый доступ к параметрам симуляции прямо из этого окна. Нас же больше всего интересует кнопка **Set Y-Scales**, которая у цифрового графика была неактивной. Через эту кнопку мы попадаем в окно установки пределов для левой и правой шкалы Y (Рис. 61). По умолчанию пределы задаются автоматически по максимальному верхнему и нижнему значениям наибольшего по размаху сигнала, соотнесенного к этой оси. Однако это не всегда красиво выглядит, поэтому я, например, предпочитаю в этом окне задать значения вручную на 1-2 значения выше и ниже от заданных при автоматическом выборе – так выглядит красивее, но это дело личного вкуса. Для этого достаточно установить флажок **Lock values**, тогда минимальное и максимальное значения станут доступными для изменения. На Рис. 61 я так сделал для левой шкалы Y, а правую оставил в автомате.

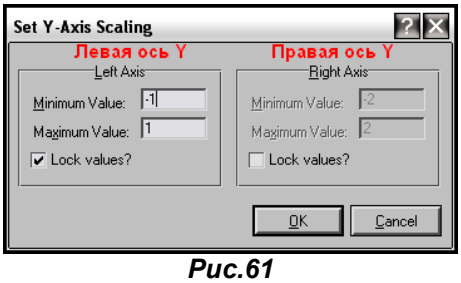


Рис.61

Ну, вот теперь по аналоговому графику почти все и всего пара слов по **Mixed** – смешанному графику. Основное его достоинство в том, что в верхней части его можно разместить цифровые сигналы, как на **Digital Graph**, а в нижней аналоговые. Поэтому, при добавлении в него трасс лучше пользоваться контекстным меню правой кнопки мышки Add Trace, при этом вы сразу можете для данной трассы выбрать тип **Trace Type**, поскольку будут доступны два варианта – **Analog** или **Digital**. В остальном он полностью сродни по аналоговой части аналоговому графику, а по цифровой – цифровому. В примере **Analog_Graph.rar** такой тип графика размещен в правой части листа проекта.

Ну и в заключение немного о возможностях аналогового и смешанного графиков при максимизации – через контекстное меню правой кнопки мыши – опция **Maximize (Show Window)**. Данный режим показан на *Рис. 62*.

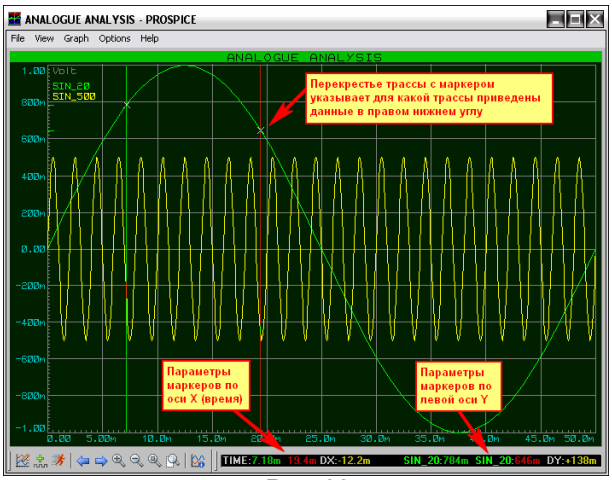


Рис.62

Так же, как и для рассмотренного в первой части FAQ цифрового графика здесь становятся доступными дополнительные опции: верхнее и нижнее меню. Так же, как и в цифровом, возможна установка двух вертикальных маркеров: зеленого – щелчком левой кнопкой мыши в нужном месте графика и красного – то же действие, но при нажатой клавише **Ctrl**. Отличие аналогового графика состоит в том, что при установке маркеров в черном нижнем окне кроме временных параметров, которые расположены слева, - справа появляются параметры точек пересечения выбранной трассы с маркером и вычисленное приращение по оси **Y**. Причем, если при установке маркера щелкнуть левой кнопкой по трассе другого сигнала (в данном случае желтого **SIN_500**), то перекрестье переместится на эту трассу и числовые параметры оси Y для данного маркера будут вычислены для нее. Ну и конечно, многие наверняка обратили внимание, что у меня вторая трасса желтая, а не красная – по умолчанию. Это я делаю потому, что красный цвет в онлайн картинках графиков менее заметен, чем более яркий – желтый. На этом, пожалуй, разборку аналоговых и смешанных графиков можно закончить. Хочу только еще раз напомнить, что в примерах – **SAMPLES**, идущих вместе с Протеусом есть целая папка **Graph Based Simulation**. Именно в ней сосредоточены примеры с применением всех типов графиков, существующих в Протеусе. Пожалуйста, не пренебрегайте имеющейся под рукой информацией, там много полезного.

4.10. PROSPICE-примитивы резисторов и конденсаторов. Немного о температурном моделировании в Proteus и использовании DC SWEEP графика.

В этом разделе я кратенько пробегусь по свойствам примитивов резисторов **R** и конденсаторов **C.** Итак, резистор наиболее простая и фундаментальная модель **SPICE**. Ток через резистор пропорционален напряжению, приложенному между выводами 1 и 2. В окне свойств резистора из библиотеки примитивов ISIS имеется только строка задания его сопротивления **Resistance**. Однако это не значит, что по сравнению с классической SPICE-моделью у него отсутствуют остальные свойства. Просто, если Вы затеете в Протеусе температурное моделирование, то придется их вводить вручную в окне **Other Properties**. А по умолчанию они следующие:

TC1	0.0	Значение линейного температурного коэффициента A
TC2	0.0	Значение квадратичного температурного коэффициента B
TEMP	27	Реальная температура резистора.
TNOM	27	Температура, при которой TC1, TC2 были «измерены» при создании модели.

Суммарно, сопротивление резистора при определенной температуре **t** определяется следующей формулой:

Rt=R+A*dt+B*dt2** где **dt=t-25**

Так как по умолчанию у нас **TC1** и **TC2** равны нулю, то и принимается значение, установленное в окне **Resistance**. Если же их задать вручную для конкретного резистора (ов) в проекте, то при моделировании температурных режимов линейная и квадратичная составляющая будут вносить свои коррективы в значения сопротивления данных резисторов.

Вот это последнее мы сейчас и рассмотрим на примере резистивного измерительного мостика, приведенном в книге Р. Хайнемана «Визуальное моделирование электронных схем в PSPICE».

Если кто-то имеет данную книгу, то может сравнить с приведенным там примером в гл.7.3. Итак, согласно приведенной там иллюстрации соберем схему термоизмерительного мостика (Рис. 63) из резисторов сопротивлением 1 кОм и источника постоянного напряжения 10В.

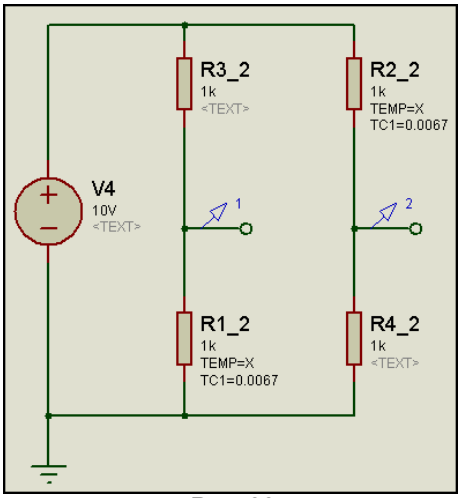


Рис.63

Обратите внимание, что для двух резисторов в диагоналях я задал линейный температурный коэффициент **TC1=0,0067** 1/K, что соответствует сопротивлению из никеля. Кроме того, для параметра текущей температуры **TEMP=X**, потому что предполагаю ее менять в качестве параметра в **DC SWEEP** анализе. Теперь тем же способом, что и аналоговый график размещаем на поле проекта график **DC SWEEP** – он предпоследний в левом меню графиков, и заходим в его свойства Edit Graph (Рис. 64).

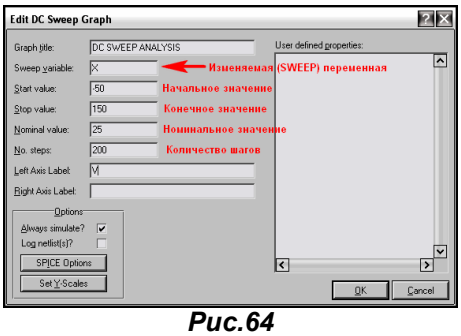


Рис.64

Ну вот, теперь, наверное, стало понятно - почему я присвоил температуре значение **X**. Ведь ее мы и будем свипить, т.е. развертывать по горизонтальной оси. Присваиваем ей в соответствии с книжным примером **Start Value** – начальное значение **-50** (имеются в виду градусы Цельсия), **Stop Value** – конечное значение **150**, **Nominal Value** номинальное значение **25** и **No. Steps** – количество шагов **200** (по 1 на градус размаха 50+150 – здесь я покривил душой, у Хайнемана шаг взят 0.1 градуса, впрочем желающие могут поставить 2000 для пущего соответствия).

Затем добавляем на график трассу **Add Trace**, которая является разностью напряжений для пробников 1 и 2 на схеме *(Рис. 63)*. Выглядеть это будет, как на *(Рис. 65)*. Здесь в принципе Вам все уже знакомо по аналоговому графику. Т.е. наша трасса будет представлять разность напряжений в точках пробников 1 и 2 (средние точки диагоналей измерительного моста). Фантазии у меня не хватило, и обозвал я ее незатейливо **P1-P2**. Обратите внимание, чтоб мне лишний раз не пояснять, что и для этого типа анализа имеется возможность использования левой и правой вертикальных осей Y. Мы используем левую, которая по умолчанию.

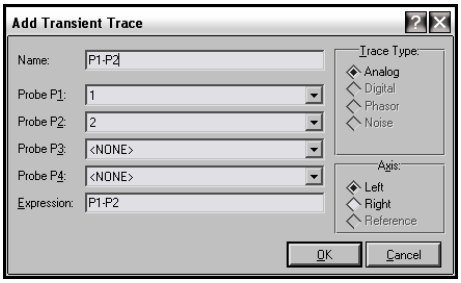


Рис.65

И еще один нюанс, - для вертикальной оси **Y** я вручную установил верхний и нижний пределы (кнопка **Set Y-Scales)** соответственно **-4V** и **4V** для большей аналогии с книжным примером.

Проделав все эти манипуляции, запускаем график на выполнение. Я внес на лист рисунки из книжки, правда качество скана оставляет желать…, но сравнить можно. Результат выполнения графика и прототип из книжки на (*Рис. 66)*.

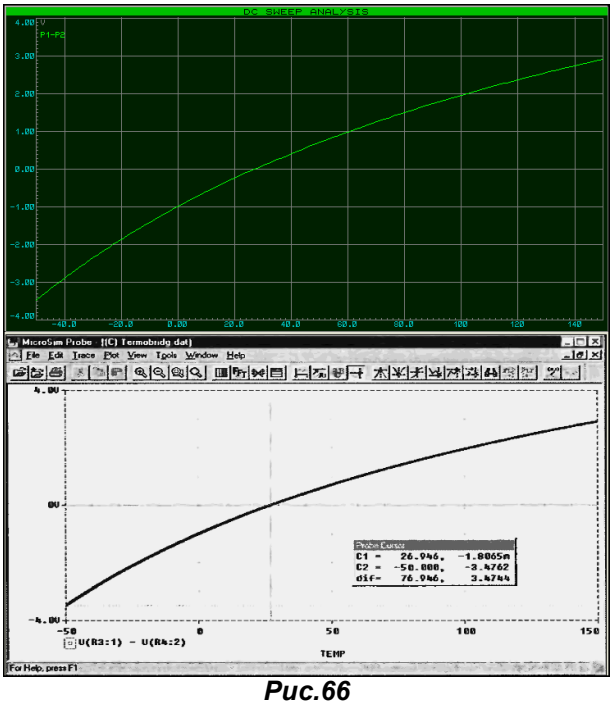


Рис.66

Этот пример содержится в прилагаемом проекте **RES_TEMP.DSN** из архива **RES&CAP.RAR** на втором листе – **Root Sheet 2**, а на первом – с помощью **DC SWEEP** я приложил график для изменения одного сопротивления моста из различных материалов: никеля, меди и платины.

Вот такие метаморфозы возможны с помощью **DC SWEEP**. Но совсем не обязательно, чтобы изменяемым параметром была температура. В качестве дополнительного примера приложен пример **RES_POWER.DSN**, где по оси **X** разворачивается сопротивление резистора в диапазоне от 1 до 150 Ом, а по оси **Y** – рассеиваемая на нем мощность, вычисляемая по формуле **P=U*I**.

Ну а теперь немного о конденсаторах. Примитив конденсатора в ISIS тоже простая модель, не учитывающая ни утечек, ни индуктивности, ни других параметров реальных конденсаторов. Единственным основным параметром является задаваемая емкость **Capacitance (Farads)**. Однако и здесь есть парочка параметров, которые задаются вручную, но могут в корне влиять на моделирование схемы. Поэтому рассмотрим, - как их правильно применять, а попутно и то, чем примитив конденсатора отличается от других моделей этого типа в **ISIS**.

PRECHARGE – этот параметр специфическое расширение **PROSPICE** по сравнению со стандартным **SPICE** и определяет начальное напряжение зарядки конденсатора. Если данный параметр строго не описан, то это напряжение принимается таким, которое соответствует данной операционной точке.

IC – в данном случае это отнюдь не **Inicial Condition**, а **Inicial Charge** (начальный заряд) и также должен определять начальное напряжение заряда конденсатора. Отличие состоит в том, что данный параметр используется симулятором только тогда, когда начальное напряжение невозможно рассчитать.

Вот так все запутанно расписано в HELP, но рассмотрим дальше – как это выглядит на практике. На Рис. 67 для цепочки **1** применены параметры по умолчанию, а для цепочки **2** установлено стартовое напряжение для конденсатора **PRECHARGE=0**.

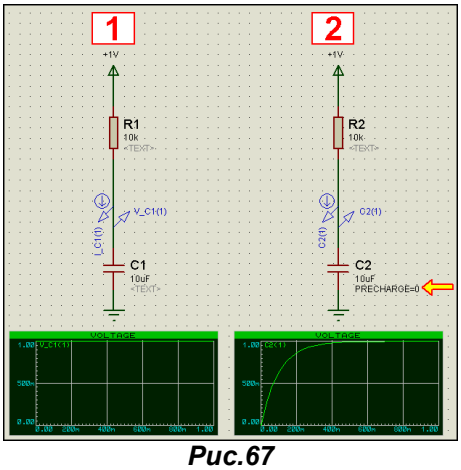


Рис.67

Вот сразу и выяснился один любопытный момент – на левом графике напрочь отсутствует кривая первоначального заряда, т.е. как и указано в HELP – напряжение на верхней по схеме обкладке конденсатора сразу же равно питанию. Делайте выводы те, кто приклеивает данную цепочку к входу сброса микроконтроллеров или цифровых счетчиков и т.п. Будет оно симулироваться или нет? Идем дальше, и на Рис. 68 разберем действие параметра **IC**.

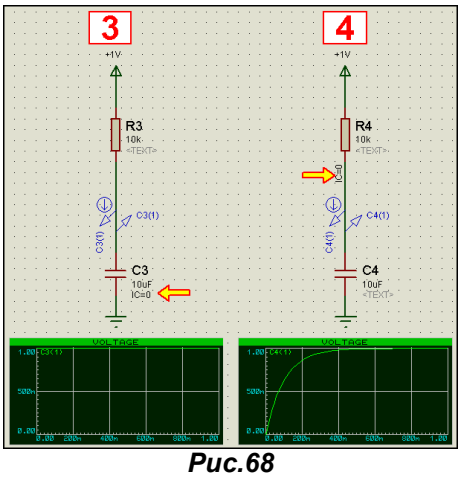


Рис.68

Когда мы подставили **IC=0** в параметры конденсатора – цепочка **3**, то получили, по сути, тоже, что и по умолчанию, т.е. в стартовый момент напряжение на верхней точке уже равно питанию. Но есть еще один интересный момент – **IC** можно использовать и для проводников. Это очень просто – присваиваем проводнику (в нашем случае верхнему) – цепочка 4, то, по сути, опускаем его напряжение в стартовый момент до нуля, а с ним и верхний вывод конденсатора. Вот и получили тот же эффект, что при варианте **2**. Все эти примеры находятся в прилагаемом в архиве проекте **CAP.DSN**.

Вариант с **IC** для проводников классически применен в примере **Ff.DSN**, идущем в поставке Протеуса в папке **\SAMPLES\Graph Based Simulation**. В этом примере рассмотрен классический транзисторный мультивибратор с коллекторными связями (Рис. 69). Я не удержался от того, чтобы не привести его здесь, подчеркнув важные моменты.

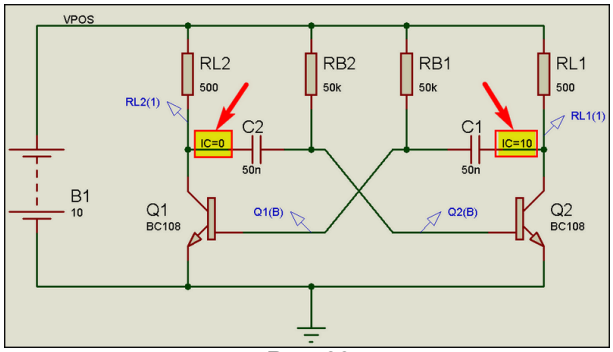


Рис.69

Те, кто внимательно изучал электронику в ВУЗах, а не просто «ходил за дипломом», да и многие радиолюбители наверняка давно знают, а для тех, кто начинает свои посты на форуме с фразы: «я в электронике полный дуб» - поясню, что в идеале данная схема не в состоянии генерировать импульсы. Это потому, что если в обоих плечах симметричного мультивибратора стоят строго одинаковые компоненты, - он будет находиться в состоянии статического равновесия. На практике из-за разброса параметров радиоэлементов такого не бывает, поэтому мультик всегда запускается. Но ведь любой симулятор и содержит строго идеальный набор моделей. Значит, если такой мультик запустился, то где-то производитель программы Вас слегка охмурил. Ну а в нашем случае все просто. Для того, чтобы внести стартовый разбаланс в плечи мультивибратора для коллекторной точки левого плеча принято **IC=0V**, а для правого **IC=10V**. Вот и получился нужный стартовый перекос. Примите на вооружение, как полезный совет от разработчиков.

Ну и в заключение этого раздела еще немного о моделях конденсаторов, расположенных уже в библиотеке **Capacitors**. Большинство из них ведет себя так же, как и примитив, на базе которого они основаны. Так что параметр **PRECHARGE** для них очень актуален. Но есть парочка моделей – это тоже базирующийся на примитиве анимированный конденсатор (подпапка **Animated**) и схематичная модель (или как принято в других пакетах такой вариант называть - макромодель) **REALCAP** (подпапка **Generic**) в которых не надо при установке в проект принудительно включать это свойство. У макромодели **REALCAP** реализованы, кстати, и все дополнительные навороты - Рис. 70, вплоть до ESR. Но прежде, чем активно использовать эту модель в своих проектах, внимательно подумайте, – а «стоит ли игра свеч». Ведь все эти навороты реализованы схемно в макромодели и

утяжелят Вам симуляцию проекта, как говорится «по полной программе», поэтому и хороши они только там, где необходимо обязательно учесть данные параметры.

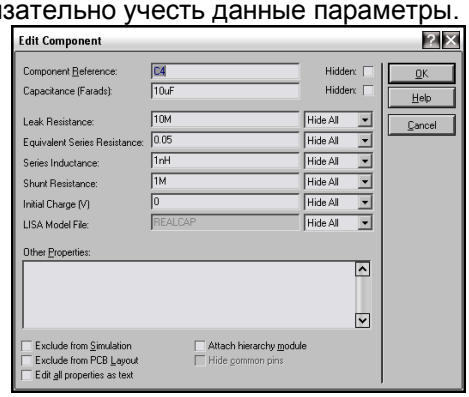


Рис.70

4.11. PROSPICE-примитив индуктивности. График AC SWEEP. Взаимосвязь индуктивностей. Особенности моделей трансформаторов в Протеусе.

Модель также **INDUCTOR** из папки **Primitivies** относится к самым простым моделям SPICE. В свойствах модели есть только один параметр, вынесенный в отдельную строку – **Inductance (Henrys)** *(Индуктивность в Генри),* который по умолчанию равен **1 мГн**. Дополнительно можно назначить вручную еще два параметра. Это:

IC – в данном случае **Inicial Current** – начальный ток через индуктивность (обратите внимание - как по разному трактуется параметр **IC** для разных компонентов);

MUTUAL_elem – коэффициент взаимосвязи между данным элементом и тем, на который указывает **elem**. Математически он трактуется так:

Коэффициент связи **K=M/(SQRT(L1*L2))** – взаимоиндуктивность **M** деленная на корень квадратный из произведения индуктивностей **L1** и **L2**. Данный параметр мы подробно рассмотрим здесь же чуть ниже, поскольку это основа для моделирования трансформаторов.

А теперь - почему я решил заострить Ваше внимание на этом примитиве и вообще индуктивных элементах в Протеусе. Дело в том, что у того же примитива индуктивности есть одна характерная особенность – нулевое сопротивление постоянному току. Поэтому, если Вы собрались включить его, где то в моделирование и не позаботились о том, чтобы добавить с ним последовательно обычный резистор, то рискуете получить сообщение об ошибке *(Рис. 71)*. Это особенность не только **PROSPICE**, но и других SPICE-симуляторов и об этом нельзя забывать при моделировании устройств с индуктивными элементами.

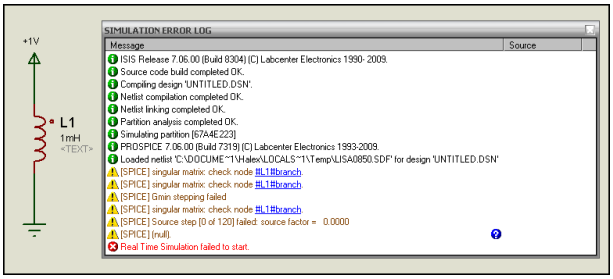


Рис.71

В тоже время в библиотеке **Inductors** как и в случае с конденсаторами в папке GENERIC находится схематичные модель **REALIND** и модели **NLINDUCTOR**, **SATIND**, в свойствах которых схемным образом уже добавлены резистивные **Equivalent Parallel Resistance** *(эквивалентное параллельное сопротивление)* **Equivalent Serial Resistance** *(эквивалентное последовательное сопротивление)* и емкостная **Equivalent Parallel Capacitance** *(эквивалентная параллельная емкость)* составляющие *(Рис. 72)*.

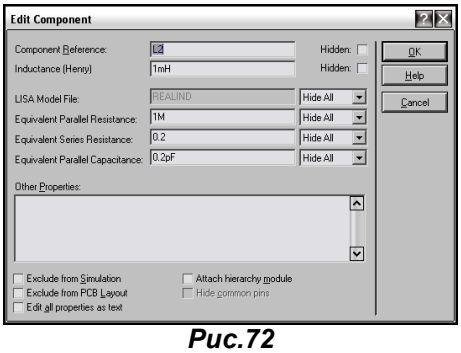


Рис.72

Рассмотрим, что они нам дают по сравнению с примитивом индуктивности. Зачем нужно последовательное сопротивление мы уже выяснили. Зачем же нужно **Equivalent Parallel Resistance**? В реальных катушках индуктивности всегда наблюдается снижение добротности при увеличении частоты свыше некоторого конкретного значения. У примитива **INDUCTOR** с ростом частоты импеданс постоянно растет, причем если отложить этот рост в логарифмическом масштабе, то график роста линейный. Для иллюстрации сказанного воспользуемся возможностями графического анализа **AC SWEEP**. Данный график несколько схож по параметрам с рассмотренным в предыдущем разделе **DC SWEEP**, но в данном случае в отличие от предыдущего изменяемым параметром является частота входного генератора. Итак, построим простейшую схему (Рис. 73), где параметры генератора зададим так, как показано на рисунке. Имя генератора можно набрать любое, но помните, что позже оно нам потребуется в свойствах графика **AC SWEEP**. Также я установил флажок **Current Source** – источник тока, поскольку мне нужен именно токовый генератор и задал амплитуду и постоянную составляющую **1 mA**. В графе Frequency может стоять любое значение, кроме нулевого, поскольку именно этот параметр генератора и будет «свипироваться», т.е. изменяться в графике **AC SWEEP**. Ну, еще попутно я снял флажок **Hide Properties?** – чтобы параметры высвечивались в проекте на месте скрытого **<TEXT>**. И последнее новшество – вместо конкретного значения индуктивности я задал значение **X1**, потому что этот параметр также будет изменяться для получения семейства (нескольких) характеристик для разных значений индуктивности. Почему именно **X1**, а не **X**, как раньше? Просто в проекте будет несколько примеров индуктивностей и несколько графиков **AC SWEEP**, и чтобы не запутать симулятор и себя я их пронумеровал.

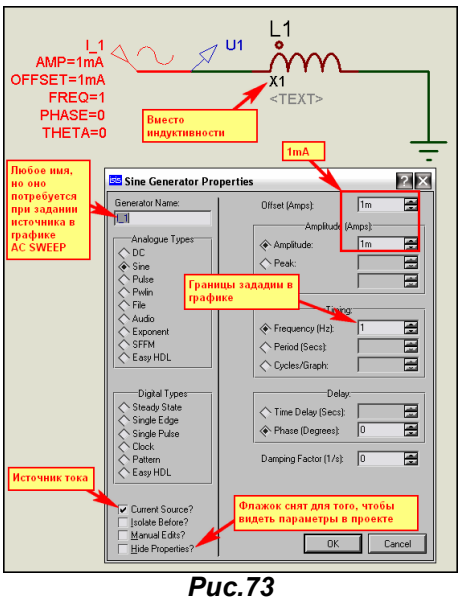


Рис.73

Теперь по аналогии с предыдущими примерами графиков растягиваем по диагонали в поле проекта график **AC SWEEP** из левого меню **GRAPH MODE**, после чего заходим в его свойства (Рис.74). Здесь уже многое нам знакомо по предыдущему **DC SWEEP**. Выбираем в качестве **Reference** через раскрывающийся список наш генератор синусоидального сигнала (у меня **I_1**), задаем начальную частоту **Start frequency:** 10k (10 килоГерц), а конечную частоту **Stop frequency:** 1G (1 ГигаГерц). В графе **Interval** оставляем **DECADE**, т.е. каждый последующий шаг по оси частот (горизонталь) в 10 раз выше частоту (доступны еще **OCTAVE** – вдвое и **LINEAR** – линейный), ну и количество шагов (интервалов) – **No. Steps/Interval** оставим равным по умолчанию 10, хотя если хотите получить плавные кривые с меньшим количеством изломов, то количество шагов необходимо увеличить, а если нужна меньшая точность, то уменьшить (минимальное значение 5).

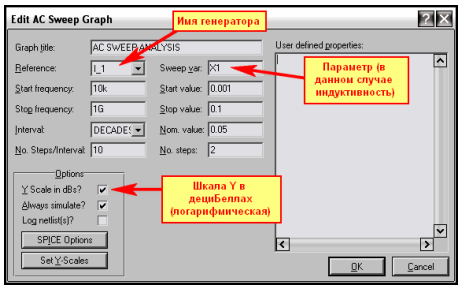


Рис.74

Теперь разберемся с осью **Y**. Ну, во-первых, в Options я оставил включенным по умолчанию флажок **Y Scale in dBs?**, поэтому вертикальная шкала будет в децибелах. Во-вторых, поскольку график позволяет получить параметрический анализ, а в качестве изменяемого параметра я

выбрал индуктивность, то я задал ее начальное (1mH или 0.001H), конечное (100mH или 0.1H) и номинальное (50mH или 0.05H) значения и количество шагов **No.Steps** 2. Для данного случая будет выведено три кривых: соответственно при начальном, номинальном и конечном значениях. Маленький нюанс – чтобы вывести 2 кривых, нужно задать **No.Steps** равным 1 (меньше нельзя), а если необходима только 1 кривая, то начальное и конечное значения должны быть равны. При этом в **Nom.value** и **No.Steps** может стоять что угодно. Ну и последнее **No.Steps** не должно превышать значение 10. Эти замечания относятся и к **DC SWEEP**.

Со свойствами графика **AC SWEEP** пока все, пора добавлять в него трассу. Я добавлю зонд напряжения **U1** (Рис. 72) через меню правой кнопки мыши **Add Traces** (Рис. 75).

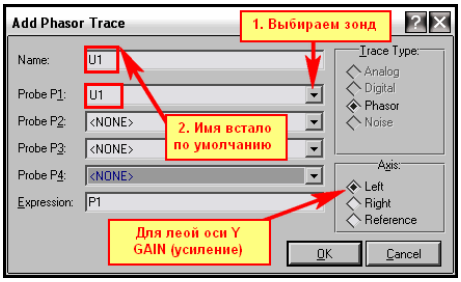


Рис.75

Добавляем к левой оси **GANE**, а если в этом случае выбрать **Axis Right**, то данный зонд будет назначен к правой вертикальной оси **PHASE** и получим кривую изменения фазы сигнала в этой точке. Можно добавить этот зонд и туда и сюда, чтобы иметь две кривых, но меня в данный момент интересует именно напряжение, поскольку оно прямо пропорционально полному (комплексному) сопротивлению катушки в зависимости от частоты. Запускаем граф на симуляцию и получаем картинку как на Рис. 76.

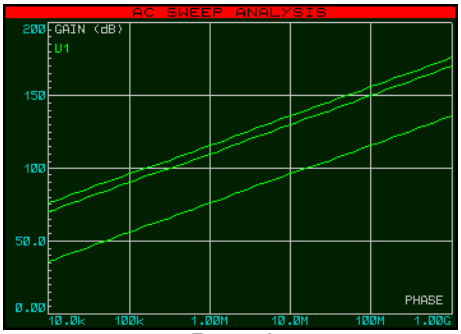


Рис.76

Итак, мы видим, что для идеальной модели усиление (ну и напряжение) на участке до 1 ГигаГерца постоянно растет, что, как сами понимаете, отнюдь не соответствует реальности. Не смущайтесь линейностью графика – напомню, что мы выбрали по **Y** логарифмическую шкалу, а по **X** десятикратные интервалы частоты. Но факт, остается фактом – напряжение на катушке, а, следовательно, и ее добротность с повышением частоты постоянно растут. Вот это парадокс, приводящий к острым выбросам на высоких частотах, и является для SPICE-модели катушки дополнительным фактором, влияющим на сходимость решения и, следовательно, успешную симуляцию индуктивности. Для того чтобы учесть реальное снижение добротности вследствие вихревых потерь и поверхностных токов необходимо параллельно катушке подсоединить шунтирующий резистор. При этом на низких частотах преобладающую роль будет иметь индуктивность, а на высоких – сопротивление резистора.

Сопротивление шунтирующего резистора нетрудно рассчитать по формуле **R=2π*f*L=6,28*f*L**, где **f**-частота (Гц), **L**-индуктивность (Гн) и **R** – сопротивление (Ом).

Например: для частоты 200кГц и индуктивности 1мГн необходим резистор 1256 Ом. Что при этом произойдет - видно на графике (Рис. 77). В районе от 100кГц до 1 МГц наблюдается перегиб кривой, после чего на более высоких частотах она практически постоянна.

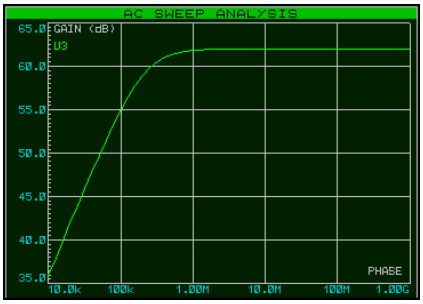


Рис.77

Ну и для окончательного «чувства глубокого удовлетворения» при моделировании индуктивностей осталось учесть паразитную межвитковую емкость, что, и сделано в схематичных моделях упомянутых выше и имеющих окно **Properties** как на Рис. 72. Все эти рассуждения я свел в пример **L_REAL.DSN**, прилагаемый в архиве **INDUCT.RAR**.

Теперь перейдем к разбору связанных индуктивностей, ну и как следствие – трансформаторов. Взаимосвязь индуктивностей в ISIS определяется как **mutual inductance** – взаимная индуктивность и подробно рассмотрена в двух прилагаемых примерах **Mutual1.DSN** и **Mutual2.DSN** из папки **Graph Based Simulation**. Для двух и более взаимосвязанных индуктивностей всегда используется одно обозначение с указанием буквенного индекса через двоеточие. Например: **L1:A**, **L1:B**, **L1:C** и т.д... Коэффициент взаимосвязи, лежащий в пределах от **-1** (ВНИМАНИЕ! В HELP по примитиву INDUCTOR указан диапазон от 0 – это неверно) до **1**, прописывается в окне **Other Properties** одной из индуктивностей так, как показано на Рис. 78 *(пример из **Mutual1.DSN**)*. Хотелось бы в данном примере особо обратить Ваше внимание на следующее:

- Поскольку модель индуктивности имеет нулевое собственное сопротивление, а в качестве источника сигнала используется генератор напряжения – обязательно присутствие добавочного резистора, имитирующего сопротивление катушки постоянному току – здесь это **R1** сопротивлением **1E-3** (1 миллиОм);
- Незакрашенной точкой у моделей индуктивностей помечены однополярные (синфазные) концы, т.е. в данном примере напряжение по фазе на входе и выходе будет совпадать, если необходимо противофазное напряжение, достаточно перевернуть включение одной из катушек. Аналогичного эффекта можно добиться использованием отрицательного значения коэффициента связи, однако при большом количестве взаимосвязанных катушек легко запутаться со знаками.
- Коэффициент связи указывается только у одной из взаимосвязанных катушек по отношению к другой (другим).

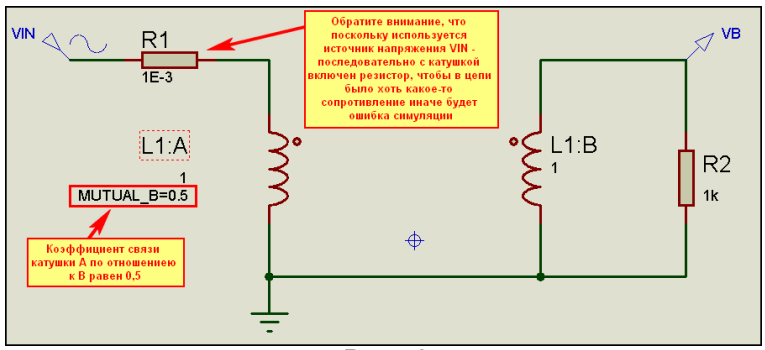


Рис.78

Вариант с несколькими катушками приведен в примере **Mutual2.DSN**. Там у катушки **L1:A** заданы коэффициенты к **L1:B** и **L1:C**, а у катушки **L1:B** уже только к **L1:C**. Хотелось бы также обратить внимание на то, как в этом примере задан коэффициент связи **K** – он задан текстовым скриптом *(левое меню **Text Script Mode**)*, помещенным в поле проекта и начинающимся со строки ***DEFINE**. При создании схематичных моделей этот вариант задания свойств мы будем активно использовать. Еще в этом примере хотелось бы заострить Ваше внимание на использовании в аналоговом графике желтой кривой **VSEC**. Ее значение вычисляется как разность напряжений зондов **VC-VB** – вот и получили на графике удвоенную амплитуду *(т.е. сумму напряжений двух вторичных секций трансформатора)*.

Ну и теперь немного остановимся на моделях трансформаторов из библиотеки **Inductors\Transformers**. Как Вы, наверное, догадались, анализируя примеры **Mutual**, магнитными свойствами сердечника там и близко не пахнет. Иными словами – все это линейные индуктивности и трансформаторы на их основе будут вести себя также. Однако в моделях есть ряд индуктивных элементов, реализованных схематичными моделями – в названии присутствует **SAT** от английского **Saturated** – насыщенный. В них предпринята попытка реализовать нелинейные свойства сердечника схемным путем. При этом следует учитывать, что керн трансформаторов **TRSAT** или катушки (модель **SATIND**) также является выводом и не должна «висеть в воздухе», иначе модель будет вести себя неадекватно. За счет введения нелинейных элементов данные модели более приближены к реальности, однако следует учитывать, что и параметров, которые для них задаются тоже намного больше. На Рис. 79 приведены для сравнения модели простого трансформатора и насыщенного, подключенные к одному синусоидальному генератору. На графиках напряжений сразу бросается в глаза, что входное и выходное напряжения простого трансформатора практически совпадают по фазе, а насыщенного значительно отличаются. Для любителей «покопаться во внутренностях» я для **TRSAT2P2S** восстановил внутреннюю структуру модели из файла MDF и приложил два примера в папке **TRANS_SAT**. Пример **MODEL_IN.DSN** предназначен для просмотра структуры трансформатора. В примере **EX_SAT_2.DSN** эта структура присоединена к графической модели в качестве дочернего листа. При этом можно тестировать модель, а на дочернем листе изменять схему и параметры. С дочернего листа и компилируется **MDF** файл. Чуть позже мы займемся этим более подробно.

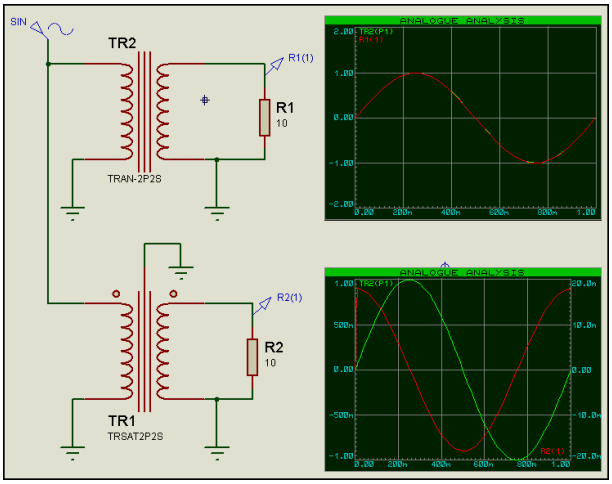


Рис.79

Пример внутренней структуры на картинке не привожу, поскольку она довольно громоздкая. И в заключение по параметрам насыщенных трансформаторов. Их там довольно много, но в большинстве они совпадают с теми, которые приводятся в книжках по программам, основанным на **PSPICE** – OrCAD, Multisim, MicroCAP и т. д. для сердечников трансформаторов – **CORE**, ну может немного отличаются названия и единицы измерения. Разбирать их подробно здесь не вижу особого смысла, т.к. после всего вышеизложенного вряд ли найдутся энтузиасты моделирования реальных трансформаторов с сердечниками в Протеусе. Всегда это гораздо проще сделать в вышеперечисленных пакетах программ, где для таких целей имеются как встроенные библиотеки сердечников, так и возможность задать свои по справочным данным с помощью имеющихся в них встроенных программ **MODEL**. Это сбережет Вам и время и нервы. Хотя, кто знает – если разработчики Протеуса в ближайшее время сделают шаг в сторону **PSPICE** (что очень даже возможно), то в ближайших версиях программы вопрос использования трансформаторов на сердечниках в ISIS разрешится сам собой.

4.12. Утилиты командной строки для работы с библиотеками SPICE и MDF моделей в Протеусе.

Так как мы уже вплотную занялись изучением моделей компонентов представленных в библиотеках **ISIS**, настало время познакомиться с несколькими программами, которые представлены в папке **\BIN** Протеуса и значительно облегчат нам последующую жизнь. Их четыре:

GETMDF.EXE – извлечение ***.MDF** из библиотек ***.LML**;
GETSPICE.EXE – извлечение ***.MDF** из библиотек ***.SML**;
PUTMDF.EXE – создание (добавление) библиотек ***.LML** из ***.MDF** файлов;
PUTSPICE.EXE – создание (добавление) библиотек ***.SML** из ***.MOD** файлов;

Это все консольные приложения и запускать их лучше из окна командной строки, чтобы после выполнения можно было проконтролировать результат выполнения, так как запуск непосредственно из-под Винды приведет к автоматическому закрытию окна, и Вы не успеете отследить – что сделала утилита. Для тех, кто с компьютера «сдувает пыль» напоминаю, что в WinXP **Командная строка** находится во вкладке **Стандартные** через **ПУСК -> Все программы**. Для того чтобы не прописывать вручную длинные пути рекомендую данные утилиты скопировать в отдельную папку, размещенную в корневом каталоге любого жесткого диска. В примерах ниже я разместил их на **F:\ProtUtil**. Так как эти программы «самодостаточны», то кроме них туда будем копировать для «разборки» только файлы соответствующих библиотек.

Поскольку я, как и многие программисты в меру ленив, то создал простейший командный файл **Consol.bat**, который запускает сеанс DOS и выводит меня в нужный каталог, а на **Рабочий стол** поместил ярлык с путем к этому файлу. Содержание файла самое тупое и для варианта с диском **F:** выглядит так:

```
@ECHO OFF
REM Открываем консоль DOS для WinXP на диске C: выглядит так:
C:\WINDOWS\SYSTEM32\CMD.EXE
REM Переходим на нужный диск в данном случае на F:
F:
REM Переходим в нужный каталог (папку) в данном случае ProtUtil
CD \ProtUtil\
```

Я нарочно расписал все поподробнее и в разных строках, чтобы легко было сообразить – где поправить под себя. Конечно, можно усовершенствовать данный «шедевр»: ввести выбор запускаемой утилиты и добавить запуск утилит с требуемыми ключами. Может на досуге и сделаю, если не одолеет все та же лень. Для наших целей пока сойдет и так. Результатом выполнения этого файла, созданного в текстовом формате (**Notepad/Блокнот**) и измененным расширением (**.TXT** меняем на **.BAT**) будет открытие командной консоли и переход в нужный каталог. После чего если ввести имя файла утилиты без ключей выскочит соответствующая подсказка (Рис. 80).

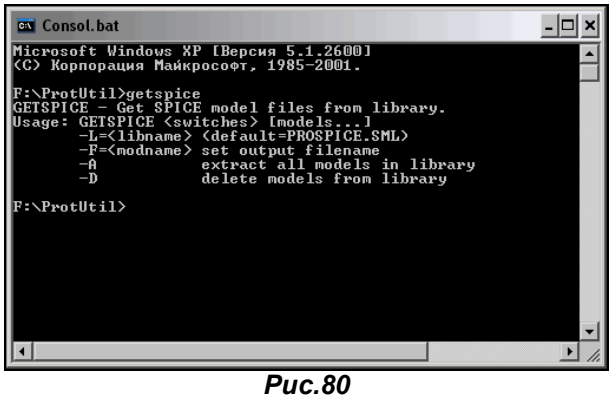


Рис.80

В данном случае имя утилиты getspice введено вручную. Я умышленно не ввел никаких ключей и после нажатия клавиши **Enter** получил подсказку по необходимым ключам.

Полезный совет для «чайников» в консольных программах. После выхода в ожидание ввода (нижняя строка на рисунке 80) чтобы не набирать тест повторно можно воспользоваться клавишами навигации стрелка вверх/стрелка вниз на дополнительной клавиатуре для повтора нужного ввода. Если было введено несколько команд, их можно будет пролистать, выбрать нужный ввод и скорректировать параметры. Этот выбор действует только в пределах данного сеанса, т.е. после закрытия консольного окна кликом по кресту **X** вверху справа введенные данные будут уничтожены и при следующем запуске недоступны.

В моем примере на Рис. 80 нажатие клавиши стрелка вверх вызовет повторное появление строки **getspice**, но теперь я добавлю нужные ключи в конце строки.

-L= <libname> – имя распаковываемой библиотеки ***.LML** (если не указан, то ищется**)**.

-F=<modname> – имя файла, в котором будут сохранены извлекаемый компоненты *(если ключ не введен, то модели будут сохранены в одноименный с **libname** файл, но с расширением .MOD)* .

-A – извлекает все модели из указанной библиотеки.

-D – стирает модели в библиотеке.

Так как в следующем разделе мы будем рассматривать модели диодов, то я, преследуя свои «шкурные интересы» изложения материала хочу достать SPICE-модель столь популярного у нас в России выпрямительного диода **1N4007**. Чтобы определить, где ее искать, выберем **1N4007** *(именно со SPICE-моделью! Не перепутайте, – там есть еще примитив)* и из библиотеки ISIS, поместим его в проект и в свойствах включим флаг **Edit all Properties as Text**. Имя библиотеки будет видно в соответствующей строке (Рис. 81).

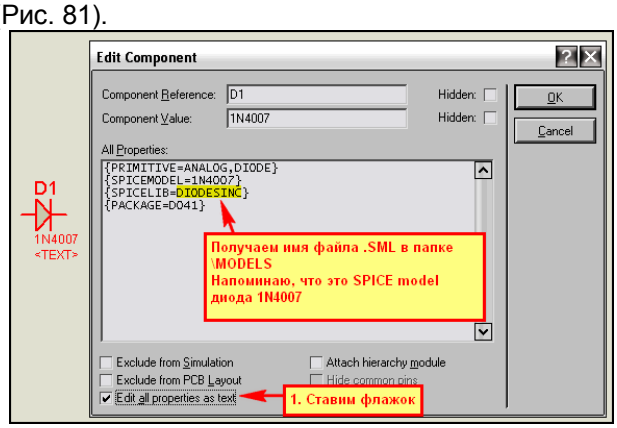


Рис.81

Теперь копируем файл **DIODESINC.SML** из папки **\MODELS** Протеуса в нашу папку и запускаем извлечение моделей с соответствующими ключами (Рис.82).

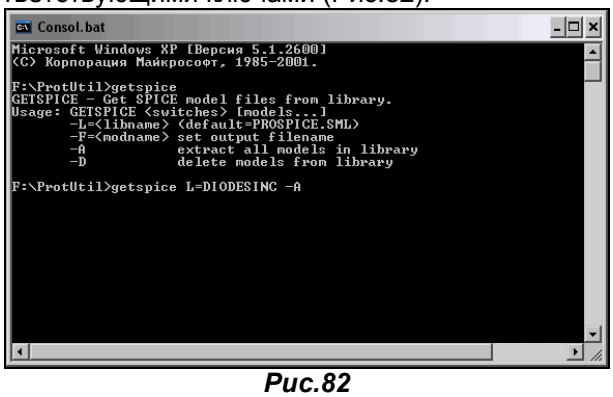


Рис.82

В данном случае я стрелкой вверх вызвал повтор команды getspice, но добавил к ней два ключа. Результатом будет извлечение всех моделей из библиотеки **DIODESINC.SML** в файл **DIODESINC.MOD**. Этот файл затем можно открыть в любом текстовом редакторе и найти там интересующую нас модель, воспользовавшись поиском. К сожалению **GETSPICE**, в отличие от **GETMDF** не раскладывает файлы при извлечении в отдельные, а просто разделяет их в текстовом файле пустыми строками. Итак, если все набрано правильно, наше окно прокрутилось, показав отчет об извлечении файлов, после чего его можно благополучно закрыть.

Полученный файл **DIODESINC.MOD** открываем в текстовом редакторе и находим поиском текст **1N4007,** строки будут выглядеть, как показано ниже:

```
*Object DIODESINC.SML/1N4007
.MODEL 1N4007 D  ( IS=76.9P RS=42.0M BV=1.00K IBV=5.00U CJO=26.5P  M=0.333 N=1.45 TT=4.32U )
```

Собственно вторая строка, начинающаяся с обязательной точки, и есть **SPICE** запись параметров нашей модели диода. Копируем ее в отдельный текстовый файл, даем ему название **1N4007.MOD** и сохраняем **SPICE**-модель для отдельно взятого диода.

Аналогично ведет себя и утилита **GETMDF**, но работает она уже с библиотеками **.LML**. На *Рис. 83* приведен пример извлечения с помощью ее моделей из библиотеки **ANALOG.LML**. Каждая модель в данном случае извлекается в отдельный файл с именем модели. После этого их также можно просматривать и править в текстовом редакторе. Именно так я «добыл» модель трансформатора в предыдущем разделе. Полученные при этом файлы **.MDF** - Model Description Format *(формат описания модели)* также можно открыть в любом текстовом редакторе. По своей структуре они практически совпадают с файлами списка цепей **.SDF**, которые мы рассматривали здесь ранее в разделе **4.4**.

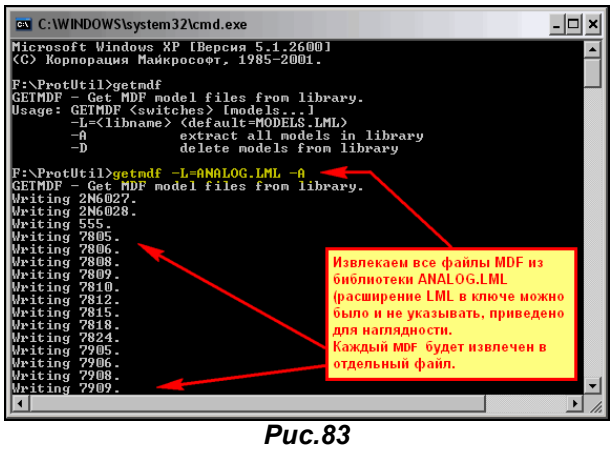


Рис.83

На следующем рисунке (Рис. 84) представлен запуск утилит **PUTSPICE.EXE** и **PUTMDF.EXE** без дополнительных ключей для получения подсказок. Как видно из рисунка, ключи у них по назначению совпадают, поэтому разберем на примере **PUTSPICE**.

-L= <libname> – как и у предыдущих утилит служит для задания имени библиотеки **SPICE** или **MDF**, которая создается или в которую добавляются файлы.

-C[=n] – служит для создания новой библиотеки с именем **<libname>** и опционально заданным количеством элементов **n**. Если **n** не указано, то создается пустая библиотека, если указано, то с резервированием на **n** элементов. На что здесь хочется обратить внимание. Если создана библиотека из 10 элементов, то в ней может находиться не более этого количества, меньше – сколько хотите. При попытке добавить 11-й элемент Вы получите сообщение: **Library is Full,** и элемент не добавится, поэтому при создании библиотеки лучше перестраховаться и задать **n** чуть больше, чем нужно, но и злоупотреблять не стоит – экономьте дисковое пространство.

-D – удаляет исходные файлы с диска при помещении их в библиотеку.

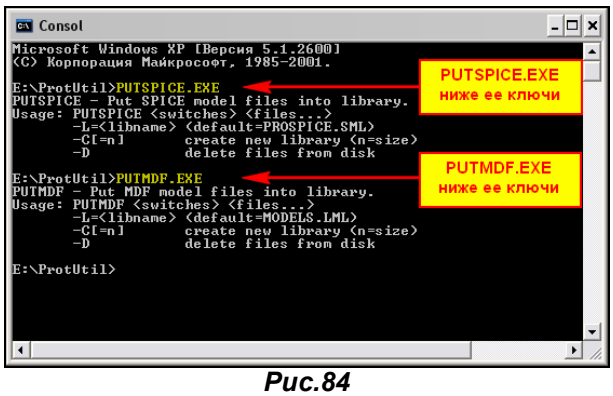


Рис.84

На Рис. 85 приведены примеры операций с утилитой **PUTSPICE.EXE**. Если кого-то смущает, что на последних двух картинках вместо диска **F:** указан диск **E:**, то это лишь потому, что данные иллюстрации делались на моем рабочем компе, а там на **F:** у меня сидит Линь. Поэтому пришлось задействовать **E:**. Надеюсь, особых затруднений этот материал у Вас не вызвал, а в дальнейшем он послужит хорошим пособием для создания собственных моделей и библиотек.

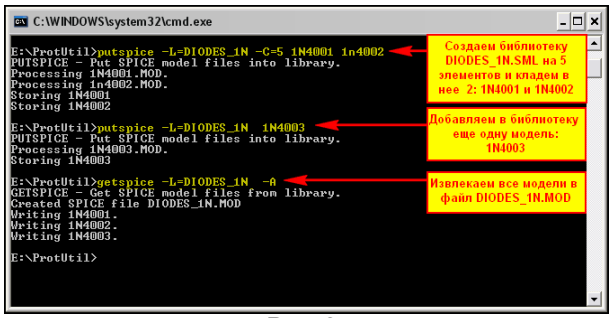


Рис.85

4.13. Примитивы управляемых ключей. Генераторы-двухполюсники.

Рассматривая набор примитивов, необходимый в дальнейшем для создания собственных моделей я чуть было не упустил еще две группы, широко используемые в стандартном SPICE моделировании. Давайте здесь их быстренько разберем.

Сначала познакомимся с управляемыми аналоговыми ключами **VSWITCH** и **CSWITCH** – соответственно первый контролирует входное напряжение (***Voltage***), а второй ток (***Current***). По своей сути они подобны электромагнитным реле. Рассмотрим свойства **VSWITCH** – *(Рис 86).*

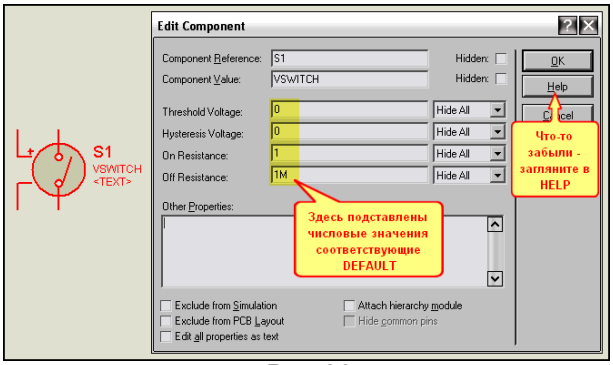


Рис.86

На рисунке вместо слова **DEFAULT** я подставил те числовые значения, которые ему соответствуют и назначены данной модели по умолчанию, т.е. когда вы только поместили ее в проект. Практически все свойства за исключением одного представлены в окне при вызове **Edit Properties**. Всегда можно посмотреть **Help**, чтоб чего-нибудь не напутать. Итак, свойства:

- **Threshold Voltage** – **VT** – по умолчанию 0 – пороговое входное напряжение срабатывания ключа;
- **Hysteresis Voltage** – **VH** – по умолчанию 0 – напряжение гистерезиса (запаздывания) срабатывания ключа. Вот здесь прошу всех быть предельно внимательными – в HELP ошибка. На самом деле ключ включается при **VT+VH** и выключается при **VT-VH** – это видно из следующего графика на *Рис. 87*. *Почему-то в **Help** прописаны значения **VH/2** – это бред.*
- **On Resistance** – **RON** – по умолчанию 1 Ом – выходное сопротивление ключа во включенном состоянии.
- **Off Resistance** – **ROFF** – по умолчанию 1 МОм – выходное сопротивление ключа в выключенном состоянии.

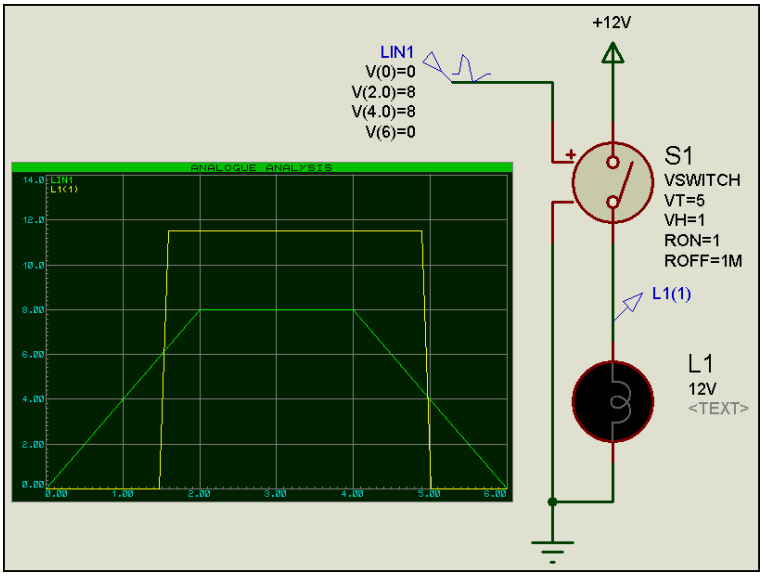


Рис.87

Ну и два логических параметра, которые не представлены в окне **Properties** – это **ON** (по умолчанию **FALSE** - ложно) и **OFF** (тоже по умолчанию **FALSE**) аналогичны рассмотренному раньше **Inicial Condition** – стартовому состоянию при начале симуляции. Вряд ли они вам пригодятся на практике, но всякое бывает. Куда интереснее следующий момент – как заставить ключ вести себя наоборот, т.е. действовать не на включение, а на выключение (аналог нормально замкнутых контактов реле). Все очень просто меняем значения **RON** и **ROFF** местами и получаем требуемое.

Все вышесказанное про **VSWITCH** применимо и **CSWITCH**, только там параметры срабатывания и гистерезиса относятся к току. Есть только одно существенное замечание – **CSWITCH** по входу обладает нулевым сопротивлением, поэтому при использовании его в проекте не забудьте включить нагрузочный резистор на входе во избежание ошибки симуляции *(Рис. 88).*

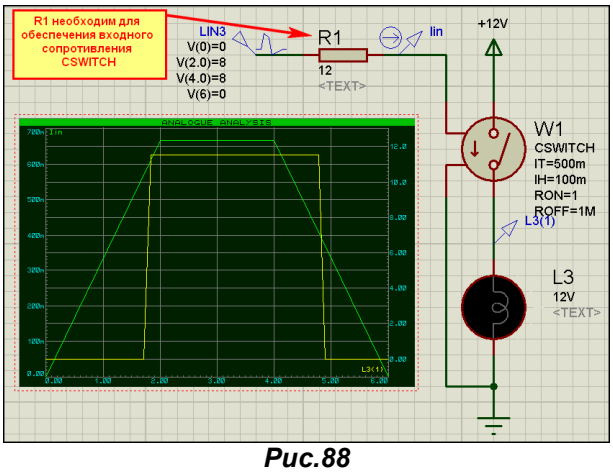


Рис.88

Ну и совсем кратко по двухполюсникам-генераторам. По иронии судьбы в библиотеках Протеуса их надо искать не в **Modelling Primitives**, где вроде бы им самое место, а в **Simulator Primitives => Sources** (Источники) – *Рис.89*.

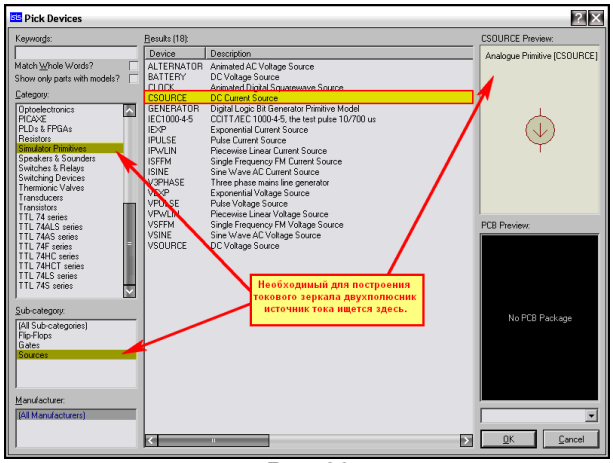


Рис.89

Почему я хочу на них заострить внимание. Да потому, что для построения даже самого простого каскада токового зеркала – типичный вход операционного усилителя необходим источник тока. Для примера приведу рисунок из книги О. Петракова с моделью 140УД7 – *Рис. 90* . Как видите, в модели применяются еще и двухполюсники напряжения в большом количестве. Они также как и **CSOURCE** расположены в этом разделе библиотек Протеуса на рисунке 89 это нижняя строка в колонке **Device** – **VSOURCE**.

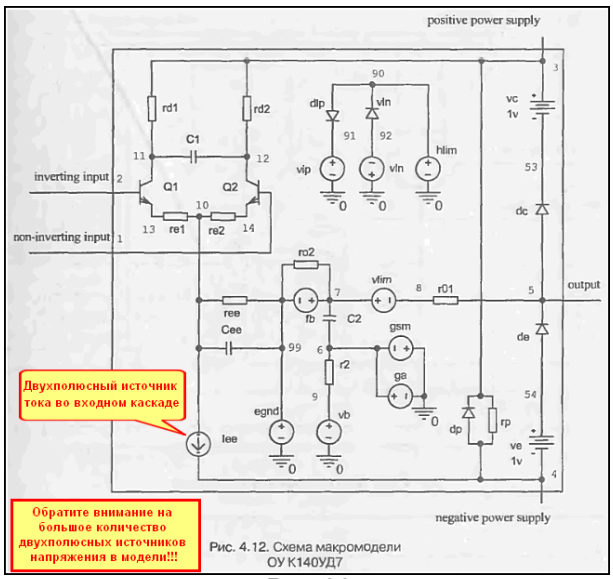


Рис.90

Пример применения ключей расположен в приложенном файле **Switches.RAR**, ну а применение источников надеюсь вопросов не вызовет – там все просто, да и все равно позже мы их будем использовать при моделировании, тогда и проявятся особенности.

4.14. Примитив диода и его параметры. Параметры реальных SPICE-моделей диодов. Вольтамперная характеристика моделей диодов на графике. Другие характеристики диодов.

Ну, вот и добрались мы до моделей активных компонентов. И начнем знакомство с примитива диода. Модель диода, расположенная в **Modelling Primitives => Analog(SPICE)** может быть использована для моделирования, как обычных диодов, так и их модификаций: стабилитронов, варикапов и пр. Если заглянуть в окно свойств модели (Рис. 91), то окажется, что большинство их спрятано в раскрывающемся списке **Advanced Properties**.

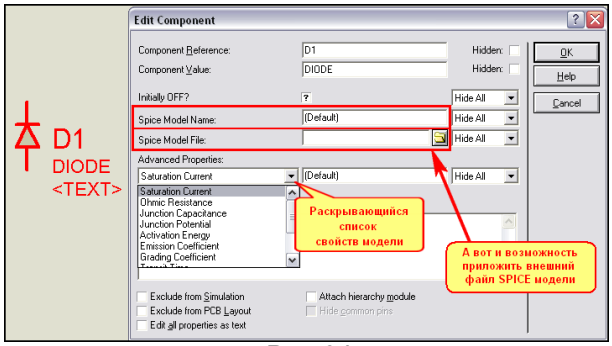


Рис.91

Ниже я перечислю все эти свойства с указанием их **Default** (по умолчанию) значений. Все эти свойства, за исключением первых двух, стандартны для большинства **SPICE**-симуляторов и встречаются при моделировании диодов не только в Протеусе, но и в OrCAD, Multisim и… «иже с ним».

- **Initially OFF** **OFF** **(-)** Начальное состояние *(изменяется кликом по знаку вопроса: вопрос – не определено, пусто – выкл., галочка – вкл.)*.

Часть параметров, которые можно задать вручную в окне Other Properties:

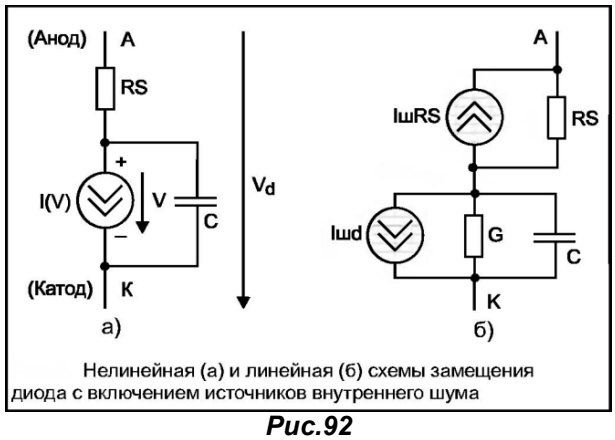
- **Initial device voltage** **IC** **(-)** Начальное напряжение *(можно задать только вручную, в списке его нет, т.к. это фишка Протеуса – еще одна трактовка IC)*.
- **Instance temperature** **TEMP** **(27)** Реальная температура диода [ºC] *(изменяется при температурном моделировании, как и для остальных моделей)*.
- **Parameter measurement temperature** **TNOM** **(27)** Температура измерений при создании модели [ºC].

- **Area factor** **AREA** **(1)** Число параллельных ветвей или масштабный множитель для параметров *(обычно увеличивается для мощных диодов, влияет на IS, RS, CJO)*.
- **Flicker noise coefficient** **KF** **(0)** Коэффициент фликер-шума.
- **Flicker noise exponent** **AF** **(1)** Показатель степени в формуле фликер-шума.
- **Forward bias junction fit parameter** **FC** **(0.5)** Коэффициент нелинейности барьерной емкости прямосмещенного перехода.

Параметры из раскрывающегося списка:

- **Saturation current** **IS** **(1e-14)** Ток насыщения перехода [A].
- **Ohmic resistance** **RS** **(0)** Объемное сопротивление [Ом].
- **Junction capacitance** **CJO** **(0)** Барьерная емкость перехода при нулевом смещении [Ф].
- **Junction potential** **VJ** **(1)** Контактная разность потенциалов [В].
- **Activation energy** **EG** **(1.11)** Энергетический барьер (ширина запрещенной зоны) [элВ]
- **Emission Coefficient** **N** **(1)** Коэффициент инжекции.
- **Grading coefficient** **M** **(0.5)** Коэффициент лавинного умножения.
- **Transit Time** **TT** **(0)** Время переноса заряда [сек].
- **Saturation current temperature exp.** **XTI** **(3)** Температурный экспоненциальный коэффициент тока насыщения IS.
- **Reverse breakdown voltage** **BV** **(∞)** Обратное напряжение пробоя [В].
- **Current at reverse breakdown voltage** **IBV** **(1mA)** Начальный ток пробоя, соответствующий **BV.**

Как видите, список задаваемых параметров достаточно велик и это не предел – для транзисторов будет еще больше. Параметры модели относятся к нелинейной и линейной схемам замещения диода (Рис. 92). На характеристики по постоянному току в основном влияют **IS**, **N**, **RS**. Для диодов Шотки параметр **EG** рекомендуется снизить до 0,69, а **N** повысить до 2. Емкостные характеристики (тех же варикапов или варакторов) в основном связаны с **CJO**, **VJ**, **M**. Для стабилитронов, использующих обратную ветвь вольтамперной характеристики, основными параметрами являются **BV**, и **IBV**.



Нелинейная (а) и линейная (б) схемы замещения диода с включением источников внутреннего шума

Рис.92

Тем, кто всерьез решил заняться SPICE-моделированием аналоговых компонентов и изучением их свойств рекомендую обратиться к соответствующей литературе. Список наиболее удачных русскоязычных публикаций я приведу в конце этого параграфа, хотя он и приводился раньше. Там вы сможете найти и кое-какие математические выкладки и рекомендации по моделированию. Этот список касается не только диодов, но и транзисторов к которым мы перейдем далее. Я же здесь не ставлю своей целью добросовестно «переписывать» чужие монографии. Свою задачу я вижу в том, чтобы показать – как это применимо в Протеусе.

Давайте еще раз вернемся к SPICE-модели **1N4007**, извлеченной нами в предыдущем параграфе и сравним ее с моделью **1N4006**, которая приведена этажом выше:

```
.MODEL 1N4006 D  (IS=76.9P RS=42.0M BV=800    IBV=5.00U CJO=26.5P M=0.333 N=1.45 TT=4.32U)

.MODEL 1N4007 D (IS=76.9P RS=42.0M BV=1.00K  IBV=5.00U CJO=26.5P M=0.333 N=1.45 TT=4.32U )
```

Как видим, две модели отличаются только предельным обратным напряжением **BV**, что и по справочнику соответствует действительности: у **1N4006** оно 800В, а у **1N4007** – 1000В. Ниже я привел модель наиболее близкого по параметрам нашего КД209А, позаимствованную из приложения к книге О. Петракова. Отмечу, что значок **+,** с которого начинается вторая строчка, означает продолжение первой.

```
.MODEL KD209A D (IS=6.22e-11 N=1.23 RS=0.17 CJO=16.2 M=0.35 TT=7.21E-7
+          VJ=0.68 BV=600 IBV=1E-10 EG=1.11 FC=0.5 XTI=3)
```

Итак, у нас есть SPICE-модели диодов, с которыми мы можем начать наши «опыты». Если кто-то уже заглядывал в **SAMPLES\Graph Based Simulation\Sweep.DSN**, то, наверное, видел пример построения вольтамперной характеристики диода. Займемся этим и мы. Для начала создадим

небольшую библиотеку – тестовый файл (можно в стандартном Блокноте/NotePad или в своем любимом текстовом редакторе ну уж только не в Word-е – это как говорил Матроскин перебор, хотя при некотором «изголении» можно и в нем). Файлу дадим расширение **.LIB** (в принципе поддерживается еще и **.INC**) и сохраним его в отдельной папке для экспериментов у меня это папка **Diode** в приложенном архиве. Мой файл будет выглядеть так:

```
*В строчках, начинающихся со звездочки пишутся комментарии, чем я и воспользуюсь в этом файле
*Я назову его Diodes.LIB, но вообще Протеус кушает еще и расширение .INC
*Этот файл должен лежать в папке нашего проекта и путь к нему должен быть относительным
* Т.е. в графе SPICE Model File должно быть указано коротко Diodes.LIB, а не абсолютным как ниже
* c:/Это дяди Петина папка/Это мой любимый каталог/Здесь мне хочется поиграться/Diodes.LIB
*
*
*Это модели из файла DIODESINC.MOD который мы получили с помощью GetSPICE.EXE ранее
*Модель 1N4005 совпадает по обратному напряжению с КД209А
.MODEL 1N4005 D  ( IS=76.9P RS=42.0M BV=600 IBV=5.00U CJO=26.5P  M=0.333 N=1.45 TT=4.32U )
*Модель1N4007  с Uобр=1000В просто я их покупаю пачками и тыкаю везде
.MODEL 1N4007 D  ( IS=76.9P RS=42.0M BV=1.00K IBV=5.00U CJO=26.5P  M=0.333 N=1.45 TT=4.32U )
*
*
*А эту модель я позаимствовал из приложения к книге О. Петракова
*Мы ее тоже покрутим в наших экспериментах
.MODEL KD209A D (IS=6.22e-11 N=1.23 RS=0.17 CJO=16.2 M=0.35 TT=7.21E-7
+            VJ=0.68 BV=600 IBV=1E-10 EG=1.11 FC=0.5 XTI=3)
```

Все, что начинается со звездочки в этом файле, является комментариями и может содержать любой текст. Сами описания моделей начинаются с точки, за которой следует слово MODEL, затем имя модели и в круглых скобках параметры. Если описание параметров модели не помещается в одну строку, то продолжение начинается со знака плюс (как у KD209). И еще одно существенное замечание – поосторожнее с русскими символами – только в комментах. Если кому-то надо более подробную информацию – в любую литературу по SPICE, в том числе и PSPICE – я предупреждал – плагиатом заниматься не намерен.

Теперь нам необходим дизайн Протеуса, который мы тоже сохраним **в той же папке**. Возьмем из **Modelling Primitives => Analog(SPICE)** нашу модель **DIODE** поместим в проект и практически воспроизведем тот вариант, что в **SAMPLES\....\ Sweep.DSN** (Рис. 93).

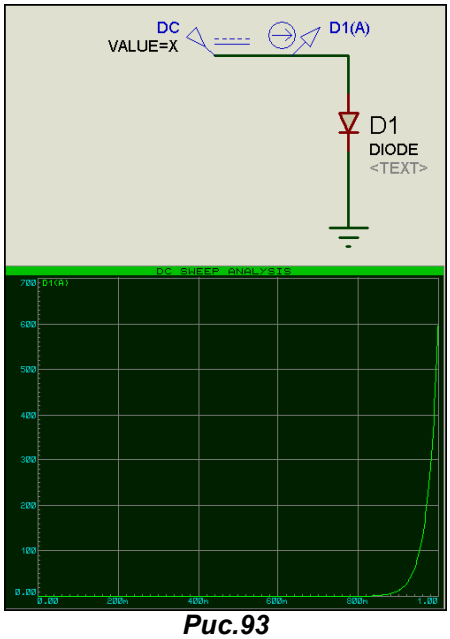


Рис.93

Здесь уместны несколько замечаний:

- Пока все параметры диода оставлены по умолчанию.
- При установке генератора (имя **DC** я ему присвоил вручную) необходимо зайти в его **Properties** (Параметры) и установить галочку **Manual Edits?** Затем в окне **Properties** присвоить **VALUE=X**. Напомню, что **X** – это параметр графика **DC SWEEP**, который мы будем использовать. Можно, конечно, вместо **X** использовать собственное имя на латинице, но тогда будьте любезны в свойствах графика заменить **X** на это имя. Кстати в стандартном примере так и сделано используется **V**. Теперь о том, что касается галочки **Manual Edits?** Если вы ее не поставите, то Протеус будет ругаться на присвоение значения **X** параметру **Voltage** для генератора, т.к. оно должно быть числовым.
- Ну и последнее – несущественное, но полезное. Обратите внимание, что я снял фигурные скобки вокруг **VALUE=X**. Если вы добавите еще генератор и поставите **Manual Edits?**, то в окне **Properties** значение **Value** будет в фигурных скобках (скрытым – **Hide**). Я убрал скобки, и оно стало видимым. Это относится ко всем параметрам любых моделей. При

ручном редактировании просто убираете фигурные скобки у тех параметров, которые надо «высветить» в проекте, или наоборот ставите тем, которые надо спрятать.

На рис. 93 график **DC SWEEP** тоже с параметрами по умолчанию, т.е. его параметры оставлены такими, как на Рис. 94, а в качестве трассы помещен токовый пробник **D1(A)**.

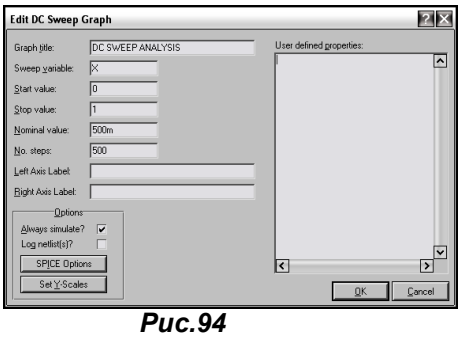


Рис.94

На графике мы имеем прямую ветвь вольтамперной характеристики (ВАХ) модели с параметрами по умолчанию. Теперь добавим в дизайне две аналогичные схемы с графиками. Проще всего это сделать, выделив все имеющееся (вместе с графиком) и использовав **Block Copy** (верхнее меню или по правой кнопке мыши). Во вновь созданных блоках нам придется проделать следующие манипуляции:

- Для диодов (которые автоматически станут **D2** и **D3**) зайти в свойства (**Properties**) и назначить файл и модель. Для **D2** – так как на Рис. 95, а для **D3** аналогично, но в графе модель присвоить **KD209A**. Обратите внимание, что путь к файлу библиотеки практически не указан, т.е. он лежит в одной папке с проектом. Протеус не очень лояльно относится к длинным абсолютным путям, но стопроцентно работает следующим образом: проверяет папку с текущим проектом, затем проверяет стандартные пути в своем **System=>Set Paths**. Конкретно для моделей там указан *(ну если уж совсем по умолчанию)*:
  C:\Program Files\Labcenter Electronics\Proteus 7 Professional\MODELS
  И если вашего файла с описанием модели там не окажется, то он Вас пошлет и достаточно далеко… Пойдете?
- В графиках и генераторах изменить **Sweep Variable** и **Value** соответственно, чтобы не повторялся скопированный оригинал. Допустим для второго вместо **X** назначить **X1**, а для третьего **X2** – так сделано в моем примере.
- В графиках для второго и третьего варианта удалить трассы **D1(A)** и соответственно «втащить мышкой» или назначить ручками через правую кнопку **Add Trace** **D2(A)** и **D3(A).**

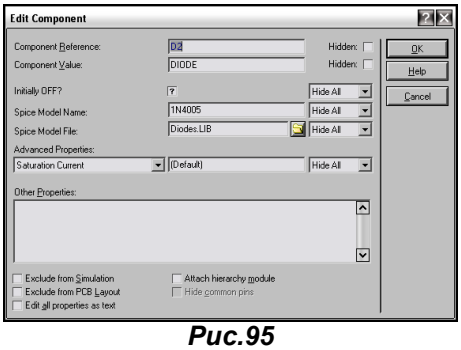


Рис.95

Затем пробуем запустить на исполнение графики для второго и третьего варианта. В результате у меня график диода **KD209A** заругался, а именно на параметр **IBV** после некоторой коррекции в сторону загрубения до **IBV=2E-6** ругань прекратилась. Результат для различных вариантов моделей на Рис. 96.

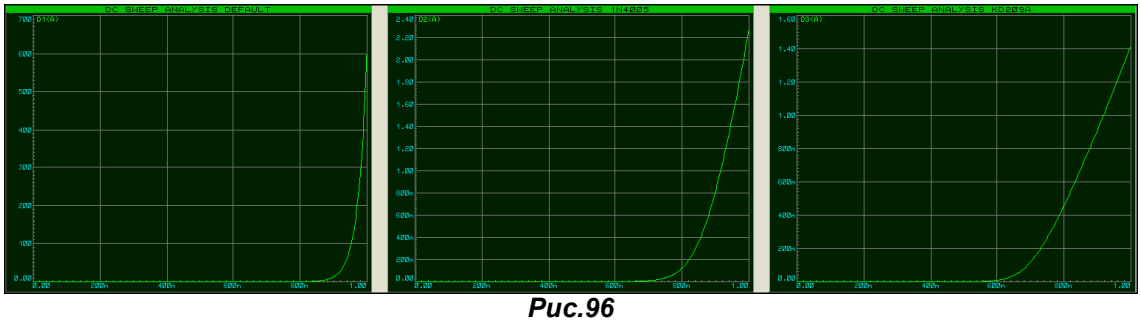


Рис.96

Как видно из рисунка изменение параметров работает по полной программе. Данный проект **Diode1.DSN** приложен в архиве. Версия **7.6SP0** для более ранних - секции **.SEC**. Здесь мы исследовали только прямую ветвь ВАХ, поскольку рассматривались выпрямительные диоды.

Давайте теперь рассмотрим обратную ветвь ВАХ для стабилитронов. Я воспользуюсь опять-таки моделью О.Петракова для **Д814А** с Uст=7,8V Iст=0,5uA (напомню, что это параметры **BV** и **IBV**). В качестве других «подопытных кроликов» выберем из библиотеки полуваттный Моторолловский **1N5236B** с Uст=7,5В Iст=20mA, а третьей моделью будет «заготовка» стабилитрона расположенная в библиотеке **Diodes/Generic** которой мы присвоим в **Properties** Uст=7.8V Iст=0,5uA (как и у советского стабилитрона). Ну и наконец, пора приучаться к сторонним моделям, используем аналогичную по параметрам SPICE модель стабилитрона от **NXP** (бывш. **Philips**). Найти их модели диодов и стабилитронов можно здесь:

http://www.ru.nxp.com/models/spicespar/diodes.html

Мне приглянулась следующая - **BZX84C7V5**. Кстати, этот стабилитрон и так живет в библиотеках ISIS, но там он реализован именно через примитив **GENERIC**, а я воспользуюсь чистейшей SPICE моделью, да еще как говорится «от производителя»**.** Кликнув по соответствующей ссылке, я получил в открывшемся окне следующий чуть ниже SPICE-код модели, который скопировал как текст и вставил в файл **stab.LIB**. Там же у меня заранее был вставлен SPICE-код D814A. Правда, мне пришлось немного «поколдовать» с ним. Дело в том, что у модели О. Петракова указан **IBV=0.5u**, ну или в переводе на наш «птичий» 0,5мкА. Не знаю, чем руководствовался автор, задавая такой маленький ток, но ISIS это жутко не понравилось, т.к. при стандартных установках улетает за допустимые пределы. Пришлось «покривить душой» и поставить 5мкА, ну или в переводе на ихний **IBV=5u**, после этого горчичники прекратились. Меня и этот вариант устраивает для наших учебных целей. Итак, вся эта кухня имитируется в прилагаемом дизайне **Diode2.DSN**. Кстати там, как сказал почтальон Печкин, - «с целью расширения кругозора» в качестве генераторов я поставил двухполюсники **VSOURCE**, описанные в предыдущем параграфе. Еще один существенный момент – в графиках я вручную ограничил верхний и нижний пределы оси **Y** в нужном мне диапазоне – от минус **50mA** до плюс **50 mA**, чтобы графики выглядели в приемлемом масштабе (автоматом ISIS там наставит, чуть ли не кило-Амперы и нужный для нашего исследования диапазон будет не виден, зато будет виден, например, ток при 2V прямого напряжения при котором реальный диод просто «взорвется»).

Не верите? Маленькое лирическое отступление или экскурс в историю радиолюбительского движения СССР. Будучи совсем молодыми, по сути, пионерами, в нашем радиоклубе мы занимались следующим «хулиганством»: на сетевую вилку паяльника приматывался параллельно точечный германиевый диод Д9 – были такие в советские времена. Если тот, кто садился после тебя за монтажный стол не проверил вилку паяльника и что воткнуто в сетевые розетки, а просто врубил сеть тумблером – получался маленький «Бух!!!». Аналогичный большой «Бух!!!» достигался закладыванием в ящик стола электролитического конденсатора, также присоединенного к сетевой розетке.

Обратите внимание и на то, как прописаны в свойствах **D3** и **D4** пути к моделям. Собственно сами модели лежат в файле **stab.LIB**, который можно посмотреть в любом текстовом редакторе. Ну и еще посмотрите на то, как прописан код от **NXP** – они каждый параметр пишут в отдельной строчке начинающейся с +. Я не стал менять их вариант, чтобы показать, что и это работает, только места больше занимает.

```
*DEVICE=BZX84C7V5,D
* BZX84C7V5 D model
* created using Parts release 7.1 on 03/31/98 at 08:46
* Parts is a MicroSim product.
.MODEL BZX84C7V5 D
+ IS=2.6665E-18
+ N=.82284
+ RS=.51617
+ IKF=11.760E-3
+ CJO=63.513E-12
+ M=.33559
+ VJ=.66795
+ ISR=1.1222E-9
+ BV=7.6329
+ IBV=.94329
+ TT=2.7411E-6
*$
```

В примере **Diode3.DSN** я оставил только два стабилитрона и положил трассы в одном окне, чтобы видеть отличие заданного вручную **GENERIC** стабилитрона от модели **D814A**.

Ну и в заключение этой темы попробуем смоделировать в Протеусе другие характеристики диодов. В частности, воспользуемся материалом от О. Петракова и получим время обратного восстановления (важная характеристика импульсных диодов) для диода **1N4148** и нашего **КД522А** (опять модель О. Петракова добавлена в **Diodes.LIB**). Этот вариант в приложенном **Diode4.DSN**.

Теперь получим вольт-фарадную характеристику. Здесь придется слегка «пофантазировать». Оставляем все выкладки г. Петракова в силе. А именно так как на Рис. 97:

Отсюда получим формулу для емкости.

$$Сд = \frac{Iд}{10^7}$$

или окончательно

Сд(пф) = 0,1Iд (мкА)

Рис.97

Теперь подаем от генератора обратное, линейно нарастающее напряжение в диапазоне до 50V, затем воспользуемся окончательной формулой, где Сд=0,1Iд и во втором графике введем этот коэффициент для трассы, т.е. поделим ток на 10, чтобы получить значение емкости в пФ. Результат этих преобразований в файле **Diode5.DSN**.

Ну а теперь подведем итог изложенному материалу:

1. Протеус, а именно ISIS, благополучно поддерживает моделирование SPICE и применение SPICE-моделей внутри программы. Для того чтобы свободно использовать SPICE-моделирование, Вам придется изучить дополнительные материалы. Эти книги отнюдь не связаны с Протеусом напрямую, но … «SPICE он и в Африке SPICE». Вот обещанный перечень книг, которые я рекомендую держать «под рукой» с моими примечаниями в скобках:
   - Разевиг В. Д. **«Система сквозного проектирования электронных устройств DesignLab 8.0»**, М.: Солон, 1999. *(Ну, в общем, то первоначальная теория была изложена здесь, и поскольку я с некоторым «благоговением» отношусь к книгам Всеволода Даниловича, то не мог пройти эту стороной).*
   - Разевиг В. Д. **«Схемотехническое моделирование с помощью MICRO-CAP 7»** М.: Горячая линия-Телеком, 2003*. (Более слабая книга, в основном все направлено на именно MICRO-CAP, но как руководство к действию – рекомендую)*.
   - Разевиг В. Д. **«Система проектирования OrCAD 9.2.»** М.: СОЛОН-Р, 2003. *(Все комментарии, как и в предыдущем варианте, только касаются OrCAD)*.
   - Амелина Петраков М. А., Амелин С. А. **«Программа схемотехнического моделирования Micro-Cap 8»** М.: Горячая линия-Телеком, 2007. *(Ну, что-ж, Всеволод Данилович к тому времени «безвременно покинул нас» в 2004 году, а эта книга добросовестно поддерживает то, что начал он – очень рекомендую, поскольку теоретические основы базируются на его выкладках и совместимы с первоисточником)*.

Ну и конечно материал от О. Петракова:

   - Петраков О. М. **«Создание аналоговых PSPICE-моделей радиоэлементов».** М.: ИП РадиоСофт, 2004.
   - Петраков О. М. **«Создание аналоговых PSPICE-моделей радиоэлементов».** Цикл статей в журнале «Схемотехника» за 2001-2002 г.г.

2. Мы в этом параграфе научились «приклеивать» SPICE-модели к нашим вариантам – все, что требуется – создать ссылку на файл с моделью и указать имя модели. SPICE-модель компонента должна лежать в файле с расширением .**LIB** (от library – библиотека). Файл библиотеки должен находиться либо в файле с проектом, либо в папке **MODELS** установленного Протеуса.
3. В папках **GENERIC** (дословный перевод – универсальные) для нужной нам модели лежат «заготовки», т.е. шаблоны моделей, которые при задании собственных свойств можно использовать в своем проекте.
4. **SPICE**-модели (именно их, а не все, что захотелось бы неискушенному пользователю, можно поискать на сайте производителей компонентов). Ключевым словом для поиска являются **SPICE** или **model (models)**. Не путайте с IBIS и прочими моделями, в Протеусе безболезненно вы можете использовать **SPICE** или **PSPICE**. В последнем случае необходимо внимательно изучить содержимое на предмет совместимости с **SPICE3F5**.
5. Ну и в следующем параграфе мы потренируемся в создании собственных библиотек и узнаем – где их искать (про **NXP** теперь уже знаем).

4.15. Биполярный транзистор и получение его характеристик. Создание графика семейства выходных характеристик на основе TRANSFER.

В качестве модели для биполярных транзисторов Протеус использует модель Гуммеля-Пуна (Рис. 98), которая при некотором сокращении параметров может быть приведена к более известной у нас модели Эберса-Молла.

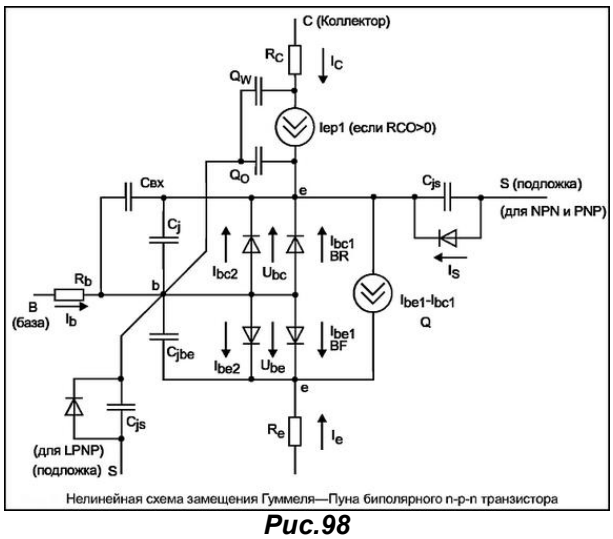


Рис.98

Ниже, как и для диода, я приведу полный список параметров, который заложен в **ISIS** для биполярных транзисторов, в том числе, как всегда с параметрами, присущими только Протеусу.

- **Initially OFF** **OFF** **(-)** Как и для диода, начальное состояние *(изменяется кликом по знаку вопроса: вопрос – не определено, пусто – выкл., галочка – вкл.)*.

Параметры из раскрывающегося списка:

- **Ideal forward beta** **BF** **(100)** Максимальный коэффициент усиления тока в нормальном режиме в схеме с ОЭ;
- **Saturation Current** **IS** **(1e-016)** Ток насыщения при температуре 27°C (А);
- **Forward emission coefficient** **NF** **(1)** Коэффициент эмиссии (неидеальности) для нормального режима;
- **Forward Early voltage** **VAF** **(∞)** Напряжение Эрли в нормальном режиме (В);
- **Forward beta roll-off corner current** **IKF** **(∞)** Ток начала спада зависимости BF от тока коллектора в нормальном режиме (А);
- **B-E leakage saturation current** **ISE** **(0)** Ток насыщения утечки перехода база-эмиттер (А);
- **B-E leakage emission coefficient** **NE** **(1.5)** Коэффициент эмиссии тока утечки эмиттерного перехода;
- **Ideal reverse beta** **BR** **(1)** Максимальный коэффициент усиления тока в инверсном режиме в схеме с ОЭ;
- **Reverse emission coefficient** **NR** **(1)** Коэффициент эмиссии (неидеальности) для инверсного режима;
- **Reverse Early voltage** **VAR** **(∞)** Напряжение Эрли в инверсном режиме (В);
- **Reverse beta roll-off corner current** **IKR** **(∞)** Ток начала спада зависимости BR от тока эмиттера в инверсном режиме (А);
- **B-C leakage saturation current** **ISC** **(0)** Ток насыщения утечки перехода база-коллектор (А);
- **B-C leakage emission coefficient** **NC** **(2)** Коэффициент эмиссии тока утечки коллекторного перехода;
- **Zero bias base resistance** **RB** **(0)** Объемное сопротивление базы (максимальное) при нулевом смещении перехода база-эмиттер (Ом);
- **Minimum base resistance at high currents** **RBM** **(RB)** Минимальное сопротивление базы при больших токах (Ом);
- **Current for base resistance=(rb+rbm)/2** **IRB** **(∞)** Ток базы, при котором сопротивление базы уменьшается на 50 % полного перепада между RB и RBM (А);
- **Emitter resistance** **RE** **(0)** Объемное сопротивление эмиттера (Ом);
- **Collector resistance** **RC** **(0)** Объемное сопротивление коллектора (Ом);
- **Zero bias B-E depletion capacitance** **CJE** **(0)** Емкость эмиттерного перехода при нулевом смещении (Ф);
- **B-E built in potential** **VJE** **(0.75)** Контактная разность потенциалов перехода база- эмиттер;
- **B-E junction grading coefficient** **MJE** **(0.33)** Коэффициент, учитывающий плавность эмиттерного перехода;
- **Ideal forward transit time** **TF** **(0)** Время переноса заряда через базу в нормальном режиме (сек);
- **Coefficient for bias dependence of TF** **XTF** **(0)** Коэффициент, определяющий зависимость TF от смещения база-коллектор;
- **Voltage giving VBC dependence of TF** **VTF** **(∞)** Напряжение, характеризующее зависимость TF от смещения база-коллектор (В);

- **High current dependence of TF** **ITF** **(0)** Ток, характеризующий зависимость TF от тока коллектора при больших токах (А);
- **Excess phase** **PTF** **(0)** Дополнительный фазовый сдвиг на граничной частоте транзистора **f=1/2π*TF**;
- **Zero bias B-C depletion capacitance** **CJC** **(0)** Емкость коллекторного перехода при нулевом смещении (Ф);
- **B-C built in potential** **VJC** **(0.75)** Контактная разность потенциалов перехода база- коллектор;
- **B-C junction grading coefficient** **MJC** **(0.33)** Коэффициент, учитывающий плавность коллекторного перехода;
- **Fraction of B-C cap to internal base** **XCJC** **(1)** Доля барьерной емкости, относящаяся к внутренней базе;
- **Ideal reverse transit time** **TR** **(0)** Время переноса заряда через базу в инверсном режиме (сек);
- **Zero bias C-S capacitance** **CJS** **(0)** Емкость перехода коллектор-подложка при нулевом смещении;
- **Substrate junction built in potential** **VJS** **(0.75)** Контактная разность потенциалов перехода коллектор-подложка (В);
- **Substrate junction grading coefficient** **MJS** **(0)** Коэффициент, учитывающий плавность перехода коллектор-подложка;
- **Forward and reverse beta temp. exp.** **XTB** **(0)** Температурный коэффициент BF и BR;
- **Energy gap for IS temp. dependency** **EG** **(1.11)** Ширина запрещенной зоны (эВ);
- **Temp. exponent for IS** **XTI** **(3)** Температурный экспоненциальный коэффициент для тока IS;
- **Forward bias junction fit parameter** **FC** **(0.5)** Коэффициент нелинейности барьерных емкостей прямосмещенных переходов;
- **Flicker Noise Coefficient** **KF** **(0)** Коэффициент, определяющий спектральную плотность фликер-шума;
- **Flicker Noise Exponent** **AF** **(0)** Показатель степени, определяющий зависимость спектральной плотности фликер-шума от тока через переход.

Как я и предупреждал, количество параметров в раскрывающемся списке для транзистора очень большое, и в него попали почти все параметры модели за исключением следующих:

- **Initial B-E voltage** **ICVBE** **(-)** Начальное (стартовое) напряжение база-эмитер;
- **Initial C-E voltage** **ICVCE** **(-)** Начальное (стартовое) напряжение коллектор-эммитер;
- **Area factor** **AREA** **(1)** Множитель для коэффициентов, используемый при моделировании мощных транзисторов;

Ну и два, уже известных нам температурных коэффициента.

- **Instance temperature** **TEMP** **(27)**
- **Parameter measurement temperature** **TNOM** **(27)**

Количество параметров впечатляет, однако на практике, при моделировании используются они далеко не все. Для примера опять рассмотрим одну модель. Я выбрал BC177 – наш аналог КТ3107А опять-таки есть у О. Петракова, так что будет, с чем сравнивать. Попутно с помощью **TRANSFER** графика поучимся строить семейства характеристик. Аналогичный пример уже есть в **SAMPLES** Протеуса, и называется **Transfer.DSN**. Расположен он конечно-же в папке **Graph Based Simulation**. Там построено семейство характеристик для **BC108**. В библиотеках Протеуса есть два транзистора BC177 – один на основе примитива, а другой на основе SPICE-модели **Zetex**. Я собрал это все в один проект и туда же прилепил и модель КТ3107А от О. Петракова, которая выглядит так:

```
.model kt3107a  PNP  (Is=6.545f  Xti=3  Eg=1.11  Vaf=86.5  Bf=105.5
+   Ne=8.56 Mje=.35 Ise=7.735n Ikf=.186 Xtb=1.5 Var=32 Br=1.62 Nc=2
+   Isc=3.35p Ikr=12m Rb=39.1 Rc=.71 Cjc=12.8p Vjc=.65 Mjc=.33 Fc=.5
+   Cje=12.6p Vje=.69 Tr=30.5n Tf=477.5p Itf=56m Vtf=35 Xtf=2)
```

Итак, на Рис. 99 семейство характеристик, взятое из даташита от фирмы Siemens.

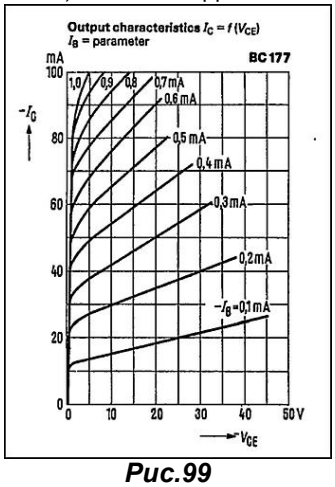


Рис.99

Для чистоты эксперимента я применил PNP транзистор, чтобы Вы могли сравнить – чем будет отличаться наш график от графика, приведенного в **SAMPLES** для NPN. Итак, строим ту же схему, что и приведенная в **Transfer.DSN** *(Рис. 100)*. Но, как мы все помним, полярность для PNP будет другая, поэтому – разворачиваем токовый зонд в другую сторону и задаем минусовые напряжения и токи. Обратите внимание, что в свойствах генератора, включенного в цепь базы необходимо поставить галку **Current Source** *(Источник тока)*.

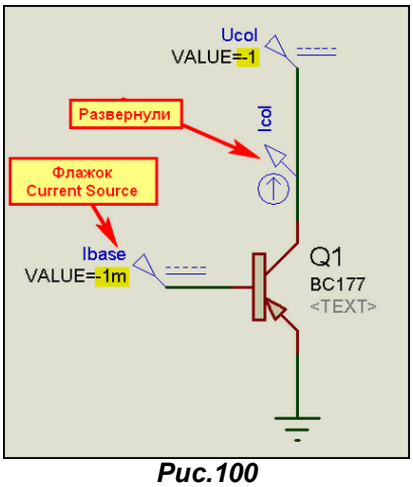


Рис.100

Затем добавляем в проект график **TRANSFER**. В его свойствах в качестве **Source1** выбираем коллекторный источник напряжения *(у меня это **Ucol**)* и задаем ему значения от 0 до -50V *(как на графике из даташита)*, а число шагов как можно больше для того, чтобы кривые получились плавными. В качестве **Source2** задаем базовый источник тока с пределами 100uA до 1mA и числом шагов 10 *(тоже, чтоб было похоже на график даташита)*. Для пущей важности я еще ограничил вручную шкалу **Y** как на том графике – от 0 до 100mA. Все это представлено на Рис. 101.

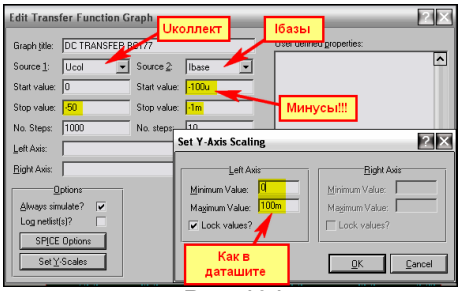


Рис.101

Теперь втаскиваем наш коллекторный зонд на поле графика и запускаем симуляцию графика. Если Вы нигде не ошиблись, то должны получить нечто как на следующем *Рис. 102.*

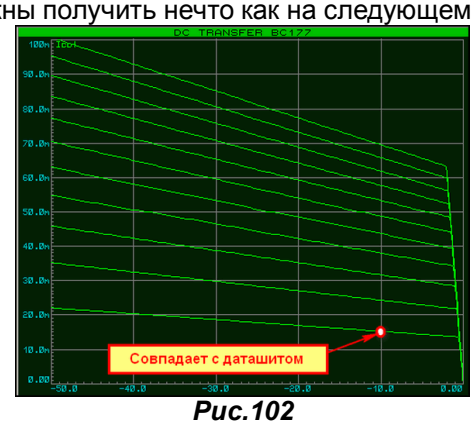


Рис.102

И вот тут нас поджидает «приятная неожиданность» - Протеус не умеет разворачивать горизонтальную шкалу, т.е. отрицательные значения всегда слева, поэтому наше семейство характеристик получилось в зеркальном отображении по сравнению с «книжным». Ведь в даташите в направлении слева направо указано **–Vce**. Но это замечание касается только **PNP** транзисторов, с **NPN** все будет выглядеть нормально. Правда, надо отдать должное – хотя бы для 100uA базы при -10V на коллекторе точки совпадают с даташитом, а вот для больших токов базы для этой модели явное расхождение. И еще один «неприятный» факт. Если я ставлю для **Ibase** количество шагов 10, то получаю 11 кривых, а если 9, то и получу 9. Ну никак не получается семейство из 10 кривых.

Ну и в заключение рассмотрения биполярных транзисторов я хочу привести еще один пример исследования зависимости напряжения насыщения от тока коллектора, рассмотренный в цикле

статей и книге О. Петракова и адаптированный для проведения исследования в ISIS. Для этого нам потребуется создать схему, изображенную на *Рис. 103*. Обратите внимание, что в качестве **I3col** использован генератор DC из левого меню с включенным флажком **Current Source**. Для изменения тока базы использован источник тока, управляемый током **CCCS** с коэффициентом передачи **0,1** *(т.е. ток базы будет в десять раз меньше тока коллектора)*.

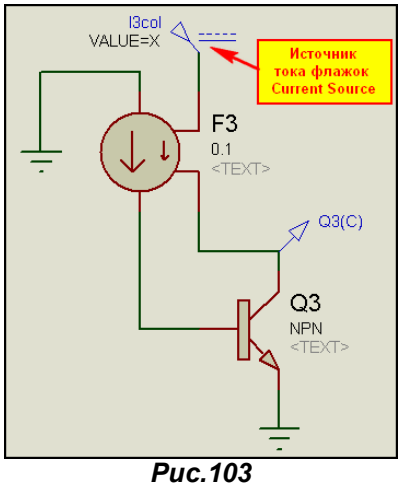


Рис.103

Далее используем уже знакомый нам график **DC SWEEP** для построения характеристики с параметрами указанными на *Рис. 104*. Параметры подобраны для максимального совпадения с моделью О. Петракова, а в качестве «подопытных кроликов» я использовал две модели из библиотек Протеуса для транзистора **BC337** и модель **KT315A** из статьи О. Петракова, присоединенную к примитиву **NPN** транзистора аналогично предыдущему примеру.

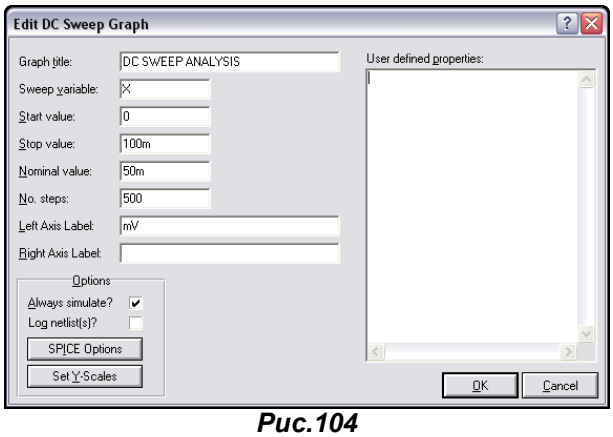


Рис.104

На *Рис. 105* приведен результат тестирования для транзистора КТ315А, проведенный в Протеусе вместе с картинкой из статьи О. Петракова. Как видим, результаты совпадают, что я и хотел подчеркнуть этим примером.

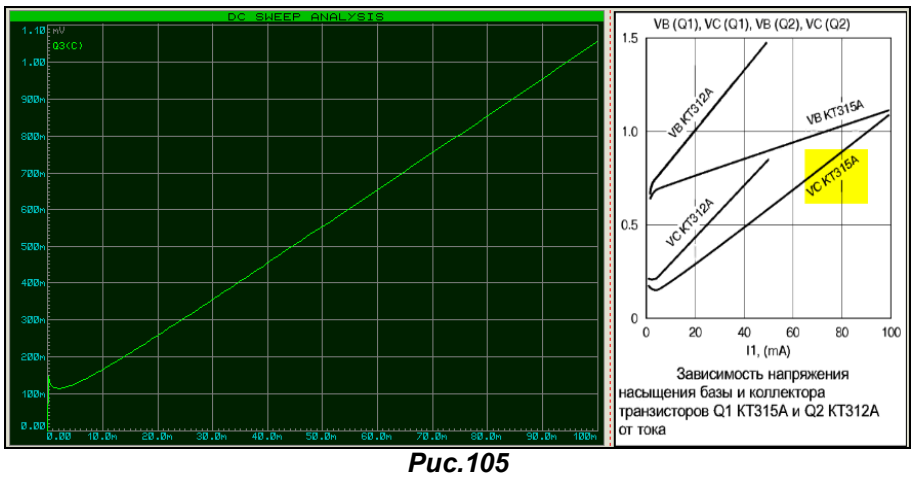


Рис.105

Но было бы нечестно с моей стороны не отметить, что модель КТ315А представленная на прилагаемом к книге диске и описанная в статье отличаются некоторыми параметрами, в частности

RC. Поэтому, для совпадения картинки из статьи мне пришлось его поставить таким, каким он приведен в тексте статьи и главы из книги, посвященной тестированию биполярных транзисторов.
Оба приведенные в этом параграфе примера находятся в соответствующих папках прилагаемого архива **Bipolar.rar**. Там же имеются файлы секций для импорта в предыдущие версии Протеуса.

4.16. Модели полевых транзисторов различных типов в ISIS а также немного про IGBT.

Все модели примитивов полевых транзисторов в ISIS можно разделить на три группы (Рис. 106).

- **JFET** – полевые транзисторы с управляющим PN-переходом.
- **MESFET** – арсенид-галлиевые полевые транзисторы. Обратите внимание, что в другой литературе по SPICE они носят название **GASFET**, а также на то, что в **ISIS** присутствует модель **PMESFET** не существующая в действительности.
- **MOSFET** – МОП-транзисторы с изолированным затвором, которые представлены двумя разновидностями: с четырьмя выводами (подложка изолирована) и тремя (подложка соединена с истоком).

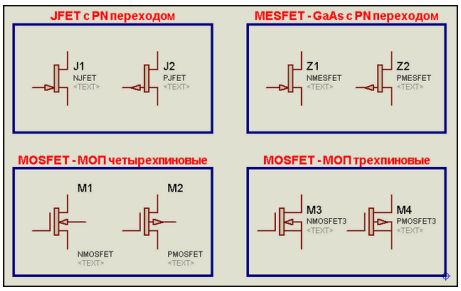


Рис.106

Рассмотрение параметров различных групп начнем с JFET-транзисторов с управляющим PN-переходом, которые базируются на модели Шихмана-Ходжеса (*Рис. 107)*.

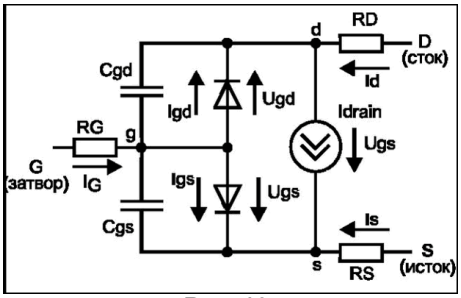


Рис.107

Поскольку большинство названий параметров для **JFET** и **MESFET** совпадают, а отличаются только значения по умолчанию для моделей **MESFET,** в следующем списке будут выделены зеленым цветом, а **JFET** – красным.

Параметры характерные только для ISIS:

- **Device initially off** **OFF** **(-)** Начальное состояние модели.
- **Initial D-S voltage** **IC-VDS** **(-)** Начальное напряжение исток-сток.
- **Initial G-S voltage** **IC-VGS** **(-)** Начальное напряжение исток-затвор.
- **Area factor** **AREA** **(1)** Масштабный множитель для мощных транзисторов (на него умножаются BETA, RD, RS CGS, CGD и IS).
- **Instance temperature** **TEMP** **(27)** Текущее значение температуры.

Типичные **SPICE** параметры моделей **JFET** или **MOSFET**:

- **Threshold voltage** **VT0** **(-2)** Пороговое напряжение [В].
- **Transconductance parameter** **BETA** **(0.0001)** **(0.0025)** Коэффициент пропорциональности.
- **Channel length modulation parameter** **LAMBDA** **(0)** Параметр модуляции длины канала [1/В].
- **Gate junction saturation current** **IS** **(1e-014)** Ток насыщения перехода затвор-канал [А].
- **Drain ohmic resistance** **RD** **(0)** Объемное сопротивление области стока [Ом].
- **Source ohmic resistance** **RS** **(0)** Объемное сопротивление области истока [Ом].
- **Zero bias G-S junction capacitance** **CGS** **(0)** Емкость перехода затвор-исток при нулевом смещении [Ф].
- **Zero bias G-D junction capacitance** **CGD** **(0)** Емкость перехода затвор-сток при нулевом смещении [Ф].
- **Gate junction potential** **PB** **(1)** Контактная разность потенциалов p-n-перехода затвора [В].

- **Forward bias junction fit parameter** **FC** **(0.5)** Коэффициент нелинейности емкостей переходов при прямом смещении.
- **Doping tail parameter** **B** **(1)** **(0.3)** Параметр легирования.
- **Flicker Noise Coefficient** **KF** **(0)** Коэффициент, определяющий спектральную плотность фликер-шума.
- **Flicker Noise Exponent** **AF** **(1)** Показатель степени, определяющий зависимость спектральной плотности фликер-шума от тока.
- **Parameter measurement temperature** **TNOM** **(27)** Температура измерений.

Хочу обратить ваше внимание на то, что в стандартном **HELP** Протеуса по модели **JFET** допущены ошибки, в частности ошибочно указаны **KF** и **AF** равными 27.

Одним из важных параметров полевого транзистора являются выходные характеристики в зависимости от напряжения на затворе. Давайте в качестве примера построим такую зависимость для транзистора **2N4416** – наш аналог **КП303**. Сама тестовая схема особенностей не имеет и аналогична снятию выходных характеристик биполярного транзистора, только там, в цепи базы применялся источник тока, а здесь мы оставляем источник напряжения (*Рис. 108*).

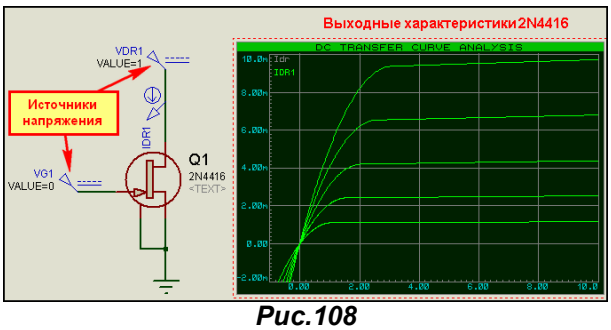


Рис.108

Настройки графика **TRANSFER**, который я применил для исследования, приведены на Рис. 109.

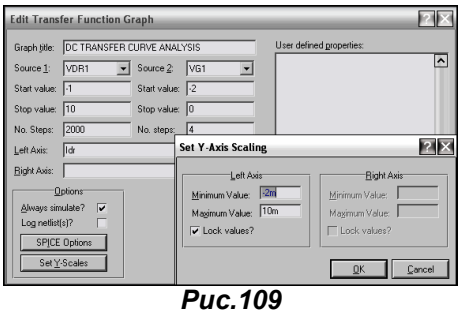


Рис.109

В прилагаемом архиве этот пример расположен в **FET/JFET_char.DSN**. Там же приведен пример с моделью **КП303Д** от О. Петракова. Поскольку модель использовалась тем же методом, что и для диодов и биполярных транзисторов я подробно на этом не останавливаюсь.

А нас ждет обширный список параметров для **MOSFET** моделей транзисторов. Но прежде небольшая преамбула. **SPICE3F5** поддерживает до семи уровней различных **MOSFET** моделей:

1. **MOS1** – модель Шихмана-Ходжеса.
2. **MOS2** – модель Владимиреску-Лиу (Беркли MOS2).
3. **MOS3** – модель Владимиреску-Лиу (Беркли MOS3).
4. **BSIM1** – оригинальная модель BSIM.
5. **BSIM2** – новая модель BSIM.
6. **MOS6** – модель Сакураи и Ньютона.
7. **BSIM3** – последняя модель BSIM версии 3.3.

Нужный тип модели может быть задан явно в свойствах, например:

PRIMITIVE=ANALOG, NMOSFET
LEVEL=5

или:

PRIMITIVE=ANALOG, NBSIM2

Обе записи равнозначны и описывают модель как **BSIM2** с каналом N-типа. Для P-типа соответственно надо использовать запись **PMOSFET** или **PBSIM2**. Два варианта записи применены для сохранения совместимости с предыдущими версиями **SPICE**. Уровни с 1 по 3 относятся к **SPICE2**, а уровни с 4 по 6 являются стандартными для **SPICE3F5**. В Протеусе добавлен уровень 7, для совместимости с последними версиями моделей. Будьте внимательны при использовании моделей взятых из **PSPICE**, поскольку этот пакет поддерживает модели выше уровня 4, и они могут

оказаться несовместимыми с **ProSPICE** Протеуса. Лабцентр рекомендует предварительно визуально (в скрипте модели) проверить – какой уровень она использует.

Протеус всегда моделирует четырехвыводной **MOSFET**: **Drain** (**D** – сток), **Gate** (**G** – затвор), **Source** (**S** – исток), **Bulk Substrate** (**B** – подложка). Если вы моделируете трехвыводной МОП транзистор, то **ProSpice** автоматически соединит подложку с истоком.

SPICE-модели **MOSFET** транзисторов сориентированы на то, чтобы в симуляторах использовать их при моделировании интегральных микросхем (ИС). При этом, поскольку в составе ИС они формируются на одном кристалле, часть их свойств, например **L**, **W**, **AD**, и **AS** будут присвоены по умолчанию в соответствии с параметрами симуляции **DEFL**, **DEFW**, **DEFAD** и **DEFAS** из вкладки **MOSFET** меню **SYSTEM=>Set Simulator Option**, если не описаны отдельно для конкретного компонента в его свойствах *(Рис. 110)*.

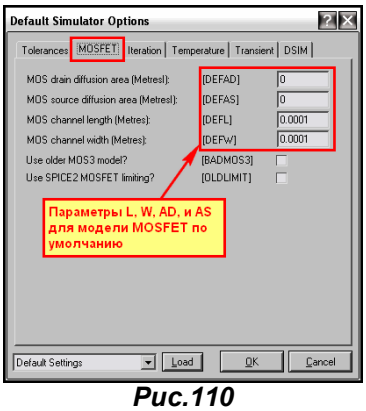


Рис.110

Итак, на *Рис. 111* приведена нелинейная схема замещения МОП транзистора, а ниже приведены свойства SPICE-моделей для типов **MOS1-MOS3** и **MOS6**.

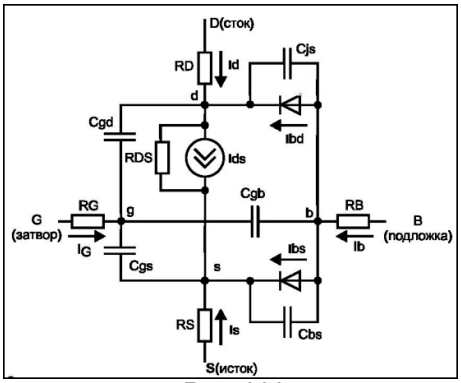


Рис.111

Параметры, выделенные в отдельные строки:

- **Length** **L** **(DEFL)** Длина канала (м);
- **Width** **W** **(DEFW)** Ширина канала (м);
- **Initially OFF** **(-)** Начальное состояние при первой итерации;

Параметры из раскрывающегося списка в свойствах примитива **MOSFET**:

- **Drain area** **AD** **(DEFAD)** Площадь диффузионной области стока ($м^2$);
- **Source area** **AS** **(DEFAS)** Площадь диффузионной области истока ($м^2$);
- **Drain perimeter** **PD** **(0)** Периметр диффузионной области стока (м);
- **Source perimeter** **PS** **(0)** Периметр диффузионной области истока (м);
- **Threshold voltage** **VTO (VT) (0)** Пороговое напряжение при нулевом смещении подложки (В);
- **Transconductance parameter** **KP (2e-5)** Параметр удельной крутизны ($А/В^2$);
- **Bulk threshold parameter** **GAMMA (0)** Коэффициент влияния потенциала подложки на пороговое напряжение ($В^{1/2}$);
- **Surface potential** **PHI (0.6)** Поверхностный потенциал сильной инверсии (В);
- **Channel length modulation** **LAMBDA (0)** Параметр модуляции длины канала (1/В только MOS1 и MOS2)
- **Bulk junction saturation current** **IS (1e-014)** Ток насыщения p-n-перехода сток-подложка (исток-подложка) (А);
- **Drain ohmic resistance** **RD (0)** Активное сопротивление стока (Ом);
- **Source ohmic resistance** **RS (0)** Активное сопротивление истока (Ом);
- **B-D junction capacitance** **CBD (0)** Емкость донной части p-n-перехода сток-подложка при нулевом смещении (Ф);
- **B-S junction capacitance** **CBS (0)** Емкость донной части p-n-перехода исток-подложка при нулевом смещении (Ф);
- **Bulk junction potential** **PB 0.8** Контактная разность потенциалов донных p-n- переходов подложки (В);
- **Gate-source overlap capacitance** **CGSO (0)** Удельная емкость перекрытия затвор-исток (Ф/м);

- **Gate-drain overlap capacitance** **CGDO (0)** Удельная емкость перекрытия затвор-сток на длину канала (Ф/м);
- **Gate-bulk overlap capacitance** **CGBO (0)** Удельная емкость перекрытия затвор-подложка (Ф/м);
- **Flicker noise coefficient** **KF (0)** Коэффициент, определяющий спектральную плотность фликер-шума;
- **Flicker noise exponent** **AF (1)** Показатель степени, определяющий зависимость спектральной плотности фликер-шума от тока через переход;
- **Sheet resistance** **RSH (0)** Удельное сопротивление диффузионных областей истока и стока (Ом/кв);
- **Bottom junction cap per area** **CJ (0)** Удельная емкость (на площадь перехода) донной части p-n-перехода сток(исток)-подложка при нулевом смещении (Ф/м^2);
- **Bottom grading coefficient** **MJ 0.5** Коэффициент, учитывающий плавность донной части перехода подложка-сток (исток);
- **Side junction cap per area** **CJSW (0)** Удельная емкость боковой поверхности перехода сток (исток)-подложка при нулевом смещении (на длину периметра) (Ф/м);
- **Side grading coefficient** **MJSW 0.33** Коэффициент, учитывающий плавность бокового перехода подложка-сток (исток) (Ф/м);
- **Bulk junction saturation current density** **JS (0)** Плотность тока насыщения переходов сток(исток)-подложка (А/м^2);
- **Oxide thickness** **TOX (0.1um)** Толщина оксида (м);
- **Lateral diffusion** **LD (0)** Глубина области боковой диффузии (м);
- **Surface mobility** **UO (600cm^2/Vs)** Поверхностная подвижность носителей (см2/В/с);
- **Substrate doping** **NSUB (0)** Уровень легирования подложки (1/см^2);
- **Surface state density** **NSS (0)** Плотность медленных поверхностных состояний на границе кремний-подзатворный оксид (1/см^2);

Параметры, задаваемые вручную в окне **Other Properties**:

- **Drain squares** **NRD** **(1)** Относительное удельное сопротивление стока;
- **Source squares** **NRS** **(1)** Относительное удельное сопротивление истока;
- **Initial D-S voltage** **ICVDS (-)** Начальное напряжение сток-исток (В);
- **Initial G-S voltage** **ICVGS (-)** Начальное напряжение затвор-исток (В);
- **Initial B-S voltage** **ICVBS (-)** Начальное напряжение подложка-исток (В);
- **Parameter measurement temperature** **TEMP (27)** Рабочая температура измерения;
- **Model Index** **LEVEL (1)** Уровень модели;
- **Critical field for mobility degradation** (MOS2 only) **UCRIT** **(10000V/cm)** Критическая напряженность поля, при которой подвижность носителей уменьшается в 2 раза (только MOS2) (В/см);
- **Critical field exponent in mobility degradation** (MOS2 only) **UEXP** **(0)** Экспоненциальный коэффициент снижения подвижности носителей (только MOS2)
- **Maximum carrier drift velocity** **VMAX (0)** Максимальная скорость дрейфа носителей (м/с);
- **Total channel charge coefficient** **NEFF (1.0)** Эмпирический коэффициент коррекции концентрации примесей в канале;
- **Coefficient for forward bias depletion capacitance formula** **FC (0.5)** Коэффициент нелинейности барьерной емкости прямосмещенного перехода подложки.

Параметры **BSIM** моделей в **HELP** Протеуса отсутствуют, так как применяются эти модели значительно реже. Их параметры создаются автоматически с помощью прибора для тестирования экспериментальных образцов. Однако, если кого-то интересуют данные модели, то параметры **BSIM1** приведены в статье О. Петракова ж. «Схемотехника» №7 2001 и его же книге, упомянутой раньше.

Наиболее «шустрыми» для симуляции считаются модели уровня 1 и 3, при этом модель уровня 1 дает более грубые вычисления, модель уровня 4 применяется для мощных МОП транзисторов.

Ну и в заключении хотелось бы чуть-чуть остановиться на **IGBT** моделях транзисторов. Как такового примитива **IGBT** в ProSPICE нет, но SPICE модели в библиотеках присутствуют. Ну и раз есть, то мы и давай пихать их куда попало, тем более что в свойствах ничего такого не указано. Но вот беда – кнопочка там справа есть и обозначена как **Device Notes** *(замечания по компоненту)*. Картинку не привожу, поскольку у меня на экране текст в замечаниях заползает за окно. Я его полностью «выковырял» и привожу ниже

If the simulation aborts with "timestep too small" then set:
RELTOL=0.005 (up to 0.01)
ITL4=300 (up to 500)
ITL1=300
and in extreme cases (in order of importance):
GMIN=1e-09
ABSTOL=1e-08
VNTOL=1e-05 (up to 1e-03) only if required
TMAX=10 to 100ns

Для непосвященных в существование других языков кроме русского поясню значения фраз выделенных красным:

Если симуляция прерывается с сообщением «timestep too small» установите: (… *далее перечисляются параметры симуляции которые необходимо изменить)*

и в крайних случаях (в порядке важности): *(опять перечисляются параметры симуляции, а для VNTOL поднятие до 1e-03)* только если потребуется.

Никаких ассоциаций в связи с этим замечанием не возникает? Тем более, если учесть, что **IGBT** это и полевой и биполярный транзисторы, а в ряде случаев и защитный диод и «все в одном флаконе». Так что перефразируем Ильфа и Петрова – «грузить IGBT бочками» в своих проектах нам не удастся и «братья Карамазовы» тут не помогут. Но это не значит, что **IGBT** уж совсем не симулируются. В доказательство приведу пример **IGBT_MOSFET.DSN**. В нем я выложил в графиках выходные характеристики для парочки тех и других транзисторов, ну а про реалтайм для **IGBT**, да еще нескольких в проекте видимо придется пока забыть до появления каких-нибудь супер-пупер процессоров или многоядерной версии Протеуса с распределенными вычислениями.

Справедливости ради надо отметить, что первую же модель в библиотеке **IGBT** - **IRG4BC10KD** мне не удалось просимулировать даже с помощью графиков, видно что-то «в консерватории не так», ну а остальные вроде ничего – живые.

На этом закончим громоздкий материал по полевым транзисторам и наконец, вернемся к заброшенной нами еще в начале этого раздела модели ОУ.

4.17. Типы моделей сложных компонентов – взгляд изнутри. Схематичное и поведенческое моделирование. SPICE модели ОУ и компараторов в Протеусе. График частотного анализа.

Мы завершили рассмотрение аналоговых примитивов. Я умышленно упустил рассмотрение некоторых аналоговых моделей – это в основном различные длинные линии: **URCLINE, LOSSYLINE, DELAY**. Они достаточно хорошо рассмотрены у В. Гололобова в упоминавшейся ранее его книге, и желающие всегда могут ознакомиться с ней самостоятельно, поскольку это редкое, но с моей точки зрения весьма похвальное решение – автор свободно выложил ее на своем сайте. Но работа с этими моделями весьма специфична, а требуются они не так уж и часто. Мы же возвращаемся к заброшенной нами еще в **п.п.4.1–4.2** FAQ графической модели ОУ и приступаем к ее «оживлению».

Но прежде я хотел бы остановиться еще на одном характерном аспекте, присущем моделированию сложных составных радиокомпонентов, каковыми являются ОУ и компараторы. До сих пор мы рассматривали модели элементарные: резисторы, диоды, конденсаторы, транзисторы. Но тот же ОУ может содержать их до нескольких десятков. Вернемся опять к упоминавшемуся ранее примеру **741.DSN** из папки **\SAMPES\Graph Based Simulation**. Откроем его, кликнем правой по графическому изображению ОУ **U1** и выберем **Goto Child Sheet** – переход на дочерний лист этой модели (Рис. 112).

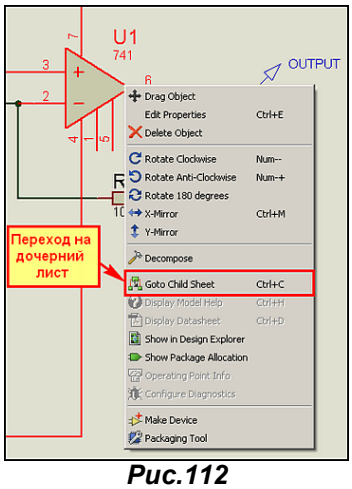


Рис.112

Вот здесь то и размещается полная внутрисхемная модель – воспроизведена структура ОУ (Рис.113). Как видим, в ней 21 транзистор и 12 резисторов.

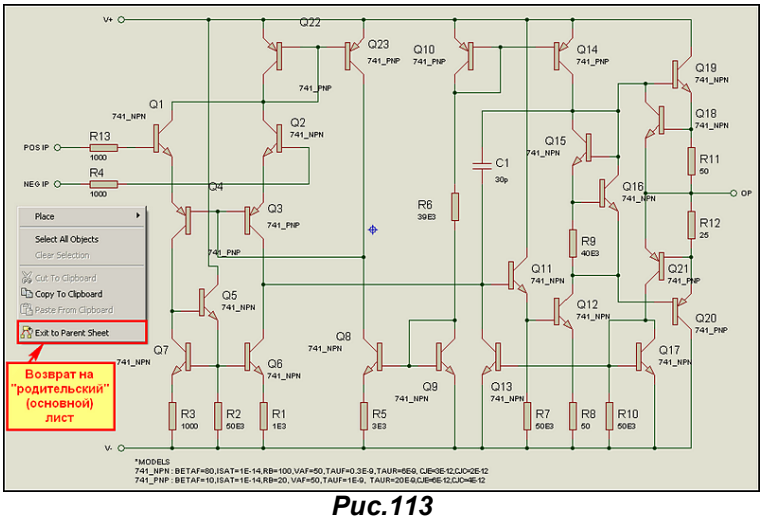


Рис.113

Из этого листа в принципе уже можно скомпилировать **MDF** файл, в котором вся эта структура и сохраниться, а впоследствии прикрепить его к графической модели ОУ. Но представьте, что при каждом использовании данной модели операционные точки данного MDF будут просчитываться программой, а их не так уж и мало. Поместив пару-тройку таких ОУ в проект, да еще подав на входы аналоговые сигналы с приличной частотой, мы рискуем полностью затормозить симуляцию – Протеус благополучно зависнет на расчетах внутренних точек моделей. Именно поэтому при создании моделей сложных компонентов разработчики в большинстве случаев отказываются от такого моделирования. Какова же альтернатива? Давайте вспомним те примитивы, которые я так настойчиво навязывал Вам в течение долгого времени. Среди них есть различные управляемые источники тока и напряжения – ну чем не усилители. Задали коэффициент передачи равным усилению ОУ, навесили несколько дополнительных элементов, чтоб имитировать входные и выходные импедансы и частотную характеристику и готово. И самое важное – все это элементарные SPICE-модели, быстрые и не требующие больших ресурсов на математику от компьютера. Такие модели получили название «поведенческие» и широко используются разработчиками.

В отношении ОУ и компараторов данный подход обычно выглядит следующим образом: моделируется входной дифференциальный каскад на транзисторах (биполярных или полевых), а все остальное заменяется управляемыми источниками и минимальным количеством пассивных компонентов: резисторов, конденсаторов и т.п. Таким образом, мы получаем макромодель сложного компонента.

В качестве примера рассмотрим модель **140УД7** О. Петракова (ж. «Схемотехника» №2, 2002 и его упоминавшаяся ранее книга). На *Рис. 114* приведена структура макромодели. Обращаю ваше внимание на то, что в схеме и в журнале и в книге (завидное постоянство!) допущена очепятка. Диод, который стоит справа от **dlp** между узлами **90-92** конечно же **dln**, а не **Vin** как на рисунке. В тексте программы он обозначен правильно.

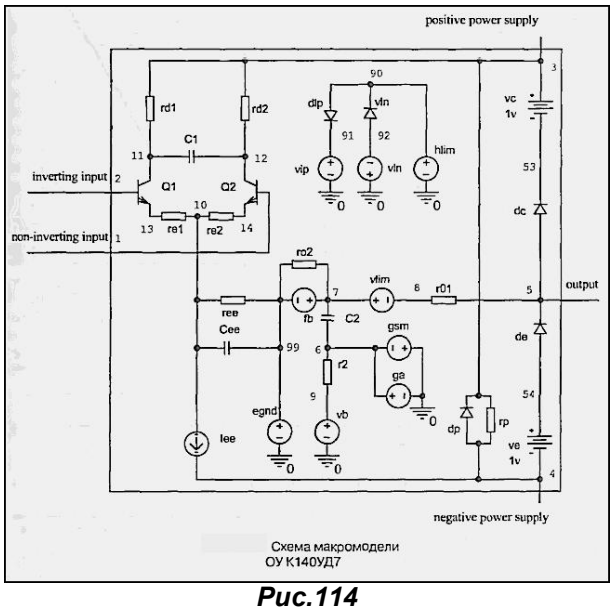


Рис.114

Ниже приведен полный текстовый скрипт PSPICE для модели ОУ. Как видим, из всей внутренней структуры сохранен только входной диф. каскад, о чем я и упоминал выше.

```
* connections:      non-inverting input
*                   | inverting input
*                   | | positive power supply
*                   | | | negative power supply
*                   | | | | output
*                   | | | | |
.subckt K140UD7   1 2 3 4 5
*
 c1   11 12 8.661E-12
 c2    6  7 30.00E-12
 dc    5 53 dy
 de   54  5 dy
 dlp  90 91 dx
 dln  92 90 dx
 dp    4  3 dx
 egnd 99  0 poly(2),(3,0),(4,0) 0 .5 .5
 fb    7 99 poly(5) vb vc ve vlp vln 0 10.61E6 -1E3 1E3 10E6 -
10E6
 ga    6  0 11 12 188.5E-6
 gcm   0  6 10 99 5.961E-9
 iee  10  4 dc 15.16E-6
 hlim 90  0 vlim 1K
 q1   11  2 13 qx
 q2   12  1 14 qx
 r2    6  9 100.0E3
 rc1   3 11 5.305E3
 rc2   3 12 5.305E3
 re1  13 10 1.836E3
 re2  14 10 1.836E3
 ree  10 99 13.19E6
 ro1   8  5 50
 ro2   7 99 100
 rp    3  4 18.16E3
 vb    9  0 dc 0
 vc    3 53 dc 1
 ve   54  4 dc 1
 vlim  7  8 dc 0
 vlp  91  0 dc 40
 vln   0 92 dc 40
.model dx D(Is=800.0E-18 Rs=1)
.model dy D(Is=800.00E-18 Rs=1m Cjo=10p)
.model qx NPN(Is=800.0E-18 Bf=93.75)
.ends
*$
```

Немного остановимся на данном скрипте, поскольку тут появилось много новых, значимых для нас элементов. Само описание модели начинается с точки и аббревиатуры **.subckt** *(subcircuit – подсхема)*, далее ее название **K140UD7** и перечень всех точек с внешними связями (иными словами выводов) **1 2 3 4 5**. Как всегда, все строки, стартующие со звездочки ***,** являются комментариями и приведены для облегчения восприятия программы. Так, например, в начале помещены описания назначения выводов, которые направлены к соответствующим номерам узлов вертикальными, расположенными один над другим разделителями. Ниже строки **.subckt** расположено описание непосредственно субсхемы в каждой строке компонент, номера узлов к которым он подключен и его номинал или ссылка на модель.

Например:

c1 11 12 8.661E-12 - конденсатор, включенный между узлами 11 и 12 емкостью 8.661 пФ

или

q1 11 2 13 qx – транзистор, подключенный к узлам 11, 2, 13 (соответственно КБЭ) с моделью **qx**

Здесь надо остановиться подробнее. Обратите внимание, что в конце скрипта расположено описание самих моделей, примененных в подсхеме так же начинающихся с точки. Это два варианта диодов – **dx** и **dy** и модель NPN транзистора – **qx**. Параметры примитивов указаны в скобках – это мы уже встречали. Почему я здесь заострил наше внимание? В данном случае все нормально, поскольку применены типовые примитивы, и модель будет работать в Протеусе. Но иногда, разработчики для экономии времени указывают здесь ссылки на конкретные компоненты, расположенные в других библиотеках – я приведу чуть ниже пример того же О. Петракова с компаратором. Вот в таком случае, если модель отсутствует в библиотеках вашей программы, вы получите ошибку симуляции со ссылкой на несуществующую модель. Об этом необходимо помнить, перетаскивая в Протеус SPICE модели из других программ: OrCAD, MicroCAP, Multisim и т.п. или используя PSPICE модели фирм-производителей компонентов, извлеченные из готовых библиотек. Ну и в конце текстового описания модели стоит завершающий оператор **ends**.

И еще один для кого-то может быть не очень приятный сюрприз. Посмотрите на структуру макромодели и сравните с графической моделью, к которой мы это будем прилеплять – использованы только входы, выход и питание, а два вывода реальной микросхемы остаются незадействованными. И это не прихоть О. Петракова, создавшего модель **K140UD7**. У большинства SPICE-моделей дополнительные выводы балансировки и коррекции как в Протеусе, так и в других симуляторах благополучно «висят в воздухе». Так что все попытки навешивать на них

дополнительные элементы и добиться при этом изменения результатов симуляции будут абсолютно бесполезной тратой времени. И встретив при запуске симуляции некоторых ОУ желтое предупреждение **Pin** такой-то **no simulated**, не пугайтесь – просто в модели нет описания для этого вывода, и ISIS его игнорировал. Но не всегда, когда пин отсутствует, вылезает такое предупреждение. Возьмем, например, столь всеми любимый **LM318** от **National Semiconductor** со SPICE-моделью из библиотеки ISIS. Включаем флажок и видим (Рис. 115):

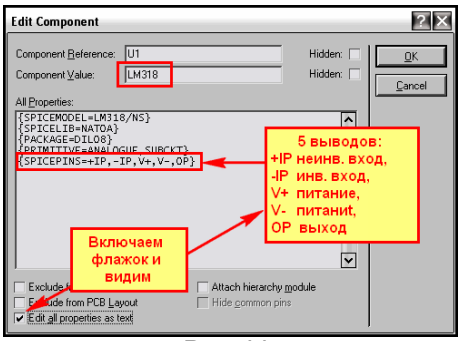


Рис.115

Какие тут могут быть цепи баланса и т.п., они даже и не упоминаются. Только входы, выход и питание.

Ну вот, теперь настала пора вытащить из забвения ту графическую модель ОУ, которую мы создали в **п.п.4.1- 4.2** этого FAQ. Мы ее тогда сохранили как **741R**. Ну а теперь будем делать из нее **K140UD7**. Процедура аналогична присоединению SPICE модели к примитивам, но есть и своя специфика. Итак, втаскиваем нашу графическую модель в проект, который сохраняем в некоторой папке у меня в примере это **140UD7**. В этой же папке должен лежать и библиотечный SPICE файл с расширением **.LIB**, в котором должна присутствовать наша модель **K140UD7**. У меня в примере это **OU_RU.LIB**. Кроме 140УД7 там еще парочка моделей. Проверяем цоколевку по любой документации. Опля! Не хватает одного пина – **СК**. Это не проблема. «Декомпозим» нашу графическую модель молотком и добавляем недостающее. Затем выделяем всю нашу графику (скрипт я удалил, но можно было и не удалять) и опять давим **Make Device**, ну, т.е. повторяем процедуру из **п.п.4.1- 4.2**, но с некоторыми отличиями. На первой вкладке назовем наш девайс **K140UD7**, префикс оставим прежним **DA** (ну или поставьте **U**, как стандартно для м/сх в ISIS). Вторую, где назначается **Package,** пока пропустим – его можно назначить и позже. Все отличие начинается на третьей вкладке, где и остановимся подробнее. Здесь необходимо проделать следующие манипуляции:

- Через кнопку **New** из всплывающего списка назначаем **SPICEMODEL** как показано на *Рис.116*. В **Property Default** в окне **Default Value** прописываем нашу модель и библиотеку, где она расположена. **K140UD7,RU_LIB** (Внимание! Важно! Здесь и далее после запятых пробел отсутствует!).
- Аналогичным образом через кнопку **New** добавляем свойство **PRIMITIVE** как показано на Рис. 117. В **Default Value** вводим **ANALOGUE,SUBCKT**.
- Ну и теперь аналогично добавляем последнее свойство **SPICEPINS**. В **Default Value** через запятую без пробелов перечисляем все активные (т.е. те которые в нашей SPICE-модели в строке **.subckt**) в соответствии с их именами (а не номерами – не путайте) в графической модели. У нас это будут – **POS IP,NEG IP,V+,V-,OP** – *Рис. 118*. (Важно! Вот здесь для входов пробел в **POS IP** и **NEG IP** присутствует, потому что он есть в именах выводов графической модели).

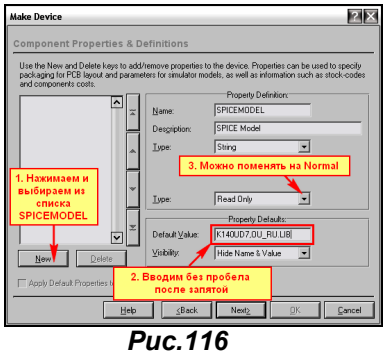


Рис.116

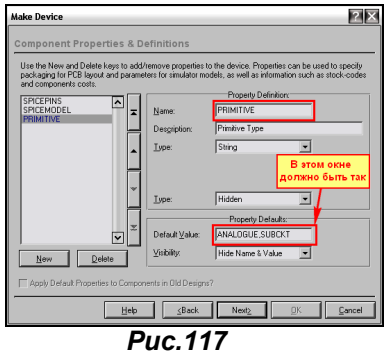


Рис.117

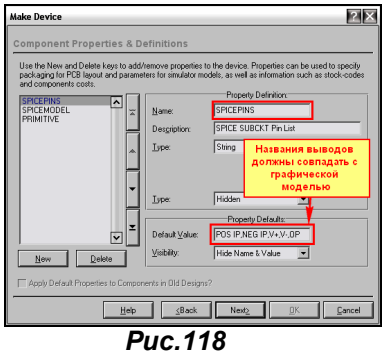


Рис.118

Далее проходим по вкладкам до конца и завершаем создание модели, как и ранее, выбрав для нее на последней вкладке из раскрывающегося списка **Device Category** – **Operational Amplifiers** (категория – операционные усилители), **Device Sub-category** – **Single** (подкатегория – одиночные), ну а в **Device Manufacturer** (производитель) – можно вбить, кому что в голову взбредет, я лично для

наших компонентов пишу **ExUSSR**. Сохраняется все это по нажатию **OK** в библиотеке **USRDVC**. Потом можно будет перенести эту модель в другую библиотеку через менеджер библиотек.

Теперь проверяем работоспособность нашей созданной модели – она должна появиться в левом окне селектора компонентов. Добавляем ее в проект, обвешиваем необходимой периферией (резисторами и генератором на входе) – я выбрал вариант неинвертирующего усилителя с коэффициентом усиления 2 (равные резисторы). Напомню, что для такого варианта **Uвых=Uвх*(1+R1/R2)**. Результат графической симуляции представлен на *Рис. 119.*

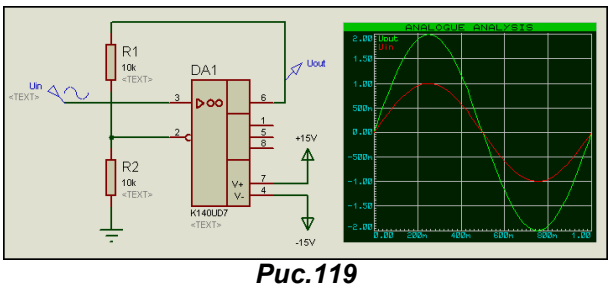


Рис.119

Ну вот, убедились, что модель работоспособна – пора загнать ее в корпус. Кликаем правой кнопкой по модели в проекте и выбираем из всплывающего меню самый нижний пункт **Packaging Tools** (То же самое можно проделать, выделив компонент и выбрав значок синяя микросхема с гаечным ключом в верхнем меню ISIS). Нам будет представлено окно **Package Device** (Рис. 120). Если вы идете туда первый раз, то предварительно всплывет сообщение **ISIS Information** с рекомендациями по изменению размеров окна **Package Device** – его можно сразу закрыть, предварительно поставив флажок – больше не напоминать.

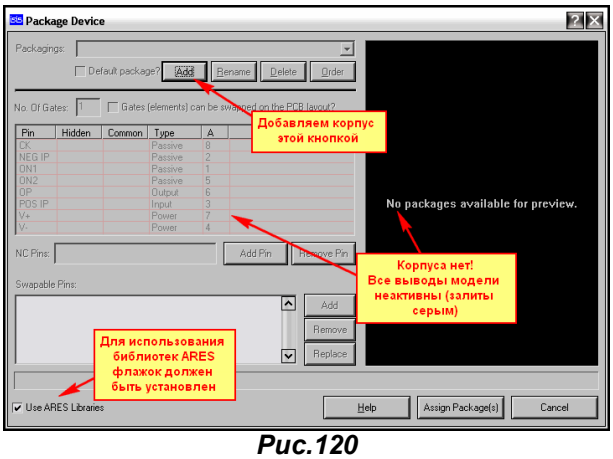


Рис.120

В **Package Device** пока корпуса нет. Нажимаем кнопку Add и выбираем из библиотек ARES нужный. Я для начала назначу корпус DIP с восьмью ногами (Рис. 121) – обратите внимание, что в ARES он называется **DIL** (Dual-In-Line), а **DIP** (Dual-In-Plane) там называются корпуса такого же типоразмера, но монтируемые не в отверстия (**Hole**), а на поверхность, т.е. планарно.

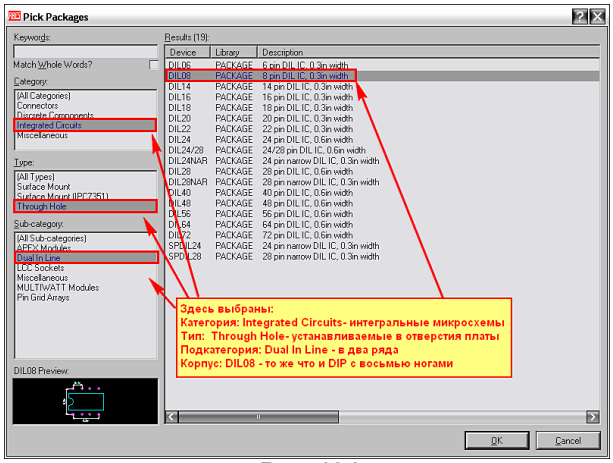


Рис.121

Конечно же, зная названия корпуса, можно было его быстро найти через строку **Keywords** как и в ISIS, введя часть названия или целиком. Когда привыкните – рекомендую пользоваться этим способом. Но, тем не менее, корпус мы нашли и нажмем **OK**. Наш корпус появился в черном окне справа, выводы стали доступными – заканчиваем назначение, нажав кнопку **Assign Package(s)**. Затем во всплывшем окне **Selext Library For Packaged Device** убеждаемся, что корпус будет сохранен в той же библиотеке, что и модель (**USRDVC**) и сохраняем все кнопкой **Save Package(s).**

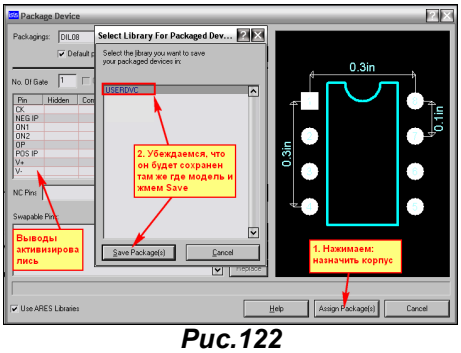


Рис.122

В принципе, мы могли бы, и сразу назначить еще один корпус для круглого варианта микросхемы, но это можно сделать и позже. Тем более, что в ARES бесполезно искать русский круглый корпус **301.8-1**, но есть похожие, – например: **TO77**, правда ключик у него находится напротив восьмой ноги.

На этом процедура создания работоспособной модели ОУ К140УД7 практически закончена. Давайте в качестве теста проведем частотный анализ нашей модели. Добавим в проект график **FREQUENCY** из левого меню **Graph Mode**. Зайдем в его свойства и установим (применительно к схеме из *Рис. 119*) параметры как на *Рис. 123*.

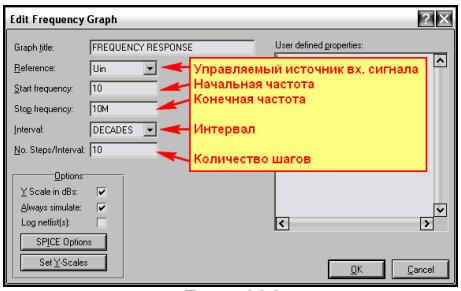


Рис.123

Здесь в качестве источника сигнала мы выбрали генератор **Uin**, задали ему начальную частоту **10 Гц**, конечную частоту **10 МГц**, интервал по оси **X** декадный (в 10 раз), ну и количество шагов **10** (можно и больше). Затем втащим на поле графика наш выходной зонд **Uout**, причем и к левой оси (в левый верхний угол – это частотная характеристика – будет зеленой) и к правой оси (в правый нижний угол – это фазовая характеристика – будет красной) и запустим график на исполнение. Результат приведен на *Рис. 124*. По частотной зеленой трассе мы видим, что наш неинвертирующий усилитель на **К140УД1** имеет спад усиления, начиная со **100 кГц**, который в районе **10 МГц** пересекает ось **0 дБ** и продолжается вплоть до **10 МГц**, где достигает **-35 дБ**. По правой шкале **Y** и красной трассе можно отследить изменение фазы сигнала в градусах.

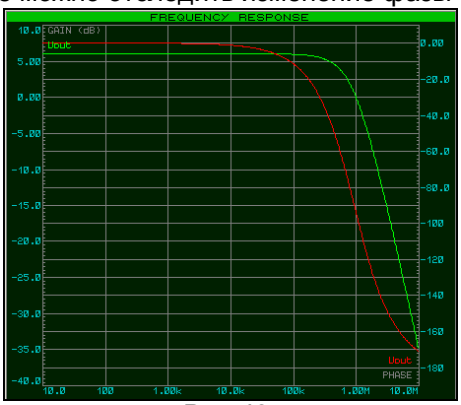


Рис.124

Весь процесс создания модели ОУ во вложении **OU_COMP1** папка **140UD7**.

Ну а теперь немного остановимся на упрощенной модели компаратора от О. Петракова, как я и обещал выше. Сама макромодель 521СА3 из литературы, упоминаемой выше, приведена на *Рис. 125*. Как видим, в качестве транзисторов автор применил свои модели КТ315А. Конечно, если выполнить модель и не вложить в нее модели этих транзисторов, то работать она не будет. Но в данном случае они полностью присутствуют, поэтому можно промоделировать компаратор в ISIS аналогично тому, как мы моделировали ОУ.

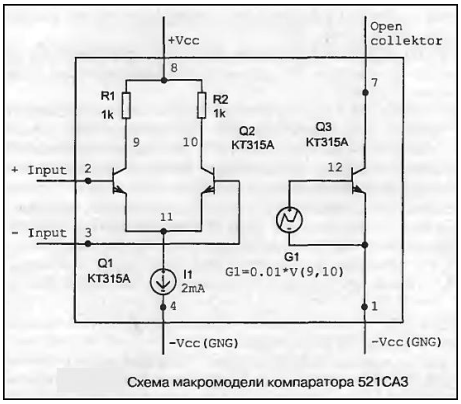


Рис.125

Ниже приведен оригинальный текст модели 521СА3 из вышеупомянутых источников. К сожалению, в таком виде модель в Протеусе полностью неработоспособна.

```
* ----- Упрощённая макромодель компаратора  521СА3 ----------
*
*                    Открытый эмиттер
*                     |  Не инвертирующий вход
*                     |  |  Инвертирующий вход
*                     |  |  |  Минус источника питания
*                     |  |  |  |  Открытый коллектор
*                     |  |  |  |  |  Плюс источника питания
*                     |  |  |  |  |  |
.SUBCKT 521CA3     1   2   3   4   7   8
R1 8  9 1K
R2 8 10 1K
*   C  B  E  - порядок перечисления выводов транзисторов.
Q1  9  2  11  KT315A
Q2  10  3  11  KT315A
Q3  7  12  1  KT315A
I1 (4 11) 2mA ; источник тока 2мА.
*
G1 (12 1) (9 10) 0.1
.model KT315A NPN (Is=23.68f Xti=3 Eg=1.11 Vaf=60 Bf=108
Ne=1.206
+      Ise=23.68f Ikf=.1224 Xtb=1.5 Br=4.387G Nc=1.8 Isc=900p
Ikr=20m Rc=5
+      Cjc=7p Mjc=.333 Vjc=.7 Fc=.5 Cje=10p Mje=.333 Vje=.7
+      Tr=130.5n Tf=1n Itf=40m Vtf=80 Xtf=1.1 Rb=10)
.ENDS
*$
```

Во-первых, автором допущена ошибка в описании источника тока **I1**. Обратите внимание – на схеме источник стоит правильно, а в тексте модели принято первой точкой указывать положительный полюс источника, т.е. узел **11**, а запись следующая: **I1 (4 11) 2mA**. Получается, что источник включен с точностью до наоборот.Но не это главное. Автор произвольно расположил узлы **SUBCKT**, и первым у него идет открытый эмиттер выходного транзистора. Не знаю, может **PSPICE** и допускает такие вольности, но **ProSPICE** Протеуса упорно не хотел корректно работать, пока я не расположил узлы с общепринятой последовательностью – входы, питание, выходы. В результате скорректированная строка получилась следующего вида применительно к схеме *Рис. 125*:

```
*                    Неинвертирующий вход
*                    | Инвертирующий вход
*                    | | Плюс источника питания
*                    | | | Минус источника питания
*                    | | | | Открытый эмиттер
*                    | | | | | Открытый коллектор
*                    | | | | | |
.SUBCKT 521CA3 2 3 8 4 1 7
```

После таких изменений модель заработала как надо. Во вложении в папке **521SA_Petrakova** присутствуют два проекта и два файла **.LIB**. В файле **521CA3.DSN** (к нему относится **521CA.LIB**) рассмотрен сам процесс создания модели и там источник тока оставлен в оригинальном включении (неправильном). Изменен только порядок перечисления выводов, чтобы добиться хоть какой-то работоспособности. В файле **521SA3correct.DSN** (к нему относится **521SA.LIB**) скорректировано включение источника тока **I1**, а также модель транзистора заменена на примитив с минимально измененными от принятых по умолчанию параметрами. Эта модель и работает как надо (сравните графики там и там), и ресурсов компа кушает меньше.

На этом я закончу процесс рассмотрения SPICE моделирования. Нам осталось только поучиться упаковывать самостоятельно SPICE библиотеки. Ну а далее на примере все тех же операционных усилителей перейдем к схематичному моделированию в Протеусе.

4.18. Создание собственных Spice-библиотек (.SML) с помощью утилиты PUTSPICE.EXE.

В **п.4.12** мы уже познакомились с утилитой **PUTSPICE.EXE**, но знакомство это носило поверхностный характер. Сейчас, когда мы уже имеем представление об использовании SPICE-моделей сторонних разработчиков в Протеусе, настала пора применить ее в действии. Для начала предлагаю потренироваться на уже существующих в ISIS моделях, для которых отсутствует симуляция (**No simulation model**). В качестве подопытных кроликов в нашем случае будем использовать модели ОУ и компараторов **Texas Instruments**. Для начала прогуляемся на сайт фирмы (а конкретно на их страничку Центра проектирования E-lab) по следующему адресу:
http://focus.ti.com/adc/docs/midlevel.tsp?contentId=31690
Здесь находятся модели для различных симуляторов (Рис. 126), в том числе независимые: **General** (их совсем немного) и множество моделей **PSPICE**. Вот ZIP файл (архив 26МБайт) с последними нас и интересует, поскольку мы уже убедились, что после их проверки на работоспособность, они вполне применимы для моделирования в ISIS.

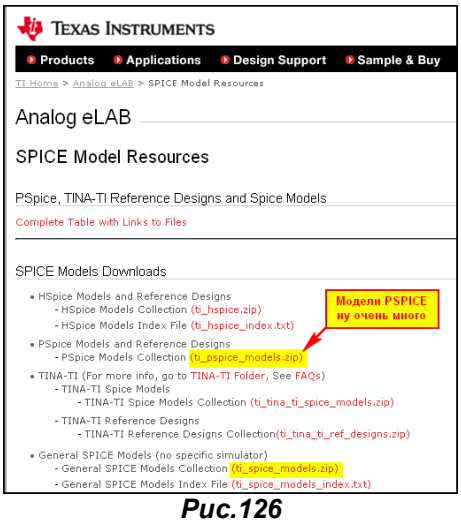


Рис.126

Распакуем архив **ti_pspice_models.zip** в какую-нибудь папку на своем компьютере. В полученном подкаталоге будет множество папок с названиями компонентов, причем часть из них будут пустыми, а часть с ZIP-файлами соответствующих моделей. Начнем с компараторов и в первую очередь с наиболее часто встречающихся – **LM111…311**. Откроем архив **LM111_PSpice_Model.zip** из папки **LM111**. Там находится одноименный файл с расширением **301**. По сути – это обычный текстовый файл, ну а расширение, судя по всему, относится к версии **PSPICE**. Нас оно интересует в том плане, что в папке **MODELS** Протеуса модели этого типа хранятся в библиотеке **TEX301.SML**, а, например, модели **Texas Instruments**, которые после распаковки будут иметь расширение **5_1** в библиотеке **TEX5_1.SML**. Нам предстоит переименовать его в **LM111.MOD**, поскольку, как мы знаем, ISIS поддерживает для SPICE-моделей расширения **MOD** и **LIB**. Ну и попутно с помощью любого текстового редактора заглянем внутрь самого файла, чтобы убедиться, что в нем нет ничего предосудительного. Ниже приведено его содержимое.

```
* LM111 VOLTAGE COMPARATOR "MACROMODEL" SUBCIRCUIT
* CREATED USING PARTS VERSION 4.03 ON 03/07/90 AT 12:28
* REV (N/A)
* CONNECTIONS:   NON-INVERTING INPUT
*                | INVERTING INPUT
*                | | POSITIVE POWER SUPPLY
*                | | | NEGATIVE POWER SUPPLY
*                | | | | OPEN COLLECTOR OUTPUT
*                | | | | | OUTPUT GROUND
*                | | | | | |
.SUBCKT LM111    1 2 3 4 5 6
*
 F1   9  3 V1 1
 IEE  3  7 DC 100.0E-6
 VI1  21  1 DC .45
 VI2  22  2 DC .45
 Q1   9 21  7 QIN
 Q2   8 22  7 QIN
 Q3   9  8  4 QMO
 Q4   8  8  4 QMI
.MODEL QIN PNP(IS=800.0E-18 BF=666.7)
.MODEL QMI NPN(IS=800.0E-18 BF=1002)
.MODEL QMO NPN(IS=800.0E-18 BF=1000 CJC=1E-15 TR=102.5E-9)
 E1   10  6  9  4  1
 V1   10 11 DC 0
 Q5   5 11  6 QOC
.MODEL QOC NPN(IS=800.0E-18 BF=103.5E3 CJC=1E-15 TF=11.60E-12 TR=48.19E-9)
 DP   4  3 DX
 RP   3  4 6.667E3
.MODEL DX  D(IS=800.0E-18)
*
.ENDS
```

Типичный файл SPICE-модели. Убедившись, что ничего лишнего в модели не присутствует – тестируем модель. Для этого создаем проект, с ее использованием. В проект добавляем **LM111** (именно ту **No simulation** от **Texas Instruments)**, затем, выделив ее, давим **Make Device**. На третьей вкладке добавляем следующие свойства:

SPICEMODEL = LM111
SPICEFILE = LM111.MOD
PACKAGE = DIL08
PRIMITIVE = ANALOGUE,SUBCKT
SPICEPINS = +IP,-IP,VCC+,VCC-,COL OUT,EMIT OUT

Обратите внимание, что я добавил свойство не **SPICELIB**, как мы делали ранее, а **SPICEFILE** и задал ему значение **LM111.MOD** – ведь именно так мы назвали файл модели (Рис. 127).

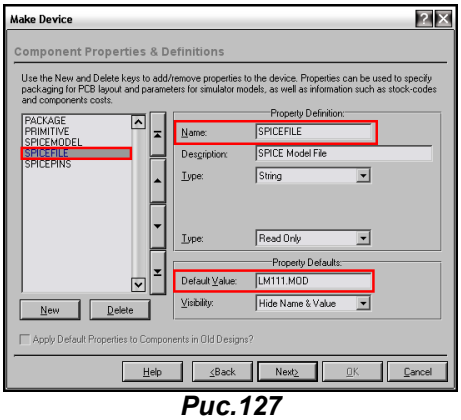


Рис.127

Все остальное оставляем без изменений и проходим весь процесс **Make Device** до конца. Модель с присоединенными параметрами сохраняем пока по умолчанию в библиотеке **USRDVC**. После этого обвешиваем нашу микросхему периферией для теста (я использовал простенький, как и выше) и проверяем, что она адекватно реагирует. Конечно, стоило бы поподробнее протестировать, но для наших целей и этого пока достаточно. Весь процесс во вложении **TEX_INS/COMPAR_TEST/LM111_PSpice_Model/LM111.DSN**.

Аналогично я проверил **LM211** и **LM311**. Все модели работают. Далее есть два пути: создать свою SML-библиотеку, или закинуть их в существующую.

Рассмотрим первый вариант. Складываем наши файлы **.MOD** в одну папку, туда же кладем саму утилиту **PUTSPICE.EXE** из папки **BIN** Протеуса. Лучше, если путь к папке будет как можно более коротким, чтобы не мучить зря клавиатуру и свои мозги. Далее запускаем командную консоль и создаем библиотеку, например **TEX301_1** на 200 элементов следующей командой:

PUTSPICE –L=TEX301_1.LML –C=200

После чего другой командой:

PUTSPICE –L=TEX301_1.LML –D LM111.MOD LM211.MOD LM311.MOD

укладываем туда наши файлы. Обратите внимание, что я использовал ключ **–D**, который сотрет файлы **.MOD** с диска после их упаковки в библиотеку. Сейчас мы туда заложили три файла, а если бы их было много – поди, разберись: что уже положил, а что нет. А так стерлись, значит упаковал. Весь процесс на Рис. 128 (там я сначала запустил **PUTSPICE** без параметров, чтоб вывести подсказку).

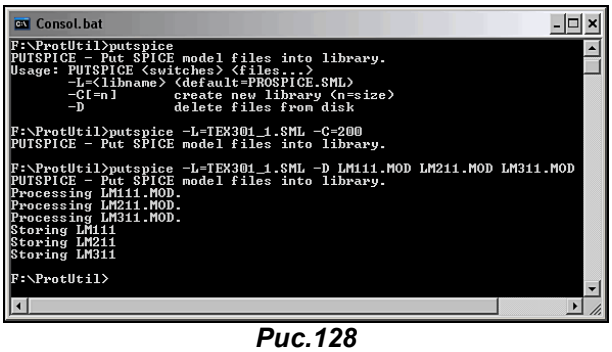


Рис.128

На законный вопрос любознательных: а зачем вообще упаковывать **.MOD** в **.SML**? – ведь можно было просто побросать их в папку **MODELS** Протеуса, отвечу – так они занимают меньше места, да и бардак в **MODELS** разводить не стоит. Таким образом, мы можем и дальше добавлять в нашу библиотеку модели вплоть до 200 штук. Когда эта процедура надоест, помещаем наш файл **TEX301_1.LML** в папку **MODELS** Протеуса. Но на этом наш творческий процесс не закончен. Ведь графические модели, то у нас в библиотеке **USRDVC**, да и в качестве исполняемых им назначены файлы **.MOD**, а не библиотека **.SML**. Для начала снова вытаскиваем наши модели в окно проекта (не перепутайте с теми, что без моделей) и проходим для каждой повторно **Make Device**. На

третьей вкладке удаляем свойство **SPICEFILE** (Рис. 128) и добавляем свойство **SPICELIB** со значением **TEX301_1.LML** (Рис. 130).

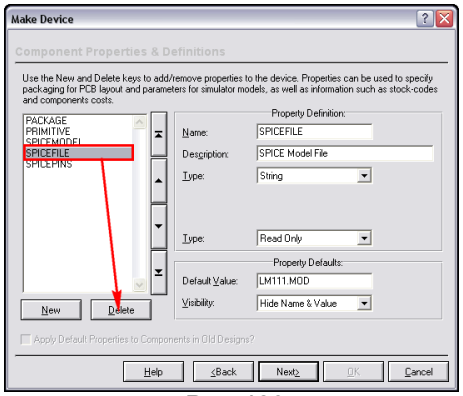


Рис.129

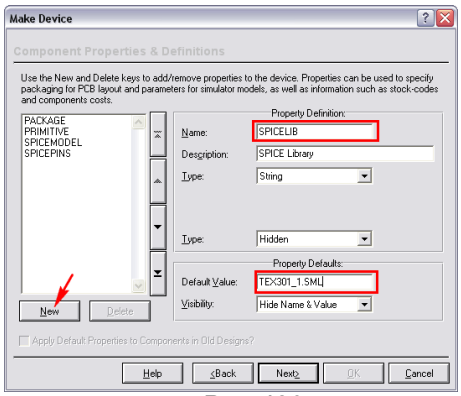


Рис.130

Теперь нам осталось зайти в менеджер библиотек - **Library => Library Manager** и перенести наши модели из **USRDVC** в **TEXOA** (Рис. 131), где им законное место.

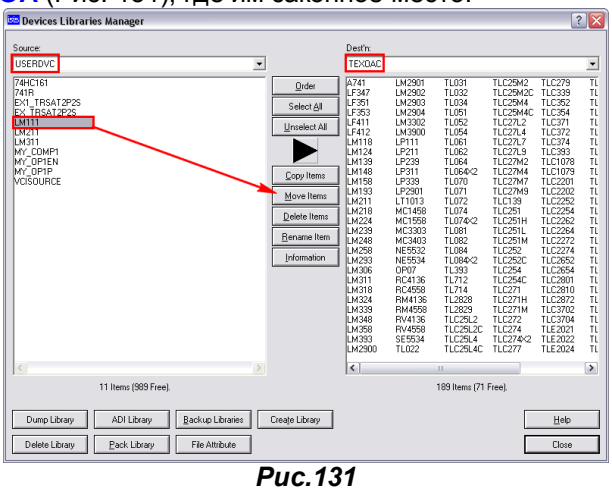


Рис.131

Однако это не так-то просто сделать. Во-первых, для файла **TEXOA.LIB** в папке **LIBRARY** Протеуса необходимо снять атрибут «только чтение». Во-вторых, **TEXOA.LIB** забита по завязку, поэтому сначала надо удалить там существующие **LM111**, **LM211** и **LM311**, выбирая их и используя кнопку **Delete Item**. Лишь потом, кнопка **Move Item** станет активной, как на Рис. 131. Ну, вот теперь наши мытарства закончены, и мы имеем в библиотеках Протеуса готовые к использованию компараторы от **Texas Instruments** со SPICE-моделями вместо опостылевшей фразы **No Simulation**. В принципе, сняв заранее защиту записи с библиотеки **TEXOA.LIB,** можно было сразу перезаписывать модели там. Но все же я не рекомендую проводить такие операции тем, кто не очень «дружит» с компьютером, ну или заранее создайте резервные копии модифицируемых библиотек – файлов с расширениями **.LIB** и **.IDX**, чтобы иметь возможность вернуть все на круги своя в случае неудачи.

Теперь рассмотрим второй вариант добавления моделей. Для этого сначала перестрахуемся и создадим резервную копию родной библиотеки - файлы **TEXOA.LIB** и **TEXOA.IDX** из папки **LIBRARY** и файл **TEX301.SML** из папки **MODELS** Протеуса. С файла **TEXOA.LIB** снимаем защиту от записи и при операции **Make Device** сохраняем изменения в существующей модели не в **USRDVC**, а непосредственно в **TEXOA**. Затем, конечно предварительно сняв атрибут «только чтение», с помощью **PUTSPICE.EXE** упакуем наши модели в родную библиотеку **TEX301.SML** так, как мы делали после создания новой. Все остальные процедуры по удалению в свойствах **SPICEFILE** и добавлению **SPICELIB** проводим также как и выше, ну естественно задав для **SPICELIB** значение родной **TEX301.SML**.

Аналогично поступаем и с моделями операционных усилителей. Примеры тестирования трех ОУ: **LM318**, **LM358**, **TL022** во вложении в папке **OU_TEST**.

Конечно же, таким способом можно добавить и отсутствующие в библиотеках Протеуса модели, но предварительно потребуется создать графическую модель и назначить ей соответствующий **Package** (корпус). Рассмотрим это на примере инструментального усилителя **INA111**, PSPICE-модель которого есть в вышеупомянутом архиве. Я хочу остановиться на этой модели потому, что на данном примере можно рассмотреть сложную субсхему SPICE. По цоколевке корпуса и назначению выводов этот усилитель совпадает с существующим в Протеусе **INA122**, поэтому процесс создания будет достаточно простым. Мы просто втаскиваем в проект прототип и в процессе создания (**Make Device**) меняем везде **INA122** на **INA111**. Для этого на первой вкладке изменяем **Name** на **INA111**, на третьей убираем свойство **SPICELIB** и добавляем **SPICEFILE** со значением

INA111.MOD. Аналогично изменяем значение **SPICEMODEL** на **INA111**. Ну и самое главное – это не забыть изменить последовательность выводов в свойстве SPICEPINS, иначе модель работать не будет. Дело в том, что у прототипа (кстати, особо любознательные могут распаковать модель **INA122** из **LM111_PSpice_Model.zip** и посмотреть в шапке на фамилию автора – весьма познавательно, испытайте гордость за соотечественника) применена иная, чем в модели **INA111** последовательность перечисления выводов, а это важно для симулятора **ProSPICE** (об этом я уже упоминал). Теперь о том, в чем изюминка SPICE-модели INA111. Приведу только начало файла **INA111.MOD**, поскольку он слишком большой.

```
* INA111 = A1_111 + A2_111 + A3_111 OP AMPS + PRECISION RESISTOR NETWORK
* CREATED ON 12/1/92 AT 10:00 HB
* REV.A
* REV.B  18 JUNE 97 NPA: MOVED LEGAL DISCLAIMER TO BEGINNING OF FILE
*
*
***** INA111 SUB-CIRCUIT
* CONNECTIONS:         NON-INVERTING INPUT
*                      |  INVERTING INPUT
*                      |  |  POSITIVE POWER SUPPLY
*                      |  |  |  NEGATIVE POWER SUPPLY
*                      |  |  |  |  OUTPUT
*                      |  |  |  |  |  REFERENCE
*                      |  |  |  |  |  |  GAIN SENSE 1
*                      |  |  |  |  |  |  |  GAIN SENSE 2
*                      |  |  |  |  |  |  |  |
.SUBCKT INA111         1  2  3  4  5  8  9  10
*
***** A1_111 SUB-CIRCUIT
* CONNECTIONS:         NON-INVERTING INPUT
*                    |  INVERTING INPUT
*                    |  |  POSITIVE POWER SUPPLY
*                    |  |  |  NEGATIVE POWER SUPPLY
*                    |  |  |  |  OUTPUT
*                    |  |  |  |  |
X1                15  17  3  4  11  A1_111
***** A2_111 SUB-CIRCUIT
* CONNECTIONS:         NON-INVERTING INPUT
*                    |  INVERTING INPUT
*                    |  |  POSITIVE POWER SUPPLY
*                    |  |  |  NEGATIVE POWER SUPPLY
*                    |  |  |  |  OUTPUT
*                    |  |  |  |  |
X2                15  16  3  4  12  A2_111
*
***** A3_111 SUB-CIRCUIT
* CONNECTIONS:         NON-INVERTING INPUT
*                    |  INVERTING INPUT
*                    |  |  POSITIVE POWER SUPPLY
*                    |  |  |  NEGATIVE POWER SUPPLY
*                    |  |  |  |  OUTPUT
*                    |  |  |  |  |
X3                14  13  3  4  5  A3_111
```

Обратите внимание на то, что внутри файла модели усилителя **.SUBCKT INA111** присутствуют еще три подсхемы усилителей, начинающиеся соответственно с **X1**, **X2** и **X3**. Сами эти усилители в модель не входят, поскольку ранее встречается завершающий оператор **.ENDS**. Их содержимое расписано уже за ним в соответствующих **.SUBCKT** для **A1_111**, **A2_111** и **A1_111** также завершающимися собственными **.ENDS**. В этом можете убедиться сами, открыв файл **INA111.MOD** в любом текстовом редакторе. Вот такая «навороченная» модель. Ну и еще одна особенность, которую мне пришлось поправить для использования в Протеусе. В конце пришлось закомментировать (поставить звездочку в начале строки) следующую директиву: **.ENDS INA111**, иначе **ProSPICE** при симуляции злостно ругался и выдавал ошибку. На этот факт следует обратить внимание тем, кто самостоятельно будет применять модели сторонних разработчиков. В остальном модель оказалась полностью работоспособной, что подтверждается ее тестированием в приложенном файле **TEX_INS\IN_AMP_TEST\INA111_PSpice_Model\INA111.DSN**. В папке **INA118_PSpice_Model** аналогично созданная модель другого инструментального усилителя. Здесь в файле **INA118DFREQ.DSN** дополнительно приведены частотные графики при различных значениях усиления, которые можно сравнить с присутствующими в даташитах на данные компоненты. Даташиты PDF «весят» немного, поэтому я их также вложил в одноименные папки.

****

Ну и в заключение материала о SPICE-моделировании приведу готовые библиотеки с нашими компонентами, которые я подогнал под Протеус, воспользовавшись некоторыми моделями О. Петракова. Для некоторых компонентов я просто использовал импортные аналоги. Библиотека получилась не такая уж обширная, да и использовать данные компоненты стоит с большой оглядкой – я их работоспособность тестировал бегло. Во всяком случае – это лучше, чем совсем ничего. Для использования данных библиотек необходимо распаковать архив **LIBRUS.RAR** и просто переложить файлы из соответствующих папок **MODELS** и **LIBRARY** в одноименные папки вашей копии Протеуса.

FAQ (ЧаВО) по PROTEUS для начинающих и не только.
ЧАСТЬ III. PROTEUS для фанатов.

Содержание:

5. Иерархия проектов Протеуса.

5.1. «Пойми, студент, сейчас к людям надо помягше, а на вопросы смотреть ширше…»(к/ф «Операция Ы и другие приключения Шурика»).

До сих пор мы в основном строили дизайны в ISIS на одном листе проекта. Также мы знаем, что всегда можно добавить дополнительные листы с помощью меню **Design** -> **New Sheet**. При этом последующие листы могут иметь отличные от первого форматы. Например, первый лист имеет формат A4, второй – A3, третий – A4, а четвертый – A1. Это так называемый «плоский» проект, поскольку все листы принадлежат к одному уровню. Если рассматривать бумажный аналог, то это как бы один том многотомника. Мы можем листать страницы и переходить с первой на вторую, пятую десятую и назад. В пределах плоского проекта действует глобальная нумерация элементов и цепей (проводов) и шин питания. Все это хорошо, когда наш проект содержит десяток-другой элементов. Но, представим себе, что мы разрабатываем многополосный фильтр или даже просто стереофонический усилитель с двумя идентичными каналами. И что? Будем рисовать одно и то же дважды, трижды и т.д. Нет, конечно. Как и в обычных бумажных вариантах, мы можем структурировать наш проект. Например, основной лист будет содержать только блок схему, с которой будут ссылки на другие листы, содержащие отдельные принципиальные схемы блоков (модулей). Мало того, мы можем применить какие-нибудь нестандартные (пока назовем их так) компоненты, например – датчики, которые тоже будут иметь собственные принципиальные схемы. Программа **ISIS** позволяет связать все это в единое целое, да еще и заставить работать в виртуальном виде. Количество вложенных уровней иерархии может достигать восьми. Графически это можно представить так, как на рисунке 1.

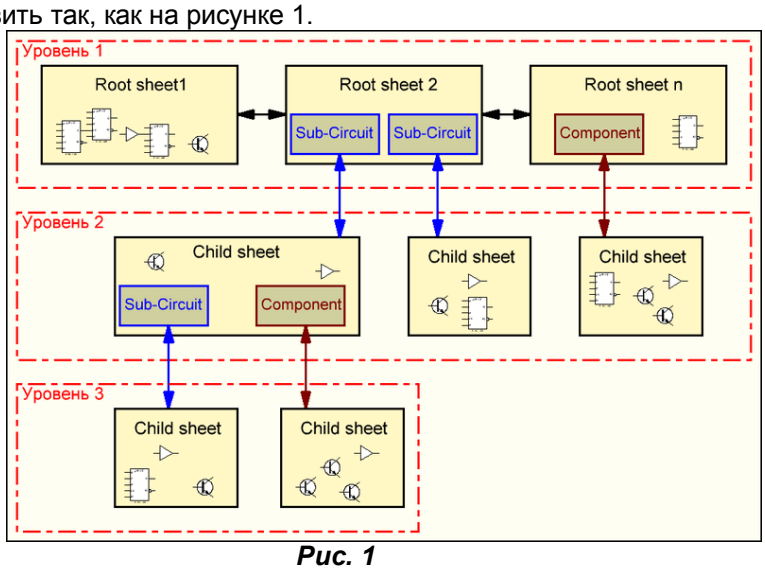


Рис. 1

Из рисунка следует, что непосредственный переход возможен только между листами верхнего (первого) уровня. Выход на уровень два возможен только со второго и третьего листов, причем с определенных элементов схемы: **Sub-Circuit** – подсхем (или модулей) и какого-то конкретного компонента собственной разработки листа 3. Ну а переход на третий уровень в данном случае возможен только с **Child sheet** (дочернего листа) одного из модулей уровня 1 и то в два приема: сначала на дочерний лист уровня 2, а уж с него на третий.

Мы уже пользовались переходами на дочерние листы и обратно, но для тех, кто страдает амнезией, напомню, что переход осуществляется с модуля или компонента щелчком по нему правой кнопкой и выбором опции **Goto Child Sheet** (или **CTRL+C**). У элементов, не имеющих дополнительных листов эта опция неактивна (серая). Возврат на родительский лист осуществляется также щелчком правой кнопкой по свободному от элементов и проводов полю дочернего листа и выбором опции **Exit to Parent Sheet**. Ну и маленькое отступление для поклонников тотального перевода и русифицированных версий. Конечно, **Child Sheet** дословно переводится как «детский лист». Но согласитесь, звучит это как-то несерьезно. Поэтому, еще с первой версии FAQ, я выбрал для себя такую пару **Parent/Child** родительский/дочерний – и в дальнейшем использую только ее. Поэтому, если в вашей русификаторе переведено как детский, то это не ко мне. Я свою стратегию изложения материала менять не собираюсь.

К сожалению, открыв чужой проект ISIS, абсолютно невозможно угадать: какое количество уровней в нем заложено, и какие элементы схемы имеют дочерние листы. Если с модулями вопросов не возникает – они просто не могут функционировать без собственных подсхем, то в отношении компонентов этого не скажешь. Одним из моих любимых занятий на заре освоения Протеуса было тщательное изучение посредством клацанья правой кнопкой мыши по всем компонентам прилагаемых примеров из папки **SAMPLES** установленного Протеуса. Поэтому, на будущее возьмите себе за правило при разработке проектов на листах верхних уровней делать пометки (вставлять текстовые скрипты) о том, какие компоненты схемы содержат дочерние листы. Этим Вы облегчите жизнь и себе, открыв проект через год-два, и другим, если будете пересылать проект на

сторону. На этом краткий экскурс по иерархии можно закончить. Примеров здесь прикладывать не буду, а просто сошлюсь на некоторые характерные из стандартных **SAMPLES**, поставляемых с программой.

Schematic & PCB Layout\Features.DSN – в правом верхнем углу листа проекта стереоусилитель с двумя одинаковыми каналами – модулями с синей рамкой. Каждый модуль содержит дочерний лист. Типичный пример использования модулей.

Schematic & PCB Layout\Epe.DSN – многостраничный проект программатора ПЗУ. На втором и третьем листах содержатся модули, имеющие дочерние листы.

Graph Based Simulation\741.DSN – операционник U1 содержит дочерний лист с внутренней структурой микросхемы. Типичный пример полного схематичного аналогового моделирования.

Graph Based SimulationDAC0808.DSN – микросхема U1 также обладает дочерним листом, но здесь уже смешанное цифро-аналоговое поведенческое (т.е. структура воспроизведена не тотально) моделирование.

Interactive Simulation\Animated Circuits\Osc03.DSN – еще один пример смешанного моделирования таймера 555. На дочернем листе таймера есть пояснения разработчика к модели.

Возврат к содержанию

5.2. Sub-Circuit - «коробочки для схем». Подробнее о модулях и их особенностях.

Ну, вот мы и добрались до первого прообраза будущих наших **Schematic** моделей – модуля или подсхемы (опять для любителей дословной трактовки). Необходимость в применении модулей возникает при использовании в проекте однотипных участков схемы. Поместить модуль в проект очень просто. Выбираем в левом меню режим **Subcircuit** или, щелкнув правой лапкой мышки по полю листа, выбираем **Place => Sub-Circuit** и уже левой кнопкой рисуем модуль нужного размера (Рис. 2).

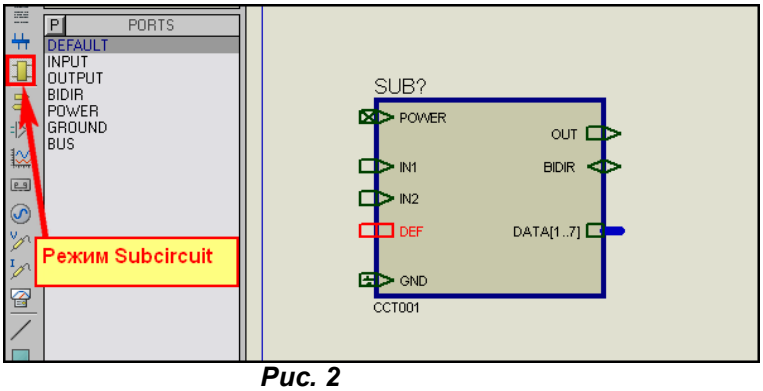


Рис. 2

По умолчанию Протеус подставит модулю имя **SUB?**, которое затем можно изменить, зайдя в свойства модуля двойным щелчком левой или через опцию **Properties** по правой кнопке мыши. После того, как модуль нарисован, настала пора расставить ему порты ввода/вывода, выбирая их в селекторе. Я расставил все возможные, чтобы Вы имели представление, как они выглядят. Здесь надо учитывать одну особенность Протеуса – порты можно ставить только слева и справа тела модуля, но никак не сверху и не снизу – ISIS Вам просто не даст этого сделать. Выбрав нужный тип порта, наводим указатель на левую или правую окантовку модуля и при появлении перекрестия **X** щелкаем мышкой для установки. По умолчанию порты не имеют имени, поэтому после расстановки заходим двойным щелчком в свойства каждого порта и присваиваем ему имя. Для портов-шин действует тот же принцип, что и для обычных шин – в квадратных скобках после имени указываются начальный и конечный номера через **ДВЕ** точки. Еще один принцип, которого рекомендуют придерживаться разработчики: входы размещать слева, а выходы справа. В принципе это не столь криминально, просто общепринято и повышает читабельность схемы, так что если очень хочется, то можно и выходы слева прицепить. Ну вот, собственно и вся процедура размещения модуля в схеме. Дальнейшие действия осуществляются уже на дочернем листе нашего модуля, который становится доступным сразу, как только мы поместили модуль в проект. Для перехода на дочерний лист щелкаем по телу модуля правой кнопкой и выбираем в открывшемся меню опцию **Goto Child Sheet**.

Вот на этом листе мы и скомпонуем нашу подсхему. В качестве «подопытного кролика» для этого упражнения я решил использовать весьма популярные у нас 8-ми разрядные сдвиговые регистры **74HC595.** Возможность последовательной загрузки данных и буферный регистр для их хранения идеально подходят для создания многоразрядной статической индикации на сегментных индикаторах на базе этих микросхем. Тема не нова, и много раз обсуждалась на различных страницах сети, в том числе и на форуме Казус.

Мы здесь рассмотрим создание универсального модуля для Протеуса на основе этих регистров, который затем можно будет перетаскивать из проекта в проект всякий раз, когда нам потребуется вывести информацию на сегментные индикаторы. Применение модуля сократит время на создание проекта и, самое главное, место на основном листе. Итак, поскольку засовывать в модуль один корпус **74HC595** не имеет смысла, их будет два. В этом варианте мы сможем управлять с помощью одного модуля 8-ми разрядным семисегментным индикатором. Поехали…

Открываем новый проект и рисуем модуль. Затем размещаем нужные нам порты. Давайте прикинем: что надо на первом этапе. Входы: для данных – назовем его **DATA**, тактовый для сдвига – назовем его **CLK**, для переноса данных в выходной регистр – назовем его **LOAD**. На первом этапе достаточно, тем более, всегда можно их добавить, что мы и сделаем позже. Теперь поговорим о выходах. Для двух регистров их будет как минимум 16. Если мы будем использовать отдельные порты, то вся выгода по экономии места в проекте теряется. Мы получим вместо двух корпусов один большой модуль с кучей выходов. Давайте спрячем их все в шину – и место не занимает, и доступность сохраняется. А чтобы это было строго по Путински, 8 мух – разрядов в одну, а 8 котлет – сегменты+точка – в другую. И ничто не помешает нам всегда мух с котлетами «замешать». Первоначальный вариант получился как на рисунке 3.

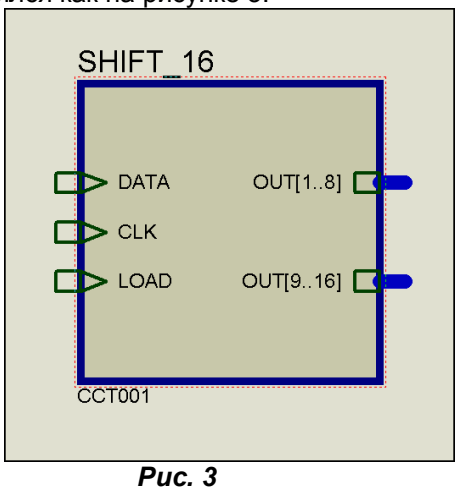


Рис. 3

Переходим на **Child Sheet** и рисуем нашу схему из двух регистров. Для организации взаимосвязи между портами на основном (родительском) листе и схемой на дочернем на последнем используются терминалы из селектора левого меню при выборе режима **Terminals Mode**. Метки (**Labels**) терминалам, когда их немного проще присвоить через раскрывающийся список. При этом там будут представлены именно те имена, которые мы присвоили портам на родительском листе (Рис. 4).

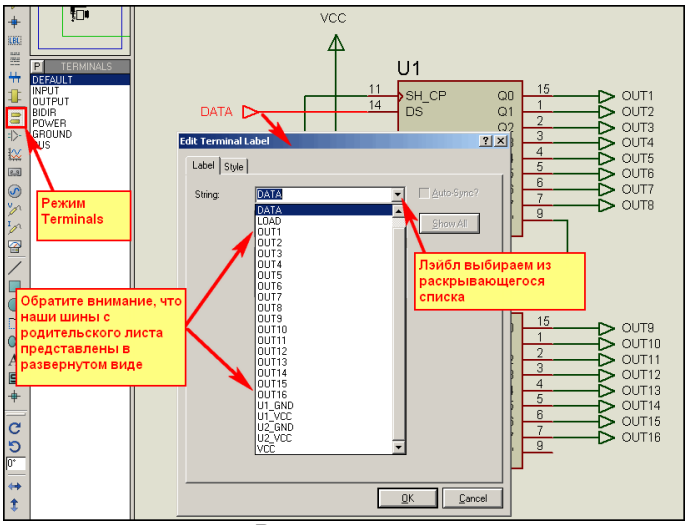


Рис. 4

Немного остановлюсь на представлении шин на дочернем листе. Их можно сформировать как из отдельных терминалов, как это сделано у меня, так и с помощью шины. Во втором случае шине необходимо присвоить лейблы проводникам, входящим в шину и на один из ее концов посадить шинный терминал с именем, совпадающим с шинным портом родительского листа (Рис. 5).

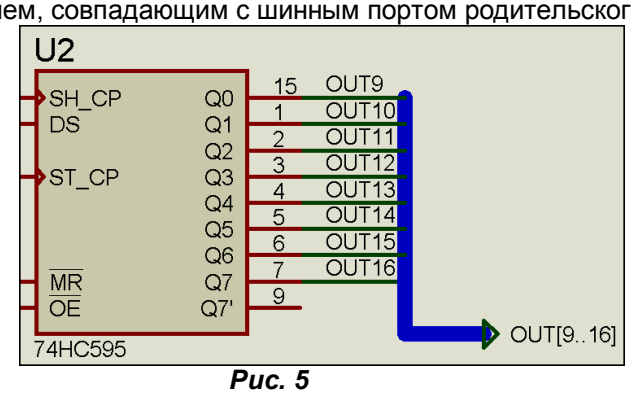


Рис. 5

Первоначальный вариант схемы модуля получился таким, как на Рис. 6.

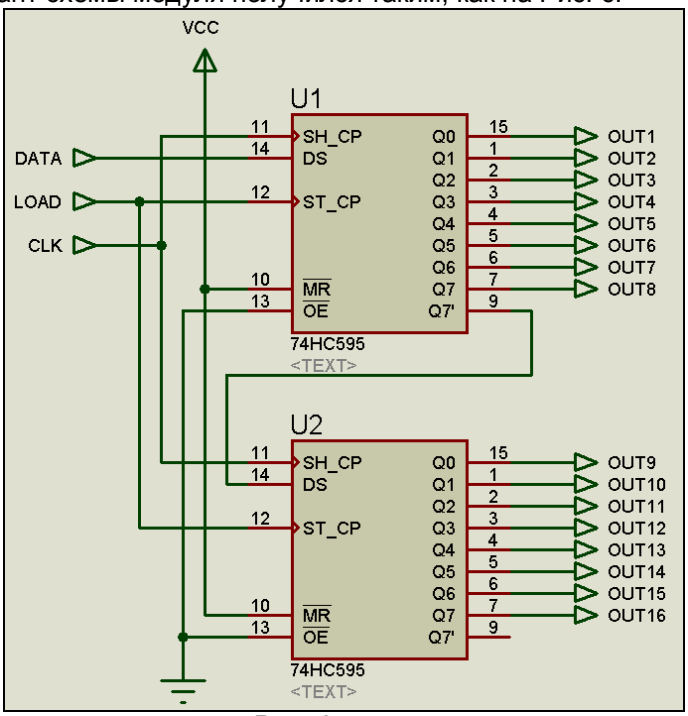


Рис. 6

Однако давайте подумаем – что мы упустили с точки зрения универсальности. Модуль получился уж слишком функционально законченным. А если мне потребуется больше разрядов? Ну, например, захочется использовать 16-ти сегментные индикаторы. Не мешало бы иметь возможность расширения. И она у нас есть – выход **Q7'** второго регистра. Поэтому в окончательном варианте я добавил еще и терминал от него, а на родительском листе соответствующий порт для соединения с входом **DATA** следующего модуля. И еще одно замечание. В реальном устройстве завешивание неиспользуемых входов на питание **VCC,** конечно-же пришлось бы делать через резисторы, но для симулятора это не принципиально.

Ну, вот и весь процесс создания модуля. Теперь переходим на родительский лист и, предварительно навесив, отладочную «мишуру» приступаем к тестированию работоспособности. В качестве отладочных средств на первом этапе очень удобно использовать элементы **LOGICPROBE**, **LOGICSTATE** и **LOGICTOGGLE** из библиотеки **Debugging Tools**. Навешиваем на соответствующие входы-выходы модуля и запускаем симуляцию (Рис. 7).

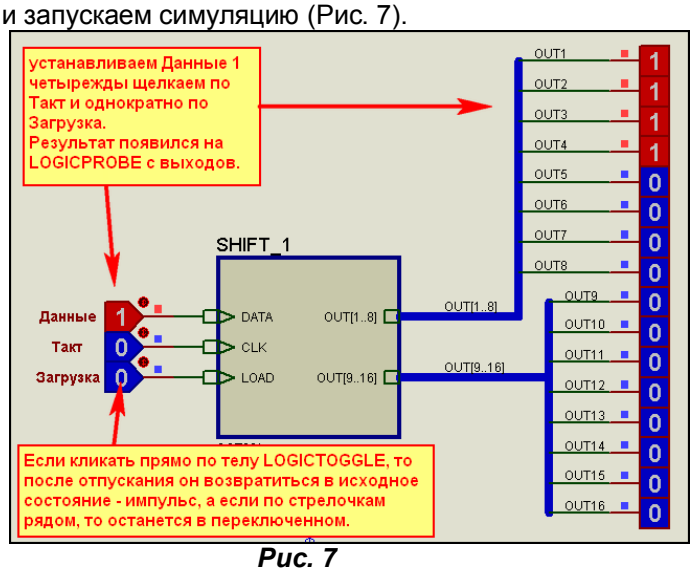


Рис. 7

Первый же тест показал, что модуль работает. Теперь поговорим о некоторых особенностях применения модулей. Во-первых, обратите внимание, что тестирование я проводил с родительского листа. Как модули, так и модульные компоненты, о которых речь пойдет ниже, не должны содержать элементы индикации на дочернем листе. В противном случае вы рискуете либо получить сообщение об ошибке, либо модуль будет вести себя неадекватно. Поэтому, если вы собираетесь поместить на дочерний лист схему, в работоспособности которой не уверены на все 100%, то лучше отладить ее в отдельном проекте, сохранить как **Export Section**, а затем импортировать на дочерний лист вашего модуля. Вторая особенность – если продублировать модуль на родительском листе и попробовать запустить симуляцию, то получим ошибку – сообщающую о наличии дубликата имени, т.е. у второго модуля необходимо сменить хотя бы один символ в имени модуля. Кроме того,

поскольку используется сквозная нумерация элементов, на дочернем листе второго модуля ISIS автоматически присвоит элементам номера, продолжающие общую нумерацию в проекте. Чтобы этого избежать, можно, находясь на дочернем листе, зайти в свойства листа через меню **Design=>Edit Sheet Properties** и установить флажок **Non-physical sheet**. И третья важная особенность – все изменения, внесенные на дочернем листе одного модуля, автоматически отражаются и на другой модуль, расположенный на этом листе и имеющий одноименное свойство **Circuit** (схема). Применимо к нашему примеру, я оставил имя схемы **CCT001**, которое программа ISIS сама присвоила по умолчанию (Рис. 3). Допустим, мы все же решили включить резистор на подтяжку входов **MR** к питанию. Установив этот резистор в одной **Sub-Circuit**, зайдите на дочерний лист другой с **CCT001**, и вы увидите тот же резистор. Ну, вот вкратце все по теме **Sub-Circuit**. Если я что-то и упустил, то оно всплывет далее, поскольку в развитие темы мы переходим к модульным компонентам, у которых много общего с **Sub-Circuit**. Во вложении описанный выше пример создания модуля из двух сдвиговых регистров.

Возврат к содержанию

5.3. Модульные компоненты – более продвинутая разновидность модулей. Сохранение подсхемы во внешнем файле для дальнейшего использования.

Конечно, использование **Sub-Circuit** значительно облегчает жизнь разработчику, но согласитесь, что все равно не очень удобно таскать подсхемы из проекта в проект. А если модуль создан давно, то и разыскать его в нагромождении файлов проектов бывает сложнее, чем создать новый. Поэтому **Sub-Circuit** наиболее удобны тогда, когда одинаковые участки схем встречаются в одном проекте. А как быть, если мы хотим создать модуль для длительного и частого применения? Выход есть. Если мы заглянем в свойства любого компонента, взятого из библиотеки ISIS, то обнаружим там возможность подключения иерархического модуля, т.е. все того же дочернего листа (Рис. 8)

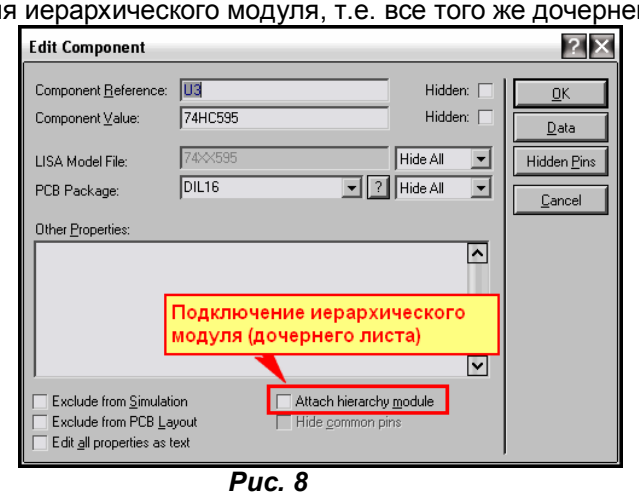


Рис. 8

Причем данная опция присутствует как у существующих в библиотеках компонентов, так и у вновь созданных. Вот этим мы сейчас и воспользуемся. Чтобы сохранить преемственность, превратим наш модуль, созданный в предыдущей главе, в модульный компонент. Для начала создадим графическую модель нашего регистра. Делается это так же, как мы рассматривали в предыдущей части FAQ на примере ОУ. Как и там, рисуем тело компонента, расставляем выводы (pins) и присваиваем им имена. Номера можно и не присваивать – ведь это не реально существующий в природе компонент. Я опять воспользуюсь возможностью сделать выход шинами, чтобы сократить количество отдельных выходов. Итак, в конечном итоге в проекте получилась следующая графика – Рисунок 9.

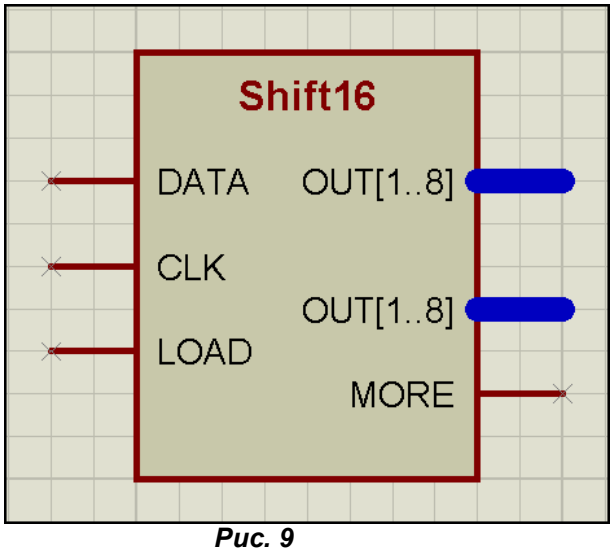


Рис. 9

Пока компонент еще не создан, вы можете это заметить по перекрестиям на концах выводов компонента, к которым у готовой модели будут подключаться проводники. Ну, дальше ничего нового не предвидится, обводим все это творчество мышкой, чтобы выделить и … **Make Device**. На первой вкладке процедуры присваиваем компоненту имя и, если есть желание, то префикс. Конечно же, надо позаботиться, чтобы имя было уникальным и не совпадало с уже имеющимися в библиотеке компонентами. Естественно, что корпуса (**Packagings**) на второй вкладке и какие либо дополнительные свойства на третьей (**Component Properties & Definitions**) мы нашей графической модели не присваиваем. По умолчанию, как я уже неоднократно подчеркивал, ISIS предложит сохранить модель в **USRDVC**. Пусть так и будет, чтоб не путаться. После того, как модель создана она автоматически появится в селекторе компонентов открытого проекта, ну и конечно занесется в библиотеки ISIS в ту категорию, которую вы присвоили на последней вкладке **Make Device**.

Теперь вытаскиваем вновь созданный компонент из селектора в поле проекта, заходим в его свойства и ставим галочку **Attach hierarchy module**. Вот теперь по меню правой кнопки мыши у нас для компонента стал активным **Goto Child Sheet**. Дальнейшие действия ничем не отличаются от процедуры создания модуля. Единственное отличие состоит в том, что в раскрывающемся списке имен терминалов на дочернем листе будут появляться не имена портов, а имена выводов (пинов) нашей графической модели.

Возвращаемся на основной лист, навешиваем к модели тестирующие компоненты и проверяем работоспособность. Ну, казалось бы, все – модель создана и функционирует. Но пять проблемы с перетаскиванием. Ведь внутренняя схема модели находится на дочернем листе. В ISIS предусмотрено решение и этой проблемы. Вспомните, как мы заходили в свойства дочернего листа и ставили галочку **Non-physical sheet**. А ведь там имеется и еще один флажок – **External .MOD file?**. Вот он-то нам и нужен. Устанавливаем этот флажок и давим **OK** (Рис. 10). Теперь в папке с проектом появится файл **SHIFT16.MOD**. Этот файл в сжатой форме содержит нашу схему с дочернего листа.

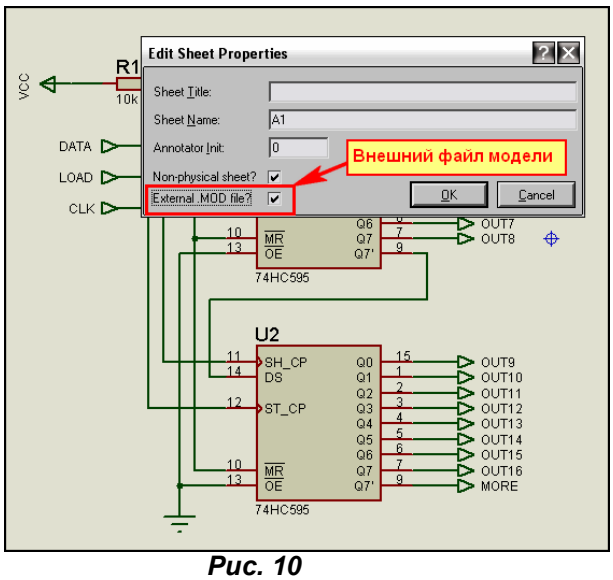


Рис. 10

Дальше придется воспользоваться избитой фразой из телепередачи «Телемагазин на диване»: «Но и это еще не все… в придачу, совершенно бесплатно…» нам необходимо снова провести процедуру **Make Device** для нашего компонента. Задержимся на первой вкладке (Рис. 11).

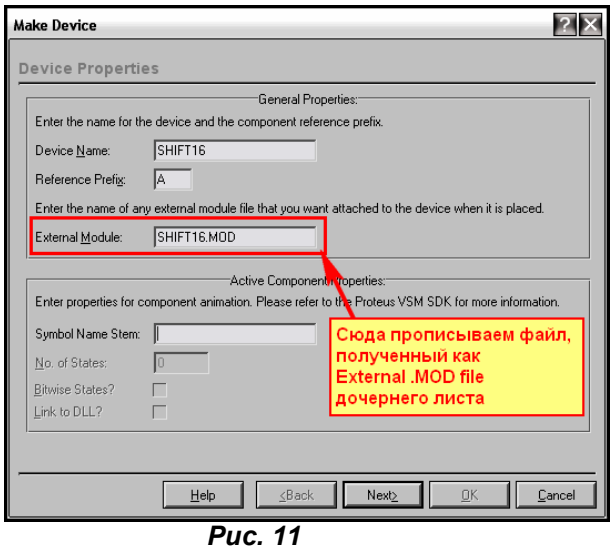


Рис. 11

Раньше мы не обращали внимания на пункт **External Module**, а вот теперь он нам пригодится. Вписываем туда полное имя с расширением нашего файла и проходим процедуру **Make Device** до победного конца. Больше ничего нигде не меняем. Ну и на финальный вопрос Протеуса заменить ли существующую модель ответим утвердительно. Вот теперь наш модульный компонент полностью готов. Тестируем его, как и в предыдущем разделе или автоматизировав процесс (Рис. 12). Здесь я добавил парочку инверторов и цифровой генератор и закольцевал два модуля. Получилась простенькая бегущая строка *(Эх, вспомнил молодость – свой курсовой проект на 22(!!!) корпусах 133ТМ2 – висела у деканата и высвечивала «Вечерний радиоприборостроительный факультет»)*.

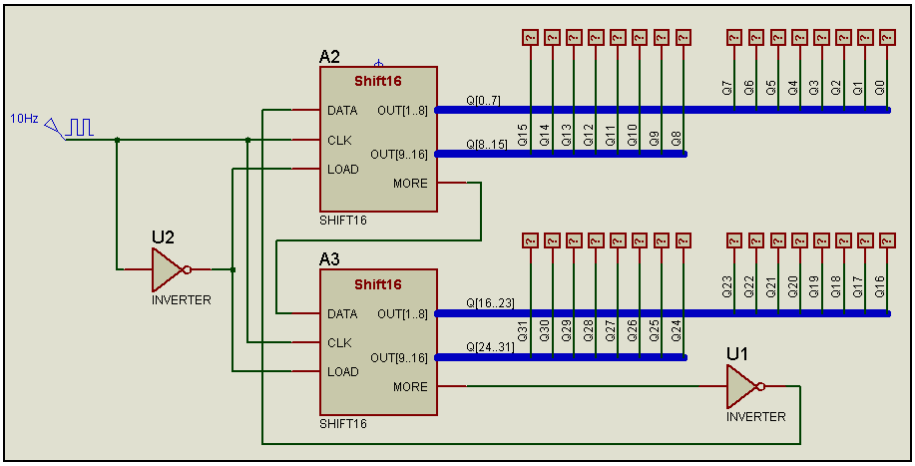


Рис. 12

Ну и как там, в телемагазине… не все, есть еще один нюанс. Наш файл **.MOD** по-прежнему лежит в папке с проектом, т.е. доступен только оттуда. Если мы хотим оставить наш компонент для использования в других проектах, его надо переложить в папку **MODELS** установленного Протеуса. Если этого не сделать, то достав компонент из библиотеки и запустив симуляцию, Вы либо получите сообщение об отсутствии нужного файла **.MOD**, либо просто на выходе компонента не будет никаких сигналов.

На этой особенности остановимся подробнее, поскольку далее в процессе изложения материала я буду выкладывать примеры, новые компоненты в которых работать будут, но для сохранения их у себя необходимо будет проделывать некоторые действия, в частности процедуру **Make Device**, ну и перекладывание файлов моделей в папку **MODELS** программы. Сейчас это **MOD**, далее будут **MDF** и **LML**. Это особенность программы. Файлы моделей **ISIS** сначала ищет в папке с открытым проектом, если их там нет, то в папке **MODELS** Протеуса, а уж если и там не находит, то посылает нас, мягко говоря, в «эротический круиз». Поэтому, даже просто открыв приложенные примеры, вы обнаруживаете работающие проекты – там в самом проекте сохранена модель и графическая и для симуляции, но в вашей копии Протеуса никаких изменений не происходит, пока вы не произведете вышеуказанных действий. Поскольку я такие учебные модели сохраняю у себя в библиотеке **USRDVC**, которую периодически очищаю от мусора, то рекомендую тоже проделывать и Вам.

Ну, вот мы и сделали последний шаг к схематичным моделям. Пора приниматься за них.

Возврат к содержанию

5.4. «Шаг вперёд, два шага назад» (В.И. Ленин). Или опять ОУ – на этот раз идеальный и его Schematic model. Введение в Model Definition Files (MDF).

Ну что это за симулятор без идеального операционного усилителя. В **MicroCAP** есть, в **Multisim**-е есть и в **OrCAD** тоже… А **ISIS**? Да тоже есть, только приберег я его до этого момента, потому что это не встроенная модель как в вышеперечисленных программах, а схематичная – на основе аналоговых примитивов. Для тех, кто подзабыл, ну или просто не знал, напомню определение идеального операционного усилителя. Под идеальным операционным усилителем подразумевается ОУ, имеющий бесконечно большой коэффициент усиления по напряжению в бесконечно широкой полосе частот, а также бесконечно большое входное и бесконечно малое выходное сопротивления. Еще у него бесконечно большое подавление синфазных помех, нулевой температурный дрейф. Конечно, в природе такое супер-творение не встречается, но в моделировании используется часто. Поэтому идеальный ОУ и включен в большинство пакетов моделирования. В ISIS модели идеальных ОУ расположены в отдельной подкатегории **Ideal** в библиотеке **Operational Amplifiers** (Рис. 13). Мы рассмотрим классический трехвыводной ОУ – прямой и инверсный входы и единственный выход. Модель этого ОУ называется **OPAMP**, и в окне предпросмотра можно убедиться, что это **Schematic Model**.

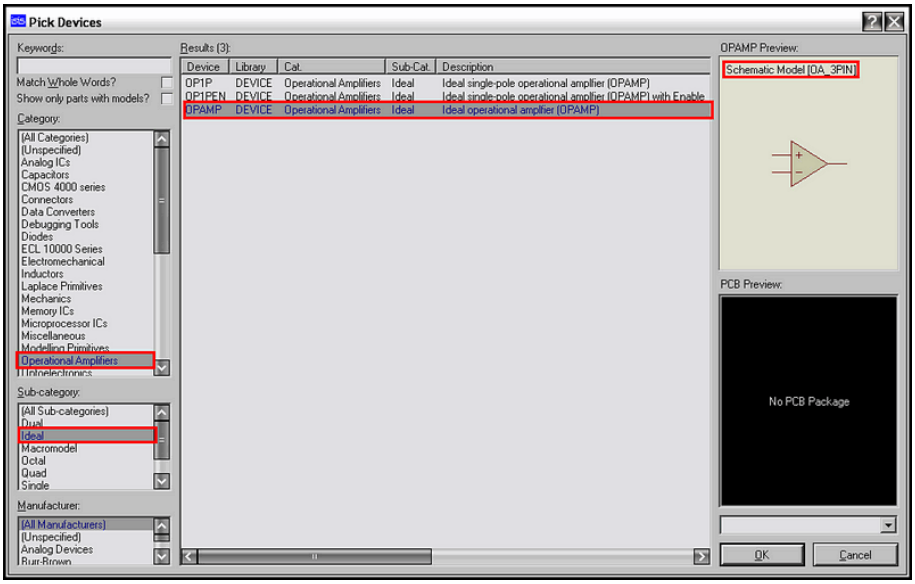


Рис. 13

Для начала достанем наш **OPAMP** из библиотеки и убедимся в его работоспособности и в том, что он отвечает требованиям идеального ОУ. Конечно, если мы начнем его тестировать во всем «бесконечном», то получим массу ошибок – возможности программы и компьютера не безграничны. Поэтому ограничимся неинвертирующим включением с коэффициентом усиления 2 и полосой частот до 1 ГГц. Желающие могут протестировать и с другими Ку, изменив соотношение (R1+R2)/R2. Результаты представлены в **TEST_OPAMP/IDEAL.DSN** и на Рис.14. Ну что, похоже, действительно – «идеальный».

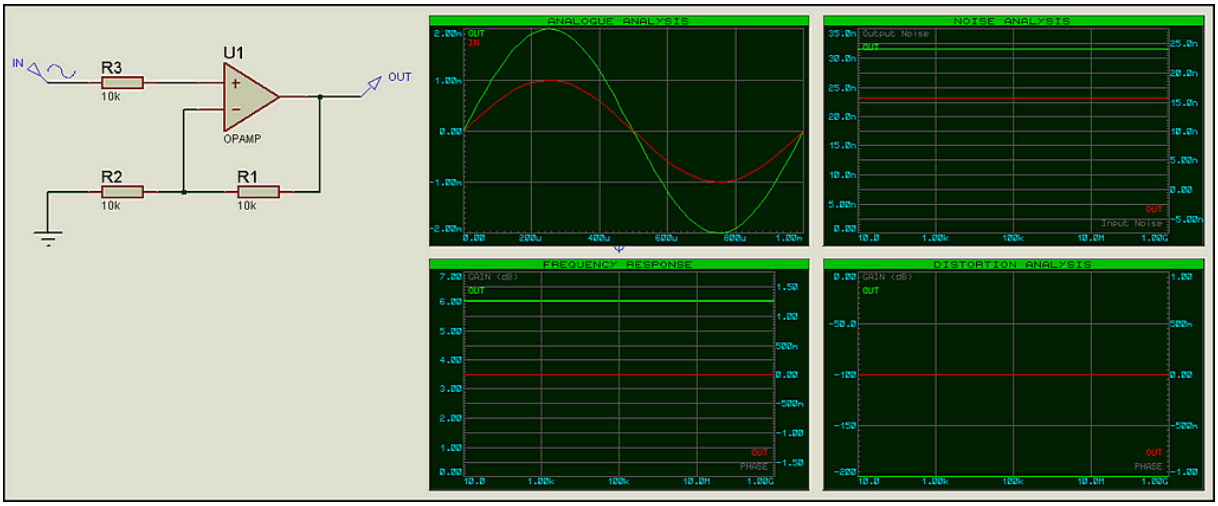


Рис. 14

Теперь займемся реинкарнацией схематичной модели **OPAMP**, ведь всегда интересно посмотреть – что там внутри. Для этого нам потребуется утилита **GETMDF.EXE**, с которой мы уже знакомы по **п.4.12** предыдущего раздела FAQ и файл библиотеки **LML**, содержащей наш **MDF**. **MDF** – расшифровывается как **Model Definition File** (файл назначений модели). Он содержит всю информацию о входящих в модель компонентах и их связях. Обратите внимание, что **MDF** модели в библиотеке **LML** называется не **OPAMP**, а **OA_3PIN**. Обнаружить это можно тремя способами.

Во-первых, в окне **Preview** (Рис. 13) имя **MDF** стоит в скобках.

Во-вторых, если модель уже стоит в проекте, входим в **Properties** и ставим галку **Edit all properties as text**. Обнаруживаем: **{MODFILE=OA_3PIN}**

Ну и третий, экзотический способ. Можно запустить утилиту **Make Device** для **OPAMP** и дойти до третьей вкладки **Component Properties & Definitions**. Там смотрим значение по умолчанию (**Defaut Value**) для **MODFILE**, а затем нажимаем **Cancel**.

В любом случае нам необходимо найти библиотеку с **OA_3PIN** в папке **MODELS** Протеуса. Тут уже каждый действует в меру своих наклонностей и навыков. Я, например, как приверженец Total Commander с незапамятных времен, использую его возможности поиска, потому что на дух не выношу стандартный «собачий» поиск Винды, да и не находит он при стандартных условиях поиска нужный файл – только что проверил. Ну, уж если совсем никак, то можно последовательно открывать каждый файл с расширением **.LML** (их в версии 7.6 всего 21) в текстовом редакторе и искать в нем текст **OA_3PIN**. Для тех же, кто использует Total Commander или аналогичные файловые менеджеры вводим в условиях поиска файл ***.LML** с текстом **OA_3PIN** и в секунды находим нужный нам **ANALOG.LML**. Далее все как в **п.4.12**. Копируем этот файл и утилиту **GETMDF.EXE** в отдельную папку и запускаем ее из консоли следующей командой:

GETMDF.EXE –L=ANALOG.LML –A

Можно и не указывать расширения:

GETMDF –L=ANALOG –A

Не забудьте про пробелы перед ключами, начинающимися с тире. В результате (вот чем мне нравится эта утилита!) мы получим более ста отдельных файлов с расширением **MDF**, среди которых и нужный нам файл **OA_3PIN.MDF**. Все остальное в помойку, а его мы будем рассматривать, открыв любым редактором текста. Для наиболее ленивых пользователей он приложен в папке **MDF_FILES**. Там же еще несколько файлов моделей ОУ и компараторов из этой библиотеки для самостоятельного изучения.

Откроем **OA_3PIN.MDF** в текстовом редакторе и познакомимся с его содержимым. Начало файла нас мало интересует – там информация о создателях и дате создания и модификации. Нас же интересуют разделы, начинающиеся с символа звездочки *****.

Раздел свойств ***PROPERTIES** содержит пять свойств:

```
*PROPERTIES,5
GAIN=1E6
VNEG=-15
VPOS=15
ZI=1E8
ZO=1
```

Мне кажется, даже непосвященному легко догадаться, что **GAIN** – усиление, **VNEG** и **VPOS** – отрицательное и положительное напряжения питания, **ZI** и **ZO** – соответственно входное и выходное сопротивления ОУ.

Далее следует пустой раздел назначений по умолчанию для модели – ***MODELDEFS,0**.

Следующий раздел ***PARTLIST** интересует нас особо, поскольку в нем перечислены все компоненты (примитивы), входящие в схему модели:

```
*PARTLIST,8
D1,DIODE,,N=100m,PRIMITIVE=ANALOG,TEMP=27
D2,DIODE,,N=100m,PRIMITIVE=ANALOG,TEMP=27
R1,RESISTOR,<ZO>,PRIMITIVE=PASSIVE
R2,RESISTOR,<ZI>,PRIMITIVE=PASSIVE
R3,RESISTOR,<ZI>,PRIMITIVE=PASSIVE
V1,VSOURCE,<VPOS>-100m,PRIMITIVE=ANALOG
V2,VSOURCE,<VNEG>+100m,PRIMITIVE=ANALOG
VCI1,VCISOURCE,<GAIN>/<ZO>,PRIMITIVE=PASSIVE
```

Всего примитивов 8 – число в заголовке раздела после запятой. Два диода D1 и D2 с жестко прописанным коэффициентом инжекции **N=100m**. Тройка резисторов R1, R2 и R3 с сопротивлениями **ZO** и **ZI** соответственно, которые (!!!) численно прописаны выше в свойствах. Далее следуют два примитива источников напряжения **VSOURCE**, обеспечивающие питание модели и также численно прописаны в **PROPERTIES**. Они понижены от заданных по умолчанию на 100 милливольт. Ну и завершает список «главное действующее лицо» - управляемый напряжением источник тока **VCISOURCE** с коэффициентом передачи **GAIN/ZO** – это и есть коэффициент усиления ОУ.

На что в списке компонентов хотелось бы обратить внимание.

- Использованы только примитивы. Это не обязательное правило, но очень существенное. Вот свежий бытовой аналог. Только что подошла жена и гонит в магазин за продуктами. Вход в магазин (повернулся и посмотрел в окно) навскидку 200м по прямой. У меня два варианта: «примитивно» дотопать напрямую ножками или поехать на Хендае, который торчит под окном у подъезда, но при этом придется сделать небольшой «крюк почета» вокруг газона и детской площадки, да еще заводиться, с кем-то разъезжаться на двух пересечениях дорог и парковаться у магазина. Делайте вывод – что быстрее. Вот так и со схематичными моделями – примитивно, но быстро или из готовых библиотечных компонентов, которые отягощены своими, подчас противоречащими нужным, свойствами.
- Те свойства, которые прописаны переменными в разделе **PROPERTIES** присваиваются компонентам в угловых скобках (знаки больше меньше). Здесь уместно и применение простых формул, как с коэффициентом у **VCI1** и напряжениями питания, в которых из **VPOS** минусуется и к **VNEG** плюсуется 100mV.
- Поскольку **ZO** стоит в знаменателе дроби оно не должно быть равно нулю. Об этом следует помнить, задавая пределы изменений величин свойств на третьей вкладке **Make Device**. Для данного параметра надо будет установить вариант **Positiv, NonZero**.

Далее идет раздел ***NETLIST**, содержащий непосредственно список цепей нашей схематичной модели. Список совсем небольшой, всего 6 цепей и мы уже рассматривали – как такой список формируется, когда знакомились с **Netlist Compiler** в соответствующем разделе **п.4.4**. Просто еще раз напомню и укажу некоторые особенности. Узлы, не имеющие внешних связей, начинаются со знака решетки, далее следует пятизначный списочный номер цепи и через запятую количество подключений к узлу. В следующих строчках перечислены непосредственно подключения. Например,

к узлу **#00001** подключены две цепи: анод диода D2 и вывод + источника V2. Те узлы, которые имеют внешние выводы компонента (на дочернем листе терминалы) начинаются с имени этого вывода (терминала). Например, неинвертирующий вход **+IP** имеет 4 подключения, а именно: два входных терминала +IP и POS_IP (это для универсальности в графической модели неинвертирующий вход можно обозвать и так и так), вход P источника VCI1 и вывод 1 резистора R3.

```
*NETLIST,6
#00000,2
D1,PS,K
V1,PS,+

#00001,2
D2,PS,A
V2,PS,+

OP,5
OP,OT
VCI1,PS,+
R1,PS,1
D1,PS,A
D2,PS,K

+IP,4
+IP,IT
POS_IP,IT
VCI1,PS,P
R3,PS,1

-IP,4
-IP,IT
NEG_IP,IT
VCI1,PS,N
R2,PS,1

GND,7
GND,PT
V1,PS,-
V2,PS,-
R3,PS,2
R2,PS,2
VCI1,PS,-
R1,PS,2
```

Ну, а завершает файл раздел внешних шлюзов, который как всегда пуст - ***GATES,0**.

Приступаем к воссозданию модели идеального трехвыводного ОУ. Для начала нам потребуется графическая модель, которую можно создать самостоятельно, а можно и просто **Decompose** существующую и убрав все лишнее – текстовый скрипт. На всякий случай убедимся, что наименования выводов соответствуют тому, что мы видели в MDF. На Рис. 15 я их слегка раздвинул, и сделал **Name** видимым для наглядности.

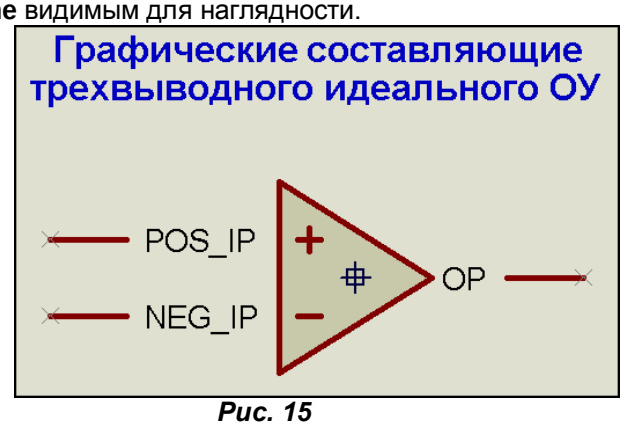


Рис. 15

Теперь сохраняем нашу графическую модель **Make Device** с собственным именем, например **MY_OPAMP** в **USRDVC**. Пока я ей никаких свойств и уж тем более корпусов не присваивал, да корпус идеальному ОУ и не нужен. Ну и теперь, как и для модульного компонента, присоединим к ней дочерний лист. После перехода на дочерний лист, рисуем схему, сообразуясь с разобранным выше MDF. Набираем нужные примитивы из библиотеки **Modelling Primitives** и соединяем их между собой в соответствии со списком цепей. В результате этого довольно нудного и кропотливого творчества у меня получилась на дочернем листе следующая схема (Рис. 16):

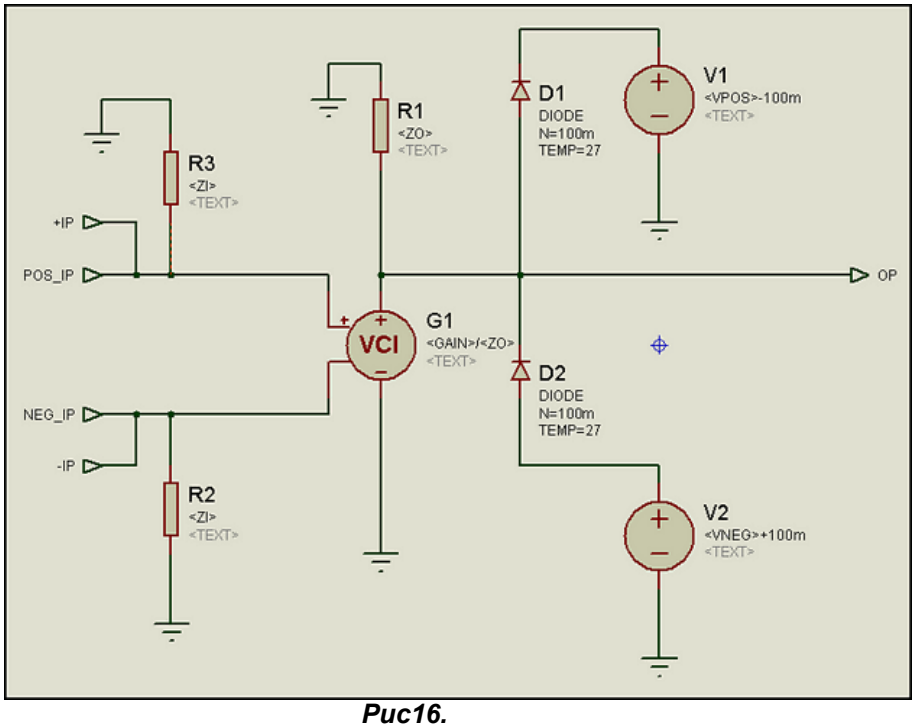


Рис16.

Вы можете заметить, что вместо числовых параметров значений для компонентов подставлены переменные, и формулы из раздела **PARTLIST** MDF-файла. Чтобы все это заработало и Протеус не ругался на отсутствие числовых (numeric) значений у компонентов желтыми «горчичниками» на дочернем листе необходимо поместить текстовый скрипт (левое меню **Text Script Mode**) со значениями, принятыми по умолчанию. Содержание скрипта будет следующим:

```
*DEFINE
GAIN=1E6
VNEG=-15
VPOS=15
ZI=1E8
ZO=1
```

Фактически я полностью скопировал раздел ***PROPERTIES** из фала **MDF**, только заменил слово PROPERTIES словом DEFINE. Ну вот, теперь можно вернуться на основной родительский лист проекта и попробовать протестировать – что у нас получилось. Если наш модульный компонент (а ведь пока это практически он) ведет себя, как и его прототип **OPAMP** из библиотек Протеуса, то все мы сделали правильно. Реинкарнация схемы модели из **MDF** состоялась. Но пока мы не сделали ничего сверхъестественного, ведь это тот же модульный компонент с дочерним листом, как и в предыдущем параграфе.

Все, хватит испытывать терпение публики – делаем MDF. Почему-то все считают, что это очень сложно и доступно только корифеям. А между тем, процедура очень проста. Обязательно уходим вновь на дочерний лист – это важно, поскольку компилятор работает именно с активного листа. Теперь заходим в верхнее меню **Tools**, где выбираем опцию **Model Compiler**. **ISIS** тут же предложит сохранить файл **MDF** с названием как у нашего дизайна в папке **MODELS**. Но меня это не устраивает. Зачем захламлять папку учебными моделями, достаточно сохранить ее в папке с нашим проектом. Ведь при запуске симулятора сначала модель все равно ищется в папке проекта. Да и имя лучше изменить на одноименное с моделью, или подходящее по смыслу. Сохраняю, как **IDEAL.MDF**. Вот теперь можно открыть этот файл в текстовом редакторе и сравнить с исходным **OA_3PIN.MDF**.

Конечно-же полного совпадения не будет. Во-первых, данные в шапке файла будут соответствовать текущим системным, ведь файл только что создан. Во-вторых, наш скрипт ***DEFINE** благополучно превратился в ***PROPERTIES**, как и в исходном файле. Вот как раз он и ***PARTLIST** должны полностью совпадать с прототипом. Если это не так, то где-то вы что-то упустили. А вот ***NETLIST** по содержанию должен совпадать, а по порядку следования узлов и цепей может и отличаться. Обусловлено это следующим. Вы не можете точно воспроизвести порядок действий первоначального разработчика, т.е. порядок прорисовки им схемы на дочернем листе. Поэтому может оказаться, что цепь с номером **#00001** в вашем **MDF** окажется с номером **#00005** или **#00013**. Кроме того, у двухполюсных компонентов: резисторов, конденсаторов, катушек наименования выводов скрыты и обычно просто носят цифровой вид **1** и **2**. Для нас и для симуляции абсолютно без разницы первый или второй вывод подключены к данному узлу схемы, но компилятор MDF воспроизводит это буквально. Поэтому там, где в исходном **MDF** стоит, например, **R2.PS,1** в новом файле окажется **R2.PS,2**. Соответственно надо убедиться, что на его место в другом узле угодил

первый вывод. Вот такие нюансы надо учитывать, когда вы пытаетесь полностью восстановить схему по чужому **MDF**. Еще раз подчеркну, что на работоспособности модели такие перестановки абсолютно не сказываются.

Итак, файл MDF создан, проверен, но пока лежит без дела. У нас по-прежнему висит приделанным дочерний лист, с которым и работает наша модель. Пришла пора вновь запускать **Make Device**. Нам все чаще и чаще предстоит к ней обращаться. На третьей вкладке через кнопку **New** добавляем нашей модели свойство **MODFILE** и прописываем ему в значении **IDEAL.MDF** (Рис. 17). В принципе, расширение можно было и не указывать, **ISIS** все равно для этого свойства будет искать именно **MDF**, но на первых порах лучше перестраховаться. Пока больше ничего не добавляем, будем последовательными и не станем торопить события. Вновь доходим до конца процедуры и давим **OK**.

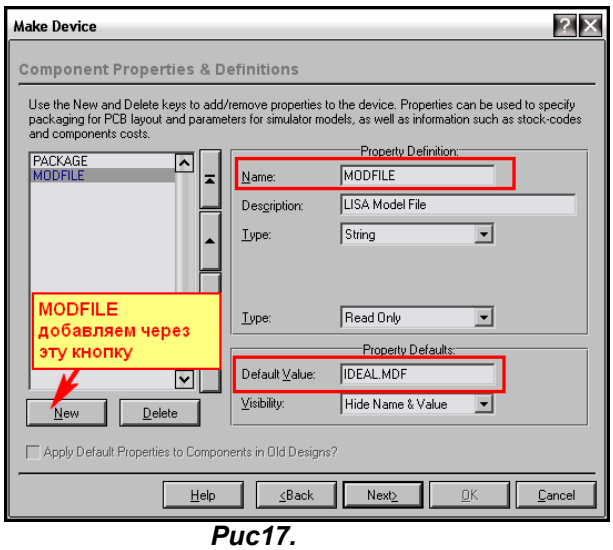


Рис17.

Теперь, если мы вытащим из селектора модель **MY_OPAMP** в поле проекта, то мы уже не сможем попасть у нее на дочерний лист, поскольку при добавлении **IDEAL.MDF ISIS** лишил нас такой возможности. Но зато мы получили свою первую **Schematic model**. Если перетащить файл **IDEAL.MDF** в папку **MODELS** программы, то мы можем ее использовать в любом другом проекте, просто добавив в него **MY_OPAMP** из библиотек Протеуса. Но вот беда, свойства то у него можно менять только в окне **Other Properties**, набирая вручную. А у прототипа **OPAMP** они присутствуют в виде набора дополнительных окон. Делать нечего, придется научиться и этому «фокусу». Вперед, на третью вкладку **Make Device**. Конечно-же, в списке стандартных по кнопке **New** мы их не найдем. Ведь для каждой модели такой набор может отличаться. Поэтому там мы выбираем **Blank Item** (Пустой пункт) и набираем нужные нам параметры вручную. Давайте добавим для примера усиление, а в добавке остального можете попрактиковаться сами. Процесс приведен на Рис. 18.

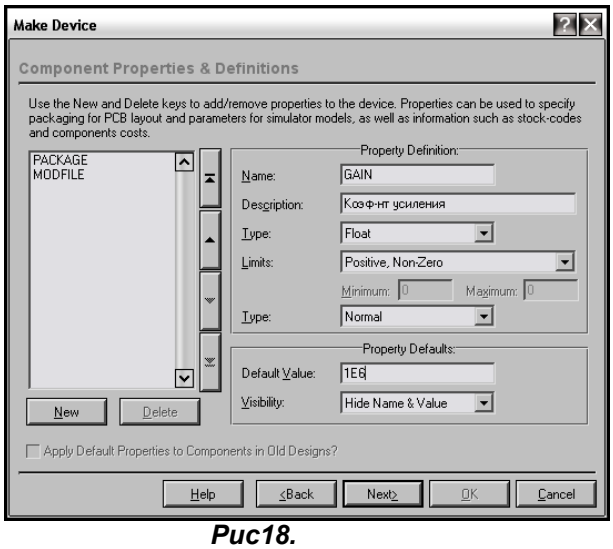


Рис18.

Итак, рассмотрим подробно.

- В графу **Name** заносим наименование нашего параметра так, как он выглядел у нас на дочернем листе и в MDF, т.е. для усиления это будет **GAIN**, а например, для положительного питания **VPOS**;
- В графу **Description** - краткое описание свойства. Эх, порадую поклонников русификаций, здесь уместен даже русский язык, и чтобы доказать я набрал **Коэф-нт усиления**;
- В графе **Type** выбираем тип значения нашего свойства. Поскольку он достаточно большой – выбираем **Float** (с плавающей запятой);

- В графе **Limits** выбираем ограничения, которые будет контролировать **ISIS** для нашего значения. Наше усиление должно быть **Positive, Non-Zero** (положительным, не равным нулю). При этом окна минимума и максимума не активны;
- В следующей графе **Type**, одноименной с той, что двумя этажами выше выбирается, как будет отображаться наше свойство: **Normal** – отображается с окном доступным для изменения, а если поставить, например, **ReadOnly** – то менять мы его уже не сможем, окно будет серым.
- В графе **Default Value** задаем численное значение нашего свойства **1E6** (миллион);
- Ну и последняя графа **Visibility** определяет, будет ли видно наше свойство под моделью при добавлении ее в проект.

Завершив процедуру **Make Device** до конца, в **Properties** нашей модели мы увидим следующую картинку (Рис. 19).

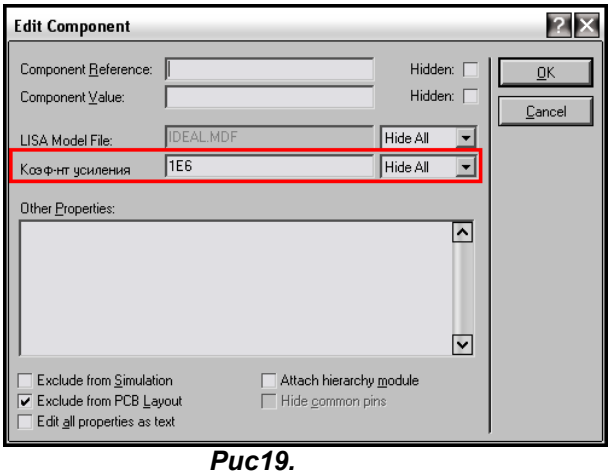


Рис19.

Аналогичным способом добавляются и остальные свойства модели, причем их можно было добавить и сразу все в одном проходе **Make Device**.

Подведем итог вышеизложенному материалу и, наконец, распрощаемся надолго с аналоговыми моделями, потому что нас ждет новая тема создания цифровых и смешанных схематичных моделей.

1. При использовании одинаковых участков схем в одном проекте проще и быстрее создать модули **Sub-Circuit**, однако для долговременного использования лучше подходят модульные компоненты.
2. Изучение чужих файлов MDF-компонентов дает нам возможность понять принципы построения схематичных моделей, обнаружить недокументированные свойства и применить их для создания собственных моделей. Труд этот кропотливый и требует терпения и внимательности для достижения нужных результатов.
3. При создании собственных схематичных моделей с компонентов лучше всего подходят примитивы, т.к. наименее нагружают компьютер и тормозят симуляцию.
4. При создании любых компонентов все «активные» элементы не должны располагаться на дочернем листе.
5. MDF-файлы или MOD-файлы должны располагаться либо в папке с разрабатываемым проектом, либо в папке **MODELS** программы Протеус, чтобы быть доступными симулятору при запуске.
6. Поскольку мы создавали идеальную модель, мы воспользовались встроенными источниками напряжения, при моделировании реальных компонентов потребовались бы выводы питания у графической модели, т.е. как у той модели, что мы делали для SPICE.

На этом пока все по этой теме. Во вложении в папке **OPAMP_REMAKE** процесс воссоздания модели, а в папке **MY_OPAMP_MDF** создание схематичной модели **MY_OPAMP** с **MDF**.

Возврат к содержанию

6. Создание схематичных цифровых (Digital) и смешанных (Mixed) моделей.

6.1. Цифровые, аналого-цифровые и цифро-аналоговые примитивы и их свойства.

Если для создания аналоговых схематичных моделей необходимы аналоговые примитивы, которые мы подробно рассмотрели в предыдущей части FAQ, то для создания цифровых моделей используются цифровые, которые значительно проще и во много раз быстрее работают. Вся информация по этим примитивам доступна в HELP Протеуса **ProSPICE Primitives** в разделе **Digital Modelling Primitives** по цифровым и **Mixed Modelling Primitives** по аналого-цифровым и цифро-аналоговым примитивам. Я не стану здесь рассматривать каждый вариант также подробно, как для аналоговых, поскольку у большинства у них набор свойств и параметров почти совпадает. Нам необходимо усвоить общую тенденцию формирования названий свойств и тогда в любой момент вы сможете извлечь это из собственной памяти, а если есть сомнения, то подсмотреть в HELP. Помощь по конкретному элементу всегда доступна непосредственно из окна проекта. Есть как всегда два варианта попадания в HELP: щелкаем по элементу правой кнопкой мыши и выбираем опцию **Display Model Help** или входим в окно **Edit Component** и там нажимаем кнопку **Help** справа. Все цифровые и смешанные примитивы располагаются в том же разделе библиотек Протеуса, что и аналоговые, но разбиты по нескольким суб-категориям (Рис. 20).

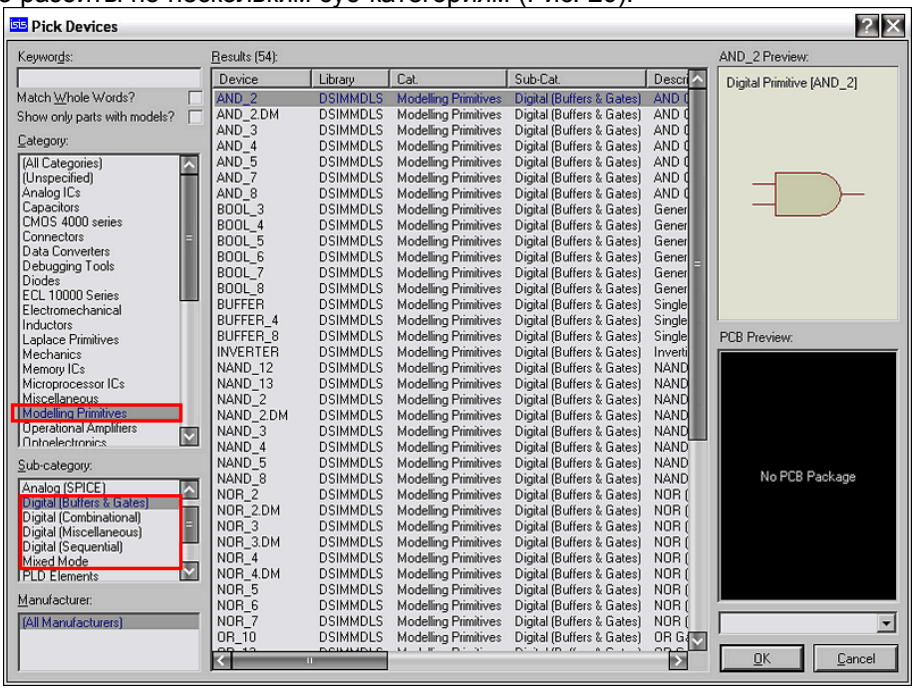


Рис20.

Если со всевозможными сложными по структуре компонентами вопросов не возникает, то по элементарной логике необходимы некоторые пояснения. Для примера на Рис. 21 я поместил «разобранный» элемент двухвходового И у которого включил подсветку наименований выводов и обычный D-триггер, у которого имена и так уже включены. Итак, у любого логического элемента входы имеют имена **D0**, **D1** и т.д., а выход именуется **Q**. Вот это и важно усвоить, чтобы понять следующий материал.

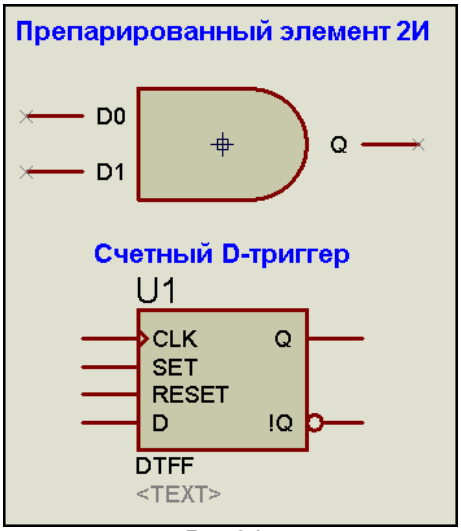


Рис21.

Зайдем в вышеупомянутый HELP, например для той же элементарной логики - раздел **The Standard Gate Models**. Нас интересуют в первую очередь временные параметры. Я полностью приведу их здесь.

Time	Type	From/To	Edge	Default	Notes
TDLHDQ	Delay	D# => Q	L => H	0	
TDHLDQ	Delay	D# => Q	H => L	0	
TGQ	Glitch	Any => Q	Pulse	TDxxDQ	

По сути именно они определяют быстродействие логического элемента той или иной серии микросхем: ТТЛ, КМОП или ЭСЛ. Давайте попробуем расшифровать ту абракадабру, которая находится в первом столбце, применив элементарные знания английского языка.

Возьмем, например, **TDLHDQ**. Раскладываем: **T**ime **D**elay – временная задержка; **L**ow – низкий; **H**igh – высокий; **D** – это вход ЛЭ; **Q** – выход. А теперь, как персонаж Крамарова из «Джентльменов удачи», произнесем это на нормальном, гражданском языке: «Временная задержка передачи переднего фронта сигнала с входа на выход логического элемента». Проверяем себя по столбцам. Действительно: во втором столбце тип указан **Delay** – задержка, в третьем **From/To** (от… / к…) указано от **D** с решеткой (напомню, что в Протеусе решетка – это номер), в четвертом **Edge** (фронт) указан с **L** на **H**, т.е. передний с 0 к 1, ну и значение по умолчанию указано **0**, т.е. задержка в примитиве полностью отсутствует. Аналогично можно разобрать и вторую строку, только там речь идет о заднем фронте, поскольку стоит сочетание **HL**.

Для третьего параметра **Time Glitch Q** переходная задержка импульса с любого (**Any**) из входов на выход значение по умолчанию жестко связано с первыми двумя и самостоятельно изменяется в зависимости от них, поэтому на нем останавливаться особо не будем.

А вот вариации первых двух в зависимости от типа логики и входа элемента нам будут встречаться довольно часто. Например, у того же D-триггера мы встретим параметры TDLHCQ или TDSQ. Нетрудно догадаться по аналогии, что первый из них временная задержка передачи переднего фронта со счетного входа (английское **C**lock) на выход **Q**, а вторая с входа предустановки **S**. Для элементов с третьим состоянием в аббревиатуре сокращения появится символ **Z** (например, TDLZOQ). Надеюсь, общая тенденция построения Протеусной «фени» для временных параметров ясна, и в дальнейшем не вызовет у вас затруднений, тем более, что всегда можно заглянуть в HELP. Но, забегая вперед, отмечу, что в большинстве случаев все прочие задержки привязаны по умолчанию к первым двум, которые мы рассмотрели. Поэтому, если они не оговорены особо, то и изменяются автоматически вместе с **TDLHDQ** и **TDHLDQ**.

Теперь рассмотрим несколько свойств, которые разбросаны по всему HELP, а иногда и не описаны, но представляют при моделировании определенный интерес.

INIT – начальное состояние. В зависимости от типа элемента при старте симуляции переводит его в определенное состояние. Например, если для триггера в окне **Other Properties** свойств записать строку **INIT=1**, то при старте он окажется «взведенным», т.е. на выходе Q будет 1, не !Q будет 0. Для счетчиков или сдвиговых регистров это свойство устанавливает начальное состояние счетчика (Рис. 22).

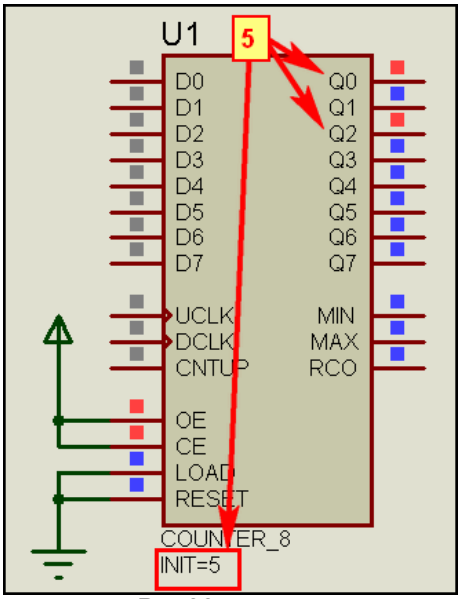


Рис22.

ARESET и **ALOAD** – еще два свойства, присущих примитивам счетчиков и регистров. Первое из них разрешает асинхронный сброс, а второе асинхронную загрузку. По умолчанию оба равны **FALSE**, для разрешения записываем, например сброс – **ARESET=TRUE**.

LOWER и **UPPER** – для примитивов счетчиков позволяют установить соответственно нижний и верхний предел счета. По умолчанию нижний предел 0, а верхний 2n-1, где n –число разрядов.

Например, если для примитива восьмиразрядного счетчика на Рис. 22 записать **UPPER=10**, то мы превратим его в десятичный счетчик.

INVERT – это свойство позволяет проинвертировать состояние любого входа/выхода цифрового примитива. Например, если для примитива D-триггера (Рис. 21) записать **INVERT=SET,RESET** (обратите внимание, что после запятой перед RESET пробел отсутствует!!!), то асинхронная установка/сброс триггера будут производиться не логическими единицами, а логическими нулями на соответствующих входах. Для счетного входа такая запись означает изменение установочного фронта импульса, т.е. если устанавливался по переднему, то будет устанавливаться по заднему.

SCHMITT – недокументированное свойство, в HELP нигде не описано. Позволяет придать входам логического элемента свойства порогового элемента с гистерезисом переключения – триггера Шмитта. У триггеров Шмитта, например 40106, 4093 включено по умолчанию. В большинстве случаев помогает для обычных логических элементов использовать их в качестве типовых RC-генераторов, т.к. типовые схемы генераторов на логических элементах в ISIS напрочь отказываются работать. Объясняется это тем, что в реальных элементах используются как раз аналоговые свойства входов, которые в моделях не реализованы. Записывается свойство так: **SCHMITT=D0** (включить свойство триггера Шмитта для входа D0).

Еще ряд свойств, присущих отдельным цифровым примитивам мы рассмотрим позже, в ходе их использования для построения схематичных моделей.

А теперь поясню – почему мы здесь же будем рассматривать и некоторые свойства смешанных аналого-цифровых примитивов. Ряд их свойств используется **ProSPICE** и для цифровых моделей, чтобы придать им большее сходство с реальными компонентами. Здесь мы познакомимся с наиболее значимыми, которые пригодятся нам в дальнейшем, а остальные рассмотрим по мере изучения моделей.

- Аналого-цифровые свойства (раздел HELP **Mixed Mode Modelling Primitives => ADC Interface Object Model**):

Здесь сразу же хотелось бы обратить внимание на то, что в отличие от чисто цифровых аналоговый вход элемента объявляется как A, а цифровой выход как D, забегая вперед, для цифро-аналоговых с точностью до наоборот – вход D, выход A.

VTL и **VTH** – расшифровываются как **Voltage Threshhold** (порог переключения по напряжению) соответственно для низкого – **L** и высокого – **H** уровня. Могут записываться как в абсолютных значениях, например, **VTL=0.6** – нижний порог 0,6В, так и в процентах к питающему напряжению **VTH=70%** - верхний порог переключения 70% от напряжения питания.

VLH и **VHH** – **Voltage Low Hysteresis** и **Voltage High Hysteresis** – соответственно гистерезис переключения с неопределенного на низкий и с неопределенного на высокий уровни. Если пороги заданы в процентах, то и гистерезисы должны задаваться в процентах, а если **VTL** и **VTH** заданы в абсолютных единицах, то и гистерезисы должны быть прописаны в них. Обратите внимание, что установка **VLH** и **VHH** в нули может привести к потере сходимости вычислений, т.е. ошибкам симулятора и перегрузке процессора компьютера.

Типичные значения по умолчанию: **VTL=30%, VTH=70%, VLH=VHH=10%.**

- Цифро-аналоговые свойства (раздел HELP **Mixed Mode Modelling Primitives => DAC Interface Object Model**):

VLO, **VHI**, **VUD** – соответственно напряжения для низкого, высокого и неопределенного уровня. По умолчанию **VLO=0%**, **VHI=100%** и **VUD=50%**. Эти свойства применимы к цифровым входам.

RLO, **RHI**, **RUD** – выходные сопротивления для сигналов низкого, высокого и неопределенного уровней. Первые два по умолчанию 1Ом, а **RUD=(RLO+RHI)/2**, т.е. тоже 1Ом. Есть еще параметр **RTS**=100МОм обозначающий выходное сопротивление для высокоимпедансного состояния.

TRISE и TFALL – время подъема и время спада (передний и задний фронты) выходного сигнала. По умолчанию оба равны 1наносек. Эти два временных параметра гарантировано моделируются только в случае, если глобальная переменная **TTOL** в параметрах симуляции **ISIS** (**System => Set Simulator Options**) меньше чем они (Рис. 23)

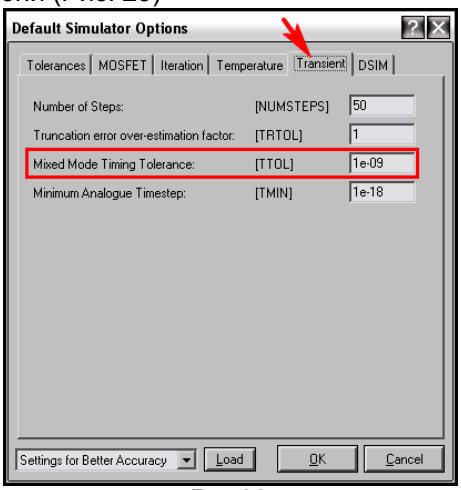


Рис23.

Думаю, что для начала достаточно свойств цифровых и смешанных примитивов, и на этом можно остановиться. Пора рассмотреть – как они используются для моделирования цифровых микросхем различных серий, и переходить непосредственно к моделированию.

Возврат к содержанию

6.2. ITFMOD – MDF-файл, определяющий параметры цифровой логики. Пример модели К176ЛА7.

Когда мы рассматривали в предыдущем параграфе свойства цифровых примитивов, вы, наверное, обратили внимание, что временные параметры примитивов по умолчанию равны нулям, т.е. сигналы обрабатываются без каких либо задержек на прохождение. Но в реальности все обстоит гораздо сложнее. Есть достаточно медленная КМОП логика, есть более быстрая ТТЛ, есть и сверхскоростная ЭСЛ серия. Мало того, существуют и различные модификации, например ТТЛ Шотки и пр. Как упростить задание типовых свойств, характерных для конкретной серии? Ведь не прописывать же каждому элементу все эти задержки, да еще на тарабарской «фене» Протеуса. Разработчики программы придумали хитрый ход. Они сгруппировали характерные свойства для каждой конкретной серии логики и поместили их в файл **ITFMOD.MDF** (нетрудно предположить, что название образовано от английского Interface Models – интерфейс моделей). А сам этот файл поместили в папку **MODELS** Протеуса. Давайте откроем его в текстовом редакторе и посмотрим на содержимое. Я не буду приводить его полностью, а только рассмотрю принцип его построения. Шапка, как и в большинстве **MDF** и она нам неинтересна. Далее следует единственный раздел ***MODELDEFS,18** содержащий восемнадцать строк, в каждой из которых, начинающейся с названия типа, после двоеточия перечислены характерные параметры данной серии. И тут не только обычная логика, а замешались еще и микроконтроллеры.

Рассмотрим для примера первую строчку CMOS, описывающую КМОП логику:

CMOS : RHI=100,RLO=100,TRISE=1u,TFALL=1u,V+=VDD,V-=VSS

Знакомые нам по предыдущему параграфу параметры, причем характерные для цифро-аналогового преобразования. Два последних параметра **V+** и **V-** указывают, к каким глобальным источникам питания привязаны аналоговые свойства входов.

И так мы можем просмотреть свойства в каждой строчке, характерные для каждой конкретной серии. Более того, поскольку файл **ITFMOD** является обычным текстовым, мы можем даже вносить в него исправления и добавления. Только не торопитесь на радостях сразу же «топтать его ногами», чуть позже мы этим займемся все вместе и добавим в него нашу старую, добрую тихоходную серию К176.

А пока рассмотрим – как же используются свойства серии из этого файла на практике. Если вы откроете свойства любого цифрового компонента определенной серии и поставите галочку **Edit all properties as text**, то обнаружите в окне параметров строчку, описывающую интерфейс модели вида: **{ITFMOD=xxx}** (Рис. 24). Вот она то и привязывает модель к определенному семейству (строчке в файле **ITFMOD.MDF**) цифровой логики. Прописывается это свойство при создании модели на третьей вкладке **Make Device** (через кнопку **New** находим его в списке стандартных).

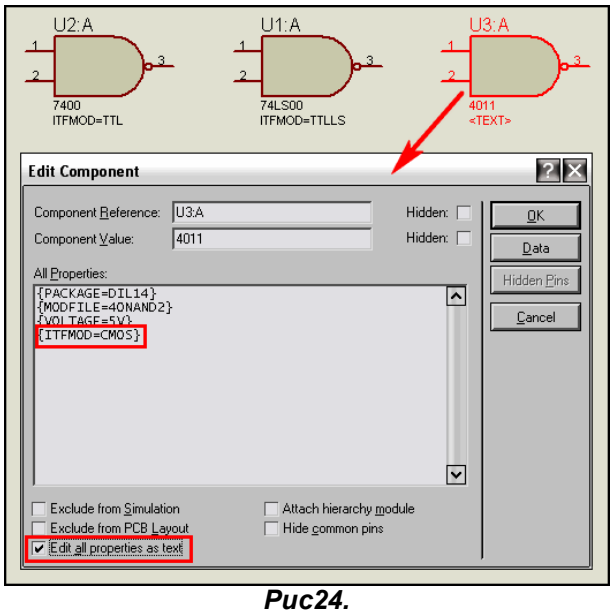


Рис24.

Более того, если вы создадите цифровую модель и не привяжете ее ни к одной из существующего там типа, ISIS будет «громко ругаться» при запуске симуляции. Проверим это на практике. Создадим учебную модель элемента 2И-НЕ 176-й серии в стиле времен СССР. Для начала нарисуем графическое изображение и сохраним его в **USRDVC** под «фамилией» **176LA7**. Вообще, можно было бы использовать ее аналог – **4011**, но мы поучимся творить свое. Я позволил себе при создании парочку вольностей. На первой вкладке создания компонента ввел **Prefix DD**, как принято, было в СССР и сейчас в России и на последней вкладке в описании использовал русский язык (Рис. 25).

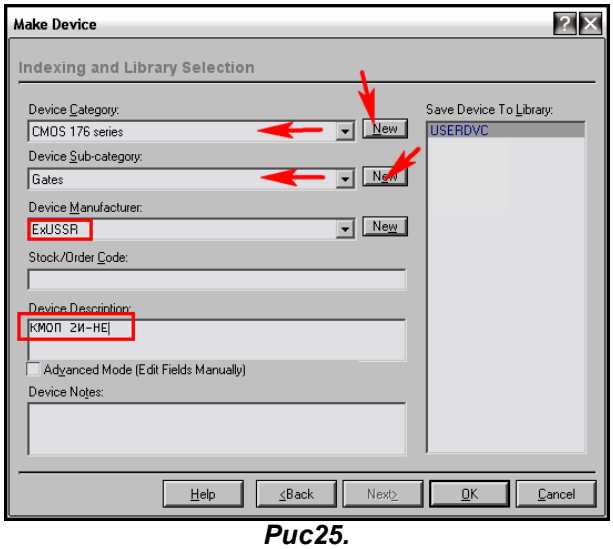


Рис25.

Далее пришпиливаем к ней дочерний лист, как делали ранее и на нем устанавливаем один единственный элемент **NAND2** из **Modelling Primitives => Digital (Bufer&Gates)**, ну и терминалы **A, B, Y** для связи с родительским листом. Сразу отвечу на законный вопрос – а почему не **D0, D1** и **Q** – как я описывал ранее? Да потому, что так принято обозначать в **MIXED** моделях АЦП и ЦАП. Вот и нужно совпадение, чтобы можно было использовать их аналоговые свойства. На рисунке 26 прототип модели и созданная графическая модель, а на рисунке 27 – содержание дочернего листа.

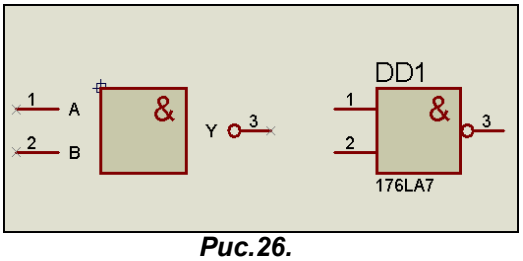


Рис.26.

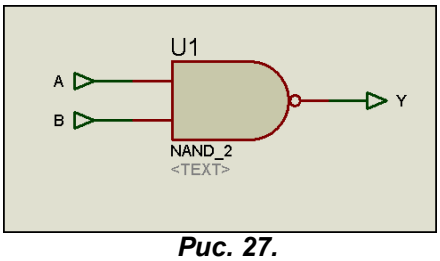


Рис. 27.

Я протестировал модель на достаточно высокой частоте – 20МГц, чтобы показать, что пока фронты импульсов практически минимальны, и обусловлены быстродействием самого симулятора и моего компьютера (Рис. 28). Тестирование на таких частотах возможно уже только с использованием графиков. Обратите внимание, что для тестирования я применил аналоговые графики, а не цифровые. На цифровых графиках мы вообще этой затяжки фронтов не увидим. Затяжка видна уже на выходе генератора, а на выходе логического элемента она просто проинвертирована. Ступеньки также обусловлены уже быстродействием самой программы – это ответ тем, кто симулирует суперскоростные ШИМ и удивляется, что там видны ступеньки. Ведь пока мы используем голый цифровой примитив, т.е. все **TDxxDQ** по умолчанию нулевые. Этот пример во вложении **ITF_MOD** папка **LA7_with_child**.

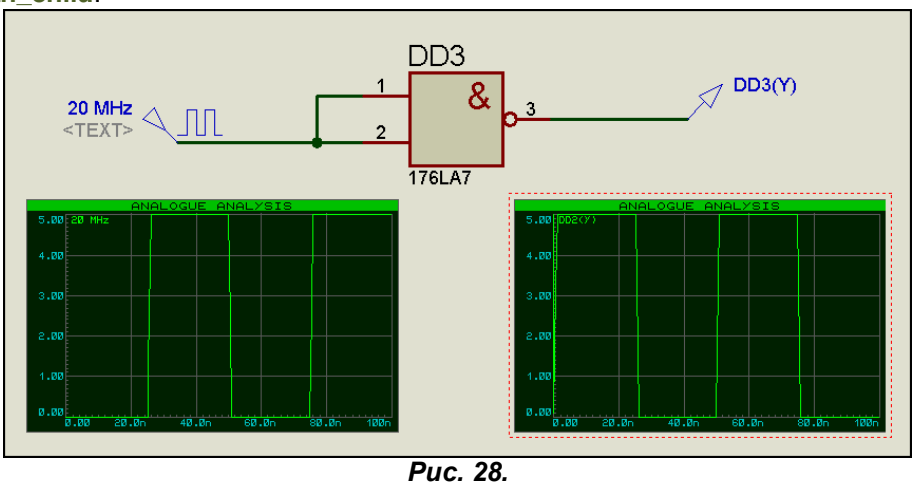


Рис. 28.

Ну а где же обещанные грязные ругательства симулятора? Терпение, ведь мы еще не создали модель до конца. Все же давайте рассмотрим наш прототип 4011 и его особенности подробней. Во-первых, в свойствах компонента присутствует параметр **Model Timing Voltage** (Рис. 29). Если мы через **Make Device** дойдем до третьей вкладки, то увидим, что этому параметру соответствует свойство **VOLTAGE,** и оно может принимать три значения – 5, 10 и 15 Вольт (Рис. 30).

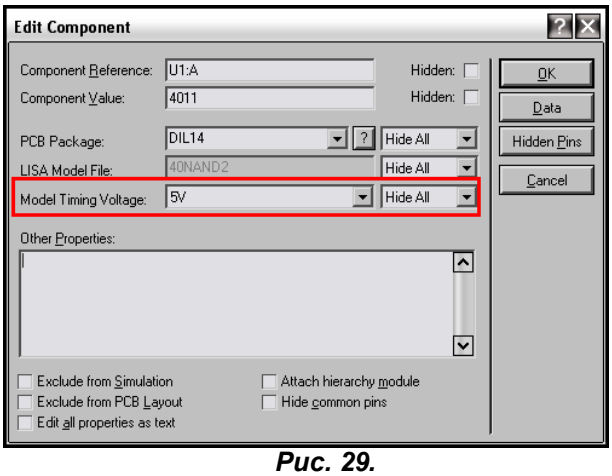


Рис. 29.

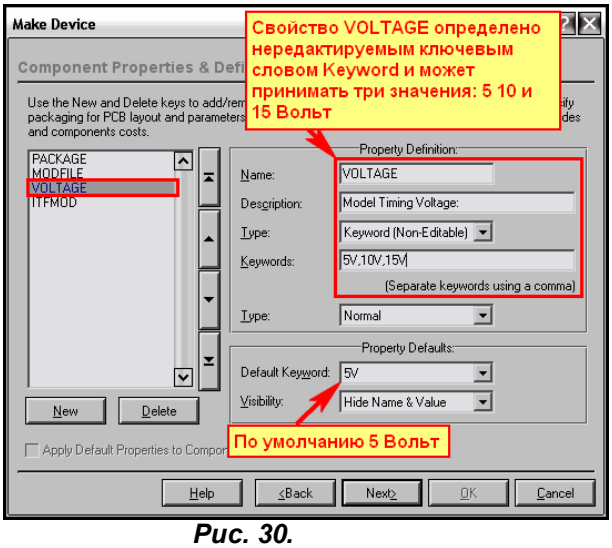


Рис. 30.

Полюбопытствуем, для чего это сделано? Придется добраться до файла MDF этого компонента и поискать ответ там. Смотрим в свойствах, поставив галку **Edit all Properties as text**, или на той же третьей вкладке **Make Device** в свойстве **MODFILE** и выясняем, что файл MDF носит название **40NAND2**. Далее обнаруживаем его в **MODELS** в библиотеке **DIGITAL.LML** и, скопировав ее куда-нибудь, извлекаем с помощью **GETMDF.EXE**. Библиотека очень большая – 962 цифровых компонента. Для ленивых пользователей я поместил файл **40NAND2.MDF** во вложение. Ниже его содержание с «урезанной» шапкой, чтобы не занимать место:

```
*PROPERTIES,1
TGQ=?

*MAPPINGS,6,VALUE+VOLTAGE
4011+5V : SCHMITT=[NULL],  TDHLDQ=55n, TDLHDQ=55n
4011+10V : SCHMITT=[NULL],  TDHLDQ=25n, TDLHDQ=25n
4011+15V : SCHMITT=[NULL],  TDHLDQ=20n, TDLHDQ=20n
4093+5V : SCHMITT="D0,D1", TDHLDQ=90n, TDLHDQ=85n
4093+10V : SCHMITT="D0,D1", TDHLDQ=40n, TDLHDQ=40n
4093+15V : SCHMITT="D0,D1", TDHLDQ=30n, TDLHDQ=30n

*MODELDEFS,0

*PARTLIST,1
U1,NAND_2,NAND_2,PRIMITIVE=DIGITAL,SCHMITT=<SCHMITT>,TDHLDQ=<TDHLDQ>,TDLHDQ=<TDLHDQ>,TGQ=<TGQ>

*NETLIST,3
Y,2
Y,OT
U1,OP,Q

A,2
A,IT
U1,IP,D0

B,2
B,IT
U1,IP,D1

*GATES,0
```

В первую очередь нас интересуют верхние разделы. В ***PROPERTIES,1** параметру **TGQ** присвоен знак вопроса, т.е. значение или явно указанное пользователем или то, что по умолчанию. А вот дальше интересный раздел ***MAPPINGS,** с которым мы еще не сталкивались. Разберем – что там есть. Ну, то, что 6 строк – это сразу понятно, а что означает **VALUE+VOLTAGE**? Вот тут еще один пример универсальности моделирования в Протеусе – в одном MDF заложены сразу две модели. **VALUE** – это то, что прописано в окне **Component Value** (опять смотрим на Рис. 29), ну а **VOLTAGE** мы только что рассмотрели. Так вот в **MAPPINGS** (дословно отборах, отображениях, а по сути своей таблицы соответствий) и осуществляется подстановка параметров в зависимости от сочетания (не зря стоит знак плюс) типа компонента и заданного вольтажа. Все шесть возможных сочетаний и прописаны ниже.

Я не зря привел пример именно этого компонента. Обратите внимание, что здесь используется описанное в предыдущем параграфе **SCHMITT**. Для обычного 2И-НЕ **4011** оно отключено **[NULL]** для 2И-НЕ с триггерами Шмитта **4093** задействовано по входам **«D0,D1»**. А на триггерах Шмитта RC-генераторы в Протеусе работают на ура! Делайте выводы…

Теперь остановимся на задержках фронтов. Числовые значения прописаны именно в **MAPPINGS** и зависят от того, что мы (подчеркиваю – мы, а не сам симулятор) выберем в окне **Model Timing Voltage**. По умолчанию используются максимальные задержки, соответствующие приведенным в справочных данных для напряжения питания 5 Вольт. Надеюсь, теперь вам стало понятно – как использовать этот параметр на практике. Ну а мы далее рассмотрим – как применить это к нашей модели.

В разделе ***PARTLIST** сиротливо притулился один элемент **NAND_2** – все, как и у нас на дочернем листе **176LA7**. Вот только в этой строке прописаны еще и параметры не с конкретными значениями, а находящимися в угловых скобках, т.е. переменными. И подставляются они туда симулятором как раз из **MAPPINGS**. Вот он – момент истины! Но «не все так просто в доме…» Лабцентра. Помните чудесное превращение скрипта ***DEFINE** на дочернем листа в ***PROPERTIES** файла MDF при компиляции идеального ОУ. Тут нас ожидает такой же сюрприз. Скрипт на дочернем листе для таблицы соответствий должен начинаться с оператора **MAP ON**. Поскольку это оператор, как и **DEFINE**, перед ним должна стоять звездочка. Заходим на дочерний лист нашей модели **176LA7** и в свойствах **NAND_2** прописываем аналогично тому, что мы видели в MDF для **4011** задержки и свойство **SCHMITT**. Проще всего это сделать все же руками в окне **Advanced Properties**, поставив (да можно и не ставить) флажок **Edit all Properties as text** (Рис. 31).

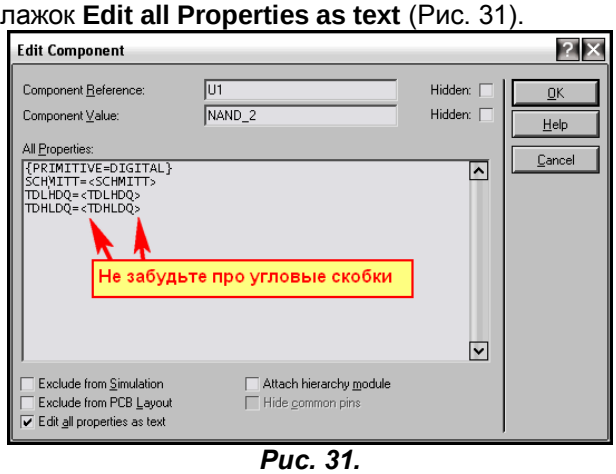


Рис. 31.

Совет: *Обратите внимание на фигурные скобки в верхней строчке на Рис. 31 – свойство Hidden (скрытое), т.е. не показывается под элементом на месте серого <TEXT> в проекте. SCHMITT и задержки не имеют фигурных скобок по краям, значит будут видимыми. Когда свойств и компонентов на листе много, это начинает мешать. Заключите их в фигурные скобки, и они скроются с глаз долой. Надо, чтоб опять стали видимыми – скобки убираем.*

Еще на дочернем листе необходимо поместить скрипт следующего содержания:

```
*DEFINE
TGQ=?
*MAP ON VOLTAGE
10V : SCHMITT=[NULL],  TDHLDQ=250n, TDLHDQ=250n
5V : SCHMITT=[NULL],  TDHLDQ=400n, TDLHDQ=400n
```

Предвижу возникающие вопросы и поясняю. Ну, что во что превратится при компиляции – ясно. Свойство Шмитта я сохранил на будущее, плюс к тому его необходимо будет включать, если мы захотим сделать генератор на нашей модели. Я не стал включать в таблицу стандартное для 176-й серии питание 9V, хотя справочные задержки взяты именно для него. Данная серия прекрасно работает и при 12V, поэтому я и поставил 10 – просто округлил. Ну и для питания 5V задержки фронтов явно увеличатся, это неоспоримый факт. Когда-то мне попадалось, что около 400 наносек, но сейчас не помню, где это было – поставил по памяти. Ну и поскольку у нас одна серия и один компонент, то необходимость в **VALUE** отпала, и оно в **MAP ON** не фигурирует.

Будем последовательными, вернемся на главный лист и снова запустим **Make Device**, чтобы добавить на третьей вкладке свойство **VOLTAGE** (Рис. 32). Конечно же, его нет в стандартных, т.е. добавляем через **New => Blank Item**.

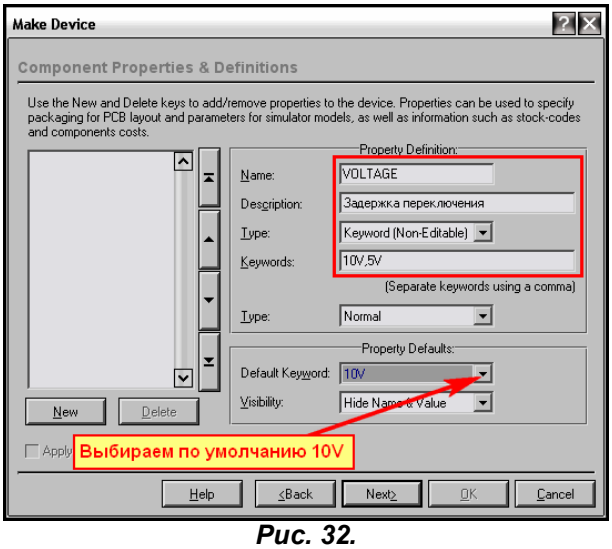


Рис. 32.

Прогоняем тестирование компонента. Теперь уже о 20 МГц нечего и мечтать, даже на 1 МГц видны задержки уже и на цифровом графике. Ну, так оно и должно быть, ведь 176-я серия разрабатывалась как тихоход для часов. Пример со скриптом на дочернем листе в папке **LA7_with_MAP** вложения.

И осталось нам заскочить на дочерний лист и сформировать MDF, а затем подключить его на третьей вкладке **Make Device**. Этот вариант в папке **LA7_with_MDF** вложения.

Ну а когда же Протеус начнет ругаться? Да он бы уже нас давно обложил трехэтажным английским, но мы его обманули в самом начале. Выводы питания то мы не приделывали и корпус не назначали. Ну, выводы питания мне приклеивать лень, а корпус придется назначить, например DIL14, хотя по большому счету шаг выводов у 176-й серии метрический 2,5 мм и немного отличается от 2,54 мм, принятого в импортных DIP (DIL). Ну, где там наш **Make Device** – пошли на вторую вкладку **Packagings**. Там щелкаем кнопку **New** и в раскрывшейся библиотеке корпусов в окне **Keywords** набираем **DIL**, чтобы не рыться по всем библиотекам. Корпус быстро нашелся по совпадению с ключевым словом (Рис. 33).

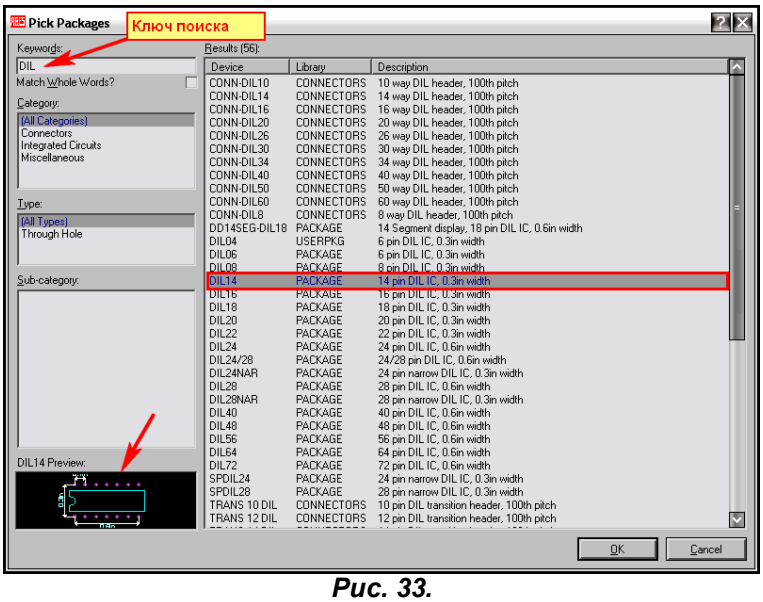


Рис. 33.

Далее назначаем его нашей микросхеме в таком порядке (Рис. 34). Сначала в окне **No of Gate** ставим 4 – ведь в корпусе 4 элемента 2И-НЕ. После этого переходим к назначению выводов и тут же у нас появляются колонки B, C, D и становится доступным установка флажка **Gates (elements) can be swapped on the PCB layout**. Этот флажок позволяет **ARES** менять местами выводы и целиком элементы A,B,C,D для улучшения трассировки печатной платы, поэтому его лучше включить. Проходим последовательно назначение выводов для этих элементов. Назначаемый в данный момент в таблице подсвечивается желтым, свободные (неназначенные) выводы на корпусе в окне справа – лиловым.

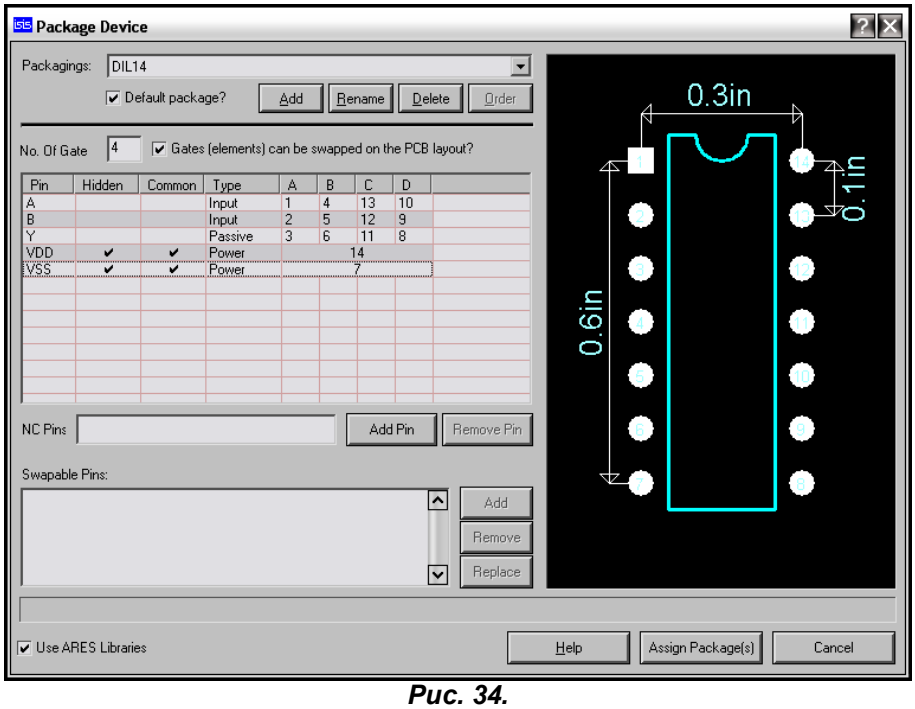


Рис. 34.

Первоначально у нас не будет строк VDD и VSS, ведь у графической модели эти выводы отсутствовали. Поэтому, когда у нас останутся неназначенными последние два вывода 7 и 14, давим кнопку Add Pin и прописываем их имена именно так, как на картинке **VDD** и **VSS**. Протеус автоматически решит, что это Power и поможет нам в этом. Достаточно только назначить номера выводов. Завершаем назначение корпуса кнопкой **Assign Package(s)**. Вот и все, проходим до конца процедуру **Make Device**. Теперь у нас на третьей вкладке будут уже три свойства: **VOLTAGE**, **MODFILE** и **PACKAGE**. Вот теперь и запустим еще раз симулятор. Сначала, вроде, как и ничего, но при остановке получим горчичники типа: **Pin 'VSS' is not modeled** и **Pin 'VDD' is not modeled**. Специально в таком виде лежит во вложении, в папке **LA7_with_PCB**.

Дождались!!! Сейчас мы их…. Пробегаем **Make Device** до третьей вкладки и добавляем через New из списка **ITFMOD** со значением по умолчанию **CMOS** (Рис. 35). Проходим до конца и сохраняем. Этот вариант в папке **LA7_FINAL**. Тестируем – никаких горчичников, потому что, какой вывод питания куда подключен – **ISIS** извлек из файла **ITFMOD.MDF** для строчки **CMOS**.

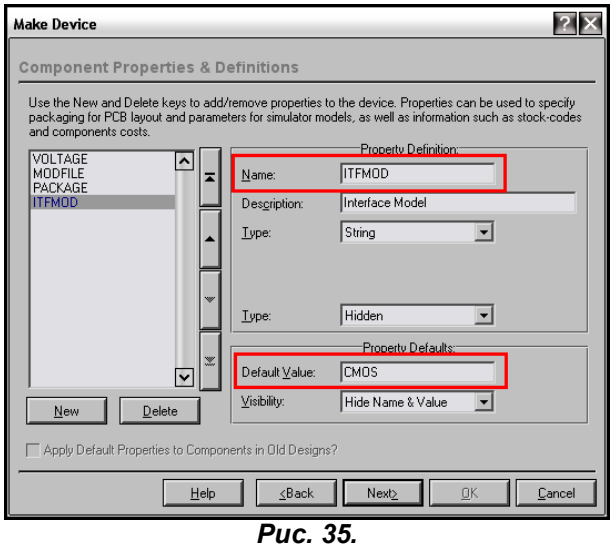


Рис. 35.

Ну вот, материал получился очень объемным, но, надеюсь, и очень полезным. В завершение только добавлю, что все примеры из этого параграфа лучше рассматривать в той последовательности, как они встречаются в тексте. Иначе, сохранив более продвинутую модель через **Make Device** заранее, вы не сможете увидеть некоторые эффекты, например, те же горчичники.

Возврат к содержанию

6.3. Генераторы на RC и LC цепях в Протеусе и несколько способов их запуска. Извечные русские вопросы: «Что делать? » и «Кто виноват? ».

Очень многих начинающих пользователей программы эти два вынесенных в заголовок вопроса начинают терзать уже на первом этапе знакомства с **ISIS** при попытке моделировать генераторы с задающей RC цепочкой. Я уже столько раз отвечал по этому поводу на форуме, что сбился со счета. Пора положить этому достойный конец и разобрать вопрос досконально. Тем более что частично этот материал нам понадобится при моделировании счетчика К176ИЕ12 буквально в следующем параграфе. Почему то на второй из извечных вопросов большинство пользователей отвечает однозначно: «Глюк программы, виноваты разработчики». Особенно прельщает частая добавка: «Да вот я в Мультисиме проверял – там сразу все работает, а в Протеусе никак». Ну, так каждому свое. Нравится – моделируйте там, ну а мы грешные уж как-нибудь справимся и здесь. Ну а теперь к делу…

Основная ошибка при попытке моделирования генераторов – это «тупое» (другого слова, извините, подобрать не могу) перерисовывание схемы генератора и попытка в лоб запустить его. Автор схемы или учебник по электронике гласит, что работает – значит так и должно быть. Ну, что ж «нарисуем» классический мультивибратор на транзисторах и попробуем его запустить (Рис. 36).

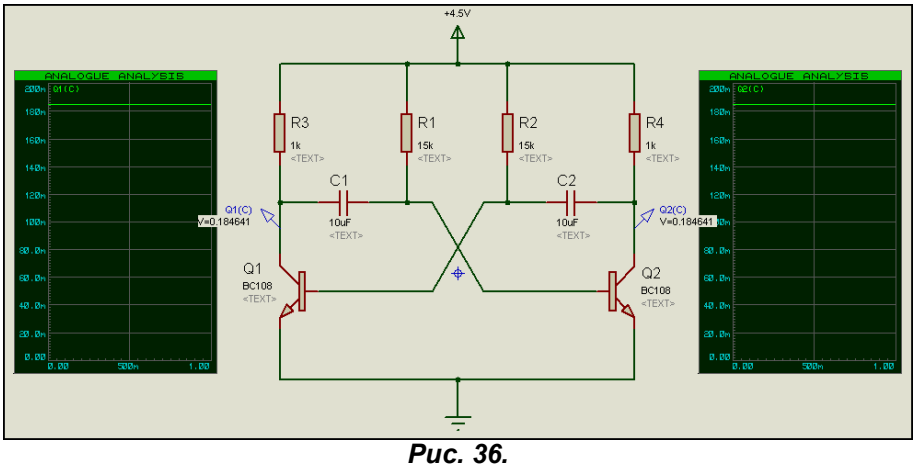


Рис. 36.

Как видим, и в графическом режиме, и в интерактивном никакой генерации не наблюдается, оба транзистора полностью открыты – напряжения на коллекторах 0,18 Вольт (пример во вложении **MULT_TRANS/ Default_mult.DSN**). Все, – Протеус в помойку, разработчики – «кАзлы». Или все-же мы слегка закозлили? Вспомним теорию – режим генерации в таких мультивибраторах возникает из-за несимметричности плеч, обусловленных разбросом параметров реальных компонентов схемы. При этом один из конденсаторов зарядится быстрее и возникнет генерация. Ну и где она у нас разброс параметров у виртуальных моделей? Значит, напрасно мы виним разработчиков. Основным времязадающим элементом здесь является конденсатор. Давайте зарядим один из них. Для предварительной зарядки (вспомним свойства примитива конденсатора) используется свойство **PRECHARGE**. Зададим для **C1** мультивибратора **PRECHARGE=2** (т.е. зарядим хотя бы до 2V, а можно и больше). Опля, заработало!!! (Рис. 37). В той же папке вложения - **Precharge_mult.DSN**.

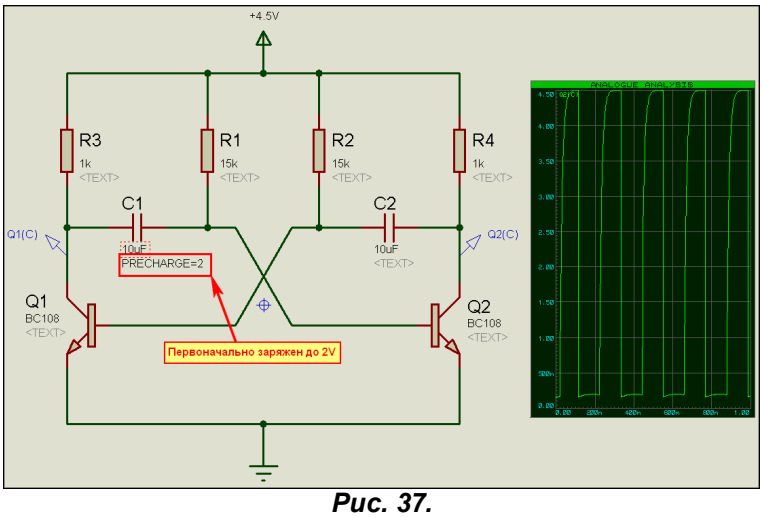


Рис. 37.

Да, «ларчик просто открывался». Но это не единственный способ заставить наш мультик работать, как положено. Вернемся к исходному варианту и уберем у конденсатора первоначальный заряд. Вспомним, что существует еще одно замечательное свойство – **Initial Condition** (первоначальное состояние). По отношению к проводникам оно означает начальное напряжение на проводе при

старте симуляции и назначается в режиме меток **LABEL** из левого меню. Давайте присвоим проводнику от базы транзистора Q1 (он же подключен и к выводу конденсатора C2) метку **IC=0**. Запустим график и … - тоже работает (Рис. 38). Вот и еще один способ обнаружился.

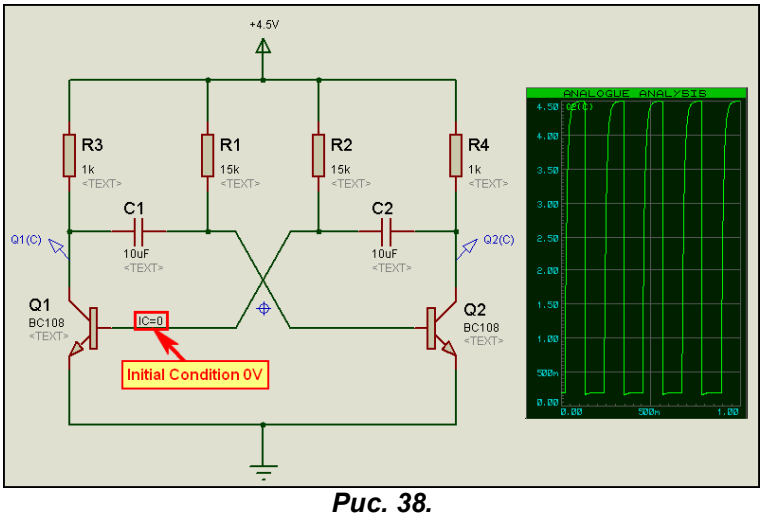


Рис. 38.

Ну а теперь несколько экзотичный, хотя и работающий в некоторых случаях способ. Иногда достаточно просто поменять конденсатор на их анимированную модель, лежащую в библиотеке **Capacitors=>Animated**. Есть у меня большое подозрение, что в этом случае срабатывает стоящее по умолчанию для этой модели в ее **MDF** свойство **PRECHARGE=0**. Правда, этот способ хорош, когда времязадающий конденсатор в единственном числе.

Как мы видим из всех приведенных примеров, в основном устойчивый запуск генераторов связан именно с проблемами конденсаторов, а уж если быть совсем точным, то с начальным условием старта симуляции. Вот этот фактор и надо учитывать при моделировании генераторов в ISIS.

Недавно на форуме всплывал вопрос по генератору трехточке Колпитца, навеянный материалом, расположенным по этой ссылке:

http://logic-bratsk.ru/radio/ewb/ewb2/CHAPTER2/2-8/2-8-1/2-8-1.htm

Автор вопроса поступил именно так, как я описывал вначале, т.е. просто перенес схему от **Electronic Workbench** в **ISIS**. Естественно, результат был плачевный. Но теперь мы «вооружены и очень опасны». Поступим так же, как и в предыдущих случаях (Рис. 39).

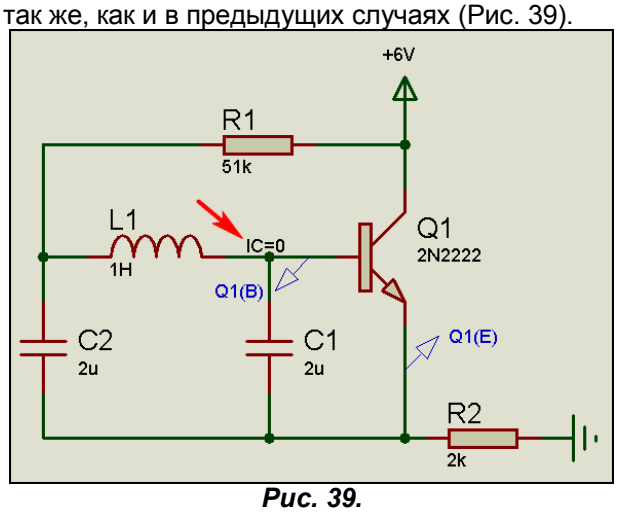


Рис. 39.

Простым добавлением **IC=0** в базовую цепь легко удалось запустить генератор. Более того, в ISIS результат симуляции именно этого варианта оказался более приемлемым. Если у этой схемы в **EWB** согласно приведенной ссылке период оказался равным 7,34 мс при теоретическом — 6,28 мс, то в ISIS период равен 6,63 мс (Рис. 40). Сравните, чей ближе. Этот вариант в приложенном файле **3_POINT\Colpitts1_IC.DSN**. В той же папке вариант с емкостной связью **Colpitts2_Precharge.DSN**, в котором использовано свойство конденсаторов **PRECHARGE**. Чтобы уж окончательно развеять мифы, я взял и грубо, не вдаваясь в расчеты, набросал схему с индуктивной связью – генератор Мейснера (или Майснера). И воспользовавшись свойством **Initial Condition**, в несколько секунд добился его работы. Этот вариант я также приложил в той же папке под именем **Meisner1_IC.DSN**. Думаю, достаточно материала по чисто аналоговым генераторам, чтобы убедиться в их работоспособности. Мы же далее немного остановимся на RC-генераторах на логических элементах, с которыми тоже у начинающих возникают большие проблемы.

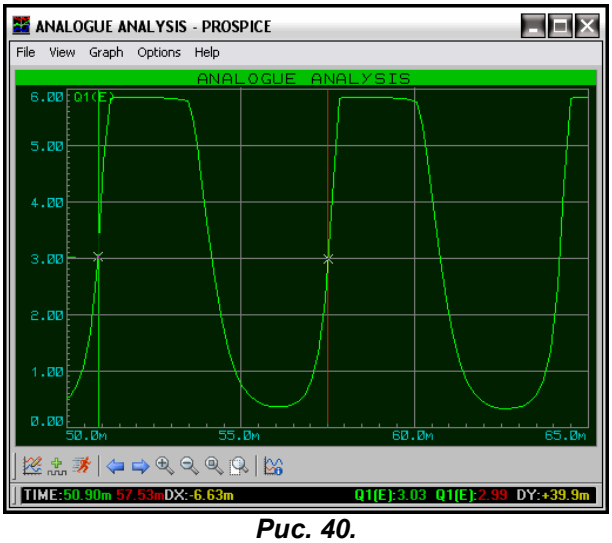


Рис. 40.

Я уже ранее указывал на свойство **SCHMITT**, которое в большинстве случаев позволяет запустить генератор на логических элементах. Но это большинство тоже строго ограничено. Как правило, все пытаются моделировать генератор на элементах 2И-НЕ микросхемы **4011** – наши аналоги ЛА7 в сериях 176, 561, 564 и т.п. Мы уже рассмотрели ее модель выше и обратили внимание, что свойство **SCHMITT** прописано в **MDF**. Быстренько набросаем типовой генератор, пропишем первому логическому элементу по входам это свойство и запустим симуляцию (Рис. 41).

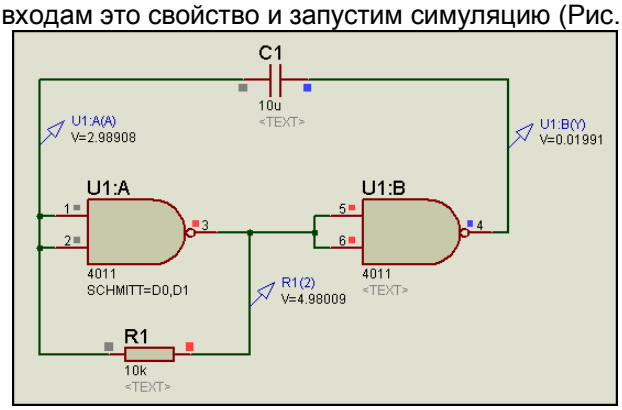


Рис. 41.

Я на рисунке 42 приведу аналоговые графики в характерных точках схемы. Обратите внимание, что на четвертом – **DIGITAL** первый пробник выглядит достаточно «коряво». Именно поэтому в данном случае я и использовал аналоговые графики для анализа этого генератора. Этот вариант во вложении **LOGIC_SCHMITT\ NAND_2.DSN**.

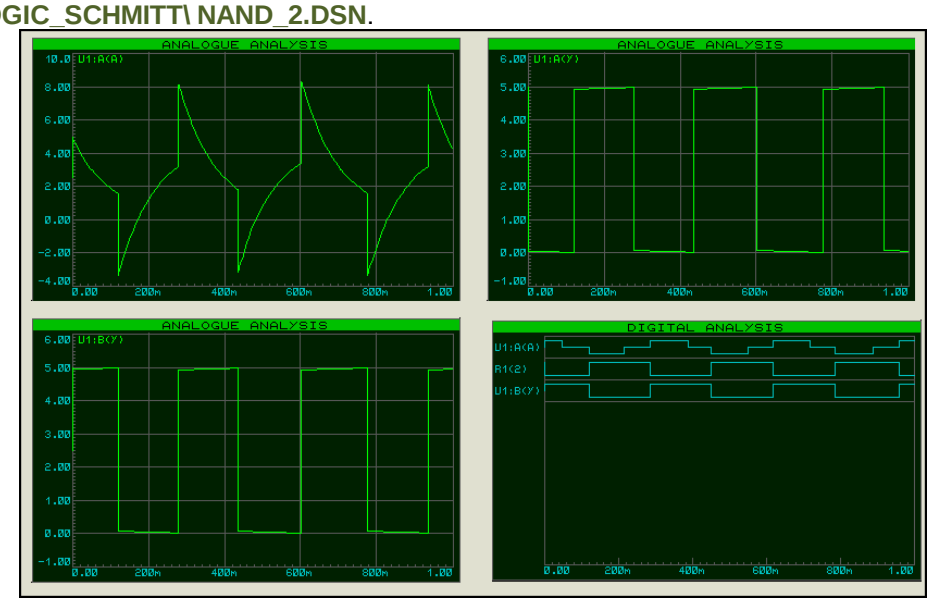


Рис. 42.

В той же папке лежит схема генератора на элементах 3И-НЕ микросхемы **4023**. И так же, как и в предыдущем случае для логического элемента стоящего первым задано **SCHMITT=D0,D1,D2**. Ну, вроде все, как учили, но при попытке запустить симуляцию мы видим унылые серые квадратики вместо веселого перемигивания и строго половинчатые уровни напряжения по всем входам/выходам (Рис. 43). Вроде кому то пора что-то набить - это я имею в виду себя. Но, я и не говорил, что это свойство будет работать всегда.

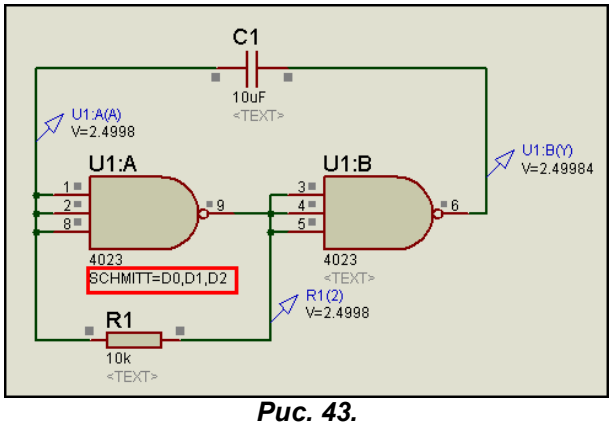


Рис. 43.

А вот теперь открою секрет из «загашника» - почему не работает. Сходите в файл **40NAND3.MDF**, назначенный для **4023** и посмотрите в его начало. Раньше просто посылать вас туда было бесполезно - вы б его и не нашли, но теперь уже пора. Ниже я приведу значимый кусок из этого файла:

```
*MAPPINGS,3,VALUE+VOLTAGE
4073+5V : TDLHDQ=45n, TDHLDQ=55n, TGQ=?
4073+10V : TDLHDQ=20n, TDHLDQ=25n, TGQ=?
4073+15V : TDLHDQ=15n, TDHLDQ=20n, TGQ=?

*MODELDEFS,0

*PARTLIST,1
U?,AND_3,AND_3,PRIMITIVE=DIGITAL,TDHLDQ=<TDHLDQ>,TDLHDQ=<TDLHDQ>,TGQ=<TGQ>
```

Вы где-нибудь видите упоминание **SCHMITT**? Представьте, я тоже не вижу. Ну и что тогда здесь будет работать? Делаем вывод – это свойство будет работать только в тех случаях, если прописано в MDF для данного компонента. А универсальными моделями, содержащими его, являются те, которые предназначены для моделирования триггеров Шмитта. Так что не надо даже лазить по всем MDF, а достаточно заглянуть в библиотеку. Вводим ключевое слово **Schmitt**, выбираем серию, например 4000 и видим, что наряду с рассмотренным в предыдущем параграфе **40NAND2**, которое назначено для 4011 и двухвходового триггера Шмитта 4093, это свойство будет еще в модели инверторов **40INV.MDF** (Рис. 44).

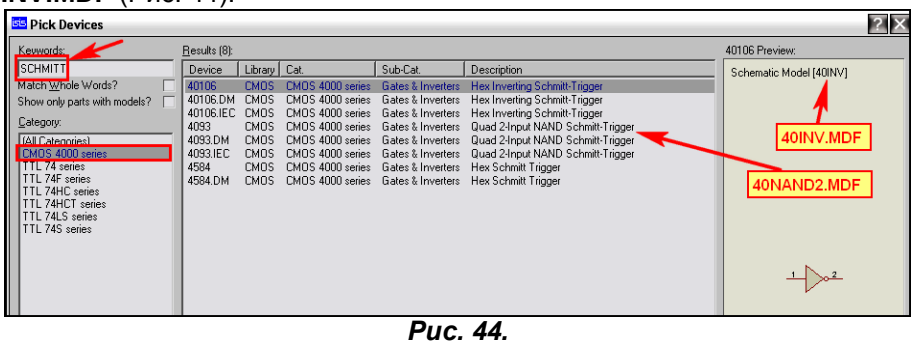


Рис. 44.

Заглянем в этот MDF и увидим, что его можно присвоить следующим инверторам: **4009**, **4049**, **4069** (конечно же, для **40106**, **4093** и **4584** оно уже присвоено). Присваиваться оно будет строчкой **SCHMITT=D**, поскольку у инверторов только один вход, и он не нумеруется. Файл **40INV.MDF** есть во вложении.

Ну а как быть, если мне приспичило использовать, например, элементы 2ИЛИ-НЕ нашей К561ЛЕ5 или ее аналога CD4001 – в Протеусе просто **4001**. Там Шмитт не катит…. Итак, второй «кролик из цилиндра» - я сегодня как факир. Подумайте логически, о чем я тут распинался вначале. Ну а теперь посмотрите на Рис. 45 и загляните в папку **LOGIC_GEN** вложения. Там представлены в двух дизайнах генераторы на двух и трех логических элементах. От комментариев воздержусь, по-моему, и так всем все ясно.

Ну, вот такой небольшой, но весьма познавательный материал по моделированию генераторов в **ISIS**. Надеюсь, что теперь со страниц форума надолго пропадут заголовки типа: «не работает генератор в Протеусе». А если начнут появляться в мое отсутствие, то каждый теперь знает, куда послать страждущего…

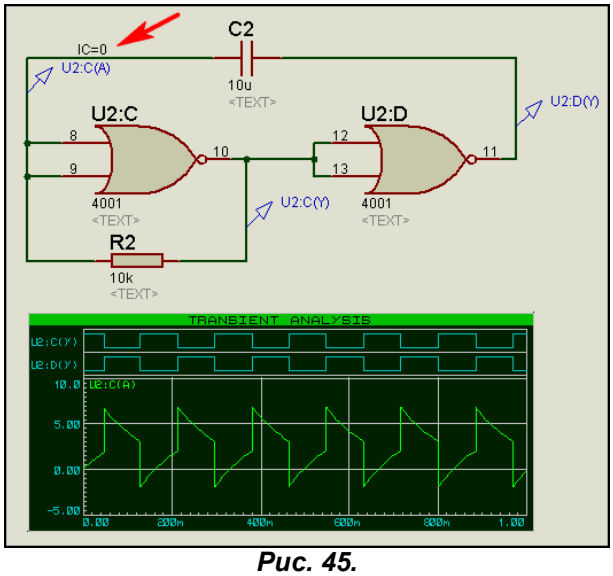


Рис. 45.

****

Материал был уже готов и выложен на форуме, когда в параллельной ветке форума:
http://kazus.ru/forums/showthread.php?t=13612
возник вопрос о генераторах на триггерах в Протеусе. Естественно, я не мог пройти мимо, так как этот вопрос напрямую связан с симуляцией генераторов, рассматриваемой в этом пункте. Чтобы не нарушать нумерацию прилагаемых рисунков, в этом дополнении их приводить не буду. Но в дополнительном вложении **Trigger_Gens** в папке **Examples_Gens** добавлю парочку вариантов генераторов на RS-триггерах, устойчиво работающих в Протеусе.

А заинтересовал меня этот вопрос еще и потому, что при исследовании модели **4042**, на которой сделана попытка просимулировать генератор по приведенной выше ссылке, обнаружена ошибка. Суть ошибки в том, что в реальном триггере **CD4042** происходит асинхронный перенос данных с входов **Dx** на выходы **Qx** при одинаковых уровнях сигнала на управляющих входах **CLK** и **POL**. Т.е. если **CLK=POL=0** или **CLK=POL=1**, то сигнал с входа D должен переноситься на соответствующий ему выход Q асинхронно. Внутренняя структура модели **4042** в ISIS выполнена на D-триггерах и сигнал переносится только по изменению фронтов на управляющих входах, что не соответствует действительности. Я не поленился, и проверил поведение аналогичной модели в **Multisim 11**, там она ведет себя вполне адекватно и генератор с использованием данной модели устойчиво запускается. Пришлось заняться исправлением модели. Что у меня получилось в результате приложено в папке вложения **New_model_4042**. В папке **Structure_Vers_1** приложен тестовый проект одного из четырех каналов микросхемы, созданного на основе даташита. Однако мне этот вариант не очень понравился и другой, более компактный приложен в папке **Structure_Vers_2**. С этим вариантом был протестирован генератор. Результат в проекте из папки **Test_Generator_With_Child**. Там схематичная модель присоединена в качестве дочернего листа к графической модели. С этого листа скомпилирован файл **4042B.MDF**. Тест модели с этим файлом приложен в папке **Test_Generator_With_MDF**. Можно просто скопировать его в папку **MODELS** Протеуса, а над моделью из этого проекта произвести Make Device и сохранить вариант **4042B** для дальнейшего использования. Если есть желание заменить существующую модель **4042**, то можно над ней проделать Make Device и на третьей вкладке поправить ссылку для **MODFILE** на **4042B.MDF**. Для желающих полностью заменить модель в библиотеке **DIGITAL.LML** рекомендую воспользоваться методикой из **п.6.17** этой части FAQ. С этой целью в папке New_4042_MDF приложен MDF-файл, в разделе ***MAPPINGS** литера B на конце имени модели отсутствует, как и в оригинальном файле модели из поставки Протеуса.

****

Возврат к содержанию

6.4. Полезные опыты с цифровым элементом в ISIS. Заключительный материал об IC, NS, PRECHARGE и SCHMITT.

Этот дополнительный материал призван поставить финальную точку в разборке ситуации с цифровыми элементами в **ISIS**. Разбирая в предыдущем параграфе генераторы, я уже привел характерные особенности их запуска в симуляторе. Но, все же, немного поразмыслив, решил более подробно разобрать – чем же характерны цифровые элементы в Протеусе, ну и попутно ознакомить вас с еще одним очень полезным свойством – **NS** (NODESET).

Внимание! Я пошел на поводу у ребят из Лабцентра и допустил серьезную «очепятку» вслед за ними. В **п.6.1** для аналого-цифрового примитива свойство гистерезиса для нижнего порога записывается как **VHL**, а не **VLH**. И в хелпе Протеуса и у меня ранее ошибка. Приношу извинения за себя и Labcenter Electronics и восстанавливаю «статус-кво».

Для начала рассмотрим – как ведет себя типовой цифровой примитив буфера или инвертора при подаче на его вход аналогового сигнала. Для этого подадим на его вход линейно изменяющееся

напряжение от генератора (треугольный сигнал). Для этой цели я воспользуюсь генератором **Pulse** из левого меню **Generator Mode** с несколько «заумными» настройками (Рис. 46).

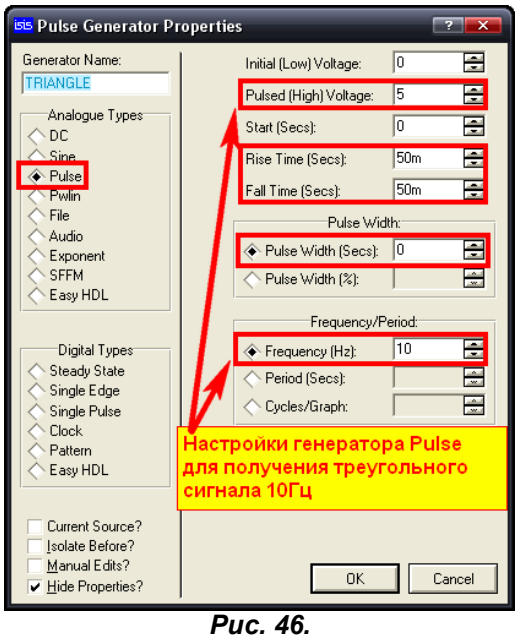


Рис. 46.

Допустим, мы приняли длительность фронта (**Rise**) и спада (**Fall**) импульса равными половине его периода, а ширину верхней полки **Width** равной нулю. Получится, что наш импульс состоит только из линейно повышающегося и понижающегося фронтов, т.е. фактически треугольного сигнала. Если частота равна 1 Гц, то период 1 сек, а фронт и спад по 500мсек. Я взял для своих экспериментов сигнал частотой 10Гц и амплитудой 5 Вольт. Соответственно фронт и спад будут в 10 раз меньше. Теперь подадим этот сигнал на вход цифрового примитива **BUFFER** из библиотеки **Modelling Primitives**. На выход повесим зонд напряжения и проанализируем с помощью аналогового графика выходной сигнал в зависимости от напряжения на входе цифрового буфера. Эта ситуация с выделенными характерными точками представлена на рисунке 47.

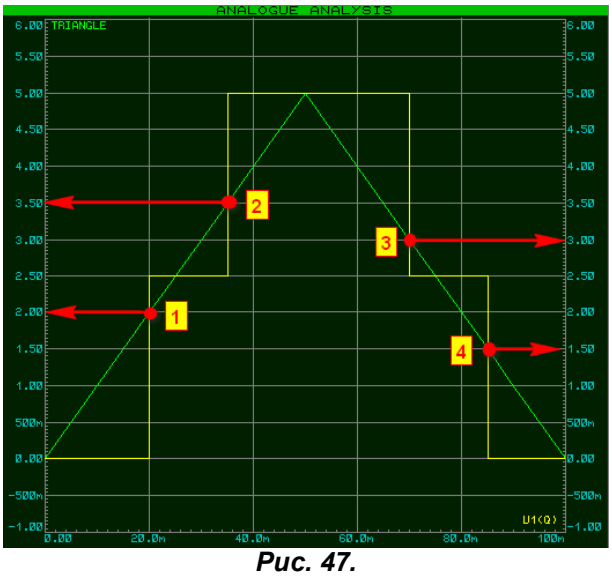


Рис. 47.

Наш цифровой буфер ведет себя как трехпороговый элемент с уровнями 0В, 2,5В (половина питания) и 5В (полное питание). Для того чтобы представить себе откуда взялись эти точки, вспомним приведенные в **п.6.1** свойства аналого-цифрового примитива по умолчанию. Вычислим абсолютные значения:

- Порог переключения на высокий уровень – **VTH=70%** – 5*0,7=**3,5В** (точка 2);
- Порог переключения на низкий уровень – **VTL=30%** – 5*0.3=**1,5В** (точка 4);
- Гистерезисы переключения – **VHH=VHL=10%** – 5*0,1=0,5В. Переход в неопределенную зону (2,5В) происходит от нуля – **VTL+VHL=2.0В** (точка 1) и от единицы – **VTH-VHH=3,0В** (точка 3).

Ну, вот мы и просчитали все наши выделенные точки. И теперь можем точно предсказать – как поведет себя цифровой элемент с этими свойствами по умолчанию при подаче на его вход сигнала определенным напряжением. Для примера я занизил амплитуду треугольника до 3В – ниже порога переключения **VTH**. Результат симуляции графика на рисунке 48, и он вполне предсказуем.

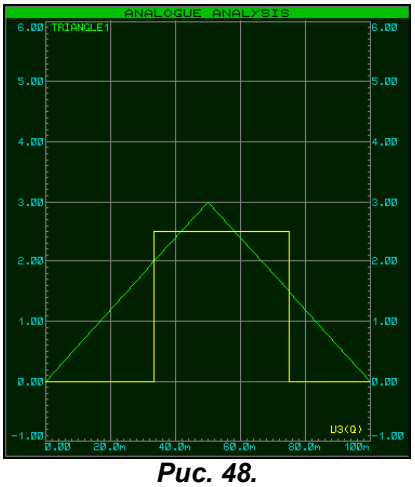


Рис. 48.

Пойдем дальше и рассмотрим – что дает нам добавление в примитиве свойства **SCHMITT**. График для примитива буфера приведен на рисунке 49. Сравнивая с Рис. 46, делаем вывод – исчезли гистерезисы. Остались только две точки переключения – переход на высокий уровень при 3,5В и возврат на низкий при 2,0В. Вот она физическая сущность этого «загадочного свойства». Но стоит еще раз заметить, что воздействует оно только на исходные цифровые примитивы, из которых построена модель реального логического элемента. Вспомните о том, что я писал в предыдущем параграфе про элементы 3И-НЕ микросхемы **4023**. Но и это еще не все сюрпризы, связанные с цифровыми примитивами **ISIS**. Дотошные пользователи решат – да вот мы обнулим гистерезисы и получим тот же **SCHMITT**. И вот тут всплывает один неприятный нюанс из серии «индейской национальной избы», как обозвал ее пес Шарик в мультике про Простоквашино. Попытки изменить аналого-цифровые свойства примитива посредством переназначения **VTH**, **VTL**, **VHH** и **VHL** к желаемому результату не приведут. Можете поэкспериментировать самостоятельно. График будет «стоять как вкопанный». И не важно, в чем вы их будете назначать – в абсолютных значениях или в процентах. Во всяком случае, я лично реального результата не получил. Результаты моих исследований для примитива буфера и инвертора, который, как и положено, ведет себя с точностью до наоборот, приведены в соответствующих проектах **Buffer.DSN** и **Inverter.DSN** вложения. Может кому-то повезет больше.

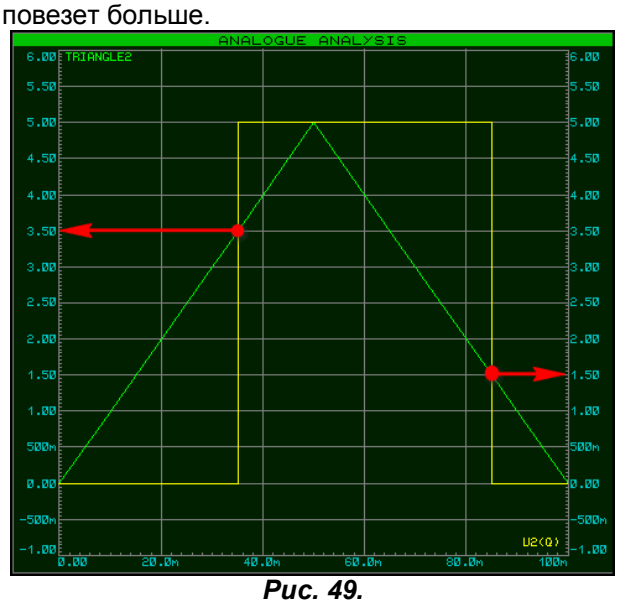


Рис. 49.

Зато картина существенно меняется, если мы задействуем свойство **ITFMOD** и причислим наш примитив к какой либо конкретной серии цифровой логики. На рисунке 50 показан вариант с присвоением примитиву **ITFMOD=CMOS**. Сравните уровни с графиком рисунка 49 – полное совпадение.

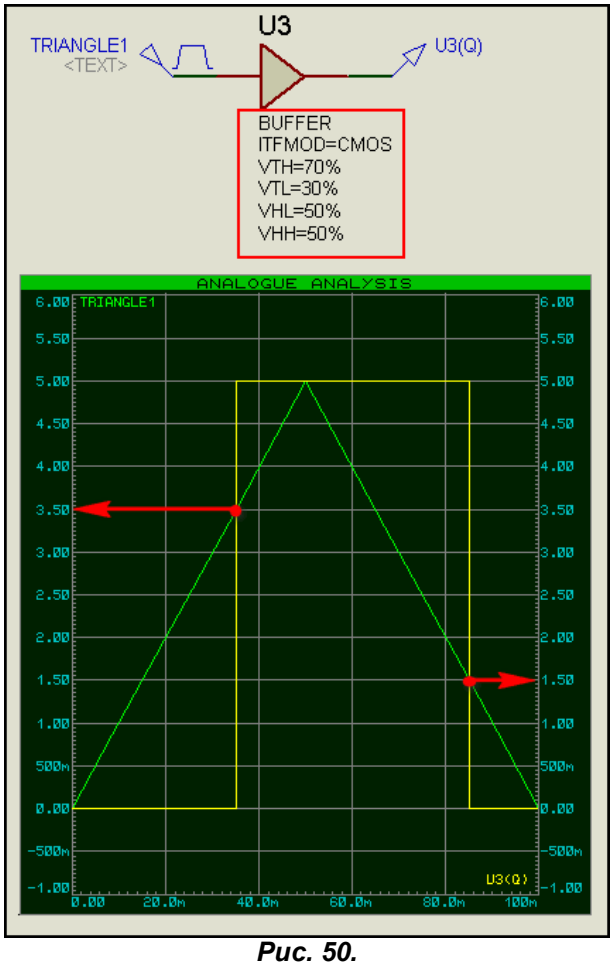


Рис. 50.

Практически мы получили тот же эффект, что и при воздействии свойства **SCHMITT**, только другим путем. В данном примере нам достаточно было задать гистерезис 40%. Простая арифметика: 40% от 5В составит 2В. При этом **VTL+VHL**=1,5+2=3,5В т.е. совпадет с точкой перехода на уровень логической единицы **VTH**. Таким образом, мы убрали ступеньку на переднем фронте импульса. Далее **VTH-VHH**=3,5-2=1,5В – совпадает с точкой перехода на низкий уровень **VTL**. Этот вариант убирает ступеньку на заднем фронте импульса. Когда я задал **VHL=VHH**=50%, что составит 2,5В я сдвинул точки перехода так, что ранее срабатывает **VTH** или **VTL**. При любом значении одного из гистерезисов менее 40% будет появляться соответствующая ступенька с уровнем 2,5В как на графике рисунка 47. Варьируя этими четырьмя значениями для логического элемента в проекте, мы можем добиваться нужных нам результатов. Следует помнить, что единицы измерения для них должны совпадать – или все в Вольтах или все в процентах. Последнее принято по умолчанию. Поэтому если вы поставите какой-либо элемент 4000 серии в схему и зададите ему в свойствах только **VTL** и **VTH**, но в абсолютных единицах, например **VTL=1.5** и **VTH=3.5**, то гистерезисы, которые остались по умолчанию в процентах работать не будут и эффект будет тот же, что и на Рисунке 50. Различные варианты с графиками для CMOS приведены в **Buffer_CMOS.DSN**. Все вышесказанное касается и других серий цифровых микросхем, только и уровни там будут несколько другие в соответствии с заданными в **ITFMOD.MDF**. Пример для TTL во вложении **Buffer_TTL.DSN**.

Теперь вернемся на секунду к нашему генератору на инверторах и с точки зрения новых познаний рассмотрим, почему он не запускается без лишних телодвижений с нашей стороны. Я еще раз повторил картинку генератора на двух элементах 2И-НЕ с установленными зондами на входах-выходах элементов (Рис. 51).

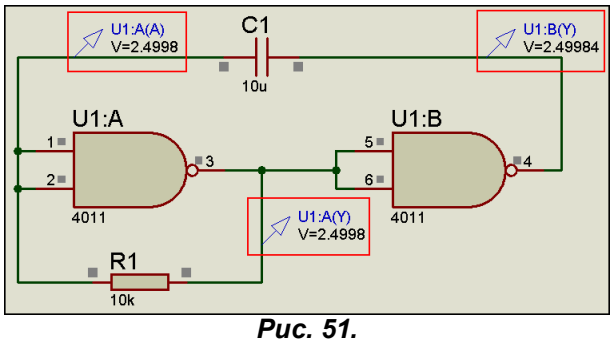


Рис. 51.

Обратите внимание, что при запуске симуляции напряжение во всех контрольных точках практически одинаковое и равно половине питания или тому пресловутому порогу 2,5В на предыдущих графиках. Значит, для того чтобы его сдвинуть хотя бы в одной из точек и нужны дополнительные меры. Это можно сделать даже кнопкой **BUTTON**, посадив кратковременно на землю, например, входы первого логического элемента при запущенной симуляции. Но мы уже знаем, что есть такое интересное свойство **IC** – **Initial Condition** для цепей, которым мы воспользовались в предыдущем параграфе. Однако и здесь тоже кроется несколько подводных камней. Главная особенность состоит в том, что данное свойство можно присваивать только аналоговым цепям. Если Вы попытаетесь назначить **IC** цепи, не содержащей аналоговых компонентов – Протеус незамедлительно при старте симуляции вывесит красное сообщение об ошибке. Для цепей, которые соединяют только цифровые компоненты, аналогичное свойство носит название **BS** (**Boot State**) – состояние при загрузке. Свойству **BS** можно назначать состояние в любом виде, которое поддерживает Протеус для цифровых цепей. Это: **1, 0, H, L, HIGH, LOW, SHI, WHI, SLO, WLO** или **FLT**. Второй особенностью **IC** является то, что это состояние принимается для цепи до начала первой итерации, т.е. до начала расчета симулятором **ProSPICE** операционных точек схемы. Для задания начальных условий по постоянному току при нулевой итерации служит другая директива **NS** (NODESET) – установка узла. Она назначается, как и **IC** через лэйбл, присваиваемый нужному проводу. Например, **NS=10** означает, что на начальной стадии расчета точки в данном узле цепи устанавливается значение 10 Вольт. Это свойство наиболее полезно, когда **ProSPICE** не может точно рассчитать потенциал узла на начальной стадии и является как бы подсказкой симулятору – каким принять значение потенциала в первый момент расчета. Оно так же полезно, когда симулируется достаточно сложная схема, поскольку позволяет симулятору сократить время на расчет операционных точек.

Обе директивы **IC** и **NODESET** являются стандартными для всех программ, базирующихся на ядре **SPICE**. Это и OrCAD, и Multisim и т.д. Открыв любую книжку, посвященную описанию этих программ, вы сможете найти эти директивы в главах, посвященных заданию начальных условий моделирования. В тоже время свойство **PRECHARGE** для конденсаторов, которое мы тоже использовали для запуска генераторов, является фирменной добавкой Лабцентра к SPICE-симулятору. Ну, и чтобы окончательно поставить точку в этом вопросе рассмотрю еще один пример применения данных свойств. Очень часто разработчики используют для сброса цифровых счетчиков, регистров, да и микроконтроллеров интегрирующую RC-цепочку. Не будьте так уверены в ее непогрешимой работе, просто прилепив ее к соответствующей цепи. Взгляните на рисунок 52. На левом графике RC-цепочка стоит с параметрами по умолчанию, на среднем – с применением **Initial Condition**, а на правом – **PRECHARGE**. Надеюсь, для вас не составит большого труда сделать вывод, – в каком случае цепочка обеспечит вам сброс, а в каком нет. Данный пример во вложении носит имя **RC_Reset.DSN**.

Ну и в заключение данного материала сошлюсь на раздел HELP Протеуса, где описан материал, посвященный начальным условиям. ProSPICE Help => ADVANCED TOPICS => INITIAL CONDITIONS. Желающие прочитать в оригинале, могут заглянуть туда.

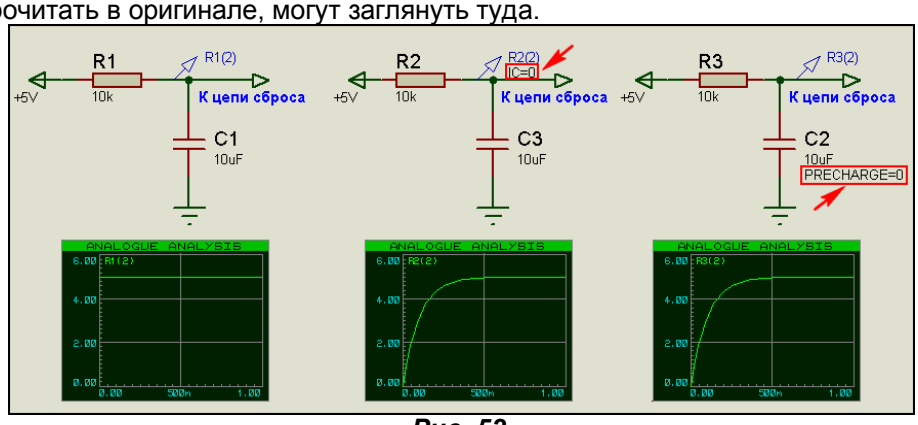


Рис. 52.

Возврат к содержанию

6.5. «Я его слепила из того, что было…». Анатомия CD4060 – прообраза будущей К176ИЕ12.

Когда я достаточно давно задумал сделать модель К176ИЕ12, то естественно подумал – а что можно взять за основу для будущей модели. Основательно перерыв все библиотеки тогдашней версии Протеуса, да и в нынешней там мало что изменилось, я остановил свое внимание на модели 4060. Микросхема представляет собой 14-разрядный счетчик со встроенной схемой генератора. Частота генератора может задаваться как RC цепочкой, так и кварцевым резонатором. Почему-то у нас в России эта микросхема не пользуется особой популярностью. Подозреваю, все дело в том, что она не имеет отечественного аналога. Ближайшее, что когда-то предлагалось в журнале «Радио» №6 за 2006 год, - это аналог на двух м/сх: К561ИЕ16 и К561ЛА7. Да и публикаций всевозможных устройств на этой микросхеме не так уж много. Правда в свое время «Мастер-Килька» отозвался выпуском набора NF239 для сборки таймера: http://www.masterkit.ru/main/set.php?code_id=129679, но и тот уже на данный момент снят с продаж. Больше всего публикаций самоделок на данной микросхеме встречается в журнале «Радиоконструктор», авторы которого до сих пор нет-нет, да и опубликуют что-то свежее с

использованием CD4060. В тоже время, данная микросхема до сих пор выпускается многими зарубежными фирмами под несколько различающимися названиями. Это и **CD4060BC** у Fairchild и **MC14060B** у On Semiconductor и **HEF4060B** у Philips (ныне NXP). Наиболее подробные даташиты на эти микросхемы у двух последних из перечисленных. Я и сам, грешен, пару лет назад использовал ее для создания 12-ти часового таймера заряда резервного аккумулятора в одном девайсе, который благополучно пашет до сих пор на одном из подведомственных объектов. Для тех, кто заинтересуется применением микросхемы всерьез, но не дружит с английским, рекомендую ознакомиться со статьей «Анатомия таймера на ИС CD4060 из набора “МАСТЕР_КИТ» в №3 за 2006 год журнала «Радиосхема» http://publ.lib.ru/ARCHIVES/R/"Radioshema"/_"Radioshema".html .

Ну а мы бегло познакомимся с особенностями ее модели в Протеусе, чтобы иметь подходящую материальную базу для создания К176ИЕ12. Если заглянуть в свойства модели 4060, то помимо типовых для всей этой серии мы обнаружим параметр **Oscillator Frequency** (Частота генератора) со значением по умолчанию **Default** (**Рис. 53**).

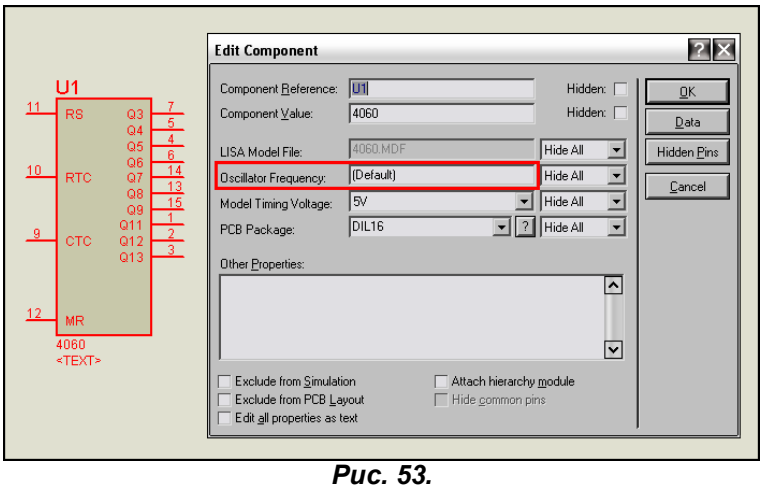


Рис. 53.

На третьей вкладке **Make Device** обнаруживается, что **Default Value** для свойства **CLOCK**, каковым и является частота генератора равно **None**, т.е. генератор отключен, а Limits для него установлен Positive or Zero (положительное или ноль). Для того чтобы привести микросхему в чувство введем вместо Default (**Рис. 53**) числовое значение, например **10** (Гц), а также завесим на землю вход сброса счетчика **MR**. Запускаем симуляцию, и наш счетчик начинает благополучно моргать выходами и входами времязадающих цепей **RS**, **RTC**, **CTC**. Прицепим к ним осциллограф, а заодно и частотомер и убедимся, что там прямоугольные импульсы с заданной нами частотой 10 Гц, т.е. работает встроенный в модель генератор (**Рис 54**). Этот пример во вложении **Internal_Osc.DSN**.

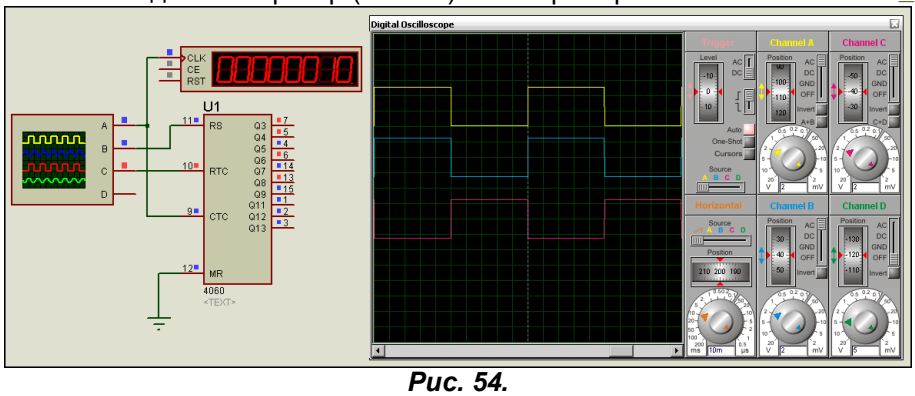


Рис. 54.

Для того, чтобы понять какое еще кроме цифровых значение можно вписать в **Oscillator Frequency**, нам придется добыть из библиотеки **DIGITAL.LML** и исследовать файл модели **4060.MDF**. Я приложил его для тех, кому лень воспользоваться утилитой **GETMDF.EXE**. Ниже приведены первые разделы файла.

```
*PROPERTIES,1
CLOCK=EXTERNAL

*MAPPINGS,3,VALUE+VOLTAGE
4060+5V : TDOSC=35n, TDCQ=25n, TDMRQ=100n
4060+10V : TDOSC=13n, TDCQ=10n, TDMRQ=40n
4060+15V : TDOSC=8.5n, TDCQ=6n, TDMRQ=30n

*MAPPINGS,2,CLOCK
DEFAULT : CLKOSC=DIGITAL,CLKGATE=NULL
EXTERNAL : CLKOSC=NULL, CLKGATE=DIGITAL

*MODELDEFS,0
```

Внимательно посмотрев на разделы ***PROPERTIES,1** и ***MAPPINGS,2,CLOCK** можно сделать вывод, что микросхема должна адекватно реагировать на значение **EXTERNAL** в окне **Oscillator Frequency**. Попробуем этот вариант. Заменим значение **10,** словом **EXTERNAL** и вновь запустим симуляцию. Микросхема тихо стоит, но симулятор не ругается и ошибок не выдает, значит, наше предположение верное. Далее придется обратиться к даташиту на сей девайс, чтобы посмотреть – как подключается времязадающая RC цепочка. Можно просто в режиме подключения к Интернету щелкнуть в свойствах кнопку **Data** (или по меню правой кнопки **Display Datasheet**) и скачать файрчилдовский даташит. Я же приведу на **Рис. 55** вариант от **MC14060B**, поскольку в нем наиболее крупно расписана формула для частотозадающей цепи. Небольшое пояснение – левый по схеме вывод **Rs** в других даташитах соединен с выводом **11** микросхемы, который здесь обозначен как **CLOCK**.

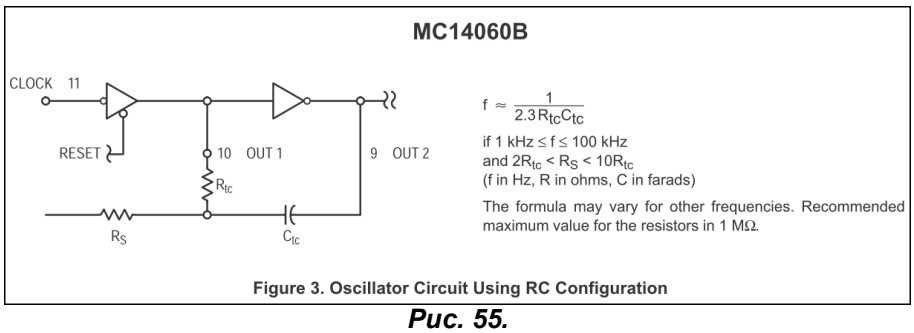


Figure 3. Oscillator Circuit Using RC Configuration

Рис. 55.

В соответствии с формулой рассчитаем для той же частоты 10 Гц сопротивление резистора **Rtc** для емкости **Ctc** в 1мкФ. Получим 43,2кОм – берем ближайшее 43к. В соответствии с упоминавшейся статьей **Rs** выбирается в пределах **2…10Rtc**. Я поставил 200кОм. Ну и чтобы закрепить материал предыдущего параграфа я приложил три варианта: **External_Animal_C.DSN** (на **Рис. 56**) – с анимированным конденсатором; **External_Precharge_C.DSN** – используется **PRECHARGE** и **External_IC_wire.DSN** – используется **Initial Condition**. Открыв соответствующие проекты во вложении и, запустив симуляцию, вы можете лично убедиться, что все они имеют право на существование. Попадание в частоту оказалось не очень, но все же удачным. Вместо расчетных 10 Гц имеем 15, но для RC генератора, да еще в программе, а не в железе – это достаточно неплохо. Желающие могут подобрать резистор, увеличив сопротивление для получения точных 10 Гц. У меня получилось при R1=68кОм – этот вариант оставлен в **External_IC_wire.DSN**.

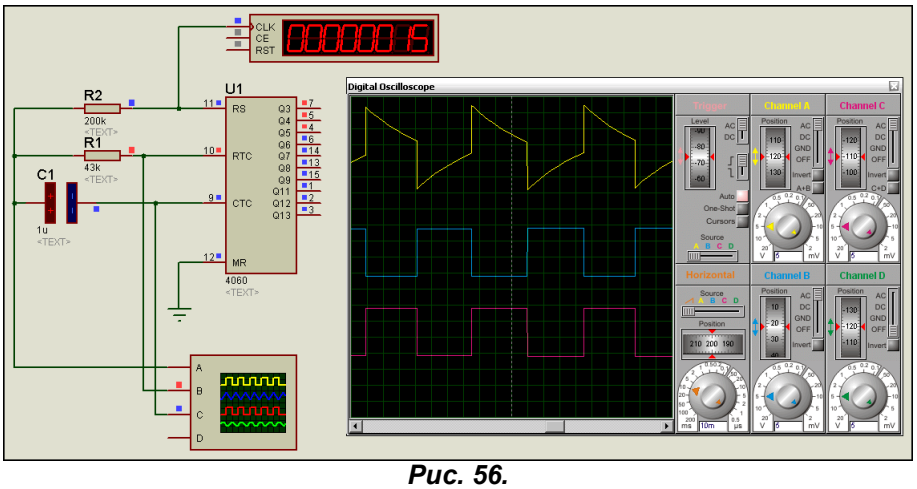


Рис. 56.

Теперь приведу еще одну картинку из даташита с очень важным в дальнейшем пояснением. На **Рис. 57** приведена упрощенная схема структуры **HEF4060B**. Я опять воспользовался тем, что в этом даташите нужный мне элемент уже заранее прорисован покрупнее. Обратите внимание, что в том месте, где подключается конденсатор **CTC** – вывод 11 микросхемы, используется триггер Шмитта! Это важно для нас в дальнейшем. Мы попытаемся пойти дальше разработчиков модели 4060 и воспользоваться этим моментом.

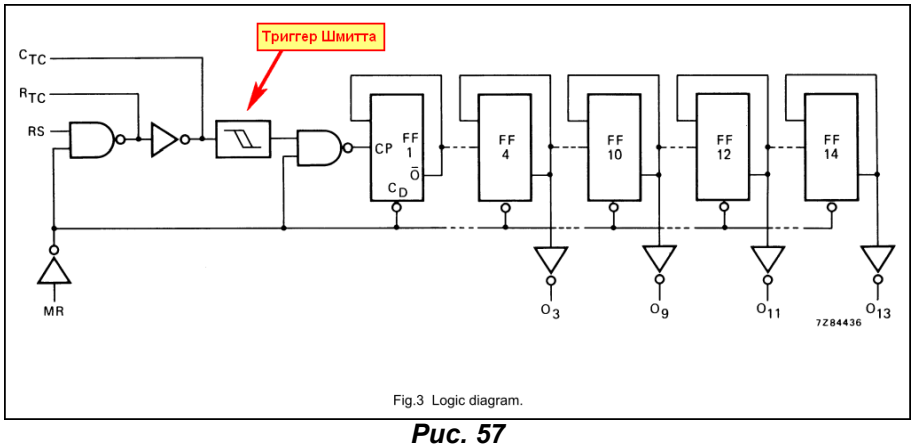


Fig.3 Logic diagram.

Рис. 57

Налюбовавшись вдоволь на структуру, представленную на **Рис. 57** вновь обратимся к файлу **4060.MDF**. Теперь нас интересует **PARTLIST**, приведенный ниже:

```
*PARTLIST,19
OSC,CLOCK,,CLOCK=<CLOCK>,INIT=0,PRIMITIVE=<CLKOSC>,PRIMTYPE=DIGITAL!
U1,INVERTER,INVERTER,PRIMITIVE=DIGITAL
U2,NAND_2,NAND_2,PRIMITIVE=DIGITAL,TDHLDQ=<TDOSC>,TDLHDQ=<TDOSC>
U3,INVERTER,INVERTER,PRIMITIVE=DIGITAL,TDHLDQ=<TDOSC>,TDLHDQ=<TDOSC>
U4,NAND_2,NAND_2,PRIMITIVE=DIGITAL,TDHLDQ=<TDOSC>,TDLHDQ=<TDOSC>
U5,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U6,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U7,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U8,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U9,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U10,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U11,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U12,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U13,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U14,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U15,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U16,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U17,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U18,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
```

Мы можем констатировать, что помимо внутреннего генератора **OSC** и элементов входной логики **U1…U4**, модель содержит 14 счетных DTFF триггеров – элементы **U5…U18**. Можно сделать вывод, что разработчик модели полностью воспроизвел внутреннюю структуру, т.е. имеет место полное схематичное моделирование. В случаях с цифровыми моделями это вполне приемлемо, поскольку симуляция в отличие от аналоговых проходит намного быстрее и не «съедает» ресурсы компьютера. На временных параметрах, заданных примитивам останавливаться не буду, поскольку мы их назначение достаточно подробно разбирали раньше. Также, не стану подробно разбирать и **NETLIST**, а просто приведу кусочек восстановленной структуры модели **4060** на **Рис. 58.** Полностью структуру рисовать не имеет смысла, т.к. соединения всех 14 счетных триггеров одинаковы и повторяют друг друга.

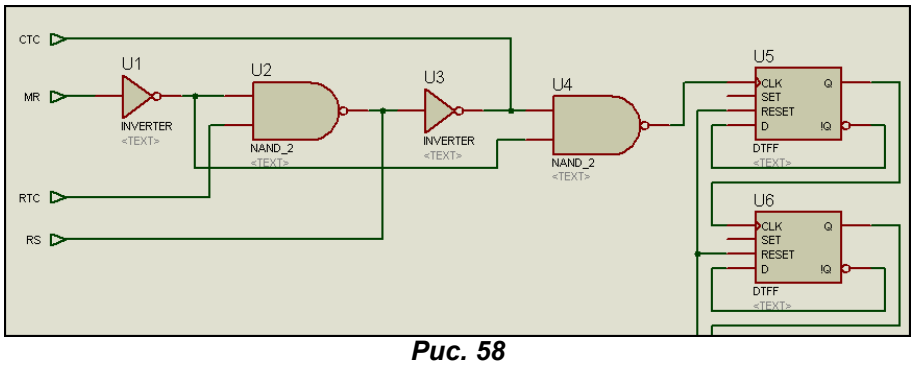


Рис. 58

Тем, кто с элементами цифровой логики знаком только понаслышке, и собирается дальше осваивать представленный здесь материал, могу порекомендовать найти и ознакомиться с одной из нижеперечисленных книг:

Угрюмов Е.П. «Цифровая схемотехника», БХВ-Петербург, - *есть два издания 2000 и 2004 года, годится любое, мне лично больше нравится старое.*

Уилкинсон Барри «Основы проектирования цифровых схем», М., ИД «Вильямс», 2004.

Точчи Рональд Дж,, Уидмер Нил С., «Цифровые системы. Теория и практика» , М., ИД «Вильямс», 2004.

Годится и любой другой учебник по основам цифровой схемотехники. Желательно иметь понятия о работе JK и D триггеров, последовательных и параллельных счетчиков, в т.ч. Джонсона и сдвиговых регистров. Само собой разумеется, что работа логических элементов И, ИЛИ, исключающее ИЛИ и

их комбинаций тоже не должна ставить вас в тупик. В последующем материале я буду свободно оперировать этими понятиями, а разжевывать работу этих элементов я не ставлю себе в задачи. Моя забота – научить вас созданию моделей в Протеусе, но никак не основам цифровой техники, иначе мы увязнем в этом материале на год, а нас ждет еще масса полезных сюрпризов из приемов работы с моделями в этой программе.

Возврат к содержанию

6.6. Пример создания полной схематичной модели счетчика К176ИЕ12. Часть 1 – подготовительные работы.

Те, кто пользовался ранними версиями FAQ по Протеусу, могут благополучно пропустить несколько последующих параграфов, потому что они мало чем будут отличаться от аналогичных старого варианта FAQ.

Итак, ищем исходные данные по микросхеме **К176ИЕ12**. Если есть старые справочники, можно воспользоваться ими. Я же приведу две ссылки в сети, где можно найти информацию по этой микросхеме:

http://lib.qrz.ru/?q=node/5376
http://lib.qrz.ru/?q=node/5480

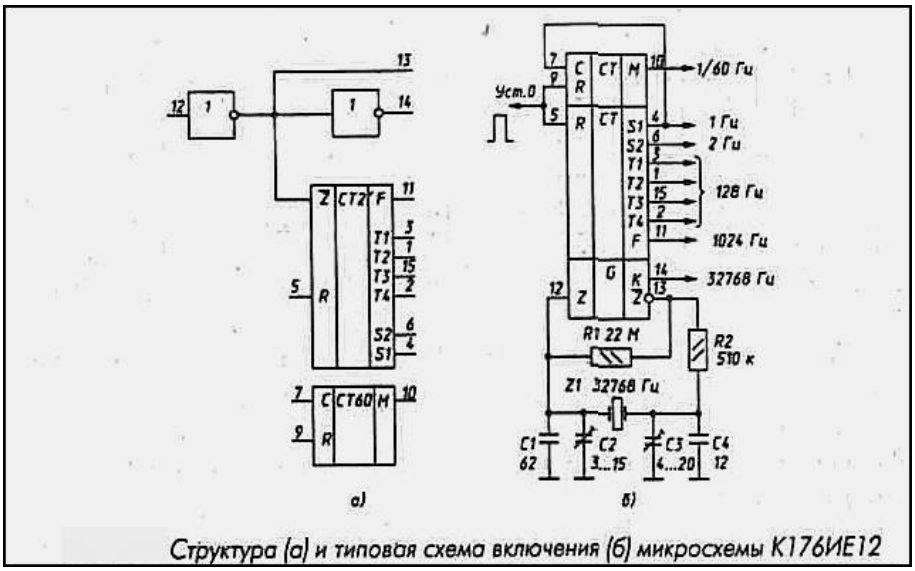
По первой ссылке есть описание ее работы и диаграммы, а по второй есть вариант генератора на **К176ИЕ12** с использованием задающей RC цепочки. Вообще, вся эта информация взята из книги: **Бирюков С.А. «Применение цифровых микросхем серий ТТЛ и КМОП»**, М.; ДМК, 2000.

Любители порыться в архивах могут воспользоваться циклом статей С. Алексеева **«Применение микросхем серии К176»**. Журналы «Радио» №№4…6 за 1984 г.

Давайте кое-что из данной информации перенесем сюда, чтобы было удобнее ориентироваться. Я преднамеренно изменил номера рисунков в тексте, взятом по верхней ссылке, чтобы они совпадали с принятой у меня нумерацией. Остальное оставлено как есть.

Микросхема К176ИЕ12 предназначена для использования в электронных часах **(рис. 59)**. В ее состав входят кварцевый генератор G с внешним кварцевым резонатором на частоту 32768 Гц и два делителя частоты: СТ2 на 32768 и СТ60 на 60. При подключении к микросхеме кварцевого резонатора по схеме **рис. 59** (б) она обеспечивает получение частот 32768, 1024, 128, 2, 1, 1/60 Гц. Импульсы с частотой 128 Гц формируются на выходах микросхемы Т1 - Т4, их скважность равна 4, сдвинуты они между собой на четверть периода. Эти импульсы предназначены для коммутации знакомест индикатора часов при динамической индикации. Импульсы с частотой 1/60 Гц подаются на счетчик минут, импульсы с частотой 1 Гц могут использоваться для подачи на счетчик секунд и для обеспечения мигания разделительной точки, для установки показаний часов могут использоваться импульсы с частотой 2 Гц. Частота 1024 Гц предназначена для звукового сигнала будильника и для опроса разрядов счетчиков при динамической индикации, выход частоты 32768 Гц - контрольный. Фазовые соотношения колебаний различных частот относительно момента снятия сигнала сброса продемонстрированы на **рис. 60**, временные масштабы различных диаграмм на этом рисунке различны. При использовании импульсов с выходов Т1 - Т4 для других целей следует обратить внимание на наличие коротких ложных импульсов на этих выходах.

Особенностью микросхемы является то, что первый спад на выходе минутных импульсов М появляется спустя 59 с после снятия сигнала установки 0 с входа R. Это заставляет при пуске часов отпускать кнопку, формирующую сигнал установки 0, спустя одну секунду после шестого сигнала поверки времени. Фронты и спады сигналов на выходе М синхронны со спадами импульсов отрицательной полярности на входе С.



Структура (а) и типовая схема включения (б) микросхемы К176ИЕ12

Рис. 59.

Подытожим найденную информацию. Если проанализировать структуру и диаграмму, то мы видим, что необходимы следующие входы/выходы модели:

F – с частотой 1024 Гц (11-й вывод – звук на будильник);

T1, **T2**, **T3**, **T4** – четырехтактный генератор 128 Гц для динамической индикации (выводы 3, 1, 15, 2);

S1 и **S2** – выходы секундных и полусекундных импульсов (выводы 4 и 6 соответственно);

C – вход счетчика-формирователя минутного импульса (7-й вывод).;

M – достаточно хитрый сигнал минутного импульса (10-й вывод);

K – выход с тактовой частотой 32768 Гц (14-й вывод) на диаграмме почему-то отсутствует);

Z и **Z**инв – для подключения кварцевого резонатора (выводы 12 и 13);

R – цепи сброса счетчика-делителя на 32768 (вывод 5) и счетчика-формирователя минутного импульса (вывод 9). Они разнесены и, в отличие от **CD4060**, не блокируют входной тактовый генератор на двух инверторах.

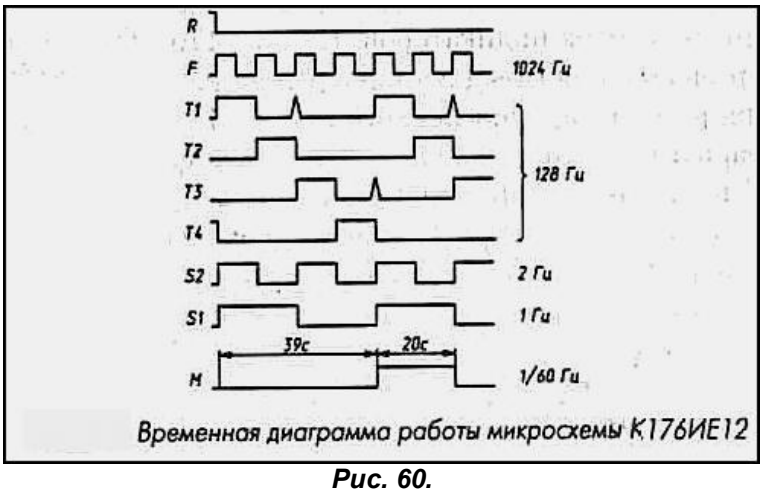


Рис. 60.

Еще одно отличие от **CD4060** в счетчике-делителе тактовой частоты. Для того чтобы поделить частоту 32768Гц до 1Гц нам потребуется на один разряд (счетный триггер) больше. Вы можете убедиться в этом, открыв приложенный в папке **CD4060_PLUS** проект **15_counter_gen.DSN**. Там для CD4060 задана нужная нам исходная частота и с помощью дополнительного к 14 разрядам микросхемы триггера мы получаем нужный нам сигнал 1 Гц. Обратите также внимание, что необходимые нам в соответствии с исходными данными сигналы 1024Гц и 2 Гц будут находиться соответственно на выходах 4-го и 14-го разрядов счетчика.

На основании этой информации и, руководствуясь типовой схемой включения, строим нашу графическую модель (файл вложения **Graphic_model.DSN** в одноименной папке). Здесь необходимо отметить, что входы сброса мне пришлось дополнительно пометить цифрами, чтобы не сбивать симулятор одноименными выводами модели. Вид получившейся графической модели представлен на Рис. 61. Тут есть два нюанса. Чтобы инверсный выход **Z** отображался с верхним надчеркиванием необходимо в имя вывода в его свойствах с двух сторон ограничить знаком доллара, т.е. в окне **Name** для вывода 13 должно быть так: **Z**. Кроме того, в модели присутствуют два скрытых вывода питания – **VDD** (сверху) и **VSS** (снизу).

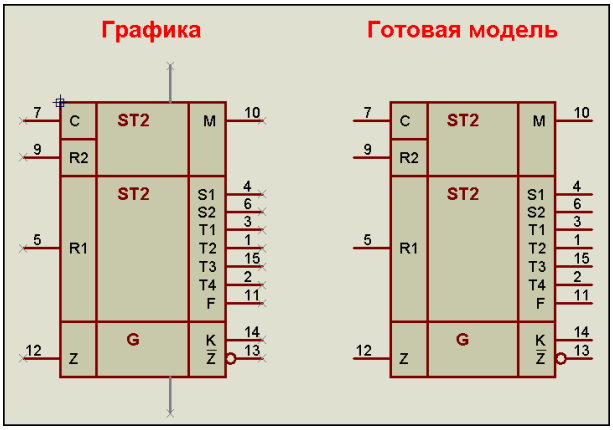


Рис. 61.

Обратите внимание, что выводы питания присутствуют только в графике, а после создания модели они исчезли (скрытые). На Рис. 62 на примере **VDD** показано – что необходимо установить в свойствах вывода перед созданием модели, чтобы он стал скрытым выводом питания.

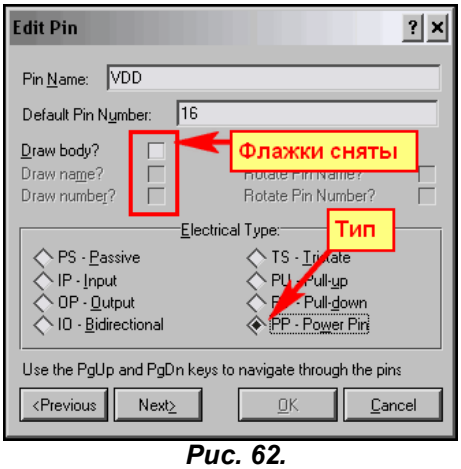


Рис. 62.

Но пока мы ничего с нашей графической моделью больше делать не будем. Отложим ее до лучших времен, а все наши промежуточные тестирования будем проводить с помощью **Subcircuit**, т.е. модулей. Почему? Да потому, что мне уже надоело то и дело писать, а вам – читать: **Make Device**. И крутимся мы без конца по этим пяти окнам вкладкам…. С модулями такие операции проводить не надо, но в тоже время они позволяют вести отладку в пределах одного проекта так же, как и в случае создания схематичной модели. Прогуляемся по второй из ссылок выше и найдем там схему RC-генератора на микросхеме **К176ИЕ12** (Рис. 63). Я ее поместил сюда, поскольку она нам тоже понадобится.

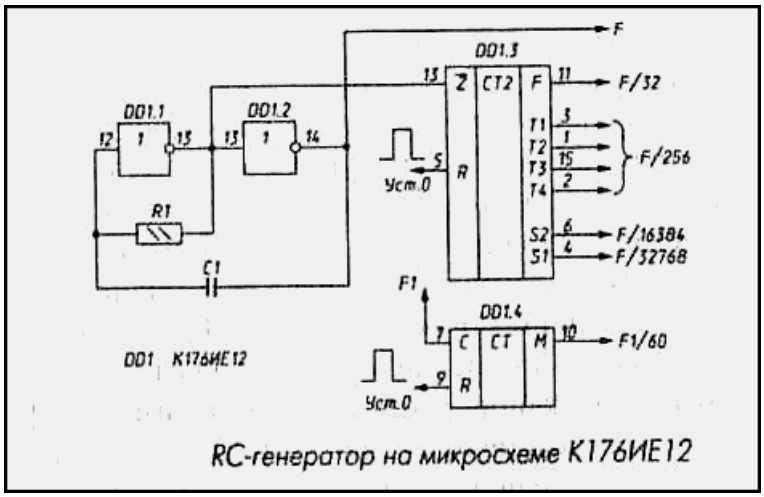


Рис. 63.

Для начала нам предстоит разобраться со встроенным генератором, ну и попутно приклеим к нему часть делителя на D-триггерах, чтобы проверить, как это будет работать в модели.

Возврат к содержанию

6.7. Пример создания полной схематичной модели счетчика К176ИЕ12. Часть 2 – встроенный генератор и делитель тактовой частоты.

Итак, первоначально воспроизведем входную цепь микросхемы, допустим первые четыре триггера делителя тактовой частоты. Я сначала делаю это обычно в отдельном дизайне, на главном листе, чтобы не париться с «прыжками» по листам туда-сюда. Нам потребуются цифровые примитивы **INVERTER** и **DTFF** из библиотеки **Modelling Primitives**. Кроме того, сразу же можно отыскать и втащить в проект цифровой генератор **CLOCK** из библиотеки **Simulator Primitives**. Мы его используем для тестирования, а в дальнейшем в модели для внутреннего генератора по аналогии с **CD4060**. Первый вариант для тестирования представлен на Рис. 64.

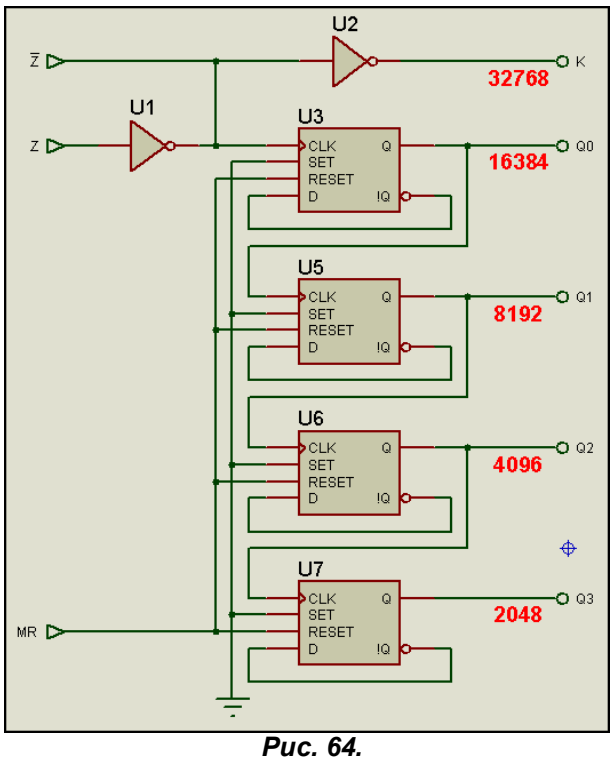


Рис. 64.

Я преднамеренно поставил у элементов в свойствах флажок **Hidden** напротив **Component Value**, воспользовавшись **Property Assignment Tools** (**PAT**). Для этого в окне **String** PAT набираем **VALUE**, в переключателе **Action** устанавливаем **Hide**, а ниже выбираем **On Click** и пробегаем щелчками левой кнопки по всем установленным компонентам. Кто еще не выучил PAT как «Отче наш…» - привыкайте. Сейчас у нас на схеме всего 7 элементов, а будет в несколько раз больше. И задавать каждому свойства вручную – каторжный труд. Еще я отключил опцию **Show Hidden Text** в меню **Temlate => Set Design Default**. Это позволило мне сэкономить пространство проекта, поскольку картинки становятся уже достаточно объемными и лишняя информация на них нам ни к чему. Итак, из литературы по ссылкам в предыдущем параграфе, а кто и по памяти, мы знаем, что каждый из D-триггеров в таком включении поделит тактовую частоту на 2. Т.е., подав на вход 32768Гц, мы получим сетку частот, выделенную красным на Рис. 64. Проверяем это в приложенном проекте **GEN_IE12_st_1.DSN**, а в проекте **GEN_IE12_st_2.DSN** проверяем функционирование нашего генератора в соответствии с Рис. 63 при внешней задающей RC-цепи. Оба проекта в папке **TEST1**. Во втором случае я воспользовался **IC=0** на инверсной цепи **Z** для запуска генератора. Теперь, когда мы убедились в успешной работе тестовых проектов, создаем модуль, а количество разрядов делителя на дочернем листе увеличиваем до требуемых пятнадцати. Как выглядит наш модуль на основном листе в действии показано на рисунке 65.

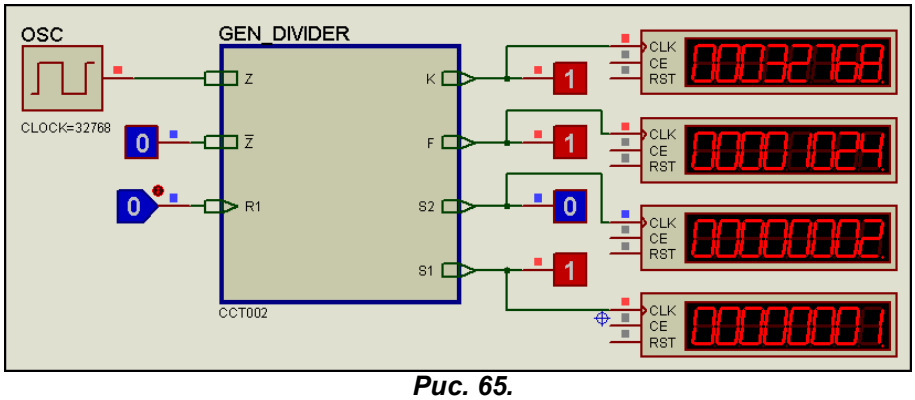


Рис. 65.

Пока наш примитив генератора **OSC** находится на основном листе. Мы получили требуемую сетку частот в соответствии с описанием микросхемы. На рисунке 66 приведены три цифровых графика работы модуля с различными временными интервалами.

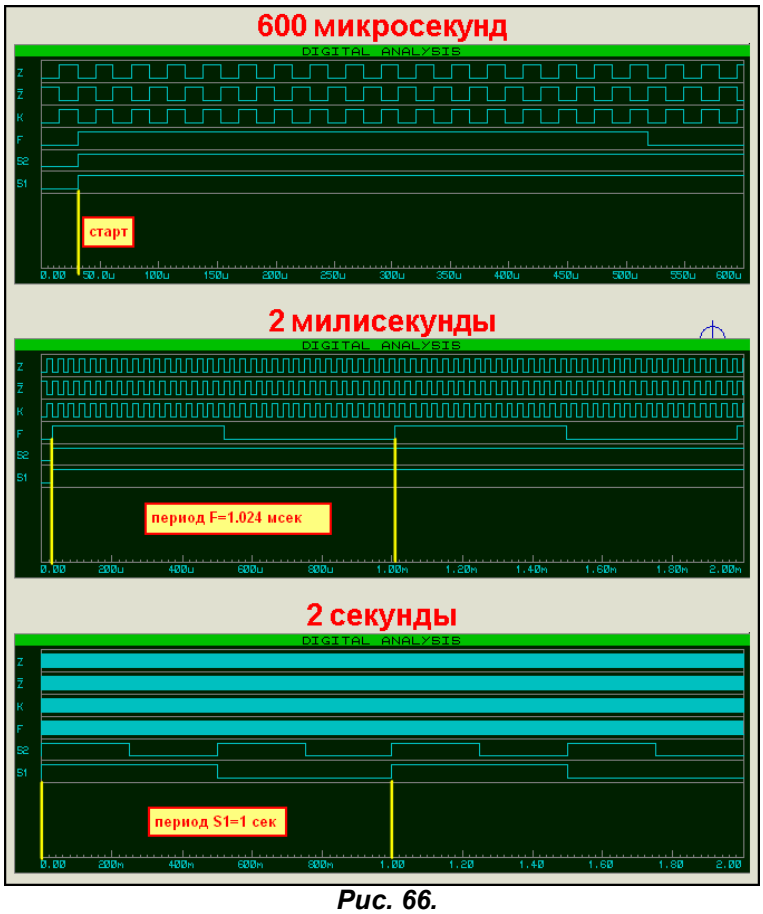


Рис. 66.

Для чего я привел эти графики? Очень важный момент, при создании сложных многофункциональных моделей – это совпадение фаз сигналов с их реальными прототипами. Если на секунду вернуться к временной диаграмме на Рис. 60, можно заметить, что передний фронт импульсов на выходах **F**, **S2** и **S1** должен совпадать, что я и проконтролировал по верхнему графику. Ну и попутно проверили периоды следования импульсов.
Я не буду здесь приводить дочерний лист с пятнадцатиразрядным счетчиком, потому что он займет слишком много места. Желающие посмотрят его самостоятельно в тестовом проекте **Modul_Gen.DSN** в папке **MODUL1** вложения. Мы же, убедившись в работоспособности делителя, идем дальше и начинаем процесс создания внутреннего генератора модели. Для этого помещаем наш примитив **CLOCK** с именем **OSC** на дочерний лист и подключаем его на вход инвертора **U1**. В свойствах генератора ставим галочку **Edit all properties as text** и вводим вручную следующие строки:
PRIMITIVE=<CLKOSC>
PRIMTYPE=DIGITAL!
CLOCK=<CLOCK>
Строка **INIT=0** там уже присутствует и в ней я просто убрал фигурные скобки, чтоб она стала видимой (Рис. 67).

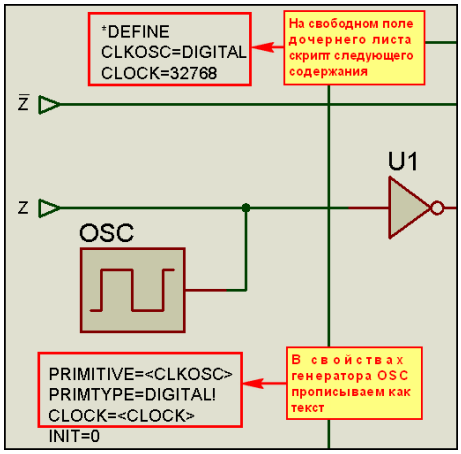


Рис. 67.

Кроме того, на свободном поле дочернего листа размещаем текстовый скрипт следующего содержания:

***DEFINE**
CLKOSC=DIGITAL
CLOCK=32768

Этот скрипт назначает нашему генератору свойства, принятые по умолчанию. Частоту я взял для стандартно используемого с К176ИЕ12 кварцевого резонатора 32768Гц. Теперь возвращаемся на основной лист и запускаем симуляцию. Если все сделано правильно, то наш модуль будет функционировать так же, как и с внешним генератором. Отдублируем модуль на этом же листе и зададим ему другое имя, чтобы не было совпадения и ошибки, например **GEN_DEVIDER_1**. Кроме того у копии модуля нам придется проследовать на дочерний лист и вручную назначить имя генератору тоже отличное от первого варианта, например **OSC1**. Теперь в свойствах модуля **GEN_DEVIDER_1** впишем строку **CLOCK=16386** и вновь запустим симуляцию (Рис. 68).

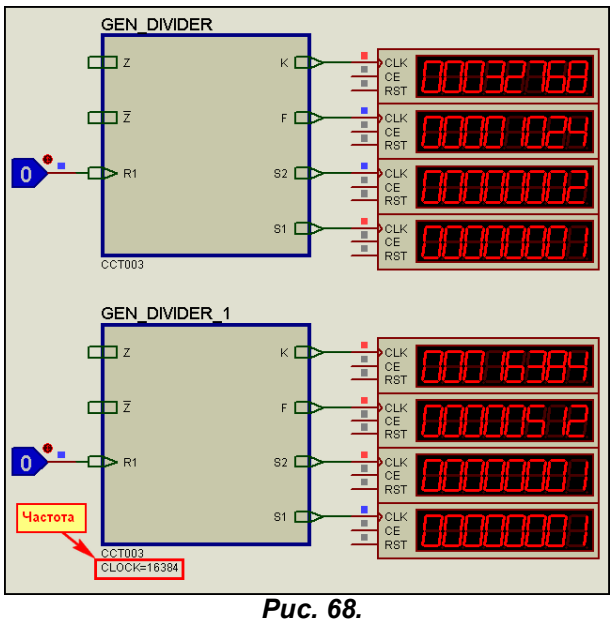


Рис. 68.

На всех выходах свежеиспеченного модуля частота упала вдвое. Ну, вот мы и получили управление частотой генератора модели с основного листа. Для чистоты эксперимента зайдем на дочерний лист первого модуля и выполним компиляцию MDF с этого листа. Я приложил получившийся файл **Modul_Gen_2.MDF** в папку **MODUL1_1** вместе с этим тестовым проектом **Modul_Gen_2.DSN**. Ниже приведено начало этого файла:

```
*PROPERTIES,2
CLKOSC=DIGITAL
CLOCK=32768

*MODELDEFS,0

*PARTLIST,18
OSC,CLOCK,,CLOCK=<CLOCK>,INIT=0,PRIMITIVE=<CLKOSC>,PRIMTYPE=DIGITAL!
U1,INVERTER,INVERTER,PRIMITIVE=DIGITAL
U2,INVERTER,INVERTER,PRIMITIVE=DIGITAL
```

Как и следовало ожидать, скрипт ***DEFINE** превратился в ***PROPERTIES**. Но самое главное, обратите внимание на первую строчку ***PARTLIST**. Сравните ее с аналогичной для генератора **OSC** модели **CD4060** и убедитесь, что они полностью совпадают. Мы на верном пути!
Теперь займемся свойством **CLKOSK**, которое определяет тип модели генератора **OSC**. По умолчанию оно равно **DIGITAL** - цифровой. Мы же на основном листе в свойствах модуля впишем ему значение **NULL** - пусто. Пробуем симуляцию и видим, что внутренний генератор перестал работать, что нам и нужно. Теперь попробуем запустить наш модуль с внешней RC цепочкой, задав свойство **IC=0** одной из внешних цепей. Наш модуль опять запустился и частота импульсов на выходе определяется внешними компонентами (Рис. 69). Этот пример приложен в проекте **Modul_Gen_2.DSN**, который расположен в папке **MODUL1_1**.
Таким образом, мы пришли к варианту построения генератора аналогичного модели **CD4060**. Нам осталось только расписать таблицу ***MAP ON** и заставить ее работать. Давайте окончательно воспроизведем вариант CD4060. Для этого изменим скрипт ***DEFINE** дочернего листа следующим образом:

```
*DEFINE
CLOCK=DEFAULT
```

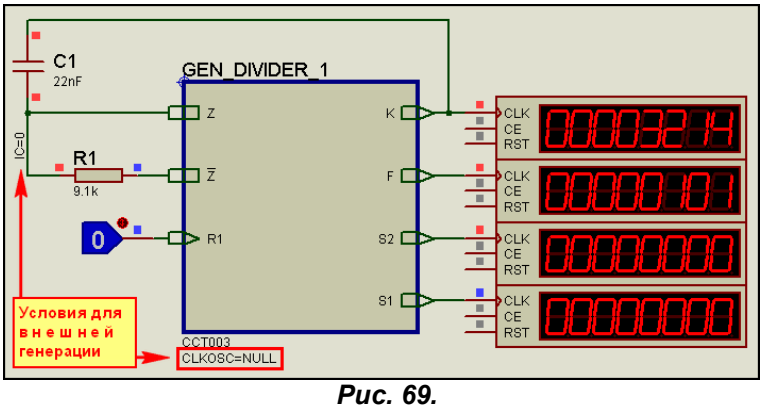


Рис. 69.

Где-нибудь на свободном поле листа поместим еще один скрипт:

```
*MAP ON CLOCK
DEFAULT : CLKOSC=DIGITAL
EXTERNAL : CLKOSC=NULL
```

Ну, а в свойствах модуля на основном листе впишем строку **CLOCK=32768**. Запустим симуляцию – работает. Для дубля модуля зададим **CLOCK=EXTERNAL** и навесим внешние RC элементы, не забыв задать **IC=0**. Тоже работает (Рис. 70). Вот теперь у нас почти полностью аналогичный генератор. Если мы задаем цифровое значение свойству **CLOCK** на основном листе, то включается внутренний генератор, а если задать ему значение **EXTERNAL** – то необходима внешняя RC-цепь или внешний генератор на вход **Z**. Таким образом, с помощью одного свойства компонента мы управляем как частотой внутреннего генератора, так и режимом его работы.

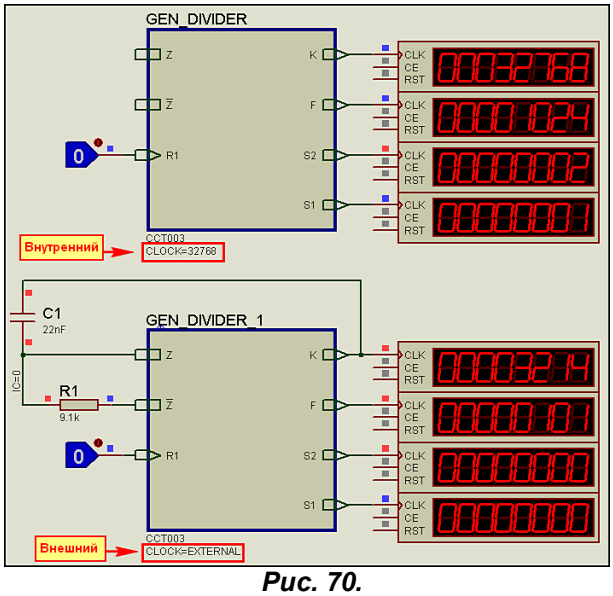


Рис. 70.

Этот пример в папке **MODUL1_2** под именем **Modul_Gen_4.DSN**. Ну и в заключение проверяем работу модуля при подаче сигнала от внешнего генератора – пример **Modul_Gen_5.DSN**. Вот и весь материал, который я хотел довести до вас в этом параграфе. Но наиболее внимательные уже, наверное, заметили, что я тихо замолчал про параметр **CLKGATE**, который имеется в **MAP ON CLOCK** для модели **4060**. Честно говоря, я так и не понял до конца – зачем он там присутствует, поскольку упоминается только в таблице и далее нигде не встречается. Наличие или отсутствие его особого значения на работу модели не оказывает, в чем мы убедились, «забыв» включить его в карту для модуля **GEN_DIVIDER**. Есть у меня подозрение, что **CLKGATE=NULL** команда симулятору **PROSPICE** не создавать виртуальный генератор для модели при использовании режима внешнего. Возможно, таким образом, автор модели **4060** преследовал цель снизить нагрузку на ЦП компьютера в режиме внешнего генератора модели. Если попадет под руку какой-нибудь доходяга Пентиум III, попробую проверить, но на тех монстрах, которые сейчас в наличии эффектов не заметно. Наиболее сложная в понимании часть будущей модели **К176ИЕ12** на этом завершена. Нам осталось сформировать генератор для динамической индикации и формирователь минутного импульса. Об этом далее…

Возврат к содержанию

6.8. Пример создания полной схематичной модели счетчика К176ИЕ12. Часть 3 – формирование сигналов динамической индикации и минутного импульса.

Четырехтактный формирователь сигналов управления динамической индикацией, пожалуй, самый простой узел внутренней структуры **К176ИЕ12**. Если проанализировать временную диаграмму на рисунке 60, мы увидим, что это последовательная цепочка импульсов с частотой 128Гц на выходах **T1**, **T2**, **T3**, **T4** микросхемы. Из литературы нам известно, что для таких целей наиболее часто используют кольцевые счетчики 1 из n на регистрах сдвига, замкнутых в кольцо. Так мы и поступим. Поскольку это модель, а не реальное устройство схема самовосстановления режима работы такого счетчика будет излишеством, а для того чтобы обеспечить предустановку одного из триггеров в единичное состояние при старте симуляции воспользуемся стандартным для триггеров свойством **INIT**. Кто забыл – смотрите HELP для примитива D-триггера. Для первого из четырех триггеров установим **INIT=1**. Ну и еще один нюанс. Нам нужна частота импульсов 128Гц, триггеров – четыре. Включаем в голове математику, а кто не изучал – арифметику. Тактовая частота для регистра сдвига должна быть в 4 раза выше, т.е. 512Гц. Проверяем наш регистр в отдельном проекте – **128Hz.DSN** (вложение папка **4Takt**). Схема формирователя и график его переключения представлены на Рис. 71.

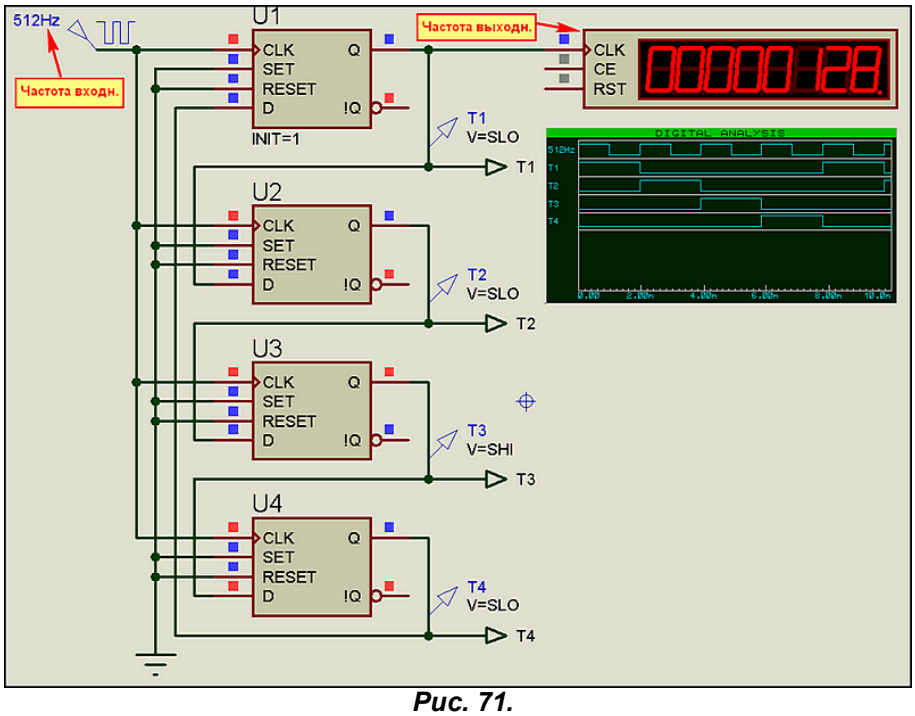


Рис. 71.

Нам осталось только поместить данный формирователь на дочерний лист модуля, созданного ранее, присоединить его к выходу делителя с частотой 512Гц – это выход элемента **U8** в делителе и привести в порядок нумерацию элементов, чтобы не было конфликта. Кроме того, добавим на основном листе нашему модулю соответствующие выходы **T1…T4**. Чтобы добавить реальности при тестировании, введем начальный сброс модуля. Для этого на его вход **R1** я подал цифровой сигнал перепада с 1 на 0 с помощью генератора **Digital Edge** (**DEDGE**) из левого меню **Generator Mode** с параметрами, представленными на Рис. 72.

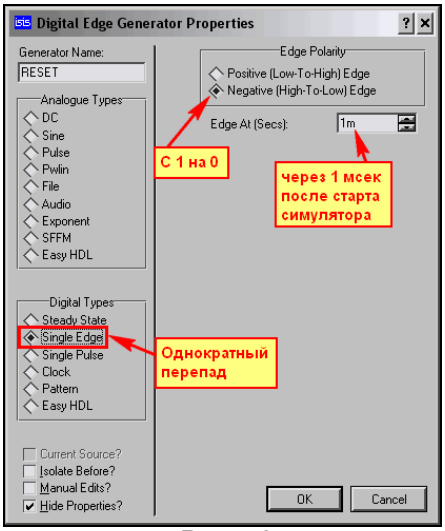


Рис. 72.

Это позволит нам более четко отслеживать поведение нашей модели. Тестируем новый вариант модуля в интерактивном режиме – все нормально, на выходе **T1** присутствуют требуемые 128Гц. Теперь проведем тест с помощью цифрового графика (Рис. 73), и тут же обнаруживается несоответствие временной диаграмме из рисунка 60.

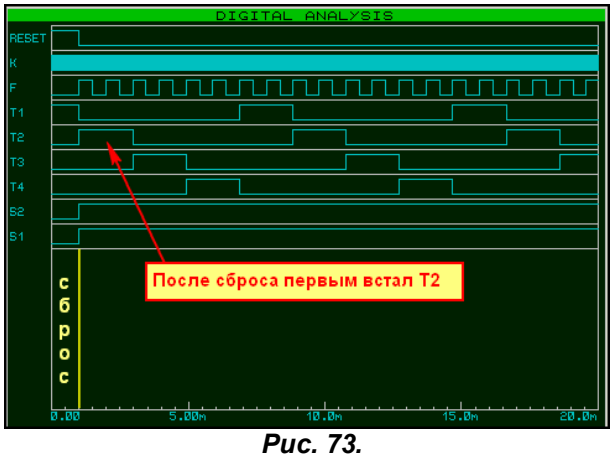


Рис. 73.

Я нарочно оставил это проект под названием **Modul_2_bad.DSN** во вложении **MODUL2**, чтобы вы могли сравнить его с хорошим **Modul_2_good.DSN**. Разберем причину моего косяка и быстро устраним. Согласно рисунку 71 я поставил предустановку **INIT=1** для первого триггера, т.е. в момент времени 0 – он установлен в единицу и по первому же положительному перепаду на счетных входах **CLK** нашего регистра она уедет во второй триггер с выходом **T2**, что мы и получили на графике рисунка 73. Ошибка настолько очевидна, что правится в шесть секунд. Переносим **INIT=1** из первого триггера формирователя в четвертый и снова тестируем (Рис. 74). Статус-кво восстановлен, и все соответствует документации.

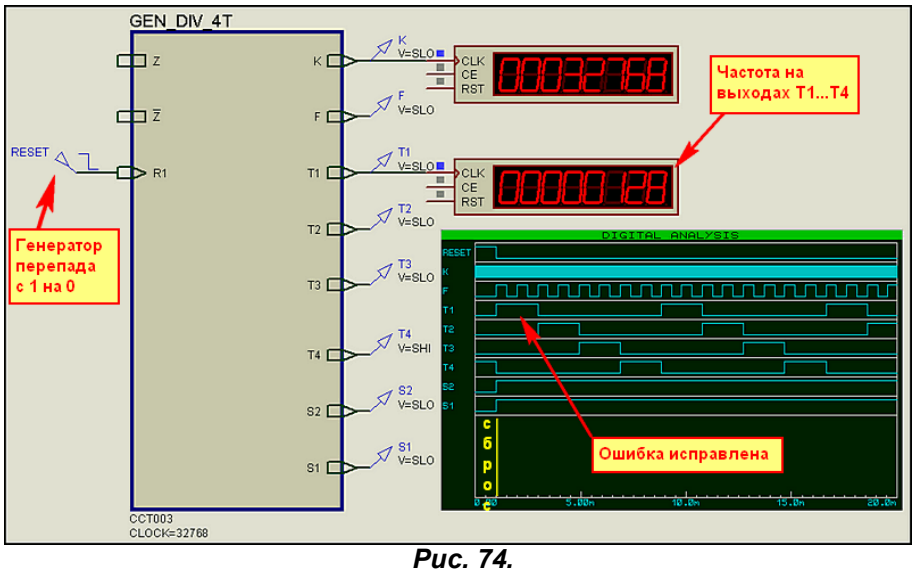


Рис. 74.

Фактически, нам осталось сформировать минутный импульс, и наша модель будет почти готова. Тут тоже придется немного «попотеть», поскольку импульс довольно хитрый. Согласно временной диаграмме передний его фронт возникает по прошествии 39 сек, а длительность составляет 20 сек, т.е. после 59 сек мы должны иметь на выходе **M** опять ноль. Эту часть структуры **К176ИЕ12** для проверки мы также соберем сначала в отдельном проекте. Не будем мудрствовать и сформируем эту часть все на тех же D-триггерах по схеме суммирующего счетчика. Десятичное число **60** эквивалентно двоичному – **111100**, т.е. потребуется 6 разрядов (триггеров). Кроме того, придется входы **RESET** раздвоить по **OR** (ИЛИ), так как нам необходимо иметь сброс и от внешнего сигнала и при достижении отсчета 60 секундных импульсов (1 минуты). В качестве формирователя длительности импульса используем все тот же примитив **DTFF**, но уже как обычный RS-триггер. На вход **S** собираем через примитив **AND** (И) число 39 (двоичное значение **100111**), на вход **R** - число 59 (двоичное – **111011**). В нашем случае это допустимо, т.к. указанные числа имеют значительный разброс значащих единиц, но в некоторых случаях такая логика работы входов невозможна и приходится использовать инверсию незначащих нулей. В цифровых примитивах Протеуса это достигается добавлением для соответствующего входа строки **INVERT=Dx** в окне свойств, где **x** – номер входа (счет ведется с нуля, т.е. верхний вход пятивходового элемента **AND** будет **D0**, а нижний **D4**). Схема достаточно громоздкая, но я ее поджал за счет применения шины и приведу полностью на Рис. 75 . Во вложенном в папку **MODUL3** файле **Counter60.DSN** представлен тестовый проект.

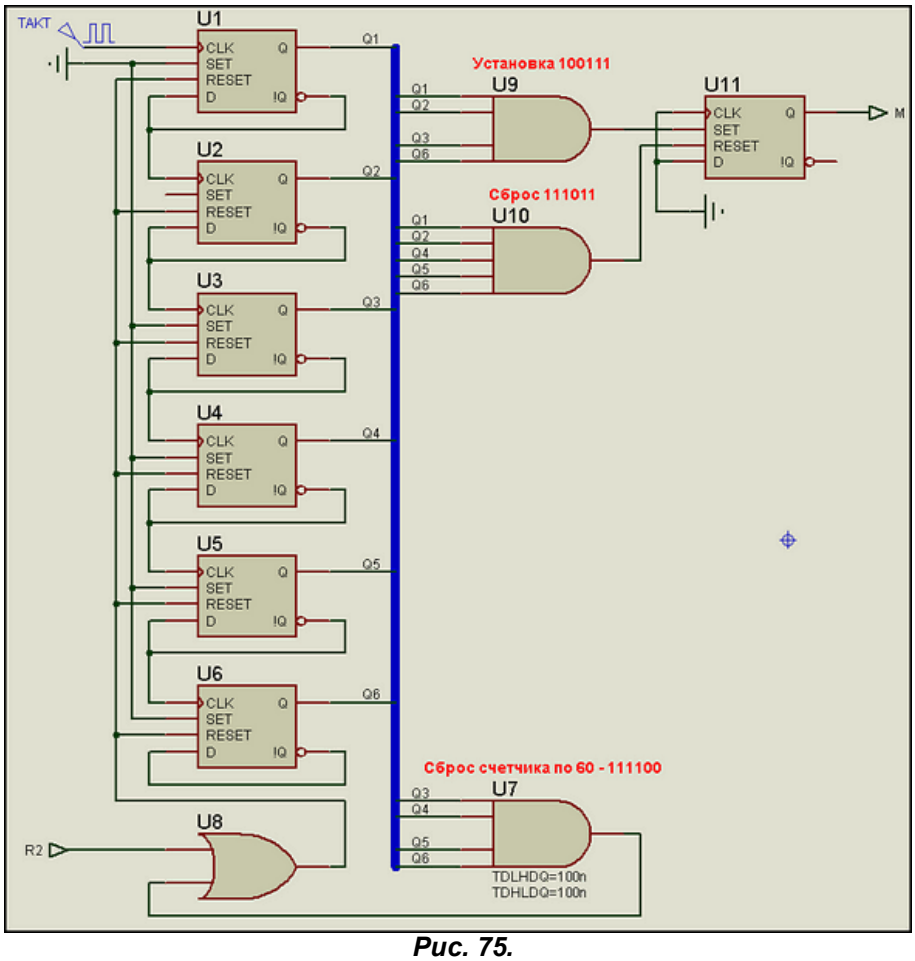


Рис. 75.

Обратите внимание, что для логического элемента **U7 (**AND_4), формирующего импульс сброса счетчика по достижении числа 60 установлены свойства **TDLHDQ=100n** и **TDHLDQ=100n** (нарастание и спад импульса по 100 наносекунд). Иначе он просто не будет виден на графике, поскольку слишком короткий. График работы схемы представлен на рисунке 76.

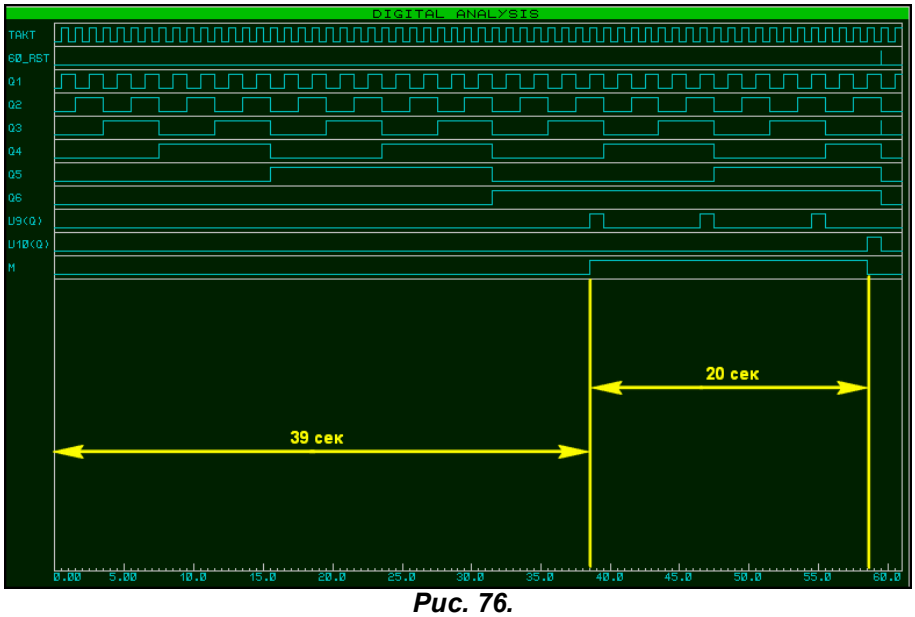


Рис. 76.

Если запустить симуляцию тестового проекта, то, манипулируя кнопкой **Gate Polarity** прибора **Counter Timer** можно измерить длительность и паузу минутного импульса. Генератор я уже поставил на 1 Гц для имитации реальных условий. Правда, занятие это в реальном времени утомительное, приходится честно высиживать минуту, наблюдая за таймером.
Нам осталось перенести формирователь минутного импульса в дочерний лист модуля и добавить вход **R2** и выход **M** к нему на основном листе. Я проделал эту манипуляцию в проекте **Modul_3.DSN** соответствующей папки вложения. Там же представлены графики с различными временными интервалами, на которых видны те или иные выходные сигналы модуля.
Нам же осталось для придания нашей будущей модели более реальных характеристик на дочернем листе для каждого элемента задать соответствующие временные задержки фронтов. Вот тут-то и вспомним добрым словом разработчиков программы Протеус. В финале структура содержит 32 цифровых элемента. Нам предстоит каждому задать **TDLHDQ**, **TDHLDQ** и т.д. Как работенка – впечатляет? Но на помощь опять приходит **PAT**. Поскольку и здесь мы воспользуемся аналогией с моделью **4060**, то в конечном итоге сведем все это к таблице **MAP ON**. Начнем с того, что зададим всем триггерам **TDHLCQ=<TDCQ>** и **TDLHCQ=<TDCQ>**. Вызываем окно **Property Assignment Tools** и набираем в окне **String** строку **TDHLCQ=<TDCQ>** (Рис. 77).

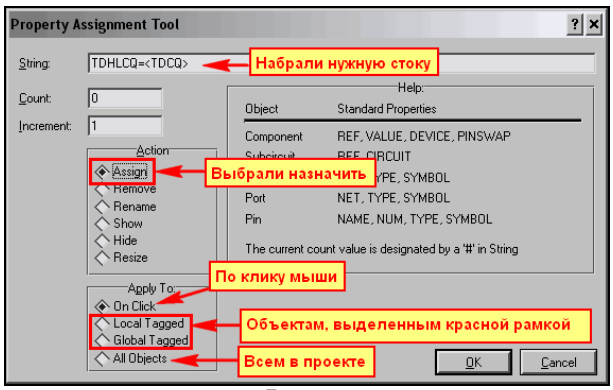


Рис. 77.

Если не хотите, чтобы это значение подсвечивалось под каждым элементом, поставьте по краям фигурные скобки. Но мне в данном случае это не нужно, я хочу себя контролировать. В рамке **Action** нам предстоит действие назначить, т.е. **Assign**. Дальше у нас три пути:

- Можно стандартно выбрать **On Click** и честно прощелкать мышью все нужные компоненты. Процедура, которую многие уже наверняка освоили. Наводим до появления «указательного перста» с зеленым прямоугольником справа и щелкаем левой кнопкой мыши. 26 триггеров – столько же щелчков. Шурик в «Кавказской пленнице» пожалел птичку, ну а мне – мышку жалко…
- Можно заранее выделить (обвести, удерживая нажатой кнопку мыши) район с нужными элементами, а лишь потом вызвать PAT. Причем он сразу предложит в рамке **Aply To** действие **Global Tagged**, т.е. для всех выделенных красным компонентов. Кстати, я так и не разобрался, чем отличается **Local Tagged** – все равно присваивается выделенным. Это уже более прогрессивный метод и за один раз при правильном построении схемы (похвалюсь – как у меня в данном случае) можно зацепить достаточно много объектов.

- Ну, и наконец, можно выбрать **All Objects**, но тут надо быть предельно осторожным. В частности, при использовании на дочернем листе модуля мы присвоим это свойство и всем объектам на основном листе – оно нам надо? Такие вещи хороши, когда питание переназначаешь, вроде **VCC** или **VSS**, но зачем примитиву генератора **TDHLCQ**? А ведь назначит, я проверял. Еще удобно, когда надо скрыть/показать что-то у всех объектов – этим мы воспользуемся позже.

Ну, в общем, в данном случае мне идеально подходит для назначения второй вариант. Я выделяю часть триггеров, стоящих в одной колонке и вызываю **PAT**. Ввожу строку как на рисунке 77 и давлю **OK**. Под соответствующими триггерами сразу появляется видимая строка (Рис. 78). Вся операция прошла в три действия, причем в последнем участвовал только один триггер минутного импульса. Последняя введенная строка сохраняется в **PAT**, и набирать мне ее пришлось только один раз.

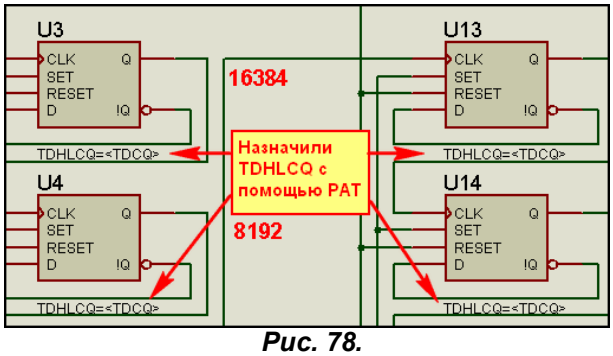


Рис. 78.

Но присмотритесь внимательно – мне ведь еще пару свойств прописывать – а куда? Со свободным местом в проекте – полный напряг. Но я не зря затеял практический повтор материала по **PAT**. Цитирую опять, пусть и не дословно, классику комедии – «Кавказскую пленницу»: «Тот, кто нам мешает – тот и поможет…», т.е. **PAT**. Для начала надо визуально убедиться, что я назначил **TDHLCQ** всем триггерам. Ну а больше мне на него любоваться незачем. Вызываю окно PAT, в **String** оставляю только аббревиатуру **TDHLCQ**, а в **Action** выбираю **Hide** – скрыть. И вот теперь в рамке **Aply To** выбираю **All Objects** – пригодилось! Все, зарыл **TDHLCQ**. Но в свойствах триггеров, если выбрать соответствующий пункт оно видно (Рис. 79). Видно его и в режиме **Edit all properties as text**.

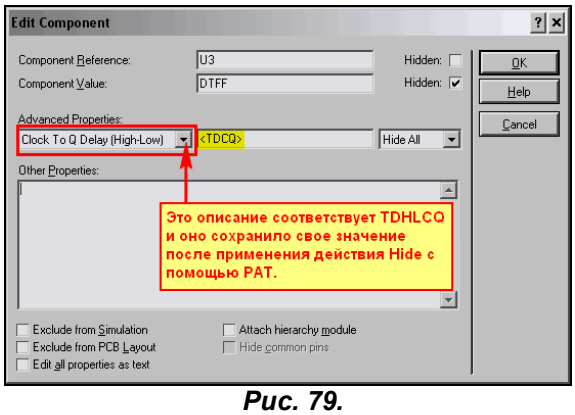


Рис. 79.

А то, что я применил **Hide** ко всем объектам в данном случае не криминально, поскольку это свойство есть только у счетчиков и триггеров и сработало оно только для них. Аналогичным образом всем триггерам назначаем и **TDLHCQ=<TDCQ>**, а затем и **TDRQ=<TDMRQ>**. Ну уж будем последовательными и всем логическим элементам в проекте назначим **TDHLDQ=<TDOSC>** и **TDLHDQ=<TDOSC>**.

Ну и, закончив все вышеуказанные операции, осталось на свободном поле дочернего листа поместить скрипт следующего содержания:

```
*MAP ON VOLTAGE
5V : TDOSC=35n, TDCQ=25n, TDMRQ=100n
10V : TDOSC=13n, TDCQ=10n, TDMRQ=40n
```

Пока я нахально слизал задержки из модели **4060**, чтоб протестировать нашу модель, но позже подумаем, как и сделать более реальными. Снова тестируем наш модуль. Опля, что-то не то. Ну да, забыл в свойствах модуля на основном листе прописать напряжение в соответствии с только что написанной картой. Добавляем там строчку **VOLTAGE=10V**, и все становится на свои места. Этот пример в файле **Modul_4.DSN** папки **MODUL4** вложения. Ну вот, теперь у нас все подготовлено и протестировано, осталось только закончить модель. Об этом в следующем параграфе.

Возврат к содержанию

6.9. Пример создания полной схематичной модели счетчика К176ИЕ12. Часть 4 – создаем MDF и законченную Schematic модель.

Наступила пора извлечь из небытия нашу графическую модель **К176ИЕ12**, созданную в **п.6.6**. Помещаем ее в поле проекта и в свойствах устанавливаем флажок **Attach Hierarchy Module**. Заходим на дочерний лист и втаскиваем туда (меню **File=>Import Section**) секцию, которую создали с дочернего листа предыдущего проекта **Modul_4.DSN**.

Да, давно это было… склероз, однако. Придется кое-что еще поправить на дочернем листе. В тестовом проекте **Modul_4** вход формирователя минутного импульса напрямую соединен с выходом **S** делителя, а в реальности это отдельный вход **C** у графической модели. Ну, это поправимо и сейчас – разрываем и добавляем на дочернем листе терминал **C** на вход минутного формирователя (Рис. 80).

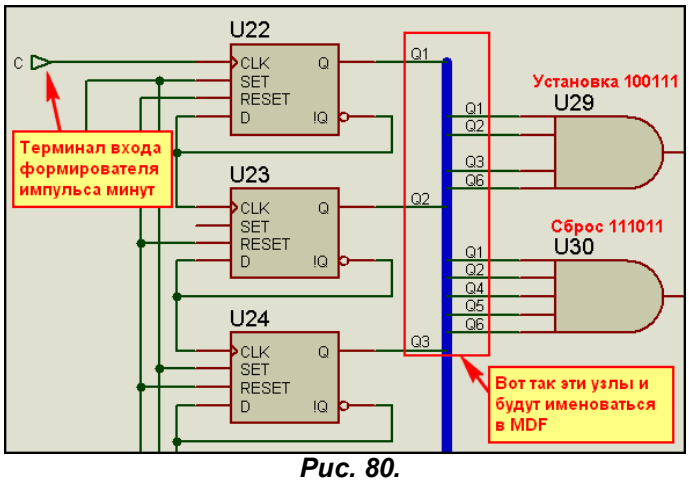


Рис. 80.

Возвращаемся на родительский лист и заходим в свойства модели. Там в первую очередь нам необходимо прописать вручную частоту и напряжение так, как мы это делали в предыдущем проекте для модуля. Иначе, при запуске симуляции вылетит «красная птичка». Также, если мы ранее не назначали нашей модели префикс и **VALUE**, то тоже пора дописать (Рис. 81), хотя пока это и не обязательное требование.

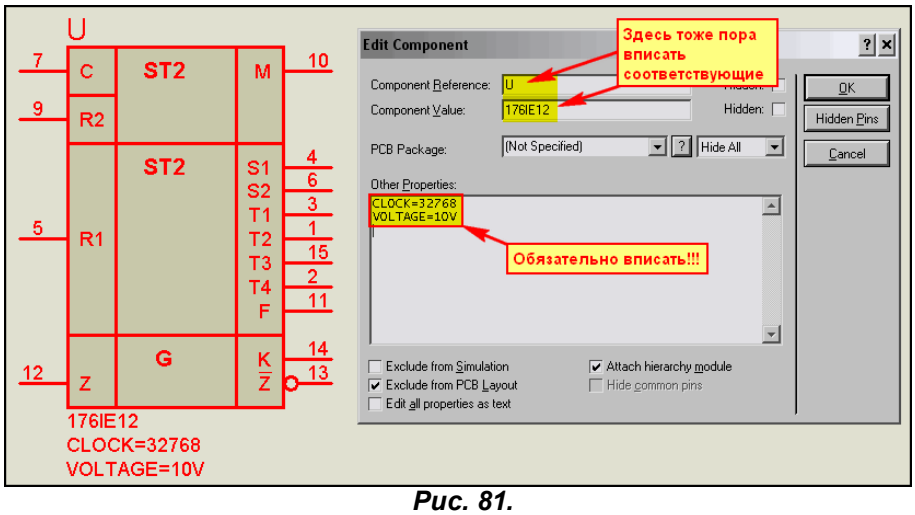


Рис. 81.

Теперь соединяем **S1** с **C** и заземляем входы сброса **R1** и **R2**, на выходы вешаем зонды и запускаем симуляцию. Если ранее нигде не накосячили, все прекрасно работает. Останавливаемся и. … А вот и горчичнички (Рис. 82).

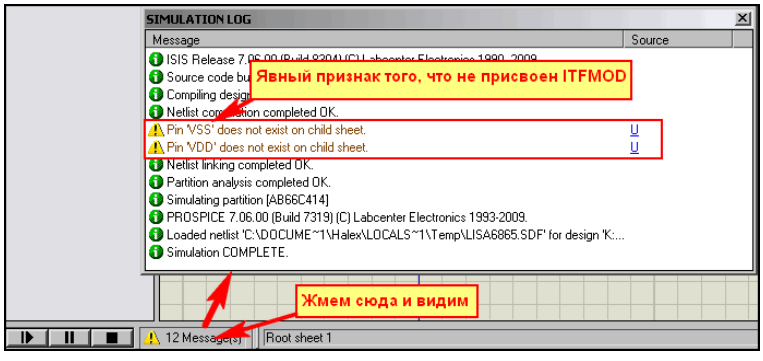


Рис. 82.

Ну конечно, про **ITFMOD** мы забыли. А ведь при создании графики мы прилепили скрытые пины питания **VSS/VDD**. Вот симулятор нас и предупредил, что они отсутствуют на дочернем листе. Но мы их и не собираемся там рисовать, а просто добавим в свойствах одну строку: **ITFMOD=CMOS**. Пробуем еще раз – все, проблема со специей (**SPICE**), т.е. горчицей разрешилась (как-бы каламбур). Я оставлю эти два примера **Graphic_with_Child** с окончаниями **bad** (плохой – без ITFMOD) и **good** (хороший – с ITFMOD) в папке **Graphic_and_Hierarchy** вложения. Результаты теста на рисунке 83.

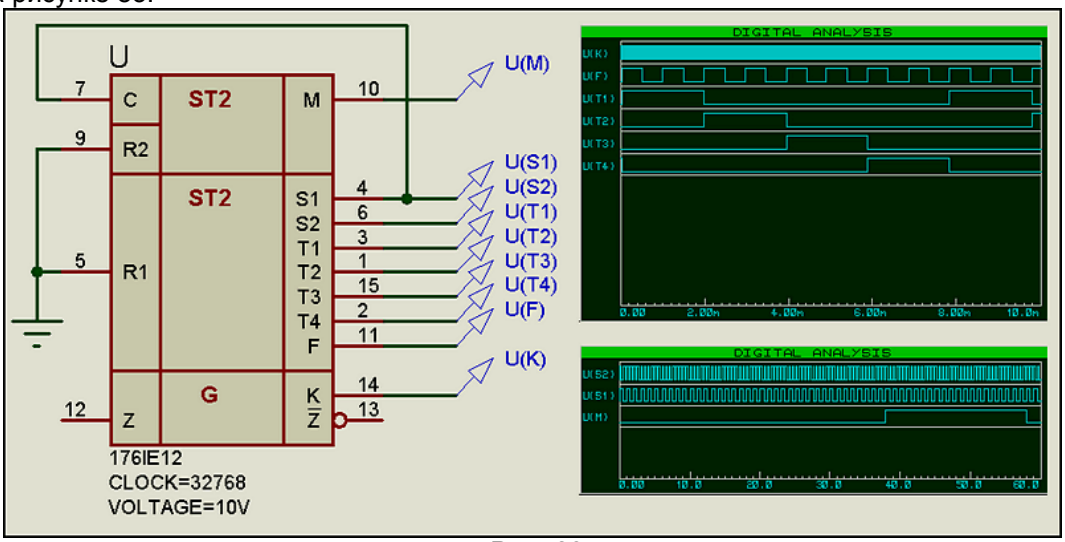


Рис. 83.

Поскольку тест прошел удачно, и все наши графики приближены к реальности, можно приступить к формированию MDF. Но прежде необходимо до конца разобраться с задержками, ведь у нас по-прежнему они используются от модели **4060**. Когда в **п.6.2** мы пробовали создать модель **176ЛА7**, я упоминал, что стандартные задержки распространения для этой серии при питании 9В равны 250 наносек. Это все, что мне удалось найти в справочниках. Ну что, давайте отталкиваться от этого. Заменим в карте **MAP ON VOLTAGE** на дочернем листе задержки следующим образом:

```
*MAP ON VOLTAGE
5V : TDOSC=400n, TDCQ=520n, TDMRQ=1200n
10V : TDOSC=250n, TDCQ=325n, TDMRQ=750n
```

Чтоб не возникало кривотолков – поясню. Для 10(9)V я взял стандартную задержку распространения для логических элементов - в нашей модели это **TDOSC**. Для 5V она увеличится и где-то мне попадалось значение 400 наносек. Значения для триггеров **TDCQ** и **TDMRQ** я просто взял и просчитал на калькуляторе пропорционально от модели 4060. Понимаю, что кривлю душой, но садиться с осциллом замерять реальные – мне, честно говоря, ну ни в жилу. Да и нет у меня пачки микросхем, а с единичных экземпляров – все равно будет лажа. Так что, примите – как есть. В общем, тестируем с этими задержками и убеждаемся, что никаких излишних эксцессов не вылезает. Протестировать придется и при **VOLTAGE=5V**, чтобы быть до конца уверенными в себе. Этот проект **Model_with_child.DSN** представлен в папке **Model_and_MDF** вложения.
Теперь у нас все готово к компиляции MDF, но прежде одно маленькое замечание, которое предшествует небольшому сюрпризу для вас. Обратимся снова к рисунку 80. Помните, я для ужимания схемы прибегнул к применению шины? Соответственно проводникам, присоединенным к шине, пришлось назначить соответствующие метки (лейблы). Так вот, при компиляции MDF эти метки и превратятся в узлы для этих цепей. Те цепи, которые не помечены и не присоединены к входным/выходным терминалам будут просто автоматически пронумерованы компилятором типа **#00001**, **#00002** и т.д. Цепи, входящие в шину и помеченные с помощью лейблов в **NETLIST** будут выглядеть так:

```
Q1,4
Q1,LBL
U30,IP,D0
U29,IP,D0
U22,OP,Q
```

Это пример узла с меткой **Q1** – выход триггера **U22**. На работоспособность модели это не повлияет, но если кого-то смущает данный момент – придется перерисовать дочерний лист, исключив из него шину, а все соединения сделать отдельными проводниками.

Заходим на дочерний лист и вызываем компилятор через меню **Tools=>Model Compiler**. Указываем путь сохранения к нашей текущей папке проекта, а также меняем имя файла, чтоб совпадало с именем модели – **176IE12**, хотя это и необязательно, но так меньше путаницы (Рис. 84).

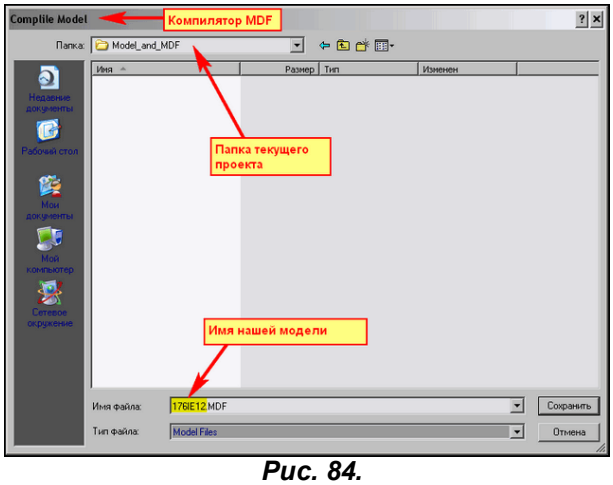


Рис. 84.

Готовый файл **176IE12.MDF** лежит в папке **Model_and_MDF** вложения. Теперь настала пора присоединить его к графической модели. Я не стану для этой цели использовать уже находящуюся в поле проекта графику с присоединенным **Child Sheet**. Причин несколько. Во-первых, я хочу оставить ее вам для сравнения. Во-вторых, там уже прописаны вручную свойства CLOCK, VOLTAGE и ITFMOD, а я хочу показать их присоединение с нуля. Ну и, в-третьих, эти уже присоединенные свойства все-равно пришлось бы править на третьей вкладке **Make Device**. Поэтому, мы начнем с голой графики, а для этого используем еще одну графическую модель на этом же листе проекта. Выделяем ее и запускаем незаслуженно подзабытую нами **Make Device**.

На первой вкладке, если еще не прописано, вставляем **Device Name** – **176IE12** и **Reference Prefix** – **U**. Я не стал здесь оригинальничать и оставил префикс, принятый в **ISIS** по умолчанию для микросхем. Лишний раз напомню, что здесь мы можем применить только английский язык. Ни какого русского языка Протеус здесь не потерпит (Рис 85). Вторую вкладку можно спокойно пропустить, там нам исправлять и добавлять нечего. На третьей вкладке нам предстоит добавить сразу несколько свойств. Начнем с **CLOCK**. Выбираем через кнопку **New** опцию **Blank Item** – (чистая позиция) и заполняем ее в соответствии с рисунком 86. **Name** должно быть только **CLOCK** и никаким другим. Именно так оно прописано у нас в скриптах на дочернем листе модели, а теперь уже и в файле **MDF**. В графе **Description** (описание) допустим русский язык. Я ввел фразу: **Частота (Внешн.=External)**. Именно так она будет отображаться в окне свойств. Тип переменной Type я оставил по умолчанию, т.е. **String**. При этом, отсутствуют какие либо ограничения по вводу значений в эту строку. В прототипе **4060** используется тип **Float** и ограничение **Positive, Non-Zero**, т.е. действует защита от ввода отрицательного и нулевого значения. Возможно, и есть смысл сделать аналогично, но я надеюсь, что в России хотя бы одной бедой когда-то станет меньше, и отрицательные значения частоты туда писать не будут. Следующий **Type**, относящийся к окошку ввода оставляем **Normal**, т.к. нам необходимо будет окно доступное для записи значения частоты. А вот **Default Value** (по умолчанию) я в последний момент решил поменять на **External**. Причина все та же, не очень надеюсь, что наши пользователи запомнят, как пишется это слово. И, хотя я и ввел подсказку в описании, но так надежнее, да и числовое значение вбить на это место всегда быстрее, даже полному чайнику в расположении символов на клавиатуре.

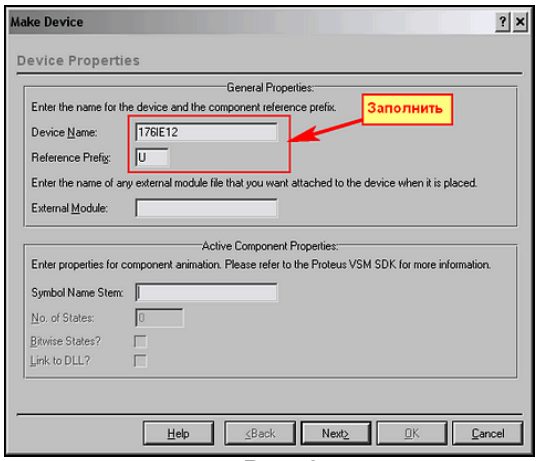


Рис. 85.

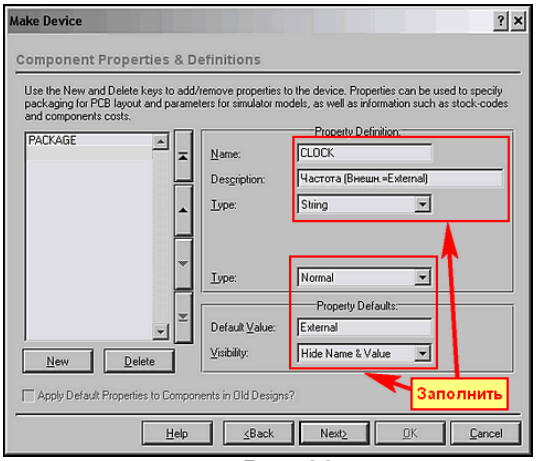


Рис. 86.

Снова давим кнопочку **New => Blank Item**. При этом свойство **CLOCK** запомнится в окне свойств модели, а нам предстанет чистое окно для заполнения следующего с именем **VOLTAGE** (Рис. 87). Заполняем его аналогично. В описании я ввел фразу **Задержка для питания**. Тип переменной в данном случае **Keyword (Non-Editable)** (ключевое слово – нередактируемое). Здесь уместна «защита от дураков», ведь упорно будут пытаться вписать 9V – пусть помучаются. В появившемся окне Keywords вводим наши возможные значения (те, что в **MAP ON VOLTAGE** в начале строк до двоеточия). Значения разделяем запятой без последующего пробела. Ну и по умолчанию **Default Value** выставляем 10V (наиболее близкое к стандартному питанию).
И опять щелкаем **New**, но на этот раз выбираем стандартное свойство из раскрывающегося списка ITFMOD. Здесь просто в **Default Value** задаем ему значение **CMOS**, а все остальное уже заранее заполнено в шаблоне (Рис. 88).

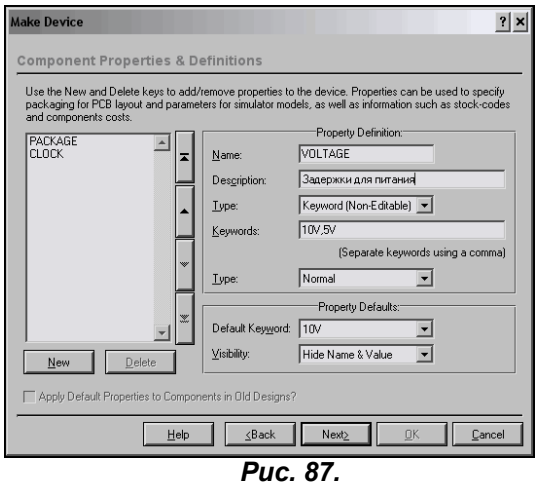


Рис. 87.

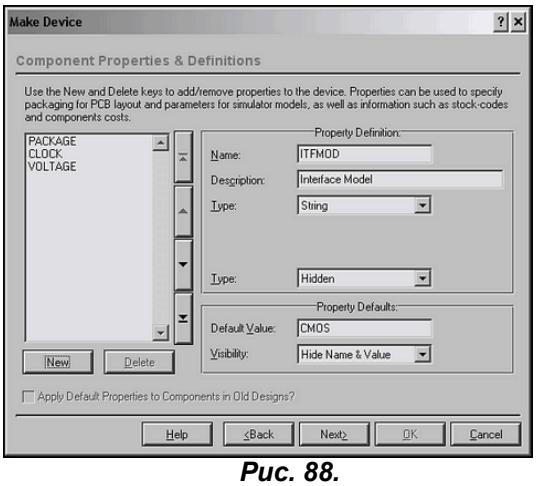


Рис. 88.

И в последний раз кликаем **New** и выбираем из списка **MODFILE**. Тут в окошке **Default Value** вводим имя нашего файла **176IE12.MDF** (Рис. 89) и на этот раз давим кнопку **Next**. Со свойствами мы покончили. На последней вкладке вводим данные так, как нам хочется (Рис. 90) и сохраняем нашу модель пока в библиотеке **USRDVC**.

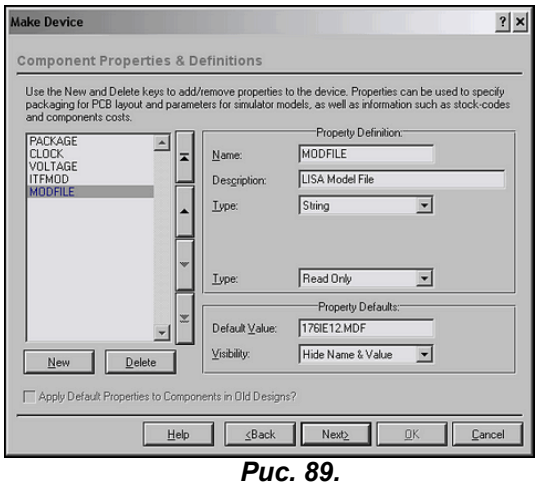


Рис. 89.

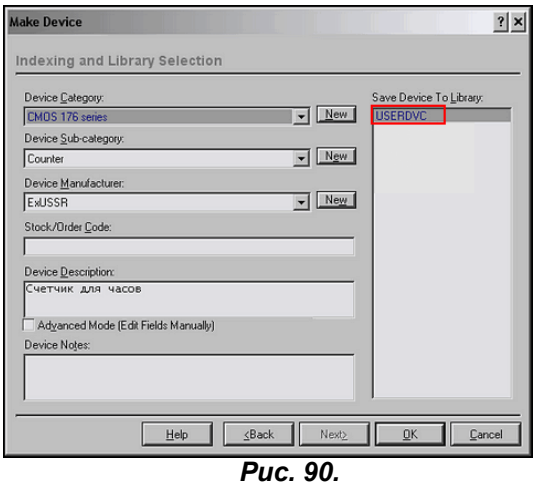


Рис. 90.

Ну, вот и все, наша модель готова к использованию. Теперь необходимо поместить файл **176IE12.MDF** в папку **MODELS** Протеуса, чтобы он был доступен из любого нового проекта. Создаем тестовые проекты и проверяем работу микросхемы **176IE12**. У меня во вложении это: **TEST_176IE12_int.DSN** для внутреннего и **TEST_176IE12_RC.DSN** для RC-генератора в папке **Model_and_MDF**. На этом наши мучения с созданием модели счетчика **К176ИЕ12** закончены.
В заключение несколько слов о путях дальнейшего развития этой модели. Первоначально у меня были замыслы использовать свойство **SCHMITT** для элемента **U2** по схеме дочернего листа для автоматического запуска генератора при наличии внешних цепей и переводе в режим **External**. Однако если вы внимательно изучали материал, посвященный генераторам в **п.6.3** и **6.4**, то обратили внимание, что при использовании этого свойства частота получаемого RC-генератора значительно ниже, чем при **IC** и **PRECHARGE**. Причем она настолько ниже, что вообще не стыкуется с расчетными данными, полученными по формулам из справочников по цифровым устройствам. Поэтому от этой затеи я отказался. Попытка втащить свойство **IC** внутрь модели тоже была обречена на провал. Наличие IC предусматривает наличие аналоговой цепи внутри модели – хоть пресловутый резистор на 1 миллиОм, а надо врезать в эту цепь. При этом сразу падает быстродействие модели, а оно у нас и так невелико – при задирании частоты внутреннего

генератора выше 600кГц или 1,2 -1,3 кГц с RC мой Core Quad с 2500ГГц тактовой и 4(3,2) Гбайт мозгов благополучно грузится на все 100% и начинает бросаться «китайскими парашютистами» в логе. Ну а про **PRECHARGE** вообще можно забыть – конденсатор у нас снаружи. Но все-же покажу, как сделать со свойством **SCHMITT** на примере **Graphic_with_Child_SCHMITT.DSN**, лежащем в папке **Graphic_and_Hierarchy**. Для элемента **U2** в свойствах вписываем **SCHMITT=< SCHMITT >**, а скрипт для генератора дописываем следующим образом:

```
*MAP ON CLOCK
DEFAULT : CLKOSC=DIGITAL, SCHMITT=NULL
EXTERNAL : CLKOSC=NULL, SCHMITT=D
```

Вот и вся модификация. Если кому то нравится этот вариант, то скомпилируйте **176IE12.MDF** с дочернего листа модуля из проекта **Graphic_with_Child_SCHMITT.DSN** самостоятельно. Я там уже заранее поставил и задержки, которые мы использовали в конечном варианте. Можно даже сразу откомпилировать в папку **MODELS** Протеуса. Она при старте компилятора предлагается автоматически.
Еще, учитывая наличие второй Российской достопримечательности после дорог, можно создать и скомпилировать (сторонней программой, а не Протеусом!!!) файл помощи типа **xxx.HLP**. Файл помещаем в папку **HELP** Протеуса, а на четвертой вкладке **Make Device** (которую пока мы просто проскакиваем) в графе **Help File** указываем путь к файлу помощи. Возможно, я так и поступлю, но сделаю единый **xxx.HELP** для всей 176-й серии позже (да проклянут меня владельцы Windows 7!!!).
Ну и последнее замечание. Если все же до конца покривить душой и оставить задержки как у модели 4060, то модель 176ИЕ12 будет меньше грузить ЦП компьютера, так что это тоже вариант модификации. А мы далее переходим к созданию счетчиков неизменно сопутствующих применению **К176ИЕ12** в часах.

Возврат к содержанию

6.10. Структура модели счетчика 4026 – основы для будущей К176ИЕ4. Полезные сведения о примитивах счетчиков и дешифраторов в ISIS.

В 176-й серии микросхем есть два счетчика-дешифратора неизменно сопутствующих **К176ИЕ12** при создании схем электронных часов с сегментными индикаторами. Это декадный счетчик **К176ИЕ4** с выходами в коде семисегментного индикатора и **К176ИЕ3** – аналогичный предыдущему счетчик с коэффициентом пересчета 6. Почему-то в таблицах аналогов микросхем нашему **К176ИЕ4** принято подставлять в аналог импортный **CD4026**. Но редко кто при этом делает ссылку на то, что аналог неполный. Я бы рискнул подчеркнуть, что аналогия наблюдается только в функции десятичного пересчета и выходов в семисегментном коде и не более того. Но, тем не менее, воспользовавшись моделью для **4026** как основой, можно попробовать создать **176ИЕ4**. Оставим пока в покое наши счетчики, и попробуем посмотреть, как выглядит модель **4026**. Файл **4026.MDF** извлекается аналогично модели **4060** и расположен в той же библиотеке **DIGITAL.LML**. Я прикреплю его к вложению для тех, кому лень лишний раз поработать с **GETMDF**. Там же вы найдете даташит на **CD4026B** от фирмы Texas Instruments. Он самый маленький по размеру из имеющихся у меня в наличии, поэтому не сильно увеличит объем вложения. А вот внутреннюю структуру на Рис. 91 я взял от **HCF4026** – этот даташит с самыми четкими картинками, но и весит в три раза больше.

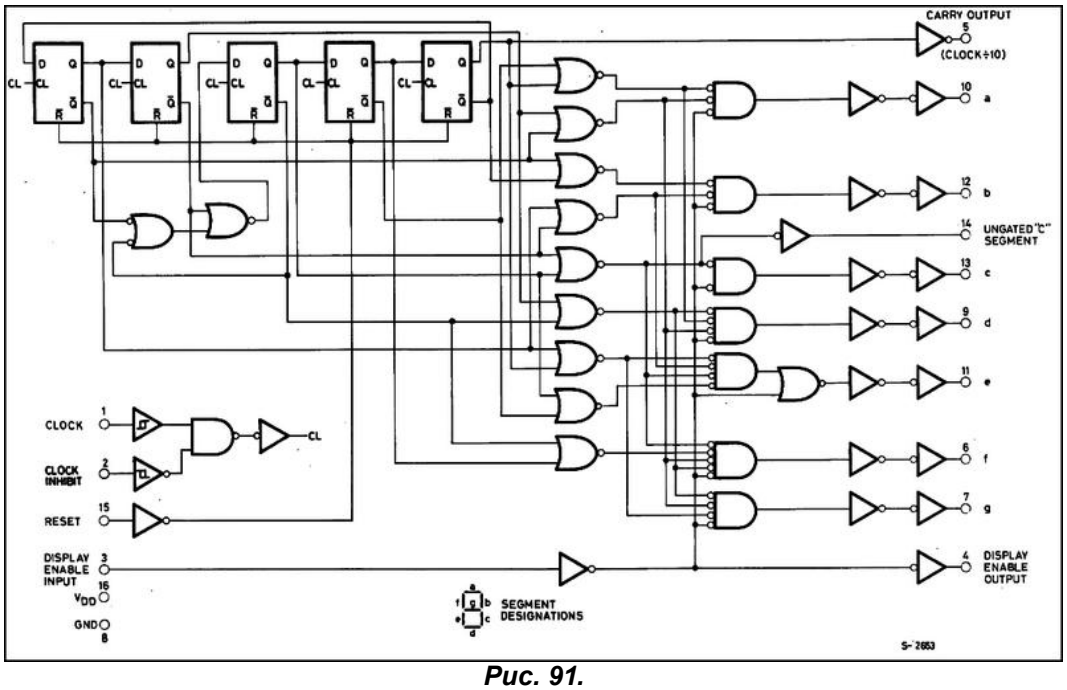


Рис. 91.

Если проанализировать структуру счетчика, то можно сделать вывод, что это пятиразрядный счетчик Джонсона на D-триггерах с дешифрацией кода в семисегментный на логических элементах. В принципе, мы могли бы, как и в случае с **4060** воспроизвести эту структуру на цифровых примитивах и получить вполне нормальную, рабочую модель счетчика. Но давайте заглянем в **4026.MDF**. Ниже приведен раздел, содержащий список компонентов модели:

```
*PARTLIST,4
U1,COUNTER_4,COUNTER_4,ALOAD=0,ARESET=1,INVERT="OE,CE,MAX",LOWER=0,PRIMITIVE=DIGITAL,UPPER=9,USEDIR=0
U2,DECODER_4_7,DECODER_4_7,PRIMITIVE=DIGITAL,TG=<TG>,TYPE=7B
U3,OR_4,OR_4,INVERT=D2,PRIMITIVE=DIGITAL,TG=<TG>
U4,BUFFER,BUFFER,PRIMITIVE=DIGITAL,TG=<TG>
```

Приятная неожиданность, не правда ли? Весь список – 4 примитива. Ни триггеров, ни многовходовых логических элементов. Остается констатировать факт – в данном случае применена поведенческая модель, т.е. функции реального компонента имитированы с помощью подходящих по назначению цифровых примитивов, путем задания им специфических свойств. Как я и обещал, в процессе создания моделей мы познакомимся с цифровыми примитивами поближе. В данном случае присутствуют два заслуживающих особого внимания компонента. Это **COUNTER_4** и **DECODER_4_7**. Логические элементы, надеюсь, не вызывают каких-либо затруднений в использовании. Кстати, изучив специфические свойства четырехразрядного счетчика и декодера из 4 разрядов в семисегментный код, мы сможем без труда использовать и остальные примитивы счетчиков и дешифраторов. Итак, приступим.

Четырехразрядный универсальный счетчик **COUNTER_4** с обвеской для тестирования представлен на Рис. 92, а тестовый проект в папке **COUNTER_TEST** вложения. Для визуальной индикации состояния его выходов я использовал модель семисегментного индикатора **7SEG-BCD**,

позволяющую в цифровом виде просматривать двоичный код. Обратите внимание, что у этой модели вывод младшего разряда находится справа. Имена выводов не индицируются, поэтому просто запомните на будущее.

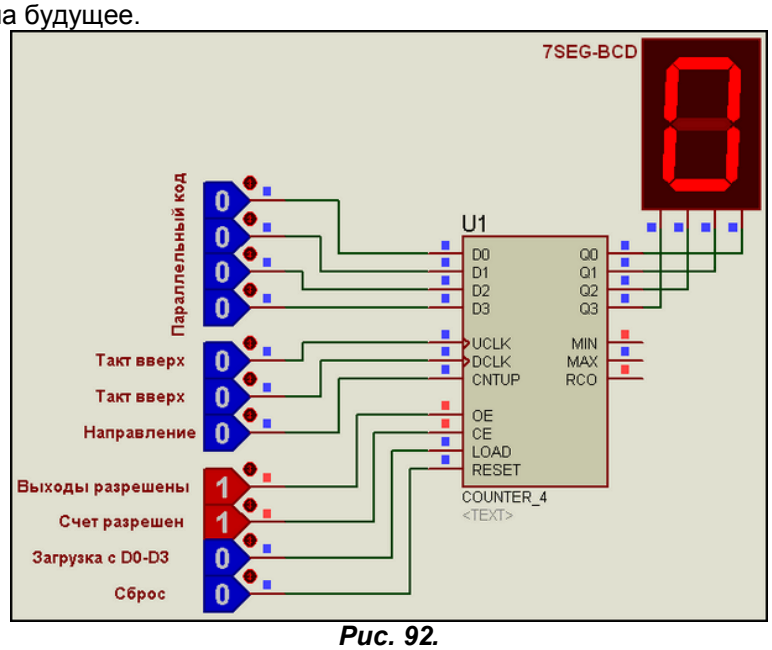


Рис. 92.

Для начала познакомимся с назначением выводов (пинов) модели.
Входы:
D0…D4 – (**Load data**) входы данных параллельной загрузки счетчика.
UCLK – (**Count-up clock**) счетный вход инкремента (приращения) по положительному фронту импульса.
DCLK – (**Count-down clock**) счетный вход декремента (уменьшения) по положительному фронту импульса.
CNTUP – (**Count up/down direction control**) вход направления счета. По умолчанию отключен. Подключается флажком в свойствах: лог.0 – счет вниз, лог.1 – счет вверх.
OE – (**Output-enable control**) разрешение выходов. Лог.1 разрешает сигнал на выходах Q0…Q4.
CE – (**Count-enable control**) вход разрешения счета. Лог.1 разрешает счет.
LOAD – (**Load input**) вход разрешения параллельной загрузки. По умолчанию синхронный, т.е. при установке в лог. 1 загрузка с входов D0…D4 осуществляется по переднему фронту UCLK или DCLK. При установке в асинхронный режим флажком в свойствах загрузка осущесвляется по переднему фронту импульса на нем независимо от счетных входов.
RESET – (**Reset input**) сброс счетчика по умолчанию синхронный по положительному фронту UCLK или DCLK. Перевод в асинхронный режим флажком в свойствах.
Выходы:
Q0…Q4 – (**Count output**) выходы счетчика.
MIN – (**Minimum count output**) выход минимального значения. Сигнал на нем формируется при направлении счета вниз и установленном значении **Minimum count value** в **Advanced** свойствах.
MAX – (**Maximum count output**) выход максимального значения. Сигнал на нем формируется при направлении счета вниз и установленном значении **Maximum count value** в **Advanced** свойствах.
RCO – (**Ripple-carry output**) выход достижения верхнего или нижнего порога счета. Сигнал на нем формируется как по верхнему **Minimum count value**, так и по нижнему **Maximum count value** пределам счета.
Окно свойств счетчика представлено на Рис. 93.

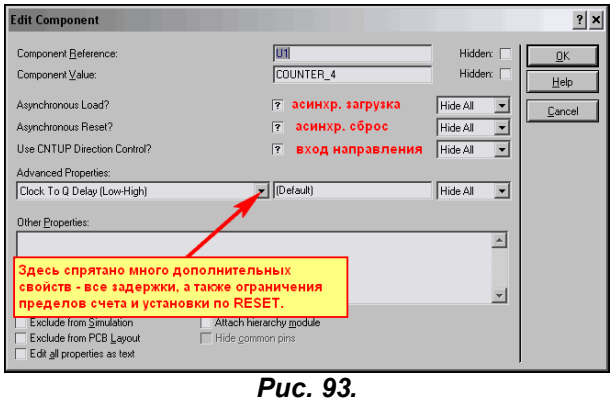


Рис. 93.

Из основного окна свойств непосредственно доступны только три флажка: **Asynchronous Load** (**ALOAD**), **Asynchronous Reset** (**ARESET**) и **Use CNTUP Direction Control** (**USEDIR**). О них я упоминал в описании выводов модели. По умолчанию они не определены – **None** (символ знака вопроса). Для установки необходимо добиться щелчком мыши галочки в окошке, для снятия – пустого окна. Все остальные свойства спрятаны в раскрывающемся списке **Advanced Properties**. В основном это временные свойства задержек. Я не думаю, что стоит приводить здесь весь список времянок с переводом Протеусной «фени» на наш великий и могучий. Как расшифровывать эту временную абракадабру я уже описывал, ничего принципиально нового здесь нет. Остановлюсь только на трех параметрах, которые нам будут интересны в данный момент. Все они сосредоточены в конце раскрывающегося списка и по умолчанию не определены, т.е. **None**.

Lower Count Value (**LOWER**) – нижнее значение предела счетчика.

Upper Count Value (**UPPER**) – верхнее значение предела счетчика.

Output Value On Reset (**RESET**) – состояние выходов счетчика после воздействия сигнала сброса по входу RESET.

Все эти три значения задаются десятичными числами. Следует помнить, что исходное состояние выходов 0000 – тоже значимое состояние. Поэтому, при задании пределов для счетчиков-делителей его необходимо учитывать и задавать предел на единицу меньше. Например, для декадного счетчика с коэффициентом пересчета 10 необходимо задать верхний предел **UPPER=9**. При этом **LOWER** и **RESET** можно и не задавать, если счетчик только суммирующий. Однако если предполагается реверсивный счет, то лучше его обозначить. Если вы хотите получить реверсивный счетчик с одним тактовым входом, то просто объедините **UCLK** и **DCLK**, включите флажок **USEDIR** и управляйте направлением счета с помощью входа **CNTUP**. Интересно назначение последнего из трех вышеуказанных свойств – **RESET**. Если задать ему ненулевое числовое значение ниже верхнего предела, то при воздействии входного сигнала **RESET** счетчик будет сбрасываться не в ноль, а в это значение, при условии, что в данный момент значение на выходах больше данного числа. Для предустановки счетчика при запуске симуляции как и у триггеров, используется свойство **INIT**, введенное вручную в окне дополнительных свойств. Если вы введете **INIT= 3**, то при старте симуляции встанут в лог. 1 выходы **Q0** и **Q1** (двоичное 0011). Также как и с другими цифровыми примитивами здесь применимо свойство **INVERT** как к входам, так и к выходам. Например, по умолчанию счет происходит по переднему фронту тактового импульса на **UCLK**. Если ввести **INVERT=UCLCK**, то счетчик будет перекидываться по заднему фронту этого импульса. В INVERT можно записывать одновременно несколько выводов модели, разделяя их имена запятыми без дополнительных пробелов. Например: **INVERT=D0,D1,D2,D3,RESET** проинвертирует сигналы на всех входах данных и входе сброса, т.е. счетчик сбрасываться будет лог. 0.

Как вы уже, наверное, догадались из рисунка 92, чтобы активировать модель счетчика – получить сигналы на выходах и разрешить счет необходимо подать на **OE** и **CE** сигналы лог. 1. Чтобы вы могли самостоятельно изучить возможности универсальной модели счетчика, я приложил два проекта **Manual_TEST.DSN** и **Misc_TEST.DSN** в папке **COUNTER_TEST**. В первом из них тестирование с различными предустановками осуществляется путем ручного изменения состояния логических переключателей на входах счетчиков. Во втором на входы поданы сигналы генераторов, тестирование по частотомерам в интерактивном режиме и с помощью графиков.

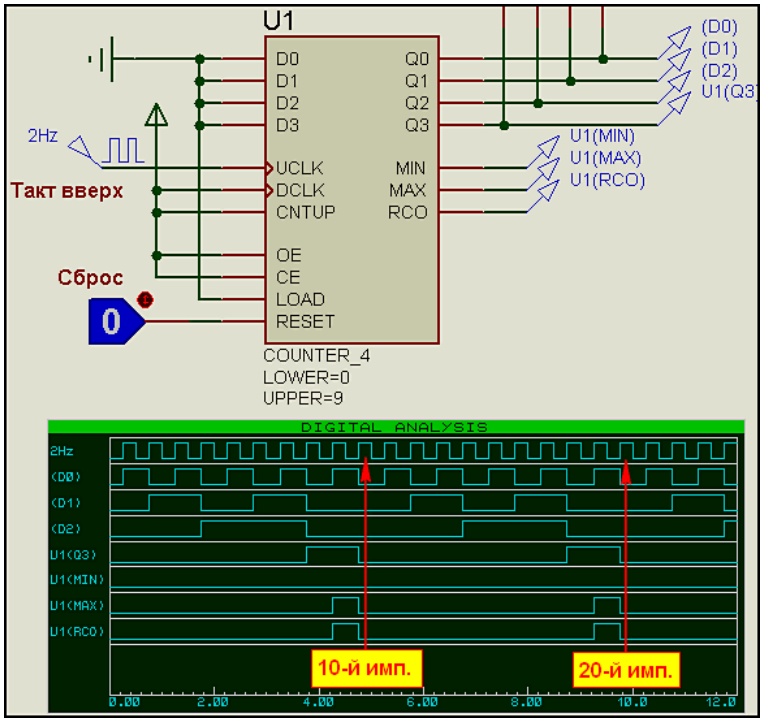


Рис. 94.

Мы тестировали четырехразрядную модель универсального счетчика, но в арсенале цифровых примитивов имеются различные варианты от трех до 14 разрядов. Свойства у всех у них аналогичны рассмотренному нами.
Следующий не менее интересный примитив дешифратор **DECODER_4_7**. Свойств у него будет поменьше, но, тем не менее, знание особенностей этой модели позволит нам успешно применять ее и другие дешифраторы для своих целей.
Вид декодера включенного в двоичный режим с обвеской для ручного теста показан на Рис. 95.

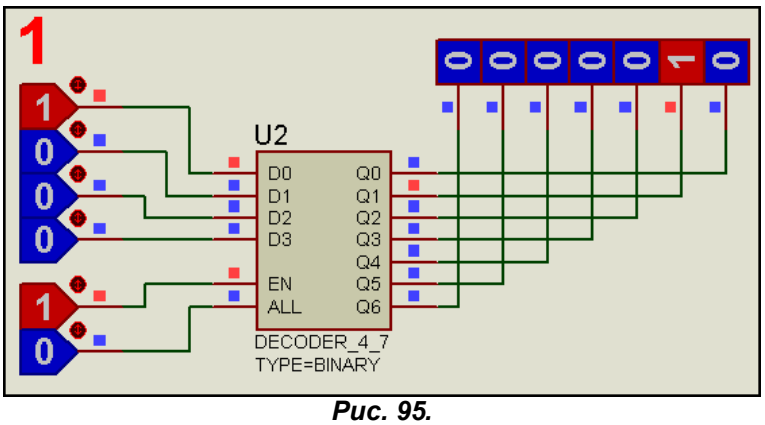


Рис. 95.

Назначение выводов модели.
Входы:
D0…D4 – (**Data input to be decoded**) входы данных для декодирования.
EN – (**Enable**) сигнал на выходах разрешен.
ALL – (**All outputs active**) се выходы в активное состояние в соответствии с выбранным в окне свойств уровнем LOW или HIGH для **All sets all outputs** (ALL). По умолчанию не выбрано – **None**.
Выходы:
Q0…Q6 – (**Decoded output value**) декодированное выходное значение.
Теперь рассмотрим окно свойств (Рис. 96).

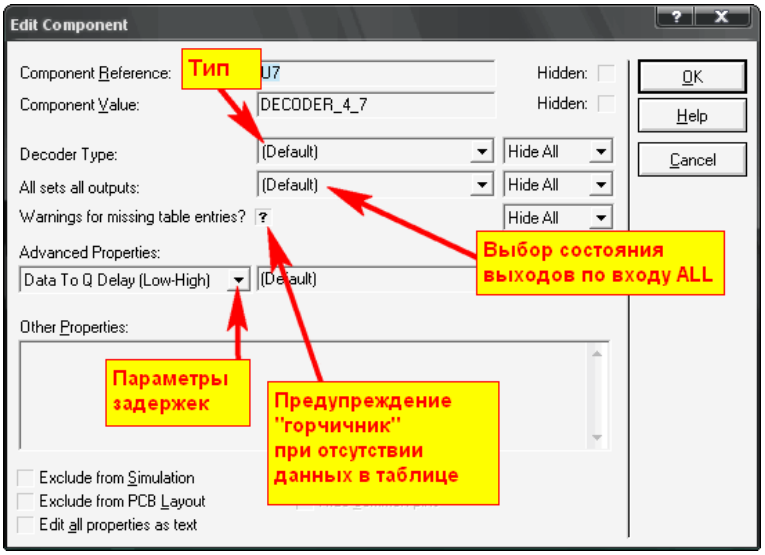


Рис. 96.

Decoder Type (**TYPE**) – тип декодированного сигнала. На этом свойстве остановимся особо чуть ниже.
All sets all outputs (**ALL**) – установка всех выходов по сигналу на входе **ALL**. Описано выше.
Warnings for missing table entries (**WARN**) – желтое предупреждение в логе в случае отсутствия данных в таблице для присутствующей на входах кодовой комбинации.
В раскрывающемся списке **Advanced Properties** прописаны все типы задержек, присущие данной модели. На них я также как и со счетчиком останавливаться не буду.
При наличии на входе **EN** высокого логического уровня данные на выходах модели устанавливаются в соответствии с двоичной кодовой комбинацией на входах и типом дешифратора, выбранным в окне свойств. Вот на этом моменте остановимся подробнее.
Возможны пять типов дешифрации входного сигнала, выбираемых из раскрывающегося списка:
BINARY – в этом режиме двоичному коду на входах **D0…D4** модели соответствует единственный активный выход в соответствии с заданным входным значением, т.е. **0000** активирует выход **D0**, **0001** – выход **D1** (на Рис. 95) … **1001** – выход **D9** … 1111 – выход **D15**. Конечно, последние два из перечисленных вариантов в нашем случае отсутствуют, но имеются, например, в модели **DECODER_4_16**.

BCD – двоично-десятичный декодер. Этот вариант до цифры 9 – код **1001** полностью совпадает с двоичным, однако последующие комбинации четырехразрядного кода не рассматриваются, и состояния выходов не меняют. Это вытекает из концепции BCD кодирования, где каждый разряд десятичного числа описывается двоичной тетрадой, т.е. числу **123** будет соответствовать в коде BCD кодовая комбинация **0001 0010 0011** (сотни-десятки-единицы).
Две кодовых комбинации для семисегментного кода – **7A** и **7B**. Варианты отображения сегментов для кода на входах представлены на Рис. 97 и всегда доступны в файле помощи по кнопке **HELP** в окне свойств. Если кто-то затрудняется найти отличие – сравните цифры 6 и 9.

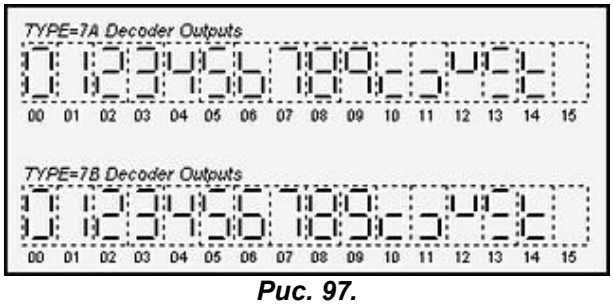


Рис. 97.

Ну и, наконец, пятый, самый интересный тип **TABLE** – дающий нам возможность проявить собственные фантазии. Этот тип туманно описан в **HELP**, поэтому я восполню пробелы в информации. Как и следует из перевода, это табличный тип представления соответствия числа на входе состоянию выходов. С ним неразрывно связано свойство **LENGTH** (длина таблицы), которое вручную надо задать в окне свойств, выбрав этого тип дешифратора. В качестве примера использования этого типа в папке **DECODER** вложения приложен проект TABLE.DSN, в котором с помощью таблицы я задал декодирование семисегментного кода, где комбинациям с 10 по 15 соответствуют выходные сигналы мнемоники букв **A, b, C, d, E, F** для индикатора с общим катодом. Ниже приведен код, заданный в свойствах:

```
LENGTH=16
TABLE0=%0111111
TABLE1=%0000110
TABLE2=%1011011
TABLE3=%1001111
TABLE4=%1100110
TABLE5=%1101101
TABLE6=%1111101
TABLE7=%0000111
TABLE8=%1111111
TABLE9=%1101111
TABLE10=%1110111
TABLE11=%1111100
TABLE12=%0111001
TABLE13=%1011110
TABLE14=%1111001
TABLE15=%1110001
```

Рассмотрим, как он сформирован. **LENGTH=16** задает количество строк таблицы, включая **TABLE0**. Далее, в каждой строке, начинающейся со слова **TABLE**, назначен двоичный код на выходах **Q** соответствующий десятичному значению после слова **TABLE** на входах **D**. Знак процента после равенства означает, что далее следует двоичное число. В двоичном значении, как и обычно, младший разряд справа, т.е. в нашем случае для шести выходов они расположены как **Q6,Q5…Q0**. Используя другие типы примитивов дешифраторов с меньшим или большим числом входов/выходов, можно таблично задать любую задуманную заранее комбинацию. Обращаю только внимание на то, что заданная длина таблицы и количество знаков в двоичном числе должны совпадать соответственно с числом строк и числом выходов модели, иначе получите «горчичник» в логе. Различные примеры для тестирования модели дешифратора приложены в проектах **BIN_BCD.DSN** и **7A_7B.DSN** папки **DECODER** вложения. Из названий видно, какие типы, где тестируются. Для вариантов **7A** и **7B** я показал, как с помощью свойства **INVERT** для выходов можно использовать индикаторы с общим катодом и общим анодом.
На этом пока закончим знакомство с примитивами и вернемся к нашей модели **4026**. В папке **CD4026_model** я привел проект, в котором по файлу **MDF** реконструирована внутренняя структура модели счетчика. На рисунке 98 она приведена полностью. Наверное, теперь, когда ясны те или иные назначения свойств использованных примитивов, излишние комментарии по поводу работы этой структуры становятся неуместными. Но все-же немного поясню, чтобы снять ненужные вопросы. Посредством назначения верхнего и нижнего пределов и запрещения параллельной загрузки примитив **U1** превращен в обычный десятичный счетчик. Посредством **INVERT** изменен активный уровень по входу **CE**, чтобы совпадал с реальным. Все ненужное завешено на **VSS**, т.е. на нулевой вывод питания, причем вход **OE** для этого проинвертирован, чтобы активным разрешением по нему стал лог. 0. Проинвертирован также и выход **MAX**, чтобы создать перепад с нуля на

единицу на выходе **CO** по достижении десятого импульса. Кстати, здесь есть небольшое расхождение с реальной микросхемой по временному формированию этого сигнала не влияющее на работу при последовательном соединении счетчиков. На элементе **U2** реализовано декодирование в сигнал семисегментного кода по таблице 7В (шестерка и девятка с «хвостиками»). На четырехвходовом ИЛИ - элементе **U3** реализовано формирование импульса по счету 2. Здесь также применена инверсия к третьему сверху входу. Ну и цифровой буфер **U4** применен чисто для «гальванического» разделения входа **DEI** и выхода **DEO**.

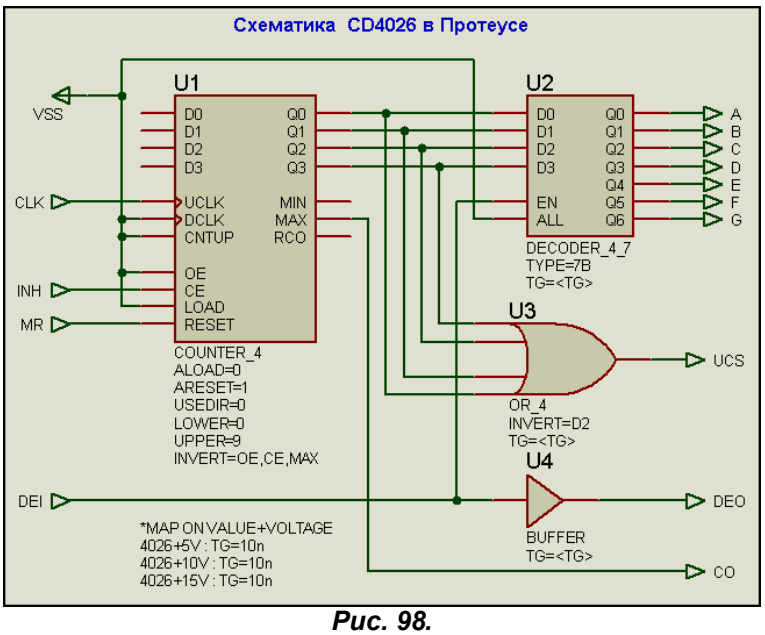


Рис. 98.

Вот эту структуру мы и возьмем за основу для формирования моделей наших **К176ИЕ3** и **К176ИЕ4**, прибегнув к некоторым корректировкам в сторону их особенностей. Об этом в следующем материале.

Возврат к содержанию

6.11. Поведенческие модели К176ИЕ4 и К176ИЕ3 для Протеуса на основе примитивов универсальных счетчиков.

Для начала сформируем графические модели этих счетчиков в соответствии с Рис. 99.

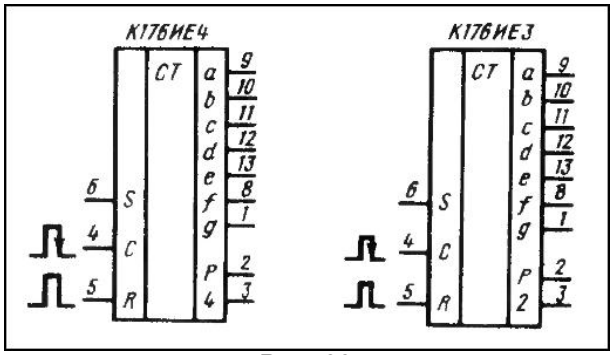


Рис. 99.

Надеюсь, что мне не стоит в очередной раз повторять процесс создания графики, я просто приложил проект **Graphic_model.DSN**, в котором есть и несформированная в модели графика и готовые модели. Обратите внимание, что моделям назначен корпус **DIL14** в отличие от шестнадцативыводной **К176ИЕ12**. Базовая графика показана на рисунке 100.

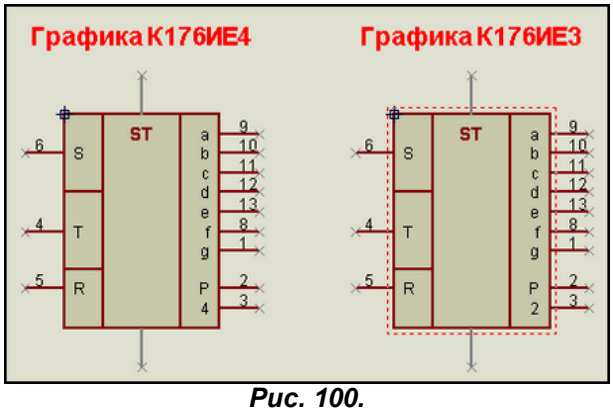


Рис. 100.

Обратите внимание, что вместо стандартного обозначения счетного вывода я применил к нему символ “**T**”. Протеус не приучен различать прописные и строчные буквы, а латинским символом “**c**” у нас уже обозначен третий выход на семисегментный индикатор. Поэтому пришлось пойти на такую замену. Порывшись в старых справочниках, в книге В. Л. Шило «Популярные цифровые микросхемы» М., Радио и связь, 1989 мне удалось найти временные диаграммы формирования сигналов на выходах **P** и **4**(**2**) этих микросхем. Диаграмма приведена на Рис. 101. Соответственно на ней **U2** – выходной сигнал на выводе 2 (**P**), а **U3** – выходной сигнал на выводах 3 счетчиков.

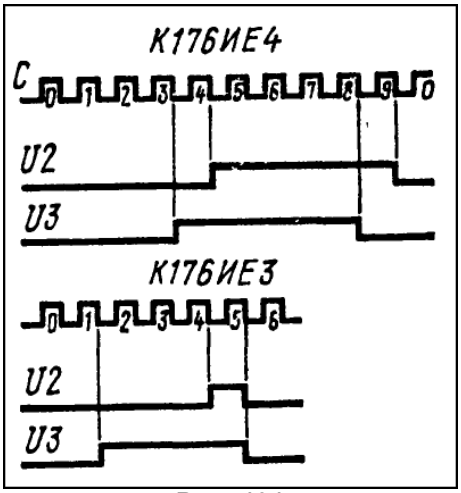


Рис. 101.

Приведу еще несколько выдержек из различных справочных материалов, посвященных этим счетчикам, которые пригодятся нам в дальнейшем. Установка триггеров счетчиков в 0 происходит подачей лог. 1 на вход **R**, переключение (счет) по спаду импульса положительной полярности на входе **C** (в моем случае **T**). Сигнал на входе **S** служит для изменения полярности выходных семисегментных кодовых комбинаций. В случае подачи на вход **S** положительного сигнала лог. 1 получаем на выходе код для индикаторов с общим анодом, а в случае лог. 0 – код для индикаторов с общим катодом. При использовании вакуумно-люминисцентных индикаторов на этот вход подается модулирующие импульсы, например 32 кГц счетчиков **К176ИЕ5** или **К176ИЕ12**. При использовании жидкокристалических индикаторов также подаются модулирующие импульсы, но с частотой на порядок ниже 30-100Гц, которые одновременно подаются и на подложку индикатора. Импульсы на выходах **P** и **4** (**2**) данных счетчиков позволяют при последовательном соединении организовать разряды индикации минут от 0 до 59 и часов от 0 до 23. Спады положительных импульсов на соответствующих выходах служат в качестве тактовых для переключения старших разрядов. Мы рассмотрим в дальнейшем организацию структуры электронных часов с использованием данных счетчиков. А пока, отталкиваясь от этих данных и приняв структуру **4026** как базовую, попробуем создать на ее основе модели данных счетчиков.

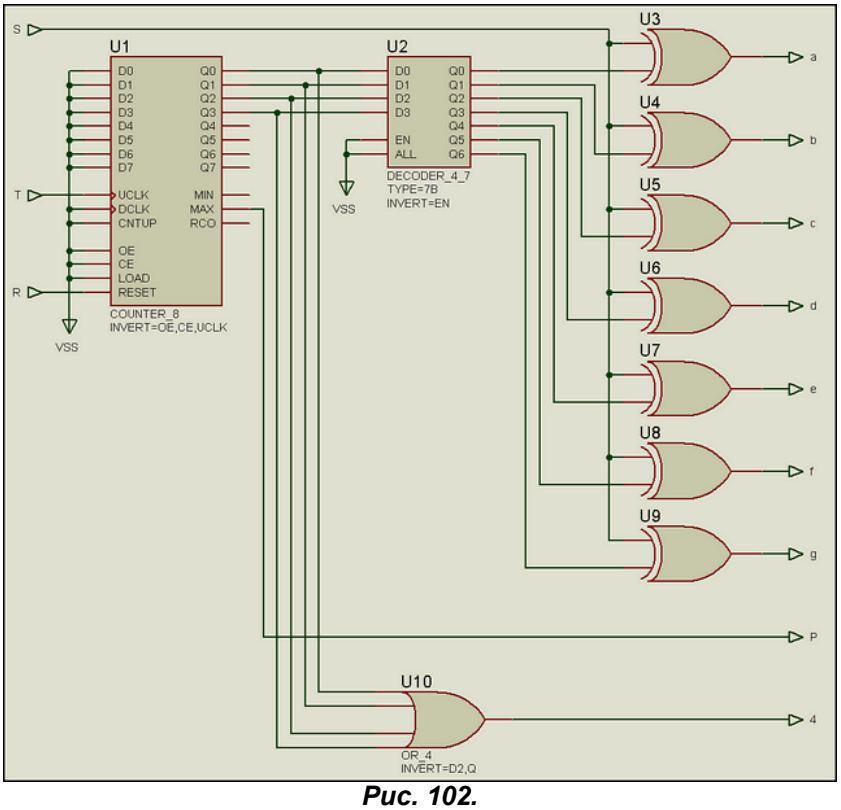


Рис. 102.

Основным отличием **К176ИЕ4** от **CD4026** является наличие входа **S**, позволяющего инвертировать выходной код. Достаточно добавить к исходной структуре дополнительную логику, реализующую данную функцию и модель приобретет все необходимые нам свойства. Проще всего реализовать этот момент с помощью двухвходовых элементов **XOR** (исключающее ИЛИ). Кроме того, нам потребуется изменить полярность сигналов на выходах **P** и **4**. Это легко делается с помощью свойства **INVERT**. Получившаяся в результате данных действий структура показана на Рис. 102.
В отличие от модели **4026** в данном случае не инвертируется сигнал **MAX** со счетчика **U1**, но зато инвертируется выход **D** элемента 4ИЛИ **U10**. На выходе дешифратора включены элементы исключающее ИЛИ, которые с помощью изменения полярности сигнала на входе **S** позволяют инвертировать выходной код. На Рис. 103 приведена временная диаграмма для данного варианта.

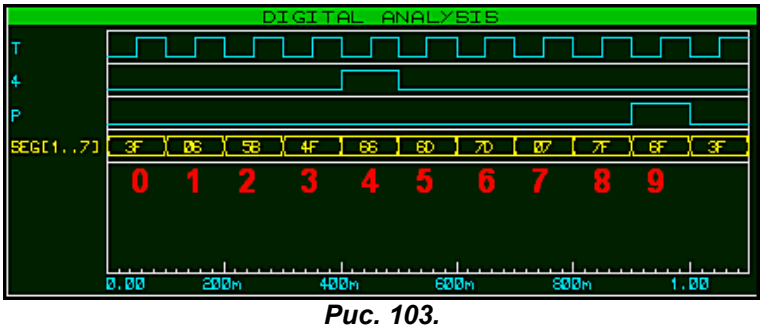


Рис. 103.

Для лучшего восприятия диаграммы я включил в нее в виде семиразрядной шины сигнал на выходах **a…g**. В данном случае на входе **S** присутствует лог. 0 и код шины соответствует сигналам для индикатора с общим катодом. Все отличие от **4026** в инверсии сигналов на выходах **P** и **4**. Данный вариант модели имеет право на существование, и даже будет корректно работать, поскольку нужные перепады уровней с лог. 0 на лог. 1 после цифры 3 и с лог. 1 на лог. 0 после цифры 9 совпадают во времени с приведенными на диаграмме Рис. 101 сигналами для счетчика **К176ИЕ4**. Но вот длительности этих импульсов оставляют желать…. Попробуем с помощью дополнительных элементов устранить данное несоответствие. Я не стал сильно мудрить и посредством двух дополнительных элементов 2И и триггеров достиг желаемого (Рис. 104).

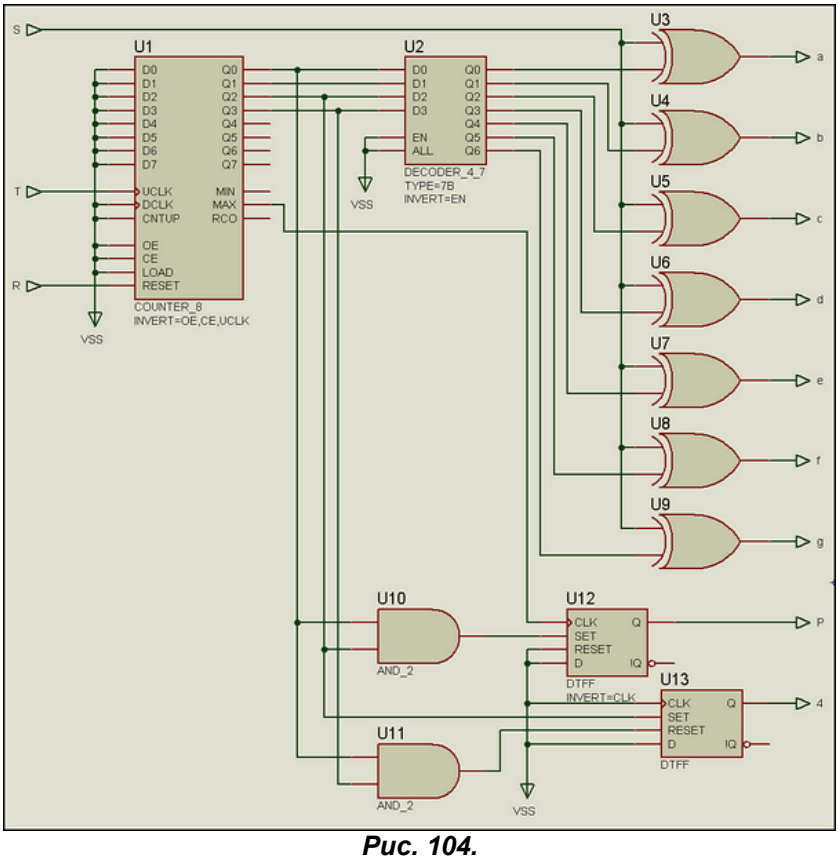


Рис. 104.

В результате проведенных коррекций картинка на диаграмме (Рис. 105) стала полностью совпадать с приведенной ранее на рисунке 101. Данные примеры приложены в папке вложения **176IE4_structure**. Проект **IE4_var_1.DSN** – с укороченными импульсами и проект **IE4_var_2.DSN** с дополнительными триггерами и импульсами соответствующими диаграмме рисунка 105. Аналогичные примеры для счетчика **К176ИЕ3** приведены в папке **176IE3_structure**. Они отличаются только логикой формирования импульсов на выходах **P** и **2**, поскольку счетчик имеет коэффициент пересчета 6. Я не стану здесь приводить лишние картинки, желающие посмотрят их в соответствующих проектах.

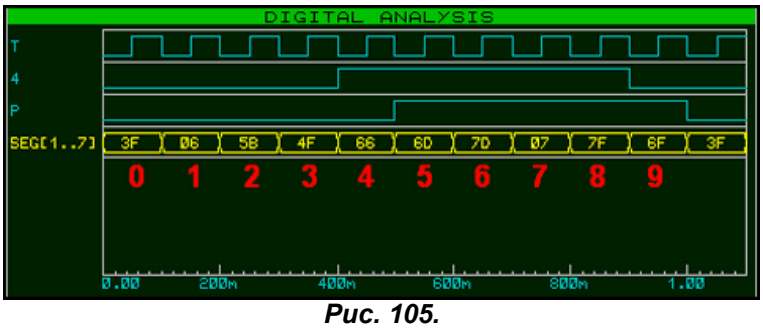


Рис. 105.

В проектах **Test_176IE3_IE4_V1.DSN** и **Test_176IE3_IE4_V2.DSN** данные структуры присоединены в виде дочерних листов к соответствующим графическим моделям, и из них построена схема счетных секций минут и часов для проверки моделей в действии. Папка с проектами называется **Test_counters**.

Но готовых файлов MDF во вложении вы не найдете. Дело в том, что в процессе построения моделей этих счетчиков меня слегка «торкнуло» и я нашел более красивое и близкое к реальности решение для этих моделей. Заодно «пристрелю еще одного ушастого» - познакомлю вас с цифровым примитивом универсального регистра сдвига. Вот из тех моделей мы и сформируем MDF для последующего использования.

Возврат к содержанию

6.12. SHIFTREG - примитив универсального регистра сдвига и модели К176ИЕ4 и К176ИЕ3 для Протеуса на его основе.

Если вспомнить описание внутренней структуры счетчика **4026**, то изначально я упоминал, что он построен на базе пятиразрядного счетчика Джонсона. Не являются исключением и наши **К176ИЕ3**, **К176ИЕ4**. Типично счетчик Джонсона выглядит как замкнутый в кольцо сдвиговый регистр, у которого инверсный выход последнего триггера соединен с входом **D** первого. Давайте попробуем построить максимально приближенную к реальности модель на базе универсального сдвигового регистра. Для начала познакомимся с примитивом **SHIFTREG** и его свойствами. Поскольку нам потребуется пятиразрядный регистр сдвига, я буду рассматривать именно **SHIFTREG_5**, у остальных свойства те же. Для ручного тестирования я обвешал примитив переключателями и пробниками из **Debugging Tools** (Рис. 106) и поместил в проект **ShiftReg5_Manual_test.DSN** в папке **SHIFTREG_TEST** вложения.

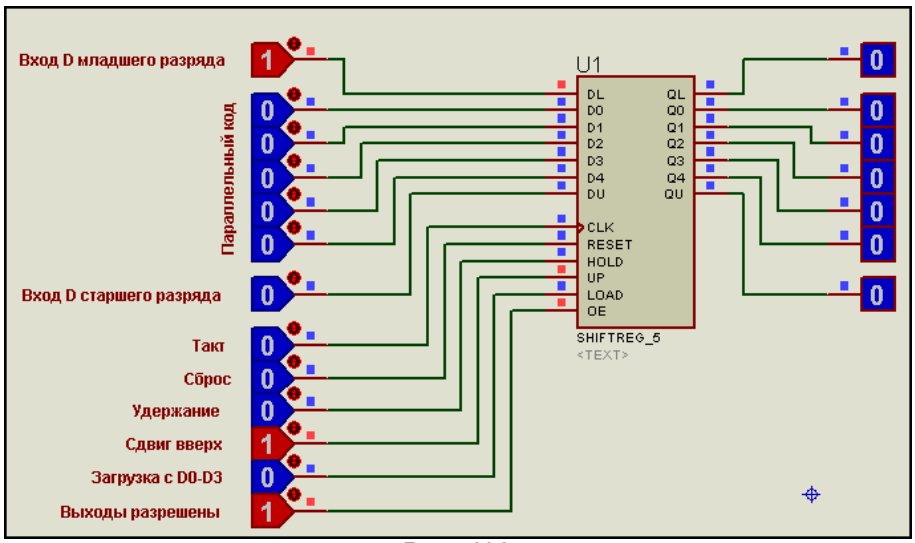


Рис. 106.

Для начала познакомимся с назначением входов и выходов примитива.
Входы:
DL – **(New lower data)** вход данных младшего разряда регистра, по сути вход D триггера младшего разряда.
D0…D4 – (**Parallel load data**) входы данных для параллельной загрузки.
DU – **(New upper data)** вход данных старшего разряда регистра, по сути вход D триггера старшего разряда.
CLK – (**Clock**) тактовый вход. По умолчанию сдвиг (а также загрузка и сброс, если не установлены флажки в свойствах) происходит по переднему фронту импульса на нем.
RESET – (**Data reset**) сброс данных. По умолчанию при лог. 1 на этом входе по переднему фронту CLK. Для асинхронного сброса только этим сигналом необходимо установить флажок в свойствах (Рис. 107).
HOLD – (**Shift-hold**) удержание данных. Лог. 1 на этом входе запрещает сдвиг данных по тактовому сигналу на входе CLK.
UP – (**Direction control**) направление сдвига. При лог. 1 на этом входе сдвиг происходит с входа **DL** по направлению вверх, т.е. **Q0=>Q1…Q4**. При лог. 0 на этом входе сдвиг происходит с входа **DU** по направлению вниз, т.е. **Q4=>Q3…Q0**.
LOAD – (**Data load**) сигнал параллельной загрузки со входов **D0…D4**. По умолчанию при лог. 1 на этом входе по переднему фронту CLK. Для асинхронной загрузки только этим сигналом необходимо установить флажок в свойствах (Рис. 107).
OE – (**Output-enable**) сигнал, разрешающий работу выходов данных **Q0…Q4**. Обратите внимание, что сигналы на выходах переноса **QL** и **QU** он не блокирует.
Выходы:
QL и **QU** – (**Lower Q and Upper Q**) выходы соответственно переноса вниз и вверх. Фактически сигнал на этих выходах дублирует соответственно Q0 и Q4, но как бы гальванически развязан от них.
Q0…Q4 – (**Data output**) выходы данных соответствующих разрядов регистра.
Теперь обратимся к окну свойств **SHIFTREG** (Рис. 107). По сути, я уже почти все описал, пока рассматривал назначение выводов. Осталось указать, что **Initial Output Value** (**INIT**) определяет состояние триггеров регистра при старте симуляции. По умолчанию оно не определено, но если ввести в него десятичное значение, например 3, то при старте симуляции соответственно выходы (триггеры) **Q0** и **Q1** встанут в состояние лог.1. Фактически, у счетчиков было то же самое. Ну и

конечно по раскрывающемуся списку **Advanced Properties** запрятаны все возможные задержки распространения сигнала для данного примитива.

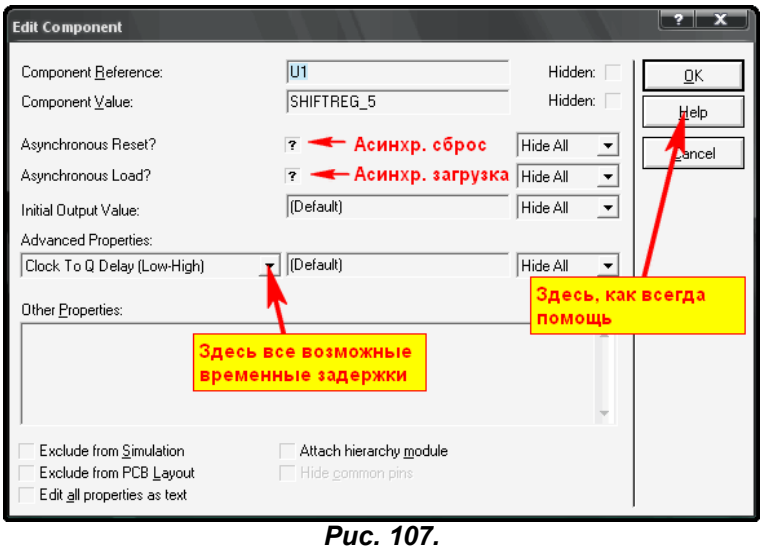


Рис. 107.

В папке **SHIFTREG_TEST** имеется проект **ShiftReg5_auto_test.DSN**, в котором приведены характерные комбинации сигналов и свойств для применения примитива в качестве сдвигового регистра. Последний пример в этом проекте показывает организацию счетчика Джонсона, который нам необходим для создания моделей наших счетчиков. На рисунке 108 приведена диаграмма его работы. Мы видим, что полный цикл от состояния от **00000** до **11111** счетчик проходит за 10 тактовых импульсов.

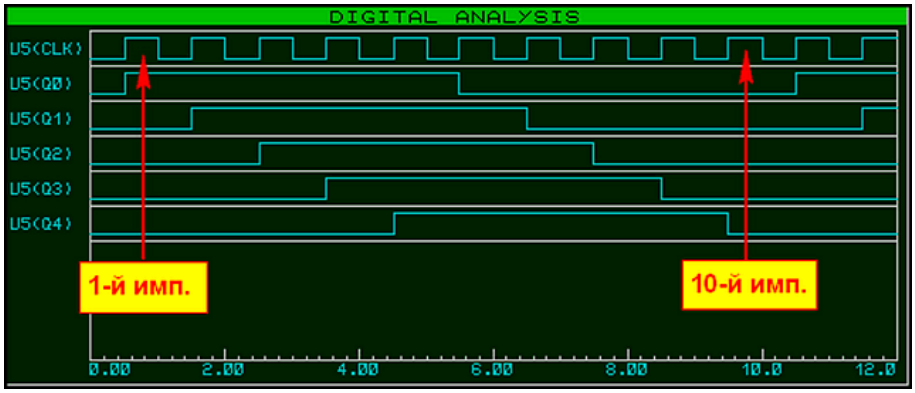


Рис. 108.

Фактически, это то, что нам нужно для построения моделей. Достаточно проинвертировать тактовый вход, чтобы срабатывание происходило по спаду тактового импульса и половина модели готова. Причем импульсы на выходах переполнения **P** (выход **Q4**) и **4** (выход **Q3**) уже будут иметь необходимую длительность. Если кому интересен первоисточник информации, рекомендую обратиться к книге: Ковалев В. Г. Лебедев О. Н. «Электронные часы на микросхемах» М.: Радио и связь, 1985. Там построение счетчиков ИЕ3, ИЕ4 рассмотрено в главе 3 «Микросхемы серии 176». Ну, что, полдела сделано, но очень не хочется крутить «хитрый» дешифратор на мелкой логике для преобразования кода Джонсона в семисегментный. И тут на помощь нам снова приходит примитив дешифратора, только на этот раз **DECODER_5_8**. Поступим просто: пять входов этого дешифратора заводим на выходы счетчика, а первые семь выходов описываем таблицей для семисегментного индикатора. Правда таблица получится достаточно длинной, 2 в степени 5, т.е. 32 строки, потому что неиспользуемые комбинации также пришлось включить с нулевыми значениями выходов. Если этого не сделать, **ISIS** будет плеваться горчичниками на отсутствующие комбинации.

Результирующий код получился следующим:

```
{LENGTH=32
TABLE0=%00111111
TABLE1=%00000110
TABLE2=0
TABLE3=%01011011
TABLE4=0
TABLE5=0
TABLE6=0
TABLE7=%01001111
TABLE8=0
TABLE9=0
TABLE10=0
TABLE11=0
TABLE12=0
TABLE13=0
TABLE14=0
TABLE15=%01100110
TABLE16=%01101111
TABLE17=0
TABLE18=0
TABLE19=0
TABLE20=0
TABLE21=0
TABLE22=0
TABLE23=0
TABLE24=%01111111
TABLE25=0
TABLE26=0
TABLE27=0
TABLE28=%00000111
TABLE29=0
TABLE30=%01111101
TABLE31=%01101101}
```

Еще раз напомню, что фигурные скобки в начале и конце кода включают **Hide** - скрытие кода, чтобы он не загромождал проект. Я привел только значимые коды (всего их 10) таблицы в двоичном виде, чтобы было нагляднее видно расположение разрядов в коде, а остальное в десятичном. Симулятор Протеуса обучен воспринимать и другие варианты. Значимые строки тоже можно было бы записать в другом виде, например в шестнадцатеричном. Тогда строка для нуля имела бы вид - **TABLE0=$3F**, а для девятки **TABLE16=$6F**. Знак доллара перед числом означает шестнадцатеричное значение. Но до знака равенства, т.е. после слова **TABLE** можно использовать только десятичное значение. Еще обратите внимание, что семиразрядные коды следуют в таблице не по порядку, поскольку код Джонсона до половины максимального значения (для пятиразрядного счетчика это 10) нарастает, а потом спадает в зависимости от количества поступивших на вход счетчика импульсов. Поэтому числу 5 соответствует **TABLE31**, а числу 9 – **TABLE16**. Кто «не въезжает» – в «Основы цифровой техники», читать про счетчик Джонсона. По этому поводу предупреждал.

Ну и еще один полезный совет тем, кто будет аналогичные операции проводить самостоятельно. Поскольку можно использовать шестнадцатеричные значения, то чтобы «не парить себе мозги», можно воспользоваться утилитами генераторов кода для семисегментных индикаторов, например, встроенными в компиляторы фирмы **Mikroelektronika**.

Оставляем выходы модели счетчика в том виде, в котором они и были, т.е. с использованием исключающего **ИЛИ** (проще тут ничего не придумаешь) и формируем наши модели. Для счетчика **К176ИЕ4** мы практически уже все сформировали, и мне остается только привести получившуюся внутреннюю структуру на Рис. 109.

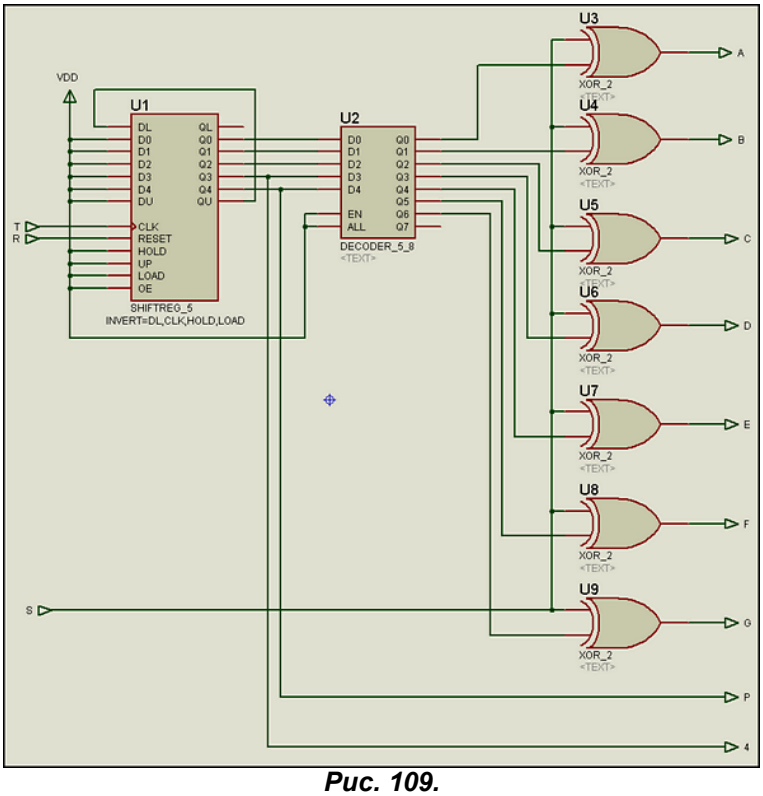


Рис. 109.

Как видно из рисунка, схема получилась даже несколько проще, чем с универсальным счетчиком. Аналогично сформируем и счетчик **К176ИЕ3**. Правда здесь придется уже немного усложнить схему. Дело в том, что менять разрядность входного регистра сдвига мы не можем, он должен быть пятиразрядным, а счетчик должен иметь коэффициент пересчета – 6, и при этом еще и сохранить возможность внешнего сброса – вход R. Поэтому пришлось применить еще один логический элемент двухвходового **ИЛИ**, а для формирования импульса сброса по достижении числа 6 можно воспользоваться незадействованным выходом дешифратора. Достаточно было прописать в нужной строке таблицы единичку в старшем разряде **TABLE30=%11111101**, и мы без всяких лишних хлопот получили нужный нам коэффициент пересчета. Структура приведена на Рис. 110.

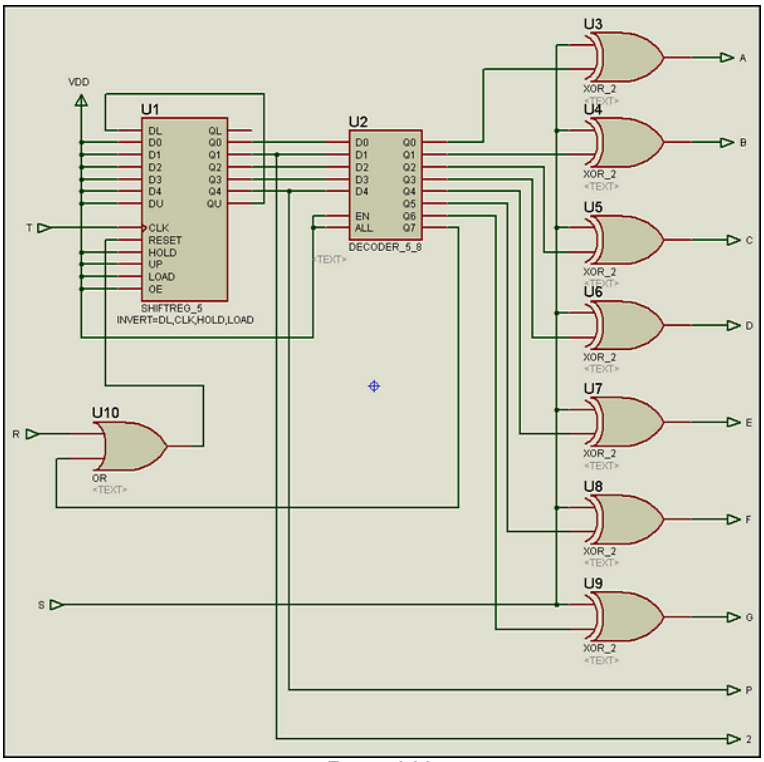


Рис. 110.

Тесты структур для каждого счетчика приведены в папке вложения **COUNTER_STRUCTURE_TEST**, а общий тест счетных декад на их основе в папке **MODELS_WITH_CHILD_LIST**. В ней модели еще имеют дочерний лист и можно туда перейти. Вот с этих листов и скомпилируем **MDF** файлы соответствующих счетчиков. Как это делается, я уже объяснял в примере со счетчиком ИЕ12, но

прежде чем компилировать добавим, также по аналогии с ИЕ12 временные задержки к элементам структуры счетчиков ИЕ3 и ИЕ4. В папке **MODELS_WITH_CHILD_LIST** они уже с добавленными свойствами, поэтому на дочерних листах присутствует скрипт **MAP ON VOLTAGE**. Поскольку у нас модель поведенческая, я добавил временные параметры только к тем элементам, которые непосредственно задействованы в передаче сигналов с входов на выходы микросхемы, т. е. сдвиговому регистру и выходной логике на исключающих **ИЛИ**. Да и у регистра прописаны только те «времянки», которые связаны со счетным входом и входом сброса.
Ну и окончательный тест моделей счетчиков с прописанными в свойствах файлами MDF приложен в папке **MODELS_WITH_MDF_TEST**. Естественно, в этой же папке лежат и сами файлы **176IE3.MDF** и **176IE4.MDF**. Тест показывает, что модели ведут себя вполне адекватно. Итак, у нас уже три собственных работающих схематичных модели 176-й серии. Пришел черед научиться формировать из своих моделей библиотеки.

Возврат к содержанию

6.13. Объединение MDF в библиотеку LML.

Итак, наши цифровые модели серии 176 успешно прирастают в количестве. Желающие могут продолжить создание моделей этой серии и далее, например, по аналогии с **176ИЕ12** создать модель **176ИЕ5**. Можно пополнить модели 176-й серии и совпадающими по назначению аналогами из **CMOS 4000**, сохранив их под новыми именами, соответственно изменив задержки при различных напряжениях питания. Мы уже проделывали такой фокус ранее, получив из **4011** модель **176ЛА7**. Я же на этом закончу рассмотрение процесса моделирования цифровой логики и напоследок покажу – как отдельные схематичные модели с файлами **MDF** объединить в единую библиотеку. Сразу оговорюсь, что совсем не обязательно, чтобы все модели в библиотеке **LML** совпадали по назначению или были только цифровыми, только аналоговыми и т.п. В родных **LML** библиотеках Протеуса полно примеров, когда и «мухи» и «котлеты» благополучно лежат в одной «тарелке». Здесь скорее наоборот, идет принцип разделения по производителю, а не по назначению. Именно поэтому в папке **MODELS** есть библиотеки **FAIRCHLD.LML**, **MAXIM.LML**, **TEXAS.LML** и т.д. Но есть и объединенные «по половому» признаку, например, **MEMORY.LML** – сразу ясно, что внутри модели микросхем памяти. По большому счету, можно и не объединять **MDF**, а просто держать их разрозненно в папке **MODELS**. Но это хорошо, когда у Вас две три своих модели, а когда счет перевалит за пару десятков…. Помимо того, что и самому становится трудно копаться в папке с большим количеством отдельных файлов, Протеусу тоже приходится проделывать эти операции. И что-то мне интуиция подсказывает, что процесс открытия нескольких отдельных файлов и поиск нужных моделей в каждом займет чуть больше времени, чем поиск их в одном открытом файле. Поэтому, для меня лично ответ на вопрос, объединять или нет, однозначен. Итак, приступим.
Для начала соберем все наши модели в единую библиотеку **LIB** в папке **LIBRARY**. Для этого воспользуемся опцией **Library Manager** из верхнего меню **Library**. Напомню, что в одноименных файлах с расширениями **LIB** и **IDX** содержатся сведения о графическом изображении компонентов, в том числе и не имеющих исполняемых моделей (**No Simulation**). Когда мы создаем новый, то по умолчанию **ISIS** предлагает сохранить его в **USRDVC.LIB**. Я эту библиотеку использую как «промежуточную» и периодически провожу там «зачистку» от накопившегося мусора – всяких недоделок и тестовых вариантов. Я решил назвать новую библиотеку **CMOS_RUS** и оставил ей количество позиций 100, предлагаемое Протеусом по умолчанию (Рис. 111).

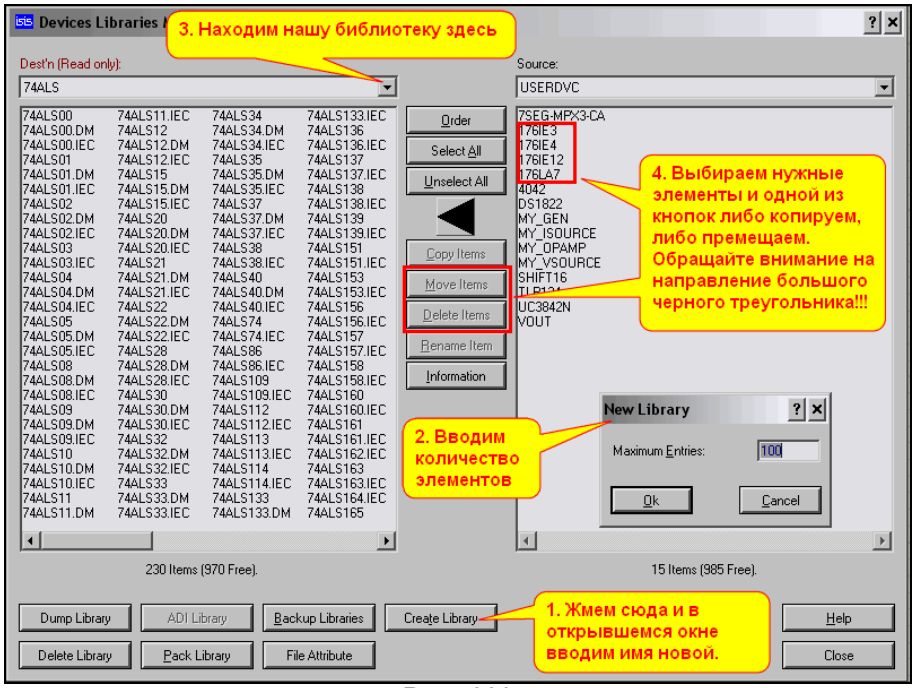


Рис. 111.

После того, как библиотека создана, находим ее через раскрывающийся список (третья операция на рисунке 111). Затем щелкаем по первой из нужных моделей левой кнопкой, зажимаем клавишу Shift и вторым щелчком по последней из нужных – выделяем группу. Я приношу извинения за столь «водянистые» подробности, но последнее время на форуме появились «индивидуумы», которые даже этих типовых операций Винды не знают. Такое «обсасывание костей» персонально для них. Давим кнопку **Move Items** и в появившемся окне подтверждаем еще раз выполнение операции. После этого наше окно примет вид, показанный на рисунке 112. Естественно, в библиотеке **USRDVC** у Вас на компьютере будут находиться совсем другие модели (те, что когда то сохранялись через операцию **Make Device**), а 176-е там будут присутствовать, только если Вы применяли эту функцию к графическим моделям, которые я прилагал в предыдущих примерах.

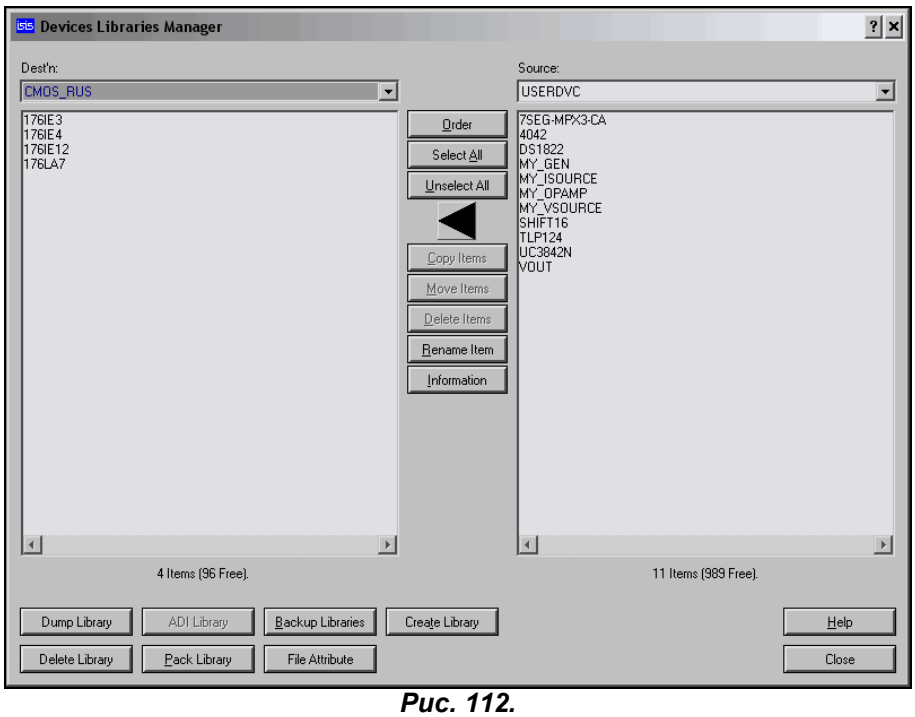


Рис. 112.

После того, как мы создали **CMOS_RUS** и перенесли в нее нужные модели, это окно нам больше не нужно и его можно **Close** соответствующей кнопкой внизу справа. На этом с папкой LIBRARY покончено. Теперь у нас там должны присутствовать **CMOS_RUS.IDX** и **CMOS_RUS.LIB**.
Наступил черед создания **LML**. Для этого в отдельную папку, лучше в корне диска копируем все нужные файлы моделей с расширением **MDF**, которые мы создали ранее и туда же копируем (а не перемещаем!!!) утилиту **PUTMDF.EXE** из папки **BIN** установленного Протеуса. Запускаем утилиту командной строки либо через **Пуск=>Все программы=>Стандартные**, либо **Пуск=>Выполнить**, набрать в появившемся окне **cmd** и нажать **ОК**.
Весь процесс создания **CMOS_RUS.LML** показан на рисунке 113. В данном случае все файлы были помещены во вновь созданную папку **MakeLML** в корне диска **C:**. Сначала я перешел в эту папку, затем вызвал **PUTMDF** без ключей, чтобы увидеть подсказку (помощь), потом создал **CMOS_RUS.LML** на 100 элементов и в последней операции поместил туда четыре ранее созданных **MDF**. Для тех, кто дружит с компьютером, а не «забивает им гвозди» – операции 3 и 4 можно объединить. В этом случае строчка будет выглядеть так:

PUTMDF –L=CMOS_RUS –C=100 176IE3 176IE4 176IE12 176LA7

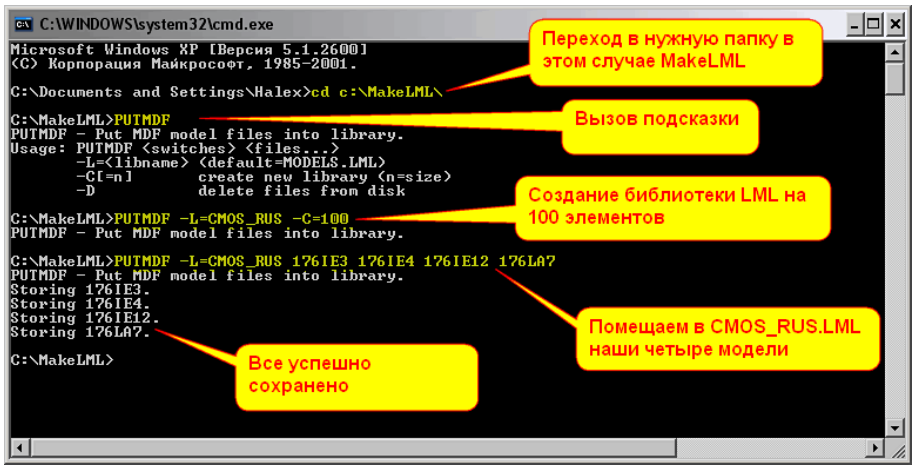


Рис. 113.

В конечном итоге в папке **MakeLML** должен появиться файл **CMOS_RUS.LML**, а после добавления в него моделей его размер увеличится. Этот файл перемещаем в папку **LIBRARY** Протеуса. Если там остались **MDF** тех моделей, которые мы поместили в нашу **LML**, то их удаляем. В дальнейшем, в эту библиотеку **LML** можно будет добавить еще **MDF** общим количеством до 100.
И еще одно замечание. Я в обоих случаях использовал одинаковое название и для **LIB** и для **LML**. Это совсем не обязательно, тут дело вашего вкуса.
Ну вот, получился такой короткий, но полезный материал. Теперь немного остановимся на **MIXED** (смешанных аналого-цифровых) моделях и перейдем к самому интересному – активным моделям. Для тех, кому «с трудом и словарем живого великорусского языка» поддается данный материал, во вложении прилагаются готовые библиотеки **CMOS_RUS**. Копируем файлы из папок архива в одноименные Протеуса и … наслаждаемся.

Возврат к содержанию

6.14. MIXED примитивы для аналого-цифровых и цифро-аналоговых преобразований.

Ранее, в **п.6.1**, мы уже познакомились с некоторыми свойствами элементарных однобитных АЦП и ЦАП и успешно применяли их при исследовании цифровых моделей. Остановимся еще раз на них подробнее и познакомимся с N-битными примитивами АЦП и ЦАП, которые нам потребуются в дальнейшем материале. Все они расположены в библиотеке **Modelling Primitives => Mixed Mode**.
Начнем со свойств элементарного АЦП (Рис. 114). Модель имеет один аналоговый вход (**A**) и один цифровой выход (**D**).

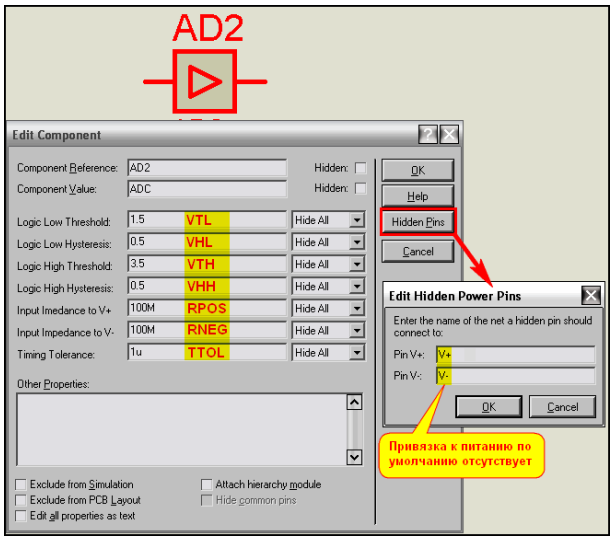


Рис. 114.

Я не буду еще раз подробно останавливаться на параметрах порогов (**VTL** и **VTH**) и гистерезисов (**VHL** и **VHH**), поскольку мы их подробно рассмотрели ранее, в **п.6.4**. Там мы рассматривали поведение примитивов цифровых буферов и инверторов и использовали только эти параметры АЦП, поскольку все остальные были уже условно привязаны через файл **ITFMOD.MDF**. Однако теперь, если мы хотим использовать при моделировании элементарный АЦП, они для нас имеют важное значение.
RPOS и **RNEG** – соответственно входной импеданс к положительному **V+** и отрицательному **V-** питающим выводам аналогового входа АЦП. Как видим, и к отрицательному и к положительному питанию сопротивление составляет по умолчанию 100 МегаОм и практически не оказывает влияние на подключаемые по входу АЦП цепи.
Выводы питания **V+ (**положительное) и **V-** (отрицательное) являются скрытыми и доступны для редактирования через кнопку **Hidden Pins** (Рис. 114). Значения по умолчанию **V+** и **V-** не являются стандартными для **Power Rails** и поэтому, если вы просто поместите в проект элементарный АЦП и запустите симуляцию, получите сообщение об ошибке. Вы вольны по своему усмотрению привязать их к любым питающим шинам и любым из рассмотренных ранее способов. Это можно проделать и через меню **Design=>Configure Power Rails** или просто добавить в окошке **Edit Hidden Power Pins** (Рис. 114) вместо **V+** и **V-** ранее сконфигурированные или существующие (например, вместо **V+** назначить **VCC**, а вместо **V-** – **GND**) питающие шины. Можно просто на листе проекта связать попарно терминалы питания **V+** и к примеру **+12V** и **V-** и **-12V**, соответственно при этом вход будет подтянут через резисторы **RPOS** и **RNEG** не к **VCC/GND**, а к этим потенциалам.
Еще одно существенное замечание состоит в том, что выход элементарного АЦП полностью оправдывает свое назначение, т.е. чисто цифровой. Он не проявляет аналоговых свойств, даже если к нему нацеплять аналоговых компонентов, соответственно он симулируется только на цифровом графике и может оказывать влияние только на вход последующего цифрового компонента. Забегая вперед, замечу, что для элементарного ЦАП по входу та же картина, но с точностью до наоборот. Для нас это означает, что если мы хотим после АЦП связать сигнал с каким либо аналоговым компонентом (даже с резистором), то необходимо на выход АЦП прилепить еще и

цифровой примитив **BUFFER**, а уж с его выхода уходить на транзисторы, резисторы, и пр. Ну и соответственно для ЦАП, при подаче на его вход сигналов с аналоговых примитивов по входу потребуется цифровой **BUFFER**. Это необходимо учитывать при создании собственных моделей с применением этих примитивов.
Ну и еще один параметр, который существенен для элементарного АЦП – **TTOL** – время переключения. По умолчанию оно составляет 1 микросекунду.
Теперь рассмотрим свойства элементарного ЦАП (Рис. 115). Модель имеет один цифровой вход (**D**) и один аналоговый выход (**A**).

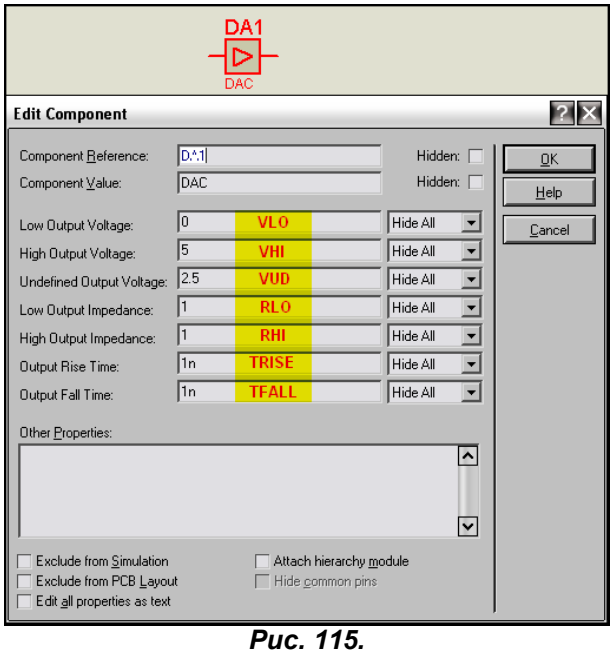


Рис. 115.

С ним мы раньше не знакомились, поэтому опишу поподробнее. Этот примитив в отличие от ADC не требует привязки каких либо напряжений к питанию, поскольку привязка выходов уже прописана в свойствах.
VLO, **VHI**, **VUD** – три выходных напряжения. Соответственно низкого уровня (**Low Voltage Output**), высокого уровня (**High Voltage Output**) и неопределенного уровня (**Undefined Voltage Output**). Ну, и их значения по умолчанию равны соответственно 0, 5 и 2,5 Вольт.
RLO и **RHI** – выходные импедансы (нагрузочные сопротивления) соответственно к нижнему (**VLO**) и верхнему (**VHI**) выходным напряжениям. По умолчанию и то и другое по 1 Ому.
TRISE и **TFAIL** – длительность соответственно переднего и заднего фронтов выходного сигнала при переключении (по умолчанию 1 наносек).
Еще один MIXED примитив, с которым мы не сталкивались раньше – это **DSWITCH** (Dual Mode Switch) – основа всех моделей аналоговых коммутаторов. Модель (Рис. 116) имеет два аналоговых, равнозначных по назначению входа/выхода – **A** (слева) и **B** (справа) и управляющий цифровой вход **EN** (сверху). Все свойства этого входа доступны через раскрывающийся список и составляют стандартный набор всевозможных задержек переключения, как и у цифровых примитивов. Поэтому на них останавливаться не будем, а рассмотрим только свойства аналогового переключателя.

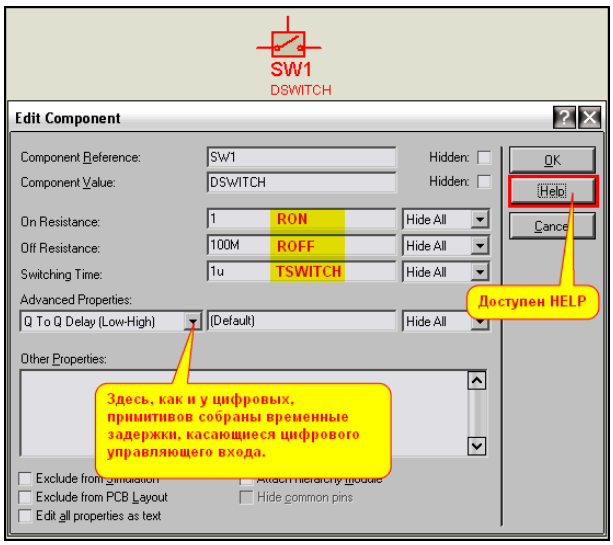


Рис. 116.

RON (1 Ом) и **ROFF** (100 МОм) – соответственно сопротивление между входами A и B во включенном и выключенном состоянии.
TSWITCH – время переключения аналогового переключателя – по умолчанию 1 микросек. Обращаю ваше внимание, что в **HELP** указаны отдельно **TON** (время включения) и **TOFF** (время выключения). Если необходимо сделать время включения и выключения различным, можно вручную задать их в окне **Other Properties**. Однако при этом **TSWITCH** надо определить как **(Default)**, иначе оно будет преобладать. Указать надо именно так, в скобках, а не цифру 0 или что-то еще. Если кто-то боится ошибиться в написании – скопируйте из окна любой из задержек и вставьте в окне **Switching Time**.
Ну, вот вроде вкратце все об элементарных смешанных моделях. Немного полезных опытов с ними сведены в проекты, представленные в папке **Mixed_Prim/OneBitPrimitives** вложения. По названию проектов **ADC**, **DAC**, **DSWITCH** – можно понять что внутри.
А мы переходим к многоразрядным АЦП и ЦАП. В отличие от элементарных, кнопка помощи для этих примитивов не доступна, но **HELP** по ним все-таки есть. Открыть его можно, выбрав соответствующие разделы через **ПУСК => Все программы => Proteus Professional=> Proteus VSM Model Help => ProSPICE Primitives** (Рис. 117).

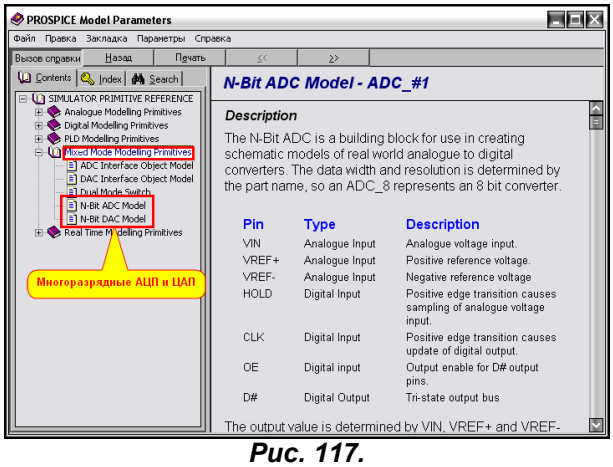


Рис. 117.

Начнем знакомство с аналого-цифровых примитивов N-bit ADC. В папке **Modelling Primitives => Mixed Mode** библиотек Протеуса представлены 8, 10, 12 и 16-ти разрядные ADC. Поскольку все параметры у них одинаковы, рассмотрим самый простой – 8-разрядный. Остальные отличаются лишь тем, что для экономии места их выходные сигналы сведены в соответствующую N-разрядную шину, а у **ADC_8** они выведены на отдельные пины. На рисунке 118 представлены вид модели в окне проекта и окно параметров по умолчанию.

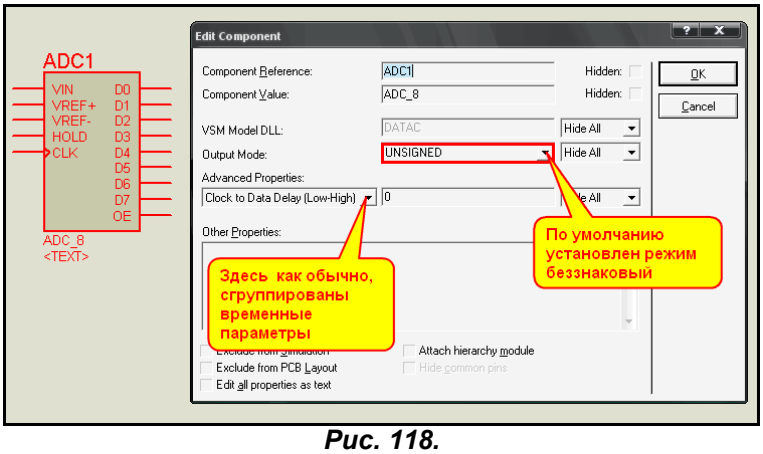


Рис. 118.

Для начала познакомимся с назначением входов/выходов модели.
Входы аналоговые:
VIN – вход аналогового сигнала АЦП.
VREF+ и **VREF-** – соответственно положительный и отрицательный входы опорного напряжения.
Входы цифровые:
HOLD – вход разрешения фиксации уровня аналогового сигнала в АЦП. По умолчанию сигнал фиксируется по положительному перепаду 0-1 на этом входе.
CLK – тактовый вход вывода сигнала на цифровые выходы. По умолчанию сигнал выводится по положительному перепаду 0-1 на этом входе.
OE – вход разрешения вывода сигнала на цифровые выходы. При лог. 1 на этом входе вывод разрешен, при лог. 0 – выходы находятся в высокоимпедансном состоянии.

Цифровые выходы:
D0…Dn – на этих выходах результат преобразования выводится в цифровом виде в соответствии с выбранным в свойствах режимом **Output Mode**.
Теперь заглянем в окно свойств. Помимо обычного набора всяческих задержек упрятанных в раскрывающийся список и по умолчанию равных 0, здесь присутствует только один важный для нас параметр **Output Mode** – режим вывода данных.
По умолчанию он принят **Unsigned**, т.е. беззнаковый. Это означает, что код на выходе АЦП задействует все разряды и равен отношению **Vin/Vref**, представленному в двоичной форме. Соответственно для **Vin=Vref** мы получим на выходе восьмиразрядного АЦП шестнадцатеричный код **0xFF**.
Если выбран режим **SIGNMAGNITUDE** – знакозависимый, то старший цифровой разряд выбранной модели используется под знак для нуля и положительных значений в этом разряде 0, а отрицательных – 1. В остальных разрядах представлен прямой двоичный код. Нулю соответствует половина **Vref**. Так, например, для 8-ми разрядного АЦП при **Vref=+5В** напряжению **Vin=+2,5В** будет соответствовать код **0x00**, **Vin=+2,51В** – код **0x01**, а **Vin=+2,49В** – код **0x81**. Соответственно необходимо помнить, что раз у нас один разряд задействован под знак, то полному положительному размаху уже будет соответствовать код с меньшей разрядностью, т.е. для **Vin=Vref** код будет **0x7F**, а для **Vin=-Vref** – код **0xFF**.
Ну и наконец, последний режим – **TWOSCOMPLEMENT**. Он аналогичен знакозависимому, но значения **Vin** ниже **Vref/2** при этом представлены в дополнительном коде. Применительно к рассмотренному выше примеру значению **Vin=+2,49В** будет соответствовать код **0xFE**, а для **Vin=-Vref** код будет **0x80**. Этот вариант наиболее часто встречается в реальности, поскольку облегчает вычисления в двоичном виде.
Если кому-то принципы преобразований непонятны – обратитесь к соответствующей литературе. В частности именно по этому вопросу рекомендую прочитать гл. 2.1. Кодирование и квантование в книге **«Аналого-цифровое преобразование» под ред. У. Кестера, М., «Техносфера, 2007**.
Ну и еще парочка практических замечаний по моделям **ADC** в Протеусе. Как Вы наверное уже догадались, для того чтобы использовать только однополярный сигнал **Vref** достаточно вход **VREF-** завесить на землю. Несколько сложнее, если необходимы аналоговые **VREF** и **VIN** никак не связанные с **GND**. Но и в этом случае все решается достаточно просто. На входы АЦП добавляются примитивы **AVCVS** или по-русски ИНУН (источник напряжения управляемый напряжением) с коэффициентом передачи **1,0** в качестве гальванических разделителей, как показано на Рис. 119. При этом не забудьте, что они имеют бесконечное входное сопротивление, поэтому при имитации реальных устройств потребуются входные резисторы с нужным сопротивлением.

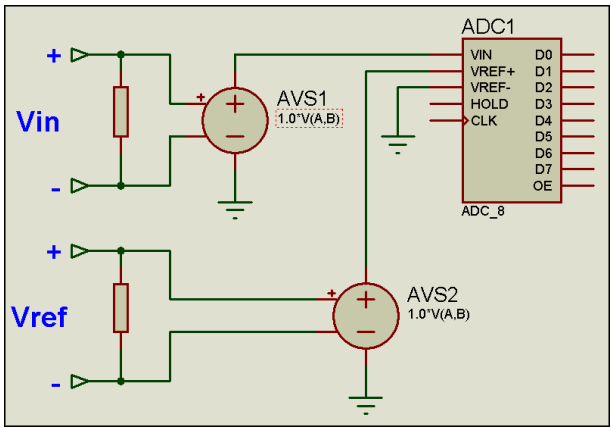


Рис. 119.

Параметры N-разрядных **DAC** (ЦАП) аналогичны параметрам АЦП, если принять во внимание, что входы/выходы здесь поменялись местами (Рис. 120).

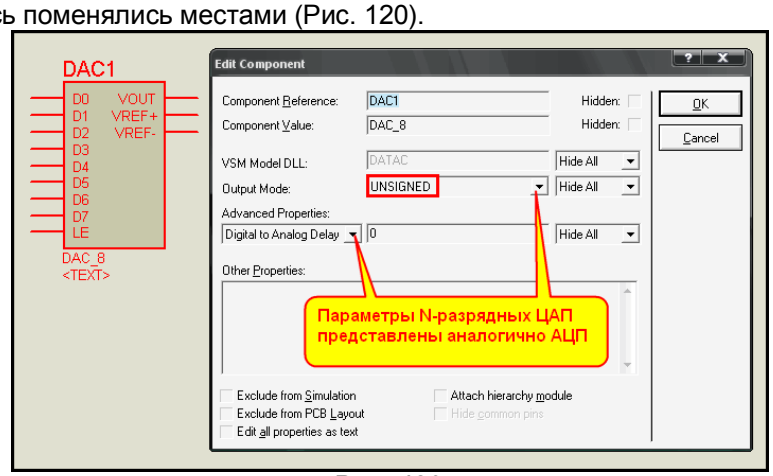


Рис. 120.

Высокий уровень на входе **LE** (**Load Enable**) позволяет запись цифрового кода со входов **D0…Dn** во внутренние регистры ЦАП. После возвращения **LE** в ноль они защелкиваются там до следующего высокого уровня на **LE**. Как видим, управление здесь еще проще, чем в ЦАП, да и времянок всего две, так что есть смысл их описать.
TDDA – **Digital To Analog Delay** – задержка преобразования из цифрового кода в аналоговый сигнал. По умолчанию она нулевая.
SLEWRATE – **Slew Rate (Volts/Sec)** – скорость нарастания выходного аналогового напряжения. По умолчанию она равна **1000000 В/сек**.
Раскрывающееся меню режима **Output Mode** в параметрах ЦАП полностью аналогично рассмотренному выше для АЦП и содержит те же опции.
Примеры применения примитивов ADC и DAC для создания схематичных моделей есть в стандартной поставке Протеуса. В папке **SAMPLES\Graph Based Simulation** находятся два примера для АЦП: **ADC0808.DSN** и **ADC0831.DSN** и пример ЦАП: **DAC0808.DSN**. Во всех примерах при щелчке правой кнопкой по изображению ADC или DAC возможен переход на дочерний лист, где и представлена соответствующая модель «вид изнутри». Разбирать эти стандартные примеры здесь не имеет особого смысла, поскольку базируются они на тех примитивах, которые рассмотрены выше.
В качестве тестовых примеров для изучения поведения 8-ми битных АЦП и ЦАП во вложении **Mixed_Prim** приложены соответствующие проекты в папках **ADC_8** и **DAC_8**. Некоторые комментарии даны прямо в проектах. Ниже мы практически познакомимся с моделированием 12-ти разрядного реального АЦП **MAX1241**, где будет задействован примитив N-битного ADC. Почему именно **MAX1241**? Да просто мне понадобилась эта модель для собственных нужд, ну и выяснилось, что модель даже в последних версиях приложена Протеуса не без ошибок. Вот мы и займемся «работой над ошибками», а заодно и поучимся моделировать реальные АЦП. Но прежде нам придется освоить еще один очень полезный примитив, описание которого отсутствует в HELP Протеуса. Это **SPISLAVE** – модель подчиненного последовательного интерфейса. Именно по **SPI** АЦП **MAX1241** общается с внешним миром.

Возврат к содержанию

6.15. Примитив SPISLAVE. Исследуем поведение последовательного интерфейса с помощью различных цифровых генераторов.

Наряду с обычными аналоговыми, цифровыми и смешанными примитивами, описание которых хоть как то представлено в HELP, в библиотеках Протеуса существует ряд моделей, на основе которых создаются схематичные компоненты, но описания их свойств полностью отсутствуют. К таковым как раз и относятся расположенные в папке **Modelling Primitives/Digital(Miscellaneous)** примитивы различных подчиненных интерфейсов **SPISLAVE** (интерфейс SPI), **I2SSLAVE** (интерфейс I2S), **D1WSLAVE** (интерфейс 1-wire). Для воссоздания MAX1241 нам потребуется SPI. Вот его-то мы и рассмотрим сейчас. Примитив **SPISLAVE**, как и АЦП/ЦАП изначально представлен для 8-ми, 12-ти и 16-тибитного интерфейсов. Как и ранее рассмотрим только 8-ми битный, остальные аналогичны. Для начала познакомимся с назначением выводов модели (Рис. 121).

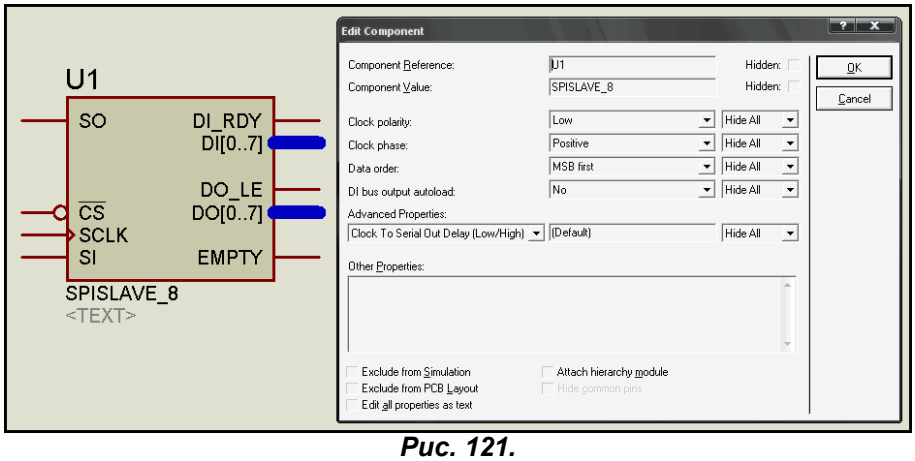


Рис. 121.

- В левой части модели расположены выводы характерные для внешнего последовательного интерфейса. К ним относятся:

SO – **SPI Output** – цифровой последовательный выход данных на другие компоненты.
SI – **SPI Input** – цифровой последовательный вход данных от других компонентов.
CS – **Crystal Selector** – цифровой вход. По умолчанию низкий уровень на нем служит для выбора (активации) модели.
SCLK – **SPI Clock** – тактовый вход. На него подаются импульсы с частотой обмена информацией по интерфейсу.
В правой части модели расположены выводы и 8-ми разрядные параллельные шины для компоновки внутренней структуры модели какого-либо компонента. Сюда входят:

DI_RDY – **Data Input Ready** – цифровой выход выдачи сигналов принятых со входа **SI** на параллельную шину **DI[0..7]**. По умолчанию вырабатывает положительный перепад 0-1 по окончанию (перепаду 0-1) входного сигнала **CS**. При этом принятые данные выводятся на шину. Поведение выхода зависит от параметра **DI bus output autoload** (см. ниже).
DO_LE – **Data Output Load Enable** – цифровой вход для выгрузки данных с параллельной шины **DO[0..7]** на выход **SO**. Для правильной (неискаженной) выдачи кода на **SO** положительный перепад 0-1 на этом входе должен предшествовать первому тактовому импульсу (по умолчанию положительному фронту) на входе **SCLK** после активизации сигнала **CS**.
EMPTY – цифровой выход. Сигнал на нем зависит от параметра **DI bus output autoload** (см. ниже).

- Теперь рассмотрим изменяемые в окне свойств параметры модели:

Два параметра: **Clock polarity** (по умолчанию **Low** – низкий уровень) и **Clock phase** (по умолчанию **Positive** – положительный 0-1) описывают требуемый входной тактовый сигнал **SCLK**. При необходимости можно поменять на **High** и **Negative** соответственно, чтобы подобрать нужный фронт тактирования модели.
Data order – порядок выдачи (приема) данных в последовательном интерфейсе. По умолчанию принят **MSB first**, т.е. старший бит данных (для 8-ми разрядной модели это **D7**) выдается (принимается) первым. Если поменять на **LSB first**, то первым в последовательности будет **D0**.
DI bus output autoload – этот параметр определяет логику работы (автозагрузку) шины данных **DI[0..7]**. По умолчанию **[Default]** эквивалентно **No** автозагрузка запрещена. Это означает, что модель не делает подсчет входных тактовых импульсов, а производит выгрузку данных на внутреннюю шину **DI** по окончании действия сигнала на входе **CS**. Чем это чревато рассмотрим сразу, не откладывая в долгий ящик.
Предположим, мы загружаем в восьмиразрядный **SPISLAVE** по входу **SI** кодовую последовательность **0x91 (b10010001)**. При этом сигнал выборки **CS** у нас значительно длиннее, чем длительность восьми тактовых импульсов, а тактовые импульсы следуют непрерывно, т.е. их тоже пройдет больше восьми. Фактически универсальная программная модель **SPISLAVE** представляет собой сдвиговый регистр, а точнее два – один внутрь на шину **DI**, другой наружу – на выход **SO**. Разрядность регистров определил программист Лабцентра при создании модели, и она нам неизвестна. Хотя, учитывая некоторый мой опыт по исследованию моделей, могу предположить, что там заложено не менее 32 разрядов. Допустим, что **CS** завершится у нас после 10-го тактового импульса. При этом два лишних импульса протолкнут данные на два разряда влево, добавив два нуля в младшие разряды. Если первым следовал **MSB**, то разряды **D7** и **D6** уйдут влево за пределы байта, на место **D6** встанет единица из бывшего **D4**, а на **D2** единица из бывшего **D0**. Проверим это на практике. На две модели подадим одинаковые сигналы, но у первой включим режим **DI bus output autoload** (**Yes**), а у второй оставим по умолчанию (Рис. 122).

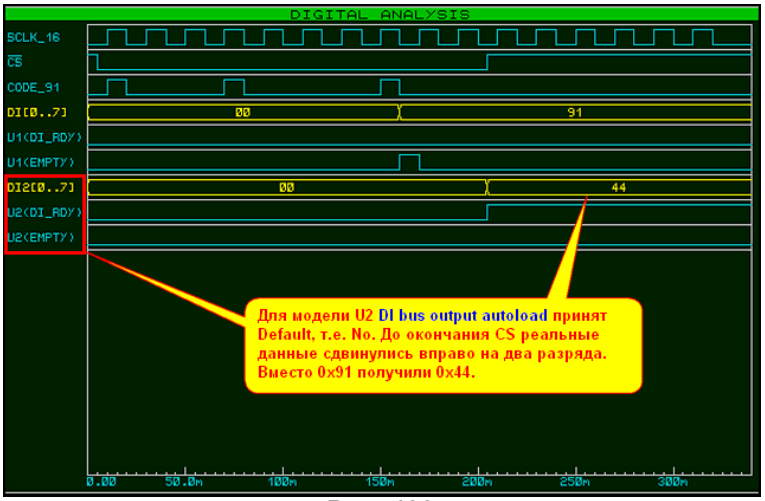


Рис. 122.

Из графика видно, что предположения полностью подтвердились, т.е. для второй модели (**U2**) мы имеем на выходе код **0x44** (**b01000100**). Справа в этом байте появились два нуля, обусловленные двумя лишними импульсами. Для модели **U1**, у которой включен режим **DI bus output autoload** данные были занесены в **DI** правильно. Я прошу вас обратить особое внимание на поведение выходов **DI_RDY** и **EMPTY** при включенном (**U1**) и выключенном (**U2**) режиме автозагрузки. В первом случае по окончании 8-го импульса (именно для **SPISLAVE_8**, а для **SPISLAVE_12** будет по окончании 12-го) на выходе **EMPTY** появился импульс и в этот момент появились правильные данные на шине **DI[0..7]**, в то время как **DI_RDY** вообще не принимал участия в обмене и оставался строго низким. Во втором случае **EMPTY** был безучастным, но зато на **DI_RDY** появился положительный перепад по окончании сигнала **CS** и именно в этот момент были выданы данные, проехавшие на пару разрядов вперед на шину **DI[0..7]**. Это может быть несколько сложный для сиюминутного восприятия, но очень важный момент поведения модели **SPISLAVE** при приеме данных и о нем нельзя забывать при использовании примитива для разработки собственных моделей компонентов.

Во временных параметрах модели **SPISLAVE** присутствуют всего две задержки, по умолчанию нулевые. Они описывают задержку выдачи сигнала в последовательный интерфейс относительно тактового **Clock** одна по перепаду 0-1, другая по перепаду 1-0.
Чтобы несколько подробнее познакомиться с поведением модели я приложил несколько простых примеров приема, передачи и одновременно приема/передачи (поскольку каналы **SI** и **SO** если их не объединить специально абсолютно не мешают друг другу, то возможно и такое) с использованием различных цифровых генераторов из левого меню **Generator Mode**. Конечно, можно было бы воспользоваться и подручной моделью микроконтроллера, написать программу обмена и т.п. Но я хочу попутно приучить вас использовать «подручные средства» для достижения своих целей. Цифровые генераторы из меню **Generator Mode** наиболее подходящие для этих целей, поскольку практически не загружают ЦП компьютера и позволяют получить нужный результат. Элементарный пример – для первоначального сброса МК многие пририсовывают в проект обычную RC-цепочку. И свято верят, что это обязательно нужно при моделировании и работает. Кроме бесполезной траты времени компьютера на вычисление начальной точки аналоговой цепи данный прием ничего хорошего не дает. В реальности она работать будет, в симуляции необходимо, по крайней мере, задать начальные условия (**IC**) для точки соединения R, C и входа сброса МК, ну или задать нулевой начальный заряд конденсатора (**PRECHARGE**). В то же время, воспользовавшись генератором перепада (**DEDGE**) расшифровывается как **Digital Edge** (цифровой перепад) можно на сколь угодно долго задержать старт МК или другого цифрового компонента, имеющего вход сброса. На рисунке 123 пример «безграмотного» и грамотного применений стартовой задержки. Как видите, гарантированные 250 миллисекунд дает только генератор цифрового перепада. И совсем не обязательно применять такой генератор именно для коротких задержек. Вы можете задать ему время перепада и через **3600s** (1 час), если требуется имитировать включение какого либо устройства.

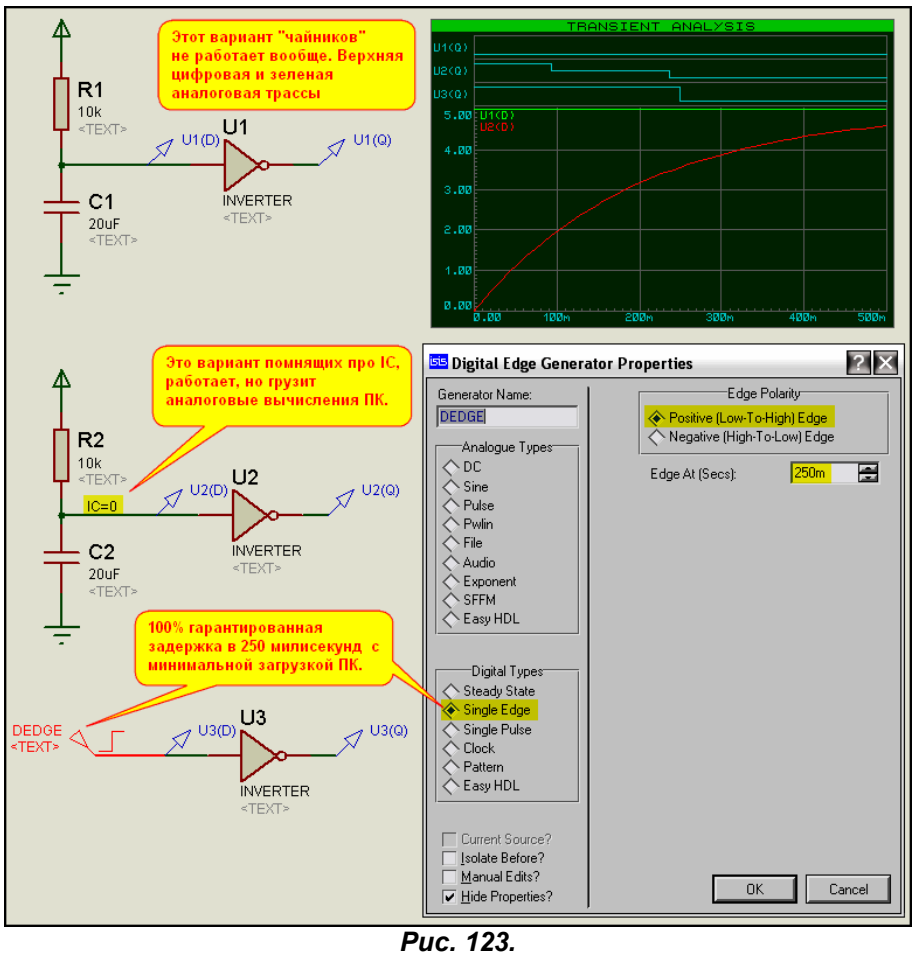


Рис. 123.

Аналогично можно задать и не единичный перепад, а единичный импульс **Single Pulse** или непрерывную тактовую **Clock** именно из генераторов **Digital Types**. Только параметров там уже можно задать больше. Почему я так настойчиво упираю на применение для цифровых устройств именно этих типов? По-прежнему на форуме вылезают кривые проекты со 100% загрузкой ЦП. Открываешь проверять – вот оно, вместо **Digital Clock** автор кривого проекта пихнул **Pulse** из расположенного выше окна **Analogue Types**. В проекте и так напихано аналоговых элементов, так еще и генератор аналоговый. Эти два селектора не зря разделены. Все что аналоговое – оно и работает как аналоговое, со всеми вытекающими при этом дополнительными вычислениями: токи, импеданс и т.п. – отсюда и лишние тормоза. Я в последний раз заостряю на этом внимание, просто устал повторяться. А затеял я разговор о цифровых генераторах по поводу генератора **Pattern**, который мы сейчас и применим для исследования нашего **SPISLAVE**. Будут там и **Single Edge** и

Single Pulse, но главным инструментом будет этот. Давайте познакомимся с назначением его свойств (Рис. 124) и применим нужный нам режим, а мне надо получать последовательность из определенного количества импульсов для входа **SCLK** и синхронизированные с **SCLK** во времени последовательные кодовые комбинации для входа **SI**.

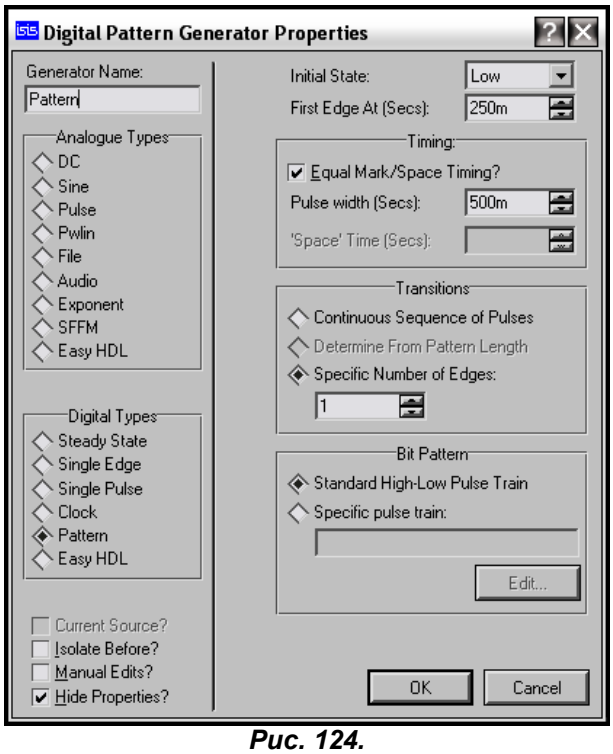


Рис. 124.

В верхней части окна свойств расположены:
Initial State (стартовое состояние) по умолчанию **Low** (лог. 0) – меня это устраивает и мы оставляем как есть.
First Edge At (Sec) – первый перепад сигнала через **250m** (миллисек) – слишком долго ждать, я поставлю **10m**.
В следующей группе **Timing** с помощью флажка **Equal Mark/Space Timing** определяется, будут ли длительности импульса и паузы одинаковыми. Если флажок убрать, то можно задать отдельно длительность импульса **Pulse width** и паузы **'Space' Time**. Я флажок оставлю, но длительности подрежу с 500 до тех же **10m**.
Следующая группа **Transitions** определяет, как будут выдаваться импульсы. Если переключатель стоит так, как на рисунке 124 **Specific Number of Edges**, то будет выдано определенное в окне ниже число импульсов (в данном случае 1). Если переключатель установить в **Continuous Sequence of Pulses**, то будет выдаваться бесконечно повторяющаяся последовательность импульсов (по сути, получим цифровой **Clock**). Третье положение **Determine From Pattern Lengh** – определяемое продолжительностью схемы пока не доступно для включения (серое). Но именно оно меня и интересует.
Для того чтобы оно стало активным переведем переключатель в группе **Bit Pattern** из положения **Standard High-Low Pulse Train** – стандартная последовательность нулей и единиц в положение **Specific pulse train** – заданная последовательность импульсов. Вот теперь в наборе выше нам стали доступны все положения. Там я ставлю **Determine From Pattern Length**, а в группе Bit Pattern щелкаю по кнопке **Edit**, Которая стала активной. В результате откроется окно редактирования последовательности импульсов **Edit Pattern** (Рис. 125).
В окне **Edit Pattern** с помощью мышки можно набрать любую требуемую нам комбинацию импульсов, причем с различной длительностью и тремя доступными уровнями: **High** (лог. 1), **Float** (неопред. уровень) и **Low** (лог. 0). Всего доступно 48 временных интервалов – горизонтальная шкала. Длительность каждого интервала определяется заданным в группе **Timing** значением **Pulse width**. Таким образом, если я задал там значение **10m**, каждая горизонтальная клетка равна 10 миллисекундам. Установка уровня в нужном месте осуществляется левой кнопкой мышки. Если необходимо заполнить несколько временных интервалов одним уровнем – просто проводим по нужному уровню с зажатой левой кнопкой мыши. Если все 48 интервалов не нужны, то незадействованные не заполняем. Они будут индицироваться пунктирной линией, и воспроизводиться при симуляции не будут. Ну и если случайно вы заполнили в конце лишние интервалы их можно затереть (пунктиром), используя правую кнопку мыши.
Ну и последний нюанс – мне в данный момент нужен режим **Determine From Pattern Lengh**, но совсем не обязательно использовать именно его. Если поставить **Continuous Sequence of Pulses**, то набранная в окне редактирования схема выдачи импульсов при запуске симуляции будет повторяться бесконечно.

Для сигнала **SCLK** на рисунке 125 я задал 16 импульсов с равными длительностью и паузой. В принципе, для 8-ми разрядного **SPI** хватило бы и 8-ми, но надо же посмотреть – как поведет себя модель, когда на вход будут поступать лишние импульсы.

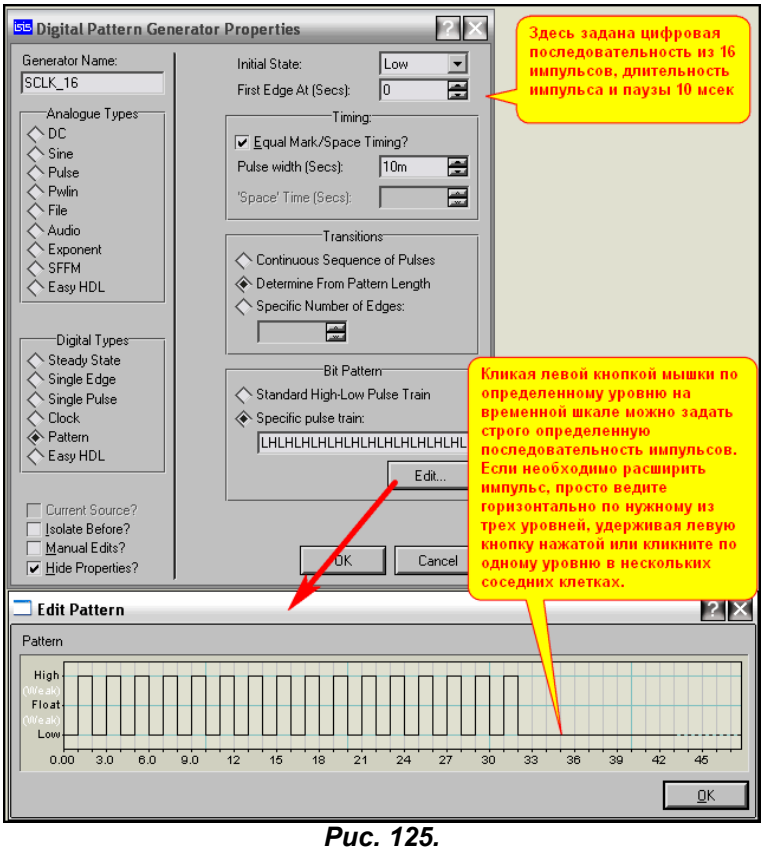


Рис. 125.

Для того чтобы подать на вход **SI** определенную кодовую комбинацию я просто через **Block Copy** скопирую в проекте этот генератор, подключу копию к входу **SI**, а затем подредактирую нужную мне комбинацию **Pattern**, убрав ненужные импульсы. На рисунке 126 показано окно **Edit Pattern** для подачи на вход **SI** кодовой комбинации **0x91**.

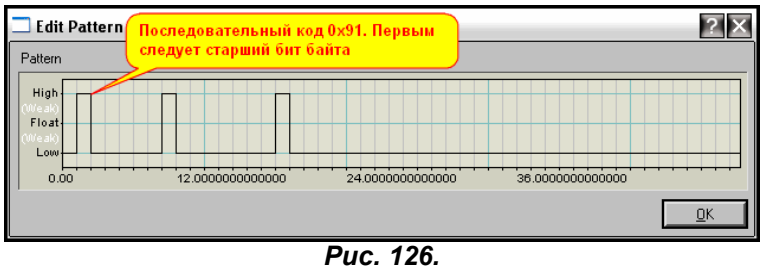


Рис. 126.

Ну, вот вроде и все об использовании генератора **Pattern**. Теперь о том, что во вложении **SPISLAVE.RAR** к этому разделу. В примере **SPI_8_autoload.DSN** показано влияние свойства **DI bus output autoload** на прием информации с входа **SI** на внутреннюю шину **DI**. В примере **SPI_8_Out.DSN** показана передача информации на вывод **SO** с внутренней шины **DO**. Ну и наконец, в примере **SPI_8_2direction.DSN** показан одновременный прием и передача информации по последовательным выводам **SI** и **SO**. Дополнительные комментарии вы найдете непосредственно в проектах.

Возврат к содержанию

6.16. 12-ти разрядный АЦП со SPI интерфейсом MAX1241. Анализируем поведение модели в Протеусе.

Для того чтобы понять – правильно ли работает модель АЦП **MAX1241** необходимо первоначально обратиться к даташиту на данный АЦП. Можно скачать непосредственно из Протеуса, поместив модель в поле проекта и нажав на кнопку **Data** в свойствах (естественно, канал Интернета при этом должен быть подключен). Но при этом вы заполучите даташит ревизии 2 от 1998 года. Более свежий вариант Rev. 5 от 2010 года доступен на сайте фирмы Maxim: http://www.maxim-ic.com
Даташит единый на **MAX1240/MAX1241**. Отличаются эти микросхемы только тем, что **MAX1240** имеет встроенный источник опорного напряжения 2,5 Вольта. Для тех, у кого с английским так и не заладилось, могу порекомендовать найти книжку **П. Гелль «Как превратить персональный компьютер в измерительный комплекс», М., ДМК,1999**. Правда, там **MAX1241** касаются только

боком, но подробно рассмотрен **MAX1243** – его десятиразрядный «собрат». Некоторые, необходимые нам в дальнейшей работе выдержки из даташита 2010 года в моем персональном, может и чуть «корявом» переводе, будут размещены и здесь. Чтобы оценить «качество перевода», эти фрагменты я буду выделять бирюзовым цветом, некоторые мои пояснения будут вставлены курсивом. Итак, познакомимся для начала с самой микросхемой. Функциональная диаграмма **MAX1241** приведена на рисунке 127.

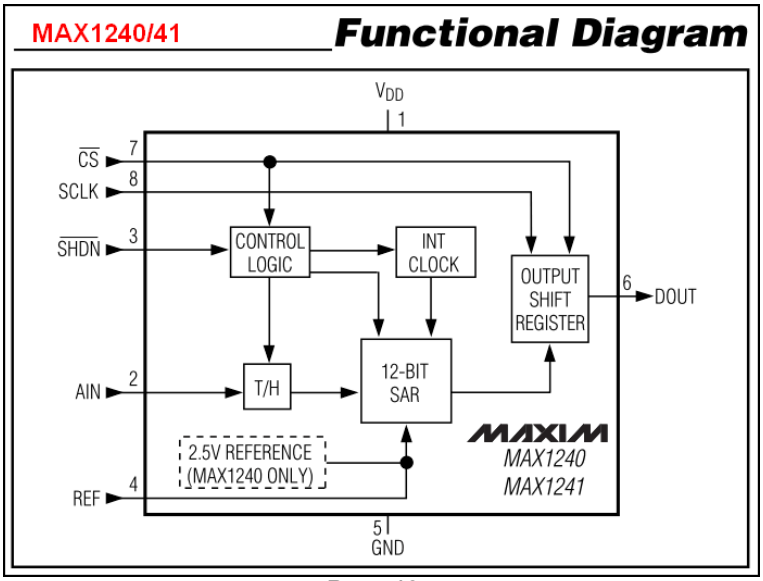


Рис. 127.

Рассмотрим назначение выводов микросхемы *(**Pin Description cmp.7**)*:

1 – **VDD** – Плюс питания: 2,7-3,6V (MAX1240); 2,7-5,25V (MAX1241).

2 – **AIN** – Вход аналогового сигнала в диапазоне от 0 до Vref.

3 – **SHDN** – Трехуровневый вход **Shutdown** (отключение). Подача на вход низкого уровня переводит MAX1240/MAX1241 в режим пониженного энергопотребления (около 15мкА). Обе микросхемы полностью работоспособны при высоком уровне или неподключенном выводе **SHDN**. Для MAX1240 подача на **SHDN** высокого уровня подключает внутренний источник опорного напряжения, если оставить **SHDN** неподключенным, подразумевается использование внешнего опорного напряжения.

4 – **REF** – Опорное напряжение для АЦП. Внутренний источник 2,5V для MAX1240 шунтируется на землю конденсатором 4,7мкФ. Вход внешнего опорного напряжения для MAX1241 и MAX1240 (при отключенном внутреннем источнике) шунтируется на землю конденсатором как минимум 0,1мкФ.

5 – **GND** – вывод цифровой и аналоговой земли.

6 – **DOUT** – выход последовательного интерфейса. Данные изменяются по спаду импульса на входе **SCLK**. При высоком уровне на входе **CS** выход **DOUT** находится в высокоимпедансном состоянии.

7 – **CS** – Низкий уровень на этом входе активизирует выбор кристалла. Когда на **CS** высокий уровень, **DOUT** находится в высокоимпедансном состоянии.

8 – **SCLK** – Вход тактовой частоты. Частота импульсов на этом входе может быть в диапазоне до 2,1МГц.

Теперь рассмотрим, как происходит синхронизация и управление **MAX1241** ***(Timing and Control cmp.10)*.**

Старт преобразования и операция считывания данных контролируются с помощью сигналов по входам **CS** и **SCLK**. Временные диаграммы иллюстрирующие данные операции приведены на рисунках 8 и 9 даташита. Первый из них, со своими комментариями я приведу на Рис. 128, поскольку он играет важную роль в понимании дальнейшего материала.

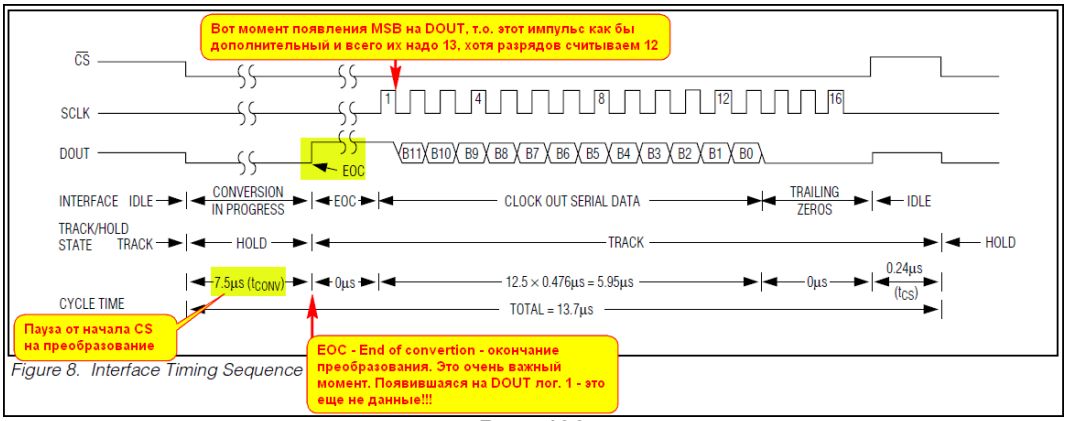


Рис. 128.

Спадающий сигнал на входе **CS** инициирует очередное преобразование: **T/H** запоминает входное напряжение (на внутреннем конденсаторе), АЦП начинает преобразование, а на выходе **DOUT** появляется логический ноль. В течение всего времени преобразования (7,5 мксек) **SCLK** должен удерживаться низким уровнем. Внутренний регистр АЦП в это время запоминает информацию.
Окончание преобразования (**EOC**) отмечается появлением высокого уровня на выходе **DOUT**. Передний фронт **DOUT** может быть использован в качестве сигнала прерывания. Каждым импульсом **SCLK** после окончания преобразования данные сдвигаются из внутреннего регистра АЦП. На **DOUT** они появляются по спаду **SCLK**. Первый спад **SCLK** инициирует появление на выходе **DOUT** бита **MSB**, следующий спад – следующий бит. Таким образом, для вывода всех 12 бит преобразования и одного предшествующего высокого уровня необходимы 13 импульсов. Последующие лишние импульсы до ближайшего изменения **CS** к высокому уровню вызывают появление на **DOUT** дополнительных нулей и не оказывают влияния на операцию преобразования.
Минимальное время цикла получения информации от **ADC** достигается с использованием переднего фронта изменения сигнала на **DOUT** в качестве сигнала **EOC**. Оно составит 12,5 циклов тактового сигнала на максимальной скорости. По окончании считывания младшего бита **LSB** необходимо поднять сигнал **CS** в логическую единицу. Выждав определенное минимальное время (**tcs**=0,24мксек), спадом сигнала **CS** можно инициировать следующее преобразование.
Ну и, пожалуй, последняя нужная нам выдержка из даташита на **MAX1240/MAX1241**, касающаяся подключения к стандартным последовательным интерфейсам (***Connection to Standard Interfaces стр. 11).***
Интерфейс **MAX1240/MAX1241** полностью совместим со стандартными интерфейсами **SPI/QSPI** и **MICROWIRE**. Рисунок 11 в даташите (Рис. 129 здесь).

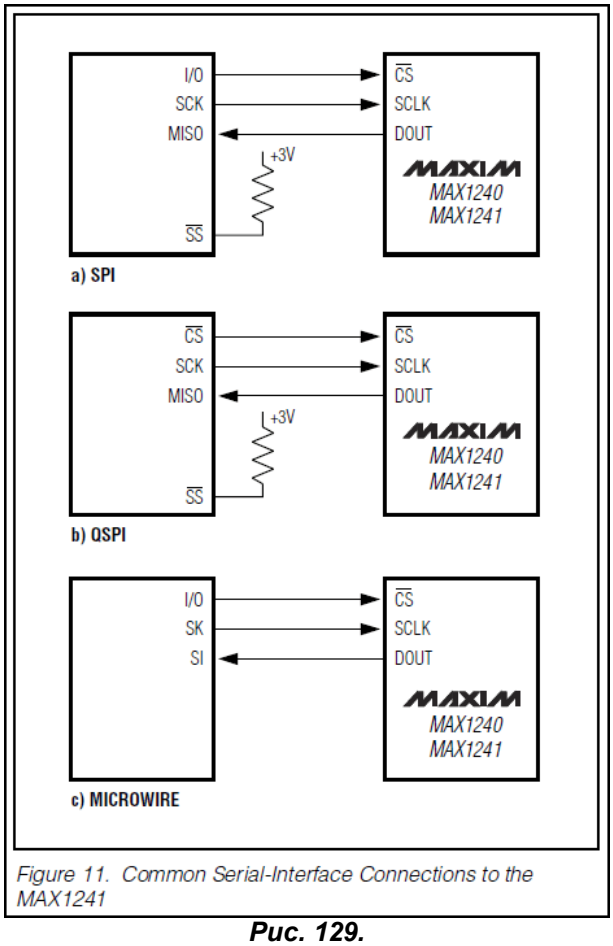


Figure 11. Common Serial-Interface Connections to the MAX1241

Рис. 129.

Если последовательный интерфейс имеется в наличии у вашего ЦПУ *(микроконтроллера - МК)*, установите его в режим «мастер» для генерации тактового сигнала. Выберите частоту тактового сигнала до 2,1 МГц.

1) Используйте стандартные команды ввода/вывода ЦПУ для изменения сигнала **CS** в логический ноль. Сохраняйте при этом **SCLK** низким.
2) Выждите максимально необходимое время преобразования перед активацией сигнала **SCLK**. Альтернативно для определения окончания преобразования можно отслеживать изменение сигнала в логическую единицу на линии **DOUT**.
3) Активируйте **SCLK** минимум на 13 тактов. Первый спад импульса выводит **MSB** цифрового результата преобразования на линию **DOUT**. Изменения выходных данных на **DOUT** происходят по заднему фронту импульсов **SCLK** в стандартном формате **MSB-first** *(старший разряд – первый)*. Соблюдайте при этом допустимые временные характеристики

SCLK. Данные могут быть синхронизированы в МК по переднему фронту *(следующего)* импульса S**CLK**.

4) Установите **CS** высоким уровнем после прохождения 13-ти задних фронтов импульсов на **SCLK**. Если **CS** остается низким, дополнительные нули на **DOUT** будут считаны по каждому лишнему импульсу **SCLK**.
5) Когда **CS=1** выждите минимально необходимое время **tcs** перед началом следующего преобразования (**CS=>0**). Если преобразование было прервано установкой **CS=1** до завершения предыдущего, выждите минимально необходимое время **tacq** перед началом следующего преобразования.

Сохраняйте **CS** низким до тех пор, пока все данные не будут считаны. Данные могут быть считаны двумя байтами или непрерывной последовательностью, как показано на рисунке 8 (даташит, 128-здесь). При считывании двумя байтами данные содержат одну стартовую лишнюю единицу в начале и дополнительные нули *(три нуля)* в конце.

Вот этот последний момент для нас и важен.

Ну, еще маленькие выдержки из даташита по стандартным последовательным интерфейсам. Для всех трех типов интерфейса устанавливается CPOL = CPHA = 0. В отличие от стандартного SPI, требующего считывания 2 отдельных байтов, QSPI использует минимально необходимое количество тактовых импульсов для считывания. На этом, пожалуй, цитаты можно закончить и перейти к практике.

Далеко заходить не будем, возьмем стандартный пример из **CodeVision AVR** для работы с **MAX1241**. Те, кто использует **CodeVision**, могут найти его в папке **Examles\MAX1241**. Для более старых версий он с использованием **AT90S8515**. Я использую версию 1.25.9 – там **ATMEGA8515**. Конечно же, кроме самого проекта **CodeVision**, который необходимо откомпилировать, нам потребуется тестовый проект в Протеусе (Рис. 130). Я опять применил на аналоговых входах собственные модели источников напряжения, просто с ними удобнее здесь работать – все видно наглядно.

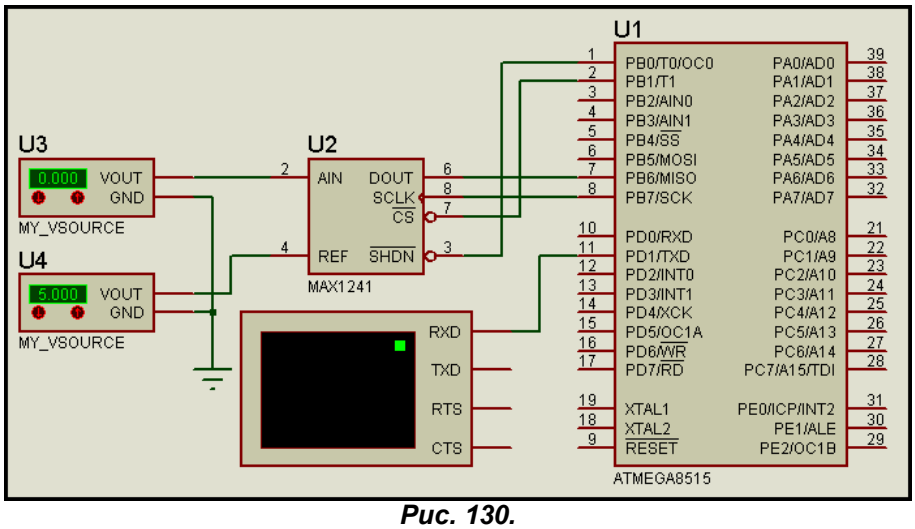


Рис. 130.

Желающие, могут запустить данный проект и убедиться, что нормальное считывание информации наблюдается только при нулевом напряжении на входе **AIN**, дальше начинается полный бред. Проект находится во вложении в папке **CV/MAX1241**. На Рис. 131 график считывания информации из этого проекта. Растянут второй цикл чтения, поскольку первый считает просто нули.

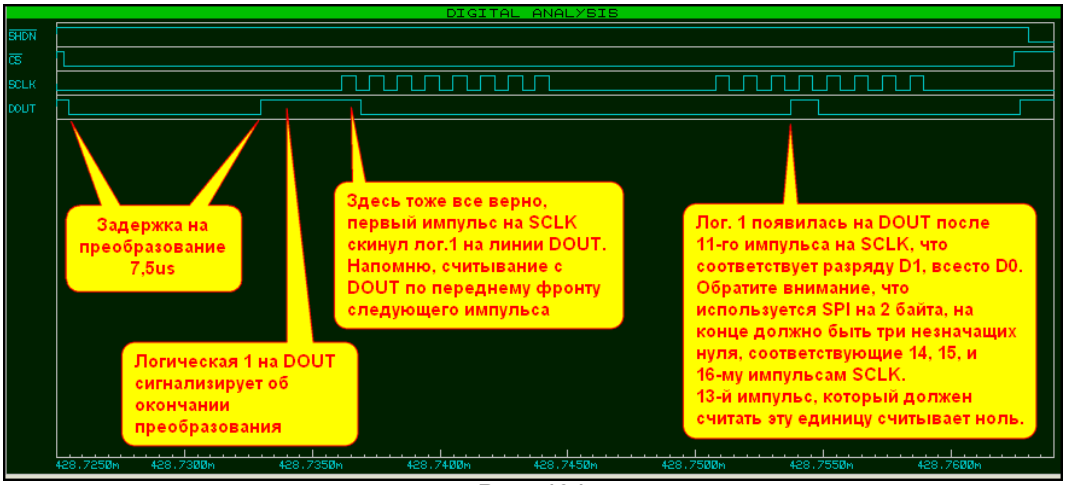


Рис. 131.

Из графика видно, что используется чистый протокол SPI, считывания 2-х байтов. Если заглянуть в проект **CodeVision** – так оно и есть, вначале имеется директива подключения библиотеки SPI:

```
#include <spi.h>
```

А в самом начале программы находится сама процедура считывания:

```
unsigned int max1241_read(void)
{
union adcu adc_data;
// exit MAX1241 from shutdown
NSHDN=1;
// wait 5us for the MAX1241 to wake up
delay_us(5);
// now select the chip to start the conversion
NCS=0;
// wait the conversion to complete
// DOUT will be 0 during conversion
while (DOUT==0);
// DOUT=1 -> conversion completed
// read MSB
adc_data.byte[1]=spi(0);
// read LSB
adc_data.byte[0]=spi(0);
// deselect the chip
NCS=1;
// enter shutdown
NSHDN=0;
// now format the result and return it
return (adc_data.word>>3)&0xfff;
}
```

Обратите внимание на последний оператор возврата. Если его записать в виде:

```
return (adc_data.word>>4)&0xfff;
```

Т.е. сдвинуть лишний раз данные вправо, то все будет работать правильно. Но это в Протеусе, а в «железе» мы получим обратный эффект – все исказится.

Чтобы было нагляднее, я несколько усложнил проект из **CodeVision** и заставил его выводить информацию еще и в двоичном коде. При этом наглядно видно, что при изменении напряжения на **AIN** на **0,001V** вместо разряда **D0** (самый правый) меняется **D1** (второй справа). Таким образом, получается, что данные как бы смещены влево на 1 разряд. Этот пример во вложении называется **MAX1241BIN**.

Ну и чтобы окончательно убедиться, что это не **CodeVision** нам портит картину, а именно модель – считаем информацию с помощью обычных генераторов, как мы это делали с примитивом **SPISLAVE**. Подадим, как велит даташит, 13 импульсов чтения на **SCLK** и посмотрим результат. Этот вариант в примере вложения **GEN_READ**. График из этого примера при чтении с **AIN** напряжения 1mV и VREF=5V приведен на рисунке 132.

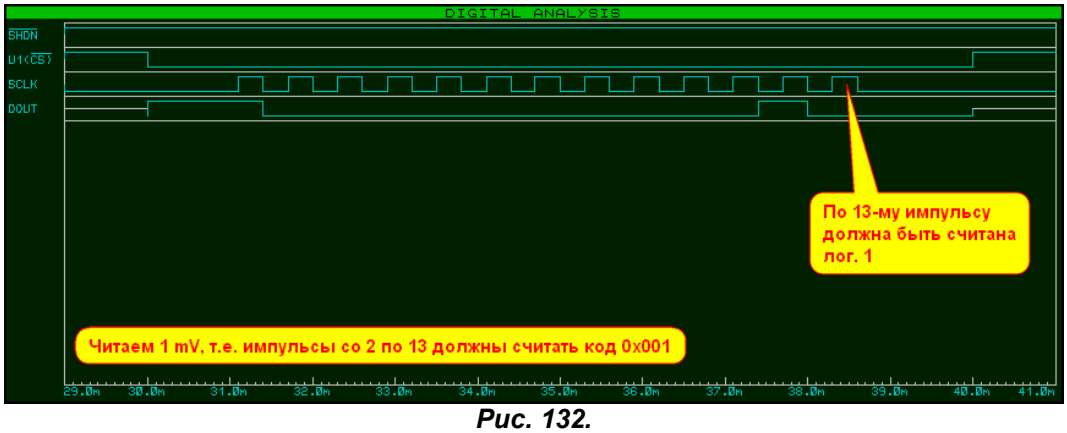


Рис. 132.

Поскольку чтение происходит с тем же дефектом, можно констатировать окончательно и бесповоротно, модель **MAX1241** глючит, но дело поправимое, т.к. организована она чисто с помощью подсхемы, с которой мы познакомимся далее и попробуем поправить положение.

Возврат к содержанию

6.17. MAX1241 Schematic Model – взгляд изнутри. Ищем и исправляем ошибку моделей MAX1241 и MAX1240.

Пора перейти к рассмотрению непосредственно модели **MAX1241**. Для этого нам потребуется извлеченный, как и ранее с помощью **GETMDF** файл **MAX1241.MDF**. Расположен он в **MODELS\MAXIM.LML**. Процедура извлечения мной уже не раз описывалась, останавливаться не буду. Для особо ленивых просто приложу извлеченный файл во вложении. Далее идем стандартным путем и по этому файлу начинаем восстанавливать изначальную схему, с которой

формировался MDF. И тут выясняется «приятная» неожиданность в разделе **PARTLIST** мы встречаем следующую строчку:

I1,SPIIO,SPISLAVE_12,MODDLL=SPIIO.DLL,PRIMITIVE=DIGITAL,SPI_AUTOLOAD=0,SPI_CPHA=0,SPI_CPOL=0,SPI_DORD=0

Но в библиотеке нет двенадцатиразрядного **SPISLAVE_12** – что делать? Ответ прост – создать. Все дело в том, что когда программист Лабцентра писал библиотеку **DLL** для модели **SPISLAVE**, то, конечно же, сделал ее универсальной. Посмотрите модели **SPISLAVE_8** и **SPISLAVE_16**. Обе они привязаны к **SPIO.DLL**. Представим себя тоже немного англичанами. Причем конкретно проживающими на Бейкер-стрит, курящими трубку и играющими по вечерам на скрипке. Раз есть 8 и 16, то почему не быть 10, 12 или 14, да можно даже 11 и 13. Проверив все свойства уже существующих примитивов, можно «дедуктивным методом» прийти к выводу – все отличия только в разрядности шин, ну и еще в имени модели – это обязательное условие. Пробуем применить на практике. Берем любую из существующих, например **SPISLAVE_8**, втаскиваем в проект и … молотком (**Decompose**). Получился набор «Сделай сам» (Рис. 133).

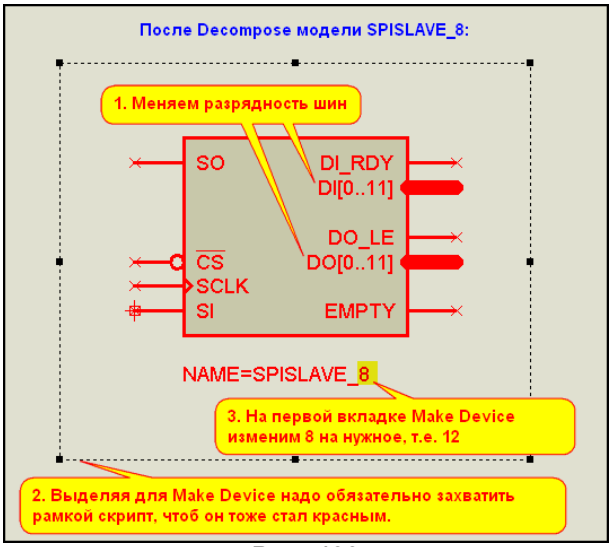


Рис. 133.

С этим набором поступаем следующим образом: меняем разрядность шин на нужную (для 12 - это с 0..11); выделяем, удерживая нажатой левую кнопку мыши, всю графику и в том числе текстовый скрипт, начинающийся с **NAME=**; нажимаем в меню **Make Device** и на первой вкладке меняем в графе **Device Name** имя **SPISLAVE_8** на **SPISLAVE_12**. После проходим процедуру создания модели до конца и сохраняем ее в библиотеке моделей, можно даже в **USRDVC**, так как нужна она нам будет временно, только для создания модели. Аналогично можно создать и 13-ти и 14-ти разрядный **SPISLAVE**. Как протестировать получившуюся модель я уже показал в **п.6.15**. Кстати, такие метаморфозы можно проделывать не только со **SPIO.DLL**, но и с некоторыми другими программными моделями, например, с теми же **ADC** и **DAC**. В этом мы убедимся чуть позже.

Ну а теперь у нас в распоряжении полный набор примитивов для воссоздания структуры модели АЦП. Процесс воссоздания ничуть не отличается от того, что мы делали раньше. Набираем в соответствии с разделом **PARTLIST** файла MDF нужные примитивы и соединяем их между собой в соответствии с **NETLIST**. Поскольку в данном случае MDF достаточно объемный, **NETLIST** содержит 53 цепи, процедура воссоздания длительная и требует внимательности. Могу порекомендовать простой способ, которым я пользуюсь в таких случаях. Текстовое содержание MDF копируем в MS WORD или любой другой редактор, поддерживающий расцветку текста. Сразу же определяемся с тем, что разводить не надо и подсвечиваем каким либо цветом. В данном случае это будут цепи, содержащие только один вывод примитивов. Такие цепи я подсветил зеленым, чтобы в процессе восстановления схемы не обращать на них внимание. Остальные цепи, по мере их прорисовывания, я постепенно подсвечиваю красным. Это удобно еще и тем, что если Вас оторвали от этого занятия, на определенном этапе можно все сохранить и продолжить в другом месте и в другое время. Впрочем, это уже из серии «бесполезных советов». Вернемся к структуре **MAX1241**. Восстановленная структура находится в проекте вложения **MAX1241_Part2\Structures\Structure_MAX1241.DSN**. Там же лежат оригинальный **MAX1241.MDF** и «расцвеченный» **MAX1241_MDF.DOC** (по которому она восстанавливалась). Я не стал помещать на лист только скрипт:

```
*MODELS
MAX1241 : RHI=100,RLO=10,VUD=2,VTL=0.8,VHL=0.2,VTH=2.5,VHH=0.2,V+=VDD,V-=GND
```

Он при компиляции превращается в раздел ***MODELDEFS**. Пока он нам не нужен, но при создании нового «рабочего» MDF понадобится. Результат моего творчества представлен на Рис. 134. Пришлось поместить его полностью и с достаточно большим разрешением, поскольку схема требует некоторых комментариев. Итак, что тут к чему относится и для чего служит. Приемы моделирования пригодятся вам в дальнейшем.

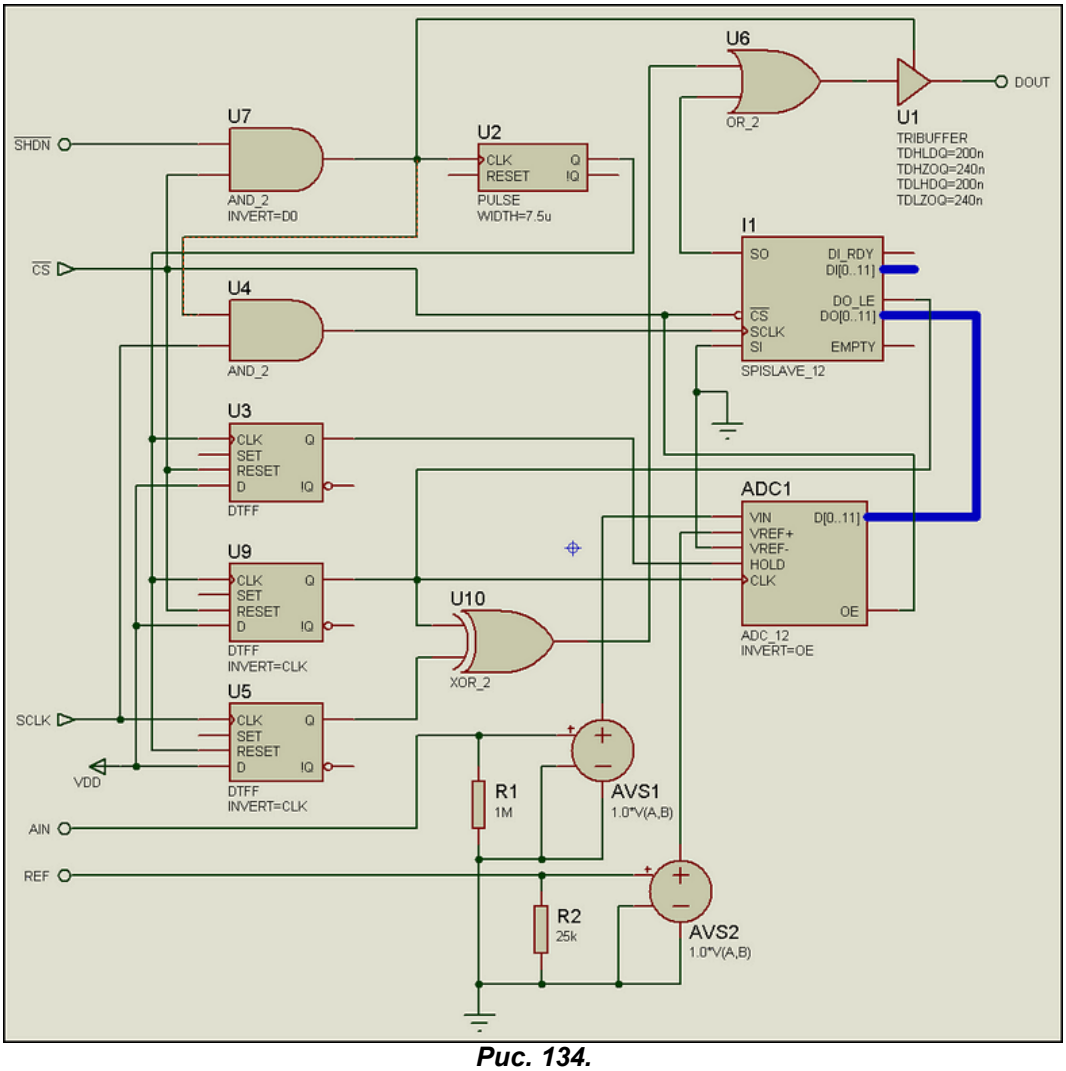


Рис. 134.

Начнем с аналоговой части. Для гальванического разделения по входам **AIN** и **REF** применены примитивы AVCVS **AVS1** и **AVS2** с коэффициентом передачи единица. На их входах помещены соответствующие резисторы, обеспечивающие имитацию входных сопротивлений по этим входам. С выходов **+** AVCVS аналоговые сигналы поступают на аналоговые входы двенадцатиразрядного АЦП **ADC1**, вход **VREF-** которого заземлен. На этом аналоговая часть кончается.

Сигнал с выходной шины **D[0..11]** элемента **ADC1** напрямую соединен с шиной **DO[0..11]** двенадцатиразрядного (**!!!**) примитива **I1**. Такое присоединение (без дополнительных меток и т.п.) ISIS интерпретирует как – «разряд в разряд». Т.е. **D0 ADC1** соединен с **DO0 I1**, **D1** с **DO1** и т.д. до **D11**–**DO11**. Это очень важно, и этим мы воспользуемся для исправления.

Далее сигнал с выхода **SO** SPISLAVE_12 в последовательном коде через элемент **U6** и буфер с тремя состояниями выхода **U1** отправляется на выход модели **DOUT.** Таков путь преобразования входного аналогового сигнала в цифровую форму в модели **MAX1241**.

Теперь о назначении вспомогательных узлов и элементов. Элемент **U7** обеспечивает запуск одновибратора **U2**, формирующего задержку 7,5 мксек при появлении на входах **SHDN** и **CS** разрешающих сигналов. Эта задержка через элемент **U4** блокирует прохождение тактового сигнала **SCLK** на вход **I1**, имитируя стандартную задержку на процесс преобразования **tconv** (см. рис. 128).

Триггер **U3** обеспечивает удержание преобразованного сигнала в цифровой форме на выходе **ADC1** в процессе одного цикла обращения к микросхеме (активность **CS**).

На триггерах **U9**, **U5** и элементе **U10** собран формирователь единичного импульса на время окончания преобразования и первый тактовый импульс (**!!!**) на выходе **DOUT**. Вот здесь и «зарыта» ошибка разработчика модели.

Я скомпоновал структуру **MAX1241** в виде модуля и разместил в стандартный проект CodeVision, который мы рассматривали ранее. Этот пример во вложении:

MAX1241_Part2\Bad_Test_Module1241\TEST_Structure_12razr.DSN

Растянутый график второго цикла преобразования (первый дает нулевой результат) представлен на рисунке 135. Обратите внимание на широкий единичный импульс – сигнал окончания преобразования на выходе **DOUT**. Он перекрывает первый тактовый импульс **SCLK**. Но первый же тактовый импульс проходит и на тактовый вход **SPISLAVE_12** – трасса **U4(Q).** Поскольку у нас модель **I1** двенадцатиразрядная, этот импульс соответствует старшему биту MSB (**D11**) интерфейса. Как говорят врачи реаниматоры: «мы теряем его…». Это нетрудно доказать. Если подать на вход **AIN** половинное напряжение от **REF** должен оказаться заполненным единицей старший разряд **D11** АЦП, т.е. двоичный код выглядит так: **1000 0000 0000**. Но на деле, в тестовом

проекте, упомянутом выше, на терминал будет выводиться нулевое напряжение – реаниматоры оказались правы.

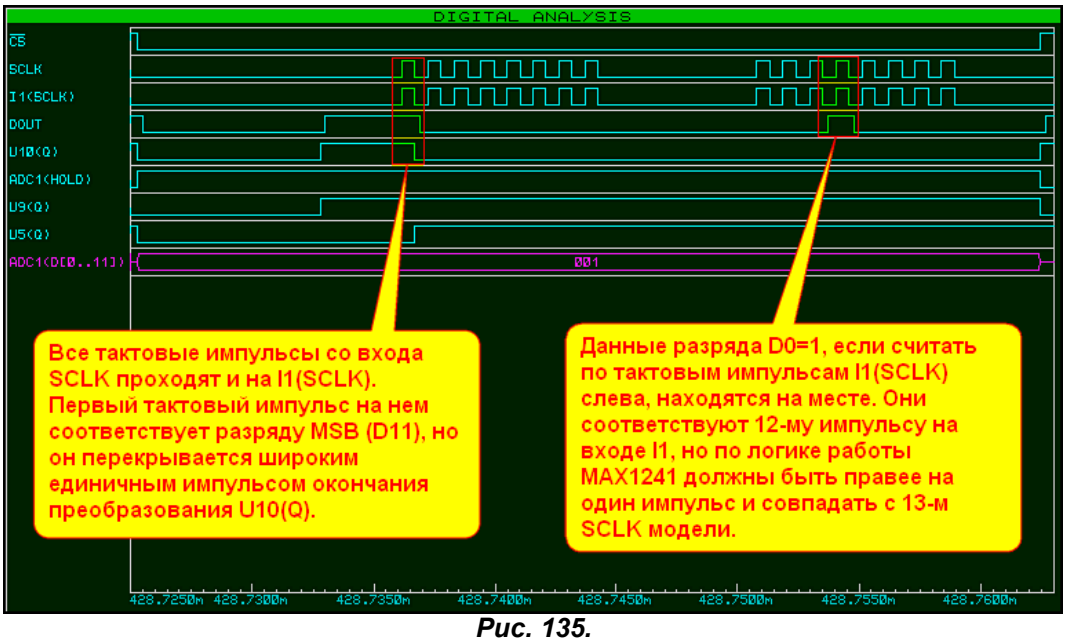


Рис. 135.

Ну что же, причина ясна. Пора приступать к хирургическому вмешательству. И тут помогает простейшая аптечная «свинцовая примочка». Помните, чуть выше я упоминал, что SPISLAVE может быть и 13-ти разрядным, а при описании структуры указал, что данные по шине передаются «разряд в разряд». Конечно, если с 12-ти разрядного АЦП передать данные в 13-ти разрядный SPI, то старший 13-й разряд последнего окажется незаполненным. Но он нам особенно и не нужен, его благополучно «скушает» импульс окончания преобразования, как он проделывал это с 12-м разрядом ранее. И, как в старом бородатом анекдоте, «пусть хомяк подавится». Проверяем на деле. В проекте вложения:
Good_Test_Module1241\TEST_Structure_13razr.DSN
представлена данная замена. Надеюсь, повторяться о том, как сделать **SPISLAVE_13** не надо. Тестируем проект и убеждаемся, что все встало на свои места (Рис. 136).

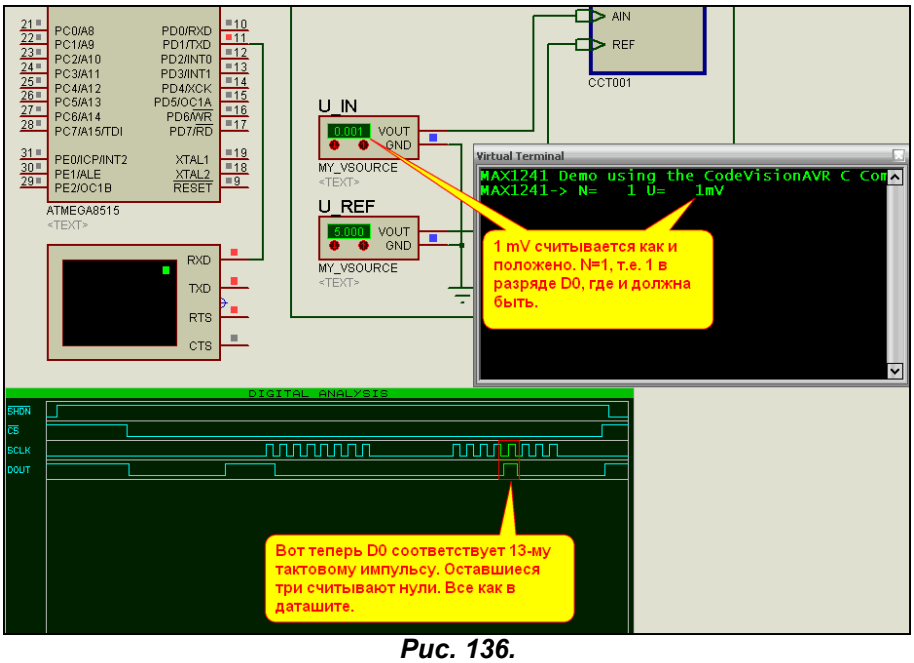


Рис. 136.

Теперь осталось восстановить «статус-кво» в самом Протеусе. Создаем проект с **MAX1241**, привязываем к МАХу дочерний лист (в свойствах ставим галочку **Attach Hierarchy Module**), устанавливаем дополнительно галочку **Edit all properties as text** и временно удаляем строчку:
{MODFILE=MAX1241.MDF}
Процесс представлен на рисунке 137.

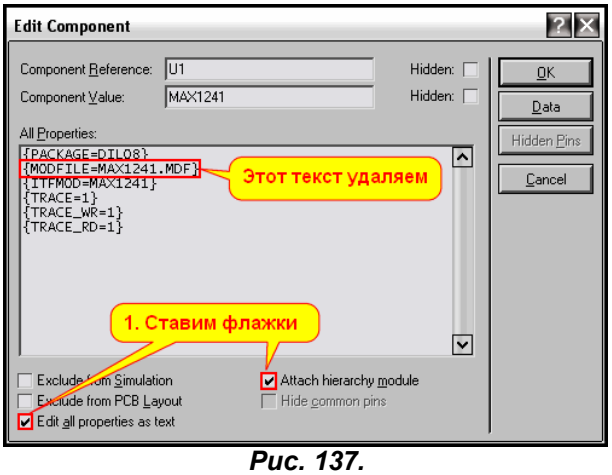


Рис. 137.

На дочернем листе располагаем восстановленную структуру (Рис.134), скопированную с дочернего листа **TEST_Structure_13razr.DSN**. Там у нас уже 13-ти разрядный SPI. Не забудьте туда же поместить тестовый скрипт ***MODELS**, который приведен ранее.

Есть еще один нюанс, который прояснился в последний момент. Вероятно, модели **MAX1240/MAX1241** разрабатывались не сотрудниками Лабцентра, а сторонним пользователем. Дело в том, что «шапка» MDF полностью заполнена, а в графе **Author** стоит **EA**. Схематичные модели, разработанные в самом Лабцентре, как правило, в шапке имеют только дату. Так вот этот самый **EA** в графической модели обозначил инвертированные выводы знаком доллара **$** не с двух сторон, как я привык и объяснял где-то вначале FAQ, а только спереди, т.е. **$SHDN** и **$CS**. Поэтому на дочернем листе одноименные терминалы надо привести в соответствие, иначе полезут ошибки. Перед компиляцией нового MDF модель можно протестировать. Вариант с дочерним листом во вложении **New_Model_MAX1241\With_Child\ 1241_Wtith_Child.DSN**.

После того, как мы убедились, что все работает и при подаче на вход напряжения 1mV импульс разряда **D0** в графике встал, где положено, т.е. 13-м по счету с дочернего листа через меню **Tools=>Model Compiler** компилируем новый **MAX1241.MDF**. Тест с новым MDF во вложении **New_Model_MAX1241\With_New_MDF\ 1241BIN_new.DSN**.

Далее можно пойти тремя путями:

- Первый и самый простой вариант. Новый файл MDF переименовываем, например, в **MAX1241N.MDF** и помещаем в папку **MODELS** Протеуса. Предварительно сняв защиту от записи с помощью **Library=>Library Manager** (Рис. 138), помещаем в проект **MAX1241** и запускаем для него **Make Device**. На третьей вкладке для **MODFILE** в графе **Default Value** задаем наш новый MDF и в последней вкладке сохраняем его не **USRDVC**, а в **MAXIM**, который теперь доступен для записи. После этого можно снова защитить библиотеку, повторив процедуру через **Library Manager**.

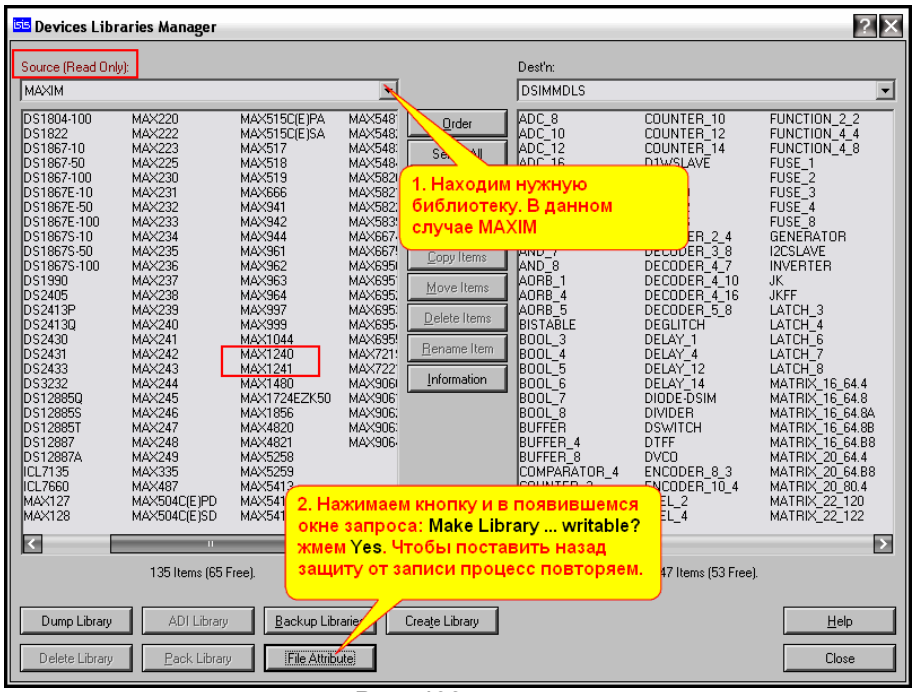


Рис. 138.

- Второй вариант – более сложный, но корректный с точки зрения Протеуса. Файл **MAXIM.LML** (не забудьте сохранить резервную копию на всякий пожарный случай!) из **MODELS** Протеуса копируем в отдельную папку и туда же помещаем утилиты **GETMDF.EXE** и **PUTMDF.EXE** из папки **BIN**. Запускаем командную консоль для этой папки и выполняем:
  GETMDF.EXE –L=MAXIM.LML –D MAX1241
  При этом ключом **–D** (delete) модель стирается из библиотеки и дополнительно извлекается в нашу папку в виде файла **MAX1241.MDF**. Его нужно удалить – это старый вариант. Правда, термин «стирается» – это слишком громко сказано. На самом деле, стирается только в бинарном заголовке файла LML. Но, если запустить поиск по ключу MAX1241, то сам текст старого MDF в библиотеке LML мы обнаружим. Впрочем, нам он не мешает, пусть живет. Затем помещаем новый файл (я приложу их в папке **New_MDF_MAX1240_MAX1241** вложения) в нашу папку и выполняем в командной консоли следующую процедуру:
  PUTMDF.EXE –L=MAXIM.LML MAX1241.MDF
  Наша новая MDF модель приплюсуется в конце библиотеки, а в заголовке LML появится ссылка уже на нее. Теперь можно **MAXIM.LML** вернуть на старое место в **MODELS**. Протеус будет работать уже с новой моделью. Корректное выполнение процедуры показано на рисунке 139.

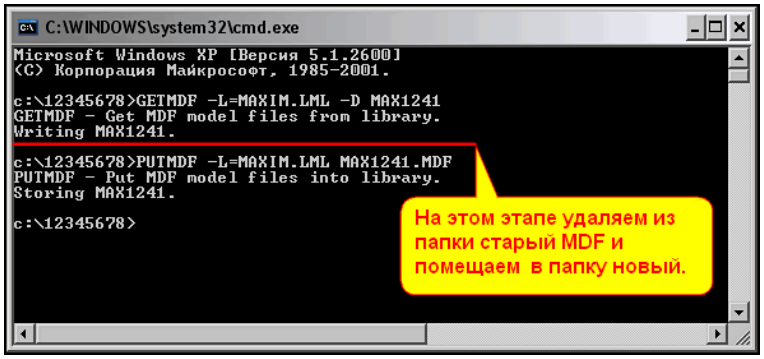


Рис. 139.

- Наконец, третий вариант – самый трудоемкий. Полная перекомпиляция библиотеки **MAXIM.LML** с заменой моделей **MAX1241** и **MAX1240** (о ней ниже) на корректные новые. Для этого придется активно поработать с утилитами командной строки, поскольку библиотека содержит 84 модели. Процесс схож с предыдущим вариантом и заключается в следующем. Сначала с помощью **GETMDF** извлекаются все MDF из библиотеки командой:
  GETMDF –L=MAXIM.LML –A
  Затем убираем из папки саму библиотеку, заменяем **MAX1241.MDF** и **MAX1240.MDF** новыми и создаем новую библиотеку **MAXIM.LML**, например, на 100 элементов командой:
  PUTMDF –L=MAXIM.LML –C=100
  Далее через **PUTMDF** постепенно (порциями по несколько штук, чтоб не запутаться) добавляем в нее все присутствующие в папке MDF. При этой операции удобно пользоваться ключом **–D**, который при добавлении моделей в библиотеку будет одновременно и удалять их MDF из папки. Допустим, мы добавляем в библиотеку четыре модели: **DG417.MDF**, **DG418.MDF**, **DG419.MDF** и **DG508.MDF**. Тогда строка будет выглядеть так:
  PUTMDF –L=MAXIM.LML –D DG417.MDF DG418.MDF DG419.MDF DG508.MDF
  На рисунке 140 показан процесс создания **MAXIM.LML** на 100 «посадочных мест» и процесс добавления моделей по 4 штуки в строке. Напомню, что в окне командной консоли можно повторно выбирать предыдущие выполненные команды с помощью клавиши навигации «стрелка вверх». Просто вызываем каждый раз предыдущую команду и в ней перебиваем список добавляемых моделей, после чего давим **Enter**. Ну и еще, как вы поняли, и я упоминал ранее – расширения файлов можно не набирать, достаточно только имен.

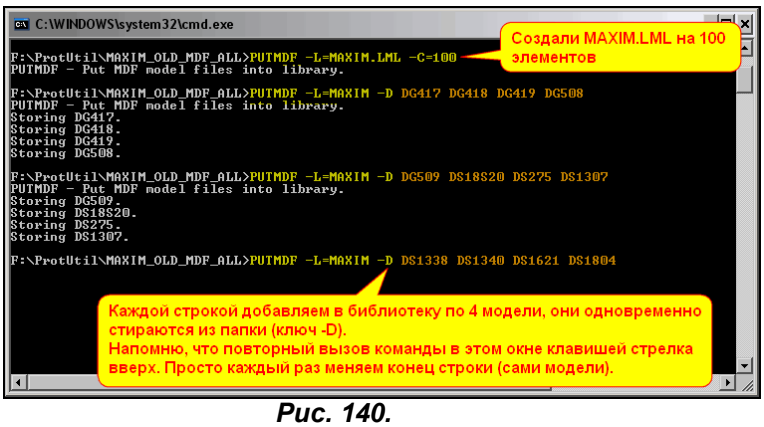


Рис. 140.

Теперь немного о **MAX1240**. Эта модель имеет схожую структуру, но отличается тем, что в ней используется внутренний источник опорного напряжения 2,5V, который подключается при условии, что на входе **SHDN** присутствует высокий уровень, либо он не подключен. Естественно, в модели присутствует и та же ошибка, что в **MAX1241**. Восстановленная с MDF структура **MAX1240** находится во вложении в файле **Structures\Structure_MAX1240.DSN**. Подробно я останавливаться на этой модели не стану, только добавлю, что в папке **Good_Test_Module1240** находится тест с уже 13-ти разрядным SPISLAVE (соответственно изменена программа CV под 2,5V), а в папке **New_Model_MAX1240\With_Child** проект, с дочернего листа которого скомпилирована новая модель. Сам новый **MAX1240.MDF** лежит в папке **New_MDF_MAX1240_MAX1241**.
В папке **New_LML_lib** лежат полностью пересобранные по третьему способу **MAXIM.LML** для версий 7.6 и 7.7 с исправленными моделями **MAX1240** и **MAX1241**.

Возврат к содержанию

6.18. Создаем модель АЦП ADS1286 от Burr-Brown, или LTC1286 от Linear Technology.

Разбираться в чужих моделях и искать ошибки дело конечно нужное, но иногда хочется и творческого полета собственной мысли. Так вот, разборка с **MAX1241** натолкнула меня на идею сваять другую, не менее популярную модель 12-ти разрядного АЦП **ADS1286**. Тем более что он полностью совместим с **LTC1286**, так что «пристрелим двух ушастых» одним выстрелом. Это послужит некоторым закреплением пройденного материала, ну и лишний АЦП в хозяйстве пригодится. Анализ протокола обмена из даташита на **ADS1286** показал, что он незначительно отличается от **MAX1241**, т.е. часть структуры последнего можно «принять за основу», как модно было выражаться на различных собраниях в эпоху недоразвитого социализма. На рисунке 141 приведена диаграмма обмена по последовательному интерфейсу.

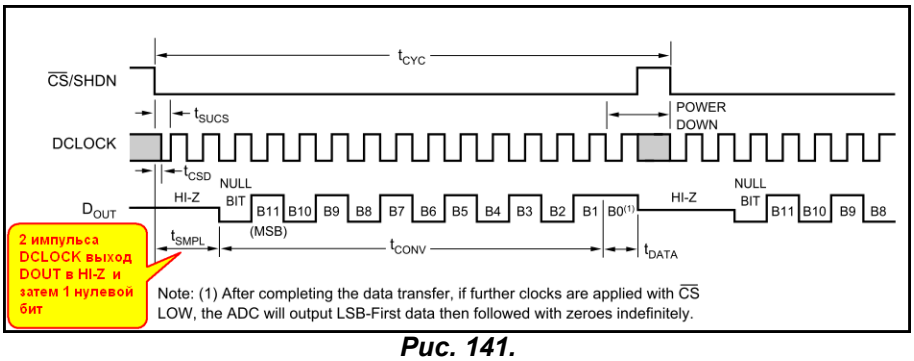


Рис. 141.

Рассмотрим основные отличия от **MAX1241**. После появления низкого уровня на входе выбора кристалла **CS** два тактовых импульса (время t_{SMPL}) выход **Dout** находится в высокоимпедансном состоянии. Затем следует один нулевой бит и далее 12 бит оцифрованного сигнала, где, как и у **MAX1241** MSB следует первым. Это и будет для нас основным фактором отличия. По большому счету вторая диаграмма (**FIGURE 1** даташита) показывает, что если **CS** и далее будет оставаться низким, и будут следовать тактовые импульсы, то возможно считывание данных в обратном порядке, пока АЦП не «заснет», но та же оговорка в сноске предупреждает, что там могут быть, и считаны и нули. Нам это не так важно в нашей модели. Поскольку играться с всякими неопределенностями себе дороже, практической ценности то, что будет правее бита **B0,** для нас не представляет, и эту информацию лучше не использовать. Структура АЦП приведена на Рис. 142.

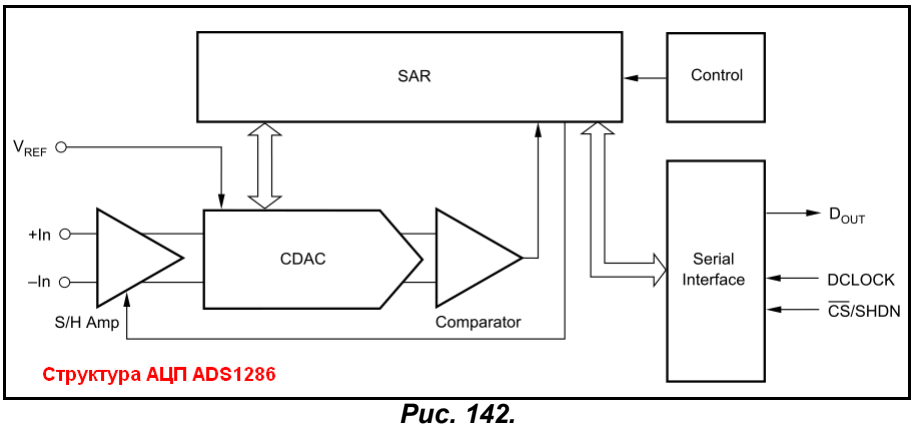


Рис. 142.

Давайте попробуем спроектировать структуру нашей модели. Связку **ADC_12** и **SPISLAVE_13** пока оставим в неприкосновенности. Останется и **TRIBUFFER**, так как нам необходим выход **Dout** с тремя состояниями. А вот всю остальную входную логику, формирующую управление будем менять. Нам необходимо отсечь первые два тактовых импульса, которые не должны изменять состояние

выхода АЦП и уж тем более влиять на сдвиг данных в последовательном интерфейсе. В этом поможет обычный счетчик на двух D-триггерах. Потребуется и несколько логических элементов совпадения по И. А вот с тем стартовым нулевым битом можно применить трюк, который мы использовали в **MAX1241**. Вспомните, модель **ADC** – 12-ти битная, а **SPISLAVE** – 13-ти. При этом старший, незадействованный 13-й бит будет читаться нулем, если мы специально не закинем в него единицу. Вот это нам и надо. Именно поэтому в данном случае оставляем модель **SPISLAVE_13**. Кроме того, подвергнется небольшой переделке и та часть модели, которая относится к аналоговым входам. Мы видим, что V_{REF} в данном случае подается относительно земли, зато входной сигнал **+In** и **–In** является дифференциальным. На рисунке 143 приведена схема получившейся у меня модели.

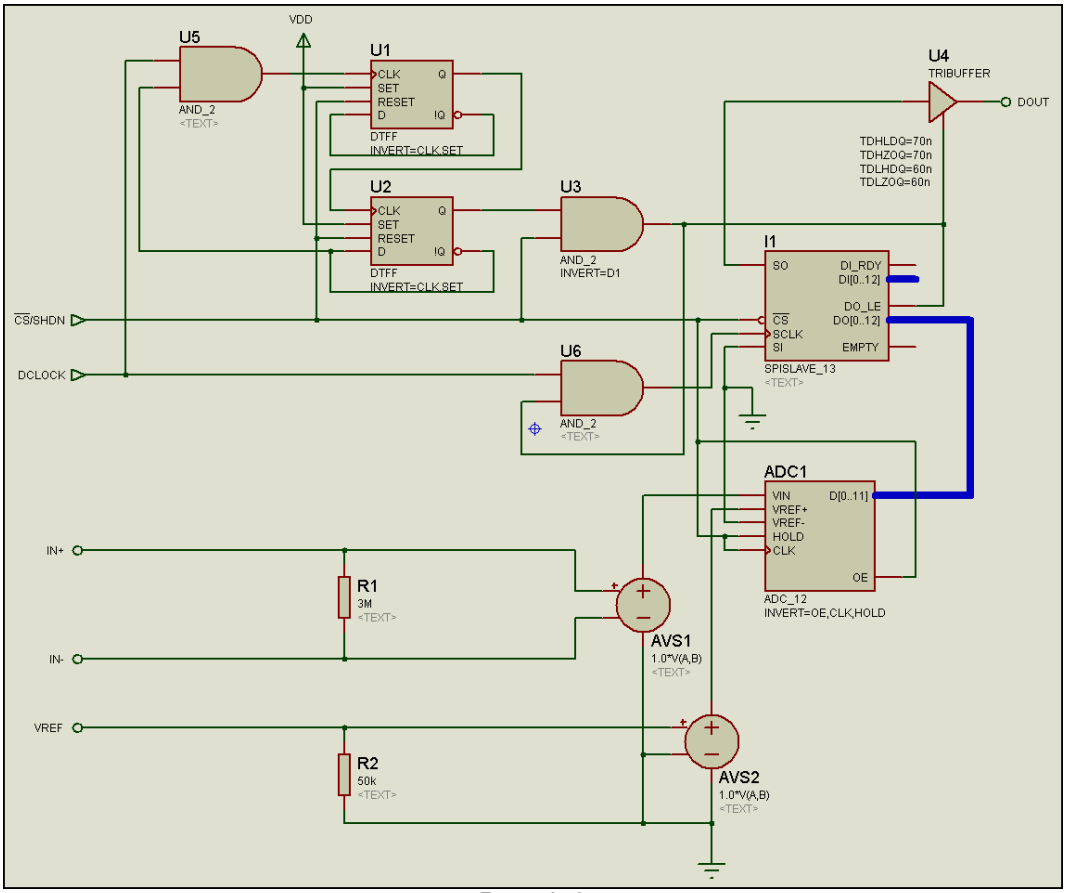


Рис. 143.

Рассмотрим ее несколько подробнее. Появление низкого уровня на входе **CS/SHDN** по входам CLK, HOLD и OE «запирает» результат аналого-цифрового преобразования на выходной шине **ADC1**. На триггерах **U1** и **U2** собран счетчик, который отсекает первые два тактовых импульса. Установка их идет по заднему фронту (в свойствах **INVERT=CLK**). После установки триггера **U2** через элемент 2И **U5** дальнейший счет блокируется до следующего появления на входе **CS/SHDN** высокого уровня, который сбросит триггера счетчика. Элементом 2И **U3** при этом активируется выходной буфер **U4** (снимается третье состояние), а через элемент **U6** разрешается прохождение тактовых импульсов на вход SCLK интерфейса **I1**. Поскольку интерфейс 13-ти разрядный, первым проследует старший разряд, в который ничего не заносилось – получится требуемый нулевой стартовый бит, а затем 12 разрядов с выходной шины **ADC1** в порядке MSB - первый. Как видите, с этой точки зрения структура получилась даже проще, чем у **MAX1241**. В аналоговой части по-прежнему стоят два гальванических разделителя **AVS1** и **AVS2**. Только теперь у **AVS2** минусовой вход заземлен (VREF подается относительно GND). Входное сопротивление по этому входу имитируется R2=50кОм. Переход в высокоимпедансное состояние при неактивном («спящем») АЦП я не стал имитировать, т.к. это неоправданно усложнило бы модель. Входы же аналогового сигнала **+In** и **–In** связаны через резистор R1=3МОм. Он рассчитан, исходя из входного тока около 1,5мкА при 5В (см. даташит раздел: **RC INPUT FILTERING**).

Теперь немного о том, что во вложении. В папке **Test_structure_module** приложены два варианта тестирования полученной структуры. С помощью обычных генераторов – **Generators** и проект с CodeVision в одноименной папке. В последнем проекте для того, чтобы считанное **N** совпадало с расчетным принято директивой **#define VREF 4096** в начале файла на Си и для вывода применен LCD 2x16. В папке **Test_with_child** уже графическая модель **ADS1286**, но с дочерним листом – с него потом скомпилирован **ADS1286.MDF**. Ну, и наконец, в папке **Test_with_MDF** тестовый проект с готовым MDF, он лежит в этой же папке.

Для использования в собственных проектах достаточно переложить **ADS1286.MDF** в папку **MODELS** Протеуса, а из проекта **Test_ADS1286_MDF.DSN** запустить и пройти до конца **Make Device** для модели **ADS1286**. На последней вкладке определитесь с библиотекой, в которой она будет

храниться. Аналогично можно сделать и модель **LTC1286** (не путайте с **LTC1298**, даташит у них единый, но они отличаются), просто переименовав на первой вкладке имя модели, а на последней изменить производителя. Поскольку они по логике работы полностью совпадают, файл MDF для них используется один и тот же. Ну и наконец, в папке **DATASHEETS** вложения даташиты на **ADS1286** и **LTC1286**, которыми я руководствовался при создании модели.

Возврат к содержанию

Заключение к части III

Материала в этой части получилось достаточно много, но надеюсь, что он вызвал определенный интерес не только у новичков в освоении Протеуса, но и у давних пользователей этого программного продукта. Я сознательно не стал углубляться в подробное изложение основ SPICE-моделирования, поскольку этой теме посвящено достаточно много литературы других авторов. Но применение SPICE-моделей, заимствованных из других программ и у производителей компонентов было рассмотрено выше. Не знаю, насколько мне это удалось, но главное, чего я добивался при написании этой части, чтобы пользователи освоили создание пусть примитивных, но своих собственных моделей с помощью применения подсхем, а также больше внимания уделяли свойствам моделей. Конечно, материал не претендует на стопроцентное рассмотрение всех особенностей Протеуса, но по мере изложения я стараюсь в примерах вводить инструментарий и приемы, которых не касался в изложенном материале. Вдумчивый пользователь самостоятельно сможет использовать это в своих разработках, ну а кто привык «с шашкой наголо»… извините. Заранее предупреждаю, что «секретной кнопочки» - нажал, и все само заработало, здесь нет. Моделирование – процесс творческий, и порой времени на то, чтобы на экране компьютера все заработало так, как и в реальном устройстве тратится больше, чем для отладки реального устройства. Причем, если там иногда достаточно только познаний в области электроники и не совсем кривых рук, то при компьютерном моделировании надо быть еще и программистом по призванию и в совершенстве знать особенности поведения конкретной программы в том или ином случае, а также грамотно оперировать свойствами моделей. Именно это я и стремился довести в вышеизложенном материале. В следующей части мы рассмотрим активные модели – наиболее интересную «изюминку» Протеуса, а также немного поучимся программировать модели на встроенном языке EASYHDL. Но, заранее предупреждаю, без навыков создания графических моделей, которые я неоднократным повторением в этой части старался «вдолбить» до уровня подсознательных действий, освоение следующего материала фактически невозможно.

FAQ (ЧаВО) по PROTEUS для начинающих и не только.
ЧАСТЬ IV. PROTEUS для фанатов – продолжение.

Содержание:

7. Активные модели.

7.1. «Всё смешалось в доме Облонских…» или небольшое лирическое отступление о том, что считать активным в Протеусе.

Когда то в очень отдаленном прошлом, в эпоху диодов серии Д2 и транзисторов П4Б было принято называть активными компоненты (или как тогда гласила терминология радиоэлементы), которые производят над электрическим сигналом определенные преобразования: усиление, выпрямление и т.п. и т.д.. Однако, прогресс не стоит на месте, и в нынешнюю эпоху повальной компьютеризации несознательного населения устоявшиеся термины приобретают совсем иное значение. И некогда считавшийся пассивным элемент, тот же резистор может с успехом при компьютерном моделировании оказаться намного «активнее» самой навороченной микросхемы. Давайте сразу расставим все на свои места, чтобы не путаться с терминологией в дальнейшем материале.

В случае компьютерного схемотехнического моделирования, как и в любой другой компьютерной программе под активностью подразумевается поведение объекта. Вспомним те же элементы **ActiveX**, применяемые в **Windows** – ползунки, движки, всплывающие меню, прогресс-бары и прочую мерцающую мишуру. Именно их поведение на экране и возможность пользователя активно вмешаться в процесс: сдвинуть, увеличить или уменьшить определяет их как активные. С той же точки зрения надо рассматривать и активные модели в схемотехническом моделировании и в частности в Протеусе. Это те модели, которые либо меняют свое отображение на экране – индикаторы, либо позволяют вмешаться в процесс симуляции – модели, имеющие актюаторы. Вот их мы и будем далее рассматривать в этом разделе.

В **ISIS**, по сравнению с другими пакетами моделирования, и сейчас достаточно много уже существующих активных моделей, способных удовлетворить даже привередливого пользователя. Тут и LED и LCD индикаторы и всевозможные кнопки, переключатели, реле и даже различные датчики физических величин и моторы. Но нашему пытливому русскому уму этого мало. В этом разделе мы будем вести себя как малые дети – попытаемся заглянуть, а что внутри у заморской куклы Барби. Все переломаем и соберем по своему, тут главное не напороть горячки и не «пришпандорить к гульфику рукав», как в известной интермедии Аркадия Райкина. Поэтому, призываю быть особенно внимательными при повторении или самостоятельном «изготовлении» активных моделей. Порой пропущенная буква или неправильно поставленный маркер могут привести к совершенно непредсказуемым результатам и полной неработоспособности вашего творения. Поэтому этот раздел действительно для фанатов, способных часами «отлаживать» поведение модели на экране компьютера. На наиболее значимых моментах я постараюсь останавливаться неоднократно, как делал это ранее с той же процедурой **Make Device**, которая по-прежнему останется для нас основной и самой применяемой. В основном, рассматриваемый далее материал полный «эксклюзив», давшийся мне, как говорится – «потом и кровью». Нигде в HELP Протеуса и сторонних публикациях это не описано и найдено методом «ненаучного», но все же интуитивно предсказуемого «тыка». На этом мое лирическое отступление заканчивается и начинается, надеюсь самый интересный и полезный раздел FAQ по Протеусу.

К содержанию

7.2. Снова в 2D графику. Графические символы – основа активных моделей. Маркер ORIGIN – Архимедова «точка опоры» для активной графики.

До сей поры, мы использовали ту часть левого меню, которая относится к **2D** или, по-русски говоря плоской двухмерной графике, поскольку трехмерная в **ISIS** не предусмотрена, только для прорисовки графического изображения модели. Надеюсь, больших затруднений рисование линий, прямоугольников и кругов у вас не вызвало. Несколько сложнее нарисовать с помощью **2D Graphic Closed Path** замкнутую многоугольную фигуру, ну и практически совсем невозможно изобразить замкнутую криволинейную – нет, к сожалению, такой опции в графическом арсенале **ISIS**. Ну, что-же, будем довольствоваться тем, что есть. В конце концов это не **AutoCAD** и не **PhotoShop**, у Протеуса совсем другие задачи и «Джоконду» или «Девочку с персиками» нам тут рисовать не потребуется. Хотя, как раз в графических символах, порой приходится обходиться упрощенными изображениями там, где явно не хватает как раз замкнутой залитой цветом криволинейной фигуры.

Итак, что же такое графический символ в **ISIS** и чем он отличается от обычного графического изображения модели компонента. Давайте еще раз вспомним, как мы рисовали компонент. На поле проекта рисовался **2D Graphic** базовый прямоугольник (квадрат, треугольник, круг или что-то еще) – тело компонента, как правило в стиле **Component** (коричневое обрамление и телесная заливка)**.** К нему пристыковывались выводы **Pins** из левой колонки меню **Device Pins Mode**, а также ставился один маркер привязки, называемый **ORIGIN**. Внутри или снаружи тела можно было дорисовать линиями или арками еще «что-нибудь ненужное», например знаки полярности или линии раздела, а

также нанести в стиле 2D надписи (Рис. 1). На этом процесс рисования заканчивался, и мы приступали к операции **Make Device** для создания графического изображения модели компонента.

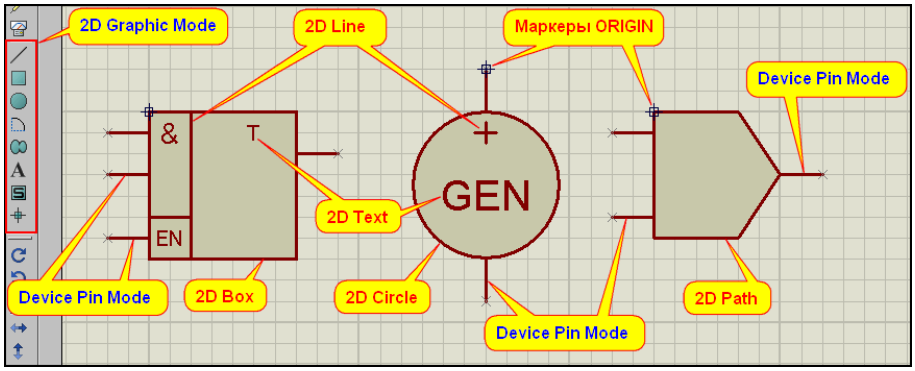


Рис. 1

Изображение для компонента у нас всегда было одно, поскольку оно единожды, раз и навсегда вставлялось в проект и в этом виде присутствовало там постоянно.
С графическими символами немного иначе. Фактически, это часть изображения компонента, которая может изменять свой вид, цвет, положение или числовое значение в процессе выполнения симуляции. Если кто-то увлекался программированием графики, это как-бы спрайт – есть такой термин у программистов графики. В процессе создания активного компонента символы интегрируются в модель (в ту часть, которая располагается в папке **LIBRARY**) и затем, в процессе симуляции воспроизводятся на экране в зависимости от наличия определенных сигналов на выводах или внутреннего состояния модели. Создание символа практически ничем не отличается от создания графического изображения компонента, но все же есть один существенный нюанс, связанный с маркером **ORIGIN**. В компоненте мы его обычно ставим либо в верхнем левом углу тела компонента, либо привязываем к концу одного из выводов (Рис. 1). В одном из начальных разделов я уже упоминал, что он служит для выравнивания компонента по координатной сетке. Но второе его назначение тогда мы не рассматривали, а состоит оно в том, что он служит еще и отправной начальной точкой, относительно которой выравниваются графические символы в активных компонентах. В качестве примера давайте разберем на составные части какой-нибудь семисегментный индикатор, например, модель **7SEG-COM-ANODE**. Весь это проект с комментариями представлен в проекте **7_SEG.DSN** вложения. Помещаем модель в поле проекта и, выделив, выбираем опцию **Decompose** либо через меню правой кнопки мыши, либо через верхнее меню – кнопка с изображением молотка. Теперь в левом вертикальном тулбаре выбираем режим отображения и редактирования символов – латинская **S** в квадратике. После этого в селекторе объектов мы увидим список всех символов, входящих в модель **7SEG-COM-ANODE** (Рис. 2).

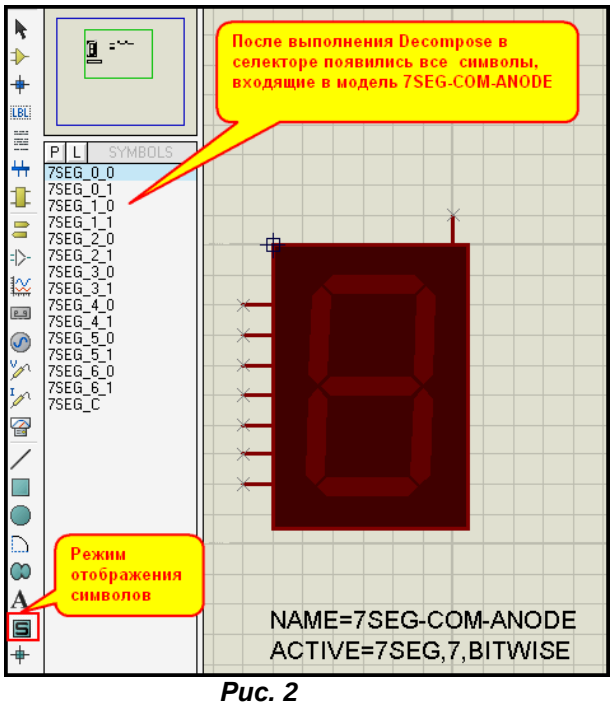


Рис. 2

Само разобранное графическое изображение в поле проекта интересует нас в данный момент только с точки зрения расположения маркера **ORIGIN** – верхний левый угол тела (прямоугольника) модели. Именно он является «базовой точкой» для всех символов, входящих в модель. Если выбрать в селекторе какой либо символ, то в окне предпросмотра вы увидите его вместе с входящим в символ маркером ORIGIN (Рис. 3). В качестве примера на этом рисунке я выделил символ **7SEG_0_1** – светящийся сегмент **«a»** семисегментного индикатора.

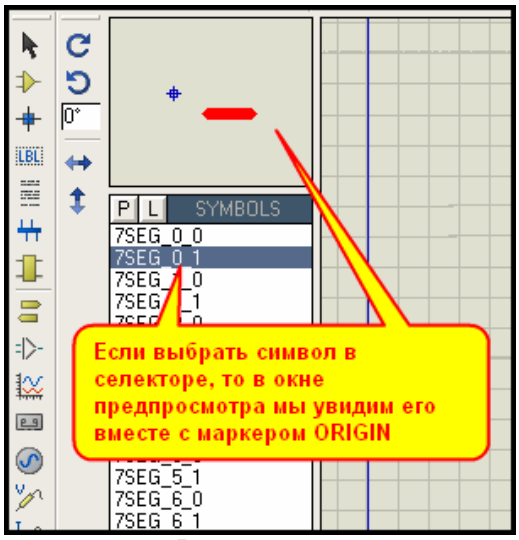


Рис. 3

Однако если вы, выбрав какой либо символ, кликните левой кнопкой мышки в поле проекта и установите его (установится он, как и компонент, вторым кликом по левой кнопке), то маркер в поле проекта будет не виден. Все дело в том, что любой графический символ сам является сложным объектом, состоящим из изображения и маркера **ORIGIN** и так же, как любая модель подлежит предварительной компоновке. Только делается это не через **Make Device**, а через **Make Simbol**. Эта опция отсутствует в верхнем и боковых тулбарах и доступна только через меню правой кнопки мыши (Рис. 4).

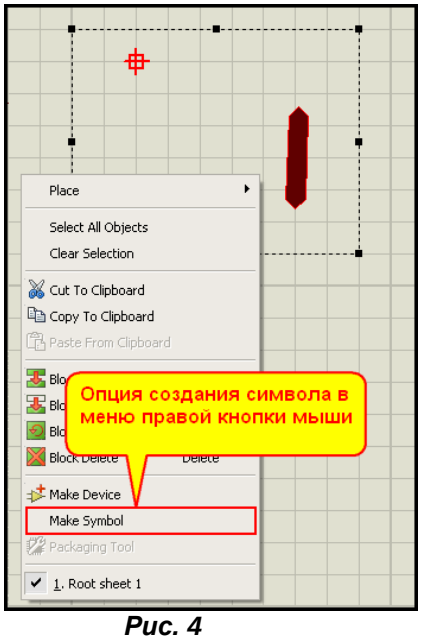


Рис. 4

Соответственно, раз мы создаем символ через **Make Simbol**, то мы можем применить к нему и «разборку на запчасти», т.е. опцию **Decompose**. Вот тогда-то мы и увидим отдельно графику и отдельно маркеры (Рис. 5).

Рис. 5

На что здесь хотелось бы обратить ваше внимание. Если выделить разобранный символ сегмента вместе с маркером и переместить на разобранное графическое изображение компонента так, чтобы маркеры сегмента и всего компонента совпали, то мы увидим, что и сегмент окажется точно на том месте, где он расположен на изображении компонента. Вот для этого и служит маркер **ORIGIN** в символе. Он точно позиционирует графическое изображение символа на изображении всего компонента в момент симуляции. Если не соблюдать этого простого, но очень строгого правила привязки, то при запуске симуляции сегменты вашего активного компонента будут съезжать относительно основного изображения или хаотично прыгать по экрану. Это первый признак того, что вы допустили ошибку при компоновке графических символов.

Теперь разберем состав и нумерацию графических символов на примере все того же семисегментника **7SEG-COM-ANODE**. Если внимательно глянуть в селектор на рисунке 2, то можно заметить некоторую закономерность. Все символы (сегменты) имеют в начале имени аббревиатуру **7SEG**. Совсем не обязательно привязывать данную часть именно к назначению нашего компонента. С тем же успехом, наши символы-сегменты могли бы именоваться и **ABC** и **123A** и вообще так, как подсказывает ваша фантазия. Тут главное, чтобы у всех символов, принадлежащих одному компоненту, эта часть наименования совпадала. Кроме того, настоятельно рекомендую использовать только латинские символы и цифры и не пользоваться спецсимволами и знаками препинания. В частности, знак подчеркивания в наименовании символа служит для разделения частей имени и применение его в других местах приведет к неработоспособности модели. За начальным именем через знак подчеркивания следует номер графического символа. Так как мы разбили семисегментный индикатор, то и таких номеров у нас будет семь, начиная с нулевого и заканчивая шестым. Для каждого символа сегмента в данной модели определено два состояния – погасший и светящийся. Эти состояния указаны через еще один знак подчеркивания. Погашенному состоянию соответствует цифра 0, а светящемуся – 1. Таким образом, полное наименование символа для погашенного сегмента **«a»** будет **7SEG_0_0**, а для засвеченного **7SEG_0_1**, а, например, для сегмента **«d»** (нижний горизонтальный) соответственно **7SEG_3_0** и **7SEG_3_1**. Отдельно стоит остановиться на символе **7SEG_C**. Если выделить его в селекторе, то в окне предпросмотра мы увидим, что он состоит из основного графического изображения – «тела» компонента и маркера **ORIGIN**. Он является базовым, как бы подложкой для наших символов-сегментов во время симуляции и на его фоне мы и будем наблюдать изменение состояния наших сегментов на экране. Для модели **7SEG-COM-ANODE** он представляет из себя базовый прямоугольник с маркером **ORIGIN** (Рис. 6).

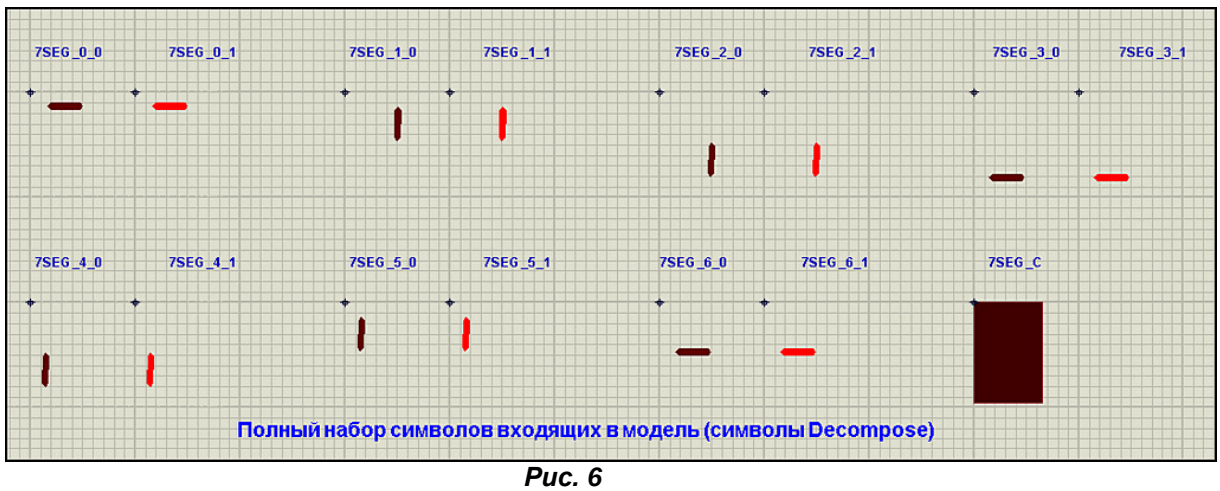


Рис. 6

На рисунке 6 представлены все символы, входящие в модель **7SEG-COM-ANODE** в «разобранном» (**Decompose**) виде. Я специально их «разобрал» в поле проекта, чтобы видно было положение маркера **ORIGIN** и смещение самого изображения сегмента относительно этого маркера. Еще раз подчеркну, что поскольку в данном случае мы имеем дело с так называемым **Bitwise** индикатором, то для каждого сегмента определена группа из двух символов неактивный (в конце символ 0) и активный (в конце символ 1). Термин **Bitwise** дословно означает «зависимый от бита». В данном случае текущее состояние символа **7SEG**_x (где x номер символа) зависит от логического состояния на определенной ножке (**pin**) модели – выводы A, B … G. Для модели с общим анодом это означает, что если там находится логический ноль, то сегмент должен светиться – отображается символ **7SEG_x_1**, если на выводе логическая единица – сегмент не светится – отображается **7SEG_x_0**. Здесь мы имеем дело с цифровым типом индикации (хотя позже я покажу, что на самом деле и не совсем цифровым), но символы совсем не обязательно могут иметь только два вида. Давайте для примера «разберем на запчасти» модель обычного светодиода **LED_GREEN**. Этот пример с комментариями представлен в проекте **LED.DSN** вложения. Попутно хочу дать полезный совет. Проводя различные исследования с активными элементами в ISIS, старайтесь не увлекаться «разборкой» нескольких моделей в одном проекте. Иначе селектор символов станет похож на отхожее место роты солдат в полевом лагере. Дерьма много, а где чье – непонятно. А в ряде случаев может оказаться, что символы одной модели имеют то же имя и заменят символы другой.
На рисунке 7 представлены символы входящие в модель **LED_GREEN**. Как видим, здесь в нумерации используется только одна цифра – номер символа от 0 до 7. Соответственно и символы имеют наименование **LED_GREEN_0** – полностью погашенный светодиод, **LED_GREEN**_1 – слегка подсвеченный и, наконец, **LED_GREEN_7** – полностью светящийся.

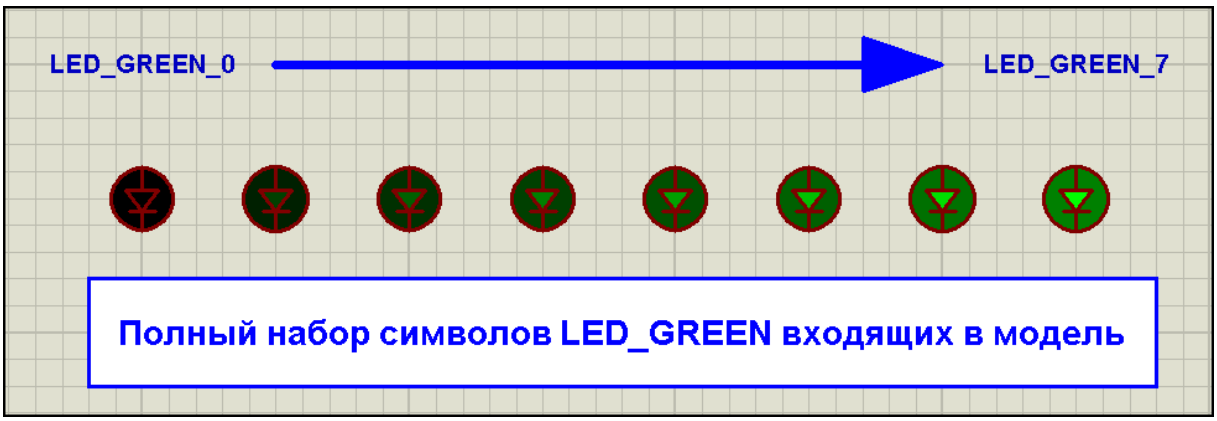


Рис. 7

Всего таких состояний в обычных активных индикаторах может быть 32 (нумерация от 0 до 31). На это тоже прошу обратить особое внимание. Это относится и к **Bitwise** индикаторам, но только в том случае, если мы не используем специализированные DLL-библиотеки для их создания. Там суммарное количество элементов определяется самой применяемой DLL и об этом речь пойдет позже.
В модели **LED_GREEN** сами символы имеют более сложную структуру из нескольких графических элементов и маркера **ORIGIN**. На рисунке 8 представлен «декомпозированный» символ **LED_GREEN_7**. Справа на рисунке все составляющие символа с указанием того в каком режиме рисуется данный элемент.

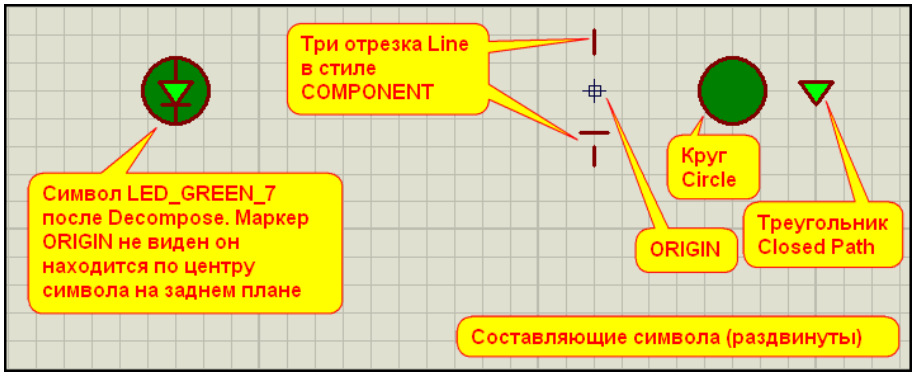


Рис. 8

Как видим, здесь мы имеем дело с более сложным по наличию графических элементов символом. И снова хочу заострить ваше внимание на этой графике, поскольку при создании таких элементов легко можно допустить характерную ошибку. Связана она с расположением графических элементов «в глубину» экрана. ISIS имеет только две опции, расположенные в верхнем меню **Edit**:
Send to back (CTRL+B) – отправить элемент на задний план.
Bring to front (CTRL+F) – выдвинуть элемент вперед.
Этими опциями надо умело пользоваться, иначе вы рискуете получить при создании символа (да и компонента тоже) «пропадающие» с экрана элементы графики. В случае символа светодиода должны быть выдвинуты на передний план отрезки и светящийся треугольник. На самом заднем плане окажется маркер **ORIGIN**, расположенный в данном случае по центру символа.
На этом, пожалуй, можно закончить наше знакомство с особенностями создания графических символов для активных моделей. Настала пора позаботиться о том, где сохранить плоды наших «графических фантазий». И здесь разработчики программы позаботились о нас заранее, но об этом в следующем материале.
Еще одно небольшое замечание по вложению. Подсветка имени выводов моделей при сохранении проекта теряется, поэтому если Вы желаете видеть ее на экране, то включите ее самостоятельно либо в свойствах соответствующих **pin** (выводов), либо через **PAT** (что быстрее). В окне **Sting PAT** набираем **NAME**, ставим переключатели **Action** => **Show** и **Apply To** => **All Objects**, жмем **OK** и получаем нужное. Привыкайте работать «с удобствами».
Внимание. *Начиная с этого раздела в **OnLine** версии имена архивов вложений как и в **OffLine**: номер раздела – символ подчеркивания – номер параграфа. Соответственно здесь: 7_2.rar.*

К содержанию

7.3. «Символические» библиотеки. Использование менеджера библиотек для библиотеки символов.

Мы уже давно привыкли пользоваться библиотеками компонентов для подбора нужных нам в конкретном проекте. Как я уже подчеркивал, для этого всего-навсего необходимо дважды щелкнуть левой кнопкой мышки в свободном поле селектора или нажать в нем кнопочку **P** в левом верхнем углу. При этом многие уже попадали в библиотеку символов по неведению, но закрывали ее «дабы от греха подальше». Теперь мы сознательно заглянем туда. Проще всего попасть в нее, находясь в режиме **2D Graphic Symbols Mode** (нажата кнопочка **S** в левом тулбаре). Процесс аналогичен визиту в библиотеку компонентов – двойной клик левой в селекторе, или одинарный по буковке **P**. Обратите внимание, что при этом справа от кнопок **P** и **L** вверху окна селектора стоит серая подсказка **SYMBOLS**. После этого откроется окно для выбора библиотеки и символов в ней (Рис. 9).

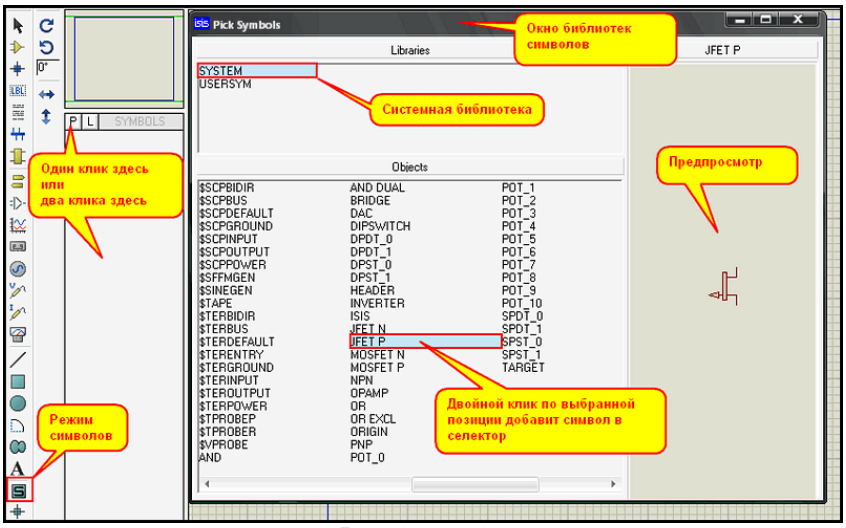


Рис. 9

Картинка немногим отличается от окна библиотеки компонентов, да и операции схожие. В разделе Libraries выбираем нужную библиотеку. Пока их всего две: **SYSTEM** – системная библиотека символов и **USERSYM** – пользовательская, которая, если мы еще не создавали собственных символов, абсолютно пустая. В разделе **Objects** выбирается требуемый символ, который отображается справа в окне предпросмотра. Двойным щелчком мышки мы можем добавить его в селектор символов **ISIS**. Сразу обращаю ваше внимание, что системная библиотека символов не защищена от записи по умолчанию, поэтому если вы сохраняете свой символ, то в первую очередь **ISIS** предложит сохранить в ней. Чтобы не создавать там каши из системных символов и своих собственных будьте внимательнее, или защитите ее от записи в менеджере библиотек. Давайте теперь заглянем в сам менеджер библиотек и попутно создадим собственную библиотеку

для хранения графических символов. Для этого в режиме символов достаточно кликнуть мышкой по кнопочке **L** вверху окна селектора или из этого же режима зайти в **Library Manager** через верхнее меню **Library**. При этом откроется уже знакомое нам окно (Рис. 10). Однако если в нем мы будем выбирать через раскрывающиеся списки **Source** (источник) или **Dest'n** (приемник), то доступны будут только все те же **SYSTEM** и **USERSYM**. Вот здесь и можно запретить запись символов в библиотеку **SYSTEM**. Не забудьте предварительно кликнуть мышкой по той части окна, где она находится. Потом нажимаем кнопку **File Attribute** и на вопрос Протеуса, установить ли режим **Read Only** для выбранной библиотеки нажимаем **OK**. После такой процедуры данная библиотека даже не будет предлагаться для сохранения символов.

Ну а теперь создадим свою библиотеку для символов. В принципе, можно пользоваться и **USERSYM**, но я сторонник крайностей – как поется в популярной ныне на Радио-Шансон песенке: «лучше маленький, но свой». Я уже пояснял по этому поводу при создании компонентов, но здесь вопрос стоит еще острее. Дело в том, что красивые графические символы требуют на свое создание гораздо больших затрат времени, чем сирые и убогие компоненты. И уж куда обиднее будет потерять «нажитое непосильным трудом» при переустановке Протеуса, или копировании чужой **USERSYM** поверх своей. Так что, к делу…

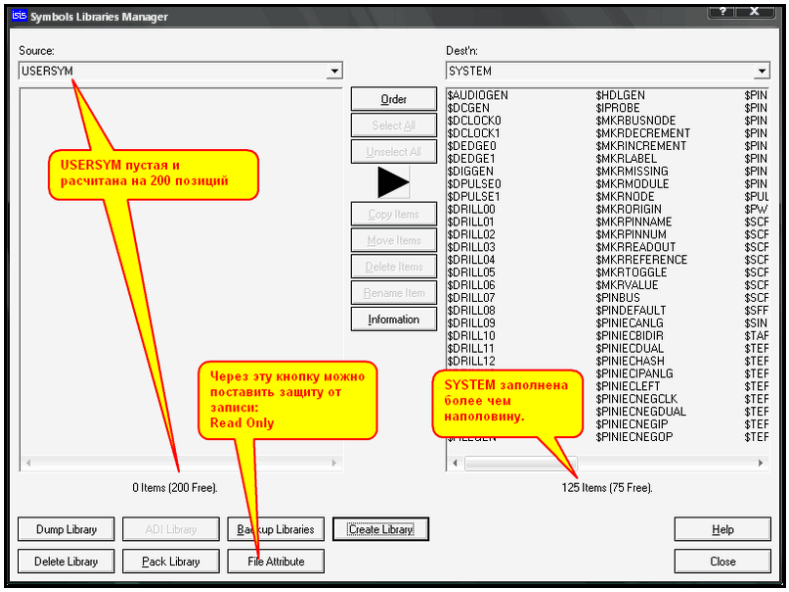


Рис. 10

В окне менеджера библиотек нажимаем кнопку **Create Library** и попадаем в окно выбора папки для сохранения (Рис. 11). Обратите внимание, что по умолчанию нам предлагается сохранить библиотеку в папке **LIBRARY** установленного Протеуса. Папку менять не будем, а вот название зададим свое, например, **MY_SYMBOLS**.

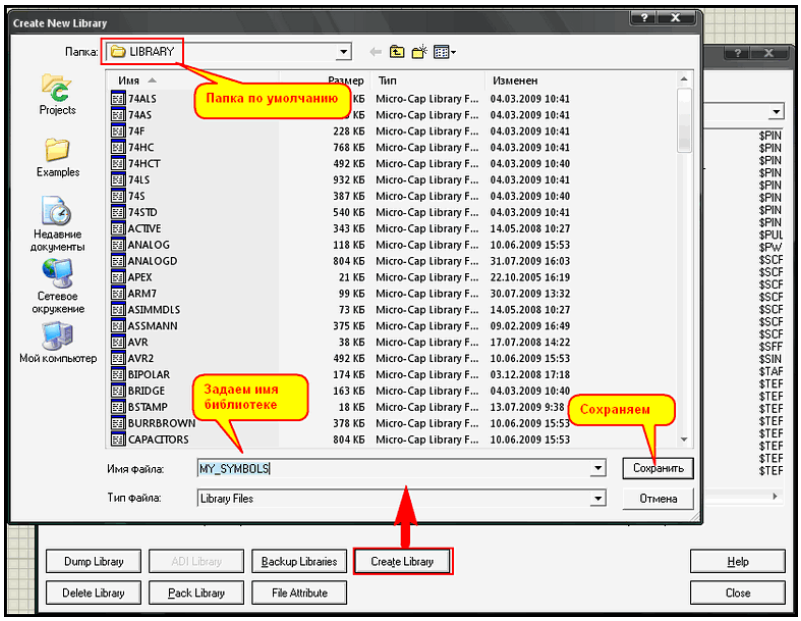


Рис. 11

После нажатия на кнопку: «**Сохранить**» нам будет предложено определить количество хранимых в библиотеке символов – **Maximum Entries**. Здесь ужиматься не стоит, поскольку количество собственной графики будет возрастать семимильными шагами. Поэтому я задал стандартное для библиотек символов количество – 200 и подтвердил создание кнопкой **OK**. После этого менеджер библиотек можно покинуть через кнопку **Close**, так как библиотеку мы уже полностью создали.
Если теперь, находясь в режиме отображения символов вновь дважды кликнуть в селекторе, то в открывшемся окне будут доступны для выбора уже три библиотеки, в том числе и наша «свежеиспеченная» **MY_SYMBOLS** (Рис.12). Конечно, пока она абсолютно пустая, но в скором времени мы заполним ее так, что еще и места будет мало. И начнем мы это делать прямо сейчас. Для начала поучимся создавать простые активные индикаторы на примере все того же семисегментника, только раскрасим его по своему и прилепим туда недостающую десятичную точку.

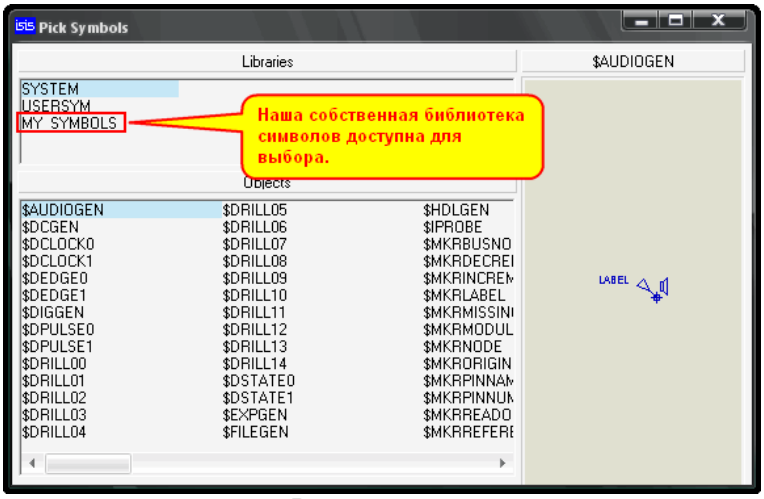


Рис. 12

К содержанию

7.4. Пример создания активного семисегментного индикатора с десятичной точкой. Часть первая – графика.

В качестве первого пробного шара в создании собственных активных компонентов я решил использовать все тот же семисегментный индикатор с общим анодом. На его основе мы поучимся создавать простые **Bitwise** индикаторы. Не будем особо усложнять задачу, а просто сделаем индикатор желтого цвета – таковой отсутствует в стандартной библиотеке и попутно приклеим к нему десятичную точку. Для этого нам потребуется уже разобранный ранее **7SEG-COM-ANODE**. От него мы используем существующую графику, только перекрасим ее. Кроме того, если вы обратили внимание, этот индикатор позиционируется как **Schematic Model**, следовательно, для него имеется файл **MDF**. Называется он **7SEGCOMA** и расположен в библиотеке **DISPLAY.LML**. Как его добыть оттуда, вы уже знаете, повторяться не буду. Я просто приложил уже извлеченный в папке **Prototip** вложения. В довесок там же **7SEGCOMK.MDF** для индикатора с общим катодом и **LED.MDF** для активного светодиода из библиотеки **ACTIVE.LML** для тех, кто хочет поупражняться самостоятельно в создании индикаторов. Итак, приступим.
Начало процесса не вызывает особых затруднений. Мы просто помещаем в новый проект модель **7SEG-COM-ANODE** и применяем к ней **Decompose**. Теперь мы имеем разобранный на запчасти индикатор и кучку символов в селекторе символов (режим **Simbol Mode**). Их мы тоже извлекаем в проект все по порядку и «разбираем на запчасти» через **Decompose**, чтобы графика отдельно - **ORIGIN** отдельно. После этого селектор символов можно очистить, чтобы не путаться со старыми символами. Делается это, как и с компонентами – щелкаем правой кнопкой мышки в селекторе и выбираем опцию **Tidy**. Селектор чист, и можно приступить к созданию собственных символов. На этом первая стадия нашего эксперимента закончена, и я сохраню ее в этом виде в папке **Begin** вложения.
Приступаем ко второй стадии. Для начала перекрасим светящиеся и погашенные сегменты. Дважды щелкаем по первому темнокоричневому сегменту «**a**» и в открывшемся окне Edit path's graphic style выбираем любой из **Colour**, а в открывающемся дополнительном селекторе кнопку **other…**. Для погашенных элементов я выбрал один из темных оттенков желтого цвета и добавил его в набор (Рис. 13).

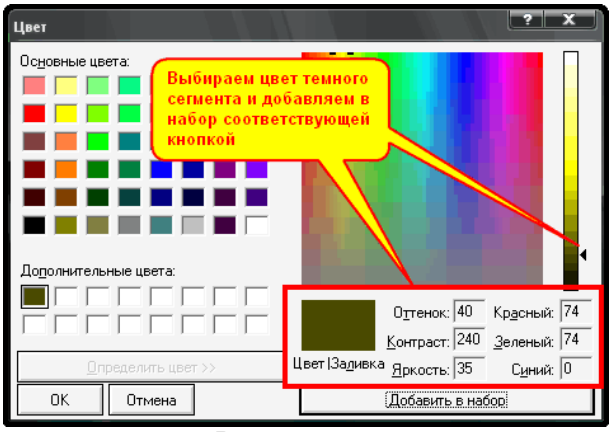


Рис. 13

Для того, чтобы перекрасить весь сегмент нам необходимо выбрать одинаковый цвет как для линии обводки (бордюра), так и для заливки фигуры (Рис. 14). После этого нажимаем кнопку **This Graphic Only**, чтобы сохранить изменения. Проделываем указанную процедуру со всеми погашенными сегментами.

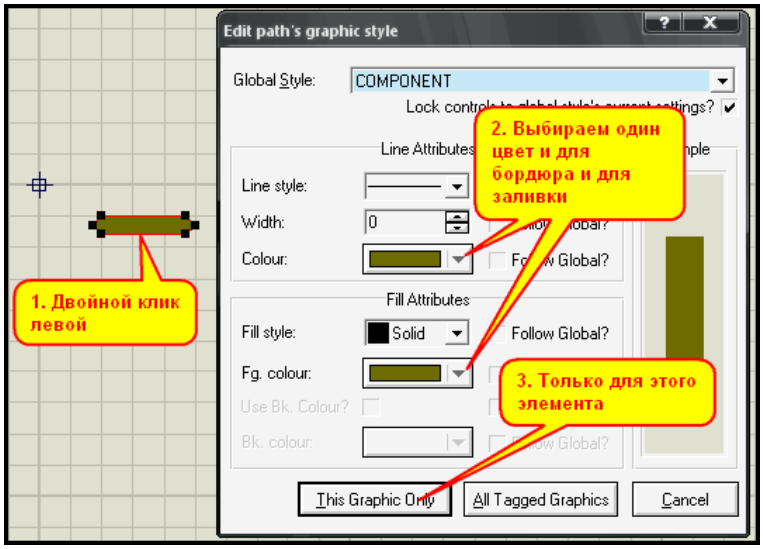


Рис. 14

Для засвеченных сегментов аналогично выбираем стандартный ярко-желтый цвет из палитры основных цветов: второй ряд – второй слева цвет. Кроме того, нам необходимо перекрасить в цвета погашенных сегменты на основном изображении компонента (Рис. 15).

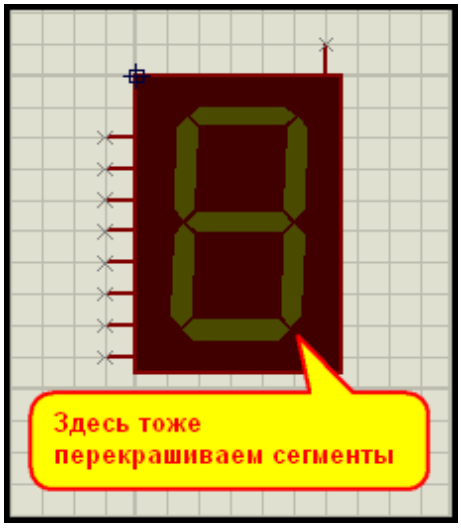


Рис. 15

Тело компонента, а также бывшая и будущая подложка с индексом _C на конце в покраске не нуждаются, хотя наиболее привередливые могут изменить цвет и у них, например, на стандартный для реальных семисегментников серый. Тут главное изменить цвет и у компонента и у подложки на один и тот же. На этом «малярные» работы заканчиваются, и мы приступаем к созданию своих символов.
Создание символов наиболее ответственный момент во всей процедуре, и тут нужно повышенное внимание, чтобы не напороть косяков. Выделяем первый по счету сегмент обязательно с принадлежащим ему маркером **ORIGIN** и через правую кнопку мышки выбираем опцию **Make Symbol** (Рис. 16).

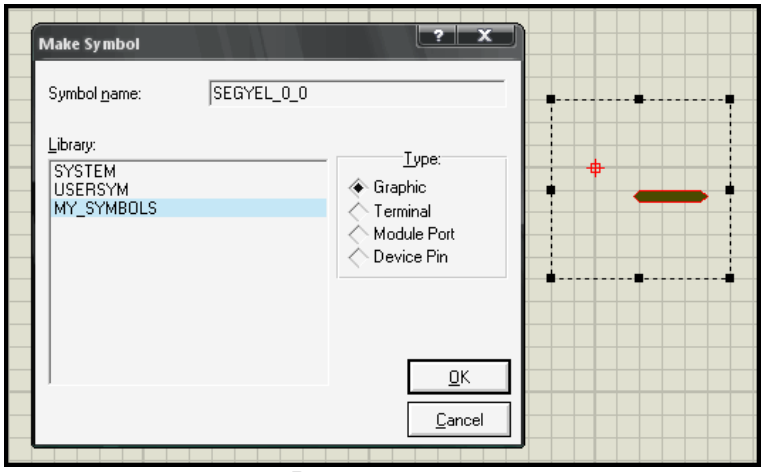


Рис. 16

Обратите внимание, что библиотека **SYSTEM** у меня не защищена и присутствует в окне выбора **Library**. Я достаточно уверен в своих действиях, как шутят автомобилисты – «мастер за рулем», поэтому сразу же выбрал библиотеку **MY_SYMBOLS**. В переключателе **Type** оставляем предлагаемый по умолчанию **Graphic** и задаем имя первого нашего символа. Я назвал его **SEGYEL_0_0** по аналогии с разбиравшимся выше.
Здесь сразу хочу предложить одну «фишку» собственного пошиба. Чтобы каждый раз не набирать имя вручную – не торопитесь давить кнопку **OK**. После набора полного имени выделите его и скопируйте в буфер обмена через CTRL+C или правую кнопку мышки – **«Копировать»**. При создании следующего символа достаточно в этом поле вставить из буфера готовое имя и поправить всего одну или две цифры. Этим вы по крайней мере гарантируете себя от ошибок в наборе имени, ну и чуть-чуть ускорите сам процесс. Есть еще один «неприятный нюанс» Протеуса. После создания каждого символа необходимо в левом тулбаре переключаться из режима символов в режим **Selection Mode** (самая верхняя кнопка с жирной косой стрелкой). Иначе курсор будет находиться в режиме рисования (карандаш) и вы не сможете выделить следующий символ. Следующим по счету будет светящийся желтый символ сегмента «**a**», а имя у него будет отличаться только последней цифрой **SEGYEL_0_1**. Третьим будет темный символ сегмента «**b**» с именем **SEGYEL_1_0** и т.д. до заключительной подложки с именем **SEGYEL_C**. В финале селектор символов у вас будет выглядеть так, как представлено на рисунке 17.

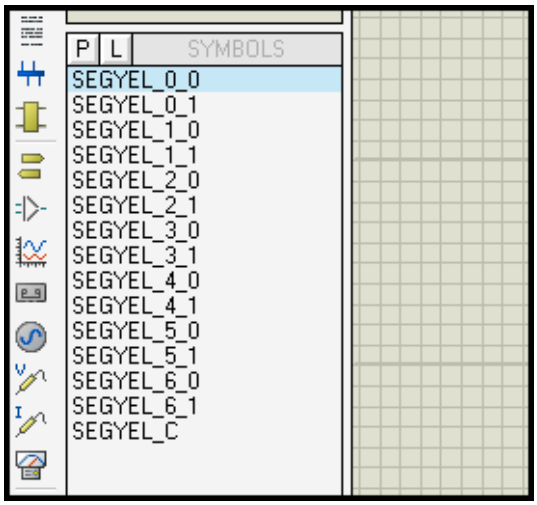


Рис. 17

Ну а если теперь заглянуть в нашу библиотеку символов, то можно убедиться, что все созданные нами символы представлены и там (Рис. 18).

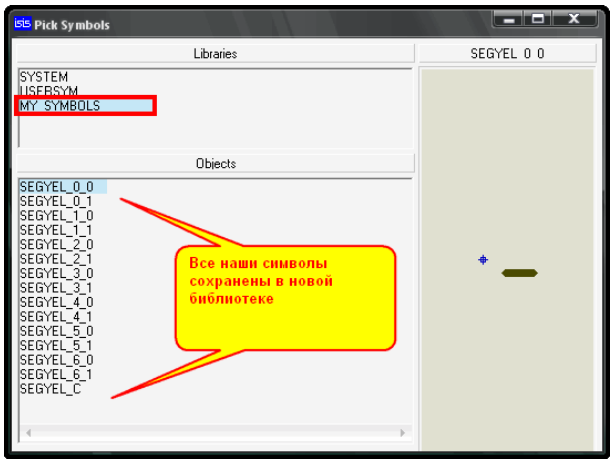


Рис. 18

Но мы собирались добавить к нашей модели еще и десятичную точку. Пора заняться и этим. Проще всего пририсовать для начала точку на полной модели с выводами. При этом у нас будет полная ориентация относительно всех сегментов и края тела модели. Здесь временно потребуется переключиться через меню **View** в режим сетки **Snap 10th**, иначе нам не удастся соблюсти нужные зазоры и размер точки. Кроме того, размер бордюра у нарисованной окружности необходимо установить в 0, как и у сегментов, ну и конечно выбрать для бордюра и заливки цвет погащенного сегмента. Сразу же добавим и необходимый вывод со скрытым именем **H** (Рис. 19). Теперь копируем полностью графическую модель на свободное место в проекте через **Block Copy** в двух экземплярах и удаляем на ней все лишнее, кроме самой точки и маркера **ORIGIN**. Таким «хитрым способом» мы точно обеспечим соблюдение смещения точки относительно маркера **ORIGIN**.

Рис. 19

Нам осталось изменить цвет для светящейся точки и создать два дополнительных символа с именами **SEGYEL_7_0** и **SEGYEL_7_1** для погашенной и светящейся точки, как мы делали ранее.
На этом процедура создания графики для нашего семисегментного индикатора завершена. В следующем параграфе займемся созданием полной модели с пробной симуляцией.

К содержанию

7.5. Пример создания активного семисегментного индикатора с десятичной точкой. Примитивы для создания индикаторов. Часть вторая – модель.

Для начала предстоит создать графическую модель компонента, которая и будет представлять его в библиотеке, но попутно мы включим туда и свойства активного компонента. Итак, графическую (не симулируемую) модель создаем из того набора, что представлен на предыдущем рисунке вместе с добавленным выводом и десятичной точкой. Как и обычно, выделяем все это рамочкой и нажимаем нашу любимую кнопку **Make Device**. Именно девайс, символов мы уже насоздавались. И вот тут с

самой первой вкладки начинаются новшества. Я уже упоминал, что мы создаем бит-зависимый активный компонент. Зададим ему имя, пусть он **7SEG-COM-ANODE-YEL** (больше просто не упишется), префикс можно и не задавать, тут это не критично и делается как всегда.

Приключения начинаются в нижней части окна – **Active Component Properties** – ведь мы создаем активный компонент. В графе **Symbol Name Stem** вводим имя наших созданных сегментов. Вводится только общая часть имени, которая слева до первого символа подчеркивания, в нашем случае это **SEGYEL**. После этого становится доступным для ввода количество состояний – **No. of States**. Оно численно равно количеству различных видов символов – цифра после первого подчеркивания с учетом нулевого. Это означает, что если у нас номер последнего символа (в нашем случае добавленной десятичной точки) равен семи и нулевой символ – сегмент «**a**», то в этой графе мы указываем число восемь. Не ошибайтесь с подсчетом в этом месте, иначе один из символов работать не будет. Ну и, наконец, поскольку мы создаем многоэлементный индикатор, в котором каждый символ связан с состоянием определенного вывода компонента, включаем флажок **Bitwise** – «бит-зависимый» (Рис. 20).

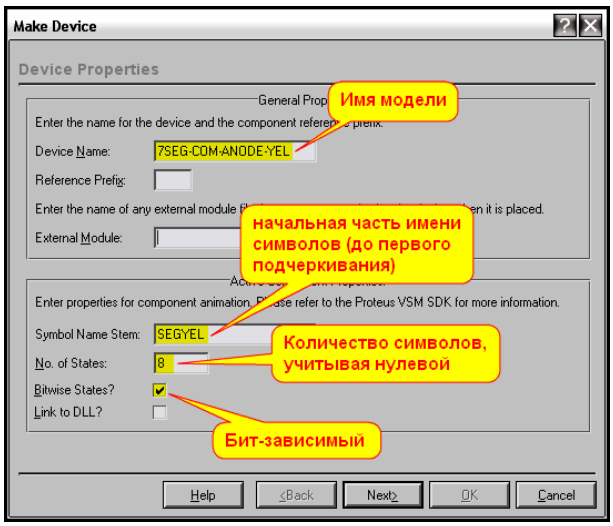


Рис. 20

В остальных вкладках функции **Make Device** можно пока ничего не заполнять и лихо проскочить их до последней, нажимая кнопку **Next**. На последней вкладке нам необходимо причислить нашу модель к какой-либо группе компонентов, либо создать новую. Я не стал мудрить в данном случае – выбрал из существующих, изменив только описание компонента – графа **Device Description**. Ну и сохранил это все пока в библиотеке **USRDVC** (Рис. 21).

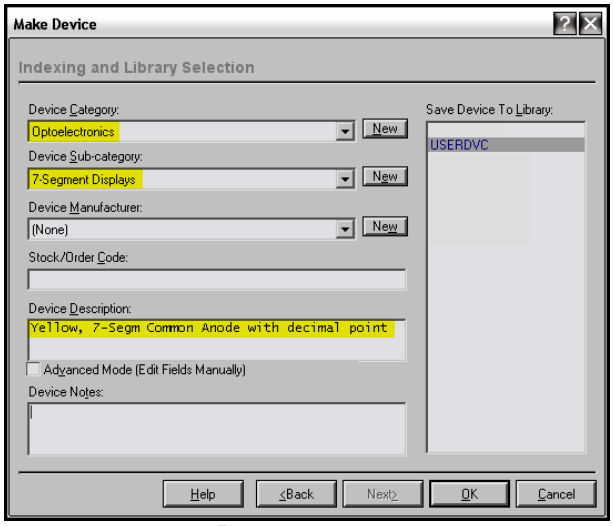


Рис. 21

Теперь наша модель присутствует в селекторе компонентов, и мы можем поместить ее в проект. Пора приделать к ней дочерний лист, чтобы на нем смоделировать внутреннюю структуру. Тут все, как и прежде – заходим в свойства и устанавливаем галочку **Attach hierarchy module**. Теперь мы можем спокойно проследовать на дочерний лист и заняться восстановлением структуры модели по

файлу **7SEGCOMK.MDF**. Если бы мы не добавили еще один элемент – десятичную точку, то можно было бы и вообще использовать этот файл, но теперь нам придется добавить недостающую часть внутренней структуры и для нее. Вообще, для активных **Bitwise** компонентов рисовать структуры – одно удовольствие. Достаточно воспроизвести структуру для одного символа и затем размножить ее через кнопку **Block Copy**. На рисунке 22 приведена часть восстановленной по **7SEGCOMK.MDF** структуры с дочернего листа нашей модели. Здесь четко просматриваются два одинаковых «канала» для сегмента «**A**» и для сегмента «**B**». Каждый канал состоит из примитивов токового пробника **RTIPROBE** и аналогового выключателя **VSWITCH**. Всего таких каналов восемь, по числу используемых символов. Все отличие состоит в номере свойства **Target Element** у токовых пробников **IP1…IP8** (на рисунке это свойство ELEMENT специально сделано видимым).

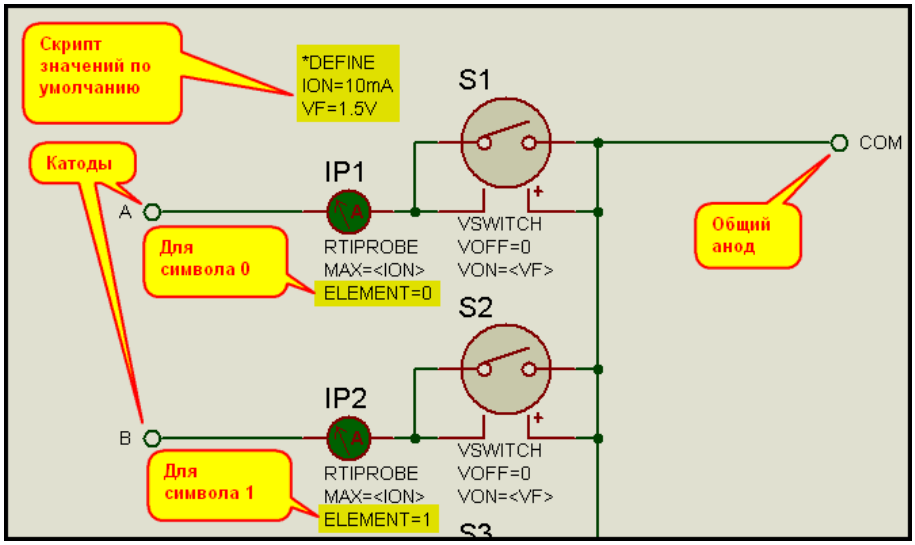


Рис. 22

Рассмотрим предназначение элементов одного канала. Аналоговый выключатель в данном включении служит пороговым элементом по напряжению с напряжением включения **VON=<VF>**. Напряжение включения **VF** задано в скрипте ***DEFINE** и равно 1.5V для всех каналов. Позже, мы пропишем его в свойствах модели с возможностью изменить из основного листа. Кроме того, в свойствах каждого **VSWITCH** в соответствии с **MDF** прототипа задано сопротивление в выключенном состоянии **Off Resistance** ROFF=100k и во включенном **On Resistance** RON=10. Таким образом, если между левыми и правыми группами выводов **VSWITCH** приложить напряжение менее полутора вольт, то он не включится и его сопротивление будет 100кОм, а если свыше 1,5В, то он перейдет во включенное состояние с сопротивлением 10 Ом между верхними по схеме выводами. Обратный переход произойдет при напряжении VOFF в нашем случае при 0В. Мы уже рассматривали управляемый ключ **VSWITCH** в **п.4.13** и больше я на нем останавливаться не буду. А вот на свойствах примитива **RTIPROBE** и его собратьев по назначению остановимся подробнее, так как они являются основой для создания активных индикаторов. Все они расположены в библиотеке **Modelling Primitives** в подпапке **Realtime (Indicators)** и назначение их служить пробниками для определения состояний точек (цепей) схемы с выводом результата в виде меняющихся графических символов. Аналоговыми индикаторами являются два из них токовый **RTIPROBE** и напряжения **RTVPROBE** (Рис. 23). Соответственно токовый включается в контролируемую цепь последовательно и индицирует ток в ней, а пробник напряжения подключается к двум контролируемым точкам и индицирует напряжение между ними. Текущее состояние (индицируемый символ) определяется по схожим в написании формулам – я привел их на рисунке над соответствующими моделями. В них:

STATE – текущее состояние (отображаемый в данный момент символ);

NUMSTATES – возможное количество состояний (символов индикации) для аналогового индикатора (вспомните восемь разных по свечению состояний светодиода на рисунке 7 из п.7.2 выше) или количество бит-зависимых символов с двумя состояниями для **Bitwise** индикатора (это то, что мы прописывали на первой вкладке **Make Device** в графе **No. of States** на рисунке 20);

CURRENT (или **VOLTAGE**) – текущее значение тока (напряжения);

MIN и **MAX** – заданные в свойствах соответственно: минимальное и максимальное значение тока (напряжения).

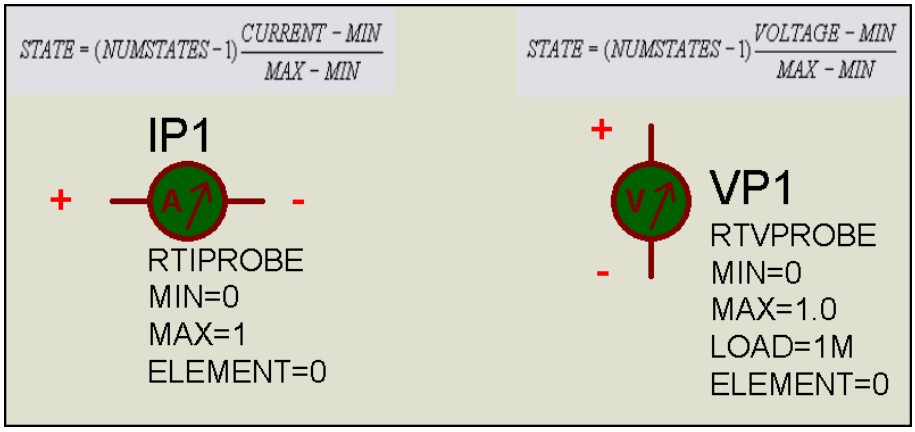


Рис. 23

Ну и сразу же рассмотрим оставшиеся свойства этих примитивов. На рисунке 24 приведено приведено окно свойств пробника напряжения – у него на одно свойство больше, чем у токового.
Load Resistance – нагрузочное сопротивление самого пробника (только для пробника напряжения).
Target Element – целевой элемент (только для **Bitwise** индикаторов). Это как раз номер того символа (сегмента), который контролируется данным пробником в нашем случае.

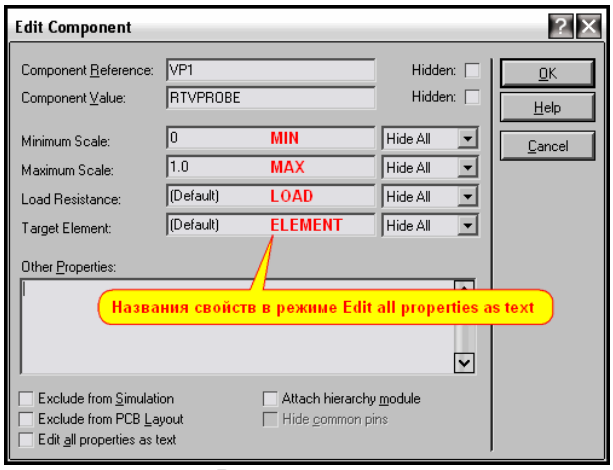


Рис. 24

Итак, в нашем случае использован токовый пробник **RTIPROBE** с установленным **MAX=<ION>**, которое в свою очередь в скрипте задано 10mA. Так как мы используем **Bitwise** режим, то соответствующий символ (сегмент) станет активным (светящимся) в том случае, если ток через пробник стане 10мА или выше, а это произойдет при срабатывании ключа **VSWITCH**. Вот собственно и весь принцип работы одного канала индикатора.
Ну и несколько слов о цифровых примитивах **RTDPROBE**. Однобитные примитивы индикаторов ведут себя предсказуемо: если на входе лог. 0, то соответствующий **Target Element** (символ) не активизирован, если на входе лог. 1, то он активизирован. В той же папке имеются многовходовые **RTDPROBE**. Детально я их не исследовал, но при беглой логика входов абсолютно непонятна – что-то похожее на исключающее ИЛИ. К сожалению и в **HELP** на них описание весьма скудное. Соответственно свойства всех рассмотренных примитивов описаны в хелпе **ProSPICE Primitives** в разделе **Real Time Modelling Primitives**. Владеющие английским языком могут прочесть в оригинале. Ну а мы вернемся к «нашим баранам», т.е. индикаторам.
Есть еще одна немаловажная «фишка», об которую я в начале освоения Протеуса набил немало шишек. При создании модели индикатора на первой вкладке **Make Device** мы не указали префикс модели – **Reference Prefix**. Я пропустил его умышленно, поскольку именно так сделано и у существующих моделей семисегментных индикаторов. Но, со всей ответственностью могу утверждать, что наша активная модель с дочерним листом без него «светиться» не будет. Будет потреблять ток, никаких ошибок и предупреждений не будет, но нормальной работы мы не добъемся, пока не применим «превентивных мер». А сделать над всего-навсего следующее. Либо задать префикс уже в свойствах готовой модели в строке **Component Reference**, либо, находясь на дочернем листе зайти в верхнем меню **Design => Edit Seet Properties** и там задать имя дочернему листу (Рис. 25). Причем имя должно быть обязательно задано только заглавными буквами латиницы или цифрами. Я и тут не стал философствовать, а просто набрал символику индикаторов **HG**.

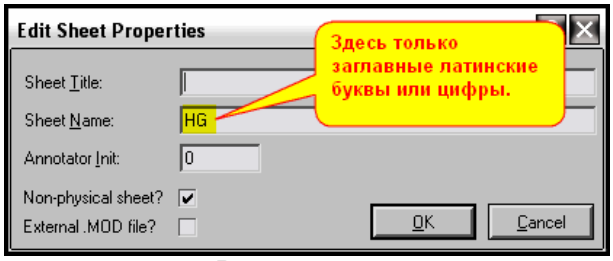


Рис. 25

Вот теперь можно вернуться на основной лист, подключить соответствующее питание и запустить симуляцию, чтобы убедиться, что наш индикатор работает (Рис.26).

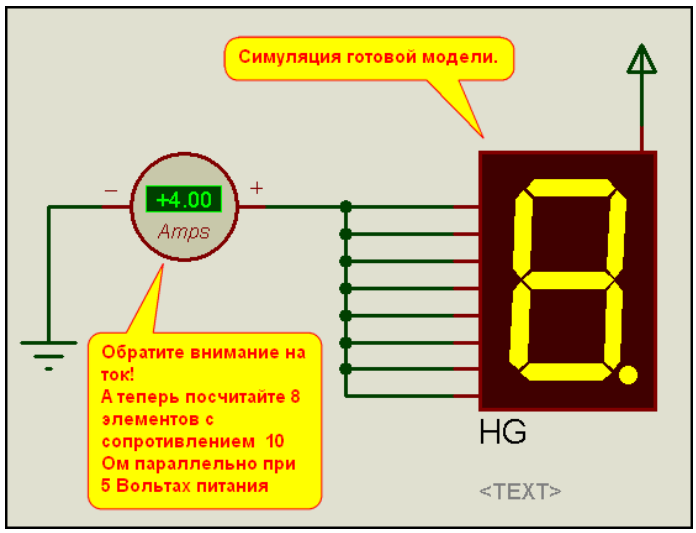


Рис. 26

Я умышленно врезал в цепь амперметр, чтобы показать, что наша модель чисто аналоговая. При питании от терминала 5V (по умолчанию) она потребляет ток 4А (!!!). Определяется он в данном случае параллельным включением всех 8 **VSWITCH** для которых задано сопротивление RON=10 Ом. Учитывайте это в своих разработках, используя подобные модели. Это не относится к индикаторам на основе DLL, о которых речь пойдет позже.
Ну и в заключение нам осталось сформировать файл MDF с дочернего листа нашего проекта, который представлен во вложении в папке **Model_with_Child**. Как это делается вы уже должны знать наизусть.
Окончательный вариант с отключенным дочерним листом представлен в папке вложения **Final_model_with_MDF**. Там же лежит и скомпилированный файл **7SEG_POINT_COM_AN.MDF**. Теперь нам осталось еще раз пройти процедуру Make Device для модели и на третьей вкладке добавить сам файл **MDF** (Рис. 27), а также через **Blank Item** по аналогии с прототипом два свойства **ION** и **VF**, которые были у нас в скрипте ***DEFINE** дочернего листа (Рис 28 и 29 соответственно). Не забудьте открепить сам дочерний лист – снять галочку.

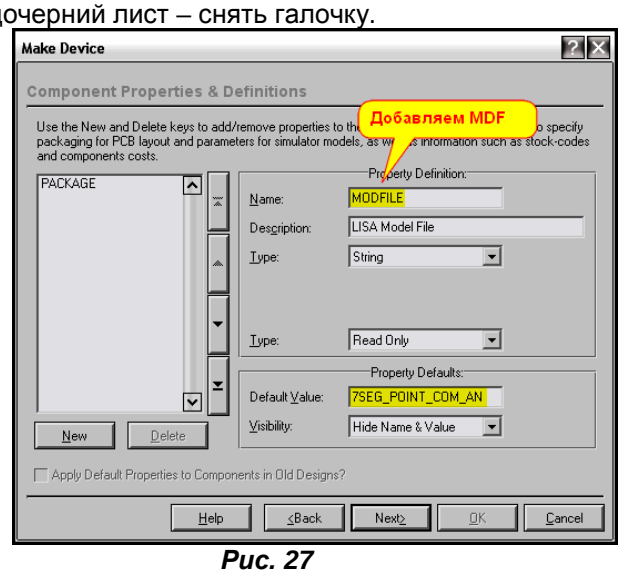


Рис. 27

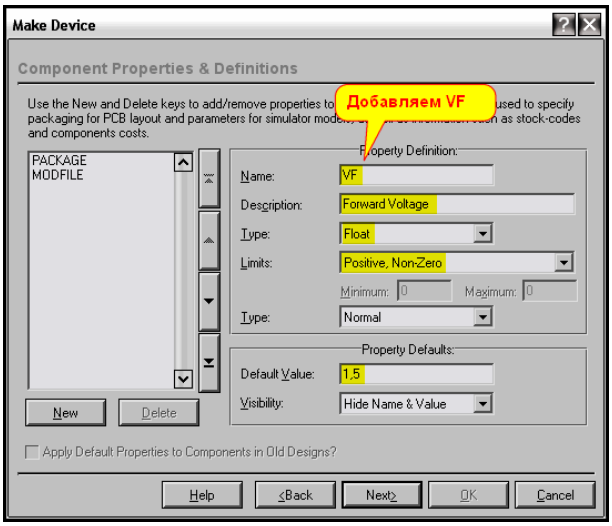


Рис. 28

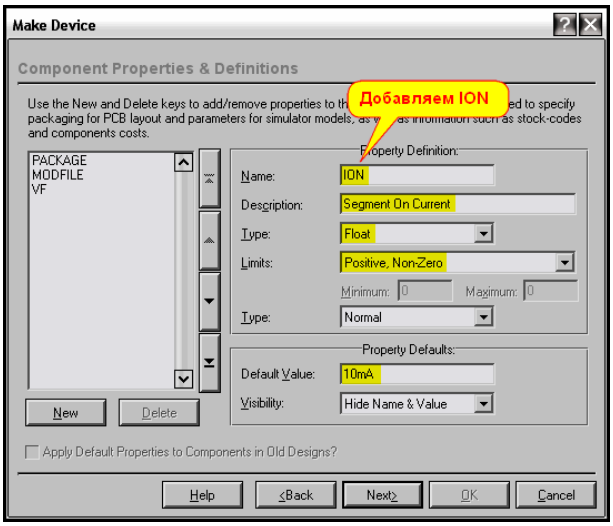


Рис. 29

На этом модель полностью готова. Желающие могут перенести ее из **USRDVC** в какую-либо другую библиотеку, а файл MDF поместить в папку **MODELS** Протеуса для дальнейшего использования. Далее можете самостоятельно попрактиковаться в «раскрашивании» семисегментных индикаторов. Кстати, если использовать символы, которые мы создали для желтого и только менять цвета, то файл **MDF** для всех будет один и тот же, т.е. процедуру «приклеивания» дочернего листа повторять уже не придется. Мы просто перекрашиваем сегменты, создаем новый набор символов нужного цвета, затем создаем модель с оригинальным именем и присоединенным набором сегментов нужного цвета, а далее на третьей вкладке добавляем наш MDF и дополнительные два свойства.

Можно также самостоятельно пораскрашивать светодиоды – там и MDF менять не надо, оставляете родной. Я же на этом временно закончу рассмотрение индикаторов и перехожу к теме создания моделей на основе существуюших **DLL**, т.е. приступаем к созданию ранее представленного и обещанного регулируемого источника питания.

К содержанию

8. Активные модели на основе существующих DLL.

8.1. Немного «тумана» о DLL и о том, что будет, а чего не будет в этом разделе.

Вот мы и подошли к тому моменту, когда количество изложенного материала даст нам и качественный скачок. Но сначала о том, что же такое **DLL**. **Dynamic-link library** – библиотека динамической компоновки – так гласит Википедия. Концепция применения **DLL** изначально заключалась в том, чтобы создать исполняемые модули, которые могли бы независимо использоваться различными приложениями. Да и структура динамических библиотек во многом схожа со структурой исполняемых файлов, они содержат исполняемый код, данные и ресурсы, которые могут полностью или частично использоваться разными приложениями Windows. Характерным примером DLL являются драйвера устройств. Компоновка программного кода в динамические библиотеки возможна в большинстве программных сред, работающих с языками высокого уровня. Возможность использования единожды загруженной в память DLL различными процессами в динамическом режиме позволяет добиться гибкости и высоких скоростей обработки информации.

Протеус, как программный продукт, работающий под управлением MS Windows, не является исключением. Именно использование DLL позволяет проводить имитацию поведения микроконтроллеров и других сложных электронных компонентов в режиме реального времени. Подавляющее количество библиотек для программных моделей в **ISIS** написано самим разработчиком – фирмой **Labcenter Electronics** в среде **MS Visual C++** и скомпоновано в динамические библиотеки с именем имитируемой модели или группы и расширением DLL. В ранних версиях Протеуса, до версии 6.2 включительно в состав установочного пакета входили и средства для разработки программных моделей - **Proteus VSM SDK**. Для «неанглоязычной» и абсолютно незнакомой с программированием публики привожу расшифровку аббревиатуры и перевод. **VSM** – Virtual System Modeling – виртуальное системное моделирование, **SDK** – Software Development Kit – комплект средств разработки. Теперь, надеюсь, понятно, о чем идет речь. Это заголовочные файлы для среды C++ с расширением **CPP**, располагавшиеся в папке **INCLUDE**, а также файл помощи по их использованию. Кстати, ссылка на этот файл до сих пор фигурирует в меню Help ISIS, только вместо реального файла помощи там торчит «заглушка». Как это не печально, но, начиная с версии 6.3, доступ к средствам разработки был закрыт, поскольку фирма опасается плагиата со

стороны возможных конкурентов. Если кто-то чувствует себя вполне уверенно в программировании на **C++**, причем не элементарном – две три функции, а с использованием классов, то может поискать старые версии Протеуса, к которым прикладывались средства для разработки. При тщательном поиске в сети можно обнаружить и PDF-вариант HELP, называется он **Proteus VSM DSK** (именно DSK а не SDK – не знаю кто, но допустил «очепятку» и так и не поправил ее).
Я же в последующем материале не ставлю перед собой цель обучить вас программированию на **C++**, и переписыванием вышеупомянутого VSM DSK на русском языке тоже увлекаться не собираюсь. Моя цель гораздо проще. Я хочу показать в этом разделе, что умело используя уже существующие **DLL**, можно создать достаточно «продвинутые» модели для применения в своих проектах. И для этого совсем не надо быть корифеем в **C++**, можно даже совсем не знать этого языка. Достаточно проявить нашу русскую смекалку и сообразительность, ну и немного «упертости» в достижении результата. Приступим…

К содержанию

8.2. Библиотека SETPOINT.DLL и задаваемые для нее параметры на примере температурного датчика LM20.

Само название библиотеки, если разложить его на составляющие и перевести, говорит о ее назначении, **Set point** в переводе с английского – «установить точку». Именно для этих целей и написана **SETPOINT.DLL**. С параметрами, которые можно использовать при использовании данной библиотеки, проще всего познакомиться на конкретном примере. **SETPOINT.DLL** достаточно широко применяется в моделях Протеуса и мы встретим ее почти везде, где встречаются маркеры **INCREMENT** и **DECREMENT** и «миниатюрный дисплей» для вывода числового значения изменяемого параметра (Рис. 30). Для изучения свойств я выбрал модель температурного датчика LM20, хотя можно было воспользоваться и любой другой моделью с присутствующими в ее составе характерными элементами. Вы уже наверняка применяли в своих проектах подобные модели и знаете, что клик мышкой по маркеру INCREMENT (стрелка вверх) дает приращение параметра на один шаг (в этой модели **Temperature Step** в свойствах), а клик по маркеру **DECREMENT** (стрелка вниз) уменьшает значение на один шаг.

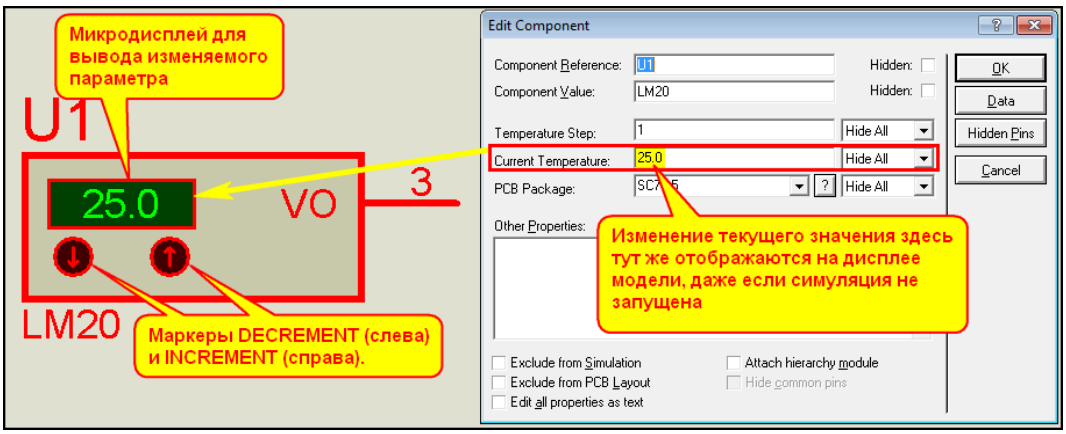


Рис. 30

Чтобы познакомиться с тем, как подключена библиотека **SETPOINT.DLL** к модели LM20, а заодно и узнать об остальных параметрах воспользуемся все той же, любимой мною, функцией **Make Device**. Выделяем LM20 в поле проекта, задаем **Make Device** и смотрим что же интересного на первой вкладке (Рис.31).

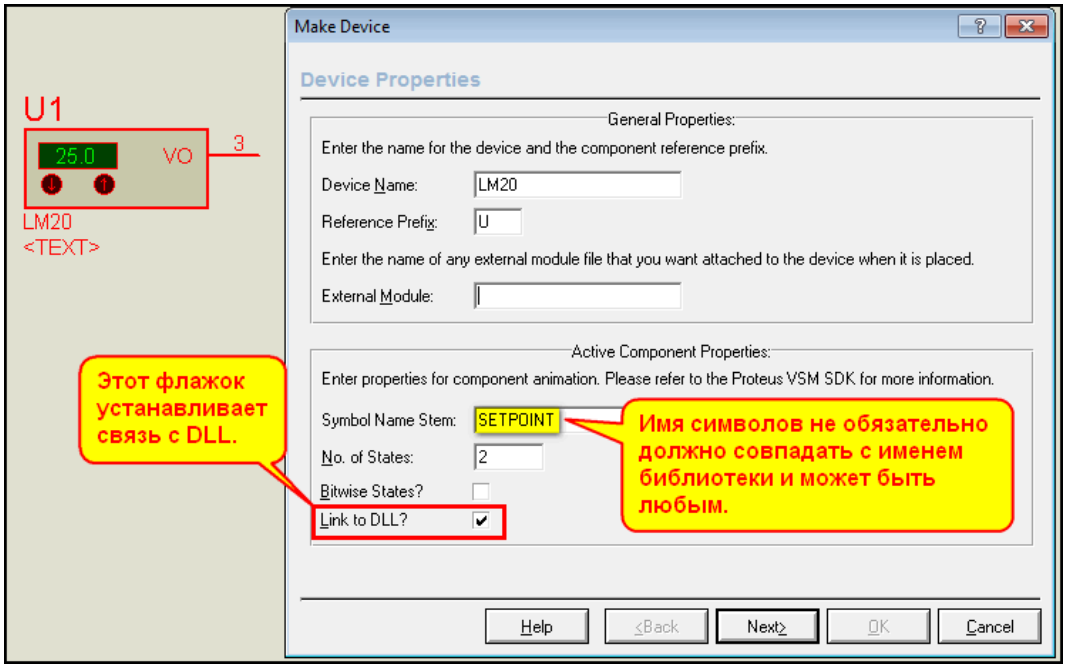


Рис. 31

Здесь мы можем убедиться, что наша модель является активной, поскольку заполнена нижняя часть вкладки – **Active Component Properties**. В составе модели имеются два символа с именем **SETPOINT**. В данном случае имя символов совпадает с именем библиотеки, но это совсем не обязательное условие работоспособности модели. Просто здесь имеет место случайное совпадение. А вот самым нижним флажком мы еще не пользовались. Именно он и привязывает активные свойства модели к программной библиотеке **DLL**. **Link to DLL** так и переводится на русский – «связь с DLL». Ну, к символам активной модели мы вернемся позже, а сейчас, чтобы лишний раз не выпадать из **Make Device**, проследуем на третью вкладку и посмотрим что еще интересного в свойствах нашей модели. И в первом же свойстве **MODDLL** мы встречаем привязку уже конкретно к библиотеке **SETPOINT.DLL** (Рис.32). Более, кроме того, что этому свойству присвоен тип **Hidden** (скрытое) нас в этом свойстве ничего не заинтересует.

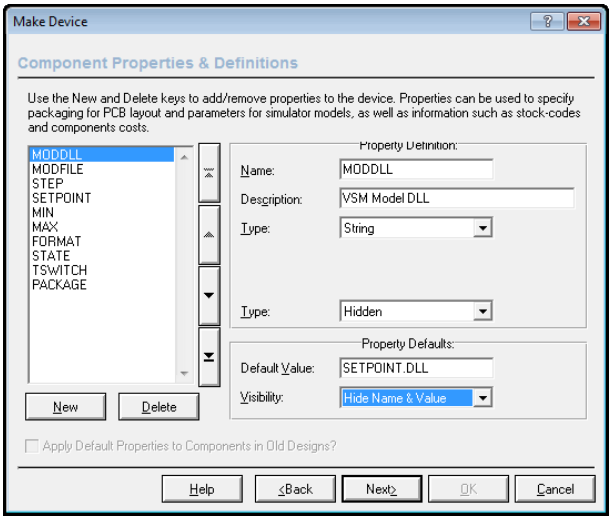


Рис. 32

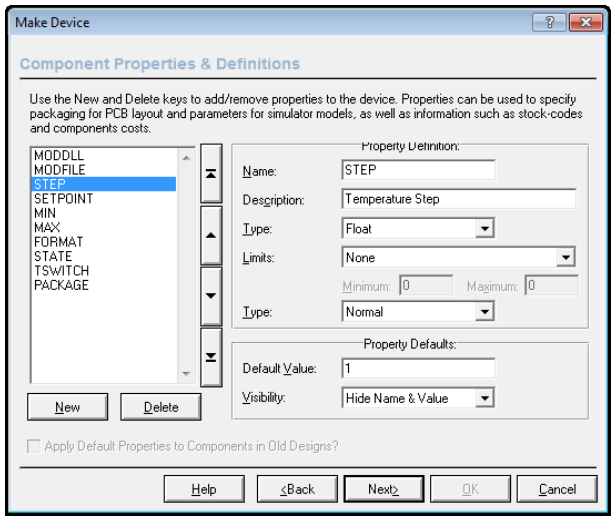


Рис. 33

Свойство **MODFILE** относится к конкретному **MDF** модели, и мы на нем пока останавливаться не будем, а рассмотрим только те, что связаны с **SETPOINT.DLL**.

STEP – шаг (Рис.33). Для этого свойства и последующих, относящихся к **SETPOINT**, при добавлении их к создаваемой модели следует выбирать опцию **New=> Blank Item** и вводить имя вручную, поскольку в стандартных они не прописаны. Написания имен должны быть именно такими, без всяких там «загогулин» и грамматических ошибок, т.к. именно так они фигурируют в **SETPOINT.DLL**. Итак, шаг определяет единичное изменение регулируемого параметра при нажатии на маркер **INCREMENT** или **DECREMENT**. В графе **Description** для этого свойства вы вольны писать все, что заблагорассудится, в том числе и на кириллице. Поскольку в данный момент мы рассматриваем датчик температуры там и написан **Temperature Step**. **Type** определен как **Float**, т.е. можно использовать дробные значения. Ограничения – **Limits** не установлены. В следующей

графе **Type** определено будет ли показан данный параметр при открытии окна свойств модели и доступен ли он для редактирования. Значение **Normal** в данной графе предусматривает и то и другое. Ну и, наконец, в **Property Defaults** назначено значение по умолчанию **1** и для **Visibility** определено **Hide Name & Value**, т.е. по умолчанию рядом с моделью на месте серого **<TEXT>** это свойство демонстрироваться не будет.

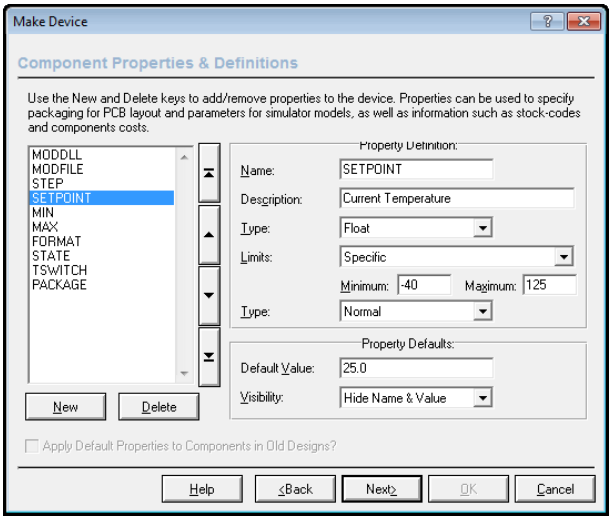


Рис. 34

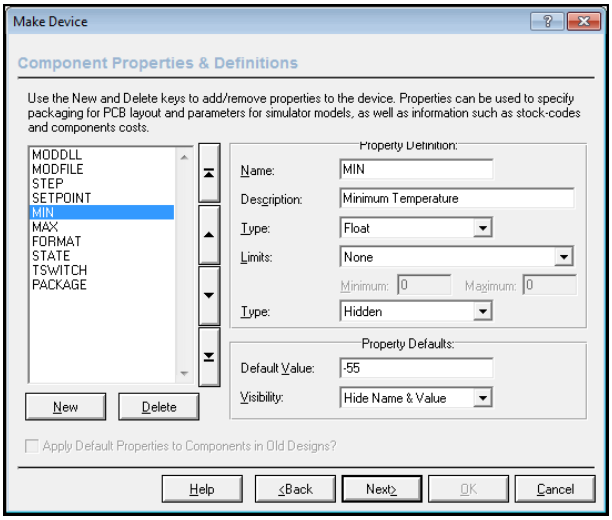


Рис. 35

SETPOINT – установленная точка (Рис.34). Это свойство задает модели начальное (стартовое) значение. Опять-таки, поскольку у нас датчик температуры, то ему и задано **Description** как **Current Temperature**. Тип задан тоже **Float**, а вот на графу **Limits** я прошу обратить особое внимание. В ней из раскрывающегося списка выбран тип Specific (надеюсь, хоть это слово переводить не надо) и ниже заданы особые ограничения минимальное **-40** и максимальное **125**. Т.е. это предельные значения для задания температуры. Я не знаю, какие цели преследовал автор модели, задавая именно эти значения здесь, так как для большинства модификаций датчиков **LM20** эти значения по даташиту равны соответственно **-55** и **130**. Но в одном я ему благодарен, поскольку у меня есть возможность показать, как это ограничение работает в **ISIS**. Если вы попробуете открыть окно свойств датчика (Рис. 30) и ввести там для **Current Temperature**, допустим значение **128** и попытаетесь потом закрыть окно свойств, то получите предупреждение о недопустимом значении температуры. Аналогичное сообщение выскочит и при задании температуры ниже **-40**. Ну, больше для этого свойства ничего интересного нет, следующий **Type** задан **Normal**, поэтому в окне свойств модели (Рис. 30) мы видим **Current Temperature** отдельной строкой с доступным для изменения окошком значения, которое по умолчанию для «свежевытащенной» из библиотеки модели равно **25** градусам (окошко **Default Value** на рисунке 34).

Ну а теперь небольшой фокус. Для отдельно стоящих свойств **MIN** (Рис. 35) и **MAX** (Рис. 36) заданы соответственно именно те значения из даташита **-55** и **130**. Эти свойства передают программной модели **SETPOINT** пределы изменения нашего параметра с помощью кнопок-маркеров **INCREMENT** и **DECREMENT**. Так вот, автор здесь задал значения шире, чем те пределы в свойстве **SETPOINT**. И с помощью этих кнопок мы можем изменять температуру именно в пределах **-55…130**, и Протеус при этом ругаться не будет. Вот такой парадокс. Ну, раз уж мы плавно перескочили к рассмотрению свойств **MIN** и **MAX**, то хочу обратить ваше внимание на то, что для них нижний **Type** задан **Hidden** (скрытый). Поэтому в окне свойств датчика **LM20** (Рис. 30) мы их не видим, но можем увидеть и даже менять заданные значения, если поставим галочку **Edit all properties as text** в левом нижнем углу этого окна. Более о них сказать нечего, **Minimum Temperature** и **Maximum Temperature** говорят сами за себя.

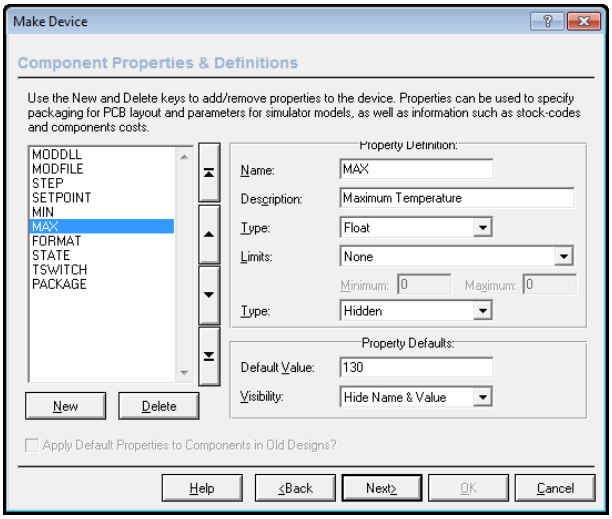


Рис. 36

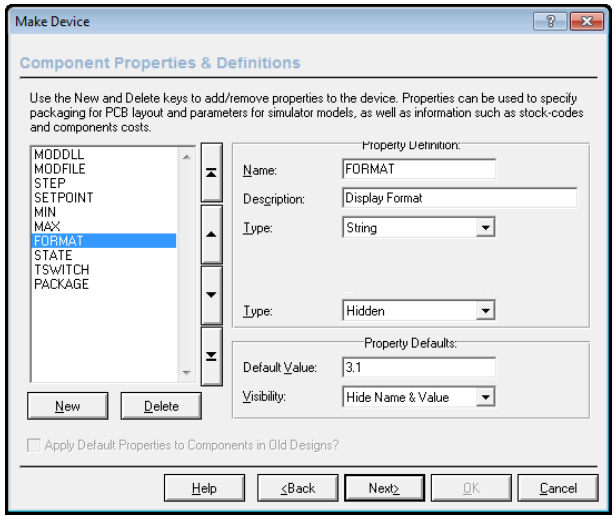


Рис. 37

FORMAT – название этого свойства тоже не нуждается в переводе, а относится оно к формату числа, выводимому в окошечко зеленого дисплея. Это подтверждается и тем, что в **Description** автор модели указал **Display Format** (Рис. 37). Обратите внимание, что верхний **Type** для этого свойства задан, как **String** – текстовая строка. Нижний **Type** тоже скрытый, поэтому оно видно только при установленном флажке **Edit all properties as text**. Ну а теперь о самом главном для этого свойства – **Default Value**. По умолчанию оно задано как **3.1**, что означает три знака до десятичной запятой и один после. При желании эти значения можно изменять, только следите за тем, чтобы общее число символов уместилось в зеленом окошке.

Следующие два свойства **STATE** – активное состояние при старте симуляции и **TSWITCH** – время переключения для большинства активных моделей на основе **SETPOINT** в основном будут иметь именно те значения, которые показаны на рисунках 36 и 37 соответственно. Первое из них отвечает за то, какой параметр будет активным (доступным для изменения) при старте симуляции. Поскольку он у нас всего один, то и **STATE=0**. Ну а время переключения **TSWITCH=1ms** – это то время, за которое наше значение изменится после нажатия на соответствующую кнопку-стрелку. Соответственно им тоже задан нижний **Type** скрытый, а в ряде случаев ими, в частности **TSWITCH**, можно и вообще пренебречь и при создании своих активных моделей не указывать.

Ну а мы на этом закончим нудную, но нужную для понимания того, что мы будем делать ниже описательную часть, и переходим к практическим действиям – созданию активных моделей на основе **SETPOINT.DLL**.

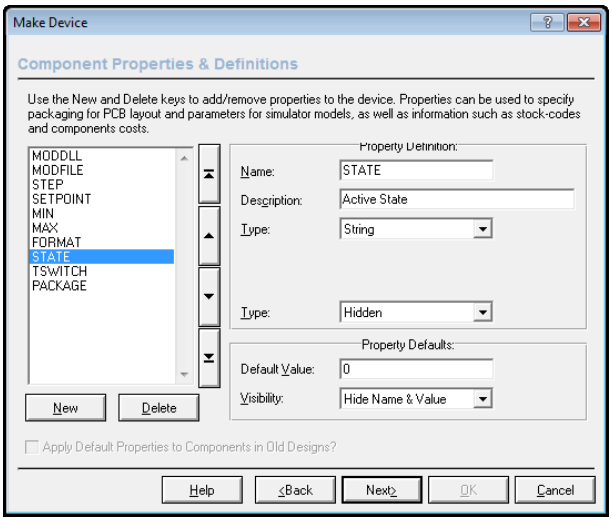


Рис. 36

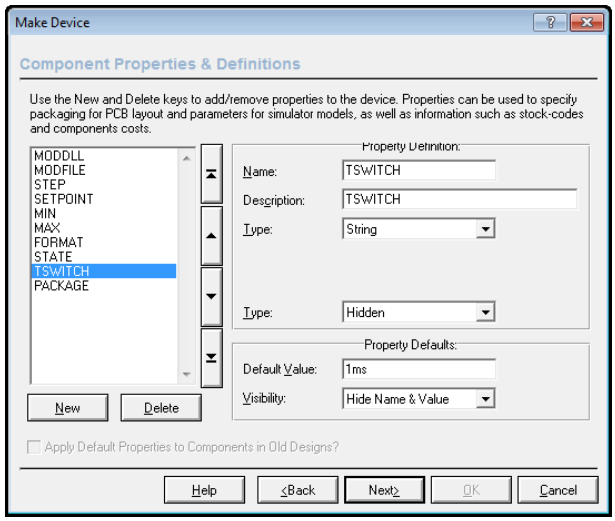


Рис. 37

К содержанию

8.3. Простые регулируемые источники на основе SETPOINT.DLL.

Отлаживая свои проекты в среде Протеуса, как то я обратил внимание на некоторое упущение разработчиков программы, доставляющее массу неудобств. В **ISIS** абсолютно отсутствуют готовые средства, с помощью которых можно в процессе выполнения симуляции изменять напряжение или ток, в какой либо цепи. Конечно, можно их реализовать с помощью обычных генераторов и тех же активных моделей потенциометров, но это получается настолько громоздко и неудобно в использовании, что весь эффект применения сводится на нет. И вот однажды, отлаживая

очередную схему, в которой присутствовала активная модель датчика температуры, я подумал – а нельзя ли подобным образом реализовать, например, изменяемый источник напряжения для питания какой либо части схемы. И удобно, и наглядно. Ну и дальше все просто, «подумано» – сделано.
Я изучил, как сделана модель датчика температуры, что мы проделали выше, а затем заглянул в ее файл **MDF**. Для любопытствующих я поместил уже извлеченный из **NATDAC.LML** файл **LM20.MDF**. И тут меня ждал весьма неприятный сюрприз. В **PARTLIST** файла модели (см. ниже) фигурировал элемент **VOUT**, который отсутствует в библиотеках Протеуса. Пошарив в других **MDF**, использующих **SETPOINT.DLL**, я обнаружил его во всех списках компонентов этих моделей.

```
*PARTLIST,6
AVS1,AVS,"(-3.88E-006*V(A,B)^2)+(-1.15E-002*V(A,B))+1.8639",PRIMITIVE=ANALOGUE
C1,CAPACITOR,1uF,PRIMITIVE=ANALOGUE
I1,CSOURCE,10uA,PRIMITIVE=ANALOGUE
R1,RESISTOR,10k,PRIMITIVE=ANALOGUE
R2,RESISTOR,100,PRIMITIVE=ANALOGUE

V1,VOUT,1V,MODDLL=SETPOINT,PRIMITIVE=ANALOG,SETPOINT=<SETPOINT>
```

Поначалу меня такой сюрприз обескуражил, но логически помыслив, я пришел к выводу, что раз он так широко используется, то, скорее всего, реализован в виде встроенного в среду примитива и можно попробовать «обмануть» Протеус, подсунув ему собственную модель с теми же свойствами. Попробуем реализовать его на основе двухполюсника генератора напряжения **VSOURCE** из библиотеки **Simulator Primitives**. Функционал – источник постоянного напряжения у них совпадает, значит осталось только придать прототипу недостающих свойств. Помещаем извлеченный из библиотеки **VSOURCE** в поле проекта и применяем к нему мою излюбленную **Make Device**. Смотрим на последнюю строчку приведенного выше **PARTLIST**. Наша модель должна иметь имя **VOUT** и префикс **V**, что и прописываем на первой вкладке (Рис. 40). Напомню, что в ней все свойства разделены запятыми, а **VALUE** стоит третьим сразу после имени **VOUT**.

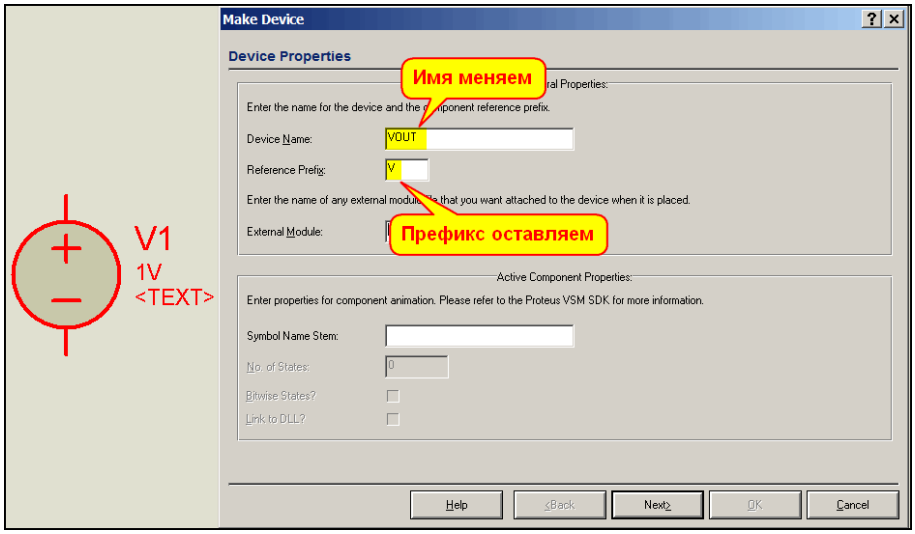


Рис. 40

Вторую вкладку проходим транзитом, поскольку корпус нашему девайсу не нужен и переходим на третью. Здесь убеждаемся, что уже присутствующие свойства **PRIMITIVE** (Рис 41) и **VALUE** (Рис. 42) соответствуют записанным в строке для **VOUT** из приведенного выше **PARTLIST**.

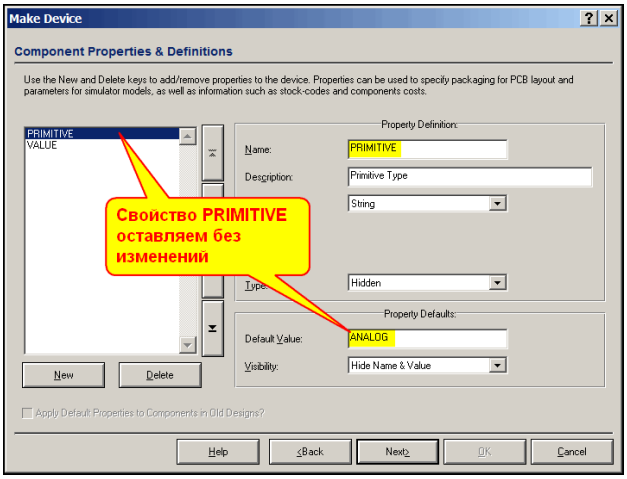


Рис. 41

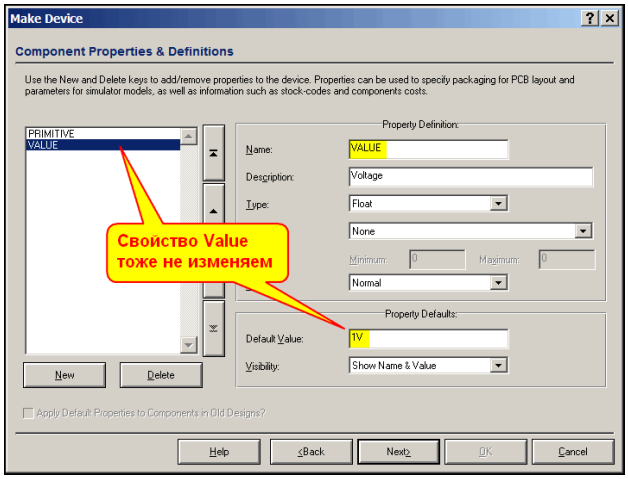


Рис. 42

Теперь добиваем через кнопку **New** недостающие **MODDLL** (Рис. 43) и **SETPOINT** (Рис. 44). Последнее, по большому счету, добивать необязательно, всегда можно добавить вручную в окне свойств компонента, но на первых порах, чтобы не забыть о его существовании - лучше вставить.

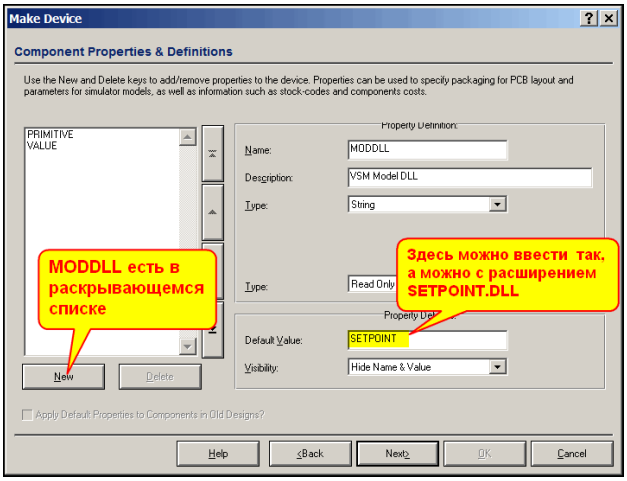


Рис. 43

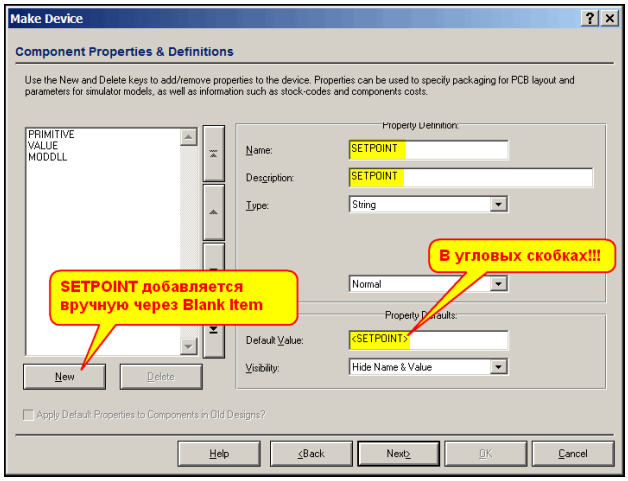


Рис. 44

Обратите внимание, что его **Default Value** стоит в угловых скобках (знаки больше-меньше на клавиатуре). Это дает команду Протеусу подставлять значение из внешнего по отношению к модели окружения. После этого доходим до последней вкладки и сохраняем наш VOUT в какой-либо библиотеке. Я тут не мудрствовал и сохранил в **USRDVC**, оставив все категории компонента, как и у **VSOURCE**, чтобы не забыть, где его впоследствии искать (Рис. 45).

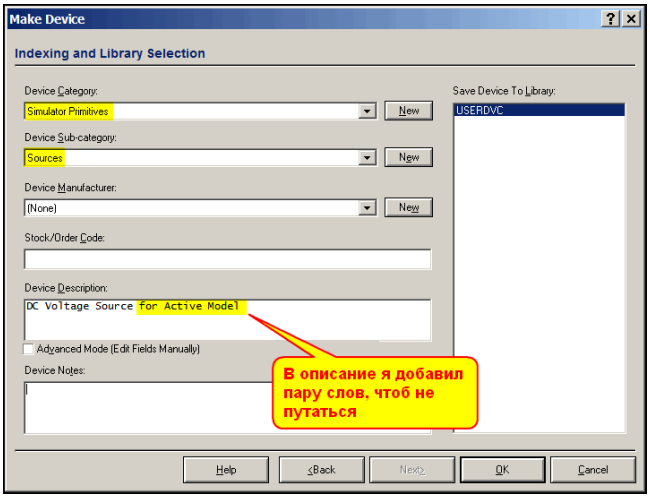


Рис. 45

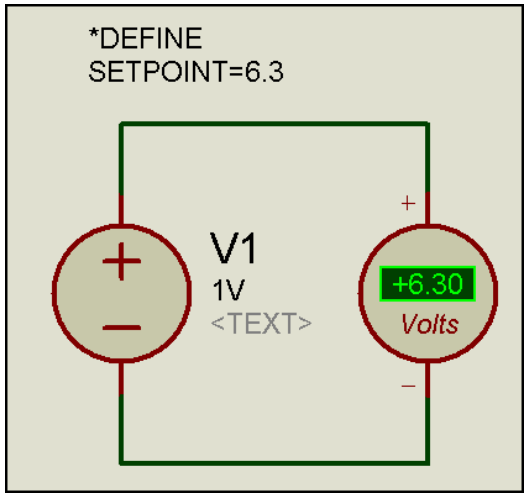


Рис. 46

Если теперь вытащить наш **VOUT** в поле проекта и запустить симуляцию, то вылезет желтое предупреждение в логе об отсутствии значения для **SETPOINT**, т.е. **ISIS** не нашел конкретного значения. Однако если поместить в поле проекта скрипт следующего содержания:

```
*DEFINE
SETPOINT=6.3
```

Наша модель начинает благополучно работать и «выдавать на гора» то напряжение, которое задано в скрипте. Такая подстановка вам уже знакома из создания схематичных моделей, но я счел нужным лишний раз напомнить – как это работает. В папке **Create_VOUT** присутствует проект, с которого сделан скриншот рисунка 46. Если кому то очень лень (некогда) заниматься такими пустяками, как создание **VOUT** – можете просто применить к стоящей в этом проекте модели **Make Device**, пройти ничего не меняя все вкладки и сохранить в собственной библиотеке Протеуса. Если ничего не трогать, то сохранится он в **USRDVC**, а в библиотеке его можно будет найти в **Simulator Primitives => Sources.**
Ну вот, теперь нам осталось создать активную графику для нашего будущего источника. Тут тоже не стоит мудрить – давайте «разберем на запчасти» все тот же **LM20** и воспользуемся его символами. Итак, помещаем в поле проекта **LM20**, **Decompose** его и видим, что у нас в селекторе символов появились два символа с именем **SETPOINT**: SETPOINT_0 и SETPOINT_1.

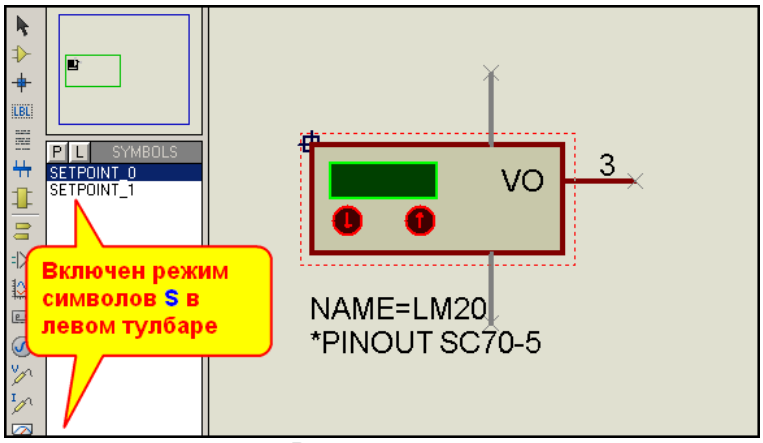


Рис. 47

Я не стану в этот раз заморачиваться с изменением символов, поскольку в активной модели, как я уже указывал, сдвиг графики относительно маркера **ORIGIN** повлечет за собой коррекцию всех остальных графических элементов. Поэтому, оставляем все как есть. На рисунке 48 приведены препарированные с помощью **Decompose** символы **SETPOINT_0** и **SETPOINT_1**, чтобы вы имели представление о местоположении маркера **ORIGIN**. Полностью разобранный на запчасти **LM20** в проекте **Prepare_LM20.DSN** папки **Create_VREG** вложения. Это как бы первый, подготовительный этап создания нашего источника. А мы переходим ко второму и заключительному.

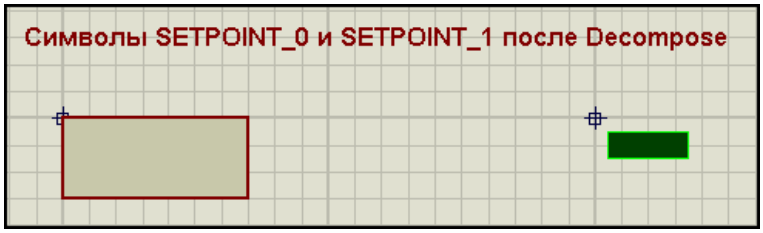


Рис. 48

Теперь нам необходимо убрать все лишнее из графики и добавить свое. Лишним в данном случае является скрипт и выводы компонента, в том числе и скрытые выводы питания, которые появились после разборки модели на запчасти. В результате у меня получилась «конструкция», показанная на рисунке 49 слева с которой я и начинаю выполнять создание нашего источника с помощью **Make Device**. На первой вкладке заполняем все так, как на рис. 49.

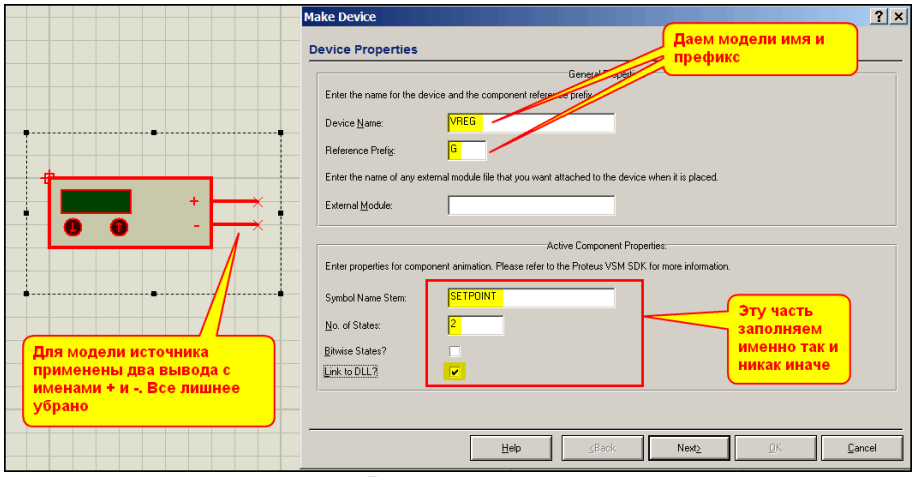


Рис. 49

Особое внимание правильности заполнения нижней части, относящейся к активным свойствам. Не забудьте, что символы **SETPOINT** должны быть доступны в селекторе **SIMBOLS** этого проекта. Я сделал так, вот тот предыдущий проект **Prepare_LM20.DSN** сохранил в этой же папке с новым именем **Create_VREG.DSN**, а после этого убрал все лишнее и добавил свои выводы (Pins). При этом символы остались доступными. Далее следуем в **Make Device** на третью вкладку и начинаем добавлять свойства по аналогии с **LM20**. У нас и так в этом параграфе переизбыток картинок, поэтому ограничусь моментом добавления **MODDLL**, при этом у нас окно свойств еще девственно чистое (Рис. 50). Не забудьте, что по умолчанию нижний **Type** для **MODDLL** будет предложен как **Read Only** (только для чтения). Это означает, что в окне свойств модели он будет показан, но не будет доступен для изменения (закрашен серым). Мне на это любоваться неохота, тем более что необходимо будет другие свойства сделать доступными, в отличие от **LM20**, поэтому здесь я выберу **Hidden** (скрытое).

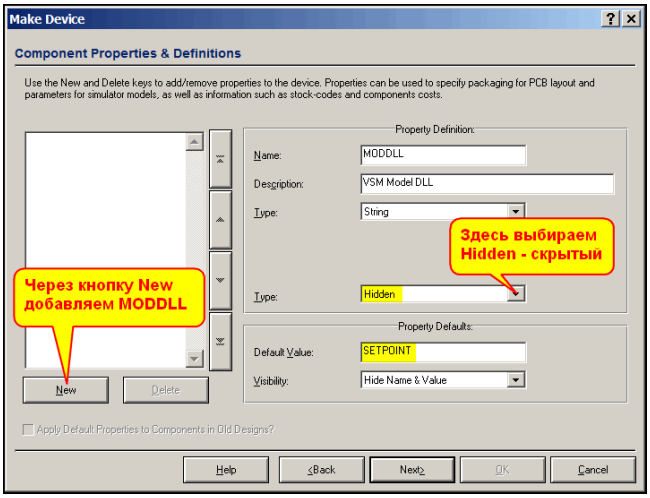


Рис. 50

Также, через кнопку **New** и опцию **Blank Item** обязательно добавляем следующие свойства:
STEP – в графе Description я набрал **Voltage Step** (Шаг напряжения), верхний **Type** – **Float**, нижний **Type** – **Normal**, **Default Value** – **0.1**.
SETPOINT – в графе Description я набрал **Current Voltage** (текущее значение напряжения), верхний **Type** – **Float**, нижний **Type** – **Normal**, **Default Value** – **0.0**. Обратите внимание, что ни здесь, нигде более я не воспользуюсь **Limits**, поскольку моя модель для отладки, но MAX и MIN заведомо достаточно большие я все же введу, иначе изменение значения с помощью кнопок будет криво работать.
MAX – в графе Description я набрал **Maximum Voltage** (Максимальное значение напряжения), верхний **Type** – **Float**, нижний **Type** – **Normal**, **Default Value** – **500**.
MIN – в графе Description я набрал **Maximum Voltage** (Минимальное значение напряжения), верхний **Type** – **Float**, нижний **Type** – **Normal**, **Default Value** – **-500**.
Я сделал максимальное и минимальное напряжения видимыми в свойствах, потому что вдруг да не хватит значения 500V, всегда можно поправить. Остальные свойства я пока вводить не буду, чтобы доказать, что и без них мы получим вполне работоспособную модель.

Доходим до последней вкладки и сохраняем нашу модель. Я сохранил ее в **Debugging Tools**, создав там дополнительную подкатегорию **Sources** (Рис. 51). Если кому то это не нравится, может сохранить в другом месте.

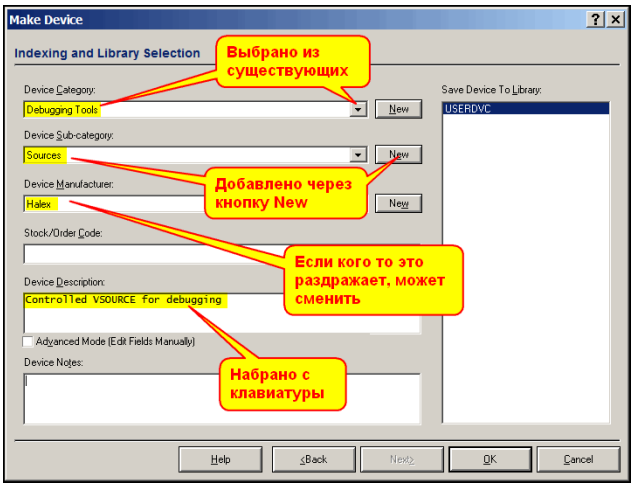


Рис. 51

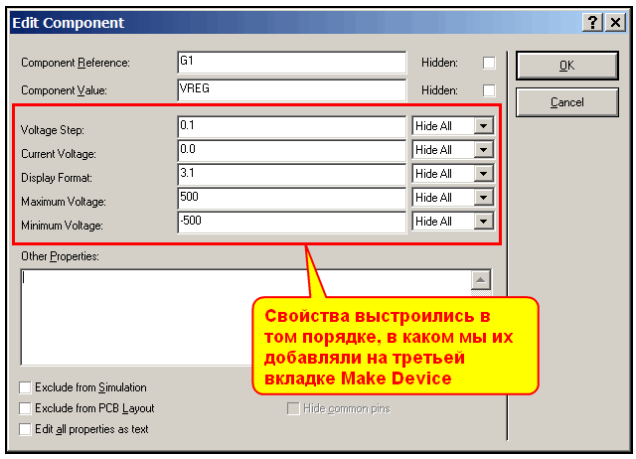


Рис. 52

Если теперь поместить нашу модель, появившуюся в селекторе компонентов в поле проекта и зайти в ее свойства, то мы увидим все наши добавки с третьей вкладки **Make Device**, причем в том порядке, в каком мы их добавляли там (Рис. 52). Обратите внимание, что **MODDLL** здесь нет, потому что для этого свойства мы выбрали нижний **Type** – **Hidden**. Порядок отображения свойств в этом окошке можно изменить. Если мы запустим еще раз **Make Device**, то на третьей вкладке можем переставить порядок расположения уже существующих свойств с помощью кнопок со стрелками справа от этого окна. Кликаем по свойству, которое надо передвинуть, и этими кнопками передвигаем его на нужное место. Средние стрелки передвигают на одну позицию, крайние верхняя и нижняя соответственно в начало или в конец. Естественно, чтобы изменения сохранились надо пройти **Make Device** до конца и подтвердить изменения при сохранении модели.
Теперь, как обычно, в свойствах устанавливаем флажок **Attach Hierarchy Module**, уходим на дочерний лист и там собираем «архисложную» субсхему нашего девайса (Рис. 53).

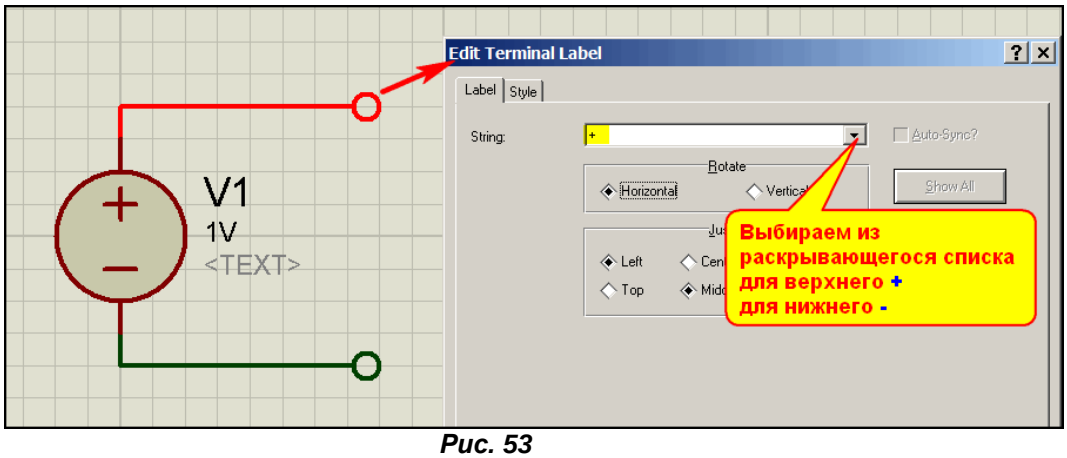


Рис. 53

Как видите, вся схема состоит из свежевыпеченного нами выше **VOUT** и двух терминалов, имена которых выбираются из раскрывающегося списка. А в списке будут фигурировать только наши два вывода модели с именами **+** и **-**. Ну вот и весь процесс, осталось вернуться на основной лист проекта, прицепить для контроля к выводам нашего источника вольтметр и запустить симуляцию. Пробуем кнопками менять напряжение и убеждаемся, что все работает как надо (Рис. 54).

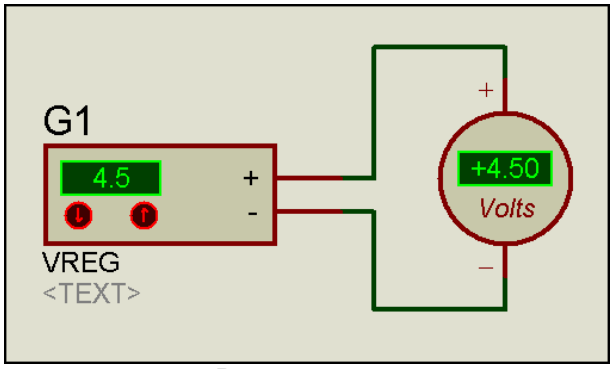


Рис. 54

Нам осталось скомпилировать с дочернего листа файл **MDF**, я назвал его **VREG.MDF**, затем запустить для нашего источника еще раз **Make Device** и на третьей вкладке добавить свойство **MODFILE**, указав ему в графе **Default Value** имя нашего скомпилированного файла. Мы это уже неоднократно проделывали при создании схематичных моделей, и, надеюсь, тут разъяснять дополнительно ничего не требуется. Окончательный вариант с файлом **MDF** представлен в папке **VREG_with_MDF** вложения.

Ну а мы теперь модифицируем наш источник и превратим его в источник тока. Модификация заключается в том, что на дочернем листе нам придется добавить еще один примитив – управляемый напряжением источник тока **VCCS** или **AVCCS** с коэффициентом передачи **1.0** (Рис. 55). После этого компилируем с дочернего листа новый файл **IREG.MDF**. Сама графика остается без изменений, но необходимо вновь пройти **Make Device**. На первой вкладке даем нашей модели другое имя, например, **IREG**. На третьей придется поправить **Description** у большинства свойств, заменив там слово **Voltage** (напряжение) на **Current** (ток), чтобы потом самим не путаться. Ну, можно еще сменить ограничения для **MIN** и **MAX** на разумные, поскольку ток в 500А нам на фиг не нужен. Ну и конечно, для **MODFILE** надо указать новое значение **IREG.MDF**. Проект с дочерним листом представлен в папке **Create_IREG** вложения. А в папке **IREG_with_MDF** уже модель с присоединенным **MDF** и сам **IREG.MDF**.

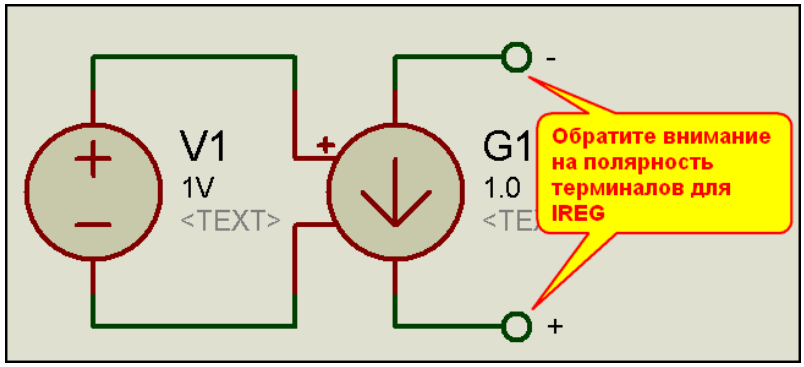


Рис. 55

Ну и несколько заключительных слов по использованию этих моделей. Я пошел навстречу «идолопоклонникам» русского интерфейса и в папке **RUS** вложения лежат проекты, в которых **Description** моделей сделан кириллицей. Для них в конце имен я добавил **RUS**. Причем файлы MDF для них остаются неизменными. Меняется только **Description** на третьей вкладке **Make Device**. Вам остается только выбрать то, что вы хотите использовать в своих проектах, протолкнуть для выбранных моделей **Make Device**, ничего не меняя во вкладках, чтобы эти модели появились в ваших библиотеках **ISIS** и переложить файлы **VREG.MDF** и **IREG.MDF** из выбранных папок в папку **MODELS** Протеуса. Обращаю ваше внимание, что если вы перед применением **Make Device** запускали симуляцию, изменили напряжение или ток, сменили в окне свойств какие либо параметры, например, **STEP** или **FORMAT**, то модель сохранится именно с этими параметрами, если конечно вы их не исправите на третьей вкладке к другим значениям.

К содержанию

8.4. Модель датчика давления с выходом 4-20мА на основе SETPOINT.DLL.

Я бы изменил своему нынешнему профессиональному эго, если бы не поделился с вами еще одной своей наработкой в области моделирования. По своей сфере теперешней деятельности – проектирование и монтаж автоматики мне постоянно приходится работать с различными датчиками: температуры, давления, расхода и т.д. и т.п. Львиную долю датчиков давления среди всего этого разнообразия занимают датчики с типовым выходным токовым сигналом – 4-20мА. Конечно, можно

использовать в этих целях и обычные примитивы генераторов тока, но это обычно доставляет массу неудобств при моделировании поведения девайса в реальном времени. Каждый раз при смене значения придется полностью останавливать симуляци. «Прикручивание» к ним активных моделей потенциометров тоже не меняет дело к лучшему. В конечном итоге, как правило схемы для моделирования представляла собой нечто вроде изображенных на рисунке 56.

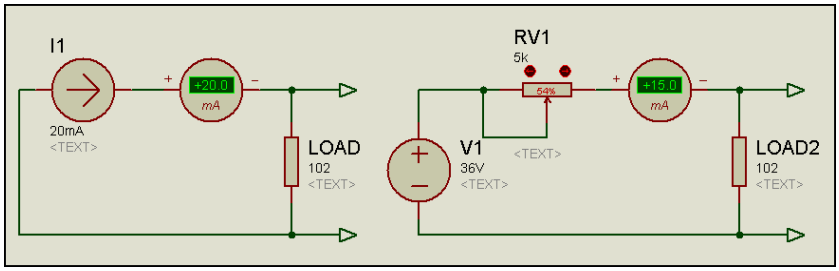


Рис. 56

Мало того, что такие «примочки» в проекте занимают много места, так еще приходится держать перед глазами и таблицу пересчета давления в ток или иметь под рукой калькулятор для этих целей. Естественно меня такое положение дел не устраивало. Ну и сам собой напрашивался прецедент иметь в **ISIS** модель универсального датчика давления для разработки своих устройств. **SETPOINT.DLL** для этих целей подходит как нельзя лучше. Для начала немного арифметики. Предположим, мы имеем датчик избыточного давления от 0 до 100кПа (ну или 0–1кгс/кв.см, что практически тоже самое). Для покрытия этого диапазона нам отводится токовый диапазон от 4мА (соответствует 0) до 20мА (соответствует верхнему пределу), т.е. 20-4=16мА. Разделив 16 на 100 получим, что изменению давления на 1кПа соответствует изменение тока на 0,16мА. Эта ориентировка поможет нам в дальнейшем.
Теперь создаем нашу модель. Художник из меня еще тот, но все же я постарался изобразить нечто напоминающее реальный датчик (Рис. 57).

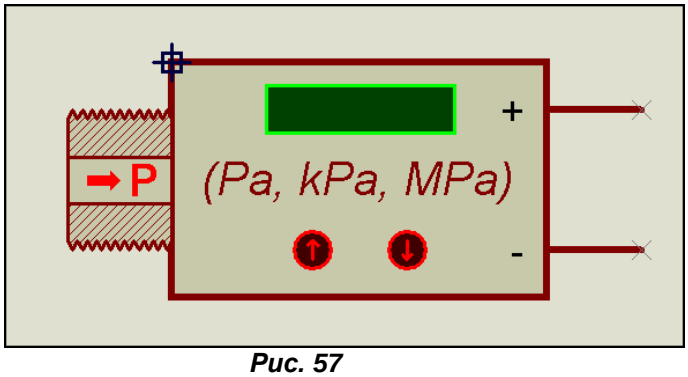


Рис. 57

Поскольку в данном случае графика модели создается «с нуля» остановлюсь на некоторых нюансах. Изображение резьбы в разрезе выполнено с помощью **Closed Path** из левого тулбара, затем в свойствах изображения снимаем галочку с **Fill style** и выбираем режим косой штриховки (Рис. 58). Там же, сняв флажок **Width,** выбираем более тонкий контур для обрисовки объекта - 8th.

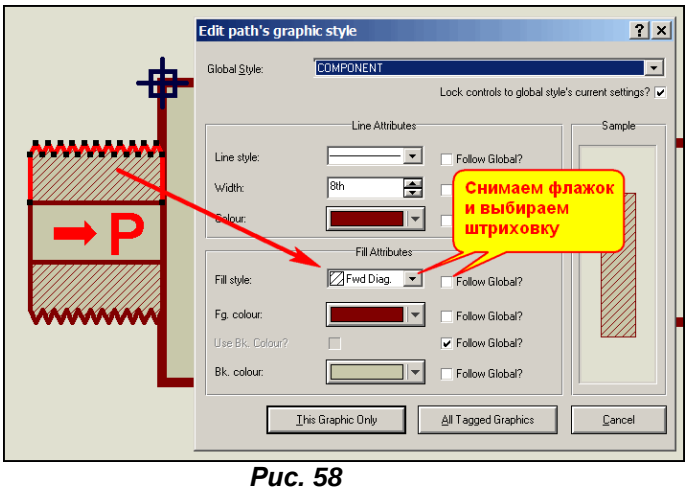


Рис. 58

Индикатор в данном случае тоже нарисован свой. Для этой цели в режиме рисования прямоугольников выбираем в селекторе опцию **INDICATOR** и рисуем наш экран в нужном месте. Чтобы не ошибиться с размерами – лучше поместить рядом что то из реальных моделей, содержащих аналогичный микродисплей, например, – вольтметр. В принципе, вы можете нарисовать индикатор хоть на поллиста проекта, но все же необходимо учитывать, что его параметры взаимодействуют с **SETPOINT.DLL**, а там о них уже до нас позаботился программист, создавший эту библиотеку. Поэтому мы будем иметь тот шрифт и цвет, которые уже есть, ну а максимально индицируемое число, я думаю, вам не составит труда определить самостоятельно, если вспомнить, что мы его в свойствах задаем как **float**.

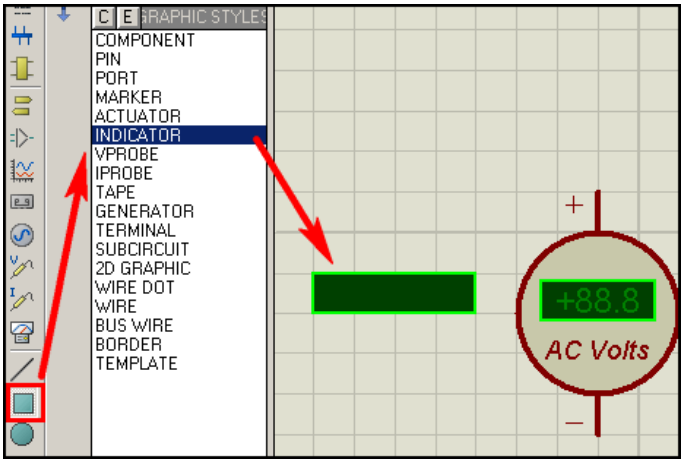


Рис. 59

Ну и, наконец, о маркерах инкремента и декремента. Поскольку это маркеры, мы их и берем из селектора в режиме **2D Graphic Markers Mode** (Рис. 60).

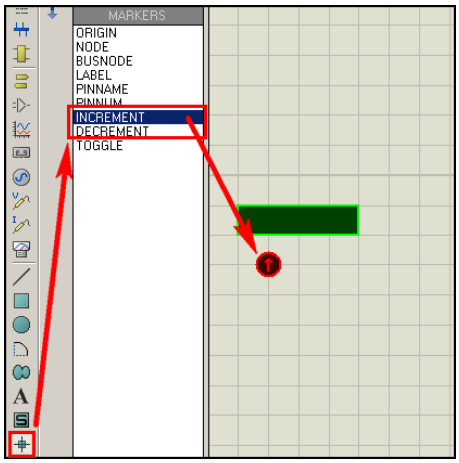


Рис. 60

С остальной графикой, надеюсь, проблем не будет. Надписи помещены чисто в «напоминательных» целях и никаких подводных камней в них не зарыто. Выводам присвоены только имена «**+**» и «**-**» , нумерация нам не нужна, поскольку корпуса эта модель содержать не будет. После того, как прорисована вся графика полностью, выделяем все ее элементы с помощью обводки с зажатой левой кнопки мышки и через **Block Copy** размножаем в двух экземплярах для создания символов. На одной копии удаляем выводы, экран и маркеры инкремента/декремента – это будет база, символ с индексом **_0**, а на другой оставляем только экран и маркер **ORIGIN** – это будет символ с индексом **_1** (Рис 61). Затем из того, что изображено на этом рисунке создаем символы. Я назвал их **SENSOR_0** и **SENSOR_1**. Создав символы, убедитесь, что при переходе в режим символов (кнопка **S** в левом тулбаре) они доступны в селекторе. Только после этого можно приступать к созданию самого девайса из того полного графического изображения вместе с выводами, что я привел на рисунке 57.

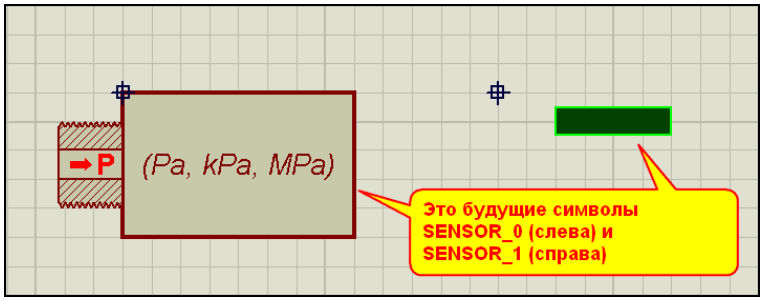


Рис. 61

Сразу же хочу обратить ваше внимание, что вот такой порядок создания графики для активных компонентов заведен и у самих разработчиков в Лабцентре. Т.е. сначала прорисовываем все в совокупности, затем копируем через **Block Copy** и тупо тыкаем левой кнопкой в нужных местах, размножая нужное количество раз полное изображение. Ну, а уж потом, как скульпторы, на копиях отсекаем (двойной клик правой) все ненужное для каждого конкретного символа. При этом минимален риск получить неправильное позиционирование нашего символа относительно маркера **ORIGIN**, если конечно сдуру нечаянно не кликнуть пару раз правой и по нему. Мне этот «фокус» лабцентровцев подсказал Тень, хотя, к тому времени я уже и своими извилинами «дополз», что так получается быстрее. Особенно прирост скорости скажется при создании многоэлементных индикаторов или клавиатур, чем мы займемся чуть позже. Но привыкать к такому порядку я призываю заранее.
Итак, с графикой разобрались. Теперь мой любимый **Make Device**. На первой вкладке задаем имя нашему девайсу и задаем параметры активной графики (Рис. 62). Ну, тут все, как и с источниками, за исключением того, что символы у нас теперь с собственными именами.
Да и на третьей странице мало что изменилось, разве что в **Description** я поставил другие названия, например для самого **SETPOINT** там стоит **Actual Pressure** – текущее давление (Рис. 63).

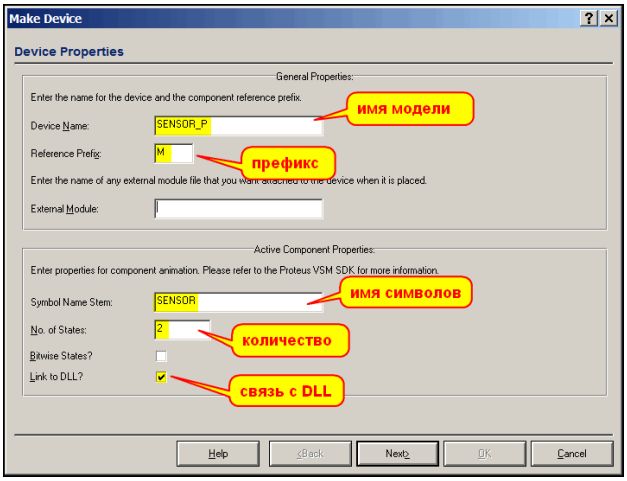


Рис. 62

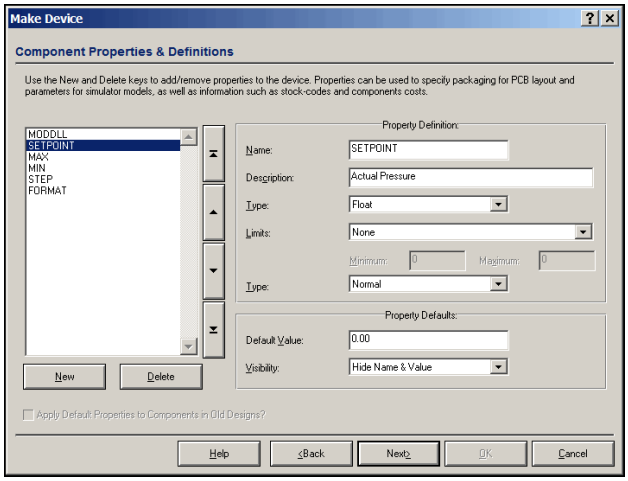


Рис. 63

Я не стану здесь приводить все скриншоты с третьей вкладки, желающие могут открыть пример **Graphic_Child.DSN** в папке вложения **Simple_Model**, войти в свойства или запустить **Make Device** для уже готовой модели и посмотреть на третьей вкладке – что и как я «обозвал». К данному моменту это вы уже должны усвоить это как дважды-два. Можете также перевести **Description** на русский, мне, честно говоря, это делать лень, тем более, что меня никакими коврижками не заманишь пользоваться «русифицированной» версией. Боюсь, что даже если Тень когда-либо сподобится раскрутить свое начальство на выпуск официальной русской локализации интерфейса. После нескольких лет работы настолько привыкаешь к оригиналу, что «уписанное» на место лаконичной английской фразы наше родное, вечно не помещающееся в строку и поэтому урезанное до невозможности типа «Тпр. пр. гр.» расшифровать гораздо труднее, чем полную аглицкую фразу. Ну, это так, небольшое отступление от темы, а мы создаем девайс до конца, сохраняем и как обычно в свойствах «приклеиваем» дочерний лист. На нем создаем внутреннюю структуру нашего девайса, или **Test jig**, как это именуется в оригинальной документации VSM SDK (Рис. 64).

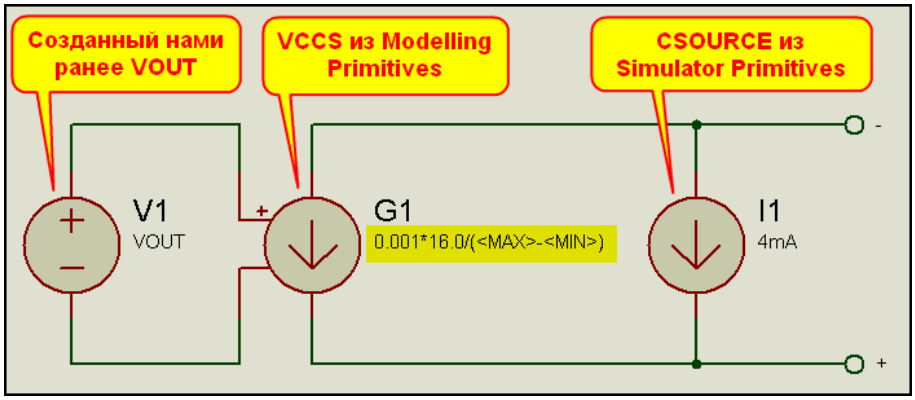


Рис. 64

Давайте разберем эту «джигу» по косточкам. Связка первых двух компонентов аналогична управляемому источнику тока, рассмотренному ранее, вот только у G1 в передаточной характеристике стоит «хитрая формула». Я ее нарочно расписал поподробнее, чтобы было проще объяснять. Первый множитель – **0.001** переводит наше значение тока в миллиамперы. Стоящая в скобках разность **<MAX>-<MIN>** дает нам полный диапазон изменения давления нашего датчика. Причем, мы вольны задавать им любые положительные значения или ноль для минимума. А отношение **16.0** (полный диапазон тока, мы определили его выше) к диапазону изменения давления даст нам коэффициент пересчета давления в ток. Таким образом, **VOUT** будет менять свое значение от **MIN** до **MAX**, а **G1** пересчитывать его в ток от нуля до **16мА**. Источник **I1** с током **4мА** в сумме сдвинет нам диапазон в **4-20мА**, что нам и надо. У кого затруднение с пониманием параллельного включения **G1** и **I1** – попросите у детей физику за 7-й класс и перечитайте законы Кирхгофа. Хотя, это я официально учил в 7-м, а неофициально – самостоятельно в 4-м, а что там сейчас учат, одному директору школы известно. Ох, опять в ностальгию ударился. Идем дальше…
Ну, в общем, идем на **Parent List** запускаем и тестируем нашу модель, используя различные диапазоны значений давления. Все это доступно в упоминаемом выше примере. Я не стал вводить ограничения для **MIN** и **MAX** на третьей вкладке, чтобы вы могли убедиться, что при использовании отрицательных значений для **MIN** мы получим полный бардак на выходе. Если кому то нужна полностью работоспособная модель с такой внутренней структурой, то рекомендую «мэйкануть» ее еще раз и на третьей вкладке для **MIN** выбрать в **Limits** из раскрывающегося списка ограничение **Positive Or Zero** (положительное или ноль), А для **MAX** предел **Positive Non-Zero** (положительное, не ноль). Затем с дочернего листа скомпилировать **MDF** и на третьей вкладке добавить его через **MODFILE**.
Я же, пошел дальше и решил сделать более универсальную модель – датчик лавления/разрежения или на языке КИПовцев – тягонапоромер. Если ей задать нижний предел ноль – это обычный датчик давления, если меньше нуля – тягонапоромер. Поменялся при этом только дочерний лист, а что за «джига» у меня получилась показано на рисунке 65, а также в примере **Uni_Sens_1.DSN** из папки **Uni_Sensor_with_Child**.

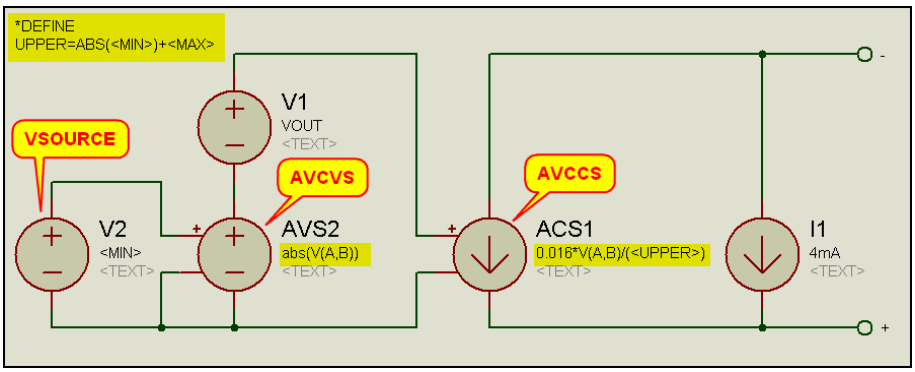


Рис. 65

Какие здесь встретились заморочки. Во-первых потребовалось сдвинуть диапазон VOUT d область положительных значений. С этой целью введены **V2** и **AVS2**. Почему получилось так сложно? Просто для обычных источников функция **abs** (абсолютное значение или по другому модуль) не работает, она предназначена только для **Arbitrary Source** (см. материал из предыдущих частей или HELP по примитивам). С этой же целью и **ACS1** тоже взят **arbitrary**. Там для обычного **VCCS** почему то не работала подстановка диапазона **<UPPER>** из скрипта ***DEFINE**. Формулу я упростил, перемножив заранее **0,001** на **16**. Для этой модели я уже скомпилировал MDF и приложил для

тестирования в папке **Uni_Sensor_MDF**. Там модель уже с присоединенным **MODFILE**. Достаточно протолкнуть для нее **Make Device** ничего не меняя и переложить **MDF** в папку **MODELS** для использования в своих разработках. Вот здесь я для **MAX** ввел **Limits**, а для **MIN** не стал. Вряд ли кому придет в голову делать **MIN** больше **MAX**, хотя у наших доморощенных умов…
На этом абстрактный материал по **SETPOIN**T закончен, но, пожалуй, добавлю далее еще одну модельку из Reality Show, тем более она мне и самому понадобилась.

К содержанию

8.5. **Модель датчика давления** Freescale MPX5010 **из модели** MPX4250.

Предыстория создания этой модели заключается в том, что мне захотелось иметь персональный электронный манометр для контроля давления природного газа на бытовых газовых котлах и промышленных газовых горелках. Есть у нас на работе и поверенные сертифицированные, но бывает так, что под рукой отсутствует, а срочно нужен. Поскольку для этих целей я буду тратить «свои кровные», захотелось создать дешевую и доступную по комплектующим конструкцию. В качестве датчика давления мне приглянулся недорогой **MPX5010** от **Freescale** (он же бывший **Motorola**). Конечно, для портативного прибора лучше было бы использовать более современный **MP3V5050** с трехвольтовым питающим напряжением, но как сказал слесарь Полесов: «при наличии отсутствия пропитанных шпал…» придется обойтись тем, что есть в продаже.
В Протеусе датчики давления представлены всего двумя довольно архаичными моделями в библиотеке **Transdusers=>Pressure**. Это датчик абсолютного давления **MPX4115** с пределом 15…115кПа и датчик относительного давления **MPX4250** с пределами 0…250кПа. Если кто-то первый раз сталкивается с датчиками давления, поясню, что датчики абсолютного давления используются в основном для контроля атмосферного давления в тех же электронных барометрах и метеостанциях (например, любимые нашими синоптиками 760 мм ртутного столба соответствуют 101,308кПа). Датчик относительного давления контролирует превышение давления на его входе относительно атмосферного. Такие датчики применяются для контроля давления в трубопроводах, резервуарах, автошинах и т.п. Мне нужен датчик относительного давления. Предел в 250 кПа меня мало устраивает, мне достаточно 10, но «за основу», как говорили всякие партократы-бюрократы эту модель принять можно, тем более, что сделана она довольно профессионально и заслуживает рассмотрения с точки зрения внутренней структуры. Подключение модели и выдаваемое ею напряжение при минимальном и максимальном давлении показано на рисунке 66.

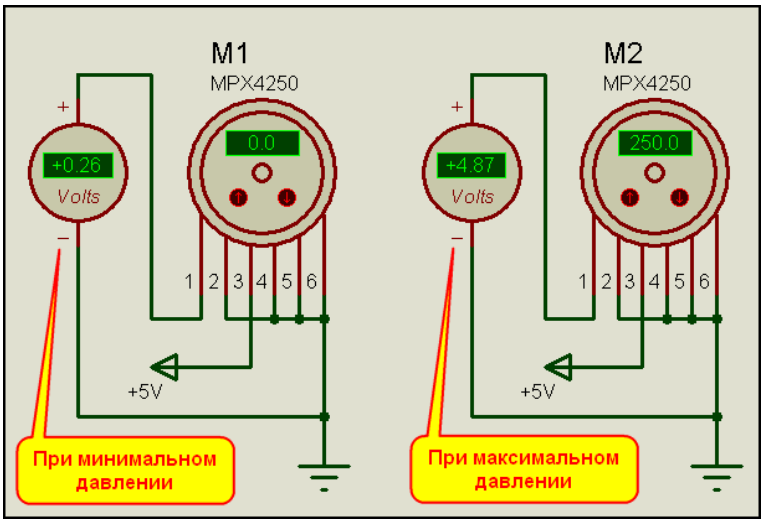


Рис. 66

Для начала заглянем в окно свойств модели этого датчика (Рис. 67) и сопоставим их с даташитом.

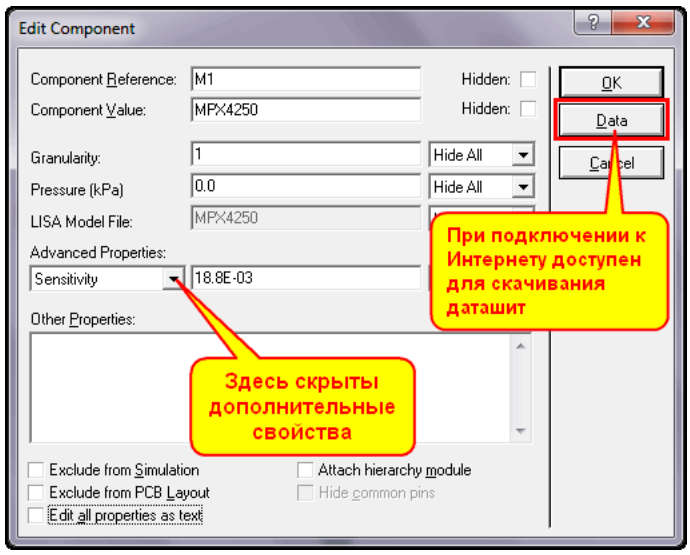


Рис. 67

В основном окне доступны для изменения только два параметра **Granularity** (шаг изменения) и **Pressure** – стартовое значение давления. Модель построена на основе все того же **SETPOINT**, поэтому назначение этих параметров не вызывает затруднений. А вот в раскрывающемся списке **Advanced Properties** «зарыты» именно свойства связанные с даташитом датчика, который, кстати, можно скачать, нажав кнопку **Data**. Рассмотрим их:

Sensitivity – **SENS** (здесь и ниже черным жирным указано как они трактуются в режиме **Edit all Properties as text** и на третьей вкладке **Make Device**). Чувствительность датчика – типовое значение 18,8 мВ/кПа по таблице **OPERATING CHARACTERISTICS** на стр. 2 даташита. Соответственно в Протеусе эта запись будет выглядеть как **18.8E-03**.

Offset – **OFFSET** – смещение. Значение 0,04 фигурирует в формуле **Transfer Function** (передаточная функция) на стр. 3 и 4 даташита.

Pressure Error Band – **PEB** – диапазон ошибки давления максимальное значение 3,45 кПа приведено на графике стр. 4 даташита.

Temperature – **TEMPERATURE** – указанное в свойствах значение TEMP (по умолчанию 27°C) берется из параметров симулятора (меню **System=>Set Simulation Options** вкладка **Temperature**) и позволяет имитировать поведение датчика в зависимости от температуры. О температурном моделировании я уже рассказывал во второй части FAQ.

Теперь восстанавливаем структуру по файлу MDF (Рис. 68).

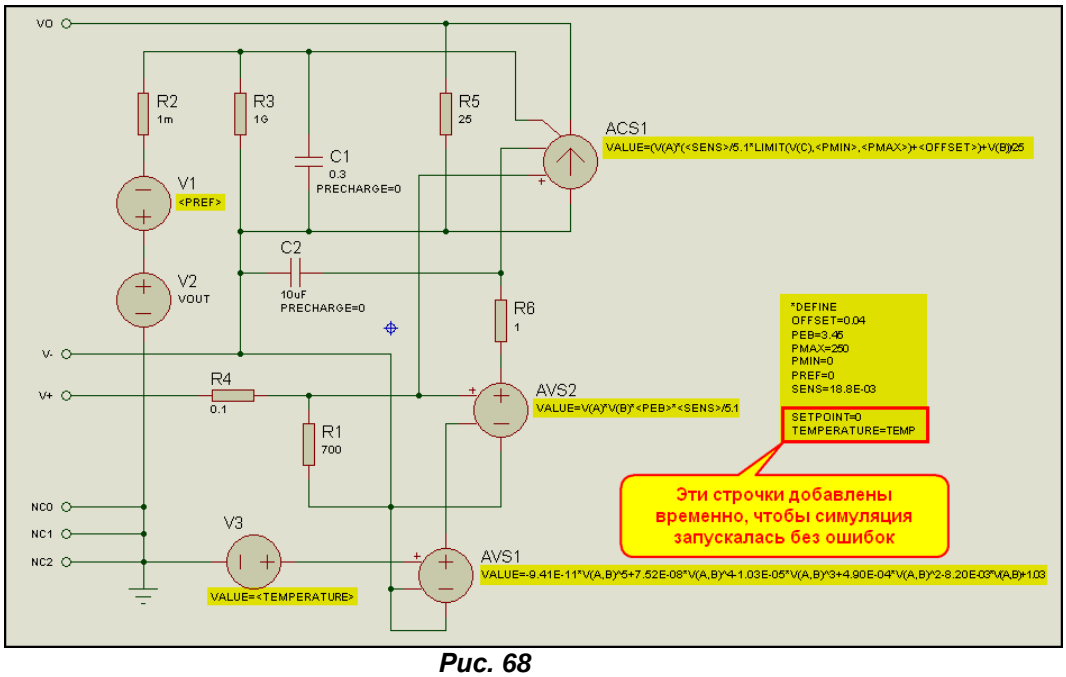


Рис. 68

Так, предчувствую скептические замечания, - опять автор «чудит». Где он взял эту сороконожку – **ACS1** – в примитивах такого нет. Да просто надо внимательно читать HELP по примитивам.

Открываем раздел **The Arbitrary Controlled Source Models - AVS, ACS** и находим там следующий абзац:

> Voltage input values are referred to as **V(A), V(B), V(C)** etc. within the expression, these values referring to the voltages at pins named A, B, C. The form **V(A,B)** is also supported, this meaning the differential voltage between pins A and B.

В вольном переводе это звучит следующим образом:
*Значения входных напряжений описываются в выражении как **V(A), V(B), V(C)** и т.д., где значения соответствуют напряжениям на ножках (пинах) с именами A, B, C. Форма записи **V(A,B)** также поддерживается и соответствует разности напряжений между выводами A и B.*
Как видим, в стандартных примитивах все **Arbitrary Source** с двумя входами – A и B, хотя их может быть … и т.д.. А вот это самое **V(C)** фигурирует в выражении для **AVS1** в MDF модели датчика (выражение выделено желтым цветом на рисунке 68). Но вывода C у примитива не было, пришлось срочно «припаять». Так что ничего сверхъестественного – я просто разобрал (**Decompose**) существующую модель **AVCCS** и приделал к ней еще один вход с именем **C**, ну а потом «мэйк девайснул» свое новое творение. Верхний слева по схеме вывод и есть этот пресловутый C. Для большей компактности схемы мне пришлось отразить модель **AVS1** по вертикали и поэтому выводы входов получились снизу вверх: A, B, C. Именно этот примитив и является основным в модели датчика, и в выражении, описывающем его передаточную характеристику, заложена основная формула преобразования.
Вообще данная модель, с моей точки зрения, обладает даже некоторой избыточностью. Попробую описать назначение остальных узлов схемы, как я их понял. Основным управляющим элементом является все тот же **VOUT V2**, связанный с **SETPOINT.DLL**. Источник **V1** со значением PREF=0 вероятно был заложен для коррекции передаточной характеристики в зависимости от **VOUT**, но поскольку в конечном итоге он оставлен с нулевым напряжением, это явное излишество. Источник **V3**, значением которого является температура, и связанный с ним источник **AVS1** с длинным полиномом в передаточной характеристике, призваны имитировать поведение датчика при температурном анализе. Ну и, наконец, источник **AVS2**, передаточная характеристика которого зависит от напряжения питания, имитирует поведение датчика при изменении питающего напряжения.
Восстановленная из MDF структура **MPX4250** приложена в проекте **Stucture_MPX4250\ Struct_MPX4250.DSN** вложения. В этой же папке приложен и извлеченный из библиотеки **TRXD.LML** файл **MPX4250.MDF**.
Я же приступаю к подгонке структуры под свои нужды, т.е. созданию **MPX5010**. Как я уже предупреждал, источник **V1** отправляю на свалку, соответственно в свойствах упраздняется параметр **PREF**. Максимальное значение в свойствах изменяем с **MAX=250** на **MAX=10**, минимальное остается неизменным. Ошибка давления для новой модели приведена на стр. 6 соответствующего даташита, и составляет: **PEB=0.5** kPa. Несколько сложнее обстоит дело с чувствительностью. В таблице характеристик датчика MPX5010 даташита есть небольшая «очепятка» верхнее значение для **Sensitivity** это как раз 450 mV/kPa, а не какие то там подозрительные mV/mm. Нижнее значение для миллиметров водяного столба указано правильно. Как видите, и на **Freescale** «бывает проруха», это лишний повод мне напомнить, что своя «соображалка» должна быть всегда начеку. Ставим в свойствах **SENS=450E-03.**
Далее необходимо заняться формулами, описывающими передаточные характеристики источников в структуре **MPX4250**. Нам необходимо сопоставить, что там требуется заменить и как в новой модели. **MPX5010**.
Для начала обратим внимание на передаточную характеристику (**Transfer Function**), представленную в даташитах на датчики.
Для **MPX4250** она описывается как: **Vout = Vs x (0.00369 x P + 0.04) ± Error**
где: **Error = Pressure Error x Temp. Factor x 0.00369 x Vs**
Vs= 5.1 ± 0.25 Vdc
Для **MPX5100** она описывается как: **Vout = Vs x (0.09 x P + 0.04) ± Error**
где: **Error = Pressure Error x Temp. Factor x 0.09 x Vs**
Vs= 5.0 ± 0.25 Vdc
Ну, даже беглый взгляд, сразу подсказывает, что там, где в формулах для **MPX4250** стоит напряжение 5,1 для **MPX5010** нам надо поставить ровно 5.
Длинный «член» в AVS1 оставим неизменным, поскольку, если внимательно посмотреть температурные зависимости в даташитах обоих датчиков, то они полностью совпадают. Получившаяся структура представлена в проекте **Struct_MPX5010.DSN** из папки **Stucture_MPX5010** вложения. Далее надо создать графическую модель MPX5010, и поменстить нашу структуру на дочерний лист для тестирования. Мудрить с графикой я не стал. Поступаем просто: берем модель **MPX4250** и применяем к ней **Make Device**. На первой вкладке переименовываем в **MPX5010**, на

третьей удаляем, ставший ненужным параметр **PREF**, а также временно удаляем **MODFILE**, позже мы присоединим новый. Там же правим значения остальных свойств: **MAX**, **PEB**, **SENS**, как было указано выше.Ну и перед сохранением на последней вкладке в окне **Device Description** исправляем предел с 250 на 10 kPa, чтобы потом не путаться самим. Теперь как обычно помещаем нашу модель в проект, присоединяем к ней дочерний лист и помещаем туда нашу структуру. Этот вариант представлен в папке **MPX5010_Child** вложения.

Тестируем, затем с дочернего листа компилируем MDF и еще раз пройдясь через **Make Device** присоединяем на третьей вкладке наш **MODFILE**. Готовый вариант с MDF представлен в папке **MPX5010_MDF** вложения.

Ну и под занавес я не удержался от того, чтобы проверить – как влияет вот тот длинный «член»-полином температурной зависимости в **AVS1**. Этот проект представлен в папке **MPX5010_TEMP**. Для того, чтобы получить зависимость выходного напряжения от температуры, я воспользовался графиками **DCSWEEP**. В свойствах датчика для **Temperature** подставляем вместо **TEMP** значение **X**, а в свойствах графика для X задаем значения от -40 до 125 – как в даташите. Я сделал два графика при нулевом и максимальном давлении. Они представлены на Рис. 69.

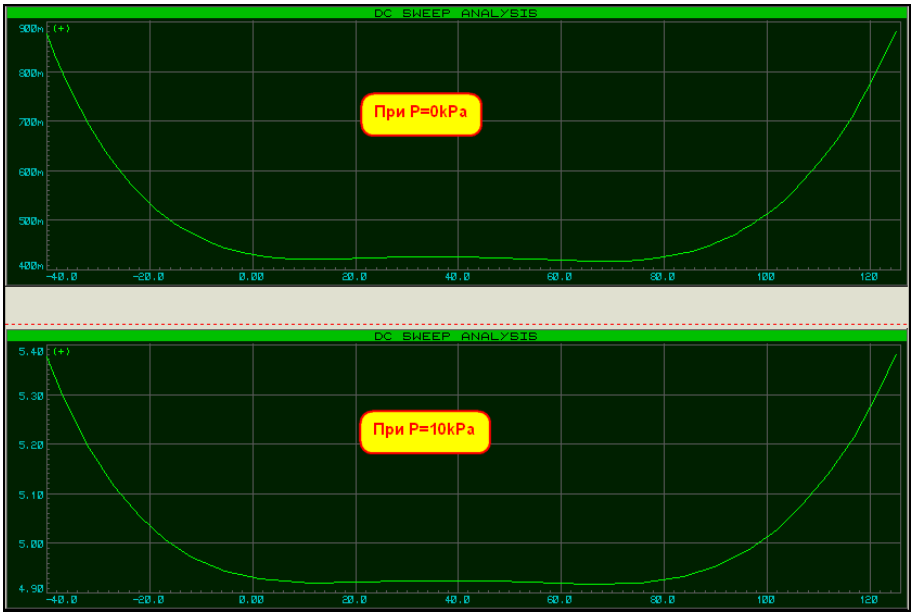


Рис. 69

Ну, по большому счету, можно сказать, что «многочлен» работает – налицо характерные завалы вверх ниже нуля и выше 80 градусов, которые есть и на графике в даташите. Даташиты на датчики **MPX4250** и **MPX5010** также присутствуют во вложении, так что можете убедиться самостоятельно. Я же на этом заканчиваю материал с моделированием на базе **SETPOINT** и перехожу к другим программным моделям на основе DLL.

К содержанию

8.6. LEDMPX.DLL – **основа всех активных «светящихся» цифровых индикаторов в** ISIS.

«Все-таки модель индикатора в протеусе штука глючная.»

shamber (30.01.2011)

Наконец-то я добрался до подробной разборки с этой библиотекой, чтобы раз и навсегда положить конец утверждениям «чайников» о глючности сегментных индикаторов в ISIS. Видимо информации в первой части FAQ по динамической индикации оказалось недостаточно. Будем «жевать» дальше.

Итак, из самого названия библиотеки можно вынести уже достаточно информации. **LED** говорит нам о том, что предназначена она для создания всевозможных светодиодных индикаторов, а **MPX** – принятое «за бугром» сокращение слова **multiplexing**. Вот именно на этот последний термин почему то многие откровенно «наплевали и забыли», а зря. Приведу дословно одно из толкований, встречающихся в Интернете: *«Процесс объединения отдельных потоков или каналов в один логический поток данных таким образом, что они позднее могут быть восстановлены в прежнем виде без ошибок. Двумя наиболее широко используемыми технологиями мультиплексирования являются частотное разделение каналов, когда для передачи различных сигналов используются несущие с различной частотой, и временное разделение каналов, когда отдельные сигналы передаются в свои временные промежутки».* Пожалуй, такая трактовка наиболее полно отражает то, что уже в самой библиотеке заложено мультиплексирование, т.е. разделенная во времени обработка сигналов. И об этом забывать никак нельзя, особенно когда вы

на эту динамику накладываете еще и свою, пытаясь симулировать динамическую индикацию со СВОИМИ временными параметрами, подчас, далеко не идеальными. Вот и попробуйте мысленно представить себе – что должно получиться в итоге, когда вы наложите друг на друга:

- Ваши собственные импульсные последовательности сканирования катодов индикаторов (раз) и анодов индикаторов (два);
- Времянки самой **LEDMPX** – по умолчанию **Minimum Trigger Time** установлено, как 1мсек (три)
- Параметры мультипликации самого симулятора на экране – по умолчанию в **System => Set Animation Options** частота смены картинок **Frames per Second** равна 20 (четыре).

Для того чтобы эти четыре составляющие дали адекватный результат нужно очень внимательно соотнести их параметры и универсального рецепта тут нет и быть не может. Всегда требуется индивидуальный подход, поскольку один применяет динамическую индикацию с частотой 25 Герц, другой – 40, а третий может вообще задрать до 200. Поэтому, только при грамотной установке временных параметров симулятора и самой модели на основе **LEDMPX.DLL** в соответствие с выходными параметрами симулируемой динамической индикации вы увидите именно ту картинку, которая должна быть в реальности. Ну и конечно не стоит забывать о гашении при переключении разрядов индикатора, о котором шла речь в первой части FAQ. На этом я заканчиваю «лирическое отступление» и перехожу непосредственно к самой **LEDMPX.DLL** и моделям на ее основе.

Определить, что модель индикатора основана на **LEDMPX.DLL** совсем не сложно, для этого достаточно обратить внимание на правый верхний угол окна **Pick Devices** (Рис. 70). Кроме того, у многих индикаторов в названии встречается само сочетание MPX, например, **7SEG-MPX4-CA**.

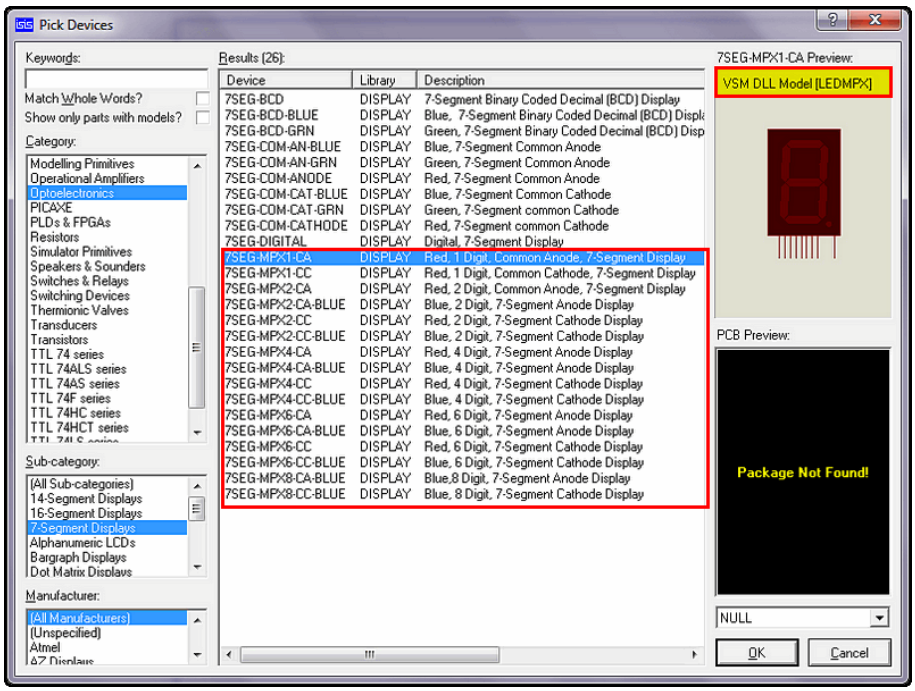


Рис. 70

Все модели на основе этой DLL расположены в категории **Optoelectronics**, а более конкретно в четырех ее подкатегориях: **14-Segment Displays**, **16-Segment Displays**, **7-Segment Displays**, **Dot Matrix Displays**. На рисунке 71 приведены наиболее характерные представители этих категорий, причем во включенном режиме. Я бы хотел здесь еще раз напомнить, чтобы вы не путали одиночные семисегментные индикаторы на основе **LEDMPX** со Schematic, которые мы рассматривали ранее – ведут эти модели себя совершенно по-разному. Характерная особенность одиночных индикаторов на основе **LEDMPX**, который их сразу выдает визуально, – уменьшенный до 50th шаг выводов. Эти модели и 14-ти 16-ти сегментные индикаторы появились только в последних версиях Протеуса, если не изменяет память, то с версии 7.4. Тогда же кардинально была изменена и сама **LEDMPX.DLL**, хотя **Help** Протеуса об этом скромно умалчивает вплоть до версии 7.8, видимо Лабцентр придерживает эту «фишку» на будущее.

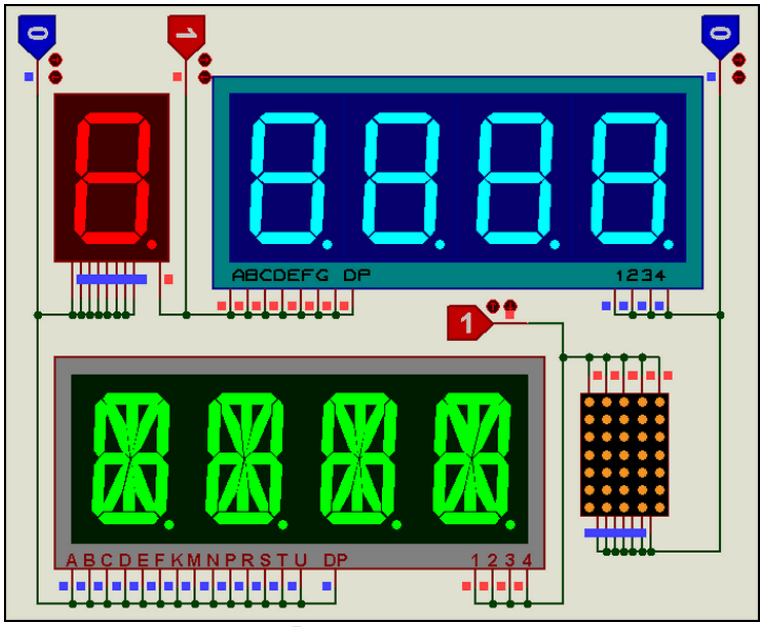


Рис. 71

До хелпа по мультиплексированным LED дисплеям можно добраться как через:
ПУСК => Все программы => Proteus 7 …. => Proteus VSM Model Help => LED and LCD Display Models,
так и непосредственно из окна свойств любой модели вышеупомянутых индикаторов, нажав кнопку Help справа. Раздел, посвященный этим моделям, называется **MULTIPLEXED LED DISPLAY**. Я не стану здесь дословно переводить весь этот раздел, но характерные особенности в моей трактовке изложу ниже.
LEDMPX.DLL может использоваться для моделирования произвольной светодиодной матрицы состоящей из знакомест (цифр или строк матричного индикатора) и сегментов (столбцов матрицы). Характерными приложениями для данной библиотеки является моделирование многоразрядных сегментных индикаторов и точечных светодиодных матриц. При этом обеспечивается двумерный массив строки (или знакоместа сегментного индикатора) обозначаются цифрами – **1, 2, 3** и т.д. а столбцы (или отдельные сегменты цифры) индикатора обозначаются буквами латинского алфавита – **A, B, C, D** и т.д. В оригинальном хелпе указано, что максимальное количество строк и столбцов может быть до 8, т.е. максимум можно было смоделировать 64 светящихся точки (сегмента). Так и было в действительности до тех пор, пока **LEDMPX.DLL** не была расширена под 14-ти и 16-ти сегментные индикаторы. Библиотеку расширили, а хелп оставили нетронутым. Но взгляните на 14-ти сегментный индикатор на рисунке 71 – он явно противоречит хелповской догме. Буковок, то бишь, столбцов матрицы (сегментов) – 14, а никак не 8, да еще и точка **DP**. Подозреваю, что на данный момент матрица имеет ограничения как минимум 16x16 строк/столбцов, проверять на максимум пока лень, дождемся пока Лабцентр «разродится» свежим хелпом.
Второй важной особенностью моделей на основе **LEDMPX.DLL** является то, что модели обладают чисто цифровыми свойствами. И об этом разработчики честно сообщили в help:

> The LEDs are modelled at the digital level only; there is no attempt to simulate their analogue behaviour (i.e. diode characteristics).

Но, первая из бед России в данном случае превалирует над дорогами, мы и по-русски то с ошибками, где уж дорасти до заглядывания в аглицкие хелпы. Вот и плодятся на форуме утверждения, подобные «эпиграфу» данного параграфа.
Ну, что-ж, давайте еще раз разжевывать эту жвачку. Поскольку модели на основе **LEDMPX.DLL** не обладают аналоговыми свойствами, они по определению цифровых моделей обладают бесконечно большим сопротивлением и не потребляют ток от источников сигнала. Проверяем данный постулат. Взгляните на рисунок 72. Тут как в избитой телерекламе, «трюк самостоятельно не повторять, опасно для жизни». Какой реальный индикатор такое издевательство выдержит? А виртуальный работает, и даже дым от экрана дисплея не идет. Обратите также еще раз внимание на нулевые показания амперметра, сейчас мы и это «разжуем и проглотим».

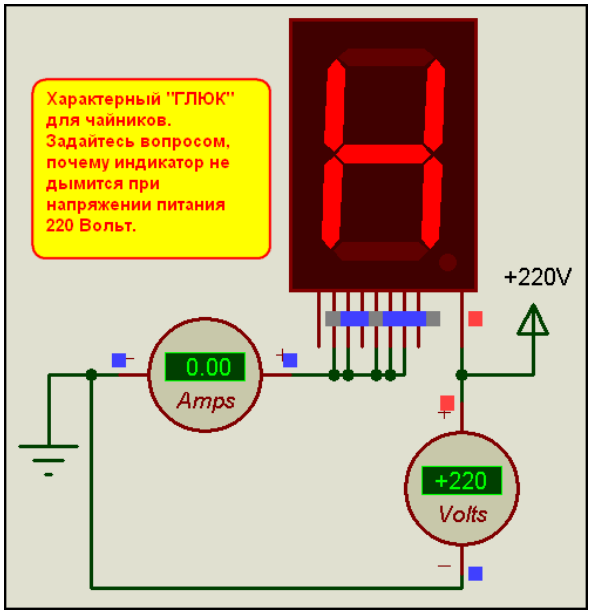


Рис. 72

Опять-таки характерной ошибкой пользователей «с дудочкой» является попытка управлять **ЦИФРОВОЙ** моделью индикатора с помощью аналоговых элементов, например, транзисторных ключей. Ну, в схеме, которую взяли из книжки, журнала, Интернета так нарисовано, значит должно работать. В жизни да, а вот в симуляторе…. Обратимся к следующему примеру (Рис. 73).

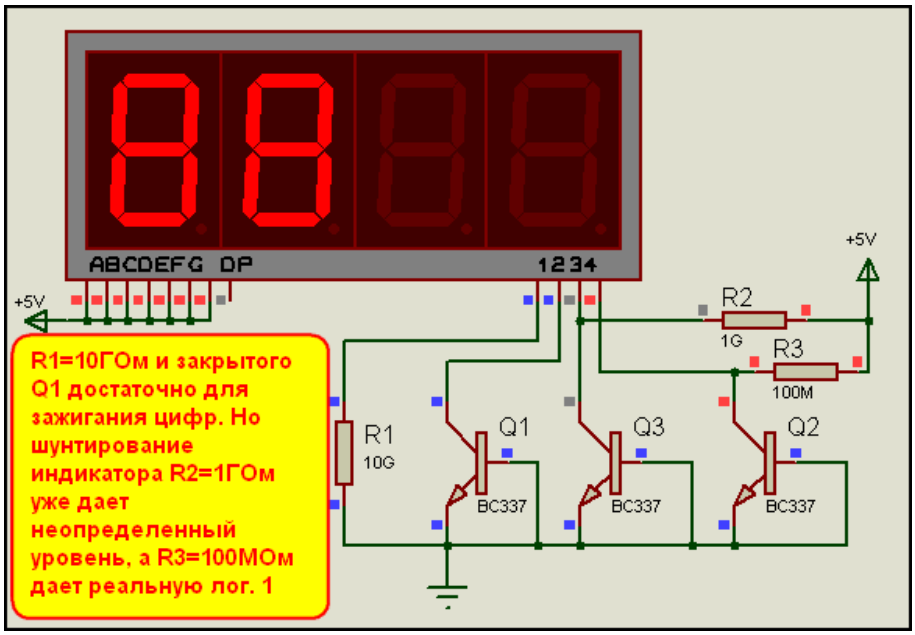


Рис. 73

Здесь у индикатора с общим катодом все четыре цифры включены по-разному. Мы видим, что даже резистора **R1** сопротивлением 10гигаОм в цепи катода достаточно, чтобы спровоцировать зажигание сегментов цифрового индикатора (левая цифра), что уж говорить о закрытом транзисторе **Q1** с заземленной базой (вторая слева цифра). Но, стоит зашунтировать цифру сопротивлением **R2**, пусть и очень большим в 1гигаОм, но аналоговым элементом (третья цифра) и на ее ножке и коллекторе **Q3** уже появился неопределенный уровень, а сопротивление в 100мегаОм (четвертая цифра) уже дает вполне реальную логическую единицу на коллекторе **Q2**. Именно этих цифровых парадоксов модели и не учитывают начинающие пользователи. Причем, в отговорки иногда пускается и такое – у меня нет аналоговых элементов, я питаю через ключ, например, столь у нас популярный **ULN2003**. Да модель то этого ключа в Протеусе схематичная и с аналоговыми свойствами. По-другому и быть не может, вы загляните в даташит этого ключа (Рис. 74) – все те же транзисторы на выходе. Поэтому подходы к симуляции здесь могут быть разными. Если вы собираетесь трассировать печатную плату в **ARES**, то надо делать два проекта – один для создания печатной платы с полным набором схемотехники, а второй для отладки – в нем «все лишнее за борт». Если же Вам необходимо только проверить проект в симуляторе, а печатную

плату вы будете делать в чем то более продвинутом, чем **ARES**, то можно сразу пойти по упрощенному пути. В частности те же транзисторные ключи на рисунке выше можно заменить обычными цифровыми инверторами, тогда и шунтирующие резисторы не нужны, и работать все будет как надо. Ну а если все-таки надо имитировать многоразрядный индикатор и аналоговые ключи, то придется идти на такие вот навороты, как на рисунке 73. Хотя, уж если подходить к этому вопросу корректно, то надо параллельно сегментам вешать не резистор, а диод и резистор, чтобы обеспечить проводимость в одном направлении – это будет ближе к реальности.

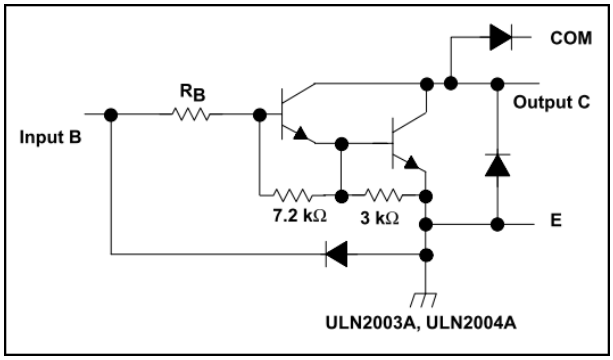


Рис. 74

Итак, надеюсь, больше на том, что все рассматриваемые далее индикаторы являются исключительно цифровыми моделями, останавливаться не надо.
Я же хочу заострить ваше внимание на последнем параграфе хелпа по **LEDMPX**, посвященном временным характеристикам и полярности сигналов, подаваемых на выводы индикатора для зажигания того или иного сегмента. Здесь тоже есть небольшая неточность в хелпе, кочующая из версии в версию, вплоть до 7.8. Специально цитирую именно оттуда:

> By default, a segment is drawn as lit if the corresponding row and column pins are both high for more than 1us during a given animation frame (typically 50ms). This gives a crude but effective representation of persistence of vision; in the real world, the segments are driven with a heavy current for short period of time, and the eye averages out the brightness.

Ну и почти дословный, слегка «облагороженный», с точки зрения фразеологии мой перевод:
По умолчанию, сегмент принимается, как светящийся, если соответствующие выводы строки и столбца находятся в состоянии логической единицы более чем 1 микросекунда в течение данного фрейма мультипликации (типовое значение 50мсек). Это позволяет грубо, но эффективно представить элемент, как светящийся, в реальности же сегменты управляются короткими сильноточными (от слова ток) импульсами, а глаз воспринимает среднюю яркость свечения.
Это как раз то, о чем я толковал в начале данного параграфа, за исключением одного маленького казуса. Если вы заглянете в свойства любого индикатора, то по умолчанию **Minimum Trigger Time** там равно **1ms**, а не **1us**, как написано в help. Это существенная разница, и зачастую простое уменьшение значения к рекомендуемому в help может значительно поправить вид индикации.
Теперь о полярности сигналов. Я вовсе не ошибся в переводе, и в самом Help так же сказано. Сегмент должен светиться при высоких уровнях на выводах соответствующих строки и столбца. Это если в свойствах модели не проинвертированы значения сигналов на одной из групп – столбцы или строки. А они обязательно проинвертированы, именно этим и достигается имитация индикаторов с общим катодом или с общим анодом и больше ничем. Заглянем для примера в окно свойств любого многоразрядного индикатора с общим анодом и поставим там галочку **Edit all properties as text** (Рис. 75).

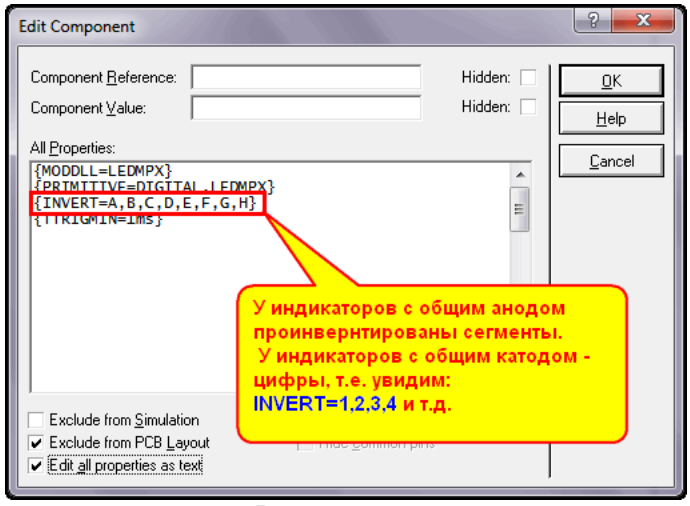


Рис. 75

Ну вот, именно об этом и идет речь в последнем абзаце Help. Мы же для себя делаем очень важную «зарубку» в памяти – для индикаторов на основе **LEDMPX.DLL** должны быть проинвертированы в свойствах состояния тех выводов (пинов), со стороны которых предусматривается управление нулевым логическим уровнем. Т.е. для индикаторов с общим анодом должны быть проинвертированы состояния сегментов:

INVERT=A,B,C,D,E,F,G,H

а для индикаторов с общим катодом состояния цифр (знакомест):

INVERT=1,2,3,4 и далее по количеству цифр.

Ну и заключительные комментарии из Help для **LEDMPX.DLL**, касающиеся непосредственно активной графики, к созданию которой мы перейдем в следующем параграфе. Индикаторы на основе данной библиотеки являются бит-зависимыми (с этим термином мы уже встречались при создании семисегментного **Schematic** индикатора). Общий символ (опять оттуда же, символ с индексом **_C**) определяет ширину знакоместа. Более понятно это станет при создании собственного индикатора, чем мы далее и займемся.

К содержанию

8.7. Активная графика сегментных индикаторов на основе LEDMPX.DLL. Трехразрядный индикатор из четырехразрядного.

Чтобы лучше понять, как строится графика индикаторов на основе **LEDMPX.DLL**, займемся непосредственно практикой. В качестве подопытного «символа года» воспользуемся красным четырехразрядным индикатором **7SEG-MPX4-CA**. Добываем его из библиотеки, помещаем в поле проекта и производим над ним варварское действие с помощью молотка, т.е. **Decompose** этот девайс. Теперь, если переключить левый тулбар в режим **Simbol Mode** (кнопка S), то в селекторе мы увидим полный набор символов из которых состоит наш индикатор (Рис. 76).

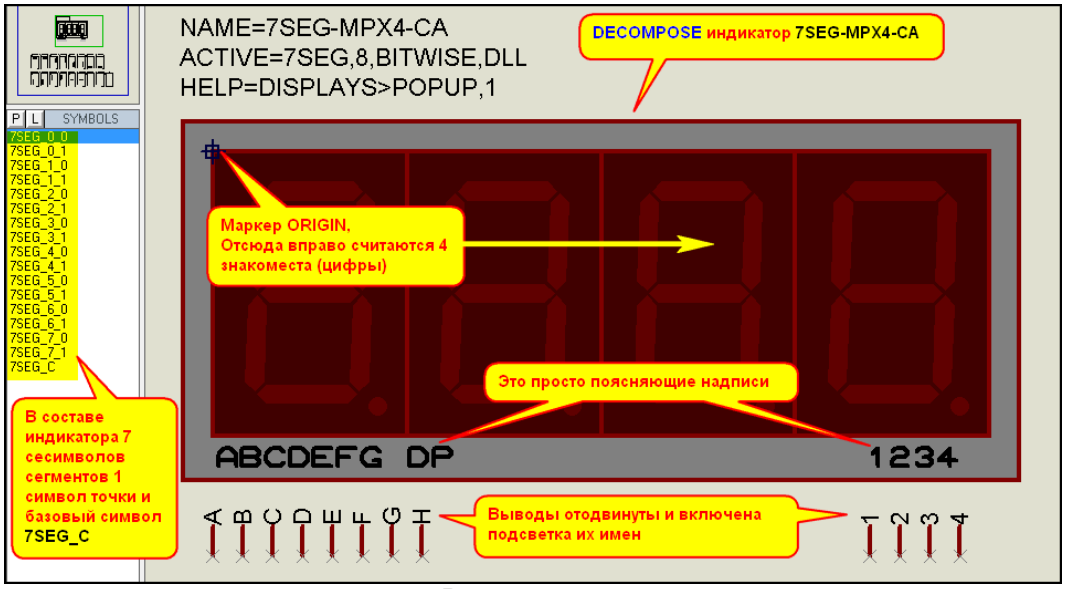


Рис. 76

Ничего нового и интересного вы тут не найдете, все это мы уже проходили в **п.7.2** нашего FAQ. Те же символы **7SEG**, погашенные с индексом **_0** на конце, зажженые – с символом **_1**. Всего в составе индикатора 7 символов для сегментов, начиная с нулевого, и символ десятичной точки – **7SEG_7**. Имеется и базовый символ (подложка) с индексом **_C**. На рисунке 77 я разобрал все эти символы с помощью Decompose, а также нанес для каждого синим пунктиром границы подложки, чтобы вы имели представление – как они построены. И тут для нас пока ничего нового нет.

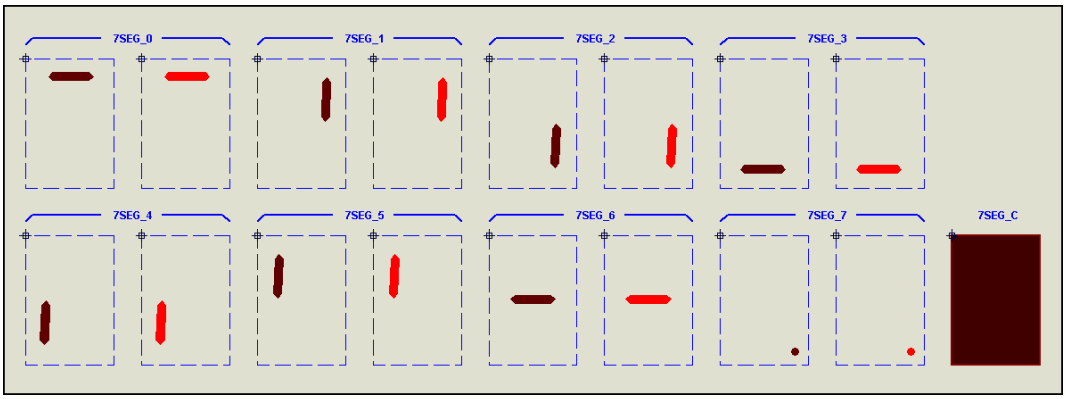


Рис. 77

Но вернемся все же к предыдущему рисунку 76 и обратим внимание на положение маркера ORIGIN – в левом верхнем углу самого левого знакоместа. Этот момент очень важен! Отсюда начинается отсчет знакомест в правую сторону. Ширина каждого знакоместа равна ширине символа подложки (**Common**) с индексом **_C**. Сколько таких «подложек» мы расставим вправо, столько и будет знакомест (столбцов) в нашем индикаторе и столько же должно быть выводов для их сканирования – выводов с именами **1, 2, 3** и т.д. Я на этом рисунке специально отодвинул выводы от тела компонента и включил подсветку их имен. Обратите также внимание, что черные надписи, обозначающие имена выводов остались при этом на месте, потому что это просто текстовые надписи и ничего более, а в модели их применили чисто в справочных целях, чтобы пользователь не заблудился – где какой вывод. Именно поэтому они нечетко совпадают по позициям с соответствующими выводами, а выводу **H** соответствует в надписи обозначение **DP** (от английского Decimal Point – десятичная точка). Это первый этап нашего эксперимента и я приложил его в папке **Step1** вложения.

Ну а теперь перейдем ко второму этапу и займемся чудесным превращением индикатора в трехразрядный. Делается это очень просто и не отнимет у нас много времени. Выполняем поэтапно с иллюстрациями. Для начала удаляем графическое изображение подложки самого правого, т.е. четвертого знакоместа (Рис. 78). Обратите внимание, что я не зря употребил термин «графическое изображение». Внутри графики компонента используются не символы, которые сами по себе содержат внутри маркер **ORIGIN**, а именно простые графические изображения – в данном случае закрашенный прямоугольник определенной ширины. Это я к тому, что если вы собрались наоборот добавлять знакоместа (столбцы), то надо либо скопировать соседнее вместе с графическими изображениями сегментов, либо вытащить на свободное место символ подложки и предварительно его разбить (**Decompose**), чтобы убрать оттуда маркер **ORIGIN**.

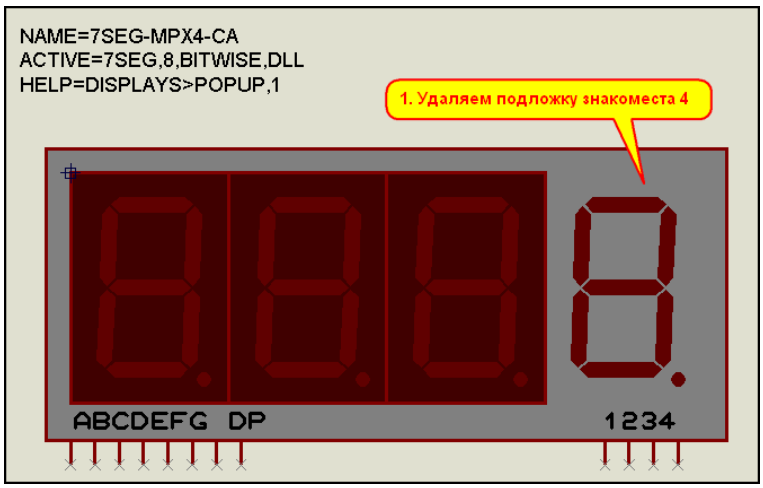


Рис. 78

Теперь выделяем и удаляем графические изображения сегментов и точки того же знакоместа (Рис. 79). Сейчас это сделать проще с помощью «лассо», т.е. обводим их с зажатой левой кнопкой мышки и потом жмем клавишу **Delete**. Теперь уже нет риска зацепить выделением что-нибудь нужное, они достаточно отдалены от соседнего знакоместа.

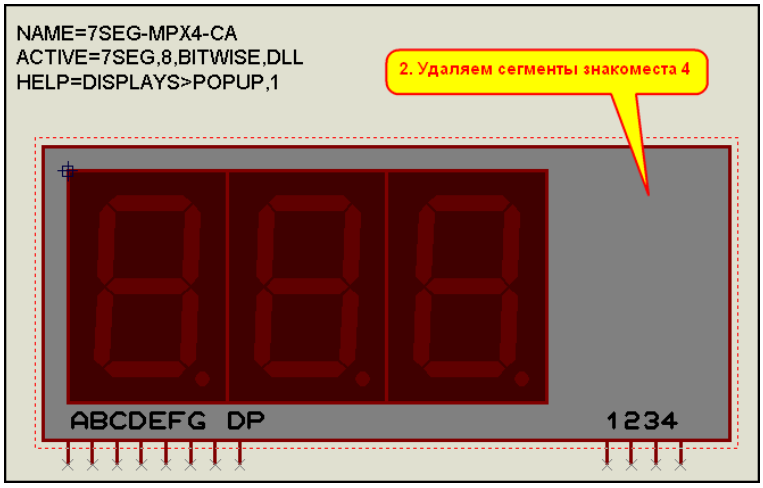


Рис. 79

Далее нам предстоит сдвинуть нужные нам выводы с именами 1, 2, и 3 левее, чтобы они оказались под третьей цифрой. Туда же сдвигаем и надпись «**1 2 3 4**» и редактируем ее, убрав ненужную цифру 4. Эта операция представлена на рисунке 80.

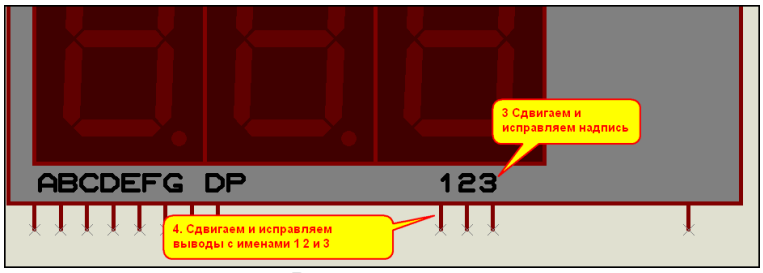


Рис. 80

Теперь нам осталось сдвинуть влево правую границу серого прямоугольника, обрамляющего весь наш индикатор и удалить оставшийся «за бортом» вывод с именем 4 (Рис.81).

Рис. 81

Все, коррекция графики на этом закончена. Выделяем с помощью лассо всю нашу модель, включая скрипт, и нажимаем любимую кнопку **Make Device** (Рис. 82).

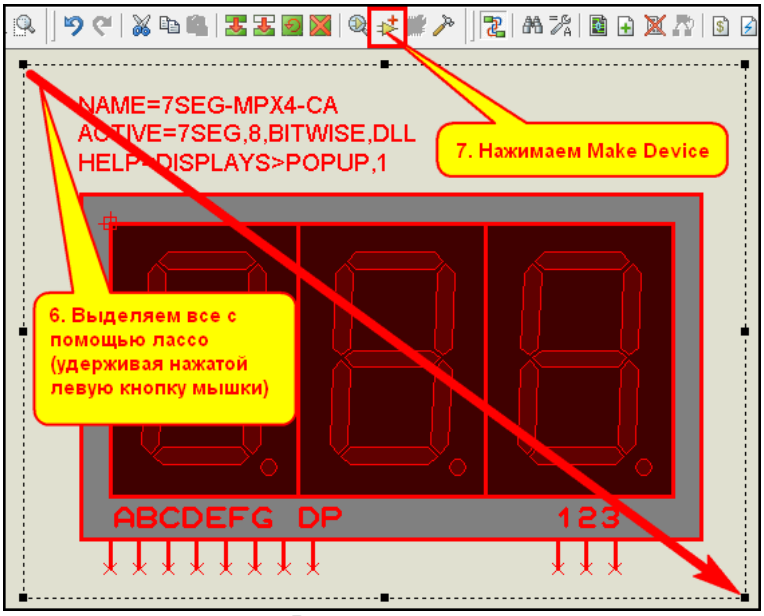


Рис. 82

Если мы не забыли зацепить скрипт, то на первой вкладке процедуры **Make Device** нам достаточно только поправить имя нашего девайса, т.е. изменить 4-ку на 3-ку (Рис. 83). Поскольку мы не меняли количество и имена символов, и они присутствуют в селекторе, то в поле **Active Component Properties** все остается без изменений. Достаточно только убедиться, что включены флажки **Bitwise States** и **Link to DLL**.

Немного остановлюсь на третьей вкладке. Здесь тоже все остается без изменений, но на будущее, если вам надо создать такой же индикатор, но с общим катодом, то достаточно еще раз запустить процедуру **Make Device** для того индикатора, который мы сейчас сделаем. На первой вкладке изменить в имени **CA** на **CC** (от английского Common Catode), а здесь для свойства **INVERT** указать не сегменты, а знакоместа, т.е. в поле **Default Value** для трехразрядного индикатора вписать **1,2,3** (Рис. 84).

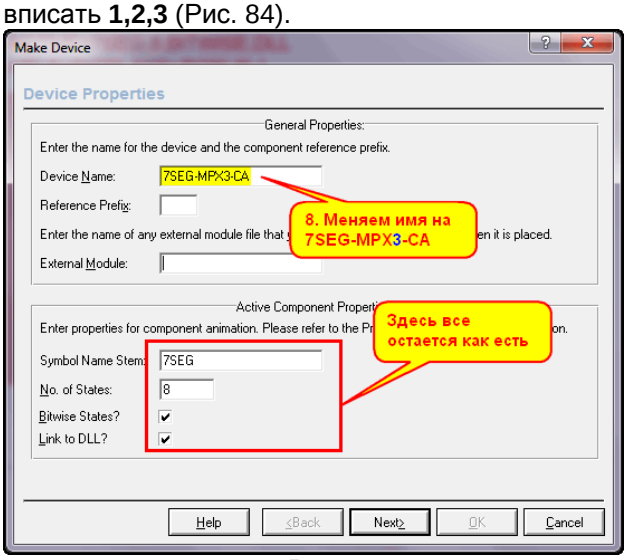


Рис. 83

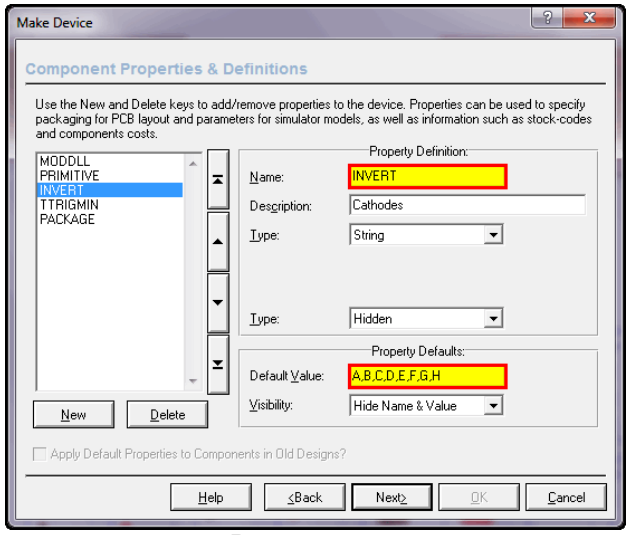


Рис. 84

Ну и теперь доходим до последней вкладки, приводим в **Description** в соответствие количесво цифр нашего индикатора и сохраняем (Рис.85). У меня для сохранения сейчас открыта только **USRDVC**, но можете заранее открыть для записи в менеджере библиотек ту, в которой хранятся семисегментники и сразу сохранить наш девайс там. Библиотека называется **Display** и на сегодняшний день там свободно 13 позиций.

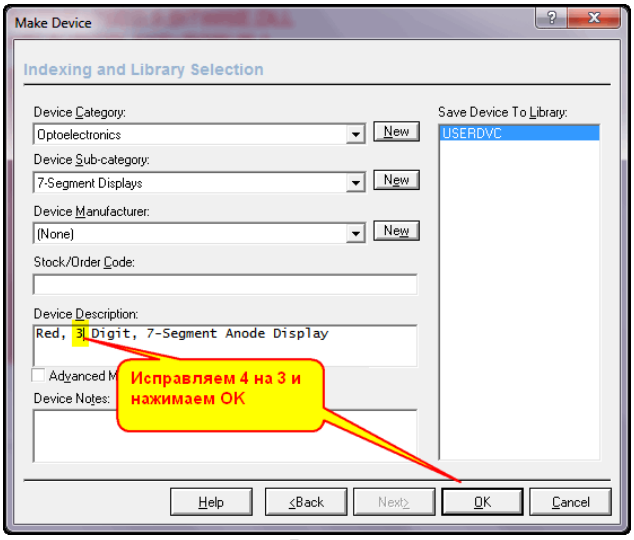


Рис. 85

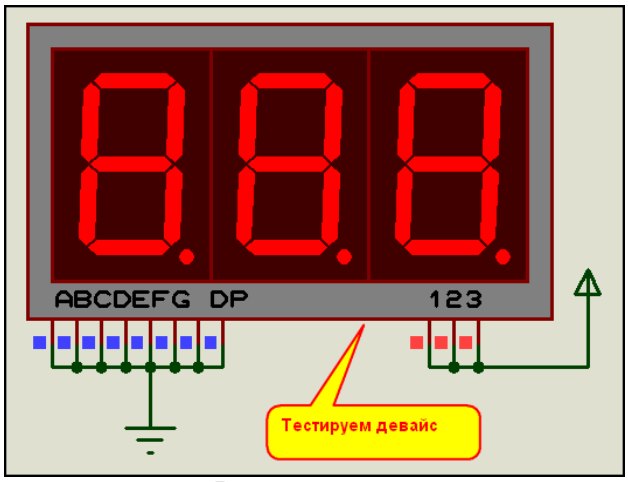


Рис. 86

Нам осталось запустить тестирование нашего устройства и убедиться, что оно работает (Рис. 86). Второй этап нашего эксперимента на этом завершен. Он находится в папке **Step2** вложения. Для того, чтобы сохранить трехразрядые индикаторы в своих библиотеках достаточно пройти процедуру Make Device ничего не меняя для уже готовых индикаторов с общим анодом и общим катодом из этого проекта. Как видите, здесь не требуется даже использовать дополнительные файлы моделей, вся необходимая информация уже содержится в свойствах самой графической модели. Аналогичным образом можете создать самостоятельно индикаторы с другим цветом свечения и с другим количеством разрядов. Необходимо только помнить, что для **LEDMPX** знакоместа располагаются в один ряд по горизонтали слева направо от маркера **ORIGIN** и постараться не превысить общее допустимое количество строк (сегментов) и столбцов (знакомест) матрицы **LEDMPX.DLL**. Из личного опыта сразу могу предупредить, что если допустимое количество будет превышено, то в вашем девайсе будут светиться одновременно не один сегмент (точка), а сразу два. Далее мы поупражняемся с точечными матрицами на основе LEDMPX и на этом закончим материал по этой DLL.

К содержанию

8.8. Активная графика точечных матриц на основе LEDMPX.DLL. Идем на рекорд – матрица 16x16.

В исходном варианте библиотеки **Optoelectronics** в ISIS точечные светодиодные матрицы (**Dot Matrix Displays**) представлены двумя наиболее распространенными видами: 5x7 и 8x8 точек. Это матрицы синего, зеленого, красного и оранжевого цвета. Все эти матрицы построены на основе **LEDMPX.DLL.**

Интерес разработчиков к использованию точечных светодиодных матриц в последнее время значительно вырос. Они широко используются как в рекламных проектах, так и для создания всевозможных индикаторов типа бегущей строки. Наряду с уже давно заполонившими наш рынок матрицами фирмы **Kingbright**, появились в продаже матрицы китайских **Betlux**, **Ningbo Foryard Optoelectronics**. Выпускает матрицы и компания **Rohm**. Причем в продукции этих фирм можно встретить матрицы форматов 5x8, 5x9, 6x8, 16x16 элементов, которые отсутствуют в библиотеках ISIS. Сразу сделаю оговорку для нетерпеливых – построить двухцветную, а уж тем более RGB-матрицу на основе **LEDMPX.DLL** не выйдет. Вспомните сам принцип работы **LEDMPX** – точка (сегмент) может иметь только два состояния (**_0** или **_1** в конце обозначения соответствующего символа) в данный момент времени, определяемый знакоместом (цифрой). Но, тем не менее, облегчить себе жизнь при разработке проектов с использованием одноцветных матриц мы попробуем.

Итак, в качестве примера создадим зеленую матрицу формата 16x16. Заодно и проверим, насколько расширились возможности **LEDMPX.DLL**. Берем зеленую матрицу 8x8 – модель **MATRIX-8X8-GREEN** и разбираем на запчасти с помощью **Decompose** (Рис. 87). Если перейти в режим отображения символов, то в селекторе мы обнаружим восемь символов точек с индексами с нулевого по седьмой и общий базовый символ с индексом **_C**.

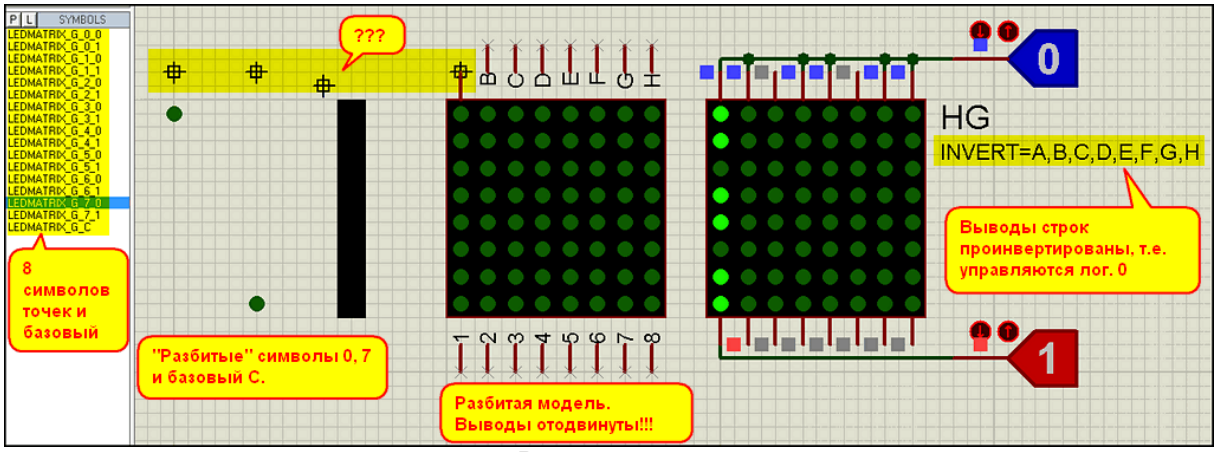


Рис. 87

Я пошел на небольшое ухищрение, чтобы не делать много скриншотов - совместил максимум в одном. Итак, на рисунке также представлены «декомпозированные» символы с индексами 0, 7 и базовый. И тут меня ожидал очень подозрительный сюрприз. Обратите внимание, что у символа **LEDMATRIX_G_C** после **Decompose** маркер **ORIGIN** сдвинут влево и ниже. Я оставил подсветку сетки с шагом **50th** на скриншоте, чтобы вы имели представление о том: куда и насколько. Честно говоря, мне непонятен этот ход разработчика модели, поэтому в своей я это применять не стану и размещу маркер как и обычно. Сам символ имеет **LEDMATRIX_G_C** форму прямоугольника для одного столбца, т.е. он как бы и определяет ширину знакоместа для одной колонки (в семисегментниках цифры). В середине рисунка 87 показана полная графика «разбитой» **MATRIX-8X8-GREEN**. Я опять отодвинул от тела выводы и подсветил их имена, за исключением вывода **A**, который оставлен на месте для лучшей ориентации. Ну и наконец справа на рисунке показана подключенная матрица 8x8. Мы видим, что сверху (со стороны сегментов-букв) для зажигания точки необходимо подать логический ноль, а снизу (со стороны знакомест-колонок) – логическую единицу. Это подверждается тем, что в свойствах матрицы прописано **INVERT=A,B,C,D,E,F,G,H**. Я убрал фигурные скобки «скрывающие» это свойство и оно видно на рисунке. Весь этот подготовительный этап в примере вложения папка **Step_1**.

Ну и теперь, зная особенности **LEDMPX**, сразу становится ясно – что нам предстоит сделать для того, чтобы получить модель формата 16x16 точек.

В первую очередь изменяем основную графическую модель, т.е. растягиваем черный квадрат подложку и расставляем на нем погашенные точки. Напомню еще раз, что здесь используются не символы из селектора с внедренным в них **ORIGIN**, а просто графические изображения соответсвующих элементов – квадрата, круга. Само-собой разумеется, что нам предстоит добавить еще по восемь выводов – сверху с именами букв латиницы – **I, J, K, L, M, N, O, P**, а снизу с именами в виде чисел – **9, 10, 11, 12, 13, 14, 15, 16** (Рис. 88).

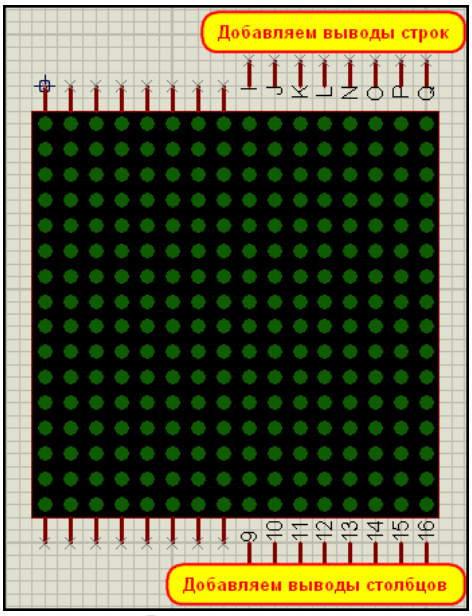


Рис. 88

Далее нам предстоит скомпилировать с помощью **Make Symbol** недостающие символы погашенных и светящихся точек в столбце и удлинить вниз прямоугольник базового символа. И вот здесь нас ожидает еще одна неприятная неожиданность Протеуса, о которой я «в пылу азарта» забыл упомянуть. Максимальное количество букв, цифр и спецсимволов подчеркивания в имени символа не должно превышать 15. Кстати, это же ограничение качается и **Device Name** на первой вкладке **Make Device**, так что возможно кто-то уже сталкивался с ним. Теперь посмотрим внимательно на исходные имена символов, начинающиеся с **LEDMATRIX_G**. Это уже 11 позиций, значит, нам под остальное осталось только 4. Делалась эта модель еще в ранних версиях Протеуса и тогда автор об этом не задумывался. Пока номера символов состояли из одной цифры, этого хватало, но, мы то пойдем дальше – будут и 10 и 11 и т.д. Ну что-ж, придется делать все символы сначала, укоротив имя. Я просто выбросил гласные буквы из слова MATRIX, и мои новые символы имеют обозначение **LEDMTRX_G**. Почему я не убрал вместо этого **_G** в конце имени? Да все из того же любопытства – не повлияет ли этот первый символ подчеркивания, находящийся в имени символа, на работу модели. Забегая вперед, скажу, что не повлиял. Следовательно, допустимо использовать в имени символа знак подчеркивания.

Но, продолжим наш титанический труд. Формирование новых символов, да и вообще работу с графикой активных индикаторов лучше делать с шагом сетки **50th**. И править их лучше в непосредственной близости от готового графического изображения всего компонента, чтобы не потерять ориентацию. Для начала удлиним базовый символ и скомпилируем его с новым именем **LEDMTRX_G_C** (Рис. 89). Попутно восстанавливаем «статус кво» маркера **ORIGIN** – я разместил его непосредственно над прямоугольником на том же расстоянии, что и в графическом изображении всей модели.

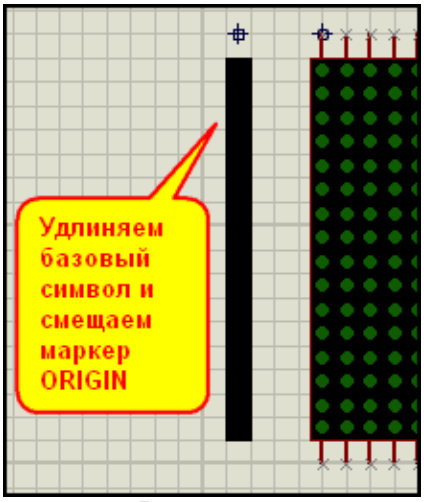


Рис. 89

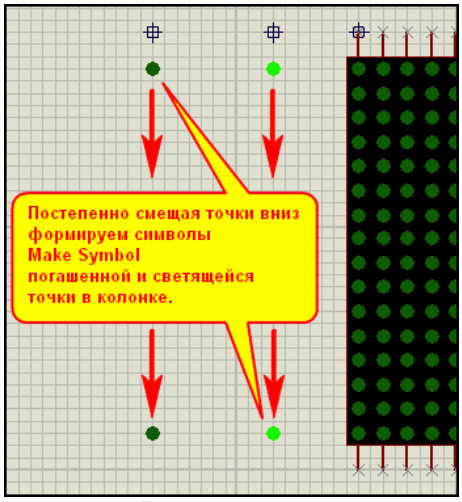


Рис. 90

Затем начинаем формирование погашенных и светящихся точек матрицы. Поступаем также, ка и с базовым символом помещаем их поблизости от изображения всего компонента на уровне первой строки. **Make Symbol** из левой погашенной точки и маркера **ORIGIN** над ней символ **LEDMTRX_0_0**, а из светящейся точки и ее маркера **ORIGIN** – символ **LEDMTRX_0_1**. Затем смещаем обе точки (только точки, без маркера) на две клеточки вниз – вторая строка матрицы, и создаем соответствующие символы **LEDMTRX_1**, опять смещаем вниз – создаем **LEDMTRX**_3 и так до символов последней строки матрицы **LEDMTRX**_15.

После этого нам осталось только **Make Device** нашу новую модель, но не забудьте, что мы поменяли имена символов и увеличили их количество. Поэтому на первой вкладке **Make Device** изменяем эти значения (Рис. 91). Если при создании модели используется скрипт от старой модели 8x8, то в имени девайса достаточно просто поправить имя, если же вы создаете устройство «с нуля», то необходимо ввести **Device Name** полностью. Не забудьте также и про флажки бит-зависимости и связи с DLL.

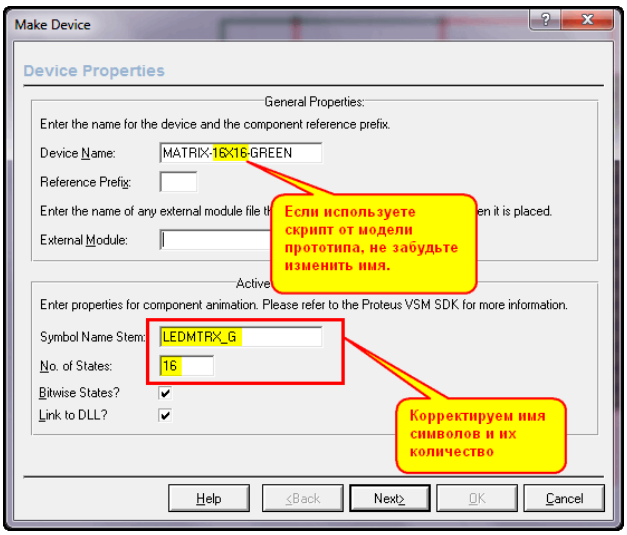


Рис. 91

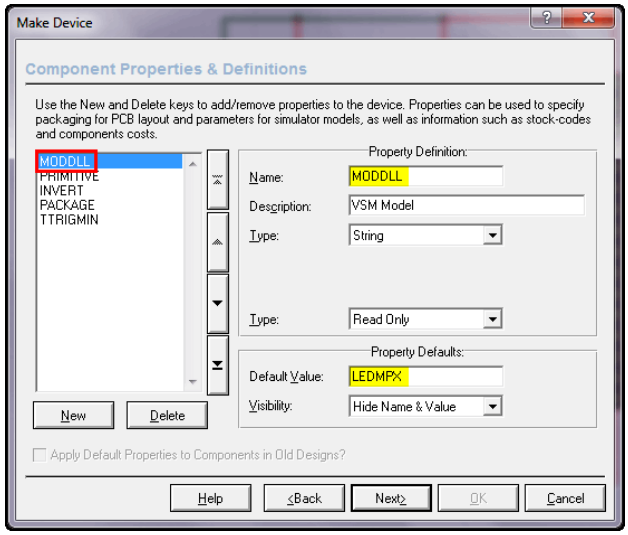


Рис. 92

Вторую вкладку назначения корпуса проходим транзитом, на третьей остановимся подробнее. Если использовался старый скрипт, то свойства будут взяты оттуда и здесь, кроме добавления имен сегментов (точек) в свойство **INVERT**, ничего менять не надо. Если же мы создаем устройство заново, то необходимо через кнопку **New** добавить и отредактировать следующие свойства:
MODDLL – В графе **Description** умолчание или по своему усмотрению, первый **Type** – **String**, второй **Type** – **Read Only**, **Default Value** – **LEDMPX**, или полностью **LEDMPX.DLL** (Рис. 92);
PRIMITIVE – В графе **Description** умолчание или по своему усмотрению, первый **Type** – **String**, второй **Type** – **Hidden**, **Default Value** – **DIGITAL,LEDMPX** – через запятую без пробелов (Рис. 93);
INVERT – В стандартных нет, добавляется через **Blank Item**. В графе **Description** умолчание или по своему усмотрению, первый **Type** – **String**, второй **Type** – **Hidden (**но тем, кто активно работает, с матрицами рекомендую поставить **Normal**, чтобы оперативно менять полярность выводов), **Default Value** – **A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P** – все через запятую без пробелов (Рис. 94);
TTRIGMIN – В стандартных нет, добавляется через **Blank Item**. В графе **Description** – **Minimum Trigger Time** (русскоязычные пользователи могут вписать – **Время переключения**), первый **Type** – **Float**, выбрать ограничения **Limits** – **Positive**, **Non-Zero**, второй **Type** – **Normal**, **Default Value** – **1ms** (Рис. 95).

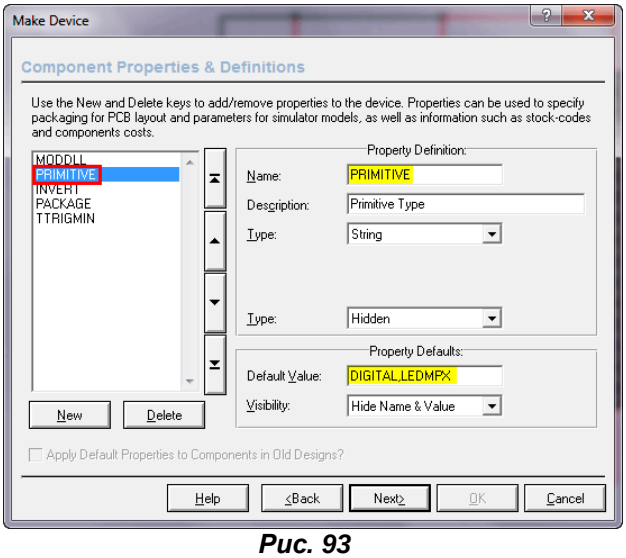


Рис. 93

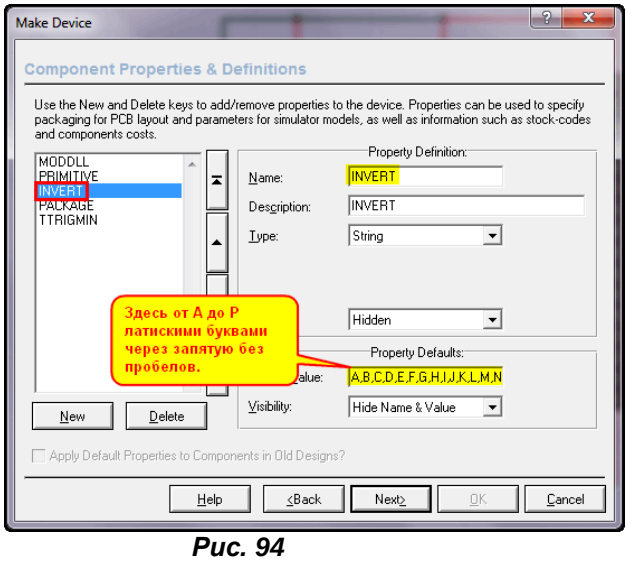


Рис. 94

Я ничего не сказал по **PACKAGE** – корпус, потому что пока мы его не добавляли и он по умолчанию примется **None** (отсутствует). Кроме того, для **TTRIGMIN** я указал значение, как и у остальных моделей – 1 миллисекунда, хотя сам же рекомендовал его уменьшать. Но это сделано только для того, чтобы не путаться при добавлении в проекты, а то у одной модели так, у другой по-другому.

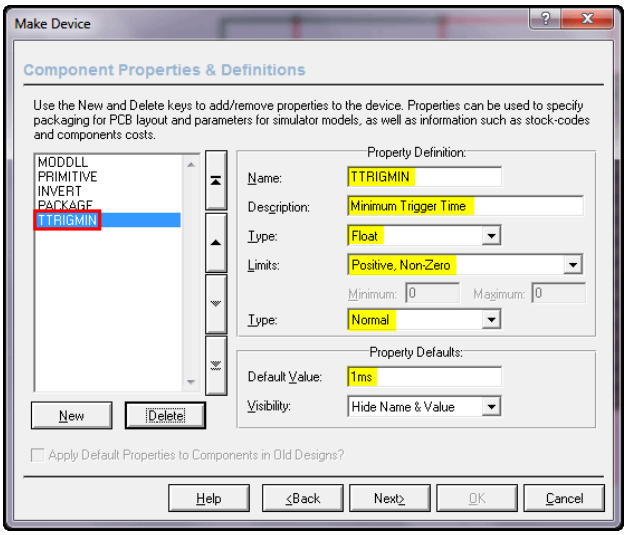


Рис. 95

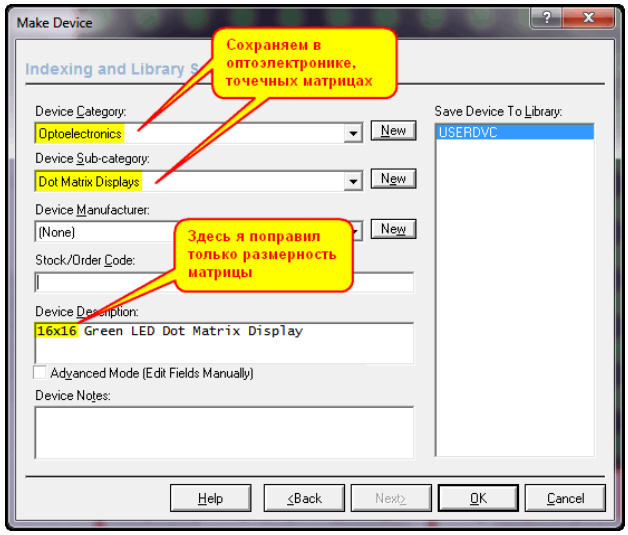


Рис. 96

Проходим нашу процедуру **Make Device** до конца и сохраняем матрицу в разделе оптоэлектроники, в подкатегории точечных матриц (Рис. 96). У меня для сохранения на данный момент открыта только **USRDVC**, но вы можете сохранить в своей библиотеке, предварительно создав ее в **Library Manager**. Напомню, что все остальные матрицы лежат в **DISPLAY**, но там только 13 свободных мест. Если собираетесь и далее создавать активные индикаторы, то лучше создать свою библиотеку, например, **DISPLAY2** на сотню «посадочных мест».

Нам осталось только проверить нашу матрицу в работе, что я и сделал (Рис. 97). Здесь я специально подсветил только угловые и центральные точки матрицы, чтобы убедиться, что нет повторных подсвечиваний точек и библиотека **LEDMPX.DLL** ведет себя адекватно с таким размерностью матрицы.

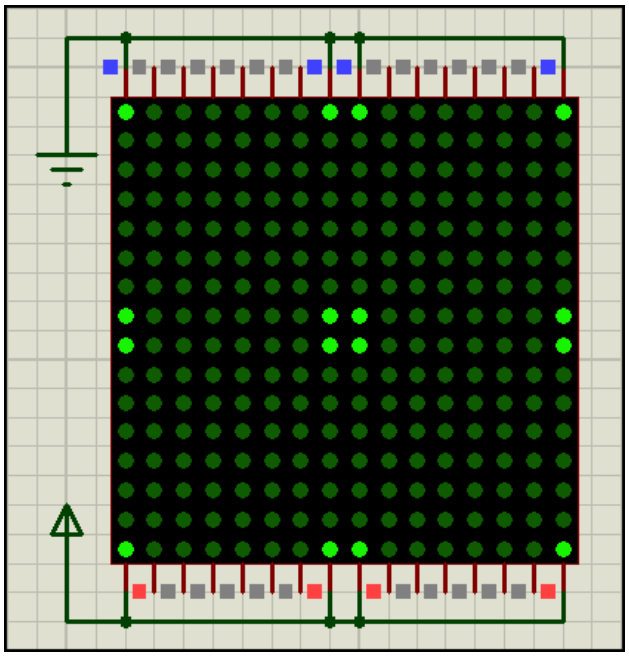


Рис. 97

На этом рассмотрение данной динамической библиотеки завершено, но не закончен материал по ее применению. Я вознамерился окончательно расставить точки над «и», да и в матрице тоже, с вопросами по динамической индикации в ISIS. Поэтому в следующем материале на примере данной матрицы мы этим и займемся. Первоначально хотел сделать это прямо здесь, но думаю, данный вопрос заслуживает отдельного параграфа.

К содержанию

8.9. О параметрах анимации и моделировании динамической индикации с помощью индикаторов на базе LEDMPX.DLL.

Я вновь возвращаюсь к теме отображения динамической индикации в ISIS, потому что теперь, когда мы знаем принцип работы многоразрядных индикаторов на основе **LEDMPX.DLL**, пора поставить огромный и жирную точку на этом вопросе. Для начала проведем несколько

экспериментов с моделью матрицы 16x16 из предыдущего раздела. Воспользуемся примитивами сдвиговых регистров **SHIFTREG_16** и соберем простейший «бегущий огонь» (Рис. 98).

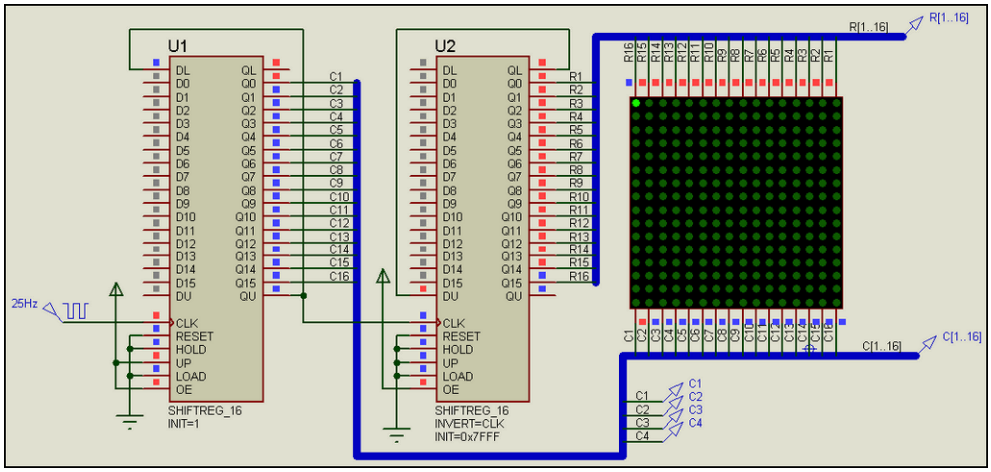


Рис. 98

В качестве тактового генератора используем **DCLOCK** из **Generator Mode**. При таком включении мы должны иметь светящуюся точку, скользящую слева направо по столбцам и постепенно переходящую сверху вниз по строкам (пример **Takt_25Hz.DSN** в папке **BAD_EXAMPLES\DOT_MATRIX** вложения).
Запустим симуляцию кнопкой **Step** или **Pause**, чтобы встать на нулевой отсчет времени. Этот момент зафиксирован на рисунке 98. В соответствии с прописанными в свойствах **INIT=1** для **U1** и **INIT=0x7FFF** для **U2** регистры приняли заданные значения, и загорелась верхняя левая точка индикатора. Выполним шаг кнопкой **Step**. Согласно **Single Step Time** (Рис. 99) наш симулятор продвинется на 50мс.

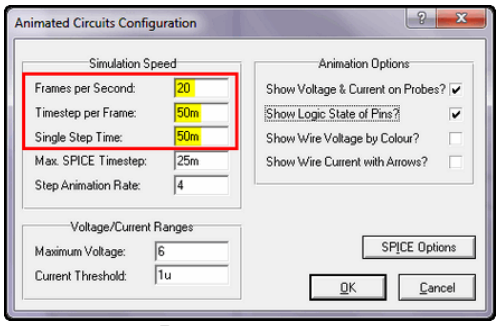


Рис. 99

При этом должна загореться вторая точка в строке, а первая погаснуть. Но тут нас ждет весьма неожиданный сюрприз (Рис. 100).

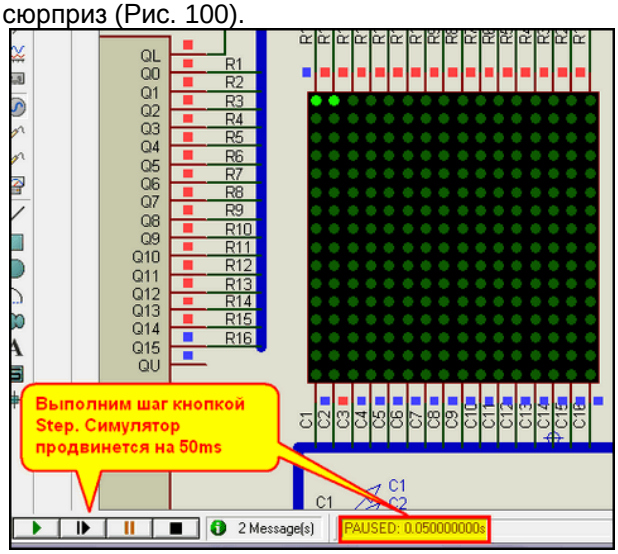


Рис. 100

По логическим уровням вроде все верно, низкий на строке **R1** и высокий на колонке **C2**, а горят сразу две точки вместо одной. Сколько бы мы далее не нажимали кнопку **Step**, время будет прибавляться по **50ms**, но светиться всегда будут две, а иногда даже три точки, т.е. мы имеем «размазанную» индикацию. Что же произошло – глюк Протеуса? Конечно глюк, но спровоцированный нами.

Для начала еще раз заглянем в меню **System => Set Animation Option** и вспомним, что означают и как соотнесены параметры, обведенные красной рамкой на Рис. 99.

Frames per Second (Кадры в секунду) – частота обновления окна программы при воспроизведении анимации (мультфильма, если так будет понятнее), запущенной кнопкой **Play**. Подчеркну, что этот параметр для окна основного поля **ISIS**, где расположена наша схема, а не всего экрана монитора. По умолчанию это значение равно **20Гц**, из чего несложно сделать расчет, что один фрейм соответствует **1000/20=50мс**. Данный параметр в **ISIS** можно менять в пределах от 10 до 50мс, дальше ограничения ISIS не позволят этого сделать.

Timestep per Frame – временной интервал анимации, приходящийся на один фрейм. Обратите внимание, что если мы разделим все те же **1000мс** (1секунда) на значение этого параметра по умолчанию **50мс**, то получим частоту **20Гц**, ту же, что и в предыдущем параметре.

Single Step Time – временной интервал анимации, приходящийся на один шаг, выполняемый по кнопке **Step**. По умолчанию он установлен равным предыдущему параметру, но при желании можно поставить и отличающееся значение.

Также хочу напомнить, что установленные параметры анимации **сохраняются в текущем проекте**, так что если вы сохранили проект и открыли новый, то там они снова встанут в значения по умолчанию, т.е. соответственно **20Гц**, **50мс** и **50мс**. Это соотношение 20 кадров в секунду и 50мс выбрано неспроста. Как гласит **ProSPICE Help**, оно обеспечивает наиболее гладкое зрительное восприятие симуляции в реальном времени в большинстве случаев и при этом не перегружает ЦП компьютера дополнительными расчетами активных элементов в кадре.

Теперь создадим цифровой график нашей индикации и попробуем разобраться по нему в причинах глюка. На этот график (Рис. 101) для наглядности я нанесу границы наших фреймов (кадров) через каждые 50мс красными вертикальными линиями. Кроме того, с помощью дополнительного генератора я имитировал время стробирования нашего индикатора **Trigger Time**, равное по умолчанию 1мс.

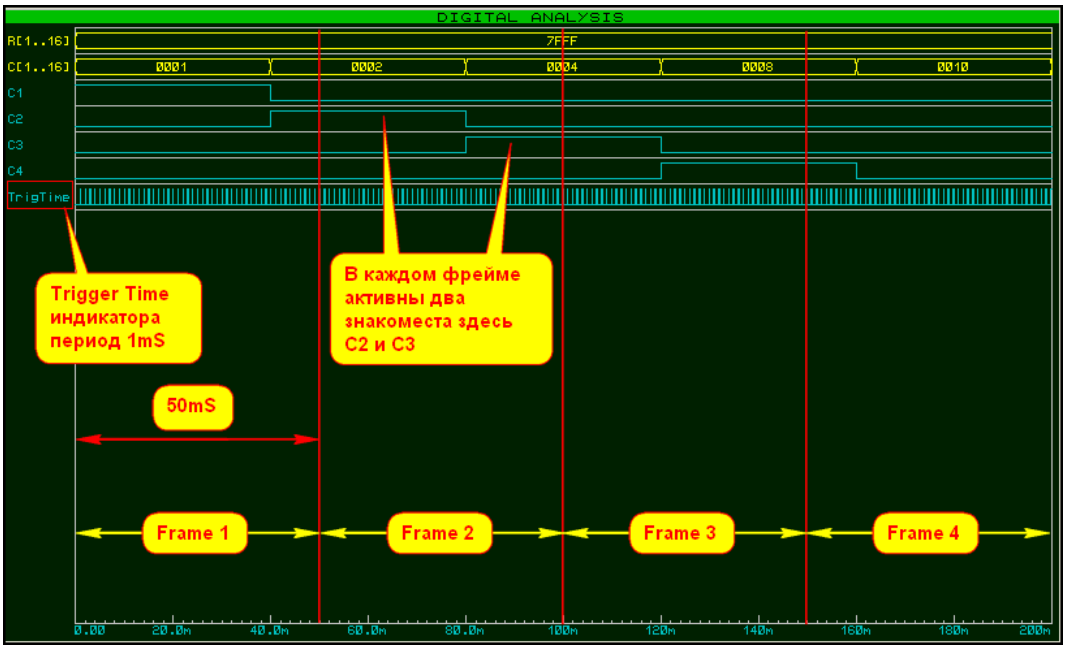


Рис. 101

Обратите внимание на первые три фрейма. В каждом фрейме мы имеем по два активных сигнала (лог. 1) знакомест-колонок **C1…C4**. Поскольку информация в строке не меняется, она имеет одно и то же значение для всех кадров , относящихся к первой строке. Модель **LEDMPX.DLL** стробирует активность элемента с периодичностью **1ms**, т.е. в кадре (фрейме) длительностью **50ms** опрос выполняется 50 раз, и, если встречается совпадение лог. 0 на строке матрицы и лог. 1 на знакоместе-колонке – зажигается соответствующая точка. У нас бы и при старте две точки загорелись, но мы стояли на временном отсчете 0, и поэтому **LEDMPX** просто еще не успела «отработать» на второе знакоместо **C2**.

Именно такая же чехарда с «наползанием» разрядов-знакомест друг на друга творится, когда мы запускаем проекты с динамической индикацией на сегментных индикаторах, если не

принять превентивных мер. Выражаясь словами телевизионщиков, мы имеем «сбитую кадровую синхронизацию». Если мы увеличим тактовую частоту задающего генератора (соответствующие примеры в папке **BAD_SAMPLES\DOT_MATRIX** вложения), то в кадр будут попадать уже не два разряда, а больше и в конечном итоге (пример с генератором 100 kHz) мы получим уже на втором шаге симуляции полностью светящуюся матрицу. Это одна из причин неадекватной динамической индикации в ISIS.

Прежде, чем мы пойдем дальше - полезный совет владельцам «медленных компьютеров», подтвержденный примерами в папке **BAD_SAMPLES\DOT_MATRIX** вложения.

СОВЕТ: *Обратите внимание на то, что часть примеров в папке **BAD_SAMPLES** имеет на конце индекс **Lite** – облегченные. Еще раз вернемся к рисунку 99, а конкретно к двум установленным флажкам в правой части свойств анимации. Первый из них – **Show Voltage & Current on Probes?** – отвечает за показ значений напряжений и токов у установленных в проекте пробников-зондов. Второй – **Show Logic State on Pins?** – отвечает за показ тех самых сине-красных квадратиков логического состояния выводов у цифровых моделей. Так как с некоторого времени мне приходится часть материала готовить на нетбуке с тактовой 1,6 ГГц и слабенькой видеокартой, то даже некоторые примеры с чисто цифровой симуляцией грузят бедного «нетбуку» на все 100%. Так вот, убрав эти флажки, я снижаю загрузку ЦП до приемлемого уровня. Это заметно и на мощных компьютерах, можете убедиться самостоятельно, запустив одноименные примеры из папки с индексом **Lite** и без такового. Примите на вооружение данный прием, хотя он наиболее эффективен только при большом количестве пробников и «многоногих» цифровых моделей в проекте.*

Итак, для того, чтобы привести рассмотренный выше пример в норму, возникает естественное желание покрутить «ручку частоты кадров», чтобы границы кадров совпали с фронтами смены знакомест-колонок **C1…C4**. В данном случае мы можем позволить себе такую роскошь. При заданной нами для входного генератора частоте 25Гц время свечения (длительность единичного импульса) на входах столбцов индикатора **Cx** составляет 40 мс. Зададим для **Timestep per Frame** и **Single Step Time** это значение. Чтобы сохранить соотношение между **Timestep per Frame** и **Frames per Second** пересчитаем последнее **1000/40=25** кадров в секунду. Для сохранения эффекта реального времени задаем **Frames per Second** найденное значение 25, пока мы еще не выходим за рамки заданных ограничений от 10 до 50. Запускаем симуляцию, но нужного эффекта не получаем, по-прежнему светятся по две точки. Можно и дальше изменять значения этих параметров, но эффекта это не даст. В конечном итоге мы упремся в верхнее ограничение 50 для параметра кадров в секунду, и дальнейшее снижение таймстепов приведет только к замедлению смены картинок анимации. Правда, справедливости ради надо отметить, что нагрузка на ЦП компьютера при этом начнет снижаться, так что в ряде случаев – это тоже полезно. Снизится загрузка и замедлится мультфильм если мы будем просто менять один из параметров, например, уменьшать время **Timestep per Frame**, не затрагивая при этом частоту кадров в секунду.

Кроме того, сейчас мы экспериментируем с одним светящимся объектом-точкой, а представьте, что у нас сегментный индикатор на несколько знакомест, где надо видеть одновременно все разряды. Это уменьшение **Timestep per Frame** до границ одного разряда даст нам мелькающие по очереди отдельные разряды и испортит целостность восприятия нашего «Ну, погоди». Вероятно, многие уже сталкивались с таким вариантом, а для тех, кто желает полюбоваться на примеры - вложение в папке **BAD_EXAMPLES\MC** соответственно для AVR с кодом на Си в CodeVision и для PIC с кодом в CCS PICC. Конечно же, для обеспечения стабильного периода смена разрядов индикации загнана в прерывание по переполнению таймера 0, а частота выбрана заведомо низкой, чтобы можно было «поиграть» с параметрами анимации в ISIS. На основании этих примеров уже можно сделать важные выводы.

Для того, чтобы при многоразрядной динамической индикации получить реальную картинку необходимо, чтобы кадр анимации в симуляторе захватывал полный цикл индикации, т.е. длительность кадра должна быть не менее суммы длительностей индикации всех разрядов.

Ну и второй, не менее важный момент, который неоднократно рассматривался на форуме и до меня – наиболее простой метод получить стабильный цикл индикации, это смена разрядов в прерывании по переполнению одного из таймеров микроконтроллера. Именно нестабильность периода индикации, который зависит от подаваемой на вход частотомера Денисова измеряемой частоты, приводила к срыву картинки в рассмотренном в самом начале FAQ примере. Там мы побороли этот глюк другим методом, которого коснемся чуть ниже. Кстати, если кто-то захочет воспользоваться кодом из примеров индикации в реальном устройстве, хочу остановиться еще на одном моменте. Я делал все исключительно для примера и соответственно упрощенно – только индикация. Поэтому у меня отсутствует запрет на прерывания в начале обработчика и разрешение в конце. В реальном девайсе это может сыграть злую шутку, если в момент обработки прерывания по таймеру прилетит еще одно с более высоким приоритетом. Об этом тоже нелишне помнить.

Ну а пока вернемся к примеру с **DOTMATRIX** индикатором и проанализируем – почему по-прежнему светятся две точки? Вот тут и выходит на первый план параметр **Minimum Trigger Time** модели на основе **LEDMPX.DLL**. Попробую объяснить это графически, поскольку чисто словесное

описание этого момента, как я не пытался, получается слишком запутанным для понимания. Обратимся к рисунку 102. Здесь я условно принял, что граница фреймов **Frame1** (бирюзовый фон) и **Frame2** (бледно-зеленый фон) совпадают с фронтами смены сигналов по строкам **R1** и **R2** и по столбцам – **C1** и **C2**, т.е. выполняется условие рассмотренное выше.

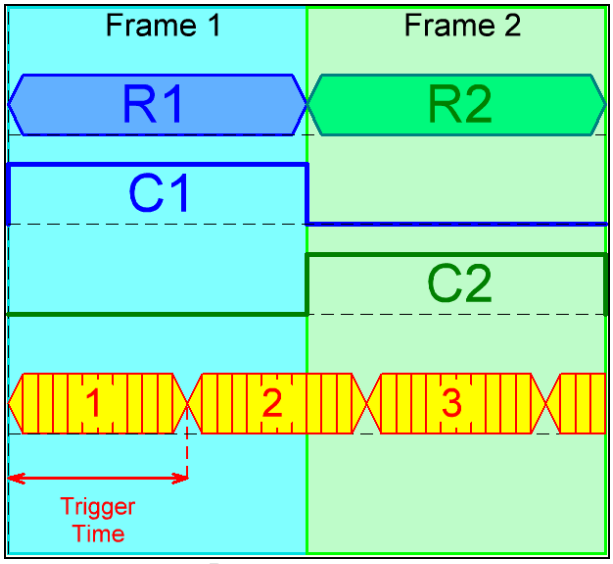


Рис. 102

По умолчанию **Minimum Trigger Time** для нашего индикатора задано равным 1мс, т.е. для одного кадра длительностью 50мс **LEDMPX.DLL** рассчитывает состояние выводов индикатора 50 раз. Причем нет никакой гарантии, что границы стробирования **Minimum Trigger Time** будут четко совпадать с границами кадров и смены знакомест и очередной строб не заползет большей своей частью в соседний разряд (знакоместо). Я даже допускаю мысль, что внутри интервала Trigger Time программист заложил усреднение нескольких рассчитанных значений, поэтому и нарисовал дополнительные красные стробы. Если бы я писал модель, то вероятно так бы и сделал. Но в реальности очень похоже на то, что «первую скрипку» тут играет конечное значение Trigger Time, т.е. как бы задний фронт этого интервала. Как мы помним, на основании рассчитанных данных **LEDMPX** принимает решение о том, какие активные элементы должны светиться в данном кадре. А теперь посмотрим на интервал обозначенный цифрой 2. Какое решение выдаст **LEDMPX** для этого интервала? Он вроде бы относится к первому кадру, но цепляет данные и для **R2** и **C2**. Естественно, это вызовет подсвечивание в данном кадре активных элементов соседнего знакоместа. А теперь применим гашение на границе смены разрядов, ну и кадров естественно. Для этого просто достаточно убрать данные, например на шине **R** на интервале длительностью чуть больше нашего **Minimum Trigger Time** (Рис. 103).

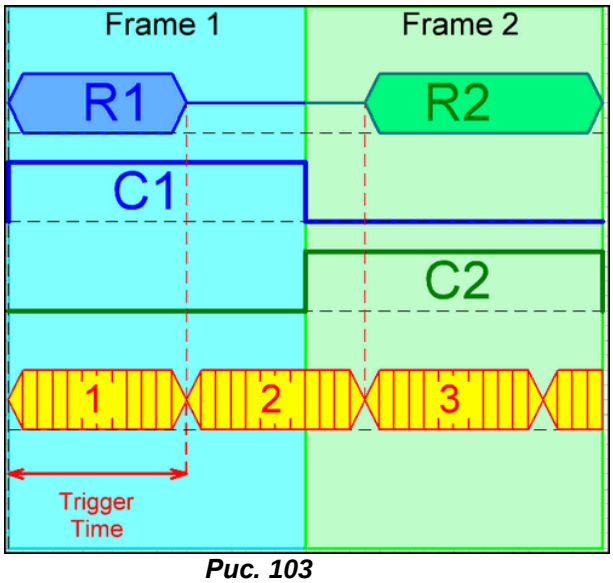


Рис. 103

Вот и вся операция. Интервал 2, хоть и заползает на второй кадр, но засветки данных из соседнего знакоместа не даст – для него отсутствуют активные сигналы в строке R, а следующий

уже полностью относится ко второму кадру. Причем, как мы убедились на примере частотомера в начале FAQ, такой прием работает и при нестабильном периоде индикации. Большого умственного напряжения это не требует. Например, в листингах на Си примеров для микроконтроллеров во вложении к этому разделу вначале обработчика прерывания стоит закомментированной всего одна строка, которая убирает сигнал с порта, управляющего сегментами на момент переключения порта знакомест. Вот она то и даст этот эффект.

Ну а сейчас мы обойдем этот момент другим путем. Правда, сразу оговорюсь, что этот прием работает только при стабильном периоде индикации. Давайте подумаем, а надо ли стробировать данные 50 раз за период индикации, если они при этом не меняются? Возьмем, да и приравняем **Minimum Trigger Time** к длительности активного сигнала для знакоместа. Вот именно равное значение в данном случае не работает, но стоит чуть-чуть увеличить **Minimum Trigger Time**, например, длительность сигнала знакоместа равна 10мс, а значение **Trigger Time** мы установим равным 10,001мс, и картинка приходит в норму. Попробую объяснить, почему такое происходит. При равных значениях видимо по-прежнему сказывается захват соседнего знакоместа. Если же **Minimum Trigger Time** чуть больше длительности знакоместа, сбой тоже происходит, но совпадение фазы сигналов настолько редко, что на общем фоне большого количества кадров этот сбой практически не мешает зрительному восприятию картинки. Иногда мелькнет неадекватная информация, и опять нормальное отображение. Причем, еще раз подчеркну, в данном случае речь идет не о полном цикле индикации, а именно о длительности одного знакоместа. Примеры использования данного приема в папке вложения **GOOD_EXAMPLES**. В случае с активной матрицей мы при равных длительности кадра и сигнале столбца со значением **Minimum Trigger Time** чуть большим, чем сигнал столбца получаем единственную светящуюся точку как в шаге, так и при запущенной кнопкой **Play** симуляции. В случае многоразрядного сегментного индикатора если долго и внимательно смотреть на индикатор, то можно засечь проскок наложения разрядов, но общей картине восприятия он не мешает.

Ну и чтобы окончательно покончить с этим вопросом, подытожим вышесказанное.

Для симуляции динамической индикации в ISIS в любом случае наиболее предпочтителен способ гашения индикатора на момент смены разрядов (знакомест). При этом длительность интервала гашения должна быть хотя бы чуть больше длительности параметра **Minimum Trigger Time**, который установлен для индикатора.

При стабильном периоде индикации можно попробовать добиться желаемого эффекта, установив параметр **Minimum Trigger Time** чуть больше длительности активного сигнала одного знакоместа.

Еще один важный момент, который я чуть не упустил. В своих примерах я использовал индикацию с минимальной частотой, чтобы можно было отследить изменение параметров анимации. На деле же динамическую индикацию стараются сделать с частотой выше 50 Гц, чтобы исключить мерцание реальных индикаторов. Естественно, подогнать параметры анимации ISIS под один период индикации в таких случаях нам не удастся из-за программных ограничений Протеуса, но задать длительность кадра таким, чтобы в него входило некоторое целое количество периодов можно и даже желательно. Для этого достаточно исследовать индикацию с помощью цифрового графика, как это сделано в приведенных примерах. Дальше все построено на банальной арифметике.

Ну и в заключение несколько слов о том, каким выбрать **Minimum Trigger Time**. На него, в отличие от параметров анимации, нет жестких ограничений. Тут тоже все зависит от конкретного построения схемы и программы динамической индикации. В большинстве случаев все работает и с параметрами по умолчанию, но иногда приходится повозиться. В сторону уменьшения значение ограничено лишь тем, что при маленьком значении на индикаторе начнут светиться данные соседних знакомест (Рис. 104). В сторону увеличения **Minimum Trigger Time** ограничено длительностью кадра анимации и если превысит его, то у нас просто начнутся пропуски данных – хаотичное моргание отдельных разрядов (Рис. 105), или индикация будет отсутствовать. Если применяется гашение при смене разрядов, то значение **Minimum Trigger Time** надо установить меньше, чем интервал гашения.

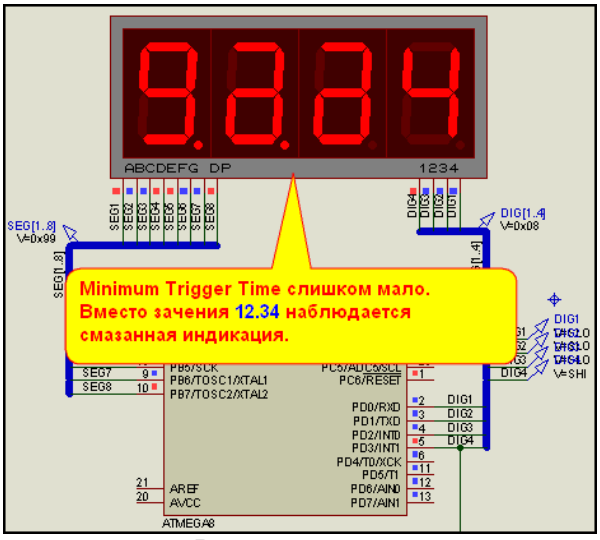


Рис. 104

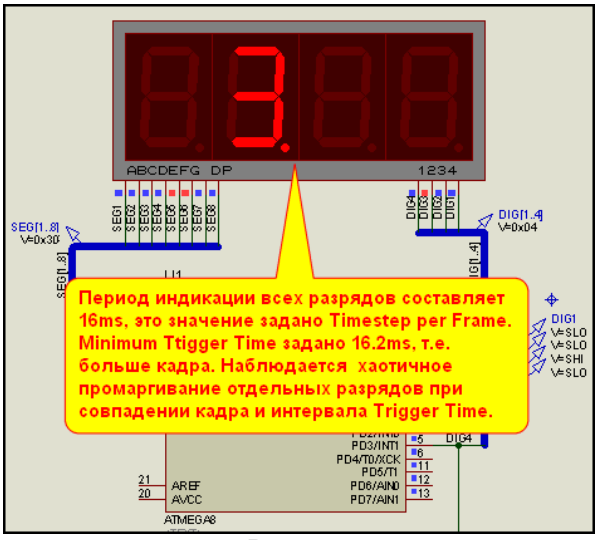


Рис. 105

Эти примеры расположены в папке **MC_TR_TIME** вложения. Соответствующие пояснения даны в комментариях на поле проектов. Ну а я завершаю окончательно и бесповоротно материал о динамической индикации в Протеусе и надеюсь больше к этому вопросу не возвращаться. Теперь вы знаете все о поведении моделей **LEDMPX** при симуляции, а выбор метода реализации динамической индикации остается за вами. Ну а мы переходим к не менее интересному материалу, который уже давно заждался, – индикаторы на базе **LCDMPX.DLL**.

К содержанию

8.10. Знакомимся с моделями на основе LCDMPX.DLL – еще одним вариантом библиотеки для построения цифровых индикаторов в ISIS. Общие принципы построения моделей ЖК индикаторов.

Из названия библиотеки по аналогии с описанной выше напрашивается элементарный вывод, что служит она для создания моделей жидкокристаллических индикаторов (**LCD** от английского **L**iquid **C**rystal **D**isplay), и так же как предыдущая мультиплексирована (**MPX** от multiplexing**)**. Ну, если второй термин себя полностью оправдывает, то **LCD** в общем то только потому, что она изначально разрабатывалась для имитации ЖК цифровых индикаторов. Но, как мы уже убедились на примере **LEDMPX**, в которой диодными свойствами и близко не пахнет, аналогично обстоит дело и здесь. **LCDMPX** тоже обладает только цифровыми свойствами, т.е. тока не «кушает» и имеет только два состояния: активный элемент «горит», активный элемент потушен. Я не стану расписывать здесь теорию управления реальными ЖК индикаторами, для этого есть соответствующая литература. Про особенности конструкций ЖК индикаторов и принципы частотного и фазового управления ими вы можете прочесть в старых справочниках, например:

- **МРБ вып. 1122. Пароль Н.В., Кайдаров С.А. Знакосинтезирующие индикаторы и их применение. М., Радио и связь, 1988 г.;**
- **Вуколов Н.И., Михайлов А.Н.. Знакосинтезирующие индикаторы. Справочник. Под ред. В.П. Балашова. М., Радио и связь, 1987 г.**

Особенности управления конкретными реальными ЖК индикаторами необходимо учитывать при разработке своих конструкций, поскольку от этого зависит «продолжительность жизни» вашего индикатора, а модель ISIS построена так, что допускает статическое управление и легко пропустит ошибки, допущенные в динамике. Еще раз хочу подчеркнуть, что пока мы ведем речь о простых индикаторах, у которых имеется один общий вывод для каждой группы сегментов индикатора, просьба «чайников» от электроники не путать с COG-индикаторами, их мы коснемся особо.

В библиотеках ISIS цифровые ЖК индикаторы представлены весьма скромно всего пятью моделями, расположенными в **Optoelextronics => LCD Panels Displays**. Объясняется этот факт по-видимому отсутствием спроса на данный тип моделей у легальных «забугорных» пользователей. Нам же, сирым и убогим, выковыривающим ЖК индикаторы из отслуживших свой срок калькуляторов, телефонов и прочей оргтехники отсутствие этих моделей доставляет массу неудобств. Да и китайско-белорусская промышленность до сих пор исправно снабжает Российский рынок сегментными ЖК-индикаторами. Это и широко распространенные индикаторы Минского **ОАО "ИНТЕГРАЛ"** http://www.integral.by и цифровые ЖК индикаторы китайской фирмы **Intech LCD Group** http://www.intech-lcd.com/standardpanel.htm. Подчас применение этих сравнительно дешевых индикаторов позволяет весьма существенно сэкономить деньги в простых приложениях типа электронных часов, таймеров, термометров и пр. «бытовухи». Ну и конечно, многим хочется иметь такие модели для предварительной отладки в ISIS. Вот этим мы сейчас и займемся.

Итак, для того чтобы активировать («зажечь») сегменты ЖК индикаторов на базе **LCDMPX** достаточно подать на них питание (пример индикатора **VI-402-DP** слева) или логические уровни (пример индикатора **VI-332-DP** справа) в правильной полярности (Рис. 106).

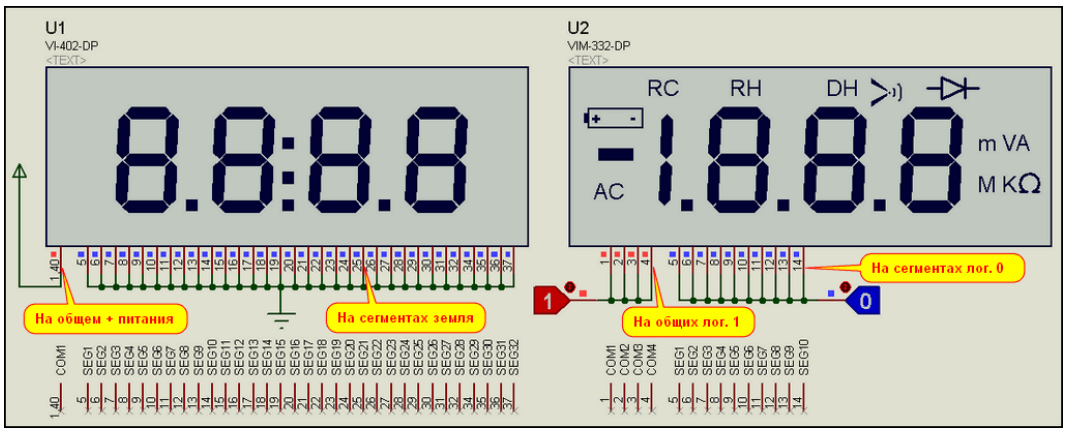


Рис. 106

По сути, этими двумя индикаторами: для часов и для мультиметра и ограничен наш выбор в ISIS. Оставшиеся три в библиотеке – заказные индикаторы для отладочных микроконтроллерных комплектов и практической ценности для рядового пользователя не представляют.
Чуть ниже индикаторов на рисунке я расположил их выводы с подсвеченными именами. Мы можем заметить, что часовой индикатор имеет один общий вывод с именем **COM1** и 32 сегментных вывода с именами **SEG1…SEG32**. Индикатор справа в отличие от первого имеет четыре общих вывода – **COM1…COM4** и меньшее количество сегментных. Наталкивает на мысль об аналогии с моделями на основе **LEDMPX**, рассмотренными ранее, но это не совсем корректно. Дело в том, что в модели **LEDMPX**, как вы вероятно помните, развертывание знакомест (нумерация) идет слева направо по горизонтали, т.е. общему выводу **1** соответствует левая цифра, выводу 2 вторая слева и далее с шагом шириной в одно знакоместо (цифру из N-сегментов с подложкой-фоном). В модели **LCDMPX** имеет место относительная двумерная пространственная ориентация по осям X-Y для сегментов, соответствующих определенному общему выводу с именем **COM**. На примере **VI-332-DP** для **COM1** и **COM4** расположение соответствующих сегментов представлено на рисунке 107. Не очень привычно, но дает более широкие возможности при создании собственных моделей.

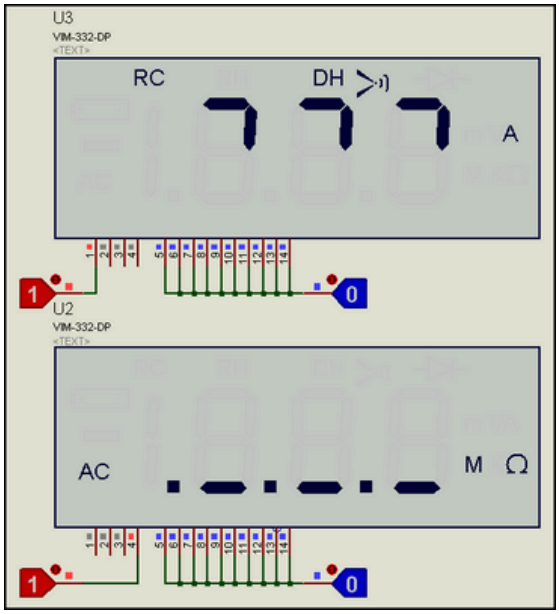


Рис. 107

Чтобы нам в дальнейшем не путаться выводы «знакомест» (конструктивно это выводы задней пластины ЖК индикатора) **COM**n от английского «common» – общий, я так и буду называть – общие выводы, потому что термин «знакоместо» здесь весьма условен, ну а выводы сегментов – **SEG**n так и останутся выводами сегментов. Итак, максимальное количество и тех и других для моделей на основе **LCDMPX** – 64. Нумерацию можно вести кому как привычнее, т.е. могут существовать **SEG1…SEG64** или **SEG0…SEG63**. Чем ограничение в 64 сегмента чревато в

действительности? Среди реально существующих цифровых ЖК индикаторов немного найдется с раздельными общими выводами. Как правило, он один на всю заднюю пластину-подложку. А вот выводов сегментов может оказаться больше, чем допустимый предел. Типичные примеры: таксофонный **ИЖЦ13-8/7** (8 цифр с точками и 8 символов надчеркивания над ними) или **Intech ITS-E0806** (8 цифр с точками плюс два разделительных двоеточия).

Дальнейшее рассмотрение продолжим на примере **VI-332-DP**, поскольку он наиболее показателен с точки зрения построения моделей на базе **LCDMPX.DLL**. Для начала «поколотим его молотком» (**Decompose**) и посмотрим на состав входящих в него символов, переключив селектор в соответствующее положение кнопкой **S** слева (Рис. 108).

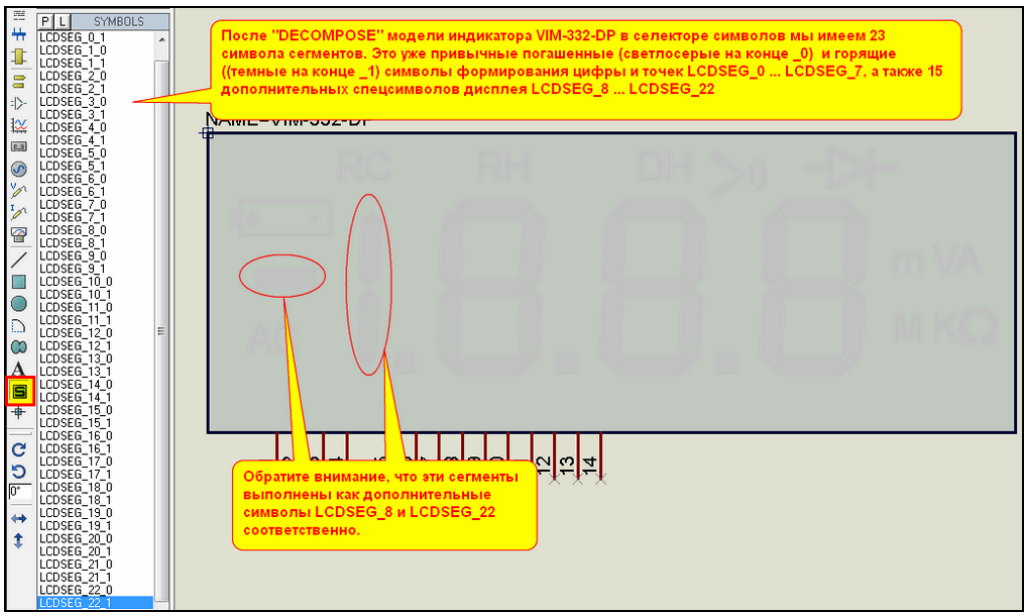


Рис. 108

Всего в составе индикатора 23 символа. Символы **LCDSEG_0**…**LCDSEG_7** нам уже знакомы по предыдущим семисегментным индикаторам. Это образующие цифру сегменты и десятичная точка. Для примера на рисунке 109 в верхнем ряду приведены первые четыре символа сегментов дополнительно «разобранные на запчасти», чтобы был виден маркер **ORIGIN**. Здесь тоже никаких новшеств по отношению к остальным сегментным индикаторам за исключением цвета погашенного (с индексом _0) и горящего (с индексом _1) символа.

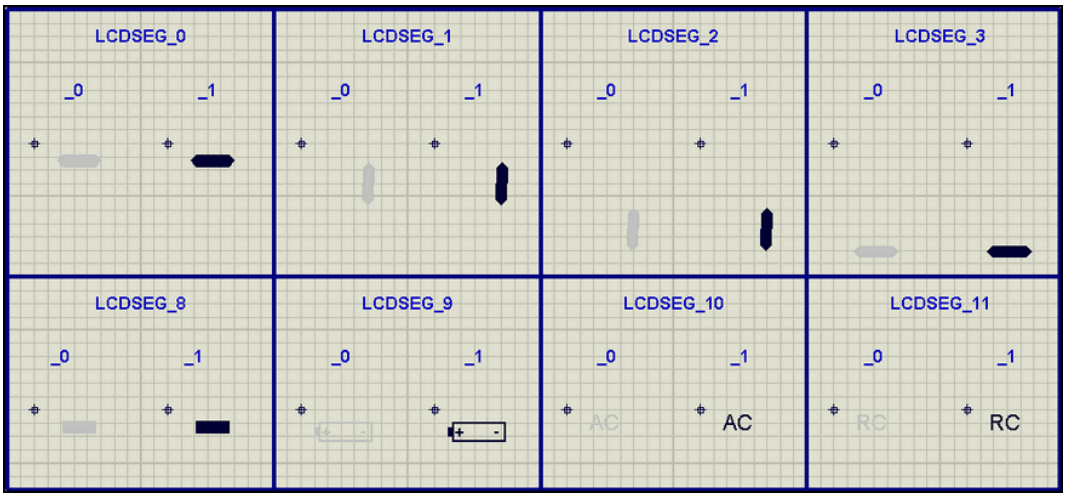


Рис. 109

В нижнем ряду также «разобранные на запчасти» первые четыре дополнительных символа. Здесь я хотел бы заострить ваше внимание на положении маркера **ORIGIN**, потому что именно с ним будет связано все, что будет изложено ниже, а именно принцип позиционирования символов в моделях на основе **LCDMPX.DLL**.

Снова вернемся к полноценной модели **VI-332-DP** и заглянем в ее свойства. В окне **Edit Properties** установим флажок **Edit all properties as text**, чтобы увидеть то, что изначально скрыто от посторонних глаз (Рис. 110).

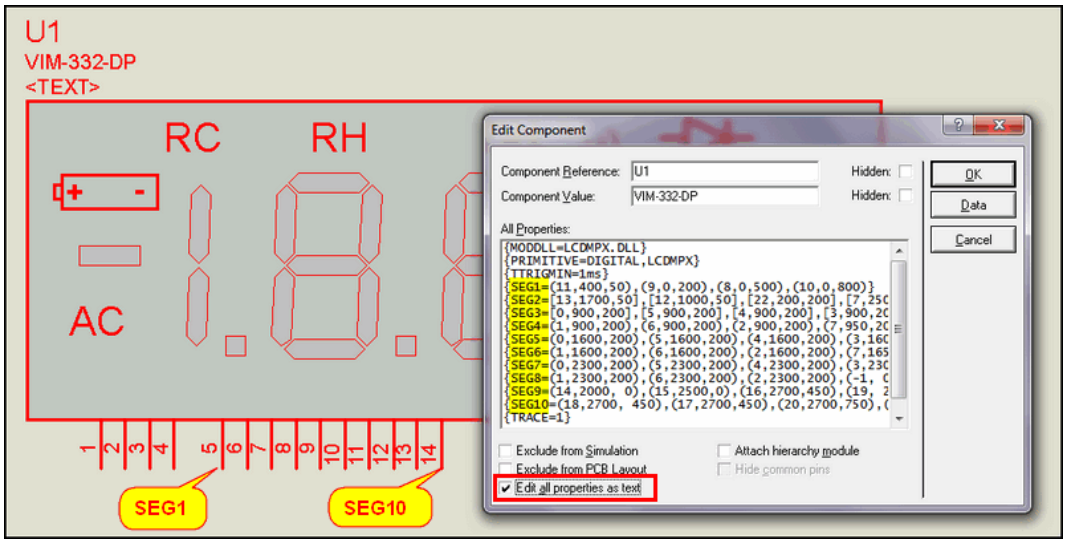


Рис. 110

Ниже я полностью привожу те десять строчек (по числу сегментных выводов SEG1…SEG10), которые нас интересуют:

```
{SEG1=(11,400,50),(9,0,200),(8,0,500),(10,0,800)}
{SEG2=[13,1700,50],[12,1000,50],[22,200,200],[7,250,200]}
{SEG3=[0,900,200],[5,900,200],[4,900,200],[3,900,200]}
{SEG4=(1,900,200),(6,900,200),(2,900,200),(7,950,200}
{SEG5=(0,1600,200),(5,1600,200),(4,1600,200),(3,1600,200)}
{SEG6=(1,1600,200),(6,1600,200),(2,1600,200),(7,1650,200)}
{SEG7=(0,2300,200),(5,2300,200),(4,2300,200),(3,2300,200)}
{SEG8=(1,2300,200),(6,2300,200),(2,2300,200),(-1, 0,0)}
{SEG9=(14,2000, 0),(15,2500,0),(16,2700,450),(19, 2700,750)}
{SEG10=(18,2700, 450),(17,2700,450),(20,2700,750),(21, 2700,750)}
```

Вот это и есть основа позиционирования сегментов и с ней нам сейчас предстоит разобраться. Итак, для начала вспомним, что фигурные скобки в начале и конце каждой строки означают сокрытие текста (**Hidden Text**) и поэтому нас в данный момент мало интересуют. Как вы наверное уже догадались, каждая строка соответствует выводу сегментов, с имени которого она начинается, т.е. **SEG1** соответствует одноименному выводу с номером 5, ну а **SEG10** – выводу с номером 14. После знака равенства имеются четыре секции по три числа, заключенные в круглые скобки и разделенные запятыми. Именно четыре общих вывода **COM** имеет данная модель, если кто подзабыл – вернитесь к рисунку 106. Нетрудно догадаться, что левая секция в круглых скобках соответствует выводу с именем **COM1**, следующая **COM2** и т.д. Нам осталось разобраться с тем, что находится внутри круглых скобок. Тут тоже ничего сверхъестественного: первое число – номер **n** символа **LCDSEG_n**, второе и третье соответственно положение по осям **X** и **Y** в координатах сетки маркера **ORIGIN** этого символа относительно маркера **ORIGIN** всей модели (виден в левом верхнем углу на Рис. 108). Вот именно это я и обозвал выше по тексту заумной фразой двумерная относительная пространственная ориентация. Чтобы было более наглядно, попробую изобразить это на картинке на примере одного символа, например, **LCDSEG_9** (Рис. 109) с координатами **X=0**, **Y=200**, соответствующего изображению батарейки. Данный символ в соответствии со второй секцией в круглых скобках для строки **SEG1=** в свойствах должен стать активным при подаче на **COM2** логической единицы, а на **SEG1** – логического нуля. Проверяем… (Рис. 111).

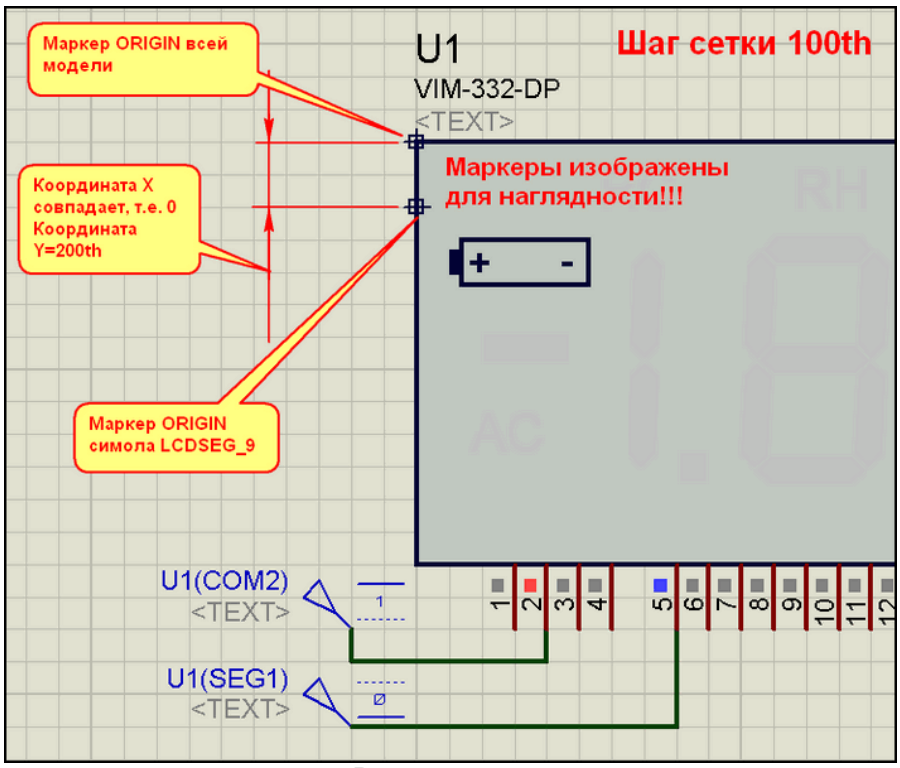


Рис. 111

Как видим, наши предположения полностью подтвердились. Сами изображения маркеров **ORIGIN** я нанес на рисунок для наглядности. По оси X смещение маркера символа отсутствует, по оси Y равно двум клеткам, т.е. при шаге сетки **100th** соответствует числу 200.

Ну и в заключении к этому познавательному материалу, прежде чем мы перейдем к практическому созданию моделей, несколько существенных замечаний.

Первое из них касается описания координат в свойствах модели, а конкретно разделительных запятых. Привычного для обычного текста пробела после запятой, а уж тем более до нее лучше не делать. Это касается как написания координат внутри круглых скобок, так и запятых между секциями. Хотя, если посмотреть на последнюю строчку, мы видим их присутствие в модели в двух местах, но исходя из личного опыта, особенно с более ранними версиями Протеуса, я все же рекомендую их не делать. В ряде случаев наблюдалась неадекватная реакция симулятора на наличие пробелов в свойствах сегментов.

Второе замечание касается относительной координаты по оси Y. Предопределяя желание некоторых любителей покричать: «Вот это глюк! Так глюк!», поясняю – реально в последней картинке смещение по Y должно быть отрицательным – ведь оно направлено вниз, и мы столкнемся с этим фактом при построении своих моделей. Почему же стоит положительное число 200? Программисты меня и автора **LCDMPX** поймут с полуслова – а стоит ли таскать внутри модуля знаковые константы, если они используются только внутри и никак не связаны со всей остальной программой? Поэтому не удивляйтесь, что конструируя свои модели мы будем получать отрицательное смещение по оси Y, но скромно забывать при этом о знаке числа.

Ну и последнее, если Вы заглянете самостоятельно в свойства модели часового индикатора VI-402-DP с установленной галочкой, то увидите, что там для описания свойств строки SEG= имеют только одну секцию с координатами, ведь у него только один общий вывод COM1, зато строк сегментов будет гораздо больше. Желающие могут скачать даташиты на эти индикаторы, они доступны при подключенном Интернете при нажатии кнопки **Data** в свойствах модели, либо через клик правой кнопкой и выборе опции **Display datasheet**. Правда, сразу могу огорчить, на просторах России я этих индикаторов не встречал – не завозят.

К содержанию

8.11. «Фальшивая» точка начала координат - наш помощник в деле создания графики индикаторов. Трансформируем модель VI-402-DP в шестиразрядный индикатор ITS-E0809.

Будем считать, что со структурой модели на основе **LCDMPX.DLL** мы познакомились. Наступил черед практического создания первой самостоятельной модели на основе данной библиотеки. В качестве прототипа для нашей будущей модели я выбрал четырехразрядный часовой ЖК-дисплей **VI-402-DP**, с которым мы познакомились в предыдущем параграфе. А создадим мы на его основе не какую-нибудь «экзотику», а более распространенный на необъятных

просторах нашей Родины шестиразрядный ЖК индикатор ITS-E0809 уже упоминавшийся мною выше китайской фирмы **Intech LCD Group**. Эти индикаторы до сих пор продаются в ряде Интернет-магазинов, как в России – «Платан», «Чип-Дип», так и в ближнем зарубежье – украинский «Космодром».

Но прежде чем мы перейдем непосредственно к созданию модели хочу рассказать об одной «фишке» Протеуса, именно всего Протеуса, т.к. данная опция работает и в ISIS, и в ARES, в ряде случаев значительно облегчающей жизнь при создании графики моделей. А заключается она в преднамеренной установке «фальшивой» точки начала координат на поле проекта. Данная опция описана в HELP по **Universal Keypad Model**, и мы будем ее также использовать при создании моделей клавиатуры, а пока я расскажу – как это работает на практике. Для начала «разберем на запчасти» (**Decompose**) наш прототип **VI-402-DP**. Я рекомендую вам сразу же увеличить параметры листа проекта (**System => Set Sheet Sizes…**), хотя бы до А3, так как создание графики требует простора для творчества. Кроме того, неплохо бы сразу выделить полностью все составляющие разбитой модели и через кнопку **Block Copy** накидать по полю проекта пару-тройку резервных копий разбитой модели, чтобы всегда иметь в запасе «неиспорченные» нашим творчеством запчасти. И еще, перед началом экспериментов убедитесь, что шаг сетки стоит так, как принято по умолчанию – **100th**. Более мелкий шаг при создании моделей обычно необходим только при прорисовке мелких деталей графики, а в данном случае он будет только мешать. Итак, совмещаем курсор мышки с центром маркера **ORIGIN** одной из разбитых моделей в ее левом верхнем углу и нажимаем клавишу с латинской буквой «**O**» (не путать с нулем!!!). Кстати, переключать раскладку клавиатуры в нашем случае совсем не обязательно, Протеус среагирует адекватно и на русскую «**Щ**». Что при этом произойдет, показано на рисунке 112.

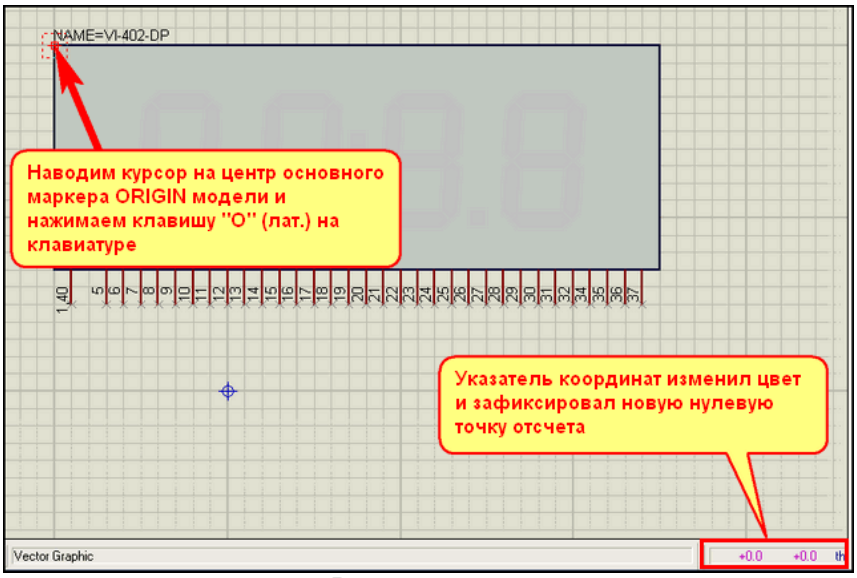


Рис. 112

Как видим, цифровые значения координат в окне программы в правом нижнем углу слегка «полиловели» и приняли нулевое значение. Теперь любое телодвижение мыши будет изменять их значение относительно новой точки отсчета – левого верхнего угла модели или установленного там маркера **ORIGIN**. Если накануне было «слегка принято на грудь» и «рука мыша дрожать устала», то никогда не поздно вернуться к первоначальному отсчету повторным нажатием клавиши «**O**». Затем, собрав волю в кулак и сфокусировав зрение на нужной точке можно повторить операцию. Теперь рассмотрим, зачем нам такие мучения…

Поместим из селектора символов какой-нибудь контрастный (засвеченный) символ на соответствующее ему место первой цифры индикатора. В качестве примера я выбрал символ сегмента «A» (верхнего горизонтального). В селекторе символов он у нас фигурирует, как **LCDSEG_0_1**. Почему я выбрал именно контрастный темный? Да потому, что потом его легко отследить и удалить, хотя при достаточном запасе резервных копий модели можно и не «париться» с этой процедурой. Теперь на этом месте мы его тоже **Decompose**, чтобы увидеть положение маркера самого символа (Рис. 113).

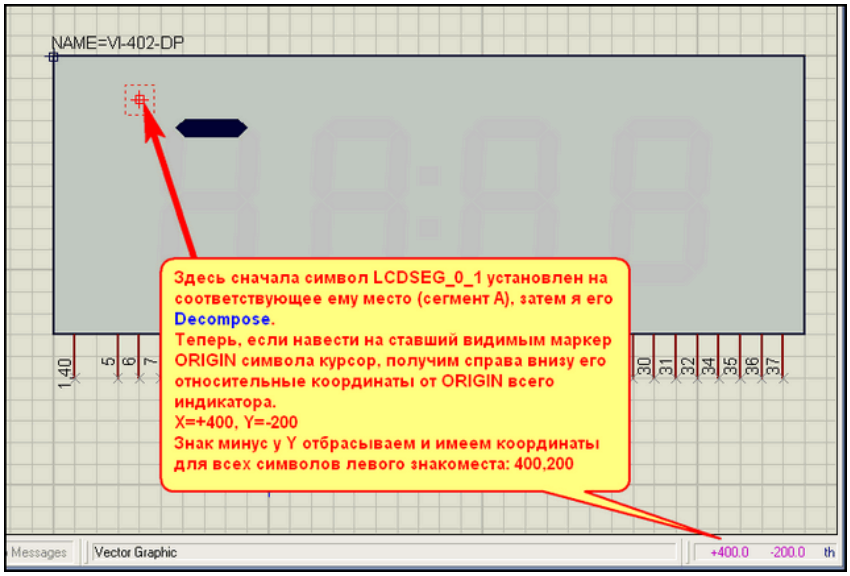


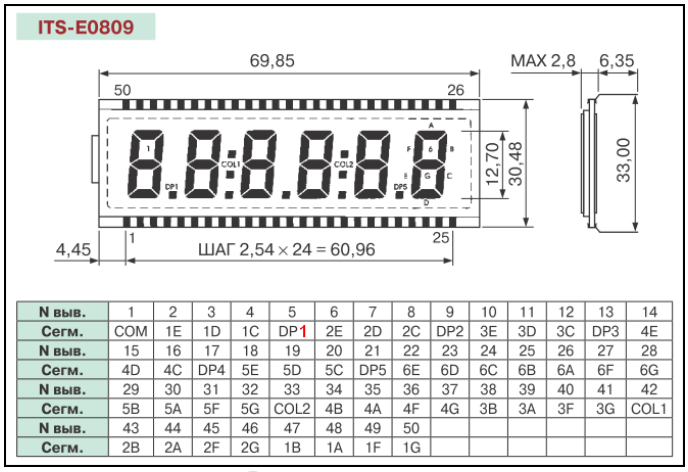
Рис. 113

Наводим курсор на появившийся маркер **ORIGIN** символа и в правом нижнем углу получаем координаты относительного смещения этого маркера от маркера **ORIGIN** всей модели. Как я уже упоминал, поскольку смещение вниз, координата **Y** имеет отрицательное значение, на знак мы не обращаем внимания. Итак, для сегментов первой цифры, мы получили значение **400,200**. Если теперь заглянуть в свойства исходного, не разбитого индикатора **VI-402-DP** при установленном флажке **Edit all properties as text**, то мы увидим, что для ряда сегментов (**SEG1…SEG4** и **SEG29…SEG32**) использована именно эта пара значений координат. Аналогичным образом определяются координаты для символов остальных знакомест (Рис. 114).

Рис. 114

Не надо быть Фурье или Лобачевским, чтобы вывести одну закономерность – для всех символов модели индикатора смещение по оси **Y** составляет **200th**, а интервал по оси **X=700th**. Именно с таким интервалом нам и предстоит расположить дополнительные две цифры нашей будущей модели. Надеюсь, что вы самостоятельно можете проделать эту работу без подробных комментариев. Вкратце, все сводится к следующему: растягиваем вправо тело-подложку на нужную ширину и, выделив нужный участок графики на разобранной модели, копируем его на новое место. После того, как работа проделана, имеет смысл проверить, что мы не промахнулись в графике, а заодно и сверить координаты добавленных элементов, как мы это только что проделали. Соответственно для пятой и шестой цифр они будут **3200,200** и **3900, 200**. Если у вас получилось по другому, значит где-то вы сдвинули знакоместо.

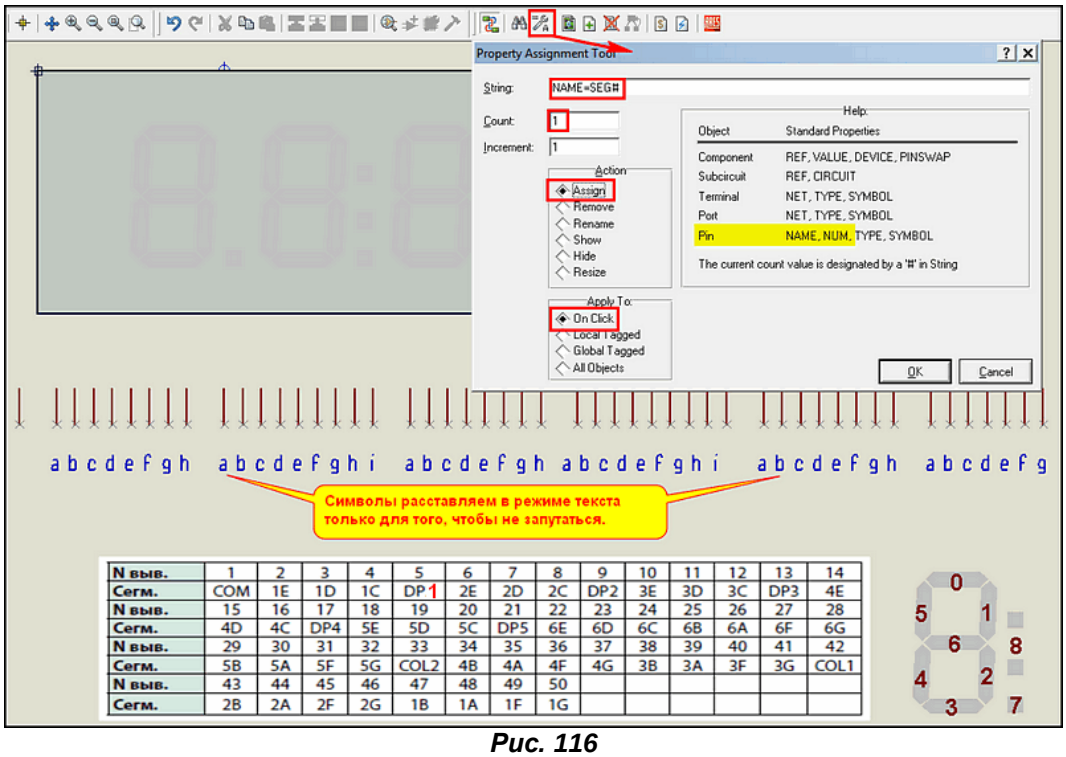
Ну, вот мы и разобрались с «крыльями» нашей модели, пора приниматься за «ноги и хвосты». Для этого нам потребуется даташит индикатора ITS-E0809, который доступен на сайте фирмы **Intech LCD Group**, по ссылке в предыдущем параграфе. Я же воспользуюсь данными на индикатор из каталога с сайта www.platan.ru. В нем они расположены в более компактной форме и помещаются здесь на одном скриншоте из данной документации (Рис. 115). Меня в данном случае интересует нижняя таблица соответствия выводов индикатора сегментам. Но, сразу предупреждаю, там есть ошибка, которую я поправил красным цветом.



N выв.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Сегм.	COM	1E	1D	1C	DP1	2E	2D	2C	DP2	3E	3D	3C	DP3	4E
N выв.	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Сегм.	4D	4C	DP4	5E	5D	5C	DP5	6E	6D	6C	6B	6A	6F	6G
N выв.	29	30	31	32	33	34	35	36	37	38	39	40	41	42
Сегм.	5B	5A	5F	5G	COL2	4B	4A	4F	4G	3B	3A	3F	3G	COL1
N выв.	43	44	45	46	47	48	49	50						
Сегм.	2B	2A	2F	2G	1B	1A	1F	1G						

Рис. 115

Далее можно идти различными путями. Кому то может понравиться вариант расположения выводов так, как у реального индикатора, а кому то захочется расположить их в один ряд. В данном случае это возможно только с шагом **50th**. При этом придется скрыть имена и номера выводов, а для удобства расположить их группами, соответствующими каждой цифре. Итак, приступаем к расстановке выводов. Хотя я и таскал из картинки в картинку выводы от прототипа, но сейчас рекомендую их удалить. Дело в том, что при большом количестве «ног» удобнее и быстрее воспользоваться опцией **Property Assignment Tools** (**PAT**), но у нее иногда бывают мелкие глюки именно при замене одного имени на другое. Чтобы обезопасить себя от этого проще поставить свежие выводы без номеров и имен, с ними проблем не бывает, тем более времени это отнимет немного. Поставили пяток выводов, потом обвели их, выделили и … **Block Copy** столько сколько нужно. Сначала я расположу выводы группами с шагом **100th** и на некотором расстоянии от модели, а когда закончу нумерацию и наименование, пододвину на нужные места. Еще один прием, которым я часто пользуюсь в таких случаях, это сохраняю в формате **BMP** картинку с расположением выводов в данном случае нижнюю таблицу с рисунка 115 и втаскиваю ее в проект с нужным масштабом через **File => Import Bitmap**. Этот прием исключит из творческого процесса постоянное перепрыгивание из окна в окно для проверки соответствия номера, названию. Можно, конечно, распечатать на бумаге, тогда будете прыгать глазами экран-бумага-экран. Наверное, этот вариант имеет право на жизнь – зарядка для глаз сохранит зрение, но макулатуры в хозяйстве добавится. Там же имеет смысл расположить одно графическое изображение всех сегментов с надписанными номерами, и просто текстовые строки под группами выводов, которые дополнительно облегчат жизнь при расстановке номеров и имен выводов (Рис.116).



N выв.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Сегм.	COM	1E	1D	1C	DP1	2E	2D	2C	DP2	3E	3D	3C	DP3	4E
N выв.	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Сегм.	4D	4C	DP4	5E	5D	5C	DP5	6E	6D	6C	6B	6A	6F	6G
N выв.	29	30	31	32	33	34	35	36	37	38	39	40	41	42
Сегм.	5B	5A	5F	5G	COL2	4B	4A	4F	4G	3B	3A	3F	3G	COL1
N выв.	43	44	45	46	47	48	49	50						
Сегм.	2B	2A	2F	2G	1B	1A	1F	1G						

Рис. 116

Итак, запускаем **PAT**, в строке **String** набираем **Name=SEG#** а в строке **Count** меняем стартовое значение с **0** на **1**, как показано на том же рисунке 116. Как вы уже догадались, на данном этапе мы будем использовать **PAT** для операций с выводами (**Pin**), конкретно сначала для имен **NAME**, затем для номеров **NUM**, я выделил желтым цветом это в **Help** окна **PAT**. Убеждаемся, что **Action** оставлено по умолчанию **Assign** (назначить) и **Apply To** – **On Click** (по клику мыши), нажимаем кнопку **OK**. Теперь пробегаемся кликами мышкой по всем выводам сегментов слева направо, общий вывод **COM1** мы назовем вручную позже. Затем снова вызываем **PAT** и набираем в строке **String** значение **NUM=#**, опять исправляем **Count** с **0** на **1** и опять **OK**. Дальше строго руководствуемся таблицей и расставляем номера выводов в соответствии с ней, не забывая присвоить номер **1** отдельно стоящему выводу **COM1**. Что после этого получилось, показано на рисунке 117.

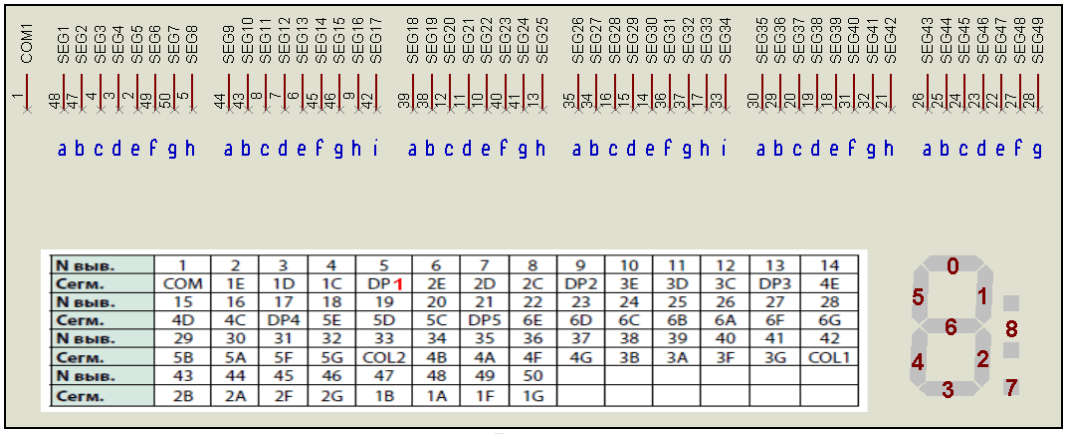


Рис. 117

Теперь тщательно проверим, что мы не наделали ошибок и воспользуемся все тем же **PAT**, чтобы скрыть имена и номера выводов, потому что когда они будут расположены с шагом **50th**, текст соседних выводов будет перекрываться. Для этого выделяем мышью все выводы модели и вновь давим кнопку **PAT** в верхнем меню. В строке **String** набираем просто **NAME**, устанавливаем переключатель **Assign** в положение **Hide** (скрыть), а **Apply To** в положение **Local Target** и нажимаем **OK** (Рис. 118). Аналогично повторно выделяем выводы и повторяем вызов **PAT**, но вместо **NAME** ставим **NUM**, чтобы скрыть номера выводов. Вы наверное уже догадались, что если вместо **Hide** выбрать **Show** (показать), то вместо скрытых имен/номеров, получим видимые. Самое время освоить **Property Assignment Tools** тем, кто этого еще не сделал.

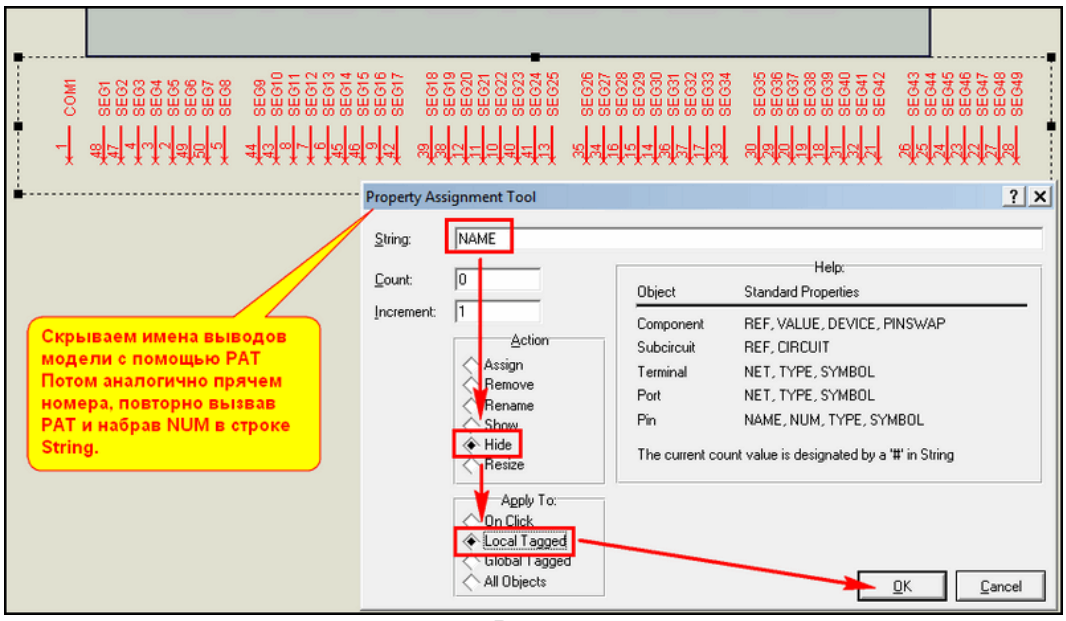


Рис. 118

Итак, я установил шаг сетки **50th**, сдвинул выводы в группах и придвинул их к модели (Рис 119). Попутно я добавил на изображение модели текстовые строки с подсказкой о назначении выводов. Теперь графика полностью готова к созданию модели индикатора **ITS-E0809**.

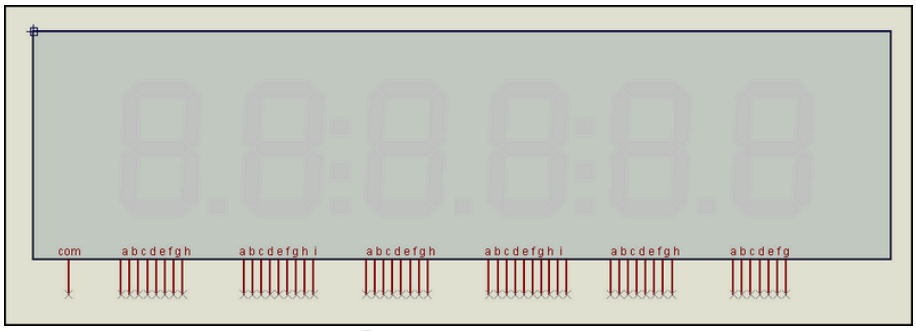


Рис. 119

Настал черед процедуры **Make Device**. Причем на первом этапе я не стану добавлять координаты сегментов, а просто создам устройство с базовым набором свойств для **LCDMPX.DLL**. Мы уже столько раз проходили **Make Device** в процессе изучения Протеуса, что, надеюсь, нет смысла приводить скриншоты с подробным видом задаваемых свойств, достаточно описать – что и где задается. На первой вкладке **Make Device** задаем нашему девайсу имя – **ITS-E0809**, префикс – **U** (а кто желает может и **H** или **HG**) , а также задаем параметры активного компонента: **Symbol Name Stem** – **LCDSEG**, количество – **9**, устанавливаем флажки бит-зависимости и связи с DLL. Окно с корпусом оставляем не заполненным, а в третьем окне добавляем следующие свойства:

Name	Description	Type	Type	Default Value	Visibility
PRIMITIVE	Primitive Type	String	Hidden	DIGITAL,LCDMPX	Hide Name & Value
MODDLL	VSM Model DLL	String	Read Only	LCDMPX.DLL	Hide Name & Value
TTRIGMIN	Trigger Time	String	Hidden	1ms	Hide Name & Value
TRACE	Diagnostic Messages	Trace mode	Hidden	Warnings Only	Hide Name & Value

Последнее, относящееся к диагностике, можно и не добавлять, чем я и воспользовался в прилагаемом примере. В последнем окне выбираем категорию **Optoelectronics**, подкатегорию **LCD Panels Dysplays**, добавляем через кнопку **New** производителя **Intech LCD Group**, и что-нибудь типа **6 Digits LCD Panel** в описание девайса. Сохраняем все это в нужной библиотеке. Как всегда, свободной для записи является **USRDVC**, но при желании можно добавить модель к существующим дисплеям – там еще 13 позиций свободно, или создать свою библиотеку дисплеев. О том, как снять защиту записи с библиотек или создать свою говорилось ранее.

В принципе, модель уже «работоспособна» с точки зрения того, что при симуляции не выскакивают ошибки, но и индикации соответственно никакой. Почему я не стал сразу добавлять координаты сегментов? Ну, во-первых, потому что это можно осуществить двумя способами, и я их сейчас опишу, а во-вторых, потому что их много и лучше добавлять их не скопом, а в несколько приемов, проверяя работоспособность на каждом этапе.

Итак, вариант первый – «классический». Описание сегментов добавляем через третье окно **Make Device**, при этом нам придется каждый раз вводить вручную все параметры. Процесс представлен на рисунке 120.

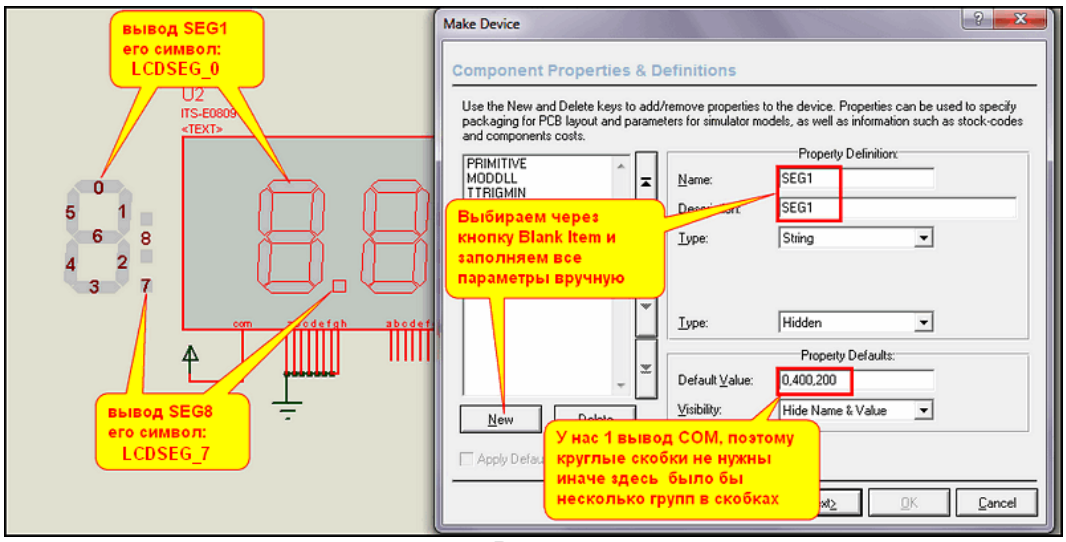


Рис. 120

Для созданной нами модели **ITS-E0809** запускаем **Make Device**, переходим в третье окно и через кнопку **New** начинаем добавлять **Blank Item**. Для каждого сегмента нам предстоит руками задать **Name**, **Description**, установить **Type=Hidden** и в графе **Default Value** задать номер символа и его координаты вручную. Для всех сегментов, составляющих первую цифру координаты одинаковы и равны **400,200**, а вот номер символа будет отличаться. Для верхнего сегмента «a» первой цифры – **SEG1** он будет равен **_0**, а десятичной точке – **SEG8** соответствует символ **LCDSEG_7**. Данный метод хорош тем, что в любой момент можно прекратить добавление сегментов, пройти процедуру **Make Device** до конца, сохранить изменения и проверить – что и как работает. Для примера на рисунке 121 приведен момент проверки, когда в модели добавлены параметры только для первых четырех сегментов **SEG1…SEG4**, т.е. сегменты «**a, b, c, d**» первой цифры. Хотя активные сигналы поданы на все выводы первой цифры, работают только эти сегменты. Таким образом, можно в любой момент проверить свои действия. Но, как видим, процедура ручного ввода при большом количестве активных элементов – довольно рутинное и утомительное занятие.

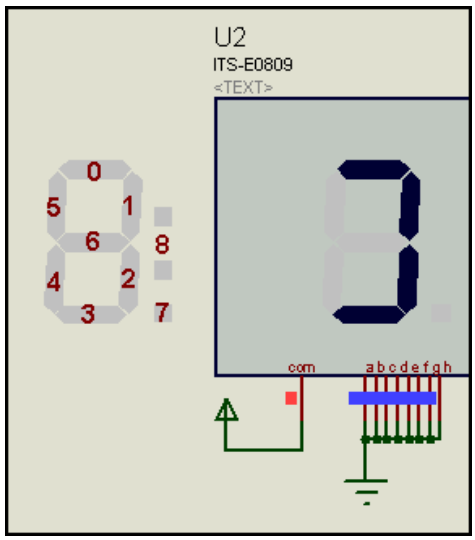


Рис. 121

Второй вариант заключается в том, что данные по сегментам заранее готовятся в любом текстовом редакторе и добавляются сразу, непосредственно в окне **All Properties** свойств нашей модели (Рис. 122). Тут тоже возможен самоконтроль на любом этапе, а процедура **Make Device** запускается только один раз, когда все сегменты добавлены и проверены.

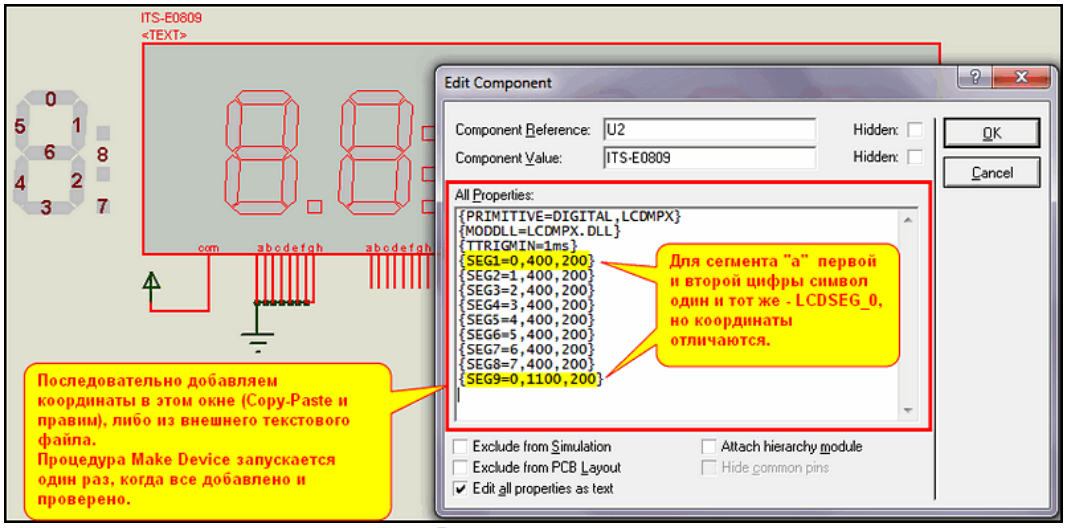


Рис. 122

Вспомним, что для того чтобы параметры были скрытыми (**Hidden**) достаточно заключить их в фигурные скобки. Но это относится только к отображению параметров в окне проекта на поле чертежа на месте серого **<TEXT>**. За отображение же параметров в окне свойств модели отвечает второй сверху параметр **Type** на третьей вкладке **Make Device**. При таком способе добавления свойств он по умолчанию остается **Normal**. Это означает, что если мы не изменим его на третьей вкладке для каждого сегмента, выбрав из раскрывающегося списка **Hidden**, то в конечном итоге

получим в готовом девайсе параметры **SEG** выделенными в отдельные строки. На рисунке 123 приведен пример как это выглядит при вызове **Edit Device Properties** для первых четырех **SEG**.

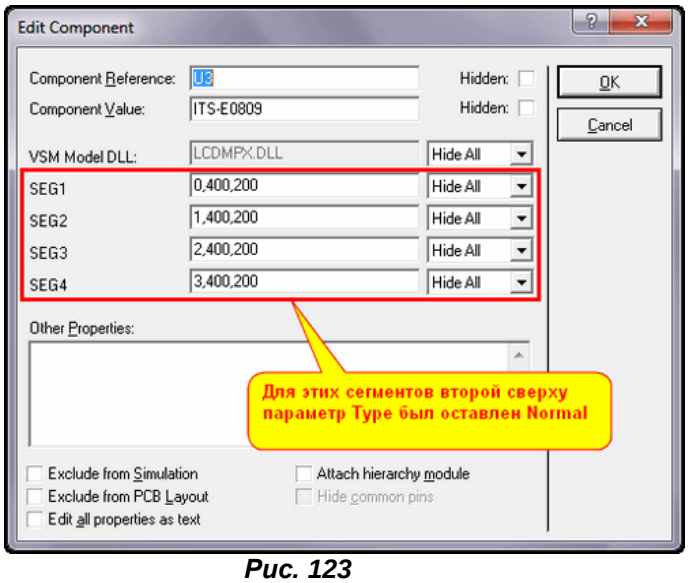


Рис. 123

А представьте, что таких строк будет почти полста? Окошко растянется больше чем экран компьютера. Поэтому, при этом варианте при запуске **Make Device** необходимо на третьей вкладке пробежать мышкой по всем добавленным **SEG** и изменить второй сверху **Type** на **Hidden**. Все же это быстрее, чем набирать на клавиатуре.

Можно пойти и на маленькую военную хитрость, чтобы обмануть Протеус. Заглянем в текстовый скрипт «разобранного» индикатора **VI-402-DP**. Там вначале мы можем найти строки вида:

{SEG1="SEG1",HIDDEN STRING}
{SEG2="SEG2",HIDDEN STRING}

Находятся эти строки в разделе, начинающемся со строки **{*PROPDEFS}**, а сами свойства в разделе, начинающемся со строки **{*COMPONENT}**. Таким образом, можно заранее подготовить аналогичный текст в текстовом редакторе для наших 49 сегментов и добавить в окно **All Properties** перед процедурой **Make Device**. Применительно к первым четырем сегментам рисунка 123 это будет выглядеть так, как показано на рисунке 124.

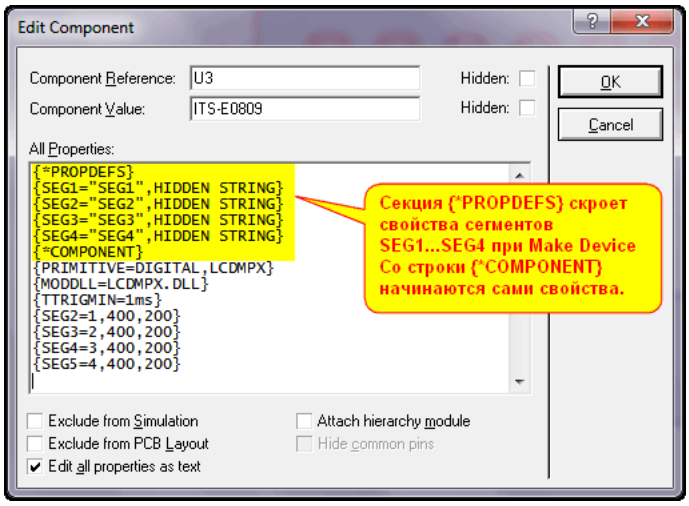


Рис. 124

Конечно же, второй способ удобнее и практичнее, и лично я всегда пользуюсь им. Если данные готовятся в хорошем текстовом редакторе, то всегда можно воспользоваться «продвинутыми» функциями автозамены в выделенном тексте и прочими удобствами редактирования, а не заниматься тупым «копи-пастом» с последующей рутинной ручной правкой.

Вот, вроде, и все хитрости создания собственной модели ЖК индикатора на основе **LCDMPX.DLL**. Как вы уже поняли, главным ее отличием от **LEDMPX** является возможность размещения сегментов по двум координатам, что дает больше возможностей для создания своих

моделей. Кроме того, в моделях на основе **LEDMPX** выводы **SEG** для каждого «знакоместа» с общим выводом **COM** индивидуальны, а не объединены, как в **LEDMPX**. Ну и еще хочу заметить, что в рассматриваемом примере мы воспользовались стандартной расцветкой символов для ЖК (черно-серым вариантом), но это совсем не догма, можно использовать символы и подложку с другой цветовой гаммой и на основе **LCDMPX** сделать модели светодиодного типа. Единственное, что пока тормозит – это ограничение на 64 сегмента, которое, надеюсь, когда-нибудь Лабцентер расширит, как это было со светодиодной библиотекой.

Ну и в заключение несколько слов о том как добавить даташит к модели. Для этого на соответствующей вкладке процедуры **Make Device** указываем имя файла даташита. В моем случае это **E0809.pdf** (Рис. 125).

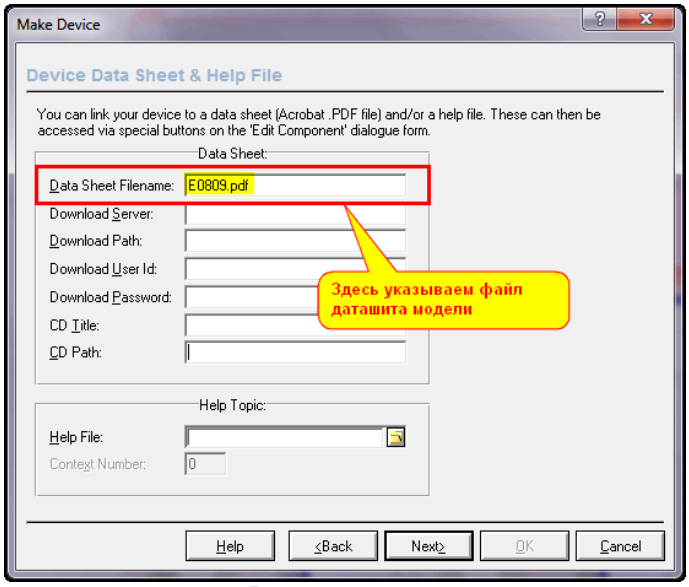


Рис. 125

Это совсем не лишняя информация, особенно, когда у модели скрыты номера и имена выводов. Сам файл даташита должен помещаться в определенной директории. Как определить в какой именно – ясно из рисунка 126. Там показан путь по умолчанию для **Windows 7**, соответственно у «счастливых» обладателей **Windows XP** он будет отличаться. Этот путь в любой момент можно изменить и тогда все скачанные из Интернета даташиты Протеус будет помещать в другую, указанную вами папку. Только не забудьте туда переместить файлы, которые Вы скачали ранее. Ну и конечно в нее же помещаем файл даташита нашей модели, который я поместил во вложении. Если этого не сделать, то Протеус будет пытаться скачать его с сайта **Labcenter**, где его нет и в помине.

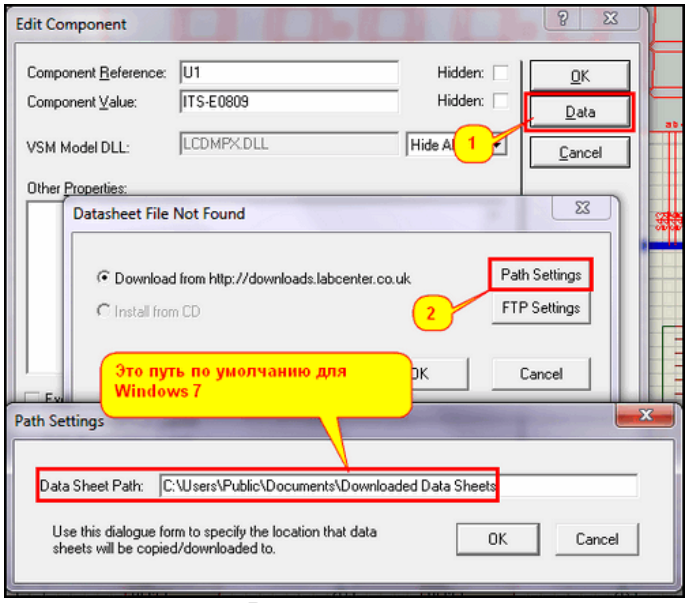


Рис. 126

Во вложении отдельно представлен проект с последовательностью создания графической модели и тестовый проект с пробной симуляцией готового девайса. Конечно же, никаких готовых файлов модели там нет. Если вам необходима модель **ITS-E0809** в ваших установленных библиотеках, достаточно тупо пройти всю процедуру **Make Device** от начала до конца для готовой модели из проекта **TEST.DSN**, ничего не меняя. Ну, можете добавить файл даташита на соответствующей вкладке, и в конце выбрать отличную от **USRDVC** библиотеку, если она у вас уже существует и не защищена от записи.

Мы же далее рассмотрим некий «симбиоз» ЖК модели на основе **LCDMPX** и схематичной модели контроллера для восполнения отсутствующих в библиотеках реальных индикаторов, например COG с контроллером **ML 1001**.

К содержанию

8.12. Реализация «составной» модели ЖК индикатора TIC5231 на основе схематичной модели COG драйвера ML1001 и модели индикатора на основе LCDMPX.DLL в Протеусе.

Возможность воспроизвести в Протеусе обычный сегментный ЖКИ – это конечно замечательно, если количество сегментов невелико, а как быть если мы имеем ЖКИ на 6 или 8 цифр. Занимать под управление таким ЖКИ несколько портов микроконтроллера – это расточительство. В то же время уже давно промышленно выпускаются драйверы для управления ЖКИ как в виде отдельных микросхем, так и интегрированные на стеклянную подложку самого индикатора. Именно последние и носят название – **COG** от английского **Chip On Glass** (микросхема на стекле). Характерным представителем таких драйверов является **ML1001**. На ее основе в свое время тайваньской фирмой **Ampire** была выпущена серия цифровых сегментных ЖК индикаторов TIC, которые еще до сих пор встречаются в продаже. Реализацию одного из них 6,5-разрядного **TIC5231** мы рассмотрим ниже.

Эта модель была разработана около года назад по просьбе одного из участников форума. К сожалению, модели на основе **LCDMPX** поддерживают управление только от пинов **COM** и **SEG**, расположенных на основном листе проекта, и попытка присоединить к модели ЖКИ контроллер, схема которого реализована на дочернем листе не удалась. Именно поэтому родился такой симбиоз: «мухи отдельно – котлеты отдельно». Отдельно была сформирована модель ЖКИ со стандартными ножками **COM** и **SEG** и отдельно сделана схематичная модель COG контроллера **ML1001**.

Сама модель индикатора (Рис. 127) никаких особенностей не имеет. Стандартный ЖКИ, имеющий один вывод **COM** и 40 выводов **SEG**.

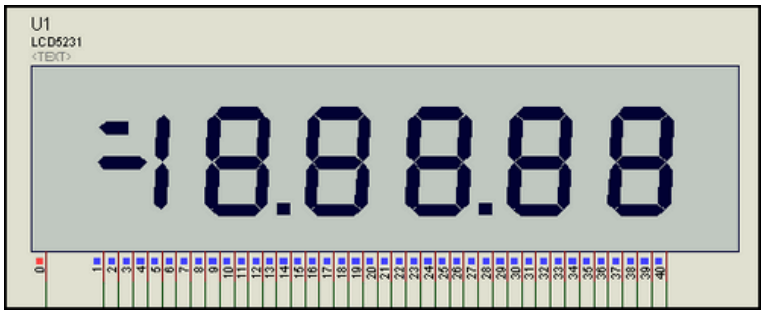


Рис. 127

Единственное, над чем пришлось потрудиться, это создать графические символы для самого левого «половинчатого» разряда (Рис. 128). В результате появились символы отдельно стоящего минуса, урезанного символа надчеркивания и полный знак единицы в виде одного символа, которые и составляют самое левое знакоместо индикатора **TIC5231**.

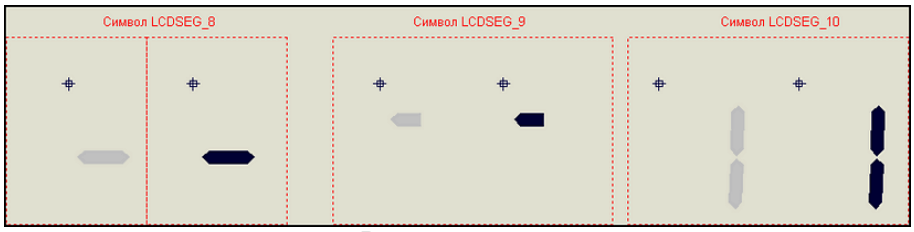


Рис. 128

Первоначальный вариант графики имел полностью 41 вывод, но немного поразмыслив, я решил, что поскольку отдельно такой индикатор практической ценности не представляет, есть

смысл оформить выводы сегментов шиной на 40 разрядов. Это позволяет более компактно располагать индикатор и присоединенный к нему контроллер в проекте.

Хочу также обратить внимание на несколько нестандартную нумерацию сегментов в данном индикаторе (Рис. 129).

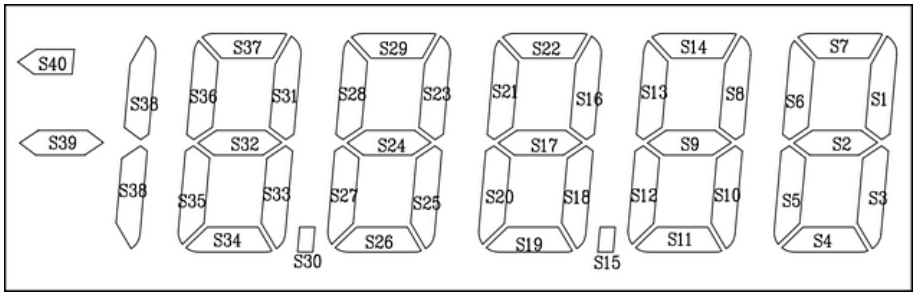


Рис. 129

Полный даташит находится во вложении, а из расположения сегментов на рисунке из него мы видим, что номер 1 принадлежит сегменту «**b**» самого правого разряда индикатора, номер 2 – сегменту «**g**», хотя для нас логичнее было бы, чтобы это были сегменты «**a**» и «**b**» соответственно. Об этом не стоит забывать, применяя реальный индикатор, иначе вместо цифр получите полную чехарду с сегментами. В остальном графическая модель особенностей не имеет, т.к. выводы нужны только для «внутреннего» соединения с контроллером, я дал им наименование **SEG** и нумерацию в соответствии с номерами сегментов реального индикатора. В окончательном варианте это просто 40-разрядная шина **SEG[1..40]**.

Теперь рассмотрим сам драйвер **ML1001** и реализацию модели для него. Во вложении имеется как оригинальный даташит от компании **Minilogic Device Corporation**, так и его перевод выполненный Игорем Данко. Сайт автора перевода на данный момент не работает, поэтому я принял решение включить всю документацию во вложение.

Блок-схема драйвера приведена на рисунке 130.

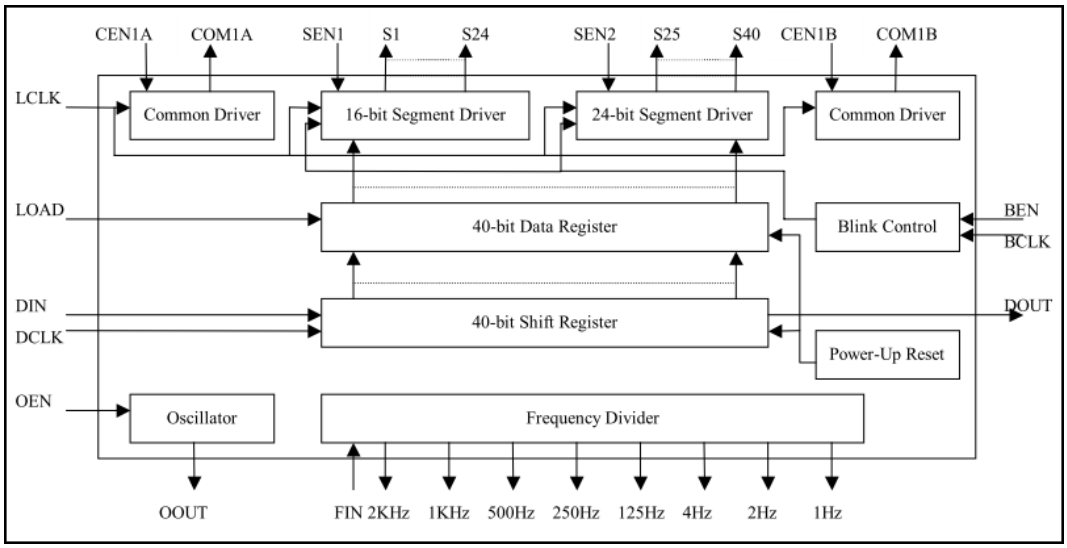


Рис. 130

Драйвер имеет в своем составе 40-разрядный сдвиговый регистр с последовательным вводом данных. Данные загружаются с входа **DIN** по тактовой частоте на входе **DCLK**. По сигналу на входе **LOAD** данные переносятся в 40-битный регистр-защелку, с выхода которого распределяются по сегментам индикатора с помощью дополнительных встроенных управляющих схем. С помощью выхода **DOUT** драйвера можно соединять последовательно, что дает возможность наращивать разрядность индикатора с кратностью 40. Конкретно для индикатора **TIC5231** используется один чип, но учитывая, что модель **ML1001** может использоваться и для индикаторов с большим количеством разрядов, есть смысл реализовать наличие этого вывода в модели. Кроме того, в реальном чипе имеются встроенный генератор и делитель частоты, но с точки зрения моделирования они нас мало интересуют. Если рассмотреть реальную реализацию в индикаторе (Рис. 131), то имеются только 7 внешних выводов, из которых мы еще не рассмотрели 3.

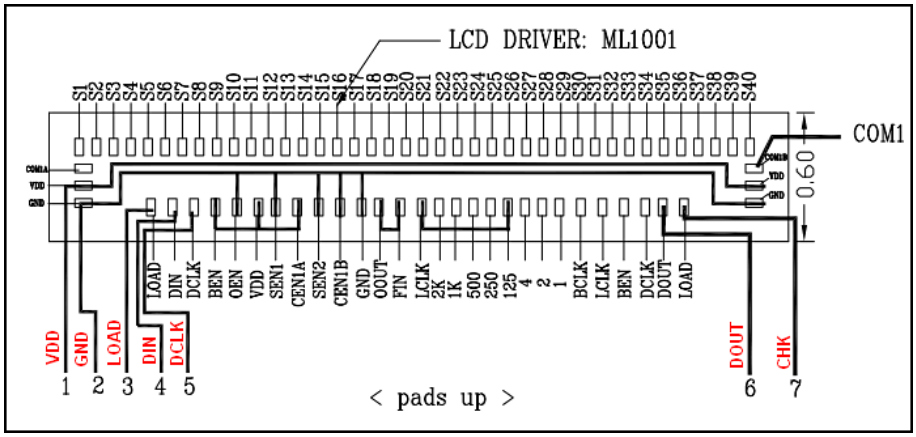


Рис. 131

Ну, с выводами **VDD** и **GND** все ясно и без комментариев – это выводы питания, а вывод **7**, обозначенный в даташите индикатора, как **CHK** – это фактически дубль вывода **LOAD**. Таким образом, для реализации модели нам нужны всего четыре «видимых» вывода: **DIN**, **DCLK**, **LOAD** и **DOUT**, ну и, конечно, выходная шина на 40 сегментов. В итоге, для начального тестового варианта графическая модель **ML1001** имеет вид, представленный на рисунке 132.

Рис. 132

Для тестирования лучше использовать вариант с выводами, а после убрать выводы выходов и заменить их шиной. Теперь перейдем к внутренней реализации модели **ML1001**, т.е. к тому «фаршу», из которого мы слепим нашу «котлету» на дочернем листе. Здесь тоже особых премудростей нет, на дочернем листе я разместил 4 стандартных примитива сдвиговых десятиразрядных регистров и 5 восьмиразрядных регистров-защелок. Фрагмент схемы с дочернего листа приведен на рисунке 133.

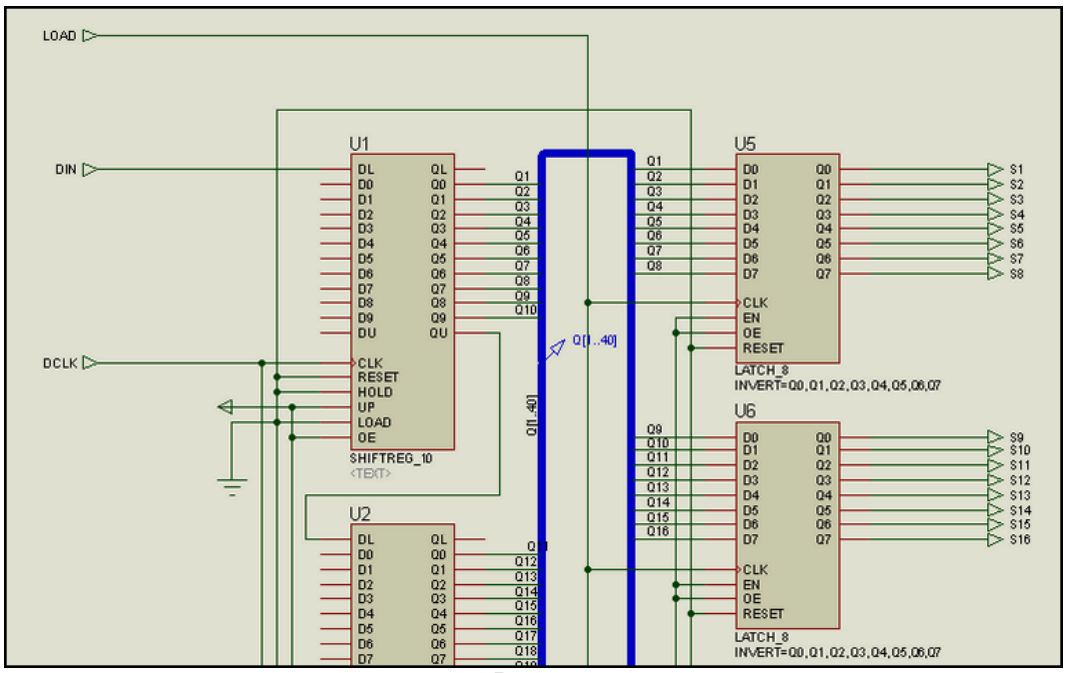


Рис. 133

Для того, чтобы согласовать по уровням сигналы с моделью ЖКИ выходы регистров-защелок инвертированы. С выхода **QU** четвертого регистра сдвига сигнал подается на ножку **DOUT**. После тестирования этого варианта с дочернего листа был сформирован файл **ML1001.MDF**. Он

будет одинаковым как для графической модели с отдельными выводами на выходе, так и с 40-разрядной шиной.

Для того, чтобы использовать данные модели в своих проектах, поместите файл **ML1001.MDF** из любой папки вложения в папку **MODELS** установленного Протеуса, а для моделей индикатора **TIC5231** и драйвера **ML1001** из проекта **TEST.DSN** в папке **ML1001_final_bus** проведите **Make Device** от начала до конца с сохранением в нужной библиотеке. Мы же в следующем параграфе рассмотрим как «обмануть» Протеус, смоделировав ЖКИ с числом сегментов свыше 64.

К содержанию

8.13. Модель ЖК индикатора TIC8148 (TIC55) на основе схематичной модели двойного драйвера ML1001 со встроенным генератором.

В качестве «подопытного» кролика для реализации модели с количеством сегментов более 64 я выбрал тоже достаточно распространенный индикатор **TIC8148**. У него есть брат-близнец – **TIC55**, так что модель можно смело использовать и для того и для другого индикатора. Данный индикатор представляет из себя 8 восьмерок с точками и спецсимволами в виде 8 галок в нижней строке. Таким образом, мы имеем в сумме 72 светящихся сегмента, что явно превышает возможности **LCDMPX.DLL** для одного общего входа **COM**. Даташит, как обычно, находится во вложении, а на рисунке 134 из этого даташита я выделил цветом значимые для нас места. Итак, нумерация цифр самого индикатора справа-налево, т.е. самая левая - цифра 8. Буквенное обозначение сегментов совпадает с общепринятым, но вот десятичная точка обозначена как **P** (по-видимому, от Point), а символом **H** обозначен дополнительный символ - галочка в нижней строке.

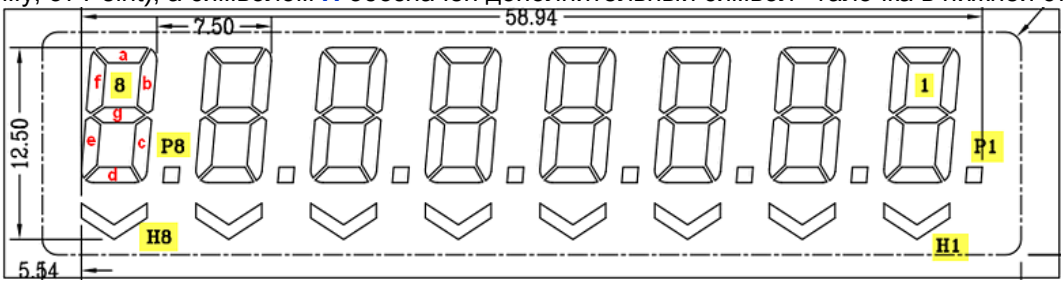


Рис. 134

Вы, конечно, догадались, что 72 сегмента превышают не только возможности **LCDMPX**, но и возможности одного драйвера **ML1001** (40 сегментов), поэтому в реальном индикаторе их 2, включенных последовательно, как рассматривалось выше. А вот если глянуть на таблицу распределения сегментов из того же даташита (Рис. 135), то тут опять начинается «китайская грамота». Немудрено, ведь на Тайване те же лица, только в профиль. Итак, согласно таблице, первому выходу **S1** первого драйвера соответствует сегмент **8D** – нижний горизонтальный самой левой цифры.

PIN	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20
COM1	8D	8E	8G	8F	8A	8B	8C	H8	P8	7D	7E	7G	7F	7A	7B	7C	H7	P7	6D	6E
PIN	S21	S22	S23	S24	S25	S26	S27	S28	S29	S30	S31	S32	S33	S34	S35	S36	S37	S38	S39	S40
COM1	6G	6F	6A	6B	6C	H6	P6	5D	5E	5G	5F	5A	5B	5C	H5	P5	4D	4E	4G	4F
PIN	S41	S42	S43	S44	S45	S46	S47	S48	S49	S50	S51	S52	S53	S54	S55	S56	S57	S58	S59	S60
COM1	4A	4B	4C	H4	P4	3D	3E	3G	3F	3A	3B	3C	H3	P3	2D	2E	2G	2F	2A	2B
PIN	S61	S62	S63	S64	S65	S66	S67	S68	S69	S70	S71	S72	S73	S74	S75	S76	S77	S78	S79	S80
COM1	2C	H2	P2	1D	1E	1G	1F	1A	1B	1C	H1	P1								

Рис. 135

Соответственно по первому тактовому импульсу на входе **DCLK** мы заносим данные для него, далее согласно строкам таблицы, вплоть до 72-го тактового импульса, который соответствует десятичной точке самой правой цифры, т.е. №1. Об этом следует помнить при написании программ для управления данными индикаторами. Еще хочу обратить ваше внимание на то, что дополнительный символ галочки в этой последовательности предшествует символу десятичной точки данного знакоместа, т.е. перебираются все сегменты, затем галка и уже последней – десятичная точка. Это тоже накладывает некоторые особенности на написание программ. В частности, если даже мы не используем в своем девайсе символы нижних галок. Допустим, для хранения информации о знакоместе используется байтовая переменная: семь бит – сегменты и

один бит – информация о точке. При последовательном выводе на индикатор нам придется перед выводом информации о десятичной точке вставлять «пустышку» для сегмента нижней галочки данного знакоместа. Так, что-то я отклонился от темы, это уже особенности программирования, а мы вернемся к нашей модели и начнем с графики.

В качестве символов сегментов я использовал все те же символы от «разобранного» **VI-402-DP**, но пришлось добавить символ нижней галочки. Кроме того, у нас в индикаторе теперь 8 цифр и для двух правых знакомест появились новые координаты – **4600,200** и **5300,200** (Рис. 136). Шаг по оси **X** по-прежнему **700**. Ну и еще небольшой «финт» с выводами. Сегменты я сразу же загнал в шину на 24 разряда **S[1..24]**, а общих выводов сделал три **COM1**, **COM2** и **COM3**, т.е. мы получим индикатор на 72 сегмента, но выводами **COM** надо управлять попеременно, а сегменты мультиплексировать. Собственно, для этого и существует **LCDMPX.DLL**.

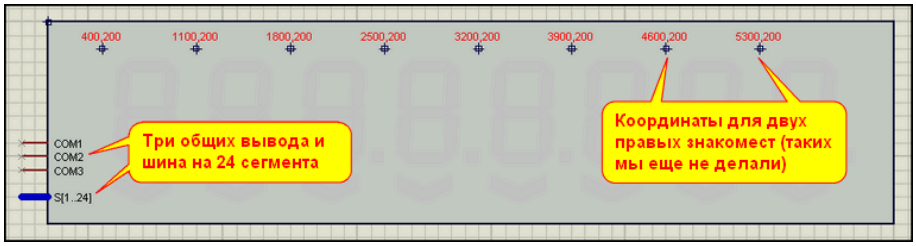


Рис. 136

Теперь небольшое лирическое отступление на тему – почему именно 3 вывода **COM**, ведь для 72 сегментов по логике достаточно было бы и двух. Я тоже первоначально надеялся, что хватит двух, но в ходе разработки модели выявился один неприятный момент **LCDMPX.DLL**. Если количество сегментов для одного вывода **COM** превышает 32, а этих выводов два или более, то при мультиплексировании начинает наблюдаться моргание всех сегментов, относящихся к выводам **COM2**, **COM3** и т.д. И избавиться от этого эффекта всеми известными мне приемами так и не удалось. Так что на будущее примите совет, при моделировании ЖК индикаторов с одним общим выводом можно использовать до 64 сегментов, при моделировании с двумя и более общими выводами количество сегментов для одного общего вывода не должно превышать 32. Чтобы не быть голословным, я оставил во вложении первоначальный вариант с двумя **COM** в папке **BAD_Model_Test**. Просто запустите симуляцию и вы увидите, что правая часть сегментов индикатора (с 41-го по 72-й) вместо постоянного свечения мерцает. Поэтому в окончательном варианте я поставил 3 вывода **COM**, при этом количество сегментов для одного общего вывода получилось 72/3=24, отсюда и соответствующая шина **S[1..24].**

Теперь нам осталось **Make Device** нашу модель, как и ранее с основными свойствами, т.е. **PRIMITIVE**, **MODDLL** и **TTRIGMIN** и начинаем добавлять сегменты. Поскольку их тут достаточно много, лучше это делать через заранее подготовленный текстовый файл. В папке вложения **8148_graphic** это файл **segments_8148.txt**. Поскольку у нас 72 светящихся элемента, получились 24 строки SEG, каждая из которых содержит записи для трех сегментов с разными координатами. Например, для **SEG1**, соответствующего в таблице сегментам **S1** и **S25** и **S49**, параметры будут выглядеть так

:

{SEG1=(3,400,200),(2,1800,200),(5,3900,200)}

Добавив все сегменты, проверяем себя с помощью теста. Во вложении **8148_graphic\Full_Test_Graphic.DSN** с помощью вспомогательной схемы этот процесс слегка «автоматизирован», чтобы можно было отследить последовательность зажигания сегментов в соответствии с таблицей. На этом этап подготовки графической модели индикатора окончен.

Теперь нам надо заняться моделью драйвера, имитирующего 2 последовательно соединенные **ML1001**. Для компоновки внутренней схемы двух-драйверного управления (назовем модель заранее **2xML1001**) я воспользуюсь модифицированными примитивами сдвиговых регистров и защелок. Мы уже ранее проводили такие вариации при моделировании АЦП (**п.6.16-6.18**). Здесь, существует еще более жесткое ограничение в 32 разряда. Но нам это не помеха, мне требуются примитивы на 24 разряда. Для большей компактности схемы дочернего листа я решил «ужать» входы/выходы разрядов в шины. В результате появились на свет следующие два примитива (Рис. 137).

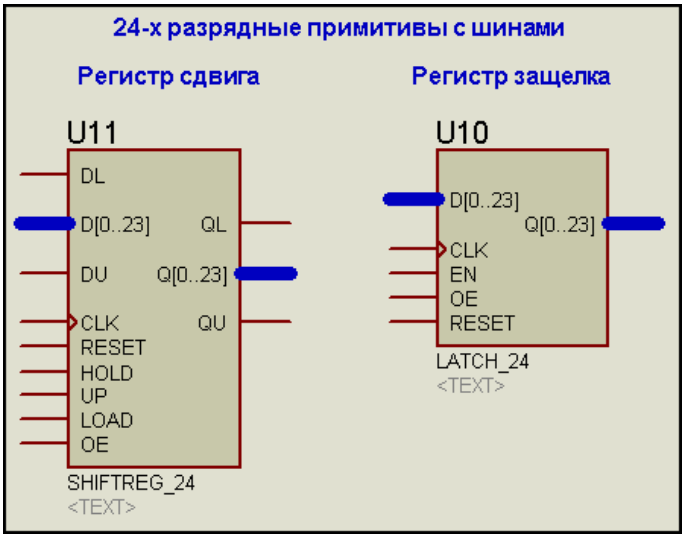


Рис. 137

Выглядят они более компактно, и схема дочернего листа будет иметь более опрятный вид. Надеюсь, что и полная картинка с дочернего листа в результате поместится непосредственно здесь. Теперь создаем графику самого сдвоенного драйвера. Тут тоже все просто. К изображению **ML1001** из предыдущего варианта у нас добавились только три управляющих выхода для выводов **COM1** и **COM2** дисплея (Рис. 138).

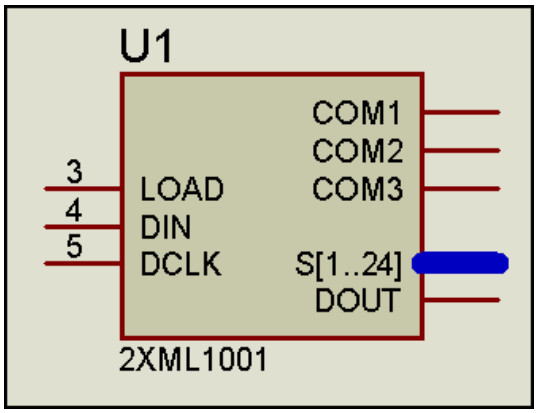


Рис. 138

Как обычно, присоединяем к нашей модели дочерний лист и с помощью вновь созданных примитивов формируем схему двух, соединенных последовательно драйверов (Рис 139). Эта структура отдельно приведена в папке вложения **2xML1001\Structure** проект **Int_Structure.DSN**.

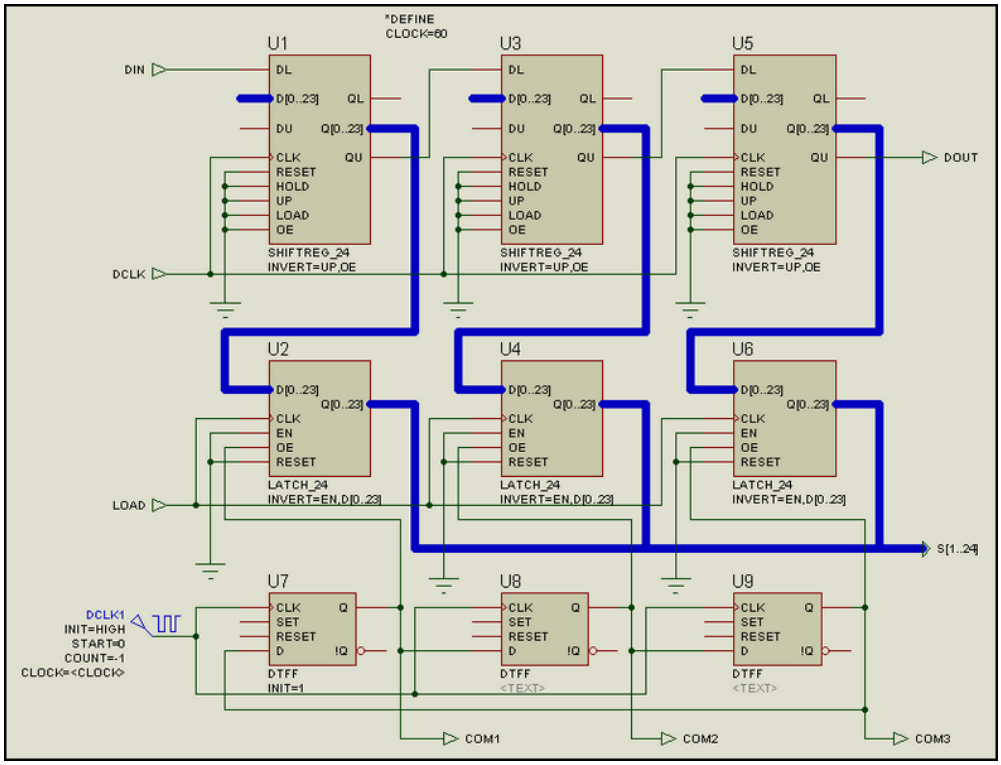


Рис. 139

Пожалуй, некоторые нюансы этой схемы заслуживают дополнительных пояснений. На регистрах сдвига **U1, U3, U5** организован последовательный сдвиговый регистр на 72 разряда. Чтобы не городить огород с завешиванием управляющих входов на шины с разными логическими уровнями, в свойствах входы **UP** и **OE** инвертированы – **INVERT=UP,OE**. Такой прием позволяет завешиванием всех управляющих входов примитива на землю сделать при этом инвертированные входы вечно активными. Аналогично в буферных регистрах защелках **U2, U4, U6** я поступил с входом **EN**. Так как выходные шины регистров сдвига соединены с входными защелок непосредственно, навешивание меток (лейблов) на них не требуется. Протеус и так поймет и соединит их «один в один», т.е. **Q0-D0**, **Q1-D1**, **Q3-D3** и т.д. Аналогично для выходов защелок и шинного терминала **S[1..24]** образуются пары **Q0-S1**, **Q1-S2**, но здесь есть одна хитрость, вернее две. Выходы защелок и шинный терминал должны иметь одинаковую разрядность, тогда образуются такие «сдвинутые» по нумерации пары. А еще выходы **U2, U4, U6** могут иметь третье (высокоимпедансное) состояние. Этим я воспользовался для коммутации выходов от трех регистров в общую шину. Напомню, что перевод в третье состояние осуществляется по входу **OE** (Output Enable). Вот его я и задействовал, чтобы на 24 выхода **S1…S24** подавать различные сигналы в зависимости от того, какой из выходов **COM** активен в данный момент. Еще нам потребуется какой-нибудь встроенный управляющий генератор. С этим тоже не проблема. Вспомните, как мы поступили при моделировании **К176ИЕ12**. Применяем тот же прием, ставим внутренний цифровой генератор **CLOCK**, а его параметры пропишем так, чтобы можно было задавать в свойствах модели. По умолчанию они прописаны в скрипте ***DEFINE** и равны 60 Герц. Этот генератор управляет трехразрядным кольцевым сдвиговым регистром на D-триггерах (**U7, U8, U9**), первый из них предустановлен в 1 (**INIT=1**). Этот регистр формирует выходные сигналы **COM**, а попутно и коммутирует буферные защелки. Тут я не стал мудрить и поставил обычные примитивы триггеров. Ну и еще пару слов о свойствах регистров защелок. Так как нам необходимо иметь активные нули для зажигания сегментов на выходах модели, инвертированы целиком входные шины защелок, т.е. в их свойствах стоит **INVERT=EN,D[0..23]** (про **EN** см. выше). В этом варианте можно было инвертировать только входы, т.к. у выходов используется третье состояние и если им задать INVERT, мы его потеряем. Чтобы исключить ложное «подмаргивание» сегментов в момент старта симуляции для защелок применена стартовая предустановка всех разрядов в единицы **INIT=0xFFFFFF**. Вот и все особенности внутренней структуры сдвоенного драйвера.

После тестирования (во вложении папка **With_Child_Sheet**) с дочернего листа этого проекта компилируем файл **2xML1001.MDF**. Тест окончательного варианта драйвера и готовый MDF лежат в папке **With_MDF_File**. Напомню, что в завершение работы над моделью драйвера необходимо прогнать **Make Device** для него еще раз и в свойствах на третьей вкладке прописать следующие:

MODFILE=2xML1001.MDF
CLOCK=60

Во вложении это уже сделано. Свойство **CLOCK** необходимо для того, чтобы можно было в **Edit Properties** сменить частоту внутреннего генератора, поэтому на третьей вкладке **Make Device** везде задаем ему **Normal**, ставим ограничение **Positive, None-Zero**, ну а в **Description** описываем его как-нибудь попонятнее, я, например, назвал **Internal Clock Generator**.

На этом я хочу завершить «показательные выступления» с **LCDMPX.DLL**. Охватить весь мировой ассортимент всевозможных LCD цифровых индикаторов, как вы понимаете, невозможно. Надеюсь, что приведенные примеры позволят вам при необходимости самостоятельно разработать требуемые для конкретного случая модели сегментных или мнемонических ЖК дисплеев. Первоначально я планировал сделать еще материал по **Holtek HT1611**, но, учитывая, что этот «раритет» практически исчез с рынка и если завалялся у кого, то только потому, что очень неудобен для использования в качестве индикатора – отсутствуют десятичные точки, я решил эту модель не делать. Если уж кому невмоготу, то по этой ссылке:

http://kazus.ru/forums/showpost.php?p=182152&postcount=518

лежит его модель от **Soir**. В свете новых знаний можете усовершенствовать этот вариант самостоятельно. Мы же переходим к моделям LCD дисплеев на основе контроллера **HD44780.**

К содержанию

8.14. LCDALFA.DLL – основа построения знакосинтезирующих дисплеев, базирующихся на контроллерах HD44780 и его клонах.

Пожалуй, дисплеи на базе **HD44780** – это наиболее востребованная и популярная у пользователей Протеуса линейка моделей ЖК индикации. На данный момент индикаторы этого типа выпускаются очень многими фирмами-производителями, как в «ближнем» зарубежье – КНР, так и в Европе, Америке и даже в России – МЭЛТ, а цены на наиболее простые, например, 8x2 без подсветки сопоставимы с ценами на сами микроконтроллеры, к которым они подключаются для Поэтому я решил уделить особенностям моделей на основе **LCDALFA.DLL** немного внимания, чтобы разобрать наиболее часто возникающие проблемы с симуляцией этих моделей, особенно у начинающих пользователей Протеуса.

Итак, все модели, базирующиеся на **LCDALFA.DLL**, расположены в двух библиотеках ISIS, а именно: **Optoelectronics\Alphanumeric LCDs** – типовые модели на базе **HD44780** и **Optoelectronics\Serial LCDs** – модели с последовательным вводом данных по одному проводу. Ну, что касается последних, то в России они практически не распространены, хотя любители экзотики могут заказать их через зарубежные Интернет-магазины. Достаточно набрать в любом поисковике фразу **Serial LCD Character Display** и вы получите массу ссылок на описание и предложения этих дисплеев, поскольку они достаточно популярны у западных разработчиков. Я же приведу здесь только одну ссылку: www.milinst.co.uk – сайт фирмы **Milford**, модели дисплеев которой и представлены в библиотеке **Serial LCDs**. А вот здесь: http://www.seetron.com/bpk000.html любители «экзотического секса» могут найти описание той самой дополнительно устанавливаемой платы **BPK** (от английского Backpack), которая вешается сзади на стандартный дисплей, чтобы он стал сериальным. И как это многочисленные наши «братья наши по разуму» до сих пор не содрали сей девайс, и не заполонили им наш рынок, особенно в свете развития проекта Arduino, где эти дисплеи очень популярны у западных разработчиков, ума не приложу. Но, хватит о грустном, вернемся к стандартным дисплеям на контроллере **HD44780**, которыми они нас снабжают в изрядном количестве. Наиболее популярная модель **LM016L** (2 строки по 16 символов) и окно ее свойств приведены на рисунке 140.

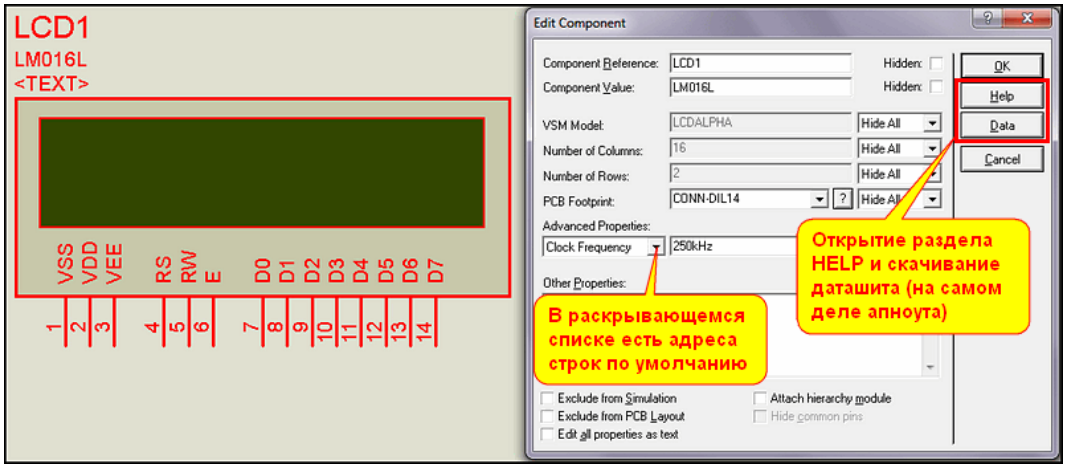


Рис. 140

Для начала немного о самой модели. Сразу же слегка «охолоним» начинающих, ножки **VSS**, **VDD** и **VEE** в данной модели приделаны просто, как недостающая часть табуретки. Модель прекрасно работает, даже если они просто висят в воздухе. Нет смысла, если конечно вы просто моделируете, а не разрабатываете печатную плату, в навешивании на них терминалов питания, каких либо батареек, выключателей и уж тем более регулирующих контрастность потенциометров. Кроме лишних «тормозов» при симуляции вы такими действиями ничего не получите, тем более, что регулировка контрастности в модели просто не реализована, ведь это чисто цифровая программная модель. Выводы **RS** (Register Selection), **RW** (Read/Write) и **E** (Enable), а также выводы 8-ми разрядной шины **D0…D7** в модели реализованы в полном объеме и используются при симуляции по прямому назначению в соответствии с даташитом.

Теперь заглянем в окно свойств модели на рисунке 140 справа. Здесь я хотел бы остановиться на двух моментах. В **Advanced Properties** по умолчанию фигурирует тактовая частота контроллера HD44780 **Clock Frequency=250kHz**. Это стандартная частота для конкретного контроллера от **Hitachi**, под которую и заданы все временные задержки модели в ISIS. В данном случае речь идет о внешней тактовой частоте обмена с контроллером, обозначенной в даташите как **External clock frequency**. Однако, не стоит забывать о том, какой именно контроллер стоит в том индикаторе, который вы собираетесь использовать. Так, в частности, не менее популярный клон **KS066U** от **Samsung** имеет стандартную **External Clock=270kHz**. Соответственно и выдержки времени при инициализации у него можно сделать несколько меньшими, хотя он и работает со стандартными для HD44780. Даташиты на наиболее распространенные **HD44780**, **KS066U** и **КБ1013ВГ6** есть во вложении. Последний, выпускаемый Зеленоградским НПО «Ангстрем», используется в популярных знакосинтезирующих ЖКИ **МЭЛТ**. Теперь поясню, почему я остановился подробно на этом моменте. Дело в том, что в моделях на основе **LCDALFA.DLL** заложен внутренний дебаггер (Рис. 141).

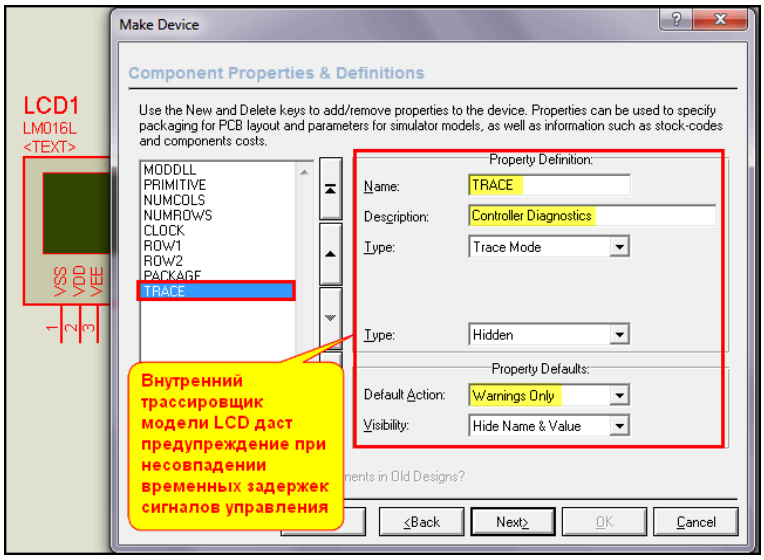


Рис. 141

В случае значительного отклонения во времени управляющих сигналов или отсутствия одного из таковых в программе инициализации дисплея в логе может появиться «горчичник», предупреждающий об этом, но далеко не всегда. Если с компиляторами с языков высокого уровня, где обычно присутствуют стандартные библиотеки для **HD44780**, такое случается очень редко, то использование в собственных разработках чужих ассемблерных программ инициализации ЖКИ, а уж тем более готовых HEX прошивок может надолго загнать вас в тупиковое положение. Причем, это совсем не значит, что прошивки не рабочие. Реальные дисплеи на основе данных контроллеров допускают некоторую вольность в обращении к ним. В результате программа, которая многократно повторялась в железе, работает в симуляторе криво, а иногда и вообще не работает. Но, сами понимаете, что написать такую математическую модель, которая будет автоматически предусматривать, что некто Вася или Петя решил подсократить выдержку при инициализации со стандартных для **HD44780** более чем 40ms до 12ms, потому что у него под рукой был русский МЭЛТ-овский дисплей практически невозможно, а уж тем более если часть команд инициализации вообще отсутсвует. К счастью, такие опусы легко распознаются в ISIS, даже при отсутствии исходника программы. До сих пор я скромно умалчивал о возможностях встроенного дебагера ISIS, но настала пора напомнить о нем, тем более, что именно в случае с ЖКИ может оказать существенную помощь.

Итак, сегодня в качестве «подопытного кролика» выступает разработка моего земляка А. Шарыпова почти десятилетней давности, которая, судя по многочисленным обсуждениям на

различных форумах, до сей поры не потеряла своей актуальности. Это опять частотомер и опять на **PIC16F84A**. Статья «Экономичный многофункциональный частотомер» была опубликована в октябрьском номере журнала «Радио» за 2002 год. Чтобы не искать, во вложении имеется DJVU вариант этой статьи. Прошивки с исходниками имеются на FTP сервере журнала, да и так их по сети разложено в изрядном количестве. Я буду рассматривать англоязычный вариант прошивки, но и в русском те же грабли. Собираем упрощенный вариант схемы и запускаем симуляцию с англоязычной прошивкой **counter.hex** (Рис. 142). Для сравнения в левом верхнем углу приведен вид дисплея из статьи. Как видим, вроде и работает, но нижняя строка полностью отсутствует. Причем, это наблюдается только в Протеусе, в железе вторая строка выводится. В чем же тут дело?

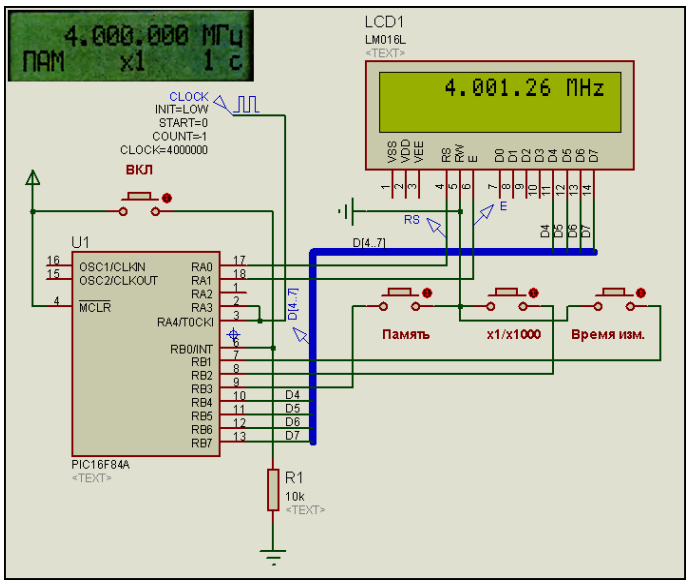


Рис. 142

Конечно, можно загнать наши сигналы обмена с контроллером дисплея в цифровой график и попытаться проанализировать по нему. Но есть более простой и наглядный метод. Переводим тот **TRACE**, что изображен на рисунке 141 из режима **Warning Only** (это те самые «горчичники»-предупреждения) в режим в режим **Debug** (отладка). Самый быстрый способ сделать это – щелкнуть правой кнопкой мышки по дисплею, выбрать **Configure Diagnostics** (фиолетовый паук) и перевести флажок для дисплея в нужное положение (Рис. 143).

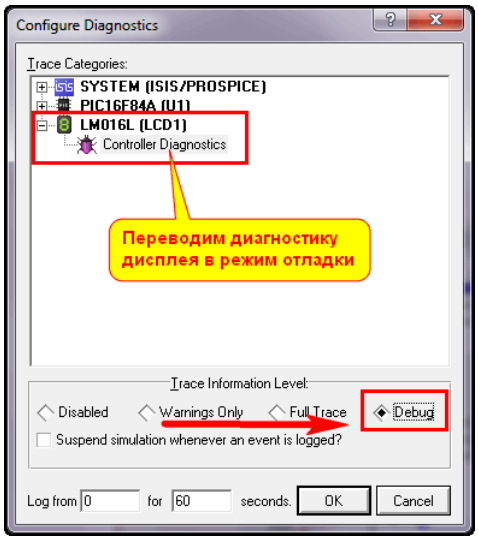


Рис. 143

После этого снова запускаем симуляцию и открываем окно **Simulation Log**, щелкнув левой кнопкой мышки справа от кнопок управления симуляцией. Результат приведен на рисунке 144. Те команды, которые относятся непосредственно к инициализации дисплея обведены рамкой. Не правда ли, скромный список по сравнению с тем, что приведен в даташитах на контроллеры для четырехбитного интерфейса. И как у нас может вывестись вторая строка, если двухстрочный режим даже не задан (**lines=1**). Самое интересное, что в железе это работает!!! Исправно выводится текст во вторую строку. Вот такой «запас прочности» у контроллера **HD44780**. Только вот вопрос – стоит

ли всегда надеяться на этот запас по любимому русскому принципу – «авось пронесет»? Вот и получается, что у автора работает, еще у 11 человек тоже, а тот пресловутый тринадцатый, у которого дисплей из «темного китайского подвала» остается в дураках.

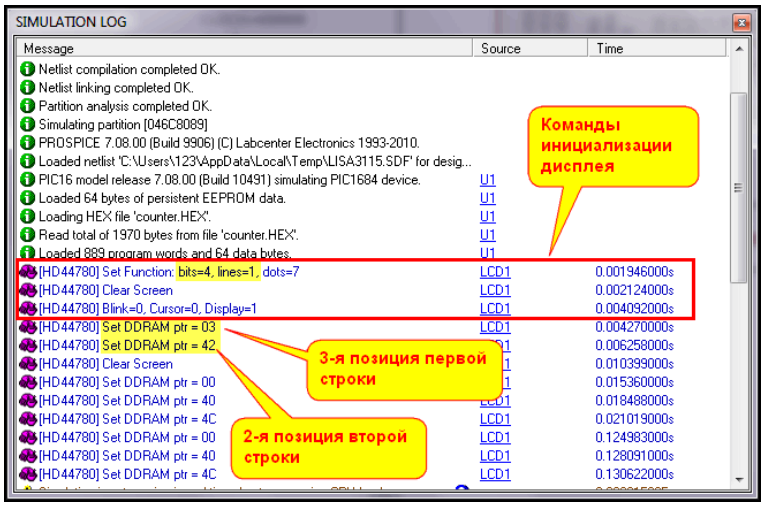


Рис. 144

Ну и законный вопрос – а можно ли в Протеусе увидеть нижнюю строку этого девайса? Конечно можно, но без кардинального вмешательства в программу – никак. На этот раз имеется исходник, будем надеяться, что я еще не совсем обленился и забыл ассемблер PIC. Мы не будем затрагивать основную часть программы, достаточно подкорректировать начальную инициализацию, а она выполняется единожды при подаче питания на девайс. Для начала заглянем в даташит. Я использовал русский от Ангстремовского контроллера, а красным цветом указал задержки из фирменного **HD44780** (Рис. 145).

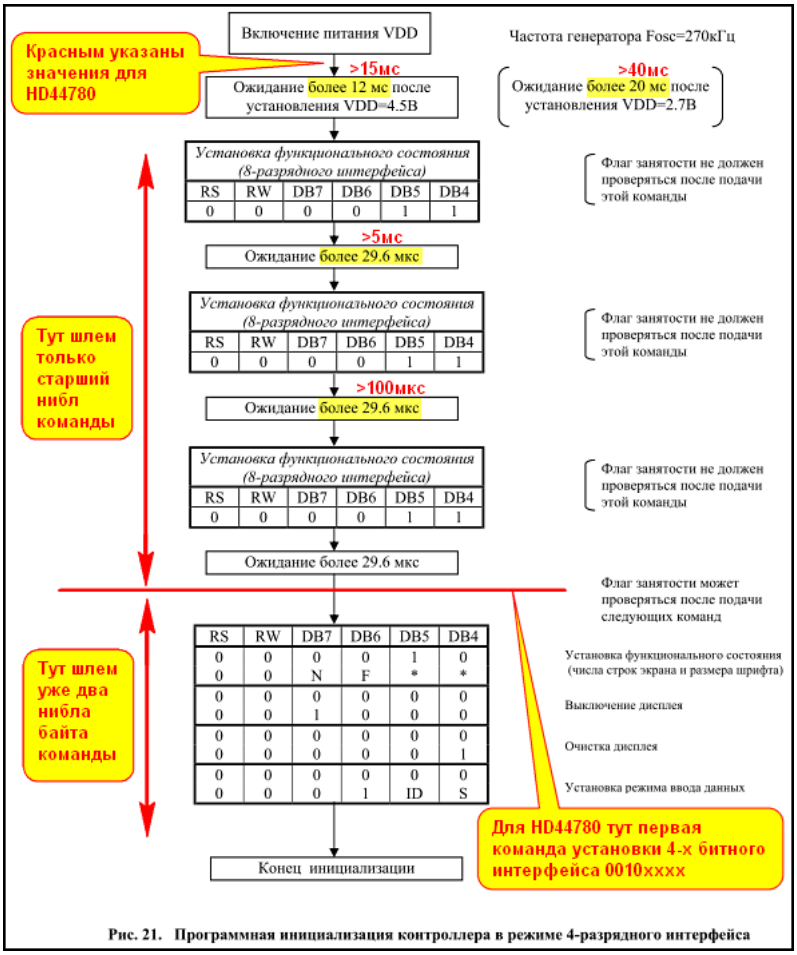


Рис. 145

Для нас значимыми в этой блок-схеме инициализации являются три задержки в 15 и 5 мс и в 100мкс. При подаче команд в 8-ми разрядном режиме. Напомню также, что команда передается при нулевом состоянии **RS** (**RW** у нас и так заземлен) и фиксируется положительным стробом на линии **E** контроллера ЖКИ. Программы инициализации для различных контроллеров несколько отличаются, но, поскольку модель в Протеусе сделана именно под **HD44780**, я буду руководствоваться его особенностями. Итак, покопавшись в ассемблерном листинге программы, и покрутив исходный вариант частотомера в **ISIS**, я обнаружил, что интервалы между стробами передачи команд и так достаточно большие около 90 мкс, что почти втрое превышает требуемые по даташиту. Начальную задержку в 15 мс я не стал делать, но желающие могут сформировать ее по аналогии с 5 мс из пачки следующих друг за другом **Pausem**. Для задержки в 5 мс используется существующая в программе **Pausem**. В авторском варианте она используется для формирования дополнительной задержки, необходимой, например, при подаче команды **Cursor Home** контроллеру ЖКИ. По даташиту она должна составлять 1,52 мс (в программе реально чуть больше – около 1,8 мс). Таким образом, три подряд вызова подпрограммы этой задержки дали мне требуемую около 5,4 мс. Несколько сложнее увеличено время между второй и третьей командой 8-ми разрядного интерфейса. В авторской программе имеется подпрограмма посылки строб-импульса по линии E - **Enbl**. Саму подпрограмму я использовал для передачи старшего нибла команд 8-ми разрядного интерфейса, а часть ее, поставив дополнительную метку **Pause25**, как подпрограмму задержки. В результате перед авторской инициализацией дисплея добавлен следующий ассемблерный код:

```
; ********** ДОБАВЛЕННЫЙ КОД ИНИЦИАЛИЗАЦИИ ЖКИ **********
      movlw   0x30       ; Первая команда установки 8-ми битного интерфейса
      movwf          PortB       ; загружаем команду в порт B
      call  Enbl       ; Отсылаем старший нибл в ЖКИ
      call  Pausem      ; Три задержки Pausem в сумме ~5,5 мсек
      call  Pausem
      call  Pausem
; Порт B уже содержит команду 0x30 и не менялся
      call  Enbl       ; Вторая установка 8-ми битного интерфейса
      call  Pause25     ; В дополнительную задержку, чтобы  было >100 мксек
; Порт B по прежнему содержит команду 0x30 и не менялся
      call  Enbl       ; Третья установка 8-ми битного интерфейса
      movlw   0x20       ; Первая команда установки 4-ми битного интерфейса
      movwf          PortB       ; загружаем команду в порт B
      call  Enbl       ; Отсылаем старший нибл в ЖКИ
; Теперь мы уже в четырехбитном режиме и посылаем следующие команды
; через подпрограмму LEDcom двумя ниблами
      movlw   0x28      ; Установки 4 битн. интерфейс 2 строки 5x7 точек
      call  LEDcom
```

Надеюсь, тем, кто занимается программированием PIC-контроллеров на ассемблере, все в нем будет понятно из комментариев. Я постарался внести минимальные изменения в авторский вариант, только с целью добиться полной работоспособности частотомера в ISIS. Конечно, если посидеть подольше над программой можно было сделать и поопрятней и покороче, но я не стал с этим заморачиваться, конечный результат достигнут (Рис. 146).

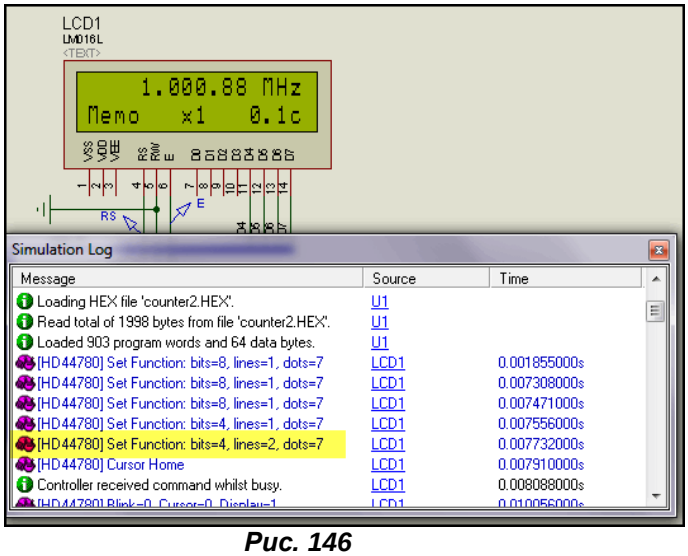


Рис. 146

Как видим, теперь начальная инициализация в логе стала выглядеть более приближенной к даташиту, а на индикаторе стала отображаться «утерянная» строка. Примеры данного частотомера

лежат в папке вложения **Sharypov** соответственно исходный вариант – папка **Eng_var** и откорректированный – **Eng_var_correct**. О том как использовать русскоязычный вариант в Протеусе чуть ниже.

В ходе поисков в интернете примеров для данной темы я натолкнулся на сайте Радиокота еще на один частотомер с ЖКИ на контроллере **PIC16F628A**. Поскольку там проблема индикации в Протеусе несколько другого плана, давайте рассмотрим и этот вариант.

Вот ссылка на саму тему у Кота: http://www.radiokot.ru/circuit/digital/measure/19/ , а здесь: http://www.radiokot.ru/forum/viewtopic.php?t=12555 страницы форума, посвященные обсуждению этой темы. Естественно, это все первоисточники, но данная схема уже благополучно расползлась по всемирной паутине и вы можете натолкнуться на нее и в других местах. Автор данного девайса тоже не особенно заморачивался, а просто переделал импортную программу, сделанную под более крутой контроллер на дешевый **PIC16F628A**. И конечно же он, и не только он, самостоятельно попробовали симуляцию программы в Протеусе. И тоже поймались на простейшей ошибке. Программа сделана изначально под однострочный LCD 16 символов. Если попытаться воспроизвести ее на однострочной модели LCD в ISIS, то половина информации будет отсутствовать, а на модели 16x2 эта информация появиться во второй строке. Тут все дело в некоторой путанице с адресацией для модели 16x1. Этот вариант дисплея нечто вроде «Оки» в линейке АвтоВАЗа, которую «любовно» именовали выкидышем. Дело в том, что это единственный производимый ЖКИ, у которого адресация в строке сделана не подряд, т.е. с первой по восьмую позицию адреса DD RAM **00-07**, а с девятой по шестнадцатую – **40-47**. Если посмотреть на адресацию двухстрочных ЖКИ, то последние совпадают с первыми восьмью второй строки. Отсюда и появляется информация на двухстрочном дисплее. Но здесь уже промашка допущена непосредственно в модели Протеуса и правится она в течение нескольких секунд. Кстати, на адресации в моделях ЖКИ на основе **LCDALFA.DLL** следует остановиться несколько подробнее, чтобы в дальнейшем она вас не смущала. Так, например, в даташитах популярных двухстрочных дисплеев 16x2 указаны адреса для первой строки – **00-0F**, для второй – **40-4F**. Для соответствующей модели **LM016L** в свойствах прописаны адреса **80-8F** и **C0-CF** соответственно (Рис. 147). В чем тут дело? Если внимательно посмотреть в даташит контроллера **HD44780**, то можно заметить, что установка старшего бита в единицу является признаком обращения к DD RAM, т.е. области памяти знакомест контроллера. Отнимите его из указанных для модели **LM016L** адресов и получится стандартная адресация. Теперь вернемся к однострочному ЖКИ на 16 символов, т.е. модели **LM020L**. В его свойствах для строки **Row 1** прописаны адреса **80-8F**, ну или в переводе на обычные – **00-0F** (подряд!!!). Это уже явный ляпсус Лабцентра, но помочь этому несложно, если заглянуть в Help модели. А там аглицким языком написано, что если адреса в строке идут не подряд, то они записываются диапазонами через пробел. Берем на вооружение данный факт и исправляем в свойствах следующим образом: **80-87 C0-C7**. Будьте внимательны! Если указанные диапазоны не совпадут с общим количеством символов в строке, ISIS при запуске симуляции пошлет вас в «эротическое путешествие» фразой красного цвета. В папке **RadioKot** вложения пример частотомера с 16-ти символьным однострочным ЖКИ и реальной индикацией частоты.

Ну и нам осталось разобраться только с модификацией **LCDALFA** под кириллицу. Изначально в библиотеке заложена таблица с европейским вариантом знакогенератора, а там, как видно из рисунка 147, русский алфавит отсутствует (эта часть таблицы выделена желтым).

Рис. 147

Таблица знакогенератора хранится в файле **LCDALFA.DLL** в виде точечного BMP рисунка размером 104x176 точек и извлечь ее оттуда и заменить на другую соответствующего размера можно любым редактором ресурсов (Restorator, PE Explorer, ResHacker и т.п.), а отредактировать даже в родном виндозном Paint. Но удобнее это сделать с помощью специализированной утилиты **charset.exe**, которая была написана нашим соотечественником, который присутствует на форуме под ником **Тень** и в данное время является штатным сотрудником **Labcenter Electronics**. Утилита довольно компактная и находится во вложении, а ее окошко приведено на рисунке 148. Пользоваться ей очень просто, с помощью кнопки открыть файл (3) открываете исходную **LCDALFA.DLL**, если есть необходимость – сохраняете исходный рисунок BMP с помощью кнопки (2). Затем подгружаете картинку с русским шрифтом с помощью кнопки (1) и сохраняете русифицированную DLL с помощью самой правой кнопки (4).

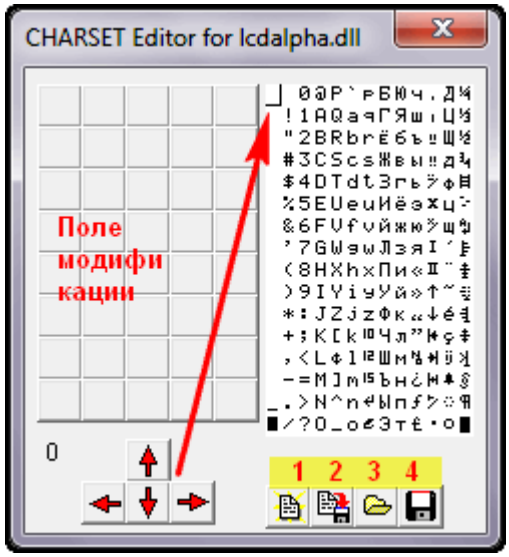


Рис. 148

На рисунке в окне программы уже загруженная таблица с русскими символами. Кроме того, можно модифицировать любой символ по своему усмотрению, выбрав его с помощью курсора (показан в левом верхнем углу таблицы). Курсор перемещается с помощью кнопок со стрелками, а выбранный символ отображается в поле модификации. Закрашенные пиксели в нем выглядят утопленными. На днях я проверил успешную работу утилиты на версии Протеуса 7.8. Единственный нюанс, под Windows 7 мне пришлось запустить ее от имени администратора. Ну и еще одна рекомендация – лучше проделать эту операцию в отдельной папке, а модифицированную DLL сохранить в папке **MODELS** под измененным именем, например, **LCDALFARUS.DLL**. Тогда можно использовать и тот и другой варианты, либо просто изменив **MODDLL** в свойствах модели LCD текущего проекта в режиме **Edit ...as text**, либо продублировав все имеющиеся модели на базе HD44780 с добавкой RUS на конце и заменив для этих вариантов значение **MODDLL** на **LCDALFARUS.DLL**. В последнем случае, уже добавляя модель LCD в проект, выбираем нужный, например, **LM016** – западный, **LM016RUS** - с кириллицей. Готовый вариант **LCDALFARUS.DLL** для версии 7.8 во вложении.

На чем еще хотелось бы заострить ваше внимание. Некоторых смущает, что отсутствуют «хвосты» у символов русского шрифта. Например, если вы будете выводить на дисплей слово «Щука», то увидите «Шука». Тут просто уместно вспомнить, что стандартный вывод для дисплея по умолчанию 5x8 точек (при инициализации в 4-х битном режиме это команда **0x28**). Переведите дисплей при инициализации в режим 5x10 (заменив командой **0x2C**) и «хвост отрастет».

Из замеченных «глюков» для моделей на основе **LCDALFA** – это неадекватный по сравнению с даташитом сдвиг экрана аппаратной командой. Так что если планируете использовать дисплей для организации бегущей строки, то лучше это сделать программно или отлаживать сразу в железе, чтоб не напрягать себе мозги неожиданными «заморочками».

Ну, и напоследок немного о модификации. Нет, например, в библиотеках самой дешевой модели 2x8, а хочется. Все просто, разбиваем исходную **LM016**. Для графики есть одна особенность – маркер **ORIGIN** находится в центре, поэтому сдвигать (подрезать) экран и базу надо будет с двух сторон (Рис. 149).

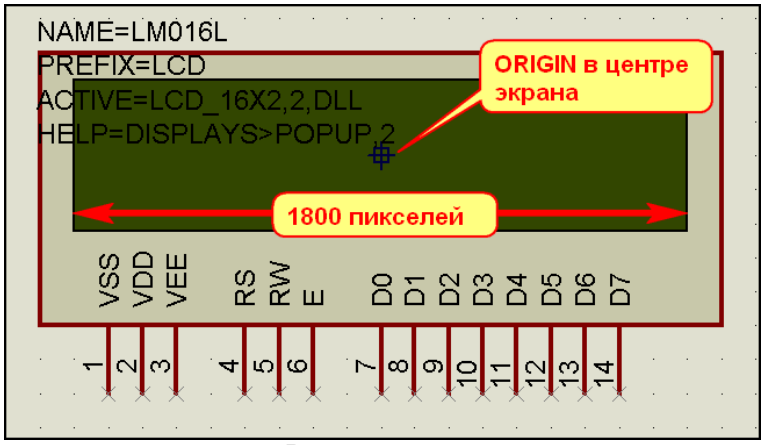


Рис. 149

Как ставить «фальшивку» маркера, вы уже знаете, и нетрудно определить, что ширина внутреннего зеленого экрана составляет 1800 пикселей (16 символов плюс два зазора по краям шириною как символ). Дальнейшее – арифметика для первоклассника. Изменить ширину придется как для основного изображения модели, так и символов потушенного и светящегося экрана, задав им соответствующие имена. Ну а при последующем **Make Device** необходимо на первой вкладке задать эти имена символов, а на третьей подкорректировать значение **NUMCOLS** (количество колонок) с 16 на 8 и задать новые конечные адреса для первой **ROW1** и второй **ROW2** строк. Кстати, можно и вообще не править графику, а поправить только адреса и количество колонок, ну и, конечно же, имя модели, но при этом пустые зазоры по краям экрана у вас будут широкими. Мы уже так часто проделывали подобные операции, что я пожалуй предоставлю вам самостоятельно поэкспериментировать с данными моделями, чтобы не надоедать лишними нудными подробностями. Дальше пойдет более интересный материал по моделям графических дисплеев, вот там и остановимся на тонкостях, тем более, что приемы редактирования будут похожими, а материал более актуален. Символьные же модели постепенно сдают позиции более популярным графическим ЖК индикаторам.

К содержанию

8.15. Обзор моделей контроллеров графических LCD, входящих в LCDPIXEL.DLL и моделей графических дисплеев на их основе. Особенности графических моделей и некоторые специфические параметры общие для всех моделей графических LCD.

Аналогично тому, как **LCDALFA** используется для создания знакосинтезирующих дисплеев на основе HD4470, библиотека **LCDPIXEL.DLL** служит основой для всех графических дисплеев, которые представлены в ISIS. Однако она объединяет функции не одного конкретного контроллера, а сразу нескольких контроллеров LCD с восьмиразрядной шиной данных. Рассмотрим кратко, какие контроллеры реализуются с помощью этой библиотеки.

T6963C (***Toshiba***) – один из самых «древних» контроллеров LCD, с помощью которого реализуются индикаторы с разрешением до 240x128 пикселей. Хотя данному контроллеру уже полтора десятка лет, дисплеи на его основе до сих пор выпускаются некоторыми производителями. В частности, на клоне этого контроллера **RA6993** тайваньской фирмы ***RAiO TECHNOLOGY INC*** www.raio.com.tw реализован ряд популярных у нас графических индикаторов фирмы ***Winstar*** http://www.winstar.com.tw/products_detail.php?CID=18&lang=ru серий WG и WB. Модель **T6963C** в ISIS не представлена в графическом виде в библиотеке **Optoelectronics/LCD Controllers**, так что о наименовании выводов и параметрах, задаваемых этой модели, можно судить только по представленным в библиотеках реальным моделям. На основе модели этого контроллера выполнены все модели ***Densitron***-овских индикаторов, начинающиеся с **LM** в библиотеке **Graphical LCDs**. Для них доступно скачивание англоязычных даташитов при нажатии кнопки **Data** в окне **Edit Properties**, поэтому подробно останавливаться на этих моделях я не стану. Английские даташит на **T6963C** и **Aplication Note** по использованию во вложении в папке Даташиты, а тех кто «хромает» в английском отправлю на http://www.gaw.ru/html.cgi/txt/lcd/chips/t6963/ , где в онлайн представлен русский перевод.

KS0108 (***Samsung***) – это тоже контроллер «долгожитель», в частности, документация самого Самсунга датируется 1997 годом. Несмотря на это, дисплеи с использованием данного драйвера выпускаются и в настоящее время, в том числе и Российской компанией МЭЛТ http://www.melt.com.ru/ , которая для этих дисплеев использует аналог **KS0108** производства ОАО «АНГСТРЕМ» – **К145ВГ10**. БИС драйвера LCD **KS0108** имеет встроенное ОЗУ матрицы и 64-канальный выход, который обычно используется для управления столбцами матрицы. Совместно с

KS0108, как правило, применяется дополнительно драйвер строк **KS0107**, который может управлять 64-мя строками матрицы. Таким образом, одна такая «сладкая парочка» способна обслуживать матрицу 64x64 точки, но чаще встречается комбинация из двух спаренных **KS0108** с разделенными выводами **CS**, которая применяется в дисплеях 128x64 пикселя. Это прародитель многочисленных клонов индикатор **AG-12864A** от **AMPIRE**, модель которого представлена в ISIS и называется **AMPIRE128X64**. Аналогичная модель, имеющая не инвертированные входы **CS**, представлена как **LGM12641BS1R**. Обе эти модели в библиотеках Протеуса отображаются как **Schematic**, поскольку для них имеется скомпилированный **KS0108X2.MDF** в котором и выполнено объединение двух контроллеров **KS0108**. Файл можно извлечь из **DISPLAY.LML** с помощью утилиты **GETMDF.EXE**. Об этом я рассказывал ранее. Модели контроллера **KS0108** и тех, о которых речь пойдет ниже, для создания подсхем имеются в библиотеке **Optoelectronics/LCD Controllers.** Англоязычный даташит на контроллер **KS0108** есть во вложении, а здесь: http://www.gaw.ru/html.cgi/txt/lcd/chips/ks0108b/index.htm находится русское описание в онлайн.

SED1520 (***Seiko Epson***) – это японское творение тоже благополучно здравствует с 1998 года. Один драйвер **SED1520** способен управлять матрицей 61x16 пикселей или двухстрочным знакосинтезирующим дисплеем формата 12x2, где знакоместо составляет 5x8 точек. На данный момент производителями дисплеев наиболее часто используется тандем из двух контроллеров **SED1520**, который обеспечивает управление матрицей графического LCD индикатора 122x32 точки. При этом память контроллера подразделяется на 4 страницы, обращение к которым происходит в зависимости от комбинации битов **D0**, **D1** в соответствующей команде **Set Page Address**. В библиотеках Протеуса представлен целый ряд моделей на основе данного драйвера: **AGM1232G *(AZ DISPLAYS INC)***, **EW12A03GLY *(EMERGING DISPLAY),*** **HDM32GS12-B** и **HDM32GS12Y-3 *(оба HANTRONIX)***. Даташиты на эти экзотические для России LCD доступны для скачивания при нажатии соответствующих кнопок в свойствах моделей. Для нас же больший интерес представляют дисплеи на основе **SED1520**, которые представлены на отечественном рынке электронных компонентов. В первую очередь это все та же компания МЭЛТ, которая выпускает графические дисплеи формата 122x32 на основе Ангстремовского клона этого драйвера – **КБ145ВГ4**. И он не единственный. Не менее распространен японский клон драйвера – **NJU6450 *(JRC*** http://www.njr.co.jp/english/index.html ***)***, а тот же **Winstar** выпускает индикаторы на базе тайваньского клона драйвера - **SBN1661G *(Avant Electronics*** http://www.avantsemi.com.tw ***)***. Чтобы не раздувать вложение даташитами до непомерных размеров, приведу только ссылку на русскоязычное описание **SED1520**: http://www.gaw.ru/html.cgi/txt/lcd/chips/sed1520/index.htm и оригинальный даташит от EPSON. Даташиты на клоны желающие могут скачать с сайтов производителей самостоятельно. В последний момент вспомнил про Минское **ОАО «ИНТЕГАЛ»**, которое также выпускает клон **NJU6450**, именуемый **IZ6450**, он по размеру не очень большой и на русском, поэтому добавлю во вложение.

SED1565 (***Seiko Epson***) – это еще один драйвер японской компании, модель которого представлена в ISIS. Контроллер обеспечивает управление матрицей до 65x132 и позволяет работать как с восьмиразрядной параллельной шиной данных, так и с последовательной загрузкой данных. В последнем случае в качестве входа данных **SI** используется пин **D7**, а в качестве входа тактового сигнала **SCL** пин **D6**. В Протеусе на основе драйвера **SED1565** реализован ряд моделей дисплеев формата 128x64 точек. Модели **HDG12864F-1 *(Hantronix)*** и **NOKIA7110** (96x65 точек) выполнены с последовательным интерфейсом. Для всех моделей дисплеев на базе **SED1565** возможно скачивание даташитов при нажатии соответствующей кнопки **DATA** в окне свойств. В данный момент почти нереально приобрести индикатор с контроллером данного типа, так как он не получил широкого распространения в отличие от предыдущих драйверов. Разве что кому то посчастливиться откопать такой раритет, как Нокия 7110 с исправным дисплеем. Соответственно и русского перевода документации по **SED1565** не существует, а английский вариант находится во вложении.

Теперь рассмотрим некоторые особенности построения моделей графических дисплеев, которые могут пригодиться нам в дальнейшем. Для начала немного о графике. Все дисплеи содержат серую подложку с выводами, экран и два символа экрана – погашенный (**_0** в конце имени) и с подсветкой (**_1** в конце имени) (Рис. 150). Маркер ORIGIN общего графического изображения и символов установлен в верхнем левом углу экрана, который у имеющихся в ISIS моделей выполнен либо в зеленых, либо в синих тонах.

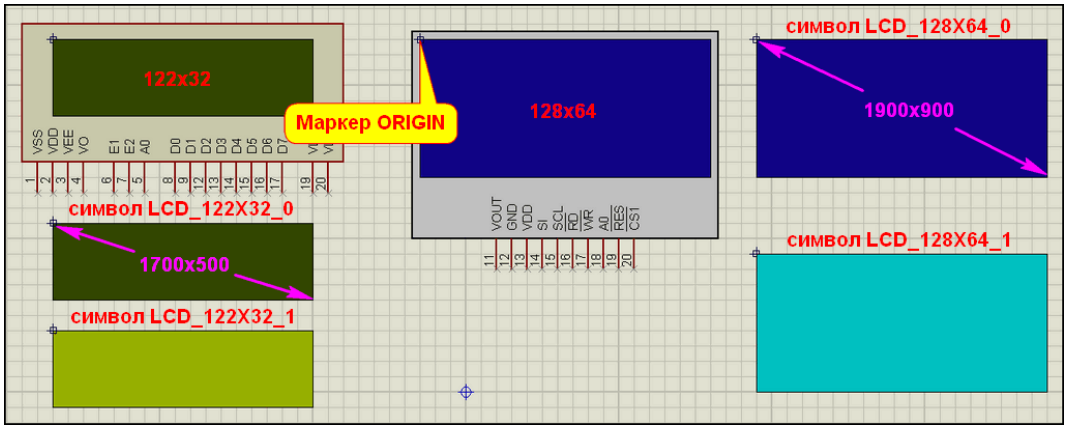


Рис. 150

Рабочую область светящегося экрана **LCDPIXEL.DLL** автоматически пропорционально делит на установленное разрешение, оставляя при этом небольшую незанятую окантовку. К примеру, на рисунке 150 слева приведен пример индикатора **EW12A03GLY** с форматом 122x32 точки. Символы погашенного **LCD_122X32_0** и подсвеченного **LCD_122X32_1** экранов составляют 1700x500 пикселей. Я «разобрал» (**Decompose**) эту модель, пропорционально увеличил вдвое ширину и длину символов, сохранив их с другим именем и конечно же увеличил и саму графическую модель. После этого сохранил ее с другим именем и подгрузил в типовой проект из примеров Протеуса **EW12A03GLY.DSN** из папки **SAMPLES\VSM for AVR\AVR and SED1520** версии 7.8. Что из этого получилось, видно на рисунке 151. Скриншот сделан в момент инверсного вывода, когда фон экрана воспроизводится черным. Здесь видна и нерабочая окантовка и рабочее поле экрана и самое главное – достигнутый эффект. Слева стандартная модель из библиотеки ISIS, справа моя модификация для себя-любимого, «слабовидящего и глухослышащего». Аналогично можно и уменьшить рабочую область дисплея. Пример из Протеуса с моими модификациями в папке **Sample_ModDLL** вложения. Надеюсь, общая концепция создания графики дисплеев ясна, пойдем дальше.

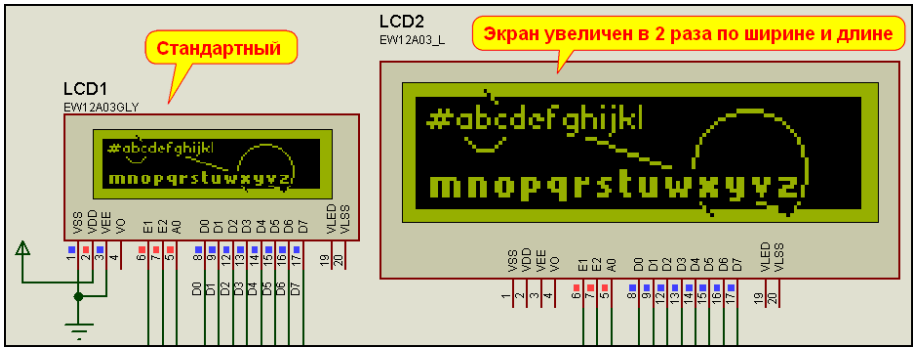


Рис. 151

Теперь немного разберемся с назначаемыми свойствами и параметрами.

MODDLL – **VSM Model DLL**. Ну, наверное, многие уже догадались, что на третьей вкладке **Make Device** этому свойству для всех графических дисплеев присвоено имя библиотеки **LCDPIXEL.DLL**. Кроме того, не забывайте на первой вкладке включать галочку **Link to DLL**, устанавливать количество символов **2** и прописывать основную часть имени символов погашенного и подсвеченного экрана для вашей модели.

MODFILE - **LISA Model File**. Этому свойству назначается MDF файл вашей модели, скомпилированный со схемы дочернего листа. **LCDPIXEL.DLL** поддерживает передачу параметров из MDF, что позволяет создавать модели с двумя и более контроллерами дисплеев. Именно поэтому большинство моделей позиционируются как **Schematic**. Т.е. на дочернем листе вы создаете внутреннюю структуру модели с использованием моделей контроллеров из библиотеки **LCD Controllers**, а также можете имитировать аналоговую часть, например нагрузку подсветки дисплея с помощью аналоговых примитивов. Мы этим займемся практически в следующем разделе.

WIDTH и **HEIGHT** – соответственно ширина и высота рабочего поля экрана индикатора в пикселях. Например, для модели **EW12A03GLY** они равны соответственно **122** и **32**. Когда я менял графику в примере чуть выше, я оставлял эти значения неизменными и симулятор пропорционально увеличивал или уменьшал размеры видимых точек в зависимости от размеров символов экрана моделей. Иными словами, они как бы задают симулятору коэффициент, на который надо поделить размер рабочей области модели, чтобы получить размер точки индикатора в реальных пикселях

экрана дисплея компьютера. (Вот это я загнул фразочку!!! Но, если пару раз перечитать, вроде понятно.)

TRACE - **Controller Diagnostics** (диагностические сообщения контроллера); **TRACE_CWR** - **Command Write Events** (сообщения при записи команд); **TRACE_MWR** - **Data Write Events** и **TRACE_MRD** - **Data Read Events** (соответственно сообщения при записи и чтении данных). Эти параметры относятся к диагностике и отладке модели и по умолчанию носят значения **Warning Only** (только предупреждения). С **TRACE** мы уже сталкивались при рассмотрении моделей на **HD44780**. Для моделей графических дисплеев добавлены еще три отладочных параметра, которые также можно перевести в режим **DEBUG** и использовать для вывода в лог симуляции дополнительных сообщений о процессе обмена с контроллером дисплея.

Конечно, у некоторых дисплеев есть и дополнительные параметры, назначение которых становится понятным только после детального изучения соответствующих даташитов. Рассматривать здесь каждый досконально я не имею ни времени, ни возможностей, поэтому на этом краткий обзор моделей завершается. А мы перейдем к практическому примеру и попробуем создать модель нашего отечественного графического дисплея, выпускаемого фирмой МЭЛТ на базе контроллера **SED1520**.

К содержанию

8.16. Графические LCD на основе контроллера SED1520 в ISIS и их особенности. Моделируем отечественные дисплеи МЭЛТ в Протеусе.

Незавершенный вариант, который позже будет дополнен практикой.

Выбор практического примера на **SED1520** выпал не случайно. Во-первых об этом просил один из участников форума, а во-вторых графические дисплеи на основе этого драйвера и его клонов имеют самую низкую стоимость и вполне доступны для приобретения. Конечно, существуют еще дисплеи от сотовых телефонов, которые можно купить еще дешевле, а иногда и просто выдернуть из старого ненужного телефона. Но в этом случае отлаживать устройство придется только в железе. А пайка шлейфов к дисплеям от сотовых, в которых обычно используются миниатюрные нестандартные разъемы, у начинающих может вызвать затруднения. Исключение составляет дисплей **Nokia 3310**, для которого модель Протеуса существует в природе, но об этом мы поговорим позже. А пока рассмотрим модель драйвера **SED1520** и как мы можем адаптировать ее под свои нужды. Все модели драйверов дисплеев находятся в библиотеке **Optoelectronics\LCD Controllers**. Конкретно модель **SED1520** и окно ее свойств показаны на рис 152. Даташит **SED1520** доступен для скачивания при наличии подключения к Интернет и нажатии соответствующей кнопки.

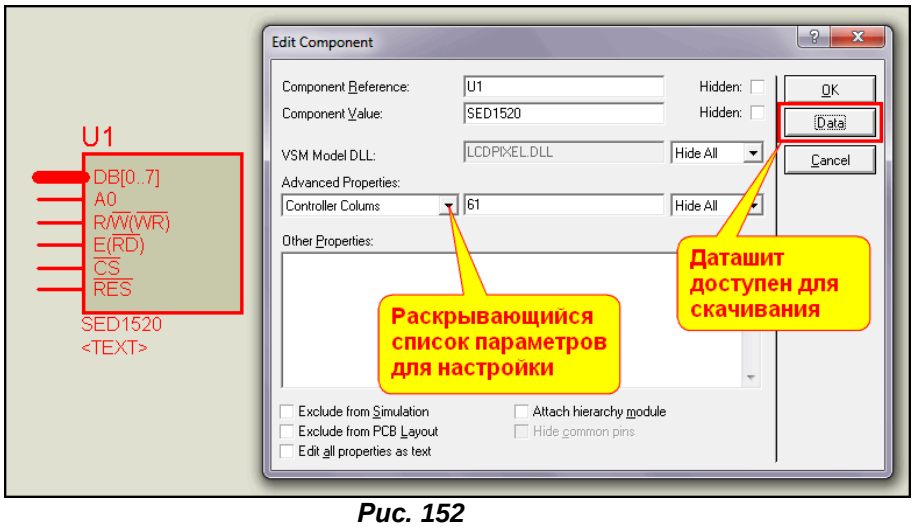


Рис. 152

Как видим, у модели имеются только выводы входов, а выходные сигналы колонок (**SEG0…SEG60**) и строк (**COM0…COM15**) уже жестко прописаны в программной модели Протеуса. Это и вызовет некоторое затруднение при моделировании **MT-12232A** (МЭЛТ), но оно вполне преодолимо с некоторыми условностями. Для начала немного информации о выводах модели из даташита. Сразу же напомню, что **SED1520** может работать в двух режимах: интерфейс с контроллером серии 68xx или интерфейс с контроллером серии 80xx. Нас интересует первый, поскольку он принят стандартом де-факто и для других микроконтроллеров (AVR, PIC). Для наглядности ниже синим цветом указано то, что относится к 68xx, а зеленым то, что относится к 80xx.

- **DB[0..7]** – двунаправленная восьмиразрядная шина команд/данных.
- **A0** – вход драйвера, определяющий что в данный момент передается по шине данных: **A0=0** – команда, **A0=1** – данные для вывода на индикатор.

- **R/W(WR)** – вход драйвера. Для 68xx определяет чтение (R/W=1) из **SED1520** или запись (R/W=0) в него. Для 80xx WR=0 при записи сигналы на шине данных стробируются положительным перепадом (передним фронтом) 0=>1 импульса на этом выводе.
- **E(RD)** – вход драйвера. Для 68xx определяет тактирование (выбор) данного драйвера. *(По E=1 производится запись/чтение в конкретный кристалл – это очень важный для нас сигнал.)* Для 80xx RD=0 означает, что шина D0…D7 **SED1520** направлена на вывод данных.
- **CS** – вход драйвера. Обычно CS=0. Эффективен при использовании внешнего генератора тактовой частоты.
- **RES** – вход сброса драйвера. Перепад сигнала на этом входе производит сброс микросхемы драйвера и установку для нее определенного интерфейса. Если был RES=0 и произошел переход 0=>1, то происходит сброс и устанавливается интерфейс 68xx. Если был RES=1 и произошел переход 1=>0, то происходит сброс и устанавливается интерфейс 80xx. *(Это тоже очень важный для нас сигнал.)*

Теперь заглянем в свойства модели **SED1520**, а конкретно в раскрывающийся список. С некоторыми параметрами оттуда мы уже знакомы, но в стандартном окне они не видны и мы сможем их увидеть только с установленной галочкой **Edit all properties as text** или в режиме **Make Device** на третьей вкладке. Итак, сначала о тех, что видны в раскрывающемся списке:

- **Controller Colums** *(да-да, именно **Colums**, а не **Columns** – и у англичан бывают грамматические ошибки)* – **CONTRWIDTH** по умолчанию равно 61 – количество колонок (**SEG**) в контроллере.
- **Controller Lines** **CONTRHEIGHT** по умолчанию равно 16 – количество строк (**COM**) для модели контроллера.
- **Controller Display X Offset** – **BMPXOFF** смещение картинки (на экране) по горизонтальной оси X. По умолчанию нулевое.
- **Controller Display Y Offset** – **BMPXOFF** смещение картинки (на экране) по вертикальной оси Y. По умолчанию нулевое.
- **Segments Output Direction** – **ADCMODE** направление вывода из памяти драйвера на экран. По умолчанию 0 – прямое, слева направо. При установке в 1 картинка выводится в обратном направлении справа-налево. Это вполне реальный параметр реального контроллера из даташита. Как мы увидим позже, именно с ним будут «заморочки» в МЭЛТ-овских индикаторах.

Остальные параметры имеют свойство **Hidden** (скрытые) и видны только при установке флажка **Edit all properties as text**, но среди них есть важные для нас и я их тоже опишу. В первую очередь это уже знакомые нам **WIDTH** (высота) и **HEIGHT** (ширина) но теперь уже экрана дисплея в пикселях. По умолчанию они соответственно 16 и 61. Также знакомы нам параметры трассировки: **TRACE**, **TRACE_CWR**, **TRACE_MWR** и **TRACE_MRD**. Все они по умолчанию **Warning Only** – режим предупреждений. Свойство **PRIMITIVE** для данной модели имеет значение **DIGITAL,SED1520**. Свойство **MODDLL** задано как **LCDPIXEL.DLL**. Надеюсь, эти свойства не нуждаются в особых комментариях. А вот со следующими двумя мы еще не встречались, поэтому коснусь их подробнее.

- **CTRLID** – идентификатор контроллера. По умолчанию равен **0x100**. Если в модели индикатора используются два или более контроллера, эти значения должны отличаться. Так у нас и будет в наших моделях – второму этот параметр мы присвоим как **0x101**.
- **RAMSIZE** – объем внутренней памяти драйвера в байтах. По умолчанию указан как **320** – это вполне реальное значение и его мы трогать не будем.

Ну, вроде с описанием модели **SED1520** пока все, пора приступать к реализации моделей индикаторов на нем. О построении графики моделей дисплеев мы уже говорили выше, но каждая модель графического индикатора имеет еще в свойствах и собственный MDF-файл, в котором реализована схематика соединения контроллеров. Вот о ней и пойдет речь. Для начала возьмем и воспроизведем по извлеченному из **DISPLAY.LML** файлу MDF схематику одной из существующих в ISIS моделей на базе **SED1520**, например, того же **EW12A03GLY**. Извлеченный MDF и воспроизведенная по нему схема находятся во вложении в папке **GLCD_recovery**. Эта же схема представлена на рисунке 153.

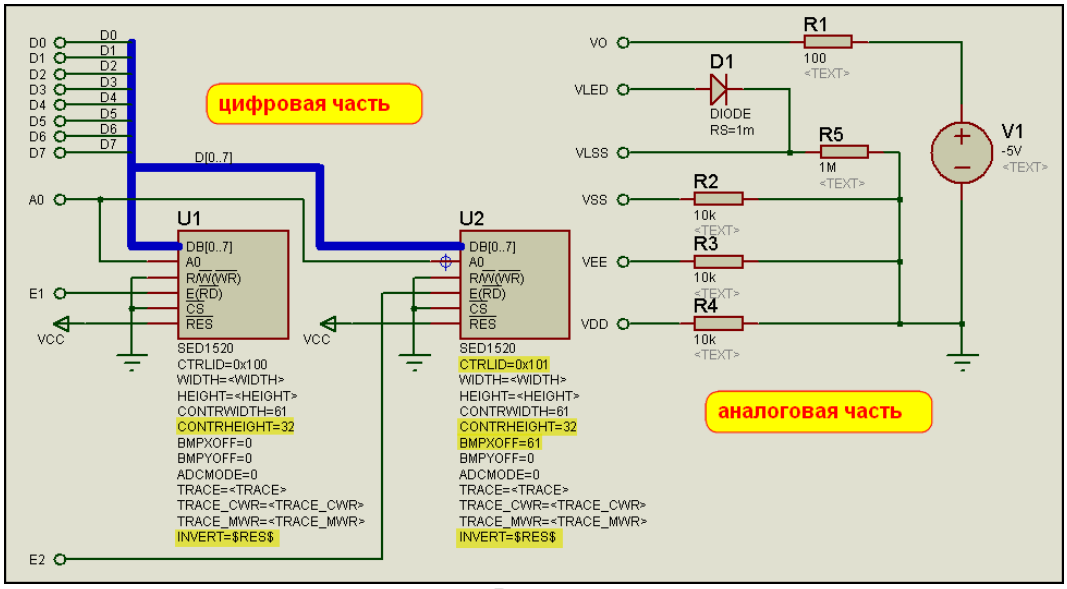


Рис. 153

Структуру модели индикатора **EW12A03GLY** можно условно разделить на две части: цифровую и аналоговую. Аналоговая часть имитирует нагрузки по выводам питания и подсветки и особенного интереса для нас не представляет, поскольку это в основном обычные резистивные нагрузки, за исключением диода, имитирующего светодиодную подсветку экрана и источника отрицательного напряжения, имитирующего встроенный в этот конкретный индикатор преобразователь напряжения. А вот на цифровой части остановимся отдельно. Мы видим, что в данном случае используются два контроллера **SED1520**, у которых большинство выводов объединено, а раздельными являются только выводы выбора **E1** для левого кристалла и **E2** для правого. Для данного конкретного индикатора выводы **R/W(WR)** завешены на землю, поскольку не производится чтение из контроллеров, а выводы сброса **RES** наоборот соединены с питанием и, кроме того, им присвоен параметр **INVERT=RES** для того, чтобы наш индикатор все время запускался в режиме интерфейса с МК серии 68xx.

Примечание. *Напомню, что **RES** ограниченное с двух сторон знаком доллара означает имя вывода со знаком надчеркивания (инверсный). При таких записях надо быть предельно внимательными, особенно, если вывод имеет длинное имя и только часть из него с надчеркиванием. Например, если вы захотите проинвертировать вывод **E** у **SED1520**, то необходимо писать его имя полностью со всеми скобками и т.п., т.е. **INVERT=E(RD)**. Иначе это свойство работать не будет.*

Выводы **CS** драйверов также завешены на землю, как и в большинстве реальных графических индикаторов на основе спаренных **SED1520**.

Какие же еще изменения внесены в свойства контроллеров прописанные по умолчанию. Как я уже и предупреждал ранее, для правого контроллера изменен **CTRLID=0x101**, чтобы кристаллы имели различные идентификаторы. Для обоих кристаллов количество строк **CONTRHEIGHT=32** вместо 16. Надеюсь, понятно почему, ведь это индикатор 122x32. По той же причине для правого кристалла, работающего на правую часть экрана индикатора, установлено горизонтальное смещение картинки по оси X на 61 пиксель - **BMPXOFF=61**. Еще хотелось бы обратить ваше внимание, что для ряда параметров – ширина и высота экрана, а также параметры отладки конкретные значения заменены на имена параметров в угловых скобках. Если кто-то подзабыл, напомню, что таким образом обеспечена возможность задания значений этим параметрам с основного родительского листа или точнее, в данном случае, из свойств графической модели (Рис. 154).

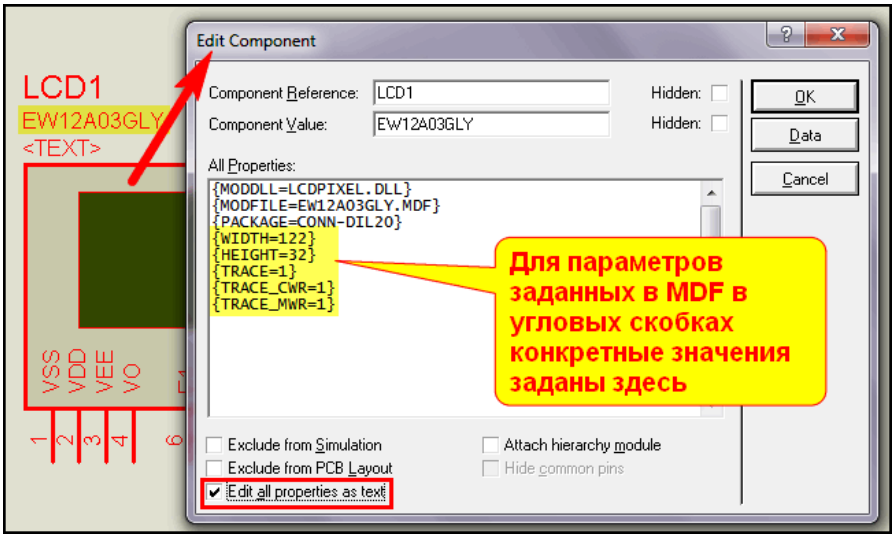


Рис. 154

Особенно хочу отметить, что не стоит пренебрегать заданием этим способом параметров трассировки (отладки) у сложных моделей, каковыми являются индикаторы. Если этого не сделать, то невозможно будет для уже скомпилированной модели включить режим отладки, а он очень важен для нас, в чем мы убедимся чуть ниже.

Ну вот, теперь надеюсь, всем стало ясно, почему программные **VSM** модели графических дисплеев прописаны как **Shematic** и имеют собственные MDF файлы. Я не стану больше останавливаться на зарубежных моделях LCD 122x32 на основе **SED1520**, поскольку в большинстве своем они построены почти одинаково с обязательным разнесением входов **E** для левого и правого кристаллов, т.е. у реального индикатора имеются входы **E1** для тактирования левого кристалла и **E2** для правого. Для примера на рисунке 155 диаграмма цикла записи из даташита **EW12A03GLY**, рассмотренного выше. Правда не знаю, зачем производитель этого индикатора приделал там и чтение, хотя и в даташите вывод **R/W(WR)** четко прорисован на землю.

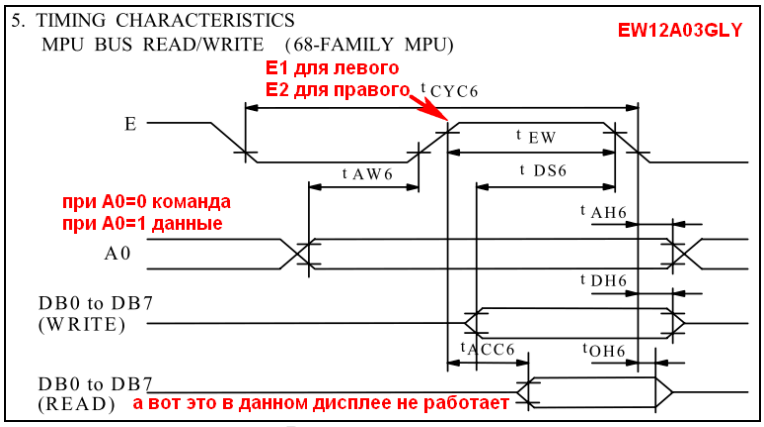


Рис. 155

Из диаграммы видно, что запись команды управления (при сигнале **A0=0**) или байта данных изображения (при сигнале **A0=1**) производится по положительному импульсу на входе **E1** в левый кристалл и на входе **E2** в правый. В других дисплеях возможно и чтение из кристаллов, но при этом учитывается логический уровень на входе **R/W(WR)**, который должен к моменту подачи тактового **E** при чтении быть в состоянии логической единицы, тогда будет верна нижняя часть диаграммы READ.

Ну а мы переходим к графическим индикаторам «местного пошива» и конкретно займемся дисплеями **МТ-12232** компании МЭЛТ. Эти дисплеи формата 122x32 точки на основе Ангстремовского клона **КБ145ВГ4**, совместимого с **SED1520,** выпускаются с различными буквенными индексами, и уже тут скрывается первый «подводный камень». Дело в том, что индикатор **МТ-12232В** среди остальных выделяется как белая ворона, поскольку выполнен по западным стандартам и полностью совместим по сигналам с большинством европейских и китайских дисплеев на базе **SED1520**. Он также имеет два раздельных входа тактирования **E1** и **E2** для разных кристаллов. Временная диаграмма для этого дисплея приведена на рисунке 156. Как видим, все отличие от предыдущей диаграммы в наличии сигнала **R/W(WR)**, обеспечивающего чтение из кристаллов.

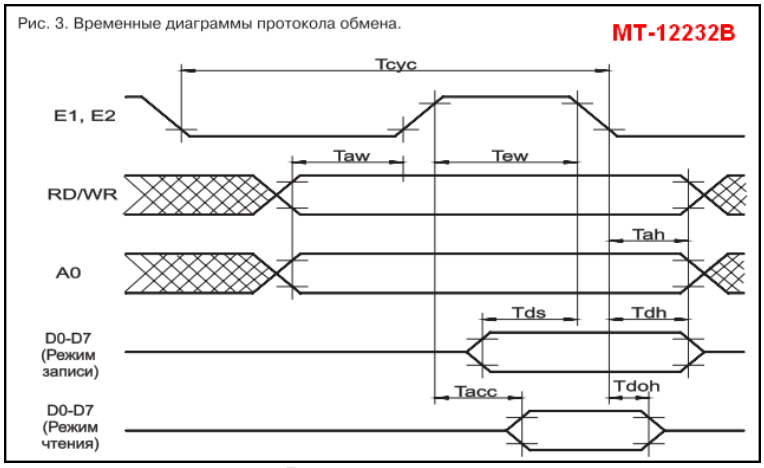


Рис. 156

Фактически для моделирования этого дисплея достаточно использовать уже существующую модель **AGM1232G**. Все отличие состоит в 3-м выводе контрастности **Vo**, который МЭЛТ не задействовал и обратной полярности выводов подсветки **BL** (19, 20). Поскольку ни то ни другое на процесс вывода данных на индикатор не участвует и реализовано только для имитации аналоговых свойств (нагрузки) эти выводы можно просто оставить «в воздухе». Ну а для тех, кому нужно соответствие «буква в букву» в папке **MT12232B** вложения находится графическая модель индикатора – поддиректория **Model_with_Child**. На дочернем листе **Model.DSN** находится переделанная под **MT12232B** подсхема **AGM1232G** (Рис. 157). Скомпилированный с нее файл модели **MT12232B.MDF** и тестовый проект находятся в подпапке **Test_MT12232B**.

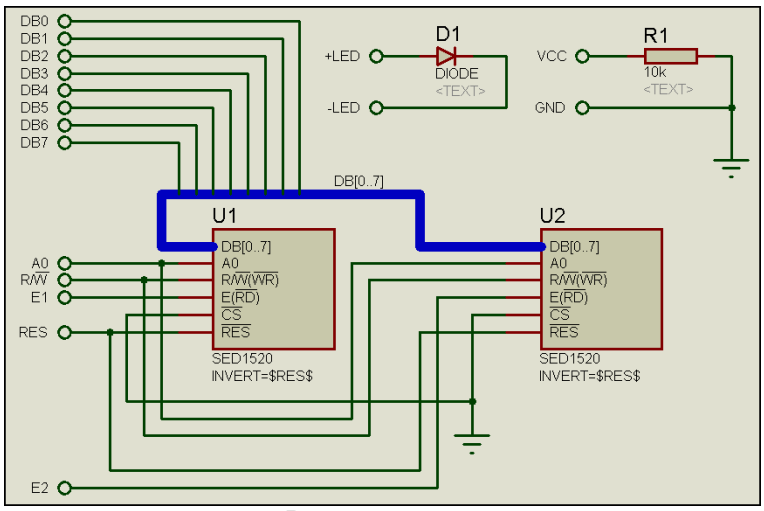


Рис. 157

Ну а мы далее на примере **MT12232A** рассмотрим особенности остальных индикаторов этой серии. Здесь МЭЛТ решил ввернуть свою «изюминку», которая проявляется в особенностях программирования и управления этими дисплеями. Не знаю, насколько это оправдано технологически, но с точки зрения моделирования здесь все становится с ног на голову.

Первая особенность дисплеев с индексами A, C, D и т.д. в отсутствии раздельных выводов **E** для управления кристаллами. Вывод **E** в этих индикаторах всего один и используется для тактирования как левого, так и правого контроллера **КБ145ВГ4**. Для обращения к конкретному кристаллу используется **CS**. Наличие лог. 1 на нем активирует обращение к левому драйверу, отвечающему за вывод информации в левую часть экрана. Лог. 0 на выводе **CS** означает работу с правым кристаллом, обслуживающим правую часть экрана.

Вторая особенность как раз и касается правого кристалла. Дело в том, что здесь МЭЛТ использовал еще и обратную последовательность подключения столбцов (колонок) драйвера к сегментам ЖКИ, т.е. выходу **SEG00** правого кристалла соответствует 122-я колонка дисплея, а выходу **SEG60** – 61-я колонка. Для нормального отображения картинки в правой части дисплея при начальной инициализации контроллеров необходимо для левого кристалла подать команду **ADC=0** (прямой вывод) а для правого **ADC=1** (обратный вывод) изображения. Эта особенность легко выполнима в реальной жизни, но при моделировании в Протеусе накладывает некоторые ограничения. Модель **SED1520** может воспроизводить данные в обратном порядке по команде

ADC=1, но выходы **SEG** мы «перепаять» задом наперед, как в реальном дисплее МЭЛТ не можем – они просто отсутствуют и жестко прописаны в программной модели. Поэтому при моделировании придется в программе инициализации для обоих кристаллов использовать **ADC=0**, а уже для реального «железного» дисплея перед прошивкой контроллера изменять это значение для одного из кристаллов в единицу. Как правило, инициализация проводится один раз при включении (запуске) устройства, поэтому очень больших проблем данный момент не вызывает. Главное – держать это на контроле и не забыть поменять значение при компиляции реальной прошивки.

Первая же особенность легко преодолима схемотехническим методом. Подсхема для компиляции MDF файла для **MT12232A** примет вид как на рисунке 158.

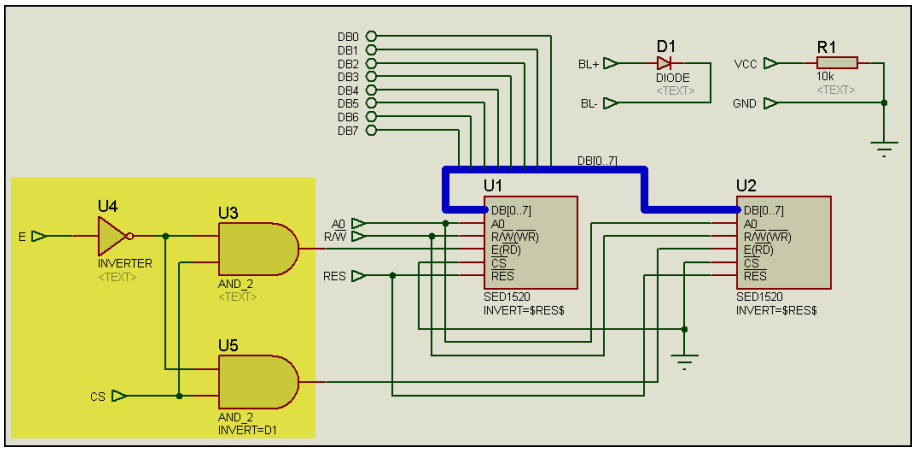


Рис. 158

Я специально оставил узел дешифрации сигнала **E** в первозданном виде, хотя можно было сделать и компактнее, убрав инвертор **U4** и задав свойство **INVERT** для входов **D0** элементов **U3** и **U5**. Вообще здесь уместны различные вариации по теме, но у меня заработало уже в таком виде, а большего и не надо. В папке вложения **MT12232A**, как и для модели с индексом B, расположены соответствующие **Model_with_Child** с дочерним листом, содержащим эту схему, и **Test_MT12232A**, в которой есть и готовый **MT12232A.MDF**.

Подводя итог этому материалу, хочу немного коснуться особенностей процедуры начальной инициализации и вывода информации на индикаторы МЭЛТ. В сети можно встретить различные варианты этой процедуры. В частности, в даташите на **MT12232A** производитель рекомендует следующую последовательность операций:

1. *после подачи напряжения питания удерживать вывод RES в состоянии логического “0” еще не менее 10 мкс;*
2. *подать перепад на вывод RES с логического “0” в логическую “1”, длительность фронта не более 10 мкс;*
3. *ожидать сброса бита RESET в байте состояния или выждать не менее 2 мс;*
4. *подать команду снятия флага RMW (END);*
5. *подать команду включения обычного режима работы (Static Drive ON/OFF);*
6. *подать команду выбора мультиплекса (Duty Select);*
7. *подать команду включения дисплея (Display ON/OFF).*

Первые три операции общие для обоих кристаллов дисплея и выполняются однократно. Они переводят дисплей в режим работы с МК 68xx. Остальные необходимо проделать для каждого кристалла в отдельности, причем в рекомендуемом производителем примере перед операцией 4 для каждого кристалла выдается еще и команда RESET (**0xE2**). Примеры программ от МЭЛТ находятся во вложении вместе с даташитами. Пункты алгоритма начальной установки, начиная с 4 можно переписать боле подробно следующим образом:

4. *подать команду (**A0=0, RD/WR=0**) сброса RESET (**DB7…DB0=0xE2**) в левый кристалл (**CS=0**), стробируем (1-0-1) сигналом **E**,*
   *подать команду (**A0=0, RD/WR=0**) сброса RESET (**DB7…DB0=0xE2**) в правый кристалл (**CS=1**), стробируем **E**.*
5. *подать команду (**A0=0, RD/WR=0**) сбросаRMW (**DB7…DB0=0xEE**) в левый кристалл (**CS=0**), стробируем **E**,*
   *подать команду (**A0=0, RD/WR=0**) сброса RMW (**DB7…DB0=0xEE**) в правый кристалл (**CS=1**), стробируем **E**.*
6. *подать команду (**A0=0, RD/WR=0**) нормальный режим (**DB7…DB0=0xA4**) в левый кристалл (**CS=0**), стробируем **E**,*
   *подать команду (**A0=0, RD/WR=0**) нормальный режим (**DB7…DB0=0xA4**) в правый кристалл (**CS=1**), стробируем **E**.*

7. *подать команду (**A0=0, RD/WR=0**) выбора мультиплекса 1/32 (**DB7…DB0=0xA9**) в левый кристалл (**CS=0**), стробируем **E**,*
   *подать команду (**A0=0, RD/WR=0**) выбора мультиплекса 1/32 (**DB7…DB0=0xA9**) в правый кристалл (**CS=1**), стробируем **E**.*
8. *подать команду (**A0=0, RD/WR=0**) установки верхней строки в 0 (**DB7…DB0=0xC0**) в левый кристалл (**CS=0**), стробируем **E**,*
   *подать команду (**A0=0, RD/WR=0**) установки верхней строки в 0 (**DB7…DB0=0xC0**) в правый кристалл (**CS=1**), стробируем **E**.*

9. *подать команду (**A0=0, RD/WR=0**) установки обратного соответствия (**DB7…DB0=0xA1**) (Внимание! Здесь для Протеуса должно быть **A0**, а для реального индикатора **A1**) в левый кристалл (**CS=0**), стробируем **E**,*
   *подать команду (**A0=0, RD/WR=0**) установки обратного соответствия (**DB7…DB0=0xA0**) в правый кристалл (**CS=1**), стробируем **E**.*
10. *подать команду (**A0=0, RD/WR=0**) включения дисплея (**DB7…DB0=0xAF**) в левый кристалл (**CS=0**), стробируем **E**,*
    *подать команду (**A0=0, RD/WR=0**) включения дисплея (**DB7…DB0=0xAF**) в правый кристалл (**CS=1**), стробируем **E**.*

Инициализацию можно выполнять как в той последовательности, которая указана выше, т.е. поочередно для каждого кристалла по одной команде, так и полностью все процедуры с 4-й по 10-ю сначала для одного (**CS=0**), а затем для другого (**CS=1**). Это уже зависит от того, кому как удобнее оформить программу. Необходимо только помнить про 9-й пункт, который в реальности будет отличаться.

К сожалению, все готовые библиотечные фунции на СИ для дисплеев **MT12232A**, найденные во всемирной паутине содержат те или иные ошибки. В основном это попытки подогнать стандартные функции вывода для дисплеев на базе **SED1520** под особенности **MT12232A**. Поэтому рекомендовать их для использования без внимательного изучения и коррекции я сейчас не могу. Но приведу в качестве примера вывод графического массива с логотипом МЭЛТ, который взят из примера производителя. В примере **LOGO.DSN** из папки **LOGO_MT12232A** вложения использован слегка модифицированный пример **MT12232-CV** приложенный к материалу о драйвере **MT12232A** с сайта **ChipEnable**:

http://www.chipenable.ru/index.php/how-connection/103-podkluchenie-mt12232-k-avr.html

К сожалению, рекомендовать полностью использовать данный материал для моделирования дисплея **MT12232A** в Протеусе не могу, но, во всяком случае, начальная инициализация и вывод графического массива на экран модели проходят корректно, что подтверждается примером (Рис. 159).

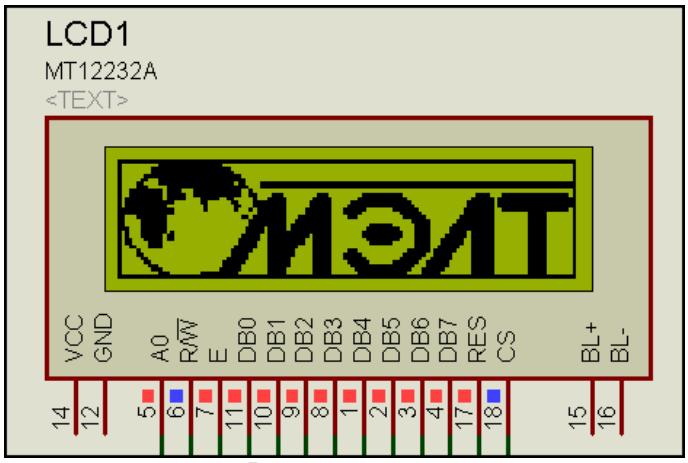
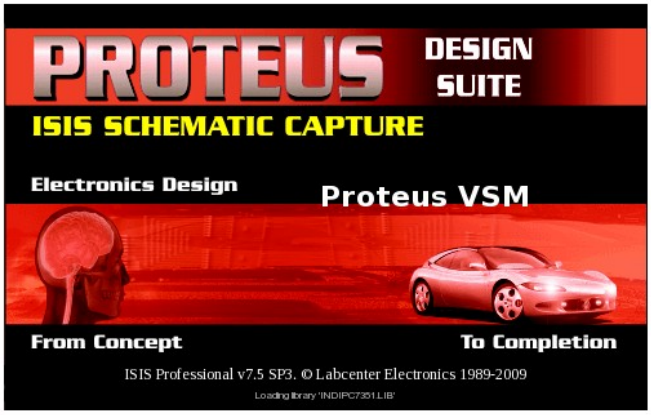


Рис. 159

Возможно, позже при наличии свободного времени я дополню этот материал новыми примерами, тем более, что сам индикатор у меня теперь есть в наличии и можно будет проверить соответствие модели «железу». А пока мы оставляем данный материал, подводим итог, поскольку уже давно не выкладывалась оффлайн версия материала, и переходим к рассмотрению активных моделей с элементами управления.

К содержанию



Intelligent Schematic Input System

Руководство пользователя

Адаптировано к версии 2010 г.

перевод В.Н. Гололобова

Issue 6.0 - November 2002

Оглавление

ВВЕДЕНИЕ

О PROTEUS VSM

Традиционно симуляция схемы была делом не интерактивным. Некогда netlist (спецификация элементов и соединений) делали вручную, и вывод состоял из кучи цифр. Если вам везло, вы получали псевдографический вывод, нарисованный с помощью звёздочек для показа формы напряжения или тока.

Ближе к нашим дням ввод схемы и экранная графика становятся нормой, но процесс симуляции всё ещё остаётся не интерактивным — вы рисуете схему, нажимаете «go», и изучаете результат в какого-нибудь рода пост-процессоре. Это прекрасно, если схема, которую вы проверяете, достаточно статична, не более осциллятора, который в ней есть и генерирует, просто, 1 МГц. Однако, если вы разрабатываете охранную сигнализацию, и хотите найти, что случится, когда будет введён неверный пароль с клавиатуры, требуемые установки становятся довольно непрактичны, и кто-то должен прибегнуть к прототипу. И это — позор, поскольку работа «в кибер-пространстве» может предложить очень много с точки зрения производительности разработки.

Только в образовательных кругах пытались создать симуляцию электрических цепей похожую на работу с электроникой в реальной жизни, когда можно интерактивно взаимодействовать со схемой в процессе симуляции. Проблема в том, что анимация компонентов трудно поддаётся кодированию в программе. Появилось только ограниченное число простых устройств, таких как переключатели, лампочки, электрические моторы и т.п., а это мало используется профессиональными пользователями. Вдобавок, качество симуляции схем часто оставляло желать лучшего. Например, один из ведущих продуктов этого типа не имел временной информации в цифровых моделях.

PROTEUS VSM даёт вам лучшее, комбинируя великолепный симулятор смешанных цепей, основанный на стандарте SPICE3F5, с анимированными моделями компонентов. И он предоставляет архитектуру, в которой дополнительные анимированные модели могут создаваться кем угодно, включая конечных пользователей. Действительно, множество типов анимированных моделей может быть выполнено без обращения к кодированию. Следовательно, PROTEUS VSM позволяет профессиональным инженерам запускать интерактивную симуляцию реальных проектов, и получать в награду результат, подходящий к симуляции схемы. А, кроме того, если этого не достаточно, мы создали ряд моделей симуляции для популярных микроконтроллеров и набор анимированных моделей для периферийных устройств, таких как светодиоды, жидкокристаллические дисплеи, клавиатуры, RS232 терминалы и т.д. Итог — вы можете симулировать полные микроконтроллерные системы, а, следовательно, разрабатывать программы для них без обращения к физическим прототипам. В мире, где время для рынка становится все более и более важным, это реальное преимущество.

Следует отметить, что производительность современных персональных компьютеров, действительно, потрясающая. 300MHz Pentium II PC может симулировать простые микроконтроллерные схемы в реальном времени и в ряде случаев даже быстрее. И в тех случаях, когда скорости не хватает, время реального отклика чаще всего не используется при

программной разработке. Если же вы серьёзно озабочены этим — идите и покупайте 2GHz двух процессорный PC, который, несомненно, много быстрее. Этим полностью развенчивается очевидное возражение против интерактивной симуляции, что она не будет достаточно быстра.

О ДОКУМЕНТАЦИИ

Это руководство предназначено восполнить информацию, предоставляемую в подсказке (on-line help). Руководство содержит дополнительную информацию и примеры, тогда как подсказка предлагает контекстно чувствительную информацию, относящуюся к отдельным иконкам, командам и диалогам. Подсказка для большинства объектов в пользовательском интерфейсе может быть получена после указания курсором и нажатия клавиши F1.

PROTEUS VSM может использоваться двумя путями — либо для интерактивной симуляции, либо для симуляции, основанной на графиках, что и отображено в структуре руководства. Обычно вы используете интерактивную симуляцию, чтобы увидеть, работает ли проект в целом, а графическую симуляцию, когда исследуете, почему нет, или когда нужны детальные измерения. Однако нет надёжных правил. Некоторые элементы системы, такие как генераторы, важны для обоих режимов использования, и имеют отдельные главы по этой причине.

Есть детальные, шаг за шагом руководства, которые позволят вам поупражняться в обоих типах симуляции. Мы очень рекомендуем, чтобы вы поработали с ними, поскольку они показывают все основные техники, требуемые для работы с программой.

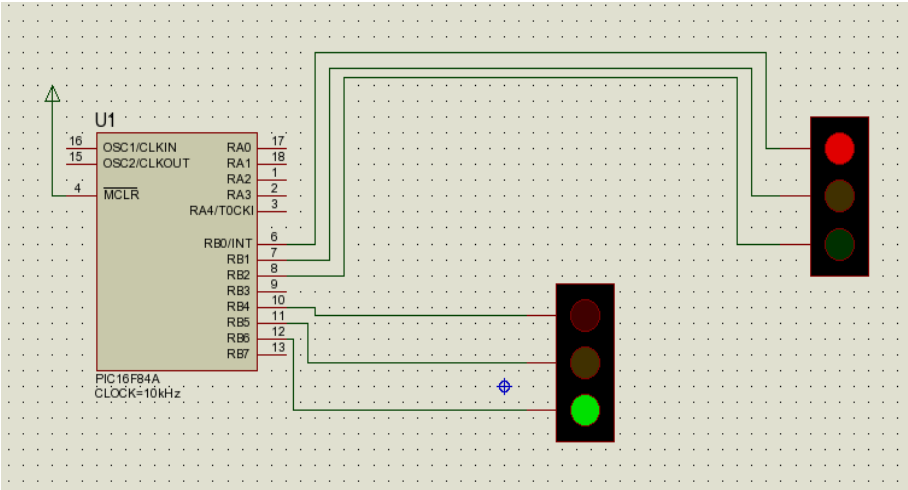
Информация о создании VSM моделей предоставлена отдельно в *VSM Software Development Kit (SDK) на он-лайновом ресурсе, но устанавливается стандартно с профессиональной версией Proteus*.

РУКОВОДСТВО ПО ИНТЕРАКТИВНОЙ СИМУЛЯЦИИ

Введение

Цель этого руководства показать вам, создавая простые схемы, как использовать интерактивную симуляцию в Proteus VSM. Пока мы сосредоточимся на использовании Active Components (активных компонентов) и возможностях отлаживания с помощью ISIS Editor, кроме того коснёмся вопроса основ планирования схем и работы с основными схемами. Полную информацию по этим вопросам можно найти в руководстве к ISIS.

Схема, которую мы используем для симуляции — это пара светофоров, соединённых с микроконтроллером PIC16F84, как показано ниже.



Чтобы не рисовать всю схему сначала, можно найти её в папке примеров «Samples\Tutorials\Traffic.DSN», устанавливаемую вместе с Proteus. Пользователи, которые знакомы с основными процедурами работы в ISIS, могут использовать этот готовый проект и перейти к секции программирования микроконтроллеров. Заметьте, однако, что этот файл проекта содержит преднамеренную ошибку — прочитайте об этом.

Если вы плохо знаете ISIS, интерфейс и основы использования обсуждаются во «Введении» руководства к ISIS. И, хотя мы коснёмся этих вопросов в следующих разделах, постарайтесь выкроить время для знакомства с программой, перед тем как продолжать.

Рисование схем

Размещение компонентов

Мы начнём с размещения двух наборов огней светофоров и PIC16F84 на новом листе схемы. Начните с нового проекта, выберите основной режим и иконку *Component Mode* (все иконки имеют контекстно чувствительную подсказку их назначения), а затем щёлкните по букве «P» над *окном выбранных объектов*. Появится диалог *Device Library* в *окне редактирования* (загляните в руководство ISIS, где больше информации).

Traffic Lights (светофор) можно найти в категории *Miscellaneous*, а PIC микроконтроллер в категории *Microprocessor ICs*. Для выбора компонента для проекта подсветите имя компонента, щёлкнув по его имени в окне *Results*, и нажмите кнопку **ОК**. Переместите курсор в нужное место рабочего поля (курсор похож на карандаш) и щёлкните левой клавишей, пока не появится нужный компонент; он же появится в *окне выбранных компонентов*.

Перемещение и ориентация

У нас появились нужные для схемы блоки, но, возможно, они размещены не совсем удачно. Чтобы переместить компонент, поместите курсор мышки на компонент, щёлкните и, когда рядом с курсором (который стал похож на руку) появится перекрестие, нажмите и удержите левую клавишу мышки, перенося компонент в новое место. К курсору будет «привязан» контур компонента. Обратите внимание, что в этот момент компонент все ещё выделен (подсвечен), щёлкните левой клавишей мышки на любом свободном месте рабочего поля, и компонент приобретёт свой привычный вид.

Для переориентации компонента щёлкните правой клавишей мышки по нему и из выпадающего меню выберите нужную операцию: поворот или отражение. Не забудьте вновь щёлкнуть по свободному месту, чтобы вернуть компонент к нормальному виду.

Потренируйтесь с перемещением и ориентацией компонентов (возможно, используя данный пример). Если у вас появятся проблемы, мы советуем вам обратиться к руководству по ISIS.

Масштабирование

В плане соединения схемы будет полезно иметь возможность менять вид (масштабировать) компонентов к заданной области. Нажатие на клавишу **F6** увеличит вид у текущей позиции курсора или, альтернативно, можно удерживать клавишу **SHIFT** и прорисовать прямоугольник левой клавишей мышки, что увеличит изображение в этой области. Чтобы уменьшить изображение, нажмите клавишу **F7** или, если хотите уменьшить так, чтобы видеть весь чертёж, нажмите клавишу **F8**. Соответствующие команды можно найти в разделе *View* основного меню.

Соединение

Самый простой способ соединять компоненты — это использовать опцию *Wire Auto Router* в разделе *Tools* основного меню. Убедитесь, что эта опция выбрана (иконка слева в меню должна быть «нажата»). Больше информации по работе WAR можно найти в руководстве к ISIS.

Увеличьте PIC пока все его выводы будут удобно видны и затем подведите курсор мышки к концу вывода 6 (RB0/INT). Вы увидите красный квадратик на конце вывода, а курсор станет похож на карандаш. Это означает, что мышка в нужном месте для проведения соединения с этим выводом. Щёлкните левой клавишей мышки, чтобы начать соединение, а затем перемещайте мышку к выводу от красного окна одного из светофоров. Когда на том выводе появится красный квадратик, щёлкните левой клавишей мышки ещё раз, чтобы завершить соединение. Повторите процесс для подключения обоих светофоров, как это сделано на схеме примера.

Есть ряд важных моментов, относящихся к процессу соединения:

- Вы можете проводить соединения в любом режиме — ISIS достаточно сообразительная программа, чтобы определить, что вы делаете.
- Когда выбрана опция *Wire Autorouter*, соединение обходит помехи и, как правило, находит путь между другими соединениями. Что означает для вас — все, что нужно сделать, это два щелчка на двух выводах, и дать возможность ISIS найти путь между этими выводами.
- ISIS автоматически панорамирует экран, если вы пересекаете край рабочего окна при создании соединения. Это означает, что вы можете увеличивать изображение до нужного уровня и, поскольку знаете нужное целевое место, просто проходите по экрану до того момента, когда увидите объект. Альтернативно, вы можете увеличивать и уменьшать изображение, размещая провода (используя клавиши **F6** и **F7**).

И, наконец, мы должны соединить вывод 4 с контактом питания. Щёлкните правой клавишей мышки на свободном месте, выберите из выпадающего меню *Place-Terminal-Power*; теперь щёлкните левой клавишей мышки в подходящем месте для размещения контакта. Определитесь с нужной ориентацией и соедините вывод 4 с контактом питания, как это описано выше.

Больше полезной информации о процессе соединения вы найдёте в руководстве к ISIS.

В этом месте мы рекомендуем загрузить полную версию схемы — это избавит вас от любых недоразумений, если версия, которую нарисовали вы отличается чем-то от нашей!

Написание программы

Листинг исходного кода

Для целей нашего руководства мы приготовили следующую программу, которая позволит PIC управлять светофорами. Эта программа предоставлена в файле, названном TL.ASM и может быть найдена в директории «Samples\Tutorials».

```
        LIST    p=16F84 ; PIC16F844 целевой процессор
        #include "P16F84.INC" ; Включить файл заголовков

        CBLOCK 0x10   ; Временное хранилище
          state
          l1,l2
        ENDC

        org     0               ; Начало вектора.
        goto    setports        ; Переход к началу кода.

          org     4             ; Вектор прерывания.
halt      goto    halt          ; Задаём бесконечный цикл и ничего не делаем.

setports  clrw                  ; Ноль в W.
        movwf   PORTA           ; Убедимся, что PORTA обнулён, прежде чем использовать.
        movwf   PORTB           ; Убедимся, что PORTB обнулён, прежде чем использовать.
        bsf     STATUS,RP0      ; Выбираем Bank 1
        clrw                    ; Маскируем все биты для выхода.
        movwf   TRISB           ; Устанавливаем регистр TRISB.
```

```
        bcf     STATUS,RP0      ; Переключаемся к Bank 0.

initialise clrw                 ; Начальное состояние.
        movwf   state           ; Устанавливаем его.

loop      call    getmask       ; Конвертируем состояние в bitmask.
        movwf   PORTB           ; Записываем его в порт.
        incf    state,W         ; Увеличиваем на единицу состояние в W.
        andlw   0x04            ; Оборачиваем его.
        movwf   state           ; Возвращаем в память.
        call    wait            ; Ждем :-)
        goto    loop            ; И цикл :-)

        ; Функция возвращения bitmask для выхода порта для текущего состояния.
        ; Верхний кусок содержит биты для одной установки света и
        ; нижний кусок биты для другой установки. Бит 1 красный, 2 жёлтый
        ; а бит три зелёный. Бит четыре не используется.

getmask    movf    state,W      ; Берём статус в W.
        addwf   PCL,F           ; Добавляем компенсацию в W для PCL для расчета. goto.
        retlw   0x41            ; state==0 это зелёный и красный.
        retlw   0x23            ; state==1 это жёлтый и красный/жёлтый.
        retlw   0x14            ; state==3 это красный и зелёный.
        retlw   0x32            ; state==4 это красный/жёлтый и жёлтый.

        ; Функция, использующая два цикла для реализации паузы.

wait      movlw   5
        movwf   l1

w1        call    wait2
        decfsz  l1
        goto    w1
        return

wait2     clrf    l2
w2        decfsz  l2
        goto    w2
        return
        END
```

Здесь есть, фактически, предумышленная ошибка в коде выше, но об этом чуть позже...

Добавление исходного файла

Следующий этап — добавление программы в проект, чтобы мы могли успешно симулировать её поведение. Мы сделаем это, используя команды *Source* основного меню. Перейдите к разделу *Source* и выберите команду *Add/Remove Source Files*. Щёлкните по кнопке **New**, переместитесь к директории «Samples\Tutorials» для выбора TL.ASM файла. Щёлкните по кнопке **Открыть** и файл появится в выпадающем списке *Source Code Filename*.

Теперь нам нужно выбрать транслятор для файла. Для нашей цели подходит MPASM. Эта опция доступна из выпадающего списка, а щелчок левой клавишей мышки выберет её обычным образом. (Заметьте, что если вы планируете использовать новый ассемблер или компилятор в этот момент, вы должны зарегистрировать его, используя команду *Define Code Generation* в разделе *Tools*).

И, наконец, необходимо указать, какой файл процессор должен запускать. В нашем примере это будет tl.hex (hex-файл производит MPASM при ассемблировании tl.asm). Чтобы добавить этот файл в процессор, откройте двойным щелчком по компоненту PIC диалоговое окно *Edit Component*, в котором есть поле для *Program File*. Если это уже не задано, как tl.hex, укажите путь к файлу вручную или перейдите к месту хранения файла с помощью кнопки «**?**» справа от поля. Когда вы задали hex-файл, нажмите **ОК** для выхода из диалога.

Теперь мы прикрепили исходный файл к проекту и указали, какой транслятор использовать. Больше информации о системе управления исходным кодом вы найдёте далее.

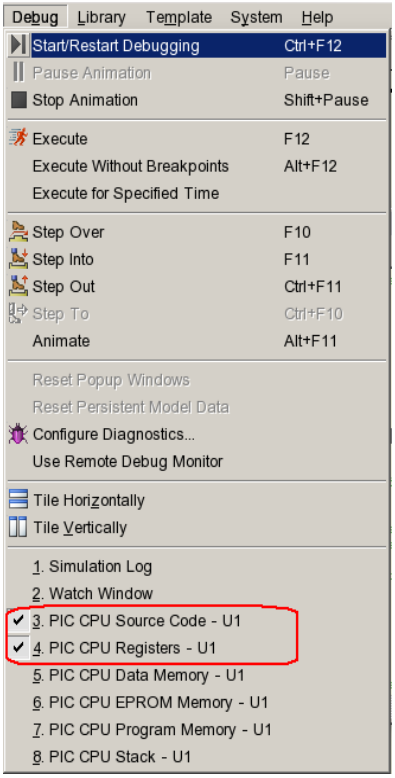
Отладка программы

Симуляция схемы

Чтобы симулировать схему, поместите указатель мышки на кнопку **Play** (►) панели анимации в левом нижнем углу экрана и нажмите её. Панель состояния покажет время активности анимации. Вы должны также отметить, что один из светофоров зелёный, тогда как другой красный, и логическое состояние выводов должно быть видно на схеме. Вместе с тем, вы заметили, что состояние светофоров не меняется. Это произошло из-за преднамеренной ошибки, которую мы сделали в коде. На этом этапе самое время отладить программу, чтобы избавиться от проблемы.

Режим отладки

Чтобы убедиться, что мы находимся в режиме отладки схемы, мы остановим текущую симуляцию. Сделав это (кнопка Stop — ■ на панели анимации) вы можете начать отладку, нажав клавиши **CTRL+F12**. Появятся два всплывающих окна — одно показывает текущие значения регистров, а другое исходный код программы. Если нет, их можно активировать из меню *Debug*, как и остальные информационные окна.



Нам также хотелось бы активировать Watch Window, в котором можно увидеть изменение в состоянии переменных.

Полное описание этих возможностей вы найдёте в разделе, озаглавленном «Watch Window — окно наблюдения».

Сконцентрируемся сейчас на окне Source, обратив внимание на красную стрелку слева. Это, вместе с подсвеченной строкой, показывает текущую позицию счётчика программы. Чтобы задать здесь точку останова (breakpoint), нажмите правую клавишу мышки (точка останова всегда устанавливается на подсвеченной строке) и из выпадающего меню выберите *Toggle (Claear/Set) Breackpoint*. Если вы хотите очистить точку останова, вы можете сделать это повторив операцию, но мы пока её оставим.

Задание точек останова

Взглянув на программу, можно заметить, что цикл возвращается к себе. Неплохо бы было установить точку останова в начале цикла, до того как мы стартуем. Вы можете сделать это подсветив линию (по адресу 000E) мышкой и затем нажав **F9**. Затем нажмите **F12**, чтобы запустить программу. Вы должны увидеть сообщение на панели состояния (Status Bar), показывающее, что цифровая точка останова достигнута и адрес программного счётчика (Program Counter, PC). Это относится к адресу первой точки останова, которую мы задали.

Список ключей отладки есть в разделе *Debug* основного меню, но для большей части работы мы используем клавишу **F11** для пошагового прохождения программы. Нажмите клавишу **F11** и заметьте, что красная стрелка слева перемещается к следующей инструкции. Все, что мы реально сделали — это выполнили команду «clrw», а затем остановились. Вы можете проверить это, взглянув на регистр W в *окне регистров* (PIC CPU Registers) и увидев, что регистр обнулился.

Все, что нам сейчас нужно сделать, это определить, что мы предполагаем должно произойти при выполнении следующей инструкции, а затем проверить, так ли это? Например, следующая инструкция должна переместить содержимое регистра «W» в PORTA. То есть, порт A должен очиститься. Выполним эту инструкцию и проверим окно регистров, убедимся, что это так. Продолжая выполнение, пока не будет достигнута второй точки останова, можно отметить, что оба порта были очищены и готовы для вывода (как это диктуется регистром TRISB), и что переменная state корректно установлена в 0.

Поскольку далее следует функция вызова (call), мы должны выбрать шаг через функцию (Stepping Over, нажатием клавиши **F10**), но для полноты мы пройдём через все инструкции. Нажатие клавиши F11 в этом месте заставляет «нас» перепрыгнуть на первую выполняемую

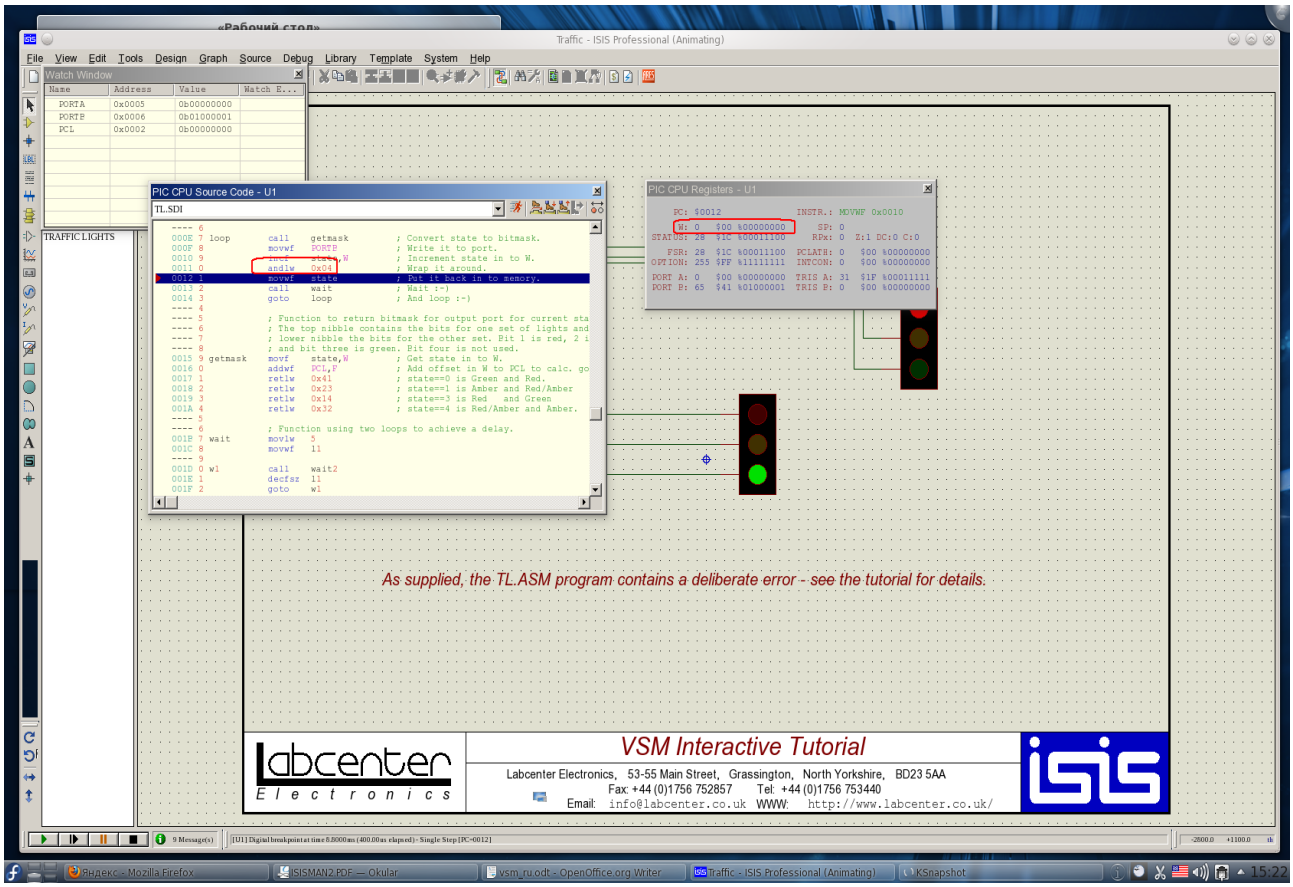
строку функции getmask. Передвигаясь вперёд, мы увидим, что операция move успешна, и что мы «приземлились» в нужном месте для добавления маски в нашу таблицу обозрения. Когда мы вернёмся в основную программу, мы получим маску, которую и предполагали. Шагая дальше и записывая маску в порт, мы увидим правильный результат на схеме. Следующий шаг увеличит на единицу state, это тоже успешно, о чем свидетельствует *окно наблюдения*, где значение регистра «W» увеличивается на 1.

Следующий шаг приводит нас к инструкции, выполняющей возврат state в ноль, когда она увеличивается больше 3. Это, как можно видеть в *окне наблюдения*, не выполняется, хотя должно бы. Переменная state остаётся со значением 1 для маски, которая должна быть правильно установлена при следующем выполнении цикла.

Поиск ошибки

Завершение расследования показывает, что проблема обнаруживается при операции AND с 4 вместо 3. Мы хотели бы, чтобы состояния были 0, 1, 2, 3 и любое из них, когда AND 4, даёт ноль. Вот почему при запуске симуляции состояние светофоров не меняется. Решение простое — заменить проблемную инструкцию AND со state с 4 на 3. Это будет означать, что когда state будет увеличиваться до 3, и когда регистр «W» увеличится до 4, переменная state будет обнуляться. Альтернативное решение — просто проверить в этом случае регистр «W», и когда он станет равен 4, обнулить его.

Этот короткий пример отладки в Proteus VSM иллюстрирует, в основном, что есть множество дополнительных функций. Очень рекомендуем, чтобы вы заглянули в раздел «Отладка на уровне исходного кода» позже в этом документе, чтобы ознакомиться со все этим детальнее.



РУКОВОДСТВО ПО СИМУЛЯЦИИ, ОСНОВАННОЙ НА ГРАФИКАХ

Введение

Цель этого руководства показать вам, используя схему простого усилителя, как выполнять симуляцию, основанную на графиках с PROTEUS VSM. Шаг-за-шагом мы пройдём через:

- Размещение графиков, пробников и генераторов.
- Выполнение актуальной симуляции.
- Использование графиков для отображения результатов и получения замеров.
- Обзор некоторых типов анализа, которые доступны.

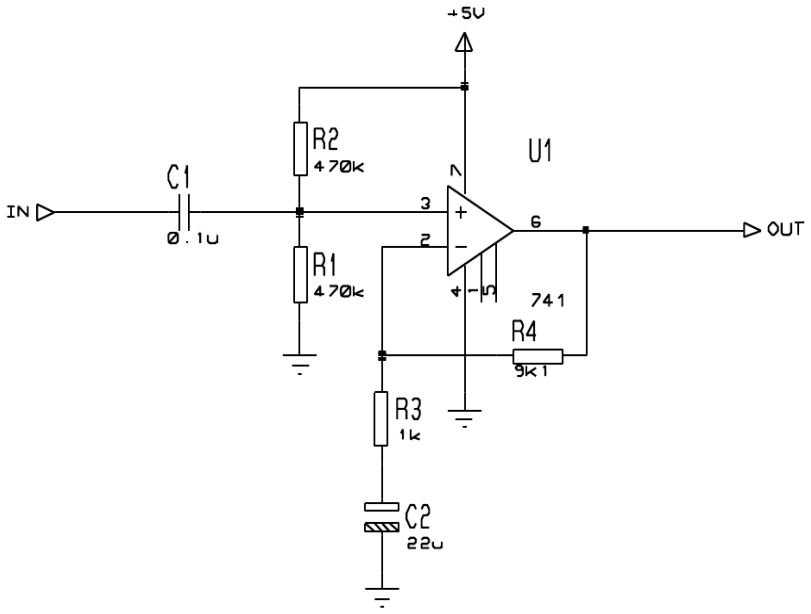
Руководство не отражает полного использование ISIS, таких процедур, как размещение, соединение компонентов, выделение объектов и т.д. Полнее это описано в руководстве «По интерактивной симуляции» и ещё детальнее в руководстве к ISIS. Если вы ещё не освоились с использованием ISIS, сделайте это до того, как обратитесь к этому руководству.

Мы очень советуем вам проработать это руководство до попыток самостоятельно использовать симуляцию, основанную на графиках: освоив концепции, много легче понять и весь материал в соответствующих разделах, что избавит вас от непроизводительных затрат времени и разочарования в дальнейшем.

Начнём

Схема, которую мы собираемся симулировать — это аудио усилитель на основе ОУ 741, как показано ниже. На ней ОУ 741 в обычной конфигурации с питанием от единственного источника 5 В. Резисторы обратной связи R3 и R4 задают усиление порядка 10. Входные компоненты R1, R2 и C1 создают «фальшивое» заземление на не инвертирующем входе, который «развязан» от сигнала.

Как и принято в подобных случаях, мы используем анализ переходных процессов (transient analysis) электрической цепи. Этот вид анализа очень полезен, даёт большой объем информации о схеме. После завершения описания симуляции с анализом переходных процессов, будет сопоставление с другими видами анализа.



Если хотите, вы можете нарисовать схему самостоятельно, но можете загрузить готовый проект: «Samples\Tutorials\ASIMTUT1.DSN». Каков бы ни был ваш выбор, убедитесь сейчас, что ISIS запущен и схема нарисована.

Генераторы

Чтобы проверить схему, нам нужно снабдить её подходящим входным сигналом. Мы будем использовать источник напряжения прямоугольной формы в качестве тестового сигнала. Для генерации требуемого напряжения будет использован объект «генератор».

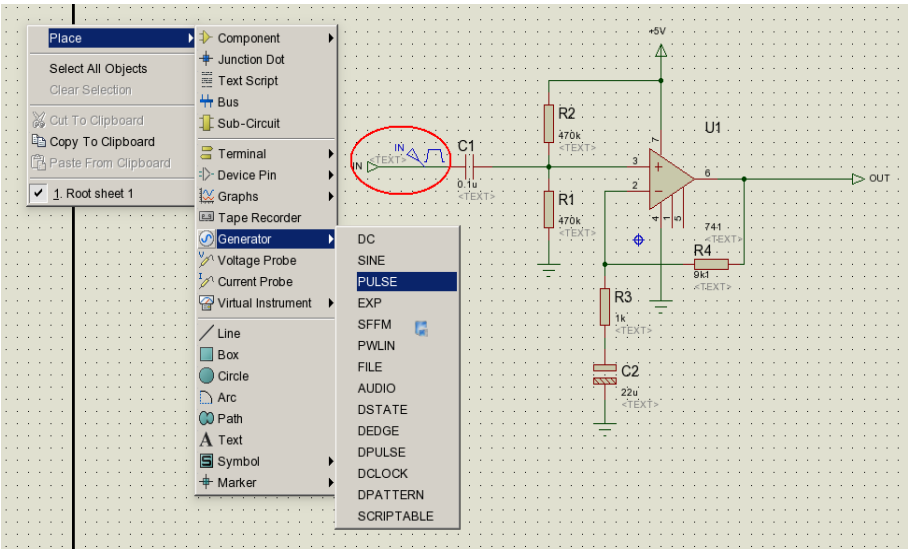
Чтобы поместить на схему генератор, щёлкните правой клавишей мышки на свободном месте рабочего поля, выберите *Place-Generator-PULSE*: для нашей симуляции нам понадобится Pulse-генератор. Выберите генератор, переместите курсор мышки в окне редактора правее контакта IN, и щёлкните левой клавишей мышки по проводу, чтобы поместить генератор.

Объект генератор похож на другие объекты в ISIS — такие же процедуры для предварительного просмотра и ориентации, редактирования генератора, перемещения или удаления (см. «Основные приёмы редактирования» в руководстве к ISIS или «Генераторы и пробники» в этом руководстве).

Также, как мы «прицепили» генератор к существующему проводу, как мы это делаем, мы можем размещать генераторы на листе и соединять со схемой обычным образом. Если оттащить генератор от соединения, тогда ISIS «решит», что вы хотите его отсоединить, и не будет «тащить» провод за ним, как это делает для обычных компонентов.

И, заметьте, как генератор автоматически присваивает ссылку — имя контакта IN.

Если генератор соединяется с объектом (или помещается непосредственно на существующий провод) он присваивает имя цепи, к которой подключён. Если цепь не имеет имени, тогда имя ближайшего вывода компонента будет использоваться по умолчанию.



И, наконец, мы должны отредактировать генератор, чтобы определить параметры импульсов, которые нам нужны. Чтобы отредактировать генератор, щёлкните по нему правой клавишей мышки, выделяя, и выберите из выпадающего меню *Edit Properties*, открывая диалог свойств. Выберите поле Pulsed (High) Voltage и задайте значение 10mV. Также установите ширину импульса 0.5s.

Нажмите клавишу ОК, чтобы изменения вступили в силу. Раздел «Генераторы и пробники» даёт исчерпывающую информацию о свойствах, распознаваемых всеми типами генераторов. Для этой схемы нужен только один генератор, но их количество для размещения не ограничено.

Пробники

После того, как мы определили входной сигнал для нашей схемы, используя генератор, мы должны теперь поместить пробники в точки, за которыми хотим наблюдать. Более всего нас интересует выход, и вход, после того, как он был настроен, тоже полезная точка для пробника. Если нужно, можно добавить ещё пробники в ключевые точки и повторить симуляцию.

Для размещения пробников щёлкните правой клавишей мышки на свободном месте чертежа и выберите из выпадающего меню *Place-Voltage Probe* (убедитесь, что выбрали ненароком не *Current Probe* — мы вернёмся к этому позже). Пробники должны помещаться поверх проводов или помещаться, а затем соединяться с проводами, так же, как и генераторы. Переместите курсор мышки в *окне редактирования* левее вывода 3 U1 и щёлкните левой клавишей мышки, чтобы добавить пробник к проводу, соединённому с выводом 3 и резисторами R1 и R2. Убедитесь, что пробник помещён на провод, поскольку он не может быть помещён непосредственно на вывод. Заметьте, что пробник приобрёл имя ближайшего компонента, к которому он присоединён, с номером вывода в скобках. Теперь поместите второй пробник левее контакта OUT на провод между точкой соединения и контактом.

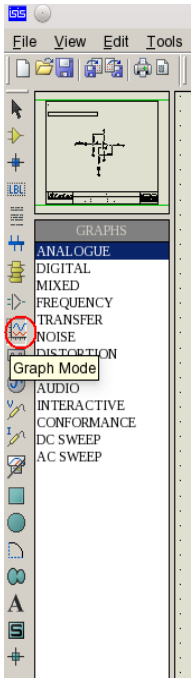
Пробник, как объект, подобен генераторам и большинству других объектов в ISIS — те же процедуры для просмотра, ориентации пробника перед размещением, редактирования, перемещения, переориентации или удаления после размещения (см. раздел « см. «Основные приёмы редактирования» в руководстве к ISIS или «Пробники» в этом руководстве).

Пробники можно редактировать, изменяя их этикетки. В нашем случае прекрасно подходят названия по умолчанию, но при необходимости используйте кончик пробника, а не тело.

Теперь, когда схема готова к симуляции, нам нужно разместить график для отображения результатов.

Графики

Графики играют важную роль в симуляции: они не только служат средой для отображения результатов, но действительно отражают все, что выполняет симуляция. Размещая один (или более) график и показывая какой из типов данных вы ожидаете увидеть на графике (цифровой, напряжение, импеданс и т.п.), вы даёте знать ISIS, какой тип симуляции следует применить и какую часть схемы следует включить в симуляцию. Для анализа переходных процессов нам нужен аналоговый (Analogue) тип графика. Он назван аналоговым, а не **transient**, чтобы отличать его от цифрового (Digital) типа, который используется для отображения результатов при цифровом анализе, действительно специализированной форме **transient** (анализ переходного процесса) анализа. Оба могут изображаться на той же оси времени, если использовать Mixed (смешанный) график.



Чтобы поместить график, вначале выберите иконку *Graph Mode*: в окне выбора компонентов отобразится список всех доступных типов графиков. Затем выберите *Analogue* тип, поместите курсор мышки на свободном месте окна редактирования, щёлкните левой клавишей мышки и «растащите» прямоугольник, чтобы он стал нужного размера, и щелкните ещё раз, размещая график.

График ведёт себя подобно другим объектам в ISIS, хотя есть и некоторые тонкости. Мы снабдили необходимым пример для руководства, но соответствующий раздел будет лучше почитать. Вы можете выделять график обычным образом, затем, используя «ручки» и левую клавишу мышки растягивать график, или поместив курсор на график послед выделения, когда курсор приобретает вид руки, перемещать сам график

Нам сейчас нужно добавить наш генератор и пробники на графике. Каждый генератор имеет пробник, связанный с ним, так что нет необходимости помещать пробник непосредственно на генераторе, чтобы увидеть входной сигнал. Есть три способа добавить генератор и пробники на график:

- Первый — выделить каждый пробник/генератор по очереди и перетащить их на график — точно так, как вы их перемещаете на новое место. ISIS обнаружит, что вы пытаетесь поместить пробник/генератор на графике, возвращает их на место и добавляет кривые на графике с теми же ссылками, что и на пробнике/генераторе. Кривые могут быть связаны с левой или правой осями в аналоговом графике, а пробник/генератор будут добавлены к ближайшей оси со стороны размещения.

Независимо от того, куда вы поместили пробник/генератор, новая кривая всегда добавляется ниже уже существующих графиков.

Второй и третий методы добавления пробников/генераторов на график оба используют команду *Add Trace* из меню графика; эта команда всегда добавляет пробники к текущему графику (если их более одного, текущий график, он один, выбирается через раздел основного меню *Graph*).

- Если команда *Add Trace* выполняется без выделенного пробника или генератора, тогда появляется диалоговая форма *Add Transient Trace*, и пробник может быть выбран из списка всех пробников проекта (включая пробники на других листах).
- Если есть выделенные пробники/генераторы, выполнение команды *Add Trace* (из основного меню) приведёт к тому, что появится приглашение *Quick Add* (быстро добавить) выделенные пробники к текущему графику; при выборе опции **Cancel**, появляется диалоговая форма *Add Transient Trace*, как описано выше. Выбор OK добавляет все выделенные пробники/генераторы к текущему графику в алфавитном порядке.

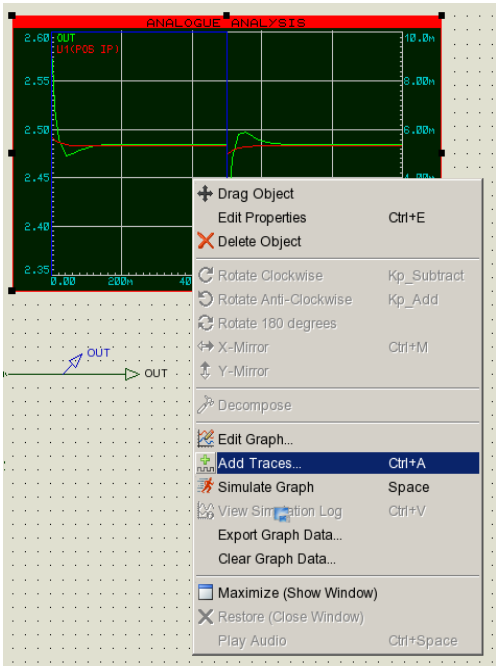
Мы используем *Quick Add* для добавления наших пробников и генератора к графику. Либо выделите пробники и генераторы индивидуально, или, ещё быстрее, сделайте прямоугольное выделение всей схемы — механизм *Quick Add* проигнорирует все выделенные объекты, кроме пробников и генераторов. Выберите команду *Add Trace* из раздела *Graph* основного меню и нажмите кнопку **ОК** в приглашении. Кривые (их заголовки) появятся на графике (поскольку график один, и он последним использовался, понятно, что он — текущий график). В этот момент кривые состоят из имени (на левой оси) и из области пустых данных (основная область графика). Если кривые не появились, возможно, они слишком малы, чтобы программа ISIS смогла нарисовать их. Увеличьте график, выделив его и растаскивая углы, чтобы получить нужный размер.

Как только это произойдёт, наши кривые (размещённые в алфавитном порядке) появятся. Однако мы можем их перемешать. Чтобы это сделать, убедитесь, что график не выделен, щёлкните левой клавишей мышки по имени кривой и перетащите, удерживая левую клавишу, её в нужное место. Кривая подсвечивается, чтобы показать, что она выделена. Кривую можно удалить, выбрав команду *Delete Trace* из выпадающего меню после щелчка правой клавишей мышки по имени кривой. Чтобы снять выделение со всех кривых, щёлкните мышкой в свободном месте чертежа.

Остался последний штрих установок перед тем, как выполнить симуляцию — это задать время симуляции. ISIS симулирует схему согласно со временем окончания на шкале x графика, а для нового графика это задано в 1 секунду. Для нашей цели мы хотели бы, чтобы входной сигнал был близко к высшей рабочей частоте, скажем, 10 кГц. Это потребует общего периода в100µs. Выделите график и щёлкните по нему левой клавишей мышки, чтобы появился диалог *Edit Transient Graph*. Форма эта имеет поля, которые позволяют вам озаглавить график, задать время начала и окончания симуляции (это относится к левому и правому наибольшим значениям оси x), озаглавить левую и правую оси (это не отображается на цифровых графиках), а также задать основные свойства для запуска симуляции. Всё, что нужно сделать нам, это изменить время окончания симуляции с 1.0 на 100u (вы можете ввести символы 100u — ISIS конвертирует это в 100E-6) и нажать **ОК**.

Теперь проект готов к симуляции. В этом месте, пожалуй, лучше загрузить нашу версию

проекта (Samples\Tutorials\ASIMTUT2.DSN), чтобы избежать появления каких-либо проблем при симуляции и для следующих разделов. Но вы можете и продолжить с проектом, который правили сами, а загрузить файл ASIMTUT2.DSN только в том случае, если проблемы появятся.



Симуляция

Чтобы симулировать схему, всё что вам нужно сделать, это дать команду *Simulate Graph* в разделе *Graph* основного меню (или использовать клавиатуру — пробел). Команда выполнит симуляцию и текущий график (тот, что отмечен в меню *Graph*) будет обновлён результатами симуляции.

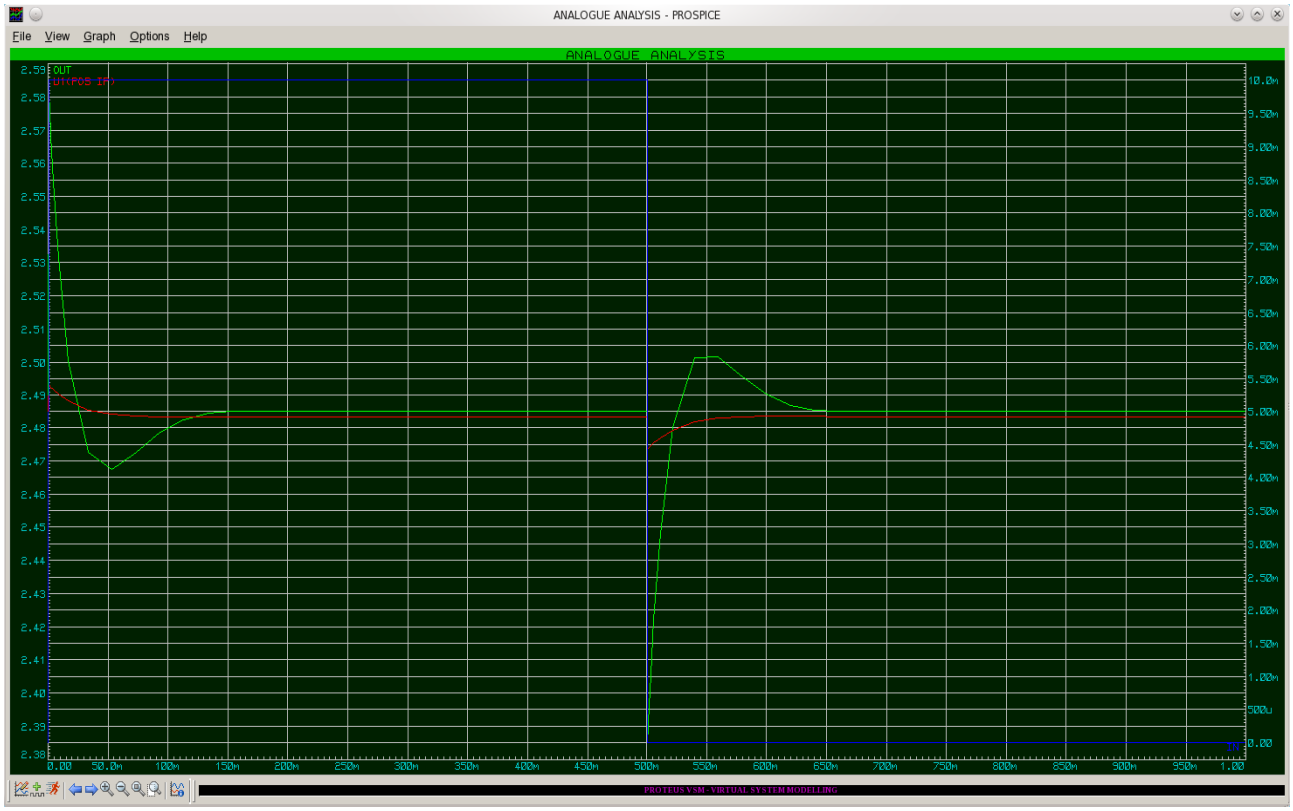
Выполните это сейчас. Панель состояния показывает, до какого места дошёл процесс симуляции. Когда симуляция завершена, график перерисовывается, используя новые данные. Для текущей версии ISIS (версии 6.0) и ядра симулятора начало графика игнорируется — симуляция всегда начинается с нулевого времени и продолжается до достижения времени остановки или до момента, когда симуляция доходит до неизменного состояния. Вы можете прервать симуляцию на середине, нажав клавишу **ESC**.

Журналирование симуляции ведётся для последней выполненной симуляции. Вы можете увидеть лог (запись журнала), используя команду *View Log* из раздела *Graph* основного меню (или с помощью горячих клавиш **CTRL+V** клавиатуры). Лог при аналоговой симуляции редко появляется для немедленного чтения, только тогда, когда есть предупреждения или ошибки, о которых нужно знать, и в этом случае вы можете просмотреть детали — что пошло не так. В некоторых случаях, однако, запись симуляции может предоставить полезную информацию, которую нелегко найти на кривых графика.

Итак, первая симуляция завершена. Бросив взгляд на график, трудно разглядеть какие-то детали. Чтобы проверить, работает ли схема должным образом, нам нужно выполнить некоторые замеры...

Выполнение измерений

График, созданный на схеме, минимизирован. Чтобы провести измерения, мы вначале должны максимизировать его. Для этого вначале убедитесь, что график не выделен, а затем щёлкните левой клавишей мышки по его заголовку — график перерисуется в своём собственном окне. В верхней части окна появится меню. Под ним в левой части экрана есть область, в которой отображаются заголовки кривых, а справа от этого сами кривые. В нижней части окна, слева, есть инструментальная панель, правее которой панель состояния, отображающая информацию о курсоре (времени/состоянии). Поскольку это новый график, и мы не проводили никаких измерений, на графике не видно курсоров, а панель состояния просто отображает заглавное сообщение.



Кривые повторяют цвета своих заголовков. Кривые OUT и U1(POS IP) собраны в верху, тогда как IN внизу. Чтобы разглядеть детали кривых, нам нужно отделить кривую IN от остальных двух. Это можно сделать, перетащив левой клавишей мышки заголовок кривой на правую часть экрана (как на рисунке выше). В этом случае появляется правая ось y, которая масштабирована независимо от левой. Кривая IN теперь выглядит гораздо больше, но это потому, что ISIS выбирает наилучший масштаб для обзора на правой оси. Для того, чтобы сделать график яснее, возможно, лучше удалить кривую IN, а U1(POS IP) пока ещё полезна. Щёлкните правой клавишей мышки по заголовку IN, и выберите *Delete Trace* из выпадающего меню. График вернулся к единственной, расположенной слева оси y.

Мы измеряем два значения:

- Усиление по напряжению схемы.
- Подходящее время спада на выходе.

Эти измерения выполняются с помощью *Cursors*.

Каждый график имеет два курсора, названные как *Reference* и *Primary* курсоры. Ссылочный (reference) отображается красным, а первичный (primary) зелёным. Курсор всегда «привязан» к кривой; кривая, к которой привязан курсор, обозначается маленьким крестиком «x», который перемещается по кривой. Маленькие маркеры на обоих осях x и y следуют за этим перекрестием «x», чтобы дать возможность правильно прочитать значение на осях. Если при перемещении использовать клавиатуру, то курсор будет перемещаться по оси x маленькими шагами.

Давайте начнём с размещения *Reference* курсора. Для доступа к обоим, *Reference* и *Primary*, курсорам используются одинаковые клавиши. Какой из них, определяется клавишей **CTRL** на клавиатуре; *Reference* курсор, используемый меньше, всегда требует нажатия клавиши **CTRL**. Чтобы поместить курсор, всё что от вас требуется, это указать точку на кривой (не этикетку кривой — она используется для другой цели), где вы хотите «прицепить» курсор, и щёлкнуть левой клавишей мышки. Если нажата клавиша **CTRL**, вы поместите (или будете перемещать) *Reference* курсор; если же клавиша **CTRL** не нажата, тогда вы поместите (или переместите) *Primary* курсор. Пока нажата клавиша мышки (и клавиша **CTRL** для *Reference* курсора), вы можете перетаскивать курсор. Итак, нажмите (и удержите) клавишу CTRL, поместите указатель мышки справа на графике над двумя кривыми и нажмите левую клавишу мышки. Появится красный *Reference* курсор. Протащите курсор (всё ещё с нажатой клавишей **CTRL**) между 70u и 80u по оси x. Заголовок в строке состояния пропадает, и теперь строка состояния отображает время (под курсором, красным, слева) и напряжение (под курсором) с именем кривой (справа). Это кривая OUT, которую мы хотели видеть.

Вы можете перемещать курсор по оси X, используя курсорные клавиши клавиатуры (влево и вправо), и вы можете «прицепить» курсор к предыдущей или следующей кривой, используя курсорные клавиши вверх и вниз. Клавиши **ВЛЕВО** и **ВПРАВО** перемещают курсор к левому или к правому ограничителю оси X, соответственно. С нажатой клавишей **CTRL** попробуйте перемещать клавишами влево-вправо *Reference* курсор маленькими шагами по оси времени.

Теперь поместите *Primary* курсор по кривой OUT между 20u и 30u. Процедура такая же, что и для Reference курсора выше, исключая то, что вам не нужно удерживать клавишу **CTRL**. Время и напряжение (зелёным) для первичного курсора добавляются теперь на панель состояния.

Также отображается разность и по времени (DX), и по напряжению (DY) между положениями обоих курсоров. Разность напряжения составляет около 100mV. Входной импульс был 10mV, так что усиление по напряжению составляет около 10. Заметьте, что значение положительное, поскольку *Primary* курсор над *Reference* курсором — дельта выхода это значение Primary минус Reference.

Мы можем также измерить время спада, используя значение разности во времени между позициями курсоров на падающем участке выходного импульса. Это можно сделать либо перетаскивая курсоры мышкой, либо курсорными клавишами (не забудьте про клавишу **CTRL** для *Reference* курсора). *Primary* курсор должен быть правее по кривой, на её спрямлении, а *Reference* курсор на сгибе начального участка спада импульса. Вы можете определить, что время спада чуть меньше10µs.

Использование пробников тока

Теперь мы должны закончить с измерениями, мы можем вернуться к схеме — достаточно закрыть окно графика обычным образом или для ускорения можно нажать клавишу **ESC** на клавиатуре. Мы используем теперь токовый пробник для проверки тока в петле обратной связи, измеряя ток через R4.

Пробники тока используются аналогично пробникам напряжения, но с одним важным отличием. Токовый пробник нуждается в направлении, связанном с ним. Пробники тока работают фактически в разрыве провода, куда они вставляются, так что они должны «знать» пути, их окружающие. Это выполняется просто способом размещения. В ориентации по умолчанию (в направлении направо) пробник тока измеряет ток в горизонтальном проводе слева направо. Чтобы измерить ток в вертикальном проводе, пробник нужно повернуть на 90° или 270°. Размещать пробник под неверным углом — это ошибка, о чем будет сообщено при выполнении симуляции. Если есть сомнения, посмотрите на стрелку на символе. Она указывает на направление тока.

Выберите пробник тока, щёлкнув по иконке *Current Probe Mode*. Щёлкните по иконке поворота по часовой стрелке, так чтобы стрелка показывала вниз. Затем поместите пробник на вертикальный провод между правым выводом R4 и выводом 6 U1. Добавьте пробник к правому краю графика, выделив и перетащив его на правую сторону минимизированного графика. Правая сторона — хороший выбор для пробника тока, поскольку обычно масштаб его амплитуды иной, чем у пробника напряжения, так что отдельная ось понадобится для детального отображения. В этот момент нет кривой, отображаемой пробником тока. Нажмите пробел для новой симуляции графика, и кривая появится.

Даже из минимизированного графика мы можем видеть, что ток в петле обратной связи следует форме выхода, как вы могли бы ожидать от операционного усилителя. Ток изменяется между 10μA и 0 в верхней и нижней части кривой соответственно. Если вы хотите, график можно максимизировать для детальной проверки кривой.

Частотный анализ

Как и анализ переходных процессов, при симуляции аналоговых цепей возможны несколько других типов анализа. Во многом они используются таким же образом: для графиков, пробников и генераторов, — но все они по разному варьируют эту тему. Следующий тип анализа, который мы проведём, это частотный анализ. При частотном анализе по оси x откладывается частота (в логарифмическом масштабе), а амплитуда и фаза могут отображаться по оси y.

Для выполнения частотного анализа требуется график FREQUENCY. Щёлкните по иконке Graph Mode, чтобы вновь отобразить список типов графиков в окне выбора, и щёлкните по типу FREQUENCY. Затем поместите график на схему, как и раньше, растащите окно графика левой клавишей мышки. Нет необходимости удалять уже существующий график переходного процесса, но вы может это сделать, в плане освободить как можно больше места (щёлкните по графику правой клавишей мышки и выберите из выпадающего меню *Delete Object*).

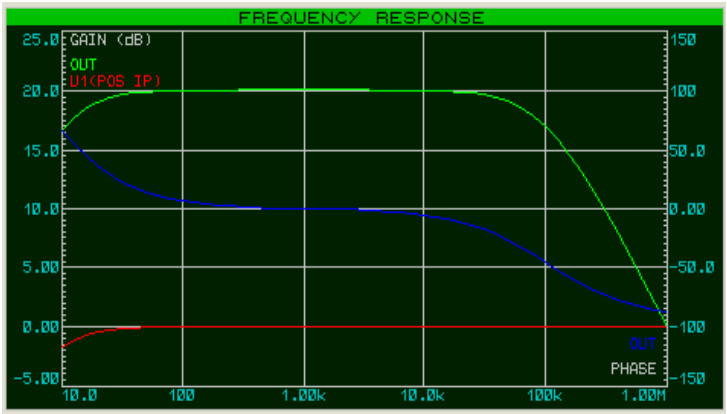
Теперь добавим пробники. Мы добавим оба пробника напряжения OUT и U1(POS IP). На частотном графике две y-оси (левая и правая) имеют специальное назначение. Левая ось y используется для отображения амплитуды сигнала, а правая ось для отображения фазы. Чтобы видеть обе, следует добавить пробники на обе стороны графика. Выделите и перетащите пробник OUT на левую сторону графика, затем перетащите его на правую

сторону. Обычно каждый график имеет свой цвет, но обе кривые имеют одинаковое имя. А теперь выделите и перетащите U1(POS IP) пробник, но только на левую сторону графика.

Значения амплитуды и фазы должны быть заданы по отношению к некоторому значению. В ISIS это выполняется заданием *Reference Generator*. Он всегда имеет выход в 0dB (1 вольт) при 0°. Любой существующий генератор может быть задан, как *reference generator*. Все другие генераторы в схеме игнорируются при частотном анализе. Чтобы задать IN генератор как «относительный» в нашей схеме, просто, выделите и перетащите его на график, как вы это делали с пробниками. ISIS «поймёт», поскольку это генератор, что он должен быть относительным генератором, о чём и сообщит вам. Удостоверьтесь, что вы это сделали, или симулятор будет работать некорректно.

Нет необходимости править свойства графика, поскольку частотный диапазон, выбираемый по умолчанию, вполне подходит для наших целей. Однако, если вы сделаете это (указывая график и нажимая **CTRL-E**), вы увидите диалог Edit Frequency Graph, который слегка отличается от аналогичного для анализа переходного процесса. Нет нужды маркировать оси, и их свойства фиксированы, но есть флажок, который позволяет отображать амплитуду в децибелах или в обычных единицах. Эту опцию лучше оставить в dB, поскольку отображаемые абсолютные значения не будут реальными значениями, присутствующими в схеме.

Теперь нажмите пробел (когда курсор помещён поверх частотного графика), чтобы начать симуляцию. Когда она закончится, щёлкните левой клавишей мышки по заголовку графика, чтобы максимизировать график. Просматривая вначале кривую амплитуды OUT, мы можем заметить, что в полосе пропускания усиления составляет 20dB (как и ожидалось), а диапазон рабочих частот лежит в полосе от 50Hz до 20kHz. Курсоры в данном случае работают так же, как и в предыдущем случае — вы можете проверить предыдущие утверждения с помощью курсоров. Кривая фазы OUT показывает ожидаемые фазовые искажения на краях диапазона, правая ветвь подходит к -90° при единичном усилении. Эффект фильтра на высоких частотах обнаруживается, если сравнить кривую OUT с U1(POS IP). Заметьте, что ось x логарифмическая, и для чтения значений на оси лучше использовать курсоры.



Анализ развёртки переменной

В ISIS можно наблюдать, как сказывается на схеме изменение некоторых параметров цепи. Есть два типа анализа, позволяющие сделать это — *DC Sweep* и *AC Sweep* (развёртка на постоянном и переменном токе). График *DC Sweep* отображает ряд значений рабочих точек в

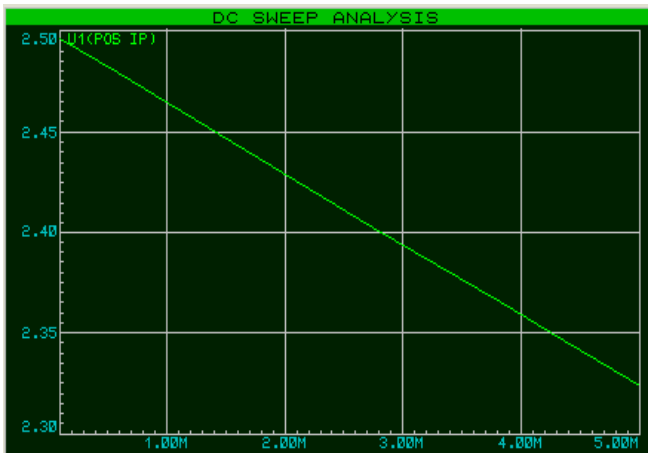
зависимости от переменной развёртки, а график *AC Sweep* отображает последовательность значений единственной точки частотного анализа амплитуды и фазы, подобно графику Frequency.

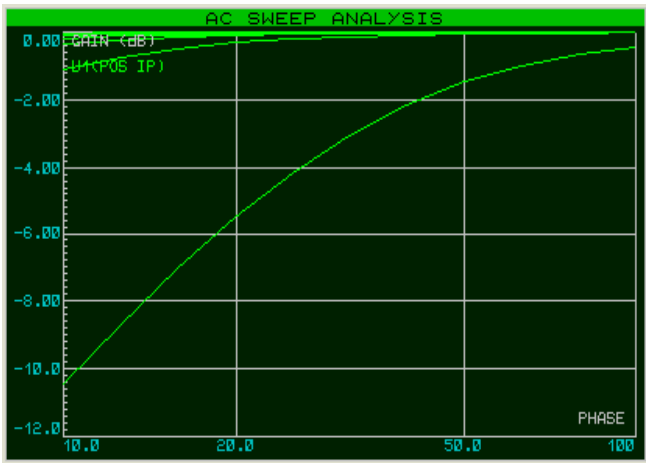
Поскольку обе формы схожи, мы обратимся к одной — *DC Sweep*. Резисторы смещения, R1 и R2, создают маленький ток для U1. Используем *DC Sweep*, чтобы увидеть, как сказывается значение этих резисторов на рабочей точке.

Для начала разместим график *DC Sweep* на свободном месте схемы. Затем выделим пробник U1(POS IP) и перетащим его на левую сторону графика. Нам нужно задать значение развёртки, а это делается с помощью редактирования графика (поместите курсор мышки на него и нажмите **CTRL-E**). Появится диалоговая форма *Edit DC Sweep Graph*, включающая поля для задания имени переменной качания, её начальное и конечное значения и количество шагов при развёртке. Мы хотели бы развернуть значения резисторов в диапазоне, скажем, от 100kΩ до 5MΩ, так что задаём для поля *Start* 100k, а для поля *Stop* 5M. Щёлкаем по **ОК**, чтобы изменения вступили в силу.

И, конечно, резисторы R1 и R2 следует подготовить к развёртке — изменить фиксированные значения, которые они имеют. Чтобы это сделать, редактируем R1 (щелчок правой клавишей мышки по резистору и выбор *Edit Properties* из выпадающего меню), меняя поле Value с 470k на X. Заметьте, что переменная развёртки на графике остаётся X. Щёлкаем по кнопке **ОК**, и повторяем все это для резистора R2.

Теперь можно симулировать схему: курсор мышки на график, нажать пробел. Затем, максимизировав график, вы можете отметить, что уровень смещения уменьшается, тогда как сопротивление рабочей цепи увеличивается. При 5MΩ все значительно меняется. Конечно, изменение этих резисторов также сказывается на частотном режиме. Мы должны провести AC Sweep анализ, чтобы посмотреть, что происходит в районе нижних частот.





Анализ шумов

Последняя форма анализа, о которой мы расскажем, это *Noise* анализ. В этом анализе симулятор показывает количество тепловых шумов, которые генерируются каждым элементом схемы. Все эти шумы затем суммируются (квадратично) в каждой испытуемой точке схемы. Результат выводится для полосы пропускания шумов.

Есть несколько важных особенностей анализа шумов:

- Время симуляции прямо пропорционально количеству пробников напряжения (и генераторов) в схеме, поскольку учитывается каждый из них.
- Пробники тока не учитываются при анализе шумов, следовательно, игнорируются.
- Значительная часть информации содержится в лог-файле симуляции.
- PROSPICE вычисляет и входной, и выходной шум. Чтобы сделать модель, вход должен быть определён как reference — это выполняется перетаскиванием генератора на график, как и при частотном анализе. Изображение входного шума при этом показывает эквивалентный шум на входе для каждого выходного пробника.

Чтобы выполнить анализ шумов нашей схемы, мы должны вначале вернуть резисторам R1 и R2 их прежнее значение 470kΩ. Сделайте это сейчас. Затем выберите тип графика Noise и поместите новый график на свободном месте чертежа. Нас реально интересует только выходной шум, так что перетащите OUT пробник напряжения на график. Как и раньше, предопределённые значения для симуляции нас вполне устраивают, но вам нужно задать reference для входного генератора IN. Диалоговая форма *Edit Noise Graph* имеет флажок для отображения результатов в dB. Если вы используете эту опцию, то убедитесь, что 0dB соответствует 1 вольту r.m.s. Щёлкните по **Cancel**, чтобы закрыть диалог.

Симулируйте график, как прежде. Когда график будет масимизирован, вы можете увидеть, что значения, производимые этим видом анализа, очень малы (nV в нашем случае), как вы можете убедиться из анализа этого типа. Но как вы можете просмотреть источники шума в вашей схеме? Ответ лежит в логе симуляции. Просмотрите запись журнала, нажав **CTRL+V**. Используйте иконку перемещения вниз (со стрелкой вниз), и вы должны увидеть линию, которая начинается словами:

Total Noise Contributions

Это список всех вкладов в шумы (во всем рабочем диапазоне частот) для каждого элемента схемы, производящего шум. Большинство элементов, фактически, внутри операционного усилителя, и имеют префикс U1_. Если вы установите флажок *Log Spectral Contributions* в диалоге *Edit Noise Graph*, то вы получите больше данных в журнале, показывающих вклад каждого компонента на каждой из частот.

ИНТЕРАКТИВНАЯ СИМУЛЯЦИЯ

БАЗОВЫЕ НАВЫКИ

Панель управления анимацией

Интерактивная симуляция управляется с панели, похожей на панель управления видеомагнитофоном, и работает как обычный пульт управления. Эта панель размещена в нижней левой части экрана. На панели четыре кнопки, которые вы используете для управления симуляцией схемы.

- Кнопка **PLAY** запускает симуляцию.
- Кнопка **STEP** позволяет вам выполнить пошаговую анимацию с заданным шагом. Если кнопка нажата и отпущена, тогда симуляция происходит на одном временном интервале; если кнопка нажата и удерживается, то анимация происходит до тех пор, пока вы не отпустите кнопку. Увеличение времени единичного шага может быть настроено в диалоге *Animated Circuit Configuration* (*System-Set Animation Options*). Возможность менять время шага позволяет более внимательно просматривать работу схемы и происходит медленнее, чем в действительности.
- Кнопка **PAUSE** останавливает анимацию, которая может возобновиться, если повторно нажать кнопку **PAUSE**, или может выполняться в пошаговом режиме при использовании кнопки **STEP**. Симуляция также приостанавливается, если обнаруживается точка останова.

  Симуляция также может быть приостановлена нажатием клавиши **PAUSE** на клавиатуре компьютера.
- Кнопка **STOP** говорит PROSPICEЮ, что нужно остановить симуляцию реального времени. Вся анимация останавливается и симулятор выгружается из памяти. Все индикаторы сбрасываются в их неактивное состояние, но приводы (выключатели и т.п.) остаются в их существующем положении.

  Симуляция может также останавливаться с клавиатуры клавишей **SHIFT-BREAK**.

Во время анимации текущее время симуляции и средняя загрузка CPU отображаются на панели состояния. Если не хватает мощности CPU для работы симулятора в реальном времени, время симуляции перестанет продвигаться в реальном времени. Отвлекаясь от этого, нет вредных последствий при симуляции очень быстрых схем, поскольку система автоматически регулирует количество симуляции, приходящейся на анимационный кадр.

Индикаторы и приводы

В отличие от обычных электронных компонентов, интерактивная симуляция обычно использует специальные Active Components (активные компоненты). Они имеют несколько графических состояний и проявляются в двух ипостасях: индикаторы и актураторы. Индикаторы отображают графическое состояние, которое меняется согласно измеряемому

параметру в схеме, тогда как Actuators (приводы) позволяют пользователю определять их состояние, что изменяет некоторые характеристики схемы.

Приводы сделаны с символами, небольшими красными маркерами, которые можно нажать мышкой для управления. Если у вас есть мышка с колёсиком, то вы можете также управлять актуратором, указав на него и прокручивая колесо в нужном направлении.

Установки интерактивной симуляции

Большей частью навык нужен в установках и запуске интерактивной симуляции, сконцентрированной на чертеже схемы в ISIS. Этот аспект наилучшим образом становится понятен, если работать с примерами в руководстве к ISIS. Однако процесс можно суммировать следующим образом:

- Укажите компонент, который вы хотите использовать, в библиотеке элементов, используя кнопку **P** в *окне выбора*.
- Разместите компоненты на схеме.
- Отредактируйте их — щелчок правой клавишей мышки и выпадающее меню или **CTRL-E** — чтобы присвоить подходящие значения и свойства. Множество моделей предоставляют контекстно чувствительную подсказку, так что информацию об индивидуальных свойствах можно увидеть, выделив поле и нажав **F1**.
- Микропроцессорный исходный код можно применить под управлением PROTEUS VSM, используя команды раздела *Source* основного меню. Не забудьте присвоить объектный код (hex-файл) компоненту на схеме.
- Соединяйте схему, щелчками по выводам.
- Удаляйте объекты двойным щелчком правой клавиши мышки.
- Перемещайте компоненты, выделив их и перетаскивая левой клавишей мышки.
- Щёлкните по клавише PLAY на панели управления анимацией, чтобы запустить симуляцию.

При использовании виртуальных инструментов или моделей микропроцессоров всплывающее окно, относящееся к компоненту, может отображаться с помощью команд раздела *Debug* основного меню.

ЭФФЕКТЫ АНИМАЦИИ

Обзор

Кроме эффектов для активных компонентов в схеме, можно создать ряд других эффектов анимации, что поможет вам изучать операции со схемами. Эти опции можно задать, используя команду *Set Animation Options* в разделе *System*. Все установки сохраняются с проектом.

Состояние логических выводов

Pin Logic States — эта опция отображает цветные квадратики на каждом выводе, который

подключён к цифровой или смешанной цепи. По умолчанию квадратики синие для логического 0, красные для логической 1 и зелёные для плавающего состояния. Цвета могут быть изменены командой *Set Design Defaults* раздела *Template* основного меню.

Установка опции довольно умеренно нагружает симулятор, но может быть очень и очень полезна, когда используется совместно с точками останова и в пошаговом режиме, поскольку позволяет вам изучить состояние выходных выводов порта контроллера при каждом шаге по коду программы.

Отображение напряжения на проводах цветом

Show Wire Voltage as Colour — эта опция вызовет то, что любой провод, как часть аналоговой или смешанной цепи, будет отображаться цветом, показывающим напряжение на нём. По умолчанию цветовой спектр задан от синего при –6V через зелёный при 0V к красному при +6V. Напряжение можно изменить в диалоге *Set Animation Options*, а цветовую гамму командой *Set Design Defaults* в разделе *Template*.

Установка этой опции не очень сильно загружает симулятор, но может быть очень эффективна, помогая понять происходящее в схеме, особенно в сочетании с опцией *Show Wire Current as Arrows* (показывать ток стрелками).

Показывать ток в проводах стрелками

Show Wire Current as Arrows — эта опция вызывает появление стрелок на всех проводах с током. Направление стрелки отражает направление тока, и она появляется, если амплитуда тока превышает пороговое значение. Пороговое значение по умолчанию — 1uA, хотя это можно изменить в диалоге *Set Animation Options*.

Вычисление тока в проводах вызывает вставку источника нулевого напряжения в каждый сегмент провода в схеме (отдельно от внутренних моделей), и это может создать большое число дополнительных узлов. Соответственно нагрузка на симулятор может стать значительной. Однако стрелки очень полезны в технике основ электричества и электроники, где схемы, как правило, достаточно просты.

УПРАВЛЕНИЕ ШАГОМ ВРЕМЕНИ ПРИ АНИМАЦИИ

Обзор

Два параметра управляют тем, как выполняется интерактивная симуляция в реальном времени. *Animation Frame Rate* (показатель кадра анимации) определяет, сколько раз в секунду будет обновляться экран, а *Animation Timestep* (шаг времени анимации) определяет, как много симуляции произойдёт в каждом из кадров. Для операций реального времени, timestep должно быть установлено обратно к показателю кадра.

Количество кадров в секунду (Frames Per Second)

Обычно нет необходимости подстраивать показатель кадров, поскольку значение по умолчанию 20 кадров в секунду даёт гладкую анимацию без перегрузки (значительной) графики современных PC. Но иногда полезно замедлить анимацию при отладочных операциях компьютерных моделей.

Шаг времени на кадр (Timestep Per Frame)

Animation Timestep можно использовать, чтобы сделать быстрое выполнение симуляции медленнее, или чтобы медленное выполнение ускорить. Для операций реального времени timestep должно быть установлено обратно к показателю кадра (frame rate). Конечно, введённое значение timestep всегда будет таким, каким оно желательно. Выбор зависит от того, достаточно ли мощности CPU для вычислений, необходимых при симуляции событий внутри времени, отведённого для каждого кадра. Процессор загружает фигуры, отображаемые в процессе анимации, представляя отношение этих двух времён. Если доступной мощности процессора не хватает, загрузка CPU будет 100%, и выполняемый шаг времени на кадр будет уменьшаться.

Время одного шага (Single Step Time)

Оставшийся параметр управления временем шага — это *Single Step Time*. Это время, за которое симуляция будет выполнена, когда кнопка одного шага на панели анимации будет нажата.

ПОЛЕЗНЫЕ СОВЕТЫ

Шкала времени схемы

Интерактивная симуляция будет обычно выглядеть в реальном времени, так что не используйте схему с тактовой частотой 1 МГц или синусоидальным сигналом 10 кГц, пока не зададите значение *Timestep per Frame* в диалоге *Animated Circuits Configuration*.

Если вы собираетесь симулировать что-то, что работает очень быстро, то держите в памяти некоторые детали:

- Мощность процессора не бесконечна, только некоторое количество времени симуляции может уложиться в фиксированное количество реального времени. PROTEUS VSM разработан, чтобы в любом случае поддерживать показатель кадра анимации (количество кадров в секунду) и сокращать некоторые кадры, которые не помещаются в заданное время. Результатом будет то, что очень быстрые схемы будут симулироваться медленнее (относительно реального времени), но ровнее.
- Модели аналоговых компонентов симулируются намного медленнее, чем цифровые. На быстрых компьютерах вы можете симулировать цифровые схемы, работающие на частоте в несколько мегагерц в реальном времени, но аналоговые электронные схемы будут работать только до частоты 10-20 кГц.

  Следовательно, не разумно пытаться симулировать тактовый генератор для цифровой схемы в аналоговой области, вместо этого используйте *Digital Clock* (цифровой тактовый генератор) и установите флажок *Isolate Before* для него, чтобы исключить симуляцию аналогового генератора.

Масштаб напряжений

Если вы решили использовать расцвеченные провода для показа напряжения узлов, вы должны уделить некоторое внимание диапазону напряжений в вашей схеме. Предопределённый диапазон для показа — это +/-6V, так что, если схема работает с

совершенно другими напряжениями, вам понадобится изменить значение *Maximum Voltage* в диалоге *Animated Circuits Configuration*.

Заземление

PROSPICE пытается определить чувствительную точку заземления для любой активной цепи, которая осталась без специально заданного земляного контакта. На практике обычно выбирается средняя точка батарейки или центральный вывод, если цепь имеет раздельное питание. Из этого следует, что положительный контакт батарейки будет над землёй, а отрицательный под ней, соотносясь с красным и синим цветом проводов. Однако, если такое поведение не то, что требуется, вы всегда можете воспользоваться возможностью явного задания точки заземления, используя контакт земли в ISIS.

Точки высокого импеданса

Автоматическое логическое заземление также проверяется для любого вывода устройства, которое не подключено к земле, и автоматически будет добавлен резистор большого сопротивления, чтобы выполнить сходимость для SPICE симуляции. Это означает, что плохо оформленные компоненты схемы (то есть, с не соединёнными или частично соединёнными элементами) будут, в общем, симулироваться — хотя временами это даст странные результаты.

ВИРТУАЛЬНЫЕ ИНСТРУМЕНТЫ

ВОЛЬТМЕТРЫ И АМПЕРМЕТРЫ

Несколько интерактивных вольтметров и амперметров предлагаются в библиотеке активных устройств. Они оперируют в реальном времени и могут быть подключены к схеме, как любые другие компоненты. Когда симуляция начинается, они отображают напряжение на их выводах или ток, протекающий через них, в удобном для чтения формате.

Предложенные модели покрывают FSD в 100, 100m и 100u с разрешением в 3 значащих цифры и максимальным числом в два десятичных разряда. Таким образом объект *VOLTMETER* может отображать значения от 0.01V до 99.9V , а *AMMETER-MILLI* от 0.01mA до 100mA и т.д.

Модели вольтметров поддерживают внутреннее сопротивление до 100 МОм, но его можно изменить, редактируя компонент обычным образом. Если оставить значение пустым, это убирает сопротивление модели.

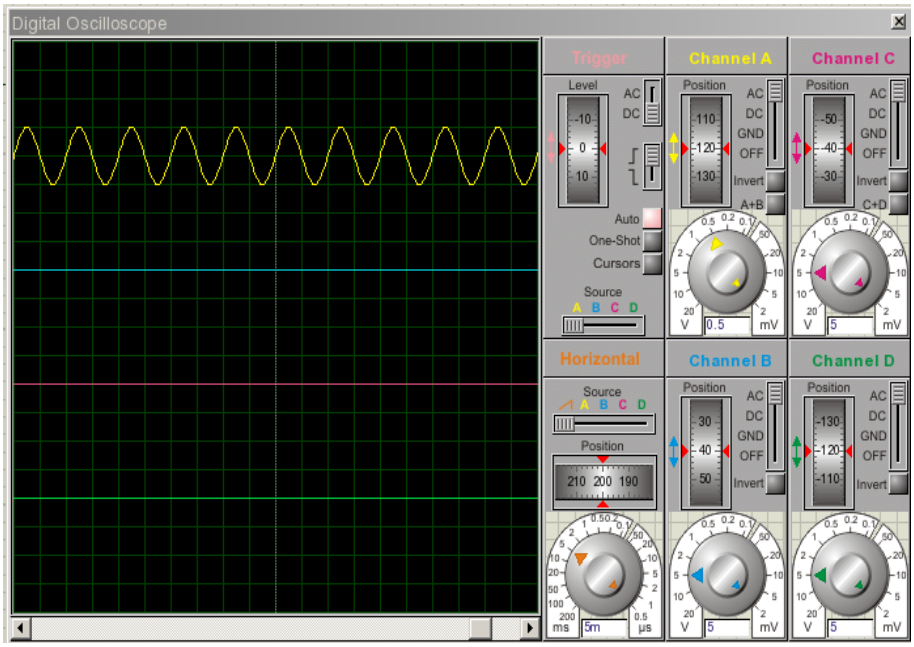
Вольтметры и амперметры переменного тока отображают RMS значения, интегрированные через определимую пользователем постоянную времени.

ОСЦИЛЛОГРАФ

Обзор

VSM осциллограф выполнен как стандартный для всех версий ProSPICE и базовых моделей аналоговых устройств, и имеет следующую спецификацию:

- Четыре канала, X-Y операции.
- Усиление канала от 20V/div до 2mV/div.
- Базовые времена от 200ms/div до 0.5us/div.
- Автоматический уровень переключения, заданный для любого канала.
- AC или DC входы.



Использование осциллографа

Чтобы отобразить аналоговый сигнал:

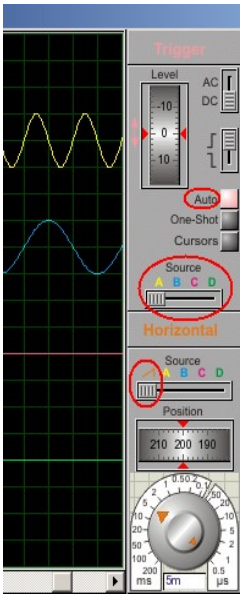
1. Щёлкните по кнопке Virtual Instruments Mode; выберите в окне выбора OSCILLOSCOPE; щёлкните на свободном месте рабочего поля, чтобы добавить символ осциллографа; добавьте соединение с точкой схемы, в которой хотите наблюдать сигнал.
2. Запустите интерактивную симуляцию, нажав на кнопку Play панели управления анимацией. Появится осциллограф.
3. Задайте время развёртки и подходящее напряжение для удобного наблюдения. Вам следует подумать о частоте сигнала, и конвертировать её в обратную величину времени развёртки.
4. Если отображаемый сигнал имеет постоянную составляющую, выберите режим AC на одном или всех каналах.
5. Подстройте усиление по Y и позицию Y, чтобы сигнал имел удобный для наблюдения вид. Если сигнал состоит из переменного напряжения малой амплитуды на большой постоянной составляющей, вам необходимо включить конденсатор между точкой наблюдения и осциллографом, так как, изменение позиции по Y имеет ограниченные возможности.
6. Решите, каким каналом вы будете запускать развёртку.
7. Поверните ручку уровня переключения (Trigger), пока на дисплее не отобразится требуемый участок входного сигнала. Он фиксируется на переднем фронте, если дисковая шкала повёрнута вверх, и на заднем фронте, если вниз.

См. «Дисковая шкала» далее, где подробнее описано, как этим пользоваться.

Режимы операций

Осциллограф может работать в режимах, обозначенных ниже:

- Отдельные каналы — при этом развёртка запускается автоматически, выбранным каналом.



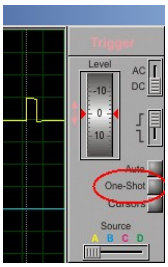
Уровень запуска (Level) определяет, при каком уровне входного сигнала выбранного канала происходит запуск развёртки (панель Trigger).

Положение каналов по оси X на экране определяется дисковой шкалой Position.

Время развёртки определяется ручкой выбора масштаба (время/деление) в нижней части блока управления развёрткой.

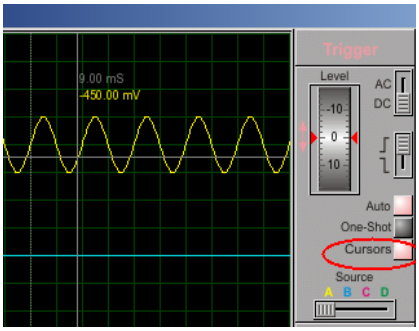
В этом режиме удобно наблюдать за фазовыми и временными сдвигами сигналов (например, на входе и на выходе).

- Снимок экрана — для этого режима на панели Trigger используется кнопка One-Shot.

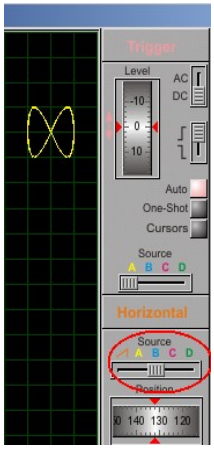


В сложных случаях, когда не удаётся синхронизировать изображение, когда «картинка» на экране не статична, можно произвести снимок экрана и разобраться с происходящим.

- Последняя кнопка в этом ряду, Cursors, позволяет уточнить величину сигнала и время его появления. При нажатой кнопке на экране появляются курсоры, перемещая которые в нужные точки, можно уточнить данные.



- Режим разностного сигнала позволяет наблюдать, например, фигуры Лиссажу, если включить развёртку в этот режим.



Четыре канала осциллографа имеют разные цвета, которым соответствуют цвета надписей на панелях и переключателях, что облегчает поиск ручек управления и положения переключателей для каждого из каналов.

Каждый из каналов может быть выключен, достаточно перевести его переключатель в положение OFF. Это даёт возможность более внимательно просматривать сигналы, если их один или два, используя смещение положения луча работающих каналов с помощью дискового регулятора Position.

Запуск (Triggering)

Осциллограф VSM поддерживает механизм автоматического запуска, что позволяет синхронизировать базовое время с входящим сигналом.

- Какой входной канал используется для запуска, определяется переключателем на панели Horizontal.
- Диск переключения (Level на панели Trigger) последовательно вращается на 360 градусов и задаёт уровень, на котором происходит переключение. Фронт переключения определяется переключателем рядом с диском.
- Если переключение не обнаруживается в течение одного базового интервала времени, базовый интервал будет «освобождён».

Входное соединение

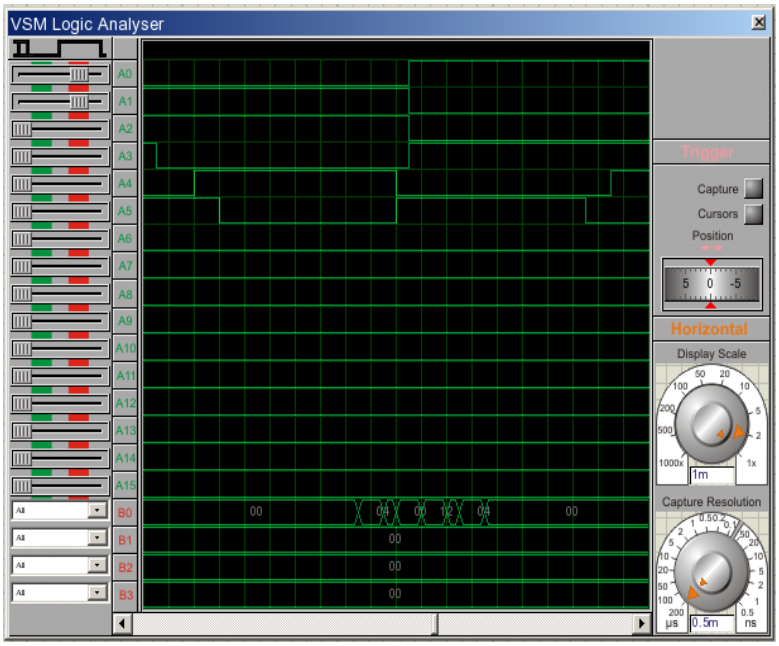
Каждый входной канал может быть непосредственно подключён к точке наблюдения (DC соединение) или подключён через конденсатор (AC соединение). Последний способ полезен тогда, когда входной сигнал имеет маленький уровень переменной составляющей и большой уровень постоянной.

Вход может также кратковременно подключаться к земле (переключатель в положении GND) при настройке координатной сетки перед измерением.

ЛОГИЧЕСКИЙ АНАЛИЗАТОР

Обзор

Логический анализатор (Logic Analyser) представлен как стандартный в PROTEUS VSM Professional, но как дополнительная опция в PROTEUS VSM Lite.

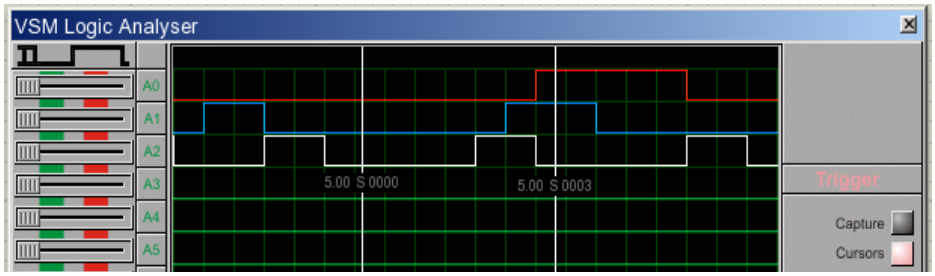


Логический анализатор оперирует последовательно записанными в большой *буфер захвата* входными цифровыми данными. Это процесс отбора, так что есть подстройка разрешения (resolution), которая определяет самый короткий импульс, который может быть записан. На панели запуска (Trigger) есть кнопка **Capture**, которой запускается процесс захвата данных, а спустя некоторое время после выполнения условий переключения останавливается; кнопка меняет свой цвет при записи и после её завершения. Результат, содержимое *буфера захвата* и до, и после переключения, отображается на дисплее. Поскольку *буфер захвата* очень большой (10000 образцов в данном случае), предусмотрено масштабирование и панорамирование изображения. И, наконец, измерительные маркеры (кнопка **Cursors**) позволяют точно определить параметры импульсов и т.п.

Логический анализатор имеет следующую спецификацию:

- 16 x 1 бит и 4 x 8 бит шины трассировки.
- 10000 x 24 бит буфер захвата.
- Разрешение при захвате от 200us на образец до 0.5ns с соответствующим временем захвата от 2s до 5ms.
- Масштаб отображения от 1000 образцов на деление до 1 образца на деление.
- Переключение с комбинацией AND входных состояний и/или фронтов и значениями на шине.

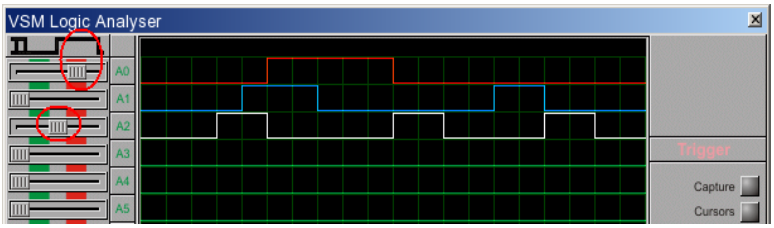
- Позиционирование переключения на 0, 25, 50, 75 и 100% *буфера захвата*.
- Курсоры для точных измерений времени.
- Возможность изменить цвета отображаемых кривых, курсоров, текста и т.д. (щелчок правой клавишей мышки по дисплею и выбор из выпадающего меню *Colours Setup*).



Использование логического анализатора

Чтобы захватить и отобразить цифровые данные:

1. Щёлкните по иконке *Virtual Instruments Mode*; выберите LOGIC ANALYSER, разместите объект на чертеже, соедините входы анализатора с сигналами, которые вы намерены записать.
2. Запустите интерактивную симуляцию, используя кнопку **Play** на *панели управления анимацией*. Появится окно анализатора.
3. Задайте разрешение, подходящее к вашему примеру. Этим устанавливается самый короткий импульс, который будет записан. Чем лучше разрешение, тем короче время захвата данных.
4. Установите на левой панели требуемые условия переключения. Например, если вы хотите переключать инструмент, когда сигнал, подключённый к каналу 1 в высоком состоянии, а сигнал, подключённый к каналу 3 переходит в высокое состояние, вы должны задать первый как «High», а третий как «Low-High».



5. Решите, хотите ли вы видеть данные преимущественно до или после выполнения условий переключения, установите шкалу Position в нужное положение переключения.
6. Когда вы готовы, нажмите кнопку **Capture**. Индикатор загорится, и анализатор будет захватывать входные данные непрерывно, пока не обнаружатся входные условия для переключения триггера. Когда это случится, индикатор станет зелёным, захват данных

продолжится до тех пор, пока не заполнится часть буфера после переключения. Индикатор погаснет, а на дисплее отобразятся анализируемые кривые.

Панорамирование и масштабирование

Поскольку *буфер захвата* удерживает 10000 образцов, а отображение имеет ширину только в 250 пиксел, появляется необходимость в панорамировании и масштабировании *буфера захвата*. *Display Scale* определяет количество образцов на деление, а полоса прокрутки позволяет перемещать изображение влево и вправо.

Заметьте, что считанные данные под *Display Scale* отображают текущее время на деление в секундах, но не являются актуальными установками собственно шкалы. Время деления вычисляется умножением установок шкалы на разрешение.

Измерения

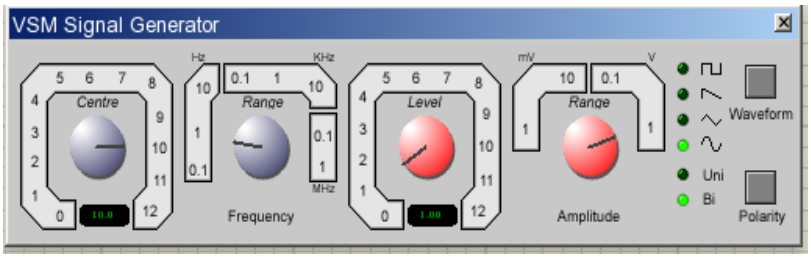
Для точного измерения времени используются маркеры. Каждый маркер можно размещать, используя соответственно окрашенную шкалу. Показания отображают точку времени, отмеченную маркером, относительно времени переключения, а показания Delta A-B — разницу между маркерами.

См. раздел «Дисковые шкалы», чтобы подробнее ознакомиться с этим управлением в логическом анализаторе.

СИГНАЛ-ГЕНЕРАТОР

Обзор

VSM Signal Generator входит как стандартный и для ProSPICE Professional, и для ProSPICE Lite.



VSM Signal Generator представляет собой простой функциональный аудио генератор со следующими возможностями:

- Генерирует выходные импульсы прямоугольной, пилообразной, треугольной и синусоидальной формы.
- Выходная частота 0-12MHz в 8 диапазонах.
- Выходная амплитуда 0-12V в 4 диапазонах.
- Входы для амплитудной и частотной модуляции.

Использование сигнал генератора

Чтобы установить простой аудио сигнал:

1. Щёлкните по иконке *Virtual Instruments Mode*; выберите SIGNAL GENERATOR в окне выбора объектов, поместите его на схему и соедините его выход со входом схемы. В большинстве случаев (то есть, когда схема, которую вы рисуете, требует баланса входного сигнала), вам понадобится заземлить клемму -ve генератора. И самое простое — использовать заземление (правая клавиша мышки, в выпадающем меню *Place-Terminal-GROUND*).

   Входы амплитудной и частотной модуляции можно оставить без подключения, пока оно вам не потребуется.

2. Запустите интерактивную симуляцию, используя кнопку **Play** на *панели управления анимацией*. Появится всплывающее окно генератора.

3. Установите частотный диапазон, который вам подходит. Значение диапазона показывает частоту, которая генерируется, когда центр верньера управления находится в положение 1.

4. Установите амплитуду сигнала, подходящую для вашей схемы. Значение диапазона показывает амплитуду, которая генерируется, когда верньер управления установлен в положение 1. Значение амплитуды представляет пиковый уровень выхода.

5. Нажимайте кнопку **Waveform**, пока индикатор рядом с изображением подходящей формы сигнала не засветится.

Использование входов AM & FM Modulation

Модель сигнал-генератора поддерживает и амплитудную, и частотную модуляцию выходного сигнала. Оба входа, амплитудной и частотной, модуляции имеют следующие возможности:

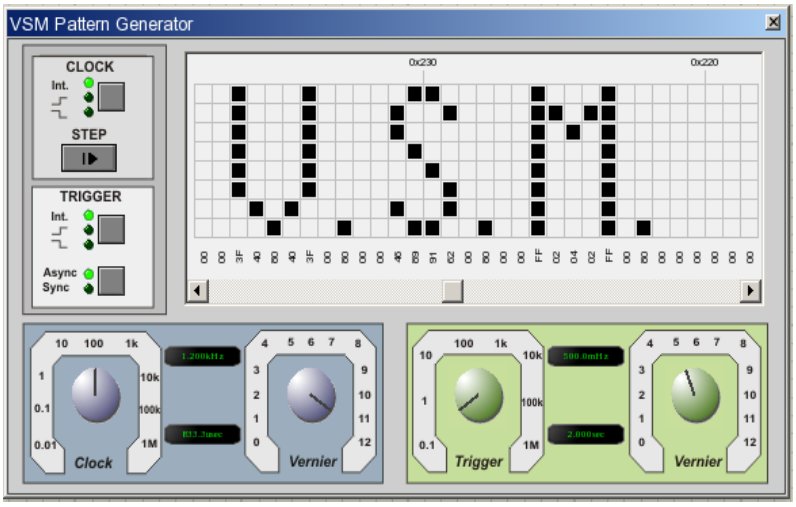
- Усиление входа модуляции в терминах Hz/V или V/V задаются с помощью управления Frequency Range и Amplitude Range, соответственно.
- Входное напряжение модуляции ограничено в пределах +/- 12V.
- Входы модуляции имеют бесконечное входное сопротивление.
- Напряжение на входе модуляции добавляется к установкам управления верньером до умножения установками диапазона, чтобы определить мгновенное значение частоты амплитуды.

Например, если частотный диапазон выбран в 1KHz, а частотный верньер установлен в 2.0, тогда уровень 2V частоты модуляции даст выходную частоту 4kHz.

ЦИФРОВОЙ ГЕНЕРАТОР ШАБЛОНА

Обзор

VSM Pattern Generator — это цифровой эквивалент аналогового сигнал-генератора и предоставляется как стандартный для всех профессиональных версий симуляторов Proteus.



VSM Pattern Generator предназначен для 8-битовых шаблонов до 1 Кбайта и поддерживает следующие возможности:

- Запускается в графике, основанной на интерактивном режиме.
- Имеет внутренний и внешний режим тактирования и переключения.
- Верньерную подстройку для тактовой частоты и шкалы переключения.
- Режим отображения шестнадцатеричной и десятичной сетки.
- Непосредственный ввод значения для большей точности.
- Загрузку и сохранение скрипта шаблона.
- Ручную спецификацию длительности периода шаблона.
- Пошаговое управление позволяет вам использовать шаблон постепенно.
- Окно указателя позволяет вам видеть непосредственно, где вы находитесь на сетке.
- Возможность внешней поддержки шаблона в его текущем состоянии.
- Команды Block Editing на сетке для облегчения конфигурирования шаблона.

Использование генератора шаблонов

Вывод шаблона Pattern Generator'a в режиме интерактивной симуляции:

1. Щёлкните по иконке *Virtual Instruments Mode*; выберите PATTERN GENERATOR в окне выбора объектов, поместите его на схему и соедините с остальной схемой.
2. Инициализируйте интерактивную симуляцию, нажав на кнопку **Play** на *панели управления анимацией*. Появится окно Pattern Generator.
3. Задайте шаблон, который вы хотите вывести, с помощью левой клавиши мышки, щелчки которой по квадратикам сетки переключают их логическое состояние.
4. Решите, будете ли вы тактировать генератор внутренним или внешним образом, установите режим с помощью кнопки **Clock**, последовательно нажимая её до свечения индикатора, соответствующего вашему выбору.
5. Если вы выбрали режим внутреннего тактирования, подстройте частоту с помощью шкалы Clock.
6. Решите, будете ли вы переключаться внутренним или внешним событием, и используйте кнопку **Trigger** для выбора соответствующего режима. Если вы будете управлять переключением от внешнего источника, вам нужно подумать, будет ли переключение синхронным или асинхронным с тактовым генератором.
7. Если вы решили переключаться внутренним генератором, подстройте шкалу Trigger к нужной частоте.
8. Нажмите кнопку **Play** на *панели управления анимацией*.
9. Чтобы использовать шаблон в одном такте, нажмите кнопку паузы на *панели управления анимацией*, а затем нажимайте кнопку **Step** слева от сетки.

См. раздел «Дисковые шкалы» далее, чтобы лучше понять, как пользоваться этим в Pattern Generator.

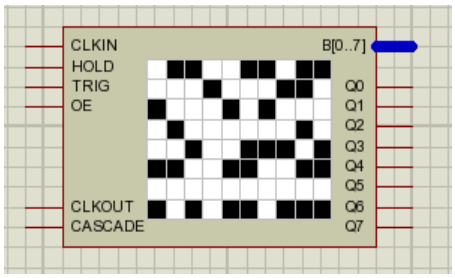
См. раздел «Режимы переключения» ниже, где детально описаны разные режимы переключения.

См. раздел «Дополнительные функции» ниже, где больше информации о конфигурировании и использовании Pattern Generator.

Вывод шаблона Pattern Generator'a в режиме графической симуляции

1. Задайте схему обычным образом.
2. Поставьте пробники на схему в интересующих вас точках и добавьте эти пробники на график.
3. Щёлкните правой клавишей мышки по Pattern Generator на схеме и вызовите диалог *Edit Component*.
4. Сконфигурируйте опции *Trigger* и *Clock*.
5. Загрузите нужный шаблон в поле *Pattern Generator Script*.
6. Покиньте диалоговую форму Pattern Generator и нажмите пробел, чтобы запустить симуляцию.

Выводы компонента Pattern Generator



Выводы выхода данных (выход с тремя состояниями, Q0-Q7, B[0-7])

Pattern Generator может выводить либо на шину и/или на индивидуальные выводы.

Выводы тактового генератора (CLKOUT)

Когда Pattern Generator тактируется внутренним генератором, вы можете сконфигурировать этот вывод для отображения импульсов внутреннего генератора. Это задаётся, как свойство, и может изменяться через диалоговую форму *Edit Component*. По умолчанию эта опция не установлена, поскольку это слегка затрудняет выполнение, обычно на высоких частотах.

Выход Cascade

Cascade Pin переходит в высокое состояние, когда первый бит шаблона выполняется, и остаётся в высоком состоянии, пока не пришёл второй бит (на один такт позже). Это означает, что он в высоком состоянии для первого такта при старте симуляции и вновь для первого цикла последующего сброса.

Вывод триггера (вход)

Этот вход используется для подачи внешнего импульса запуска в Pattern Generator. Есть четыре режима переключения, которые обсуждаются в разделе «Режимы сброса».

Вывод CLKIN (вход)

Этот вход используется для подключения внешнего тактового генератора к Pattern Generator. Есть два режима внешнего тактирования, которые обсуждаются более подробно в разделе «Режимы тактирования».

Вывод Hold (вход)

Этот вывод, когда переходит в высокое состояние, может использоваться для остановки (паузы) Pattern Generator. Шаблон останется в той же точке, пока не будет освобождён вывод hold. Для внутреннего тактового генератора и/или триггера тактирование будет возобновлено относительно точки, в которой было приостановлено. Например, при 1Hz внутреннего генератора, если шаблон остановлен на времени 3.6 секунды и запущен в 5.2 секунды, тогда следующий спад импульса будет в 5.6 секунд.

Вывод разрешения выхода (вход)

Этот вывод (OE) должен быть установлен в высокое состояние, чтобы разрешить работу выводам выхода. Если это вывод не в высоком состоянии, тогда Pattern Generator, продолжая работать с заданным шаблоном, не будет передавать шаблон на выходные выводы.

Режимы тактирования

Внутреннее тактирование

Внутреннее тактирование переключается отрицательным фронтом. Из чего следует, что состояние импульсов будет «низкое-высокое-низкое» за один такт.

Внутреннее тактирование может быть задано либо до симуляции через диалог *Edit Component*, либо в процессе симуляции с помощью кнопки выбора режима **Clock**.

Вывод CLKOUT, когда разрешён, отображает внутренние импульсы. По умолчанию эта опция не выбрана, из-за возможных затруднений (обычно на высокой частоте) в работе, но может быть установлена через диалог Edit Component генератора шаблонов.

Внешнее тактирование

Есть два режима внешнего тактирования — отрицательным фронтом (состояния низкое-высокое-низкое) и положительным (высокое-низкое-высокое).

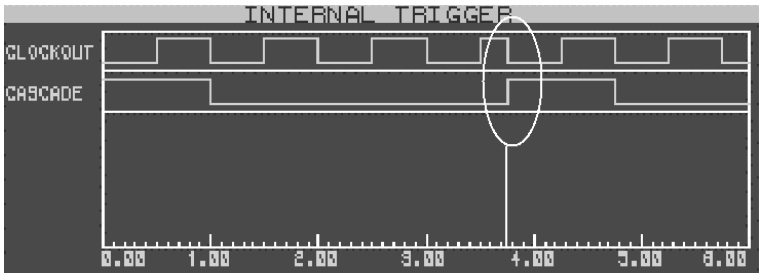
Чтобы тактировать внешним генератором, соедините выходные импульсы с выводом CLKIN и выберите один из двух режимов.

Как и при внутреннем тактировании, вы можете изменить режим либо редактированием компонента до симуляции, либо с помощью кнопки выбора режима **Clock** при паузе в симуляции.

Режимы переключения

Внутренний триггер

Режим Internal Trigger Mode (внутреннее переключение) в Pattern Generator переключает шаблон в заданных интервалах. Если тактирование внутреннее, тактовый импульс в этот момент сбрасывается. Это поведение показано на рисунке ниже.

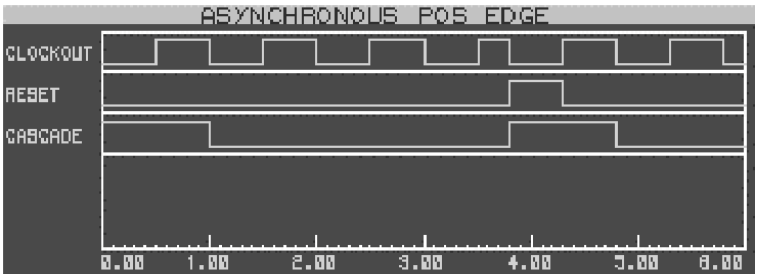


Внутренний генератор работает на частоте 1Гц и время внутреннего триггера установлено в 3.75 сек. Вывод Cascade в высоком состоянии, когда первый бит шаблона поступает на выходы и в низком всё остальное время.

Заметьте, что в момент переключения внутренний тактовый генератор сбрасывается. Первый бит шаблона выводится на выходные выводы (как показывалось при переходе вывода Cascade в высокое состояние).

Переключение внешним асинхронным положительным фронтом

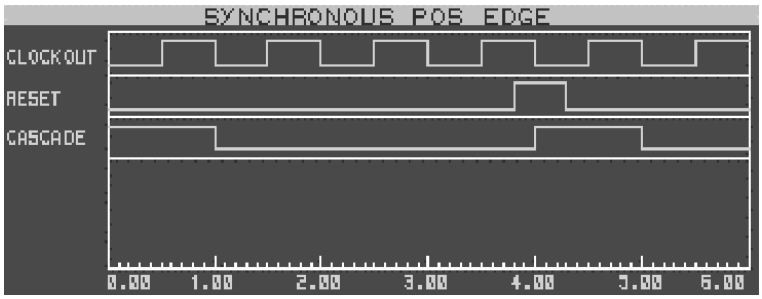
Переключение задаётся переходом положительным фронтом на выводе Trigger. Переключение происходит немедленно и следующий фронт тактового импульса будет восходящий в момент bitclock/2, следующий за временем сброса, как показано ниже.



Внутренний генератор работает на частоте 1Гц и вывод триггера переходит в высокое состояние в момент 3.75 сек. Немедленно на положительном фронте вывода триггера тактовый генератор сбрасывается и первый бит шаблона появляется на выходах.

Переключение внешним синхронным положительным фронтом

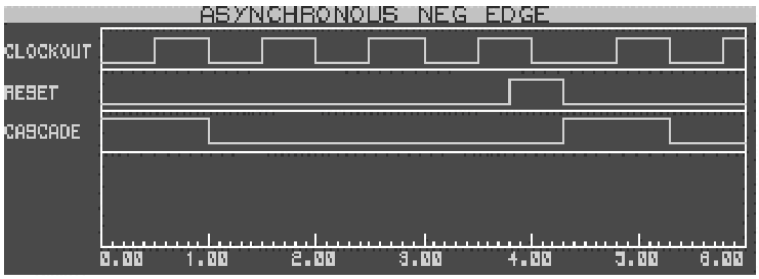
Переключение задаётся переходом положительным фронтом на выводе Trigger. Триггер защёлкивается и будет синхронизирован следующим спадающим фронтом тактового импульса, как показано ниже.



Внутренний генератор работает на частоте 1Гц. Заметьте, что на тактовом генераторе не сказывается работа триггера, и что переключение на спадающем фронте такта следует за положительным фронтом импульса.

Переключение внешним асинхронным отрицательным фронтом

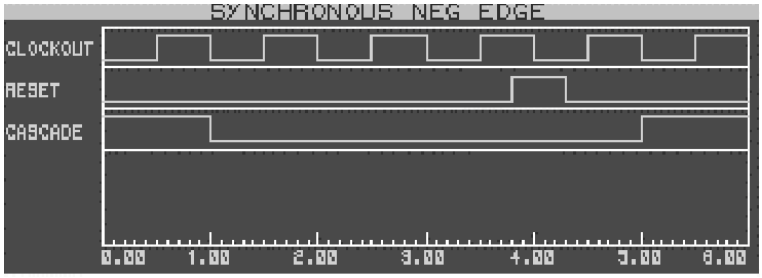
Триггер устанавливается в режим переключения отрицательным фронтом на выводе Trigger. Переключение происходит немедленно и первый бит шаблона выводится на выходные выводы.



Внутренний генератор работает на частоте 1Гц. Можно видеть, что тактовый генератор сбрасывается при отрицательном фронте импульса переключения, а первый бит шаблона выводится в этот момент.

Переключение внешним синхронным отрицательным фронтом

Переключение задано отрицательным фронтом на выводе Trigger. Триггер защёлкивается и действует синхронно со следующим спадающим фронтом тактового генератора, как показано ниже.

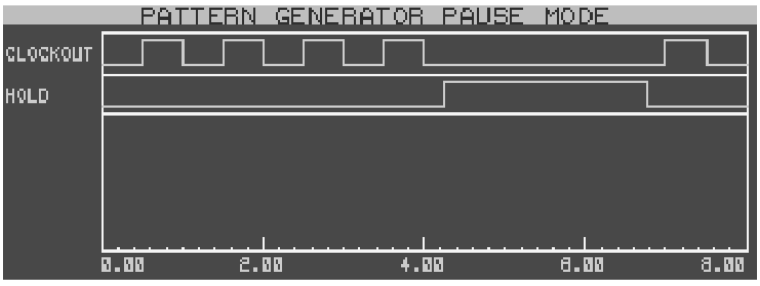


Внутренний генератор работает на частоте 1Гц. Заметьте, что переключение имеет место на спаде импульса переключения, и шаблон не сбрасывается, пока следом не приходит спадающий фронт импульса тактового генератора.

Внешнее удержание

Удержание шаблона в его текущем состоянии

Если вам нужно удержать шаблон на некоторое время, вы можете сделать это с помощью вывода Hold, который должен быть установлен в высокое состояние на время паузы, которая вам нужна. Перевод вывода Hold в низкое состояние синхронно вернёт движение шаблона, если вы используете внутренний тактовый генератор. Вместе с тем, если сигнал на выводе Hold переходит в высокое состояние на середине тактового цикла, тогда при его переходе в низкое состояние следующий бит появится на выходных выводах на полтакта позже.

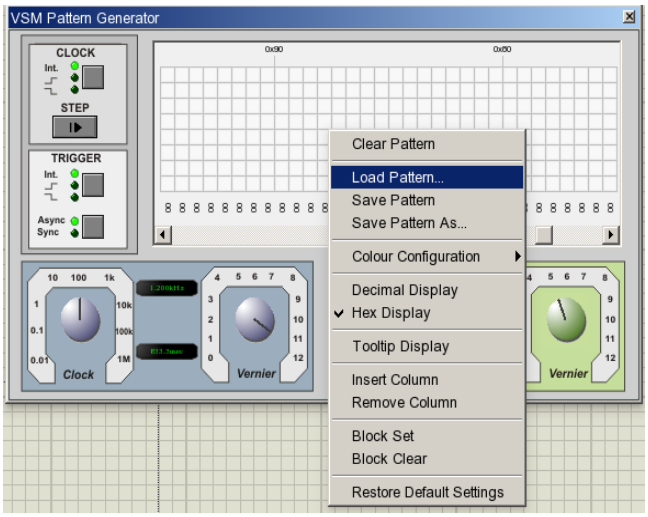


Когда сигнал на выводе Hold переходит в высокое состояние, внутренний тактовый генератор останавливается. При переходе вывода Hold в низкое состояние тактовый генератор запускается относительно точки в цикле, где он был остановлен.

Дополнительная функциональность

Загрузка и сохранение скрипта шаблона

Pattern Scripts могут загружаться или сохраняться, если щёлкнуть правой клавишей мышки по сетке при интерактивной симуляции и выбрать команду из выпадающего меню.



Это полезно, когда выполненный шаблон предполагается использовать в других проектах.

Скрипты шаблонов — это просто текст и простой, разделённый запятыми, список байт, где каждое значение байта представляет колонку на сетке. Любая линия, начинающаяся с точки с запятой, трактуется как линия комментария и игнорируется анализатором программы. По умолчанию формат байта шестнадцатеричный, хотя, если вы создаёте свой собственный скрипт, вы можете вводить десятичные значения, двоичные или шестнадцатеричные.

Установка специальных значений для шкал

Вы можете задавать внешние значения и для бит, и для частоты триггера после двойного щелчка мышки по соответствующей шкале. Этим вызывается появление плавающего окна редактирования, в которое вы можете ввести значение. По умолчанию введённое значение рассматривается как частота, но вы можете задать значение в секундах или как дробь, добавив подходящий суффикс к значению (sec, ms и т.д.). Дополнительно, если вы захотите, чтобы триггер точно работал кратно битам тактовой частоты, вы можете добавить суффикс «bits» к нужному множителю (то есть, 5bits).

Для подтверждения ввода нажмите клавишу **Enter** или, если решили отказаться от ввода, нажмите клавишу **Escape** или, просто, щёлкните где-нибудь в окне Pattern Generator.

Эти значения можно также задать до симуляции через соответствующие свойства в диалоговой форме *Edit Component*.

Установка специальных значений для сетки шаблона

Вы можете задать специальные значения для любой колонки на сетке, щёлкнув левой клавишей мышки по тексту, отображающему текущее значение для этой колонки. Появится плавающее текстовое окно, в которое вы можете вписать нужное значение. В можете задать значение в десятичном (то есть, 135), шестнадцатеричном (0xA7) или двоичном (0b10110101) виде.

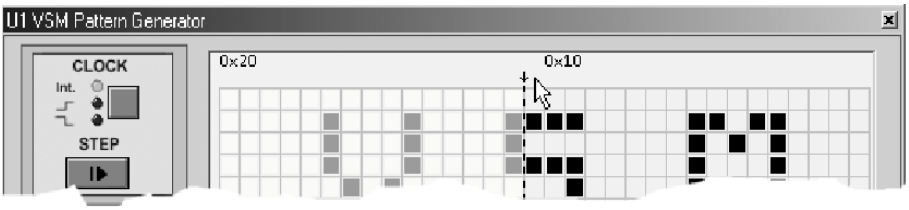
Для подтверждения ввода нажмите клавишу **Enter**, для отказа клавишу **Escape** или щёлкните левой клавишей мышки где-нибудь в окне *Pattern Generator*.

Для удобства вы можете использовать задание в колонке через горячие клавиши **CTRL+1**

клавиатуры и очистить колонку клавишами **CTRL+SHIFT+1**, когда мышка находится на нужной колонке.

Задание длины периода шаблона вручную

Вы можете вручную задать период, щёлкнув левой клавишей мышки над сеткой и колонкой, где вы хотели бы, чтобы шаблон заканчивался. Чтобы снять период, щёлкните правой клавишей мышки один раз в том же месте. Всё это показано ниже.



Мышка расположена в области, где может быть задан нужный период. Мы можем видеть, что метка периода показывает точку прерывания шаблона, и что всё слева от выбранного места побледнело, чтобы показать, шаблон будет остановлен у метки периода.

Пошаговое выполнение в продвижении по шаблону

Кнопка **STEP** может использоваться для продвижения при симуляции на время периода, эквивалентное заданному bitclock либо внутреннего, либо внешнего тактового генератора. Симуляция будет идти, пока не завершится следующий цикл тактового генератора, а затем вновь остановится.

Изменение режима отображения сетки

Отображение сетки может переключаться между шестнадцатеричным и десятичным режимом. Это можно сделать, либо щёлкнув правой клавишей мышки по сетке и выбрав нужную опцию из меню, либо используя горячие клавиши **CTRL+X** (шестнадцатеричное отображение) и **CTRL+D** (десятичное отображение).

Задание выхода

Выделите элемент схемы *Pattern Generator*, чтобы войти в диалог *Edit Component*. Свойство внизу (*Output Configuration:*) позволяет вам конфигурировать, как вы хотите видеть шаблон: и на шине, и на выводах; только на выводах; только на шине.

Дисплей указателя контекстного окна

Вы можете разрешить указатель контекстного окна, который будет следовать за мышкой, показывая информацию о текущий строке и столбце. Это можно переключать, включая и выключая, либо через контекст мышки по щелчку правой клавиши, либо через горячие клавиши **CTRL+Q**. Заметьте, что режим указателя отменяется в процессе *Block set* или *clear*.

Редактирование блока

Вы можете использовать команды Block Set и Block Clear, что поможет вам быстро сконфигурировать сетку к нужному шаблону. Это доступно через контекстное меню по щелчку правой клавиши мышки или через горячие клавиши (**CTRL+S** для *Block set* и **CTRL+C** для *Block clear*). Заметьте, что команды Block Editing запрещены в режиме указателя (tooltip mode).

ПОЛЬЗОВАТЕЛЬСКИЕ ЭЛЕМЕНТЫ ИНТЕРФЕЙСА VSM

Дисковые шкалы

VSM Virtual Instruments используют управляемые мышкой дисковые шкалы (ручки) для настройки некоторых параметров. Процедура настройки следующая.

Чтобы установить дисковую шкалу:

1. Установите курсор где-нибудь на шкале.
2. Нажмите левую клавишу мышки и удержите её.
3. Переместите указатель мышки от этой точки вокруг центра шкалы, выписывая дугу, чтобы повернуть ручку к нужному значению шкалы.
4. Шкала будет поворачиваться на угол, следующий за указателем мышки от её центра. Чем дальше вы будете перемещать мышку, тем на большее число градусов повернётся ручка.
5. Отпустите клавишу мышки, чтобы зафиксировать положение шкалы.

РАБОТА С МИКРОПРОЦЕССОРАМИ

ВВЕДЕНИЕ

Комбинация из смешанного режима симуляции и анимации схемы становится наиболее привлекательна, когда используется в контексте систем, основанных на микроконтроллерах. Множество таких систем затрагивают интерфейс пользователя, или, хотя бы, комплексную последовательность внешних событий, которые не могут легко симулироваться вне интерактивного окружения, и PROTEUS VSM в первую очередь был создан, адресуясь к этим проблемам. Следовательно, значительный кусок его функциональности сосредоточен вокруг разработки на основе микропроцессоров.

На практике, редактирование и компиляция исходного кода, интегрируется в окружение проекта так, чтобы вы могли редактировать код и просматривать результат внесённых изменений максимально легко. Исходный файл может быть вызван для редактирования двумя нажатиями клавиш, а симуляция возобновлена с этого места двумя другими.

СИСТЕМА УПРАВЛЕНИЯ ИСХОДНЫМ КОДОМ

Обзор

Система управления исходным кодом поддержана двумя основными функциями:

- Регистрацией файлов исходного кода в ISIS, так чтобы они могли быть вызваны для редактирования без ручного переключения на другое приложение.
- Определением правил для компиляции исходного кода в объектный код. Однажды заданные, эти правила появляются каждый раз при выполнении симуляции, так что объектный код привязывается к датам.

Заметьте, что нет необходимости использовать систему управления исходным кодом для симуляции проекта, основанного на микропроцессорах. Действительно, если вы выбрали ассемблер или компилятор, имеющий свой собственный IDE, вы можете прекрасно работать в нем, а переключаться в Proteus, когда проверяете выполнение программы. Если вы планируете работать таким образом, тогда перейдите к следующему разделу «Использование IDE других производителей».

Добавление исходного кода в проект

Чтобы добавить файл исходного кода в проект:

1. Выберите команду *Add/Remove Source Files* из раздела *Source* основного меню.
2. Выберите *Code Generation Tool* для исходного файла. Если вы планируете использовать новый ассемблер или компилятор впервые, вам нужно зарегистрировать их, используя команду *Define Code Generation Tools*.
3. Щёлкните кнопку **New** и выберите или введите имя для файла исходного кода с помощью селектора файлов. Вы можете ввести текстовое имя файла, если он не существует.

4. Установите флажки, требуемые для задания работы с этим исходным файлом в поле *Flags*. Флажки, необходимые каждый раз при использовании обычных инструментов, могут быть введены при регистрации инструмента.
5. Щёлкните **ОК** для добавления исходного файла в проект.

Не забудьте отредактировать микропроцессор и присвоить имя файлу объектного кода (обычно это hex-файл) в его *PROGRAM* свойстве. ISIS не может сделать этого автоматически, поскольку у вас может быть не один процессор на схеме!

Работа над вашим исходным кодом

Чтобы отредактировать исходный код:

1. Нажмите **ALT-S**.
2. Нажмите порядковый номер файла исходного кода в меню *Source*.

Если вы предпочитаете использовать более развитый текстовый редактор, посмотрите соответствующий раздел.

Чтобы переключиться обратно в ISIS, оттранслируйте (build) исходный код и запустите симуляцию:

1. Из текстового редактора нажмите **ALT-TAB**, чтобы вернуться в ISIS.
2. Нажмите **F12** для выполнения или **CTRL-F12** для начала отладки.

В любом случае ISIS проинструктирует текстовый редактор сохранить свои файлы, проверит дату файлов исходного и объектного кодов, и запустит подходящий инструмент трансляции для создания объектного кода.

Чтобы перетранслировать весь объектный код:

1. Выберите команду *Build All* из раздела *Source* основного меню.

ISIS вызовет все инструменты генерации, требуемые для трансляции в объектный код, безотносительно меток время/дата файлов объектного кода. Командная строка вывода инструмента будет отображена в окне. Этим превосходно поддерживается проверка, что все оттранслировано без ошибок и предупреждений.

Установка инструментов генерации кода других производителей

Некоторое количество свободных ассемблеров и компиляторов может быть установлено в директорию TOOLS с системного CD, и они будут автоматически заданы, как средства генерации кода программой установки PROTEUS. Однако, если вы хотите использовать другие инструменты, вам понадобится команда *Define Code Generation Tools* в разделе *Source* основного меню.

Чтобы зарегистрировать новый инструмент генерации кода:

1. Выберите команду *Define Code Generation Tools* в разделе *Source* основного меню.
2. Щёлкните **New** и используйте селектор файлов для указания пути к исполняемому файлу инструмента. Вы можете также зарегистрировать batch-файлы (командные файлы), как инструмент генерации кода.
3. Введите расширения для файлов исходного и объектного кода. Этим определяется тип файла, по которому ISIS будет определять при решении, запускать ли инструмент для отдельного исходного файла или нет. Если вы установили Always Build, тогда инструмент трансляции запускается всегда, а расширение объектного кода не требуется.
4. Задайте нужную командную строку для инструмента. Используйте %1 для представления файла исходного кода и %2 для файла объектного кода. Вы можете также использовать %$ для пути к директории PROTEUS и %~ для директории, в которой размещается DSN файл.

И хорошо одновременно поместить флаги командной строки, которые нужны для фонового запуска инструмента трансляции (то есть, без паузы для пользовательского ввода), и хорошо бы задать пути к директории *include* файлов заголовков процессоров и т.п.

Если вы хотите использовать уровень отладки исходного кода PROTEUS VSM, вам понадобится Debug Data Extractor для вашего ассемблера или компилятора. Это маленькая программа командной строки, которая выбирает строку объектного кода/исходного из информации кросс-ссылок списка файлов, произведённых ассемблером или компилятором. Мы надеемся поддерживать все популярные ассемблеры и компиляторы, так что проверяйте наш сайт на предмет последней информации.

Если у вас есть DDX программа, введите расширение для файла списка или символьного отладочного файла, который производится инструментом генерации кода, и щёлкните **Browse**, чтобы выбрать путь и имя файла DDX программы.

Использование программы MAKE

В некоторых случаях простые правила, встроенные в ISIS, могут быть недостаточны для обслуживания вашего приложения — особенно, если происходит выполнение множества исходных и объектных файлов, и происходит компоновка. В таких случаях вам нужно использовать внешнюю программу MAKE, и вы можете сделать это следующим образом:

Чтобы установить использование внешней программы Make в проекте:

1. Установите вашу программу Make (обычно поставляемую с ассемблер/компилятор окружением), как *code generation tool*. Задайте расширение исходника как MAK и установите флажок *Always Build*. Для типичной программы Make задайте командную строку:

   ```
   -f%1
   ```

2. Используйте команду *Add/Remove Source Files* для добавления makefile (то есть, MYPROJECT.MAK) к проекту.

3. Также добавьте файлы исходного кода, но для *Code Generation Tool* выберите <NONE>. Каждый раз при построении проекта (built), ISIS запустит внешнюю программу MAKE с файлом проекта makefile в качестве параметра. Затем идёт обращение к программе MAKE, чтобы определиться, какой следует запустить инструмент генерации кода. Хорошая make-программа подразумевает большую гибкость.

Использование редактора исходного кода других производителей

PROTEUS VSM имеет простой текстовый редактор — SRCEDIT, который можно использовать для редактирования файлов исходного кода. SRCEDIT — это модифицированная версия NOTEPAD, которая может открывать множество исходных файлов и может реагировать на запросы DDE, чтобы сохранить модифицированные буферы.

Если у вас есть более совершенный текстовый редактор, как, например, UltraEdit, вы можете проинструктировать ISIS использовать этот редактор вместо штатного. Заметьте, что IDE окружение, возможно, не будет реагировать на DDE команды, и подобное интегрирование может оказаться неподходящим. Однако вы всегда можете запросить производителя добавить эту поддержку — это не так сложно.

Чтобы установить альтернативный редактор исходного кода:

1. Выберите команду *Setup External Text Editor* из раздела *Source*.
2. Щёлкните по кнопке Browse и используйте селектор файлов для указания исполняемого файла вашего текстового редактора.
3. ISIS инструктирует текстовый редактор открывать и сохранять файлы, используя DDE протокол. Обратитесь к документации на текстовый редактор или к поставщику, чтобы разобраться с синтаксисом команд. Если вы не уверены в имени сервиса, попробуйте использовать имя продукта, то есть, ULTRAEDIT.

ИСПОЛЬЗОВАНИЕ IDE ДРУГИХ ПРОИЗВОДИТЕЛЕЙ

Большинство профессиональных компиляторов и ассемблеров имеют собственное интегрированное окружение разработки или IDE. Примеры этого — IAR Embedded Workbench, Keil uVision 2, Microchip MP-LAB и Atmel AVR studio. Если вы разрабатываете ваш код с одним из этих инструментов, вы можете обнаружить, что легче выполнить шаги редактирования и компиляции в этих IDE, а затем переключиться в Proteus VSM, но только тогда, когда вы получили исполняемый образ (то есть, HEX или COD файл), и вы готовы симулировать их.

Proteus VSM поддерживает два способа работы с внешними IDE:

- Использование Proteus в качестве внешнего отладчика — управление отладкой из ISIS, в основном, как если бы при работе с нормальной CAD симуляцией.
- Использование Proteus, как дополнительного симулятора — управление отладкой из отладчика IDE. Proteus работает, как разновидность виртуального сетевого симулятора, контактируя с IDE через TCP/IP. В этом режиме вы запускаете ваш IDE отладчик на одном компьютере, а симулятор Proteus на другом.

Использование Proteus VSM в качестве внешнего отладчика

Для использования Proteus в качестве внешнего отладчика требуется, чтобы формат символьной отладки, произведённый вашим компилятором, поддерживался как один из доступных для загрузки в Proteus. Загрузчик извлекает адреса каждой строки исходника в программу языка высокого уровня, и, где возможно, локализацию программных переменых.

Общие форматы отладчика — это COD (используемый в мире PIC), UBROF — для всех компиляторов IAR и OMF (используемый для 8051). Мы также предоставляем загрузчики для других коммерческих форматов, как файлы списков, производимых Crownhill PICBasic. Загляните на наш сайт в раздел поддержки «3 rd Party Compilers», где есть последняя информация.

При условии, что загрузчик выбранного вами компилятора есть, процедура загрузки программы в симулируемый микропроцессор совсем проста.

Чтобы загрузить программу, произведённую во внешнем IDE:

1. Убедитесь, что программа откомпилирована и скомпонована без ошибок.
2. Отредактируйте свойство PROGRAM модели CPU, чтобы было имя образа выполняемого файла, произведённого компилятором или компоновщиком, то есть, MYPROG.COD.

   Не вводите имя исходного файла — Proteus VSM не симулирует «C» или «ASM» файлы; CPU модели загружают и выполняют двоичные машинные коды.
3. Нажмите кнопку **PLAY** на панели управления анимацией, чтобы начать симуляцию в реальном времени или нажмите кнопку STEP, чтобы открыть инструкцию первого уровня исходника. В последнем случае появится Source Window (окно исходника) и вы можете начать пошаговую отладку вашего кода.

Использование Proteus VSM в качестве виртуального встроенного эмулятора (ICE)

К моменту написания только Keil uVision 2 для 8051поддерживает виртуальный ICE, хотя мы работаем с IAR, Microchip and Atmel, да и другими, чтобы интегрировать с ними Proteus VSM. Это быстро развивающаяся область, и мы настоятельно рекомендуем заглядывать на наш сайт в раздел «3 rd Party Compilers» за последней информацией и документацией. Инструкции по использованию установок uVision2 для работы с Proteus можно найти именно там.

ВСПЛЫВАЮЩИЕ ОКНА

Большинство моделей микропроцессоров, написанных для PROTEUS VSM, будут создавать несколько всплывающих окон, которые могут отображаться и скрываться с помощью раздела *Debug* основного меню. Эти окна есть трёх основных типов:

- *Status Windows* (окна состояния) — модель процессора будет, как правило, использовать одно из них для отображения значений его регистров.
- *Memory Windows* (окна памяти) — обычно есть одно из них для каждой области памяти в архитектуре процессора. Устройства памяти (RAM и ROM) также создают эти окна.
- *Source Code Windows* (окна исходного кода) — по одному из них будет создано для каждого процессора в схеме.

Чтобы отобразить всплывающее окно:

1. Запустите режим отладки, нажав **CTRL-F12**, или, если он уже запущен, щёлкните на кнопку **Pause** панели анимации (Animation Control Panel).
2. Нажмите **ALT-D**, а затем порядковый номер требуемого окна в разделе *Debug* основного меню.

Эти типы окон могут отображаться только тогда, когда симуляция приостановлена и скрываются автоматически, когда она запущена, чтобы дать вам лучший доступ к активным компонентам на схеме. Когда симуляция приостановлена (вручную или с помощью точки останова, breakpoint) окна, которые были показаны, будут вновь открыты.

Все окна отладки имеют контекстное меню — если вы поместите курсор мышки на окно и нажмёте правую клавишу мышки, появится меню, в котором вы можете управлять появлением и форматом данных в этом окне.

Положение и видимость окон отладки сохраняются автоматически в PWI файле с тем же именем, что и текущий проект. Файл PWI также содержит положение любых точек останова, которые были заданы, и содержание окна наблюдения (watch window).

ОТЛАДКА ИСХОДНОГО КОДА ВНУТРИ PROTEUS VSM

Обзор

PROTEUS VSM поддерживает отладку исходного кода через использование загрузчика отладки для поддерживаемых ассемблеров и компиляторов. Текущий набор загрузчиков отладки содержится в системном файле LOADERS.DLL, а количество инструментальных средств, поддерживаемых ISIS, растёт довольно быстро. Последняя информация на это счёт доступна на сайте в разделе «3 rd Party Compilers».

Если вы используете поддерживаемый ассемблер или компилятор, PROTEUS VSM создаёт окно исходного кода для каждого исходного файла в проекте, и эти окна появятся в меню *Debug*.

Окно исходного кода

Окно исходного кода имеет некоторые особенности:

- Синее выделение представляет текущую строку, на которой можно установить точку останова (breakpoint), нажав **F9** (Toggle Breakpoint), и до которой дойдёт программа, если вы нажмёте **CTRL-F10** (Step To).
- Красный указатель показывает текущее положение программного счётчика процессора.
- Красная окружность маркирует линию, на которой была установлена точка останова.

Контекстное меню, вызываемое правой клавишей мышки, поддерживает ряд опций, включающий: *Goto Line, Goto Address, Find Text* и переключение отображения номеров строк, адресов и байт объектного кода.

Единичные шаги

Поддерживается ряд опций для пошагового режима, все они доступны на инструментальной панели окна исходного кода или из раздела *Debug* основного меню.

- *Step Over* — продвижение вперёд по одной строке, пока строка не инструкция вызова подпрограммы, в этом случае вся подпрограмма выполняется.
- *Step Into* — выполнение одной инструкции кода. Если окно исходного кода не активировано, выполняется инструкция одного машинного кода. Обычно это одно и то же, если вы не отлаживаете на языке высокого уровня.
- *Step Out* — выполняется до возврата из подпрограммы.
- *Step To* — выполняется, пока программа не достигнет выделенной строки. Эта опция доступна только при активном окне исходного кода.

Заметьте, что, исключая Step To, пошаговые команды будут работать без активации *окна исходного кода*. Это возможно, хотя и не так легко, использовать для отладки исходного кода, сгенерированного программой, для которой нет поддержки загрузки.

Использование точек останова (Breakpoints)

Точки останова предлагают очень мощное средство выявления проблем в программе или программно-аппаратном взаимодействии в проекте. Обычно, вы устанавливаете точки останова в начале подпрограммы, вызывающей проблемы, начинаете выполнение симуляции, и затем работаете с проектом, пока программа не достигнет точки останова. Здесь симуляция приостанавливается. И теперь вы можете пошагово пройти код программы, проверяя значения регистров, положение в памяти и другие условия в схеме, пока вы движетесь. Обращение к *Show Logic State of Pins effect* также может быть очень поучительно.

Когда *окно исходного кода* активно, точки останова могут устанавливаться или сбрасываться на текущей строке нажатием на клавишу **F9**. Вы можете устанавливать точки останова только на строке, имеющей объектный код.

Если исходный код меняется, PROTEUS VSM попытается переставить точку останова, основываясь на адресах подпрограммы в файле и на совпадении с байтами объектного кода.

Очевидно, если вы меняете код радикально, это может увести «в сторону», но обычно это работает довольно хорошо, и вам нет нужды об этом думать.

Окно переменных (Variables Window)

Большинство загрузчиков, приходящих с Proteus VSM, способны извлечь положение программных переменных, как и адреса номеров строк исходного кода. Когда это возможно, Proteus отображает *окно переменных* вместе с *окном исходного кода*.

Есть некоторые особенности, на которые следует обратить внимание, относящиеся к окну переменных:

- При пошаговой отладке любые переменные, которые меняют значение, подсвечиваются.
- Формат, в котором каждая переменная отображается, может быть настроен, если щёлкнуть правой клавишей мышки по переменной и выбрать альтернативный формат из контекстного меню.
- Хотя окно переменных скрыто, пока программа работает, вы можете drag & drop (взять и перетащить) переменные в *окно наблюдения*, где они будут оставаться видимы.
- В зависимости от того, как компилятор «переиспользует» память, локальные переменные, выходящие за границы переменных, могут отображать неверные значения.

ОКНО НАБЛЮДЕНИЯ (WATCH WINDOW)

В то время, как *окно памяти* и *окно регистров* принадлежат модели процессора, и отображаются только во время паузы симуляции, *окно наблюдения* существует для отображения значений, которые обновляются в реальном времени. Это также подразумевает присваивание имён индивидуальным областям памяти, которые могут легче обнаруживаться, чем при поиске в *окне памяти*.

Для добавления объекта в окно наблюдения:

1. Нажмите **CTRL-F12**, чтобы начать отладку или приостановите симуляцию, если она уже запущена.
2. Отобразите *окно памяти* (memory window), содержащее объект для наблюдения, и окно наблюдения и окно наблюдения, используя нумерованные опции в разделе *Debug* основного меню.
3. Отметьте позицию в памяти или диапазон памяти, используя левую клавишу мышки. Выделенный диапазон появляется в инверсных цветах.
4. Перетащите выделенный объект(ы) из *окна памяти* в *окно наблюдения*.

Вы можете также добавлять объекты в *окно наблюдения*, используя его команду *Add Item* из контекстного меню после щелчка правой клавишей мышки.

Модификация объектов в окне наблюдения

Получив объект или объекты в окне наблюдения, вы теперь можете выбрать объект левой клавишей мышки, а затем:

- Переименовать его, нажав **CTRL-R** или **F2**.
- Изменить размер данных по любой из опций, полученных из контекстного меню после нажатия правой клавиши мышки. Для размеров данных, которые содержат несколько байт (то есть, 16 или 32 битовых слов или строки), второму и последующим байтам присваиваются следующие за объектом адреса. Следовательно, для отображения многобайтных слов или строк вам нужно только перетащить первый байт из *окна памяти*.
- Изменить формат чисел на двоичный, восьмеричный, десятичный или шестнадцатеричный.

ТРИГЕРЫ ТОЧЕК ОСТАНОВА

Обзор

Некоторые компонентные объекты поддерживают то, какой триггер приостанавливает симуляцию, когда достигаются особые условия. Что особенно полезно, когда симуляция комбинируется с пошаговым режимом, поскольку схема может симулироваться нормально до выполнения частных условий, после чего можно продолжить работу в пошаговом режиме, чтобы увидеть непосредственно, что происходит дальше.

Устройства переключения точек останова можно найти в библиотеке REALTIME устройств.

Триггер напряжения точки останова — RTVBREAK

Это устройство доступно с одним и двумя выводами, и переключает точку останова, когда напряжение на его единственном выводе или напряжение между двумя выводами больше заданного значения. Вы можете связать это устройство с произвольно управляемым источником напряжения (arbitrary controlled voltage source, AVCS) для переключения точки останова при сложных, задаваемых формулами условиях, содержащих множество напряжений, токов и т.д.

Когда напряжение превысило напряжение срабатывания, устройство не повторит переключение до тех пор, пока напряжение не опустится ниже порогового и не повысится вновь.

Триггер тока точки останова — RTIBREAK

Это устройство имеет два вывода и переключает точку останова, когда ток, проходящий через них, больше заданного значения.

Когда ток превысил ток срабатывания, устройство не повторит переключение до тех пор, пока ток не опустится ниже порогового и не повысится вновь.

Цифровой триггер точки останова — RTDBREAK

Это устройство доступно с разным количеством выводов, и вы можете его менять, если это нужно. Переключение точки останова происходит, когда двоичное значение на его входах эквивалентно значению, присвоенному компоненту. Например, задание значения RTDBREAK_8

```
0x80
```

приведёт к переключению, когда D7 в высоком состоянии, а D0-D6 в низком.

Когда условия переключения наступают, устройство не повторит переключения, пока напряжение с другим входным значением не будет получено.

СИМУЛЯЦИЯ НА БАЗЕ ГРАФИКОВ

ВВЕДЕНИЕ

Хотя интерактивная симуляция имеет много преимуществ, остаётся ещё ряд ситуаций, в которых полезнее провести симуляцию в графиках и изучить полученные результаты без спешки. В частности, возможно увеличивать отдельные события симуляции и получить возможность более детального измерения. Симуляция, основанная на графиках, также единственный способ проведения анализа, который нельзя осуществить в реальном времени, как, например, малосигнальный анализ (small-signal AC analysis), анализ шумов (Noise Analysis) и измерения при развёртке параметров (swept parameter).

Симуляция, основанная на графиках, не доступна в PROTEUS Lite.

УСТАНОВКА СИМУЛЯЦИИ НА БАЗЕ ГРАФИКОВ

Обзор

Симуляция, основанная на графиках, выполняется в пять основных этапов. Всё это суммировано ниже с детальными пояснениями на каждом этапе в последующих разделах:

- Ввод схемы для симуляции.
- Размещение сигнал-генераторов в точках, требующих стимулов, и пробников в точках, которые требуют проверки.
- Размещение графиков, соответствующих типу анализа, который вы хотите выполнить — то есть, например, частотный график для отображения АЧХ.
- Добавление генераторов и пробников на график, чтобы отобразить данные, которые они генерируют/фиксируют.
- Установка параметров симуляции (таких, как время выполнения) и выполнение симуляции.

Ввод схемы

Ввод схемы, которую вы хотите симулировать, точно такой же, что и в других проектах ISIS; техника выполнения этого детально описана в руководстве к ISIS.

Размещение пробников и генераторов

Второй этап процесса симуляции — это установка сигнал-генераторов в точках, требующих стимулов, и пробников в точках проверки. Установка сигнал-генераторов и пробников совершенно проста, поскольку выполняется подобно тому, как это делается для других компонентов, контактов или точек соединения в ISIS. Всё, что нужно сделать, это выбрать подходящую иконку, указать тип генератора или пробника в *окне селектора* и поместить объект на схему там, где вам нужно. Это можно сделать непосредственно на существующий провод, или поступить, как с другими компонентами — поместить в рабочее поле, а затем соединить с нужной точкой схемы. Определение сигнала, производимого генератором, это предмет редактирования свойств объекта и задания требуемых установок в его диалоге.

На этом этапе вы можете также изолировать часть проекта, так что только некоторые компоненты будут вовлечены в симуляцию. Это выполняется установкой флажков *Isolate Before* (изолировать до) и *Isolate After* (изолировать после) в пробниках и генераторах. Использование пробников и генераторов таким путём не только ускоряет симуляцию, но также означает, что ошибки из-за удалённых проводов, о которых забыли и заменили, не будут вкрадываться в результаты.

Размещение графиков

Третий этап процесса симуляции — это определение, какой тип анализа или типы вы хотите выполнить. Типы анализа включают аналоговый и цифровой анализ переходных процессов, частотный анализ, анализ «качания» параметров и т.д. В ISIS определение типа анализа — это синоним размещения объекта график требуемого типа анализа. И вновь, поскольку график подобен большинству других объектов в ISIS, его размещение сводится к выбору подходящей иконки, выбору требуемого типа графика и размещение его в проекте рядом со схемой. Но не только так можно увидеть несколько типов анализа одновременно, при подготовке можно использовать метод «drag-and-drop, перетаскивания», принятый в ISIS, что даёт дополнительные удобства, и вы можете увидеть (и вывести в виде твёрдой копии) графики рядом со схемой, которая генерирует их.

Добавление кривых на графики

После размещения одного или больше графиков, вы должны задать, какие данные пробников/генераторов вы хотите видеть на графиках. Каждый график отображает несколько кривых. Данные для построения кривой обычно получены от единственного пробника или генератора. Однако ISIS предлагает для кривой, отображающей данные, до четырёх отдельных пробников/генераторов, скомбинированных математически с помощью *Trace Expression* (выражение для кривой). Например, кривая может быть задана для отображения данных, полученных от пробника напряжения и тока (оба контролируют ту же точку), так что эффективно отображается мощность в контрольной точке.

Задание кривых, отображаемых на графике, может быть сделано несколькими путями: вы можете выделить и перетащить пробник на график или можете выделить несколько пробников/генераторов и добавить их все на график за одну операцию, или для кривых, требующих выражений (trace expressions), вы можете использовать диалоговую форму для выбора пробников и задания выражений.

Процесс симуляции

Симуляция, основанная на графиках, это *Demand Driven* (вывод по запросу). Что означает, что акцент сделан на установке генераторов, пробников и графиков в плане определения, что вы хотите измерять, а не на установках симуляции с последующим, какого-либо рода, постпроцессором, который проверял бы результаты. Любые параметры, специфические для для запуска данной симуляции, задаются либо редактированием подходящих свойств самого графика (то есть, время начала и окончания для симуляции) или присвоения дополнительных свойств графику (то есть, для цифровой симуляции вы можете передать симулятору «randomise time delays, случайные временные задержки»).

Итак, что происходит, когда вы начинаете симуляцию? Вкратце, действие проходит через следующие шаги:

- *Netlist generation* (генерация спецификации схемы) — это обычный процесс трассировки соединений от вывода к выводу и создание списка компонентов, а также списка групп соединений выводов или сетей (nets). Вдобавок, любые компоненты в проекте, которые должны симулироваться файловыми моделями, заменяются компонентами, содержащимися в этих файлах.
- *Partitioning* (разбиение) — ISIS просматривает точки, где вы поместили пробники и обратные пути от них к тем, где вы внедрили сигналы. Результатом этого анализа станет создание одной или более частей, которые могут нуждаться в симуляции, а, следовательно, которые должны симулироваться. Когда это произойдет, результаты сохраняются в новом файле этих частей.
- *Results Processing* (результаты обработки) — наконец ISIS берёт файлы частей для построения разных кривых на графике. График после этого обновляется и может быть максимизирован для измерения и т.д.

Если в процессе выполнения любого из этих этапов обнаруживаются ошибки, тогда детали этого записываются в файл журнала симуляции (simulation log). Некоторые ошибки фатальны, а другие требуют только предупреждения. В случае фатальных ошибок, запись журнала отображается для немедленного обозрения. Если обнаруживаются только предупреждения, тогда график обновляется, а за вами остаётся выбор, хотите ли вы видеть запись в журнале (log). Большинство ошибок относятся либо к плохо нарисованной схеме (которая не может, по тем или иным причинам, математически обработаться), либо к пропущенным или некорректно присоединённым файлам моделей.

ОБЪЕКТ ГРАФИК

Обзор

График — это объект, который можно разместить в проекте. Его назначение — управлять частичной симуляцией и отображать результаты этой симуляции. Тип анализа, выполняемый симуляцией, определяется типом размещённого графика. Часть проекта, которая симулируется, и данные, которые отображены на графике, определяются такими объектами, как пробники и генераторы, которые были добавлены на график.

Текущий график

Все специфические команды, относящиеся к графику, находятся в разделе меню *Graph*. В нижней части этого меню также содержится список всех графиков дизайна, с текущим графиком, отмеченным маленьким маркером слева от имени. Текущий график — это последний график, который симулировался или был выполнен командой.

Любые команды из меню Graph могут быть выполнены для заданного графика, если указать на него мышкой и использовать горячие клавиши (показанные в меню справа от команды). Если указатель мышки не находится на графике, или если вы выбрали команду непосредственно из меню, команда будет выполнена для текущего графика.

Размещение графика

Чтобы разместить график:

1. Получите список типов графиков, выбрав иконку *Graph Mode*. Список типов графиков отображён в *окне селектора*.
2. Выберите тип графика, который вы хотите разместить из *окна селектора объектов*.
3. Поместите указатель мышки в *окне редактора* в точке, где вы хотели бы видеть верхний левый угол графика. Нажмите левую клавишу мышки и «растащите» прямоугольник такого размера, какой вам нужен для графика, а затем отпустите клавишу мышки.

Редактирование графиков

Все графики могут быть перемещены, масштабированы или отредактированы, чтобы изменить их свойства, используя стандартную технику редактирования ISIS.

Свойства графика можно изменить через диалог *Edit...*, вызываемый, как любой объект в ISIS, либо первоначальным выделением с последующим щелчком левой клавишей мышки по графику, либо указав на него мышкой и нажав **CTRL+E** на клавиатуре.

Добавление кривых к графику

Каждый график отображает одну или более кривых. Каждая кривая обычно отражает данные, ассоциированные с одним генератором или пробником. Однако для аналоговых или смешанных типов графиков есть возможность отобразить на отдельном графике данные между одним и четырьмя пробниками, скомбинированными математической формулой, которую мы назвали *trace expression* (выражение кривой).

Каждая кривая имеет этикетку, которая отображается возле оси y, к которой относится кривая. Некоторые типы графиков имеют только одну ось y, и нет опции, чтобы соединить кривую с отдельной осью y. По умолчанию, имя новой кривой то же, что и имя пробника (или первого пробника в выражении кривой) — это можно изменить, отредактировав кривую.

Кривые могут быть определены тремя путями:

- Перетаскиванием отдельных генераторов или пробников на соответствующий график.
- Выделением пробников и использованием инструмента *Quick Add* (быстрое добавление) команды *Add Trace* (добавить кривую).
- Использованием диалоговой формы команды *Add Trace*.

Первые два метода легче в использовании, но ограничивают вас в добавлении новых кривых для каждого генератора или пробника, заданных на графике. Третий метод кажется более сложным, но даёт больше возможностей в контроле над типом кривой, добавленной на график. На практике кривые, которые требуют выражений, должны добавляться к графику через диалоговую форму *Add Trace*.

Чтобы «перетащить» пробник или генератор на график:

1. Выделите генератор или пробник, который вы хотите добавить на график.
2. Щёлкните и удержите левую клавишу мышки на генераторе или пробнике и перетащите пробник на график, где отпустите клавишу мышки.

Новые кривые создаются и добавляются на график; новая кривая отображает данные, ассоциированные с индивидуальным пробником/генератором. Заметьте, что любая существующая кривая может уменьшиться, чтобы график приспособился к новой кривой.

Для графиков с двумя осями y, добавление пробника или генератора на левую или правую половину графика связывает новую кривую с соответствующей осью. Более того, для смешанных, аналоговых и цифровых, типов графиков переходных процессов добавление генератора или пробника к уже существующим цифровым или аналоговым кривым создаёт новую кривую соответствующего типа (первые пробник или генератор, добавленные на график смешанного типа, всегда создают аналоговую кривую).

Чтобы быстро добавить несколько генераторов или пробников на график:

1. Убедитесь, что график, на который вы хотите перенести генераторы или пробники, это текущий график. Текущий график — это график, заголовок которого показан выбранным в меню *Graph*.
2. Выделите каждый генератор или пробник, который вы хотите добавить на график.
3. Выберите команду *Add Trace* в разделе *Graph* основного меню. В результате для выделенных пробников и генераторов команда вначале отобразит приглашение «Quick add tagged probes?».
4. Выберите кнопку **Yes**, чтобы добавить выделенные генераторы и пробники к текущему графику.

Новая кривая создаётся для каждого выделенного генератора или пробника и добавляется к графику в алфавитном порядке; каждая новая кривая отображает данные, ассоциированные со своим генератором или пробником. Кривые всегда связываются с левой осью y для тех типов графиков, которые поддерживают две оси.

Диалоговая форма команды Add Trace

Если вы не выполняли *Quick Add*, команда *Add Trace* отобразит диалоговую форму Add Trace (каждый тип графика имеет некоторые отличия). Эта форма позволяет вам выбрать имя новой кривой, её тип (аналоговая, цифровая и т.д.), ось y (левая или правая), к которой она будет добавлена, до четырёх генераторов или пробников, чьи данные она использует, и выражение, которое комбинирует данные выделенных генераторов или пробников.

Чтобы добавить кривую к графику, используя команду Add Trace:

1. Выполните команду *Add Trace* из раздела *Graph* основного меню напротив графика, к которому вы хотите добавить новую кривую.

   Если есть выделенные пробники или генераторы, последует приглашение «Quick add tagged probes?». Откажитесь, используя кнопку **No**.

2. Выберите тип кривой, которую вы хотите добавить на график, выбрав подходящий с помощью кнопки **Trace Type**. Только те типы кривых, которые допустимы для графика, будут присутствовать в списке.
3. Выберите ось y, к которой новая кривая будет относиться. Только те оси, которые допустимы для графика и выбранного типа кривой, будут присутствовать в списке.
4. Выберите одну из кнопок **Selected Probes** от P1 до P4, и затем выберите пробник или генератор из списка в окне *Probes*, чтобы связать его с выбранной кривой. Имя выбранного генератора или пробника появится рядом с кнопкой **Selected Probes** и имя *Selected Probe* (от P1 до P4) появится в поле *Expression*, если его ещё нет.

   Если выражения кривой не допустимо, будет разрешена только P1 кривая — и новая кривая будет отражать данные, ассоциированные с выбранным P1 пробником.
5. Повторите шаги от 3 до 4, пока вы выберете все генераторы и пробники, которые вам требуются для кривой.
6. Введите выражения кривой в поле *Expression*. В выражении выделенные пробники будут представлены именами P1, P2, P3 и P4, которые соответствуют выбранным пробникам рядом с кнопками **P1**, **P2**, **P3** и **P4**.

   Если выбранный тип кривой не поддерживает выражения, содержимое поля *Expression* будет игнорироваться.
7. Выберите кнопку **ОК**, чтобы добавить новую кривую на график.

Редактирование кривых графика

Индивидуальные кривые на графике можно редактировать, чтобы изменить их имена или выражения, определяющие их.

Чтобы выделить, отредактировать и снять выделение с кривой:

1. Убедитесь, что график, отображающий кривую, которую следует редактировать, не выделен.
2. Выделите кривую, щёлкнув по её имени. Выделенная кривая отображается подсвеченным именем.
3. После щелчка по имени кривой, появляется её диалоговое окно Edit Graph Trace.
4. Отредактируйте имя кривой или выражение, как это требуется. Для типа кривой, не поддерживающей выражения, любые изменения в поле *Expression* будут игнорироваться.
5. Выберите ОК, чтобы принять изменения.
6. Чтобы снять выделение с кривой, щёлкните по графику, но не по имени кривой.

Изменение последовательности и/или цвета кривой на графике

Последовательность кривых на графике может быть настроена перетаскиванием этикеток с помощью левой клавиши мышки. Кривая может быть выделена или наоборот, как вы хотите.

- Для цифровых кривых цель их перемещения, просто, в получении особой последовательности расположения.
- Для аналоговых графиков, вы можете перетаскивать кривые с левой на правую ось и назад.

  Актуальные цвета, назначенные по позиции в последовательности, могут быть реконфигурированы после максимизации графика с последующим выполнением команды *Set Graph Colours* из раздела *Template*.

ПРОЦЕСС СИМУЛЯЦИИ

Симуляция, выполняемая по требованию

Когда мы разрабатывали Proteus, одной из наших главных целей было сделать использование симуляции, как можно более интуитивным, более удобным, чем это было прежде. Множество проблем с прежними пакетами симуляции произрастало из того факта, что ряд фрагментов было написано раньше, а другие аспекты, такие как графическое отображение результатов, были добавлены позже, скорее, как запоздалые раздумья. Результатом этой тенденции стал фрагментарный вид работы, при которой вы запускаете одну программу, чтобы нарисовать схему, другую для симуляции схемы, и третью для отображения результатов.

Proteus очень отличается в этом, нарисовав схему, вы начинаете с ответа на вопрос, что вы хотите видеть, размещая и настраивая график. Этот график затем сохраняется до тех пор, пока вы не удалите его, и каждый раз, когда вы хотите видеть эффект от изменения схемы, вы только обновляете график, указав на него мышкой и нажав пробел. Мы назвали это Demand Driven Simulation (симуляция по запросу), поскольку ISIS должен работать с графиком, который действительно требует симуляции, и чтобы вам не приходилось делать этого с помощью ввода текстов. Более того, вы можете поместить несколько графиков, которые отображают разные эксперименты, и каждый «запоминает» разные установки для симуляторов.

Наиболее важное преимущество имеет место от того, что данный график определяет ряд интересующих точек в проекте через пробники, которые связаны с кривыми. ISIS затем может использовать эту информацию для выявления, какие части проекта действительно нуждаются в симуляции, вместо того, чтобы заставлять вас вручную «прошерстить» весь проект. В результате, весь проект, готовый к разводке печатной платы, может симулироваться без интенсивного редактирования, когда вам и нужно-то было симулировать только отдельные его части.

Ещё одно преимущество, нет риска, если вы забыли отменить модификации, сделанные для целей симуляции, поскольку нет такой отмены; гаджеты пробника и генератора, которые вы размещали, игнорируются при генерации netlists для разводки печатной платы (PCB layout).

Выполнение симуляций

Когда график размещён с пробниками и генераторами, связанными с ним, вы можете инициировать симуляцию, указав график и нажав пробел. ISIS определит, какие части проекта следует симулировать, с тем чтобы обновить график, запустит симуляцию и отобразит новые данные.

В процессе симуляции формируется отчёт (simulation log) о симуляции. Обычно эта запись не представляет особого интереса. Однако, если при симуляции обнаруживается ошибка, или вы запрашивали очёт о netlist при симуляции (см. «Редактирование графиков»), или результаты анализа получены в виде данных, которые не могут быть отображены графически (то есть, аналоговый анализ шумов), тогда вам нужно просмотреть запись в журнале (log) по окончании симуляции.

Чтобы обновить график и увидеть отчёт о симуляции:

1. Вначале убедитесь, что график, который вы хотите обновить, это текущий график, а затем выберите команду *Simulate* из раздела *Graph* или установите указатель мышки на график, нужный вам, и нажмите пробел.
2. По окончании симуляции, если обнаруживается ошибка, вы получите приглашение: «Load partial results?». Если вы ответите **YES**, загрузятся данные симуляции до момента возникновения ошибки и отобразятся на графике. Если вы ответите **NO**, во всплывающем окне просмотра текста отобразится отчёт о симуляции.

Если ошибок не обнаруживается или если вы выбрали **YES**, вы всё ещё можете посмотреть отчёт о симуляции либо выбрав команду *View Log* из раздела Graph основного меню, либо используя горячие клавиши **CTRL+V**.

Что происходит, когда вы нажимаете пробел

Когда вы обновляете график, либо используя команду *Simulate* из разделе *Graph*, либо с помощью горячих клавиш команды (пробел) — будет проделана большая работа по анализу и во время, и до, собственно, аналогового или цифрового анализа. Сейчас мы вкратце покажем разные шаги этого процесса:

- *Netlist compilation* (компиляция спецификации) — это обычный процесс трассировки связей от вывода к выводу и составление списка компонентов и списка групп соединения выводов или сетей. Результирующий netlist на этом этапе хранится в памяти.
- *Netlist linking* (компоновка спецификации) — некоторые компоненты моделируются с использованием подсхем (sub-netlists) или файлов моделей, которые обычно хранятся в директории, заданной в поле *Module Path* диалоговой формы команды *Set Paths*. Множество моделей приходят с Proteus, и вы можете, конечно, создавать свои собственные. Вы можете представлять себе модели, как подлисты иерархического дизайна, в котором родительский объект — это компонент, который нужно моделировать.

  Каждый раз, когда компонент моделируется таким образом, оригинальный элемент удаляется из netlist и замещается содержимым файла модели со входами и выходами модели, соединёнными в том месте, где соединялись выводы оригинального компонента.

  В конце этого процесса каждый компонент, остающийся в netlist, будет иметь свойство PRIMITIVE, которое означает, что он может непосредственно симулироваться PROSPICE.
- *Partitioning* (разбиение на части) — ISIS просматривает точки, где вы поместили пробники и обратные связи отсюда ко входным сигналам. Сигналы считаются

подключёнными к любому из: шина питания, генератор с установленным флагом *Isolate Before* или выходом записи. Этот анализ результируется в создании одной или более частей, которые нужно симулировать.

Дальнейший анализ затем работает в порядке, в котором они должны симулироваться. Например, если часть A имеет выход, который соединён со входом части B, тогда ясно, что A должна симулироваться первой. Если же выясняется, что B имеет выход, приходящий к A, тогда все теряется — и это уже к вам, используйте запись объектов таким образом, чтобы циклические зависимости не появлялись.

Всё же больше анализов устанавливается там, где есть существующие результаты, в директории, заданной через поле *Results Path* (командой *Set Paths* меню *System*), которые относятся к идентичным запускам симуляции для любого текущего набора частей. Если они есть, и если участвующие части не стимулированы теми, что сами будут заново симулироваться, тогда ISIS не спешит пересимулировать части, но использует вместо этого существующие результаты. Из чего следует, если вы работаете на другом конце проекта, нет нужды поддерживать пересимуляцию сначала.

Вы можете убедиться, что эта часть системы очень умна!

- *Simulator Invocation* (вызов симулятора) — ISIS теперь вызывает PROSPICE для каждой части в порядке выполнения действительной симуляции.

  Каждый вызов симулятора приводит к созданию нового частичного файла, а информация о прогрессе симуляции также добавляется к отчёту о симуляции, который поддерживается во время всего процесса симуляции.

- *Results Processing* (завершение процесса) — окончательно, ISIS проходит по частичным файлам, чтобы выстроить разные кривые на графике. График после этого обновляется и может быть максимизирован для проведения измерений и т.д.

Если в процессе выполнения любого из этих этапов обнаруживаются ошибки, тогда детали этого записываются в отчёт (simulation log). Некоторые ошибки фатальны, а другие вызывают только предупреждения. В случае фатальных ошибок отчёт о симуляции отображается для немедленного просмотра. Если обнаруживается только необходимость в предупреждениях, тогда график обновляется и у вас остаётся возможность посмотреть отчёт, если вы хотите этого. Большая часть ошибок относится либо к плохо сделанному чертежу (которые не могут, по тем или иным причинам, быть разрешены математически), либо из-за пропущенных или плохо связанных файлов моделей.

ТИПЫ АНАЛИЗА

ВВЕДЕНИЕ

Есть тринадцать типов графиков; каждый отображает результаты разных типов анализа схемы, поддержанных PROSPICE.

Analogue (аналоговый) Выводит напряжения и/или токи, как функцию времени, и более всего похоже на работу осциллографа. Дополнительно, могут выводиться выражения, выполняемые несколькими пробниками; например, ток может умножаться на напряжение, давая соответствующую мощность. Этот режим анализа часто называют анализом переходных процессов (Transient Analysis).

Digital (цифровой) Выводит логические уровни, как функцию времени, и более всего это похоже на логический анализатор. Кривые могут представлять единственные биты данных или бинарное значение шины.

Цифровые графики вычисляются с использованием Event Driven Simulation (событийно осуществляемая симуляция).

Mixed Mode (смешанный режим) Комбинирует оба, аналоговый и цифровой, сигналы на одном и том же графике.

Frequency (частотный) Выводит малосигнальное напряжение или ток, как функцию частоты. Известен также, как Bode plot (плоттер Боде), и может отображать и амплитуду, и фазу. Дополнительно, используя Trace Expressions, вы можете выводить входной и выходной импеданс.

Заметьте, что выведенные значения зависят от заданного входного генератора.

DC Sweep (развёртка на постоянном токе) Устоявшееся состояние рабочей точки по напряжению или току, как функция переменной развёртки. Как и в аналоговом анализе, могут выводиться выражения, комбинирующие несколько пробников.

AC Sweep (развёртка на переменном токе) Создаёт семейство частотных характеристик с одним откликом для каждого значения переменной развёртки.

Transfer (передаточный) Выводит характеристические кривые или семейство кривых, «качая» значение одного или двух входных генераторов и выводя установившиеся состояния напряжения или тока. Значение переменной первого генератора отображается по оси x; отдельная кривая производится для каждого значения второй переменной.

Noise (шумы) Входное или выходное напряжение шумов, как функция частоты. Также производит список отдельных шумовых составляющих.

Distortion (искажения) Выводит 2 и 3 гармоники искажений, как функцию частоты. Может также использоваться для вывода интермодуляционных искажений.

Fourier (Фурье)	Показывает гармонический спектр переходных процессов. Похоже на использование анализатора спектра вместо осциллографа.
Audio (аудио)	Выполняет анализ переходных процессов, а затем проигрывает результат через вашу звуковую карту. Может также генерировать Windows WAV файлы с выхода вашей схемы.
Interactive (интерактивный)	Выполняет интерактивную симуляцию и отображает результаты на графике. Этот тип анализа позволяет вам комбинировать преимущество интерактивного и графического видов симуляции.
Conformance (согласования)	Выполняет цифровую симуляцию, а затем сравнивает результаты с сохранёнными результатами предыдущего запуска. Это особенно важно при создании программных тестов для приложений на базе микроконтроллеров.

Дополнительно, все типы анализа начинаются с вычисления рабочей точки, то есть, начальных значений для всех узлов напряжения, ветвей тока и состояния переменных в момент времени равный 0. Информация, относящаяся к рабочей точке, доступна в ISIS через интерфейс *point and shoot*.

АНАЛОГОВЫЙ АНАЛИЗ ПЕРЕХОДНЫХ ПРОЦЕССОВ

Обзор

Этот тип графика представляет то, что вы могли бы увидеть на экране осциллографа. Ось x показывает продвижение по времени, а ось y отображает напряжение или ток. Мы часто ссылаемся на этот тип анализа, как на Transient Analysis, поскольку он имеет место во временной зоне.

Transient analysis, возможно, наиболее общеупотребительный из всех форм анализа. Все, что вы можете измерить с реальным осциллографом, можно измерить на графике *analogue*. Он может использоваться для проверки, что схема работает ожидаемым образом, для быстрого измерения усиления, для визуальной оценки искажений, для измерения тока от источника или проходящего через отдельный компонент и т.д.

Метод вычисления

Технически говоря, этот тип симуляции можно отнести к нелинейному узловому анализу, а это форма вычисления, используемая всеми SPICE симуляторами. В каждый момент времени каждый компонент в схеме представляется, как комбинация источников тока и/или резисторов. Это представление затем описывается системой совместных уравнений, использующих закон Ома и законы Кирхгофа, а уравнения решаются Гауссовым методом исключения. Для каждого момента времени, для которого решаются уравнения, значения источников тока и резисторов задаются законами, встроенными в модели компонентов и процесс повторяется, пока не будет получен стабильный набор результирующих значений.

Для симуляции, связанной с продвижением по времени, есть два отдельных этапа. На первом задача состоит в вычислении рабочих точек схемы — это напряжения по всей цепи в начальный момент симуляции. При этом учитывается эффект изменения времени для схемы, для чего идёт пересчёт напряжения на каждом шаге (итерируемом до схождения, как описано выше). Шаг по времени критичен для стабильности вычислений, и поэтому PROSPICE погоняет его автоматически, в рамках, определённых пользователем. Схема, которая меняется быстро, например, ряд быстро переключающихся ключей, нуждается в меньших временных шагах, а, значит, и большем количестве попыток, чем схема, которая меняется более плавно. Использование управления временными шагами и итеративный расчёт — все это добавляет вычислений, которые могут сделать transient analysis сравнительно медленным.

Поскольку алгоритм использует итерации, всегда существует вероятность, что решение не сойдётся. Чаще всего это происходит в начальной точке времени, означая, что симулятор не может установить стабильный или уникальный набор значений для рабочей точки. Временами, однако, это может обнаружиться в дальнейшем, когда обычное поведение схемы становится совершенно непредсказуемо. PROSPICE снабжён рядом техник, помогающих решить подобные проблемы, но не способных их полностью убрать. Скажем, использование в качестве основы Berkeley SPICE3F5 настолько хорошо, насколько вы можете этим воспользоваться.

Использование analogue графиков

Графики analogue могут иметь либо только левую ось y, либо и левую, и правую. Пробники могут размещаться на отдельных осях двумя способами:

- Выделив и перетащив пробник к подходящей стороне графика.
- Используя диалоговую форму *Add Transient Trace*.

См. раздел «Добавление кривых на график», где больше информации о том, как добавить кривые на существующий график.

Очень часто оказывается удобно использовать левую и правую оси y для отдельных кривых с разными единицами измерения. Например, если отображаются оба пробника: и напряжения, и тока, — тогда левая сторона может использоваться для отображения напряжения, а правая для тока. Можно размещать цифровые пробники на аналоговом графике, но *Mixed* график, как правило, удобнее.

Для выполнения анализа переходного процесса (transient analysis) аналоговой цепи:

1. Добавьте в схему генераторы, как необходимо для получения входных сигналов.

См. разделы «Размещение генераторов» и «Генераторы и пробники», где подробнее описаны разные типы аналоговых генераторов.

2. Поместите пробники на схему в тех точках, что вас интересуют. Это могут быть и точки внутри схемы, и очевидные выходы схемы.
3. Поместите *Analogue* график.
4. Добавьте пробники (и генераторы, если нужно) на график.
5. Отредактируйте график (укажите на него и нажмите **CTRL+E**, задайте требуемое

время остановки симуляции, этикетки на график и управляемые свойства, если это нужно.

6. Запустите симуляцию либо выбором команды *Simulate* раздела *Graph*, либо нажатием пробела.

Определение Analogue Trace Expressions (выражений)

Обычно вам достаточно добавить пробники напряжения и тока на график, чтобы увидеть процессы в схеме. Однако в аналоговом анализе переходных процессов можно создавать кривые, которые используют математические выражения, основанные на показаниях пробников. Например, положим, что схема имеет и пробник напряжения, и пробник тока, размещённые в схеме. Умножая значения этих пробников, мы можем нарисовать график выходной мощности в этой точке.

Чтобы отрисовать график выходной мощности:

1. Добавьте пробники тока и напряжения на выход схемы.
2. Вызовите диалоговую форму *Add Transient Trace* (нажмите **CTRL+A**) для графика.
3. Присвойте пробник напряжения P1.
4. Присвойте пробник тока P2.
5. Измените выражение (expression) на P1*P2.
6. Щёлкните по кнопке **ОК**.
7. Нажмите пробел, чтобы выполнить симуляцию.

См. раздел «Диалоговая форма команды *Add Trace*», где детально описано использование формы *Add Transient Trace*.

ЦИФРОВОЙ АНАЛИЗ ПЕРЕХОДНЫХ ПРОЦЕССОВ

Обзор

Цифровой график отображает то, что вы могли бы увидеть на экране логического анализатора. Ось x отображает время, а ось y показывает вертикальный стек сигналов. Это могут быть либо одиночный биты данных, либо двоичное представление вывода на шину.

Метод вычисления

Цифровой анализ переходных процессов использует технику, известную как событийно-ориентированная симуляция (Event Driven Simulation). Она отличается от аналоговой тем, что процесс осуществляется только тогда, когда какой-нибудь элемент схемы меняет своё состояние. Вдобавок, только дискретные логические уровни принимают в нем участие, что позволяет представить функционирование компонентов на более высоком уровне. Например, мы можем представить себе счётчик в терминах регистра значений, которые увеличиваются на единицу при каждом тактовом импульсе, а не в терминах нескольких сотен транзисторов. Это делает событийно-ориентированную симуляцию на несколько порядков быстрее, чем аналоговую симуляцию этой же схемы.

Этап загрузки (Boot Pass)

Цель этапа загрузки — определить начальные состояния всех цепей схемы, в первую очередь для выполнения симуляции. Boot pass выполняется следующим образом:

- Все входные выводы, присоединённые к цепям VCC и/или VDD, считаются имеющими высокий уровень.
- Все входные выводы, присоединённые к цепям GND и/или VSS, считаются имеющими низкий уровень.
- Все входные выводы, присоединённые к цепям, к которым присоединён генератор, считаются имеющими то же состояние, что и в свойстве *INIT* генератора.
- Все остальные выводы считаются имеющими плавающее начальное значение.
- Все модели запрашиваются (не в установленном порядке) для вычисления их входных и установки их выходных выводов соответственно. Когда каждый выходной вывод задан, состояние цепи, к которой вывод подключён, перерассчитывается.
- Все изменения состояния цепей, моделей, присоединённых к ним, запрашиваются для перерасчёта их выходов. Этот процесс продолжается, пока не находится устойчивое состояние.

Циклический процесс событий

Следом за этапом загрузки DSIM начинает собственно процесс симуляции. Симуляция выполняется в цикле, который проходит повторно через следующие два шага:

- Все события, меняющие состояние на текущий момент, прочитываются по очереди и применяются к существенным цепям. Этот процесс завершается новым набором состояний цепей.
- Там, где цепь меняет состояние, все модели, входы которых к ней присоединены, симулируются вновь. Там, где их выходы меняются, это создаёт новые события, которые помещаются в последовательность событий.

Конечно, разные модели создадут события, которые могут закончиться неудачно для обработки в разное время. Поэтому ядро DSIM должно упорядочить все новые события, созданные в конце каждого прохода цикла.

Также ценно то, что наша схема довольно удачно поддерживает модели, которые имеют нулевые временные задержки. В этом контексте события, генерируемые с одинаковым временным кодом, происходят в пакете (один пакет эквивалентен одному проходу по циклу), согласно тому, как они сгенерированы.

Условия прерывания

Симуляция останавливается, когда обнаруживается одно из следующего:

- Достигается заданное время остановки.
- Запрос о событиях становится пуст. Это означает, что схема достигла устойчиво стабильного состояния.
- Логический парадокс с нулевой задержкой времени обнаруживается так, что текущее время перестаёт меняться, несмотря на повторение циклов прохода при обработке событий в цикле.
- Системная ошибка, как выбег запросов событий за пределы области памяти. Это не обнаруживается при нормальной работе, если не возникает нечто нестабильное в вашем проекте, возможно, склонность к высокочастотной (то есть, 100 МГц) осцилляции где-нибудь.

Использование цифровых графиков

Чтобы выполнить анализ переходных процессов для цифровой схемы:

1. Добавьте генераторы, как нужно для работы схемы, чтобы запитать входы.
2. Поместите пробники на схеме в интересующих вас точках. Это могут быть как точки внутри схемы, так и очевидные выходы.
3. Поместите *Digital* график.
4. Добавьте пробники (и генераторы, если нужно) на график.

Можно использовать только *Digital* генераторы для генерации цифровых сигналов — заметьте, что обычно *Pulse* (импульсный) генератор рассматривается, как источник аналоговых импульсов.

Только цифровые пробники должны использоваться в цифровых цепях — использование токовых пробников будет заставлять PROSPICE выполнять симуляцию смешанного режима.

5. Отредактируйте график и задайте требуемое время остановки (Stop Time) симуляции, а также этикетки графика и управляемые свойства, если это требуется.
6. Выполните команду *Simulate* из раздела *Graph* основного меню напротив графика или нажмите пробел.

См. раздел «Процесс симуляции» с общей информацией о том, как симулировать графики.

См. разделы «Размещение генераторов» и «Размещение пробников» по размещению генераторов и пробников.

См. раздел «Добавление кривых на график», где описано, как добавить пробники на графики.

Как отображаются цифровые данные

Нормальные кривые

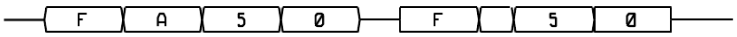
Все выходные значения от симулятора DSIM получаются в терминах шести цифровых состояний:

Тип состояния	**Ключевое слово**	**Появление на графике**
Строго высокое	**SHI**	Высокий уровень, голубой
Слабо высокое	**WHI**	Высокий уровень, синий
Плавающее	**FLT**	Средний уровень, белый
Спорное	**CON**	Средний уровень, жёлтый
Слабое низкое	**WLO**	Низкий уровень, синий
Строгое низкое	**SLO**	Низкий уровень, голубой

Вы можете, таким образом, определить состояние кривой в отдельное время, либо посмотрев на цвет и позицию линии, либо, максимизировав график, установить курсор в выбранном месте и прочитав состояние в строке состояния. Логические уровни в позиции курсора также отображаются справа от этикетки кривой.

Кривые шины

Кривые шины (результат от размещения пробника напряжения на проводе шины) отображают шестнадцатеричные значения битов шины между пересечениями линий:



Когда один или более битов шины не имеет ни высокого, ни низкого уровня, линии шины рисуются на среднем уровне. Также, если переход уровней слишком узкий для отображения шестнадцатеричного значения, тогда оно опускается. Значение ещё может быть прочитано, если на него поместить курсор.

АНАЛИЗ ПЕРЕХОДНОГО ПРОЦЕССА СМЕШАННОГО РЕЖИМА

Обзор

График смешанного режима (mixed mode) позволяет и цифровым кривым, и аналоговым отображаться над той же осью x, на которой представлено время. Анализ смешанного режима — фактически, применим тогда, когда в схеме есть и аналоговые, и цифровые модели, и график смешанного режима единственный тип, способный отображать оба вида сигналов вместе.

Метод вычисления

Смешанный режим анализа переходных процессов комбинирует вместе нелинейный узловой анализ SPICE3F5 и событийно-ориентированную симуляцию DSIM. Детальная реализация этого весьма сложна, но основной алгоритм может быть суммирован в следующем:

- Перед симуляцией каждая цепь анализируется на предмет, соединён ли с ней аналоговый, цифровой или оба типа моделей. Там, где аналоговые источники приходят на цифровые входы, PROSPICE вставляет объекты АЦП интерфейса, а там, где цифровые выходы управляют аналоговыми компонентами, вставляет объекты ЦАП интерфейса.

  Там, где несколько цифровых входов запитываются от одной цепи, создаются множественные АЦП объекты, так что переключения разных логических уровней и загружаемых характеристик может быть смоделировано для каждого входа. Аналогично, в случае, когда несколько цифровых выходов соединяются вместе, создаются множественные объекты ЦАП.

- Рабочие точки формируются так, как описано ниже.

- Симуляция затем протекает, как для обычного аналогового анализа, исключая то, что АЦП генерирует цифровые события, когда их входное напряжение пересекает порог переключения. Когда это происходит, аналоговая симуляция приостанавливается, а начинается цифровая симуляция для обработки эффекта от этих новых событий.

- Когда цифровая симуляция сказывается на изменении состояния для объектов интерфейса ЦАП, аналоговая симуляция возобновляется для тщательной симуляции около времени переключения. Фактически, выходы ЦАП моделируют время перехода (восходящее или спадающее) и несколько временных точек будет симулироваться за этот период.

Поиск рабочей точки

Обработка рабочей точки — это особый «фокус» для симуляции смешанного режима, поскольку состояние аналоговой цепи сказывается на состоянии цифровых компонентов и наоборот.

В сущности, PROSPICE делает следующее:

- Значения начальных условий (Initial condition, IC) присваиваются обоим, аналоговым и цифровым, цепям; остальные цепи считаются стартующими с нулевого напряжения.
- Узловые матрицы затем конструируются и рассчитываются, как при обычной SPICE симуляции.
- Для каждой итерации цифровая схема заново обрабатывается, чтобы изменить входы АЦП. Когда эти изменения распространяются через несколько цифровых моделей, цифровой схеме позволяется произвести итерацию до стабильного состояния.
- Объекты ЦАП заново присваивают свои выходы согласно любым изменениям в состоянии цифровой схемы. Если цифровая схема не «успокоилась», это игнорируется, поскольку это может быть кратковременным всплеском.
- Проходы через цикл продолжаются до обнаружения полной устойчивости состояния или по достижении ограничений на итерации.

Использование смешанных графиков

Смешанные (Mixed) графики используются таким же образом, что и аналоговые графики, исключая то, что вы можете добавлять на них цифровые кривые. Чтобы добавить первую цифровую кривую на смешанный график, вы должны использовать диалог *Add Trace*. Затем перетащите пробник на аналоговую часть графика, создавая новую аналоговую кривую, тогда как перетаскивание на цифровую часть создаст новую цифровую кривую.

ЧАСТОТНЫЙ АНАЛИЗ

Обзор

При частотном анализе (Frequency Analysis) вы можете видеть, как схема ведёт себя на разных частотах. Только одна частота рассматривается в один момент времени, так что вывод не похож на тот, что даёт спектральный анализ, где все частоты рассматриваются вместе. Скорее, это похоже на то, что вы подключаете генератор ко входу и следите за выходом, к которому подключён вольтметр переменного напряжения. На графике кроме амплитуды может отображаться и фаза сигнала пробника.

Частотный анализ производит частотную характеристику или диаграмму Боде (Bode plots). Это полезно, когда вы проверяете, ведёт ли себя фильтр так, как ожидается, или проверяете усилитель, работает ли он корректно в требуемой полосе частот.

Частотный график (Frequency) может также быть использован для отображения входного и выходного импеданса как функции частоты на малом сигнале.

Метод вычисления

При выполнении частотного анализа вначале находится рабочая точка схемы, затем все активные компоненты заменяются линейными моделями. Внутренние ёмкости активных устройств вычисляются в рабочей точке, принимается, что рабочее напряжение в схеме отклоняется не на много. Все генераторы, отдельно опорный генератор (см. ниже), замещаются их внутренними сопротивлениями. В этом случае линии питания эффективно соединены вместе, что нормально для частотных вычислений. Затем анализ выполняется с комплексными числами в линейном виде. Частота постепенно увеличивается от начальной до конечной с одинаковым приращением. Линейная природа этого анализа делает его более быстрым, чем анализ переходных процессов, не смотря на использование комплексных чисел.

Очень важно помнить, что частотный анализ применим к линейным цепям. Это означает, что чистый синусоидальный сигнал на входе производит чистый синусоидальных сигнал на выходе на всех частотах. Конечно, нет реальных активных цепей, которые были бы чисто линейными, но многие очень к ним близки, позволяя использовать эту форму анализа. Есть также схемы, которые в принципе нелинейны, как, например, линейные повторители с триггерами Шмитта на входе. Для нелинейных цепей термин «частотная характеристика» не имеет реального математического значения, поэтому частотная симуляция не даст значимых результатов. Если вас интересует поведение таких схем в частотной области, тогда вам больше подойдёт Фурье анализ.

Использование частотных графиков

Чтобы рассчитать амплитуду синусоидального сигнала на выходе, мы должны добавить на вход опорное синусоидальное напряжение. PROSPICE сделает это автоматически, но ему нужно знать, где вход схемы. Чтобы это сделать, вы должны задать для каждого частотного графика *Reference Generator*, чтобы дать знать симулятору, где расположен опорный сигнал. Есть три способа сделать это:

- Использовать поле *Reference* диалоговой формы *Edit Frequency Graph* для выбора входного генератора. Этот объект может быть обычным, одновыводным генератором или примитивом FREQREF.
- Выделить и перетащить любой аналоговый генератор на частотный график.
- Использовать диалоговую форму *Add Trace* для добавления генератора как REF.

Примитив FREQREF полезен для задания моделей, которые приводят в действие схему, например, модели микрофона. Он будет содержать явно названный генератор, используемый как опорный, тогда как эффект от остального в модели микрофона (как моделирование частотной характеристики) потребует уточнения.

Второй и третий приёмы будут использоваться чаще. Действие по перетаскиванию генератора на частотный график отличается от перетаскивания его на график переходного процесса. Вместо добавления пробника, генератор нужен как опорная точка схемы. Нужен генератор не синусоидальный — DC, Pulse и Pwlin, однако, вполне подойдут. Если вы все-таки хотите добавить генератор как пробник, это все ещё можно сделать, используя диалоговую форму Add Trace.

Амплитуда опорного источника всегда 1 вольт, фаза всегда 0 градусов. Внутреннее

сопротивление опорного генератора будет зависеть от того, что определено для генератора на первом месте. Это используется для расчётов децибел, то есть, 0dB = 1 volt. ISIS ограничивает очень маленькие значения, избегая log(0) ограничением до -200dB.

Частотный график всегда имеет обе, левую и правую, оси у. Левая ось используется для отображения амплитуды сигнала, а правая для отображения фазы. Если вы перетаскиваете пробник на левую сторону графика, будет отображаться амплитуда, а если на правую, фаза. Ось x используется для отображения частоты опорного генератора (reference generator). Для отображения частоты всегда используется логарифмическая шкала. Левая ось может иметь либо dB, либо обычные единицы, а правая всегда проградуирована в градусах.

Чтобы получить частотную характеристику цепи:

1. Разместите пробники на схеме в точках, которые вас интересуют.
2. Поместите график Frequency.
3. Добавьте на него пробники: для отображения амплитуды к левой стороне, для отображения фазы к правой. Вы можете добавлять пробник дважды, чтобы получить и фазу, и амплитуду.
4. Отредактируйте график (укажите на него и нажмите **CTRL+E**), задайте начальную и конечную частоту, и все требуемые *Simulation Control Properties*.
5. Для вызова PROSPICE нажмите пробел.

Файлы примеров ZIN.DSN и ZOUT.DSN показывают, как комбинировать частотный график с выражениями кривой для получения входного и выходного импеданса.

АНАЛИЗ РАЗВЁРТКИ НА ПОСТОЯННОМ ТОКЕ

Обзор

При анализе развёртки на постоянном токе (DC Sweep analysis) вы можете видеть, как изменения цепи скажутся на её работе. Это выполняются с помощью *Property Expression Evaluation* симулятора PROSPICE. График развёртки определяет переменную, которая будет «развёрнута» в несколько шагов в заданном пользователем диапазоне. Переменная развёртки может появиться в свойствах любого элемента схемы, такого как значение сопротивления, усиление транзистора или температура схемы.

Кривая DC Sweep показывает уровень установившегося состояние напряжения (или тока) в точках наблюдения на схеме при увеличении переменной развёртки. Она может использоваться для вывода передаточной характеристики цепи на постоянном токе, если присвоить переменную качания значению генератора, или для вывода эффекта влияния значения компонента на рабочую точку цепи.

Метод вычисления

При анализе развёртки на постоянном токе PROSPICE многократно определяет рабочую точку цепи, увеличивая переменную качания между вычислениями. PROSPICE будет пересчитывать все переменные, используемые между шагами, так что переменная развёртки

может использоваться так часто, как это нужно, а также могут использоваться переменные, основанные на переменной развёртки. Любые параметры инициализации схемы будут приняты во внимание для каждого вычисления рабочей точки.

Использование графика DC Sweep

Как и при аналоговом анализе переходных процессов (см. «Аналоговый анализ переходных процессов»), можно использовать либо левую, либо правую, либо обе оси y. По оси x откладывается переменная развёртки. Выражения кривой, описанные для диалоговой формы команды *Add Trace*, могут использоваться и для *DC sweep*. При выборе количества шагов не забывайте, что время симуляции прямо пропорционально выбранному количеству шагов.

Для вывода передаточной функции схемы:

1. Поместите *DC generator* на входе схемы. Задайте его значение как X.
2. Поместите один или несколько пробников на выход схемы.
3. Поместите график *DC Sweep* и добавьте пробники на него — см. «Добавление кривых на график».
4. Отредактируйте график (**CTRL+E**) и задайте начальное и конечное значения в пределах входной развёртки, которые вам нужны. Проверьте, что переменная развёртки — это X.
5. Нажмите пробел для вызова PROSPICE.
6. Если полученные кривые выглядят не связанно или угловато, когда вы увеличиваете график, увеличьте количество шагов в диалоге *Edit DC Sweep Graph*.

Чтобы увидеть эффект от альтернативных значений цепи:

1. Подготовьте схему, как сделали бы для анализа переходных процессов. Добавьте пробники и генераторы в подходящих точках схемы.
2. Отредактируйте компоненты, влияние значений которых вас интересует. Задайте их значения, как выражения, содержащие X, переменную развёртки. Вы можете редактировать только один или несколько компонентов.
3. Разместите график DC Sweep и добавьте пробники и генераторы на нем — см. «Добавление кривых на график».
4. Отредактируйте график и задайте параметры качания параметров.
5. Нажмите пробел для вызова PROSPICE.

АНАЛИЗ РАЗВЁРТКИ НА ПЕРЕМЕННОМ ТОКЕ

Обзор

Этот тип анализа создаёт семейство кривых АЧХ для разных значений переменной развёртки.

Основное использование этого типа графика — посмотреть, как значения отдельных компонентов влияют на амплитудно-частотную характеристику (АЧХ) вашей схемы.

Метод вычисления

При этом анализе вычисления происходят, как при обычном частотном анализе, исключая то, что выполняются много раз, по одному расчёту для каждого значения переменной качания.

Ограничение на линейность схемы, присущие этому анализу, такие же, как и для обычного частотного анализа — см. «Частотный анализ».

Использование графиков AC Sweep

Как и при Frequency анализе, левая и правая оси отображают, соответственно, усиление и фазу. Также входной генератор должен быть задан, как опорная точка для расчёта усиления.

Чтобы увидеть эффект от альтернативных параметров на частотной характеристике:

1. Подготовьте схему, как для проведения частотного анализа.
2. Отредактируйте нужные вам компоненты. Задайте их свойства, обратив внимание на то, чтобы выражения содержали X, переменную развёртки. Вы можете редактировать один или несколько компонентов.
3. Поместите график AC Sweep и добавьте пробники на него — см. «Добавление кривых на график».
4. Если у вас нет генератора на входе схемы, добавьте его.
5. Выделите и перетащите генератор, который на входе схемы, на график, как опорный генератор.
6. Отредактируйте график (**CTRL+E**) и задайте параметры качания. Задайте параметр *Freq* с интересующей вас частотой.
7. Нажмите пробел для вызова PROSPICE.

АНАЛИЗ ПЕРЕДАТОЧНОЙ КРИВОЙ НА ПОСТОЯННОМ ТОКЕ

Обзор

Этот тип графика был специально разработан для получения семейства характеристических кривых полупроводниковых устройств, хотя подчас он применим и к другим приложениям. Каждая кривая состоит из выводимых напряжений или токов рабочей точки, как функции установленного входного генератора, который «качается» от одного DC значения до другого. Может также устанавливаться дополнительный генератор для получения набора кривых.

Метод вычисления

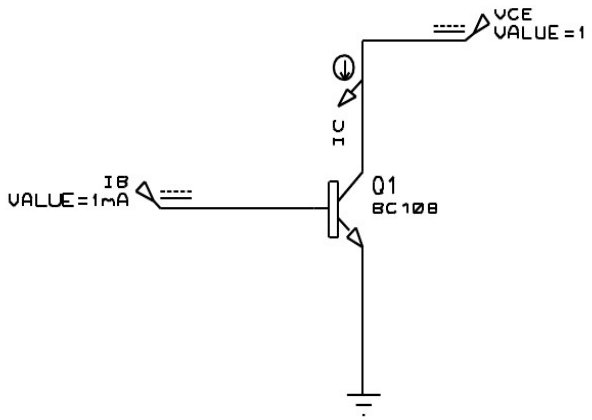
Он очень похож на вычисления для DC Sweep анализа, исключая то, что могут «качаться» два значения. Рабочая точка находится для начального значения каждого генератора, а первый затем проходит по шагам, заданным в установленном диапазоне. После каждого шага первого генератора, значение второго генератора увеличивается; новая кривая выводится для каждого отдельного значения второго генератора.

Использование передаточных (Transfer) графиков

Как и при аналоговом анализе переходных процессов (см. «Аналоговый анализ переходных процессов»), может использоваться либо левая, либо правая, либо обе оси y. Ось x показывает первую переменную качания, а отдельная кривая результат для каждого значения второго генератора. Выражения кривой, описанные в «Диалоговая форма команды Add Trace», могут также использоваться при необходимости.

Для вывода передаточных кривых для транзистора:

1. Постройте схему, похожую на изображённую ниже:



   IB — это источник тока, а *VCE* — источник напряжения. *IC* измеряет ток коллектора.

2. Разместите график *Transfer* и добавьте пробник *IC* на него.

3. Укажите на график и нажмите **CTRL+E,** чтобы отредактировать его.
4. Присвойте *Source 1* источнику *VCE*, а *Source 2* источнику *IB*.
5. Задайте диапазоны напряжения и тока в пределах, чувствительных для транзистора — то есть, 0 -> 10V для *VCE* и 100uA -> 1mA для *IB*.
6. Подстройте число шагов для *IB*, дающих значимые промежуточные результаты. Не забывайте, что они дадут на одну кривую больше, чем количество шагов, поскольку один шаг подразумевает два отдельных значения.
7. Закройте диалоговую форму и выберите **ОК** для симуляции графика.

АНАЛИЗ ШУМОВ

Обзор

Движок симулятора SPICE может моделировать тепловой шум, генерируемый в резисторах и полупроводниковых компонентах. Индивидуальный вклад от шумов суммируется на пробнике напряжения в схеме для некоторого диапазона частот. Напряжение шумов, нормализованное корнем квадратным из полосы частот, может затем выводиться как функция частоты. Кривые графика шумов всегда имеют единицы V/ √Hz.

Вычисляются два типа шумов — выходные шумы (Output Noise) и эквивалентные входные шумы (Equivalent Input Noise). Расчёт последних возможен относительно уровня входного сигнала или входных шумов, поскольку представляет уровень шумов на входе, которые потребуют создания действительных шумов на выходе, взятых с усилением схемы на отдельных частотах.

Размещение пробника на левой оси отображает выходной шум, а приведённый ко входу шум можно отобразить, если перетащить пробник на правую сторону графика.

Анализ шумов имеет тенденцию к выводу крайне малых значений (порядка нановольт). По этой причине есть опция отображения их в dB. 0dB относится к 1V/ √Hz.

Корректное моделирование шумов для схем, использующих макромодели IC, не гарантировано, поскольку эти модели могут использовать линейные управляемые источники для моделирования только базового поведения устройства.

Метод вычисления

Рабочая точка цепи вычисляется обычным образом, а затем модель цепи модифицируется для корректного суммирования вкладов шумов. Все генераторы, исключая опорный входной, игнорируются при анализе шумов (исключая момент вычисления рабочей точки), и их можно не удалять перед анализом. PROSPICE будет вычислять тепловой шум для каждого пробника напряжения в схеме, включая те, что связаны с генераторами и записывающими устройствами, поскольку нет способа узнать, какой пробник интересует вас при анализе. Шумовые токи не поддерживаются, поэтому PROSPICE игнорирует все токовые пробники. При анализе шумов производятся отдельные симуляции для каждого пробника, поэтому время симуляции прямо пропорционально количеству размещённых пробников напряжения.

Использование графика шумов

При использовании анализа шумов важно помнить, что эффекты от вклада внешних электрических и магнитных полей не моделируются. Во многих ситуациях шумы, привносимые на вход, могут превосходить шумы, производимые системой.

Если вы разбиваете схему, используя записи (tapes, см. «Записи и разбиение»), вы должны быть уверены, что только текущая часть учитывается PROSPICE симулятором. Вид шума, взятый в изоляции, всё ещё полезен, но самописцы (tapes) должны удаляться, чтобы увидеть эквивалентный шум всей цепи.

В порядке определения источника шума в цепи отдельные напряжения шумов добавляются в журнал симуляции (simulation log). По порядку рассматривается каждый пробник, и для каждого компонента получается вклад шума. Вклады обсчитываются и выводятся как квадраты значений. Этот процесс выполняется для начальной и конечной частот. Если вы имеете много пробников, может быть, лучше выводить результирующие данные.

Чтобы провести анализ шумов схемы:

1. Разместите график *Noise* и отредактируйте его, чтобы задать нужный диапазон частот и входной опорный (reference) генератор.
2. Добавьте пробники напряжения на выходы схемы или в другие интересующие вас точки, и на график (см. «Добавление кривых на график»).
3. Нажмите пробел, чтобы запустить симулятор PROSPICE.
4. Если уровень шума требуется уменьшить, проверьте отчёт о симуляции (simulation log, **CTRL+V**), чтобы определить источник шума.

АНАЛИЗ ИСКАЖЕНИЙ

Обзор

Анализ искажений (Distortion analysis) определяет уровень гармонических искажений, получаемых в схеме при тестировании. Это могут быть либо 2я, либо 3я гармоника основного сигнала или интермодуляционные продукты двух тестовых сигналов.

Искажения создаются нелинейностями передаточной функции схемы — схемы, составленные только из линейных компонентов (резисторов, конденсаторов, индуктивностей, линейных управляемых источников) не будут создавать искажений. Анализ искажений SPICE моделей подразумевает искажения на диодах, биполярных транзисторов, JFET и MOSFET.

Корректное моделирование искажений для цепей, включающих IC макро модели, не гарантировано, поскольку эти модели могут использовать линейные управляемые источники для моделирования только основного поведения устройства.

Похожая информация может быть получена при использовании Фурье анализа (Fourier analysis), но *Distortion* анализ также способен показать, как меняются искажения, когда основная частота «качается».

Метод вычисления

Этот анализ основан на малосигнальных (AC) моделях для устройств в схеме, так что первый шаг — это расчёт рабочей точки. Каждая модель нелинейного компонента затем добавляет комплексные значения искажений для подходящих гармоник, в зависимости от того, как много устройств влияет на входной основной сигнал. Величины, которой достигают эти гармоники, появляются на выходе и определяют выводимые значения. Процесс повторяется в заданном диапазоне входных частот. Фактически, математика этого анализа крайне сложна и подразумевает конструкции и манипуляции с рядами Тэйлора, чтобы представить нелинейные устройства.

Заметьте, что используются комплексные значения, так что анализ предоставляет информацию и об амплитуде, и о фазе каждой гармоники.

Для одночастотного гармонического анализа искажений получаются две кривые на графике — одна для 2й гармоники, а вторая для 3й.

Для интермодуляционных искажений используются две входные частоты, заданные в терминах отношения между 2й частотой (F2) и основной (F1). Отображаются три кривые, показывающие интермодуляционные артефакты на F1+F2, F1-F2 и 2F1-F2.

Некоторого внимания потребует выбор отношения F2/F1, поскольку иначе могут обнаружиться математические странности. Например, если F2/F1 равно 0.5, то значение F1-F2 равно F2, а вывод значения F1-F2 будет бессмысленным, поскольку оно совпадает со значением второй основной частоты. В общем, лучше остановить свой выбор для этого отношения на иррациональном числе. Так F2/F1=49/100 будет много лучше. Заметьте, что $F2/F1 < 1$.

Использование графика Distortion

График искажений показывает амплитуду гармоник на левой оси и фазу (обычно менее интересующую) на правой оси. Тестовая частота (F1) выводится на оси x.

Для гармонического анализа (Harmonic Distortion) при одной частоте на входе (F1) выводятся две кривые — по одной для каждой составляющей, 2F1 и 3F1.

Для интермодуляционного анализа (две входные частоты, F1 и F2) выводятся три кривые, показывающие составляющие на F1+F2, F1-F2 и 2F1-F2.

В любом случае, какая кривая есть какая, может быть определено, если вывести курсор на график — кривая, на которую вы указали, будет идентифицирована на правой стороне строки состояния.

ФУРЬЕ АНАЛИЗ

Обзор

Фурье анализ (Fourier Analysis) — это процесс преобразования данных временной области в частотную, а результаты похожи на те, что получаются при подключении спектрального анализатора вместо осциллографа. Он особенно полезен при анализе гармонических составляющих сигнала, возможно, при исследовании отдельных типов искажений, но и имеет множество других приложений.

Метод вычисления

Фурье анализ начинается с выполнения анализа переходных процессов (Transient), а затем выполняется быстрое преобразование Фурье над полученными данными. Этот процесс выполняется дискретным по времени выбором образов из данных временной области с последующим применением критерия Найквиста. Положим просто, что наивысшая частота, которая рассматривается, равна половине выбранной частоты. Однако могут обнаружиться другие, вводящие в заблуждение эффекты при наложении выбранной частоты с гармониками входного сигнала, которые выше половины выбранной частоты. Чтобы минимизировать подобные эффекты, можно применить к входным данным разного типа интервалы до быстрого преобразования Фурье.

Использование графиков Fourier

В первом приближении вам нужно приготовить вашу схему, как для анализа переходного процесса, исключая то, что вы должны использовать график *Fourier* вместо (или также как!) графика *Analogue*.

Для анализа частотного содержимого сигнала:

1. Приготовьте схему, как для анализа переходных процессов.
2. Добавьте график *Fourier* в проект и перетащите на него пробник, соединённый с нужными вам точками.
3. Выберите начальное и конечное время и значение частота/разрешение (frequency/resolution), подходящее для сигнала, который вы анализируете. Если возможно, выберите временной интервал и частотное разрешение, которые соответствуют основной частоте анализируемого сигнала.
4. Нажмите пробел для вызова PROSPICE.

Если время начала и окончания одинаковы и для графика переходного процесса, и для графика анализа Фурье, PROSPICE не нужно будет повторять симуляцию схемы между этими двумя анализами. Вместо этого ISIS выполнит только быстрое преобразование Фурье с данными на существующей временной области.

АУДИО АНАЛИЗ

Обзор

PROTEUS VSM включает ряд возможностей, которые позволяют вам услышать выход вашей схемы (при наличии, конечно, звуковой карты!). Главный компонент этого — график *Audio*. Это, в основном, то же, что и график *Analogue*, исключая то, что после симуляции генерируется файл Windows WAV из данных временной области, который и воспроизводится через вашу звуковую карту.

Файлы WAV могут также экспортироваться для использования в других приложениях.

Метод вычисления

Аудио анализ выполняется точно так же, как и анализ переходных процессов, исключая то, что после симуляции данные заново обрабатываются на одной из стандартных для PC частот дискретизации (11025, 22050 или 44100 Гц), а затем записываются в формат WAV, используя стандартную функцию Windows, предназначенную для этой цели. В завершение подаётся команда проиграть WAV файл на вашем аудио оборудовании.

Использование графика Audio

В первом приближении вам нужно приготовить схему, как для анализа переходного процесса, исключая то, что вы должны использовать график *Audio* вместо *Analogue*.

Чтобы услышать аудио выход схемы:

1. Подготовьте схему, как для анализа переходных процессов.
2. Добавьте график *Audio* в проект и перетащите пробник с выхода схемы на график.
3. Выберите начальное, конечное время и время цикла для генерации сигнала подходящей длительности, но с минимумом актуальной симуляции. Создание одной секунды аудио сигнала с анализом 1 мс и циклом в 1000 раз значительно быстрее, чем анализ схемы для целой секунды.
4. Выберите разрешение для образца и диапазон, подходящие для естественного прослушивания сигнала.

   Используйте 16-битовое разрешение, если вы не создаёте большие файлы и/или при нехватке дискового пространства.

   Большинство звуковых систем PC не работают с частотой дискретизации больше, чем 44.1 кГц.
5. Нажмите пробел, чтобы вызвать PROSPICE.
6. Нажмите **CTRL-SPACE** для повторного воспроизведения без повторения симуляции.

ИНТЕРАКТИВНЫЙ АНАЛИЗ

Обзор

Интерактивный анализ комбинирует преимущества симуляции, основанной на интерактивности и графиках. Симулятор стартует в интерактивном режиме, а результаты записываются и отображаются на графике, как при анализе переходных процессов.

Этот вид анализа особенно полезен при проверке того, что случилось при проведении отдельных операций в проекте, и его можно рассматривать, как комбинацию запоминающего осциллографа и логического анализатора в одном устройстве.

Метод вычисления

Метод вычисления идентичен тому, что выполняется для анализа переходных процессов в смешанном режиме, исключая то, что симулятор работает в интерактивном режиме. Следовательно, операции переключения, операции с клавиатурой и другими приводами в схеме будут сказываться на результатах. Также симуляция будет происходить со скоростью, определяемой параметром *Animation Timestep*, а не с наиболее возможной скоростью.

- Остерегайтесь захвата очень большого количества данных. Тактовый генератор процессора, работающий на реальной скорости, генерирует миллионы событий в секунду, а это будет занимать много мегабайт, если будет захвачено и отображено на графике — особенно, если вы просматриваете шину данных или адресную шину. Вы можете легко «подвесить» вашу систему, если ISIS загрузит 20 или 30 Мбайт результирующих данных.

  Возможно, лучше использовать в этом случае *Logic Analyser*, если нет возможности получить требуемые данные за относительно короткий период симуляции.

- Как и с обычной интерактивной симуляцией, многофрагментарные схемы не поддерживаются, а любые записывающие объекты, не установленные в режим проигрывания, будут автоматически удалены из схемы.

Использование интерактивных графиков

Обычный сценарий для интерактивного анализа таков, что вы проверяете проект с помощью интерактивной симуляции и находите, что происходит нечто странное при обычных управляющих операциях. В первом приближении вы можете попробовать использовать виртуальные инструменты, чтобы увидеть, что происходит, но в некоторых случаях есть необходимость захватить результаты для графика и внимательно исследовать их.

Чтобы выполнить интерактивный анализ:

1. Добавьте пробники в интересующие вас точки.
2. Разместите график *Interactive* в свободной области схемы и перетащите на него пробник.
3. Отредактируйте график и выберите подходящие времена начала и окончания.

Заметьте, что PROSPICE не будет захватывать данные от датчика до момента старта — это избавляет от «горы» данных, которые получит ISIS.

4. Установите любые интерактивные компоненты на схему в подходящем начальном состоянии.
5. Приготовьтесь к выполнению интерактивных операций и затем нажмите клавишу **SPACE** (пробел). Вы должны уложиться с операциями с активными компонентами в промежуток времени, который задали на шаге 4, если эффект от этих действий должен быть записан. Если это трудно выполнить, вы можете либо увеличить время останова, либо увеличить параметр *Timestep Per Frame* (время шага на кадр).

ЦИФРОВОЙ АНАЛИЗ СООТВЕТСТВИЯ

Обзор

Анализ соответствия (conformance analysis) сравнивает один набор результатов цифровой симуляции с другим. Идея в том, что проект, который был ранее признан рабочим, может быть быстро перепроверен после модификации, с тем чтобы удостовериться в отсутствии нежелательных сторонних эффектов, как результата изменений. Это обычно относится к приложениям на основе микроконтроллеров, где всё приложение может нуждаться в перепроверке после изменений, которые были внесены в код программы.

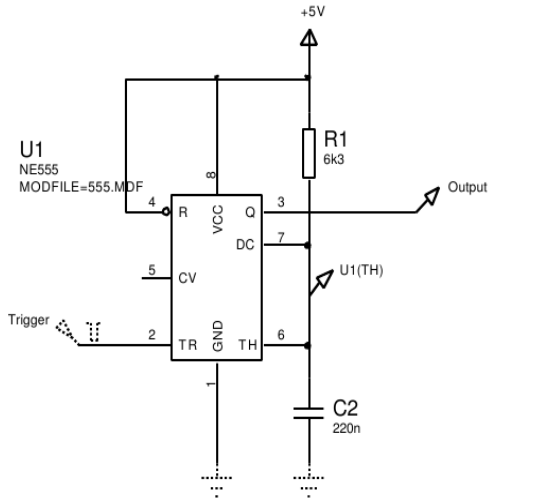
Метод вычисления

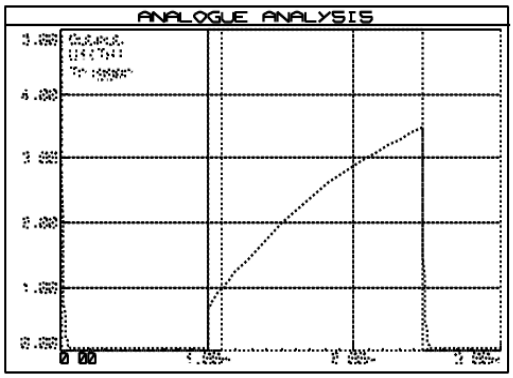
Метод вычисления идентичен тому, что выполняется для цифрового анализа переходных процессов, исключая то, что могут запоминаться два набора результатов на графике. Мы назвали два набора результатов: проверяемые результаты и опорные результаты.

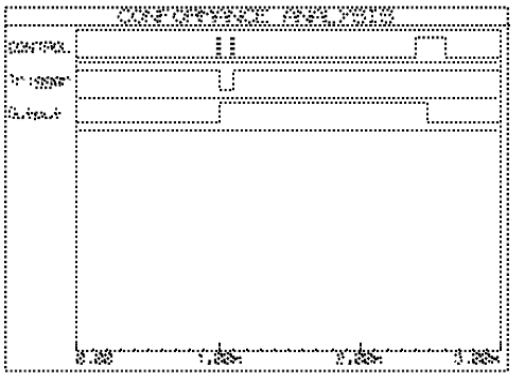
Соответствие или нет — определяется сравнением проверяемых и опорных результатов на каждом фронте первой кривой. Мы назвали эту кривую контрольной, и она отображается иным цветом, чтобы отличать её от остальных. Очень важно, что нет требования для фронтов в тестовой и опорной копиях контролируемой кривой быть совпадающими по времени. Это означает, что изменения в абсолютном времени событий внутри данных результата, не свидетельствуют несомненно об отсутствии соответствия. Это обычно относится к приложениям с микроконтроллерами, где любые изменения кода будут ограничены во влиянии на абсолютное время событий внутри системы. В этих случаях контрольная кривая может быть сгенерирована самим кодом на входе и/или выходе в программные процедуры при тестировании.

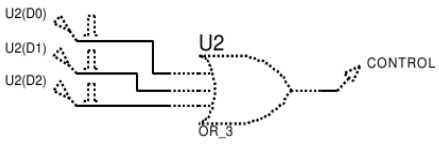
Использование графиков Conformance

Обычно анализ соответствия используется, как часть стратегии тестирования, и чаще во встроенных системных приложениях, хотя мы также используем их в нашей фирме для тестирования наших моделей симуляторов. На простом примере ниже будет показано, как использовать график *Conformance* для тестирования операций моностабильного 555.









Проверяемая схема состоит из U1, R1 и C2, которые соединены в классическую схему моностабильного 555.

Схема запускается в 1 мс цифровым импульсным генератором, и выходное напряжение отображается на графике аналогового анализа. Чтобы проверить корректность работы схемы, было бы правильно установить следующие факты:

- Что выходной сигнал в низком состоянии до прихода управляющего импульса.
- Что выходной сигнал переходит в высокое состояние вскоре после перехода управляющего сигнала в низкое состояние.
- Что выходной сигнал остаётся в высоком состоянии, когда управляющий сигнал становится высоким.
- Что выходной сигнал остаётся в высоком состоянии на время около 1.5 мс.
- Что выходной сигнал переходит в низкое состояние после этого интервала времени.

Чтобы получить это, используя график *Conformance*, нам нужно отобрать выходные данные около каждого фронта перехода сигнала управления, а также с любой стороны выходного сигнала. Мы можем определить допуск для времени задержки одновибратора, выбирая промежуток между последними двумя точками пробы.

Это получается, если использовать последовательность импульсов цифрового генератора,

выход которого комбинируется с вентилем ИЛИ. Ширина каждого импульса генератора определяет временной допуск для каждого ожидаемого события. Например, третий импульс генератора задаёт переключение между 2.4 мс и 2.6 мс, давая допуск в +/- 100us на ширину ожидаемого выходного импульса.

Заметьте, что вы должны использовать цифровой генератор или генератор шаблонов для управляющего сигнала; использование простого тактового генератора приведёт к проверке в регулярные временные интервалы, которые, фактически, будут достаточны для многих приложений.

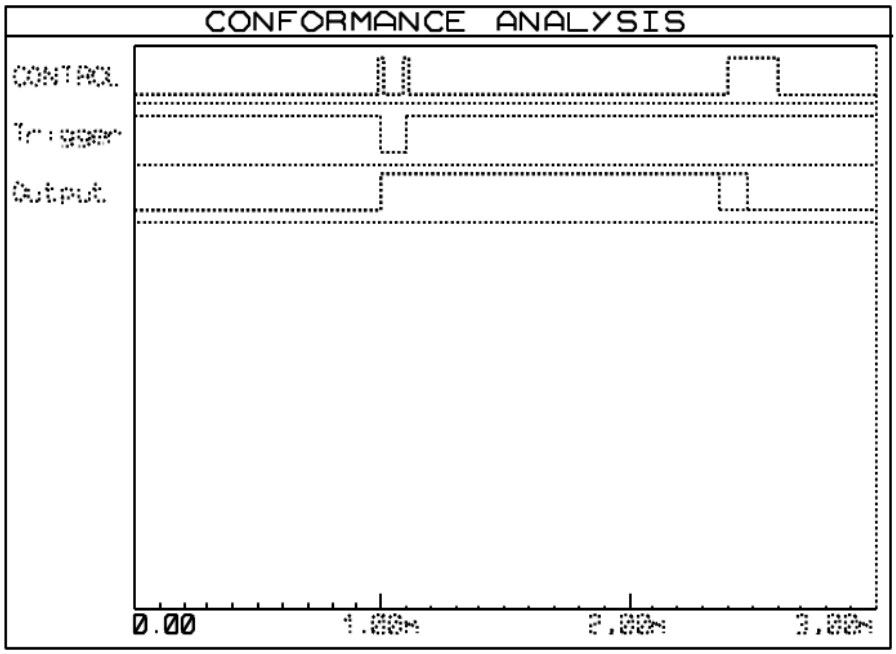
В основном процедура может быть суммирована следующим образом.

Чтобы подготовить анализ соответствия:

1. Решите, что вы хотите проверить и в какое время.
2. Постройте тестовую схему, которая генерирует выходной сигнал, который вы проверяете и управляющий сигнал, меняющий состояние каждый раз, когда вы хотите подтвердить результаты.
3. Поместите график *Conformance* и подготовьте его точно так, как вы сделали бы для цифрового анализа, удостоверившись, что контрольная кривая находится в верхней части графика.
4. Запустите симуляцию и проверьте, что результаты таковы, как ожидалось, и что переходы на контрольной кривой обнаруживаются в те моменты, когда вы хотели бы проверить результаты.
5. Отредактируйте график *Conformance* и нажмите кнопку **Store Results** (сохранить результаты). Это сделает текущие отображаемые результаты опорными результатами. Затем это отобразится приглушённым цветом и график будет сравниваться с любыми новыми результатами, когда он будет заново симулирован.
6. Сохраните проект — теперь он пригоден для целей проверки в будущем.

Теперь давайте предположим, что из-за нехватки компонентов, значения R1 и C2 должны быть изменены. Если мы сделаем C2 равной 100nf, то можем ли взять R1 равным 15k, и будет ли проект ещё работать с такой спецификацией?

Результаты этого эксперимента показаны ниже:



Ответ — нет. Выходной импульс теперь стал слишком коротким, и в отчёте симуляции для графика появляется следующая информация:

```
Comparing Results...
Data mismatch at time 2.40m in trace 'Output'.
Data signatures were 'L' and 'H'.
Conformance analysis FAILED.
```

Заметьте, что курсор на графике в позиции времени первой разницы, то есть, 2.4 мс.

Есть фактически три пути запуска анализа соответствия:

- Новой симуляцией графика обычным образом, то есть, нажатием на пробел или использованием команды *Simulate Graph* из раздела *Graph*.
- Использованием команды *Conformance Analysis* из раздела *Graph* основного меню. Это заново симулирует все графики соответствия в проекте и сообщит о любых ошибках.
- Только для опытных пользователей — можно использовать командную строку. Ввод

  ISIS <filename> /V

  выполнит глобальный анализ соответствия, как выше. Если какой-то из графиков станет причиной отказа, ISIS установит уровень ошибки 1. Это позволяет использовать пакетные файлы для автоматического выполнения множественных тестов.

РАБОЧАЯ ТОЧКА НА ПОСТОЯННОМ ТОКЕ

Это единственный тип анализа (*DC OPERATING POINT*), для которого нет соответствующего графика. Тем не менее, вычисленную информацию можно увидеть, используя интерфейс «point and shoot, прицельсяи стреляй» в ISIS.

Важно понимать, что рабочая точка вычисляется для графика тока (*Current Graph*), и что на схеме должен быть график для расчёта рабочей точки. Основанием для этого служит то, что без графика у системы нет понимания, какие части схемы следует симулировать. Вдобавок, функциональная зависимость от *Property Expression Evaluation* (оценка выражений свойств) и возможности задать свойства на графике означает, что актуальные значения компонентов могут зависеть от того, какой используется график.

Чтобы увидеть значения рабочей точки:

1. Настройте схему как для анализа переходных процессов.
2. Добавьте график на схему и добавьте пробники и генераторы на него, соответствующие секциям схемы, которые вы хотите симулировать.
3. Симулируйте рабочую точку, нажав **ALT+SPACE**. Пробники напряжения и тока будут отображать их значения рабочей точки на постоянном токе немедленно.
4. Щёлкните по иконке *Multimeter*.
5. Укажите на любой компонент и щёлкните левой клавишей мышки, чтобы увидеть его информацию о рабочей точке.

Пара замечаний:

- Можно вычислять рабочую точку без графика. В этом случае выполняется симуляция всей схемы, как единой секции — то есть, независимо от любых пробников, генераторов или самописцев. Это может окончиться неудачей, если есть объекты, не имеющие моделей.
- Компоненты, которые фактически являются родительскими для подсхем, или которые моделируются файлами MDF или SPICE, будут отображать только информацию о базовых рабочих точках, то есть, узловые напряжения.

ГЕНЕРАТОРЫ И ПРОБНИКИ

ГЕНЕРАТОРЫ

Обзор

Генератор — это объект, который может быть установлен, чтобы дать сигнал в той точке, к которой он подключён. Есть множество типов генераторов, каждый из которых генерирует разного рода сигналы:

- *DC* — источник постоянного напряжения.
- *Sine* — генератор синусоидального напряжения с управляемыми амплитудой, частотой и фазой.
- *Pulse* — аналоговый импульсный генератор с управляемыми амплитудой, периодом и временем нарастающего и спадающего фронтов.
- *Exp* — экспоненциальный импульсный генератор; производит импульсы такого же вида, какой имеют RC цепи при заряде/разряде.
- *SFFM* — частотномодулированный одночастотный генератор — производит сигнал, определяемый частотной модуляцией одного синусоидального сигнала другим.
- *Pwlin* — генератор кусочно линейных сигналов; для создания импульсов и сигналов произвольной формы.
- *File* — как и выше, но данные берутся из ASCII файла.
- *Audio* — используются Windows WAV файлы для входных сигналов. Особый интерес представляет комбинация с графиком *Audio*, поскольку вы можете прослушать полученный от вашей схемы эффект на аудио выходе компьютера.
- *DState* — установившийся логический уровень.
- *DEdge* — единственный логического уровня переход или фронт.
- *DPulse* — единственный цифровой тактовый импульс.
- *DClock* — цифровой тактовый сигнал.
- *DPattern* — произвольная последовательность логических уровней.

Первые восемь типов генераторов предполагаются быть аналоговыми компонентами, и симулируются ядром SPICE симулятора, тогда как остальные относятся к цифровым цепям и поддерживаются DSIM.

Размещение генераторов

Чтобы разместить генератор:

1. Получите список типов генераторов, выбрав иконку *Generator Mode*. Список типов генераторов отображается в *окне выбора*.
2. Выберите тип генератора, который вы хотите поместить, в окне выбора. ISIS покажет вид генератора в *окне просмотра*.
3. Используйте иконки поворота и отражения (Rotation и Mirror), чтобы придать нужное положение генератора, как считаете удобным.
4. Переместите мышку в *окно редактирования*, и щёлкните левой клавишей в нужном вам месте.

Вы можете разместить генератор непосредственно на существующем проводе, расположив его так, чтобы его точка соединения касалась провода. В качестве альтернативы — размещение нескольких генераторов на свободном месте с последующим их соединением со схемой.

Когда генератор размещён без соединения с существующими проводами, он имеет имя по умолчанию со значком вопроса (?), чтобы показать, что он не отмечен. Когда генератор соединён с сетью (возможно, когда размещён на ней, если помещается непосредственно на существующий провод), он получает имя этой сети, или, если сеть сама не помечена, ссылку на компонент и имя вывода первого, что соединён с сетью. Имя генератора будет автоматически обновляться, когда он отсоединяется или когда перетаскивается от одной сети к другой. Вы можете задать ваше собственное имя генератору, отредактировав объект, в последнем случае имя принадлежит только ему и не обновляется.

См. «Имена сетей» в руководстве к ISIS, где рассказано о сетях и их наименовании.

Редактирование генераторов

Любой генератор можно отредактировать, используя одну из основных техник редактирования ISIS, самая простая из которых — щёлкнуть правой клавишей мышки по объекту и выбрать из выпадающего меню *Edit Properties*, или указать на генератор и нажать **CTRL+E**.

Диалоговая форма Edit Generator предлагает ряд общих полей, а затем дальнейший набор полей, который меняется согласно типу генератора. Общие поля описаны ниже:

Name Имя генератора.

Вы можете изменить имя генератора, вписав новое в поле *Name*. Когда вы меняете имя генератора вручную, ISIS никогда не изменит его, даже если вы переместите генератор к новой цепи.

Если вам удобнее вернуться к автоматическому наименованию, очистите это имя, оставив строку пустой, нажав один раз **ESC**, а затем щёлкнув **ОК**.

Type Тип генератора. Вы можете изменить тип генератора (от размещённого типа), выбрав кнопку для нового требуемого типа.

Это управление определяется специальным полем генератора, которое отображается на правой стороне диалога.

Current Source За исключением DIGITAL генератора все остальные типы способны оперировать либо с источником напряжения, либо с источником тока. Установка последнего флажка превращает генератор в источник тока.

Isolate Before Этот флажок управляет, будет или нет генератор размещён в середине провода, образуя обрыв провода, к которому он подключён, изолируя сеть, отходящую от генератора, от сети за генератором.

Установка флажка не сказывается на генераторе, соединённом непосредственно с проводом сети из единственного провода.

Manual Edits Когда флажок установлен, свойства генератора отображаются как текстовый список свойств. Это поддерживается вручную для обратной совместимости с предыдущими версиями программы. Однако опытные пользователи могут использовать выражения свойств в свойствах генератора, а это возможно только в ручном режиме редактирования.

Генераторы постоянного тока

DC generator используется для генерации постоянного уровня аналогового напряжения или тока. Генератор имеет единственное свойство, которое задаёт выходной уровень.

Генератор синуса

Sine generator используется для создания непрерывного синусоидального сигнала одной частоты. Некоторые параметры могут настраиваться:

- Выходной уровень задаётся как пиковая амплитуда (*VA*) с дополнительной постоянной составляющей (*VO*). Амплитуда может также задаваться в терминах действующего (RMS) или от пика до пика значений.
- Частота генерации может задаваться в Гц (*FREQ*) или как период (*PER*), или в терминах числа циклов на всем графике.
- Фазовый сдвиг может задаваться либо в градусах (*PHASE*), либо временной задержкой (*TD*). В последнем случае генерация не начнётся раньше заданного времени.
- Экспоненциальный спад кривой после старта осцилляции задаётся коэффициентом затухания *THETA*.

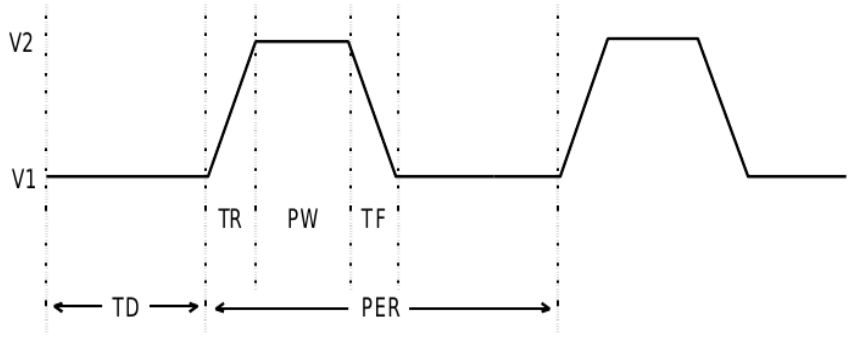
Математически выход задаётся:

$$V = VO + VAe^{-(t-TD)\,THETA}\sin(2\pi FREQ(t + TD))$$

для t >= TD. Для t < TD выход — просто постоянная составляющая (DC offset) *VO*.

Импульсный генератор

Pulse generator используется для получения разных повторяющихся входных сигналов для аналогового анализа. могут быть заданы прямоугольные, пилообразные и треугольные импульсы, равно как и единичные короткие импульсы. Заметьте, что времена подъёма и спада не могут быть нулевыми, так что реально прямоугольные импульсы не позволительны. Причина этого в том, что мгновенные изменения в основном не позволительны в PROSPICE. Операции с импульсным генератором лучше описываются следующей диаграммой:

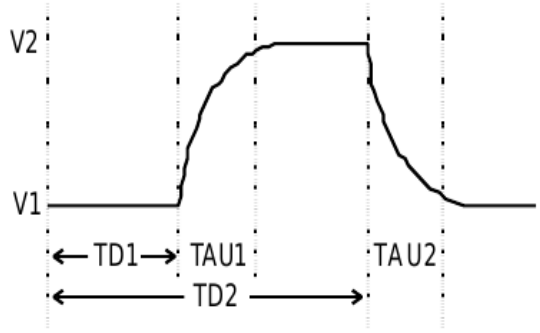


где

PER	Период сигнала. Если не задан, то используется **FREQ**.
FREQ	Частота сигнала. Она предопределена для одного периода анализа переходного процесса.
V1	Значение низкого уровня выходного сигнала.
V2	Высокий уровень выхода.
PW	Время за которое выход становится **V2** в каждом цикле. Это не включает **TR** и **TF**.
TR	Время переднего фронта — время между LOW и HIGH в каждом цикле. Это предопределено как1ns.
TF	Время заднего фронта — время между HIGH и LOW в каждом цикле. Предопределено как **TR**.
TD	Время задержки. Выход генератора стартует при **V1** и будет оставаться на этом уровне в течение **TD** секунд.

Экспоненциальные генераторы

Exp generator производит сигналы похожие на заряд и разряд RC цепи. Параметры лучше всего описываются диаграммой ниже.



где параметры это:

V1	Значение низкого уровня выходного сигнала.
V2	Высокий уровень выхода.
TD1	Начальное время подъёма сигнала.
TAU1	Постоянная времени подъёма сигнала. Это время, за которое напряжение увеличивается примерно до 0.63 значения полного потенциала.
TD2	Начальное время спада сигнала. Заметьте, что **TD2** отсчитывается от нуля.
TAU2	Постоянная времени спада сигнала.

Математически сигнал описывается на трёх интервалах:

0 to TD1 $V1$

TD1 to TD2 $V1+(V2-V1)\left(1-e^{\frac{-(t-TD1)}{TAU1}}\right)$

TD2 to TSTOP $V1+(V2-V1)\left(1-e^{\frac{-(t-TD1)}{TAU1}}\right)+(V1-V2)\left(1-e^{\frac{-(t-TD2)}{TAU2}}\right)$

Генераторы частотно моделированной частоты

SFFM generator производит сигнал, который представляет результат частотной модуляции одной синусоидой другой. Математически это представлено так:

$$V = VO + VA\sin\left(2\pi FCt + MDI\sin(2\pi FSt)\right)$$

где параметры следующие

VO	Постоянная составляющая (DC offset) напряжения.
VA	Амплитуда несущей.
FC	Частота несущей.
FS	Частота сигнала.
MDI	Индекс модуляции.

Генераторы кусочно-линейных форм сигнала

Piece-wise Linear generator используется для создания аналоговых сигналов, которые слишком сложны для импульсного генератора или для воспроизводства измеренных сигналов. Выходной сигнал описывают, используя пары значений для времени и выходной амплитуды. Выход генератора в промежутках времени затем линейно интерполируется.

Диалоговая форма генератора кусочно-линейных форм состоит из графического редактора на котором вы можете перетаскивать точки данных. Операции можно суммировать так:

- Щелчком левой клавиши мышки поместите новую точку данных.
- Чтобы переместить точку данных, перетащите её, используя левую клавишу мышки.
- Чтобы удалить точку данных щёлкните правой клавиши мышки.

Есть следующие ограничения:

- Всегда должна быть точка при нулевом времени, хотя её y-значение может быть изменено, то есть, вы можете только перетаскивать их вверх или вниз.
- Если вы хотите создать вертикальный фронт, графический редактор разделит две точки данных минимальным значением времени подъёма/спада.

Если у вас есть табличные данные, легче может оказаться режим ручного редактирования. В этом случае каждая точка данных задаётся свойством формы V(t). Следующий пример показывает, как это работает.

```
V(0)=0
V(2n)=0
V(3n)=1
V(5n)=1
V(6n)=0
```

Если количество данных очень велико, может оказаться легче использовать *FILE generator*.

Файловые генераторы

File generator используется для испытания схемы аналоговым сигналом, который задаётся серией временных точек и значений данных, содержащихся в ASCII файле. Это очень похоже на работу с кусочно-линейным генератором, исключая то, что значения данных находятся вовне, а не в свойствах устройства.

Диалоговая форма имеет только одно поле, в котором задаётся имя файла данных. Нет предопределённых выражений для этих файлов, и файл должен располагаться в той же директории, что и файл проекта, если не задан полный путь к файлу.

Формат файла данных

Файл ASCII данных должен быть отформатирован парами время/напряжение для каждой линией, разделёнными пробелом (пробелы или табуляция, а не запятые). Значения времени должны последовательно возрастать и все значения должны быть просто числами с

плавающей точкой (суффиксы не допустимы).

Пример

Следующий пример файла данных производит три цикла пилообразных импульсов с временем подъёма 0.9ms, временем спада 0.1ms и амплитудой 1V.

```
0          0
9E-4       1
1E-3       0
1.9E-3     1
2E-3       0
2.9E-3     1
3E-3       0
```

Аудио генераторы

Audio generator используется для испытания схемы файлом Windows WAV. В соединении с *Audio graphs* (аудио графиками) возможно услышать результаты воспроизведения звукового сигнала через симулируемую схему.

- Имя файла подразумевается с предопределённым расширением WAV, файл должен располагаться в той же директории, что DSN файл, если не задан полный путь к файлу.
- Амплитуда может быть задана либо в терминах максимального абсолютного значения для положительной и отрицательной полуволн, либо как значение от пика до пика.
- Может быть задана постоянная составляющая (DC offset); если она нулевая, выходное напряжение осциллирует около нуля.
- Для стерео WAV файлов вы можете выбрать, какой канал будет проигрываться, либо определить данные как моно.

Цифровые генераторы

Есть пять подтипов цифровых генераторов (digital generator):

- *Single Edge* — единственный переход от низкого к высокому или от высокого к низкому уровням.
- *Single Pulse* — пара переходов в противоположных направлениях, которые вместе формируют положительный или отрицательный импульс. Вы можете задать либо времена каждого фронта (начальное и конечное время), либо начальное время и ширину импульса.
- *Clock* — последовательность импульсов, разделённых паузами. Вы можете задать начальное значение и время первого фронта, но можно задать период или частоту. Период задаёт время целого цикла, не ширину единственного импульса или паузы.

- *Pattern* — наиболее гибкий и может, фактически, генерировать любые другие типы. *Pattern generator* определён в терминах следующих параметров:

  Initial State Это значение в нулевой момент времени, а также значение, используемое при поиске рабочей точки схемы в смешанном режиме.

  First Edge Это время, когда шаблон действительно стартует; выход будет оставаться в значении начального состояния, пока не придёт время.

  Timing Каждый шаг шаблона может получить то же время (равное времени импульс/пауза) или может иметь разные времена для высокого и низкого уровня. В этом случае ширина *Pulse* задаётся временем для значений логической «1» и *Space Time* задаётся для значений логического «0».

  Transitions Выход может быть определён для постоянного сигнала до окончания симуляции с повторением шаблона или для только фиксированного числа переходов.

  Bit Pattern Предопределённый шаблон — это просто альтернатива последовательности высокое-низкое состояние. Альтернативно, может быть задана строка шаблона. Строка шаблона может состоять из следующих символов:

  0,L Выходной сигнал переходит на строго низкий уровень (заметьте, «L» в верхнем регистре).

  1,H Выходной сигнал переходит на строго высокий уровень (заметьте, «H» в верхнем регистре).

  1 Выходной сигнал переходит на слабо низкий уровень (заметьте, «l» в нижнем регистре).

  h Выходной сигнал переходит на слабо высокий уровень (заметьте, «h» в нижнем регистре).

  F,f Выходной сигнал на плавающем уровне.

- *Script* — генератор будет управляться скриптом DIGITAL BASIC. Генератор получает доступ к скрипту через декларацию переменной PIN с тем же именем, что и ссылка генератора.

ПРОБНИКИ

Обзор

Пробник (*probe*) — это объект, который может быть задан для записи состояния цепи, к которой он подключён.

Есть два типа пробников:

- *Voltage probes* (пробники напряжения) — они могут использоваться и для аналоговой, и для цифровой симуляции. В шаблоне они записывают действительные уровни напряжения, хотя в последнем случае они записывают логические уровни и их длительность.
- *Current probes* (пробники тока) — они могут использоваться только для аналоговой симуляции и должны располагаться на проводе так, чтобы провод проходил через пробник. Направление измерения отображается на графике токового пробника.

  Вы не можете использовать токовый пробник для цифровой симуляции или на шине.

Пробники в большинстве случаев используются при симуляции, основанной на графиках (Graph Based Simulations), но могут использоваться и при интерактивной симуляции для отображения данных рабочей точки и для части схемы.

Размещение пробников

Чтобы разместить пробник:

1. Вначале выберите либо *Voltage Probe Mode*, либо *Current Probe Mode* иконку. ISIS покажет предварительный вид пробника в *окне предпросмотра*.
2. Используйте иконки поворота и отражения для придания пробнику нужного положения, в зависимости от того, как вы их намерены разместить.

   Заметьте, что для пробника тока ориентация важна. Направление измерения тока указано стрелкой, заключённой в кружок, формирующий символ пробника.
3. Переместите мышку в окно редактирования, нажмите левую клавишу мышки и перетащите пробник в нужное место. Щёлкните левой клавишей мышки ещё раз.

   Вы можете размещать пробник непосредственно на существующем проводе так, чтобы его точка подключения касалась провода. Но вы можете разместить несколько пробников на свободном месте вашего проекта и соединить со схемой позже.

Когда пробник размещён так, что не соединяется ни с одним из существующих проводов, он получает предопределённое имя в виде знака вопроса «?», чтобы показать, что он не подключён. Когда пробник присоединён к цепи (возможно, при размещении его непосредственно на проводе), ему присваивается имя цепи или, если цепь сама не имеет имени, он получает ссылку на компонент и имя вывода, который первый подключён к цепи. Имя пробника будет автоматически обновляться, когда он отсоединяется или когда перетаскивается от одной цепи к другой. Вы можете присвоить пробнику собственное имя,

отредактировав пробник, но в этом случае имя остаётся постоянным и не будет обновляться.

Установки пробника

Диалоговая форма *Edit Probe* позволяет подправить два параметра:

LOAD Resistance (сопротивление нагрузки)

Пробник напряжения может быть задан так, чтобы иметь сопротивление нагрузки для схемы. Это полезно, там где нет пути протекания постоянного тока на землю от точки измерения.

Record Filename

Оба побника, и напряжения и тока, могут записывать данные в файл, который может затем проигрываться с помощью *Tape Generator* (генератор записи). Эта возможность позволяет вам создавать тестовые сигналы, используя одну схему, а затем проигрывать её в другой.

См. «Записи и сегментирование», где информации больше.

ИСПОЛЬЗОВАНИЕ SPICE МОДЕЛЕЙ

ОБЗОР

Многие из основных производителей компонентов сегодня поддерживают SPICE совместимые модели для симуляции для своей линейки продуктов. Поскольку симулятор PROSPICE основан на оригинальной версии Berkeley SPICE, у вас могут быть некоторые проблемы с использованием таких моделей. Однако есть и несколько решений, с которыми вам следует познакомиться до попыток использовать модели других производителей:

- SPICE модель — это ASCII файл спецификации (netlist). Он совсем не содержит графической информации и не может быть непосредственно размещён на схеме. Это означает, что вам нужен элемент библиотеки ISIS для этого устройства, связанный с этой SPICE моделью. К сожалению, нет стандартного формата файлов для любой библиотечной части схемы, так что вам обычно следует нарисовать её самостоятельно.

- В netlist SPICE модель вызывается ссылкой на подсхему, используя X карту. Связывание узлов схемы с узлами модели определяется порядком, в котором узлы схемы задаются в X карте. Например,

  ```
  XU1 46 43 32
  ```

  означает, что узел 46 схемы соединён с узлом 1 модели. Аналогично узел 43 соединён с узлом 2, а узел 32 соединён с узлом 3. К сожалению, эта схема означает, что узлы модели не именованы. И что ещё хуже, крайне редко есть некая согласованность между номерами узлов модели и номерами физических выводов устройства — хотя бы то, что физические корпуса могут иметь не соединённые с кристаллом выводы, и это сложно оформить при нумерации узлов SPICE модели. Есть также ряд трудностей с многоэлементными устройствами, как TL074 — имеет ли модель один ОУ или четыре?

  На практике это означает, что нужны некоторого рода явные кросс-ссылки, чтобы сказать ISIS, какие номера узлов SPICE модели использовать для каждого вывода устройства, поскольку ни имена выводов ISIS, ни номера выводов не могут использоваться по причинам, указанным выше. Было введено специальное свойство устройства SPICEPINS, чтобы сделать это как можно более безболезненно.

- Существует множество вариантов SPICE, начиная со SPICE 2, для стандарта которого были написаны большинство появившихся моделей. Однако значительное число производителей объявило, что их модели имеют совместимость с PSPICE™, коммерческой версии SPICE 2, которая была разработана в значительной мере далеко от стандарта оригинала. Пока SPICE3F5 имеет много особенностей, не поддерживаемых PSPICE™, PSPICE ™ имеет ряд типов примитивов, которые различны, и также использует разный синтаксис для некоторых из нововведений, которые были добавлены в оба продукта. Реализация этого, конечно, означает, что модели специально написанные для PSPICE™ могут не работать с PROSPICE.

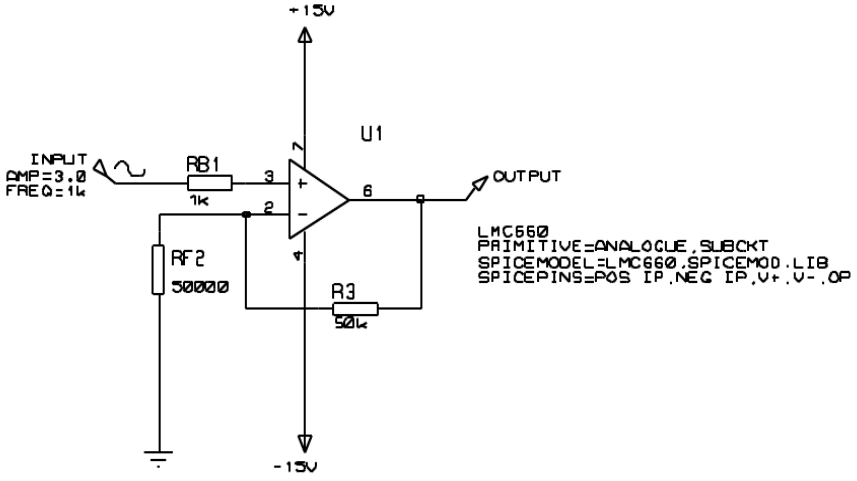
  Вкратце, если модель других производителей не работает правильно, первое, что следует проверить, это для какой версии, SPICE 2 или SPICE 3, она была написана.

Действительно хорошая новость — это то, что у нас собрано вместе большое количество (более 1500 к моменту написания этого руководства) моделей от производителей, и мы создали соответствующие элементы библиотеки ISIS для них.

Вы должны, конечно, понимать, что мы можем не согласиться нести ответственность за точность и даже функциональность этих моделей, и что и мы, и производители озабочены этим отказом от ответственности за потери любого рода возникающие при их использовании. В любом случае, никогда нет гарантий, что симуляция схемы будет в точности отражать работу реального устройства, и что всегда рекомендуется создать и протестировать физический прототип до начала массового производства.

ИСПОЛЬЗОВАНИЕ SPICE ПОДСХЕМ (SUBCKT ОПРЕДЕЛЕНИЕ)

Наш первый пример — это модель LMC660 операционного усилителя, которая есть в SPICE файле SPICEMOD.LIB. Этот пример можно найти также в проекте SPICE1.DSN в директории примеров, и он показан ниже.



Компонент операционный усилитель имеет три присвоенных свойства — PRIMITIVE, SPICEMODEL и SPICEPINS. Рассмотрим каждое из них по порядку:

PRIMITIVE=ANALOGUE,SUBCKT

Это задание всегда одинаково для компонентов, которые моделируются как SPICE подсхемы. Оно означает, что компонент будет симулироваться непосредственно SPICE3F5 и что он представлен подсхемой SPICE, а не действительным SPICE примитивом.

SPICEMODEL=LMC660,SPICEMOD.LIB

Эта строка показывает имя используемой подсхемы и имя ASCII файла, который содержит определение подсхемы. Альтернативно вы можете использовать

SPICEMODEL=LMC660

SPICEFILE=SPICEMOD.LIB

Некоторые производители добавляют много моделей в один файл, другие только одну в файле. Это ничего не значит. Имя подсхемы должно в точности совпадать с тем, что используется в файле модели, так что, если подсхема была названа как LMC660/NS, вы должны включить именно этот текст в свойство SPICEMODEL.

Файлы SPICE моделей ищутся в текущей директории и в *Module Path*, как задано в диалоговой форме *Set Paths*.

```
SPICEPINS=POS IP,NEG IP,V+,V-,OP
```

Свойство SPICEPINS задаётся с целью привязать имена выводов ISIS к номерам выводов SPICE модели. Чтобы понять, как это свойство используется, вам нужно взглянуть на комментарии в оригинальном файле SPICE модели:

```
*//////////////////////////////////////////////////////
*LMC660AM/AI/C CMOS Quad OP-AMP MACRO-MODEL
*//////////////////////////////////////////////////////
*
* connections:      non-inverting input
*                   |   inverting input
*                   |   |   positive power supply
*                   |   |   |   negative power supply
*                   |   |   |   |   output
*                   |   |   |   |   |
*                   |   |   |   |   |
.SUBCKT LMC660      1   2  99  50  41
*
```

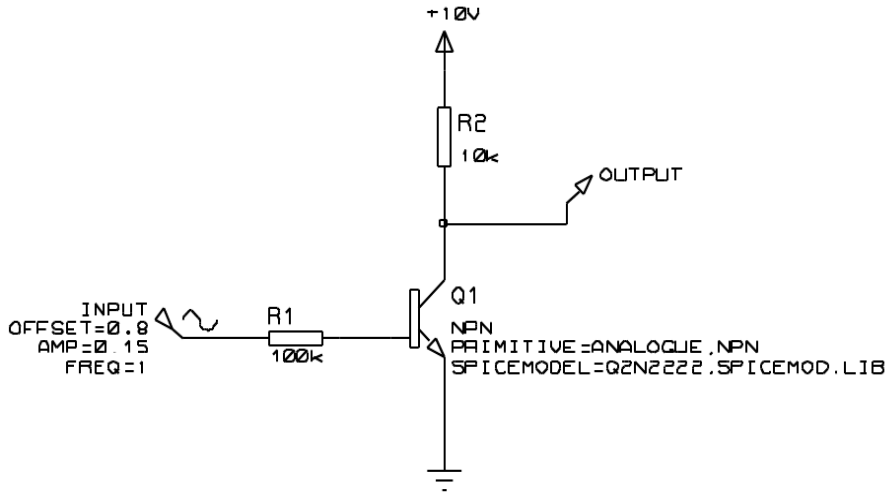
На «диаграмме» перечисляются функции выводов и их последовательность в строке SUBCKT. Существенно, что номера, присвоенные относительно внутренних определений модели, фактически не важны для наших целей. Ключевая информация — это то, что неинвертирующий вход первый вывод, инвертирующий вход второй и т.д. Этим определяется, как вы сконструируете значение свойства SPICEPINS, а именно в том, что вы дадите имена выводам ISIS в соответствующей последовательности. Имена выводов с пробелами допустимы, но будьте внимательны, чтобы не оставлять пробелов перед запятыми. Альтернативно, вы можете заключить каждое имя вывода в кавычки — это важно, если у вас есть имена выводов, которые содержат запятые.

Вам нужно, конечно, знать имена выводов ISIS, которые часто бывают скрытыми. Это решается перемещением мышки на конец вывода компонента, а затем следует прочитать строку состояния.

ИСПОЛЬЗОВАНИЕ SPICE МОДЕЛИ (КАРТА МОДЕЛИ)

Для полупроводниковых приборов, особенно диодов, транзисторов, JFET и MOSFET, SPICE модели обычно задаются в терминах единственной MODEL card (карты модели). Она перечисляет значения параметров для подходящего SPICE примитива. Процедура похожа на ту, что есть для SUBCKT модели, но в действительности несколько проще, поскольку не происходит преобразования имён выводов. Дополнительно разделы библиотеки ISIS уже содержат разные типы SPICE примитивов — вы найдёте их в ASIMMDLS.LIB.

Наш второй пример — это транзистор 2N2222, он вновь есть в ASCII файле SPICEMOD.LIB. Также можно найти пример SPICE2.DSN в директории *Samples*.



Транзистор имеет два присвоенных свойства — PRIMITIVE и SPICEMODEL.

Рассмотрим каждое из них последовательно:

PRIMITIVE=ANALOGUE,NPN

Это присваивание показывает, что устройство будет симулироваться непосредственно PROSPICE, как примитив типа NPN. Если вы выбрали диод, транзистор, JFET или MOSFET из библиотеки ASIMMDLS, он уже будет иметь подходящее присвоение PRIMITIVE.

SPICEMODEL=Q2N2222,SPICEMOD.LIB

Значение SPICEMODEL такое же, что и для SUBCKT модели. Оно задаёт имя используемой модели и SPICE netlist файл, который её содержит. Большинство производителей включают много определений MODEL в единственный файл.

Файлы SPICE model ищутся в текущей директории и через *Module Path*, как он задан в диалоговой форме *Set Paths*.

БИБЛИОТЕКИ SPICE МОДЕЛЕЙ

Библиотеки других производителей SPICE моделей, приходящие с PROSPICE, содержатся в двоичных библиотечных файлах, похожих на те, что есть для устройств ISIS и библиотек символов. Сделано это было из двух соображений:

- Многие файлы моделей исключительно малы и многочисленны, и на всё тратится огромное количество места на жёстком диске с большим размером кластера, если оно хранится индивидуально.
- Когда много моделей содержатся в единственном ASCII файле, PROSPICE вынужден проходить весь файл, чтобы использовать только одну модель, а это довольно медленно.

Когда SPICE модель содержится в библиотеке SPICE моделей (SML), синтаксис для её поддержки для раздела ISIS библиотеки сильно отличается от нужного. Например, LMC660 в первом примере будет иметь вместо нужных следующие свойства:

PRIMITIVE=ANALOGUE,SUBCKT

SPICEMODEL=LMC660

SPICEPINS=POS IP,NEG IP,V+,V-,OP

SPICELIB=NATSEMI

(как ранее)

(как ранее)

Свойство SPICELIB задаёт имя для файла SPICE модели библиотеки, которое содержит элемент. Эти файлы ищутся в текущей директории, и в *Module Path*, как задано в диалоговой форме *Set Paths*.

Библиотеками SPICE model можно манипулировать в командной строке инструментов PUTSPICE.EXE и GETSPICE.EXE, хотя мы не ожидаем, что обычные пользователи создадут или соберут достаточное количество моделей, чтобы обеспечить их использование. Запуск любой из этих программ без параметров даст информацию об их использовании.

**SPICE SCRIPTS*

Реально возможно ввести определение типа SPICE модели непосредственно в ISIS, а затем вызвать модель для подходящего компонента на схеме. Например, модель транзистора для 2N2222 должна быть вписана в ISIS скрипт следующим образом:

```
*SCRIPT SPICE
.MODEL Q2N2222 NPN(IS=3.108E-15 XTI=3 EG=1.11 VAF=131.5 BF=300
NE=1.541
+ISE=190.7E-15 IKF=1.296 XTB=1.5 BR=6.18 NC=2 ISC=0 IKR=0 RC=1
+CJC=14.57E-12 VJC=.75 MJC=.3333 FC=.5 CJE=26.08E-12 VJE=.75
+MJE=.3333 TR=51.35E-9 TF=451E-12 ITF=.1 VTF=10 XTF=2)
*ENDSCRIPT
```

Свойство транзистора SPICEMODEL будет затем изменено на

SPICEMODEL=Q2N2222

то есть, без задания имени файла.

Эта возможность может быть особенно полезна, когда модель получена на бумаге, или когда вы хотите интерактивно поэкспериментировать со значениями параметров.

ПОДДЕРЖКА АВАРИЙ ПРИ СИМУЛЯЦИИ МОДЕЛИ

Модели подсхем очень различаются по сложности и разработке. И как результат, одни модели симулировать легче, чем другие. В качестве основного правила — если PROSPICE не может симулировать вашу схему с её преобразованной моделью, тогда и ничто не сможет.

Мы часто находили, что проблемы в большинстве своём усугубляются плохим проектом схемы. Если у вас есть примечания в справочном листке к устройству, тогда сравните его с вашей схемой, чтобы убедиться, например, нет ли несоответствия во входном токе для входа в рабочей точке. Разработчики моделей всегда стремятся сделать модель работающей с

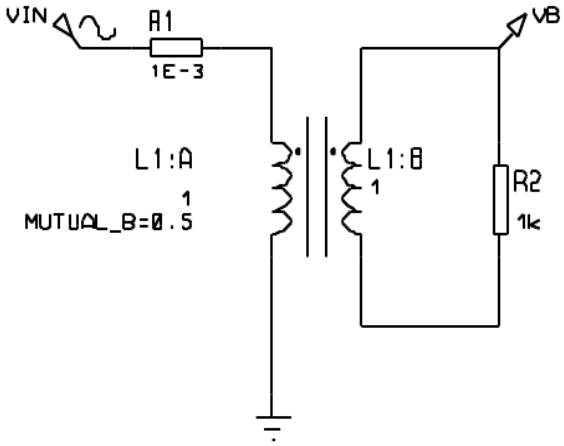
симулятором, что отмечают в примечаниях, а не в общем случае.

ProSPICE можно сделать более стабильным (но менее точным), если увеличить значение GMIN. По умолчанию это 1E-14, так что есть смысл попробовать1E-13 и 1E-12. При значениях около 1E-6, похоже, любые результаты симуляции совсем не будут иметь отношения к реальности, так что указывайте наинизшие из значений, но из возможных.

ШИНЫ ПИТАНИЯ И ЗЕМЛИ

Почему вам нужна земля

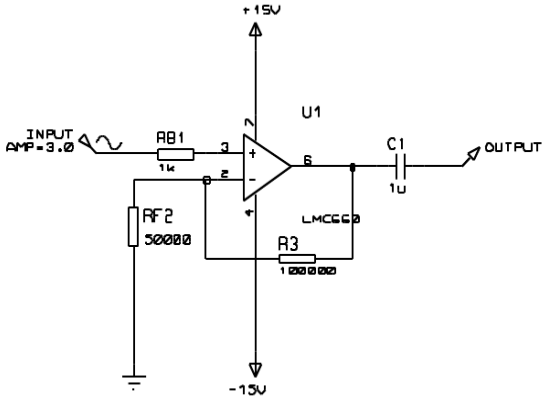
Все симуляции требуют, чтобы была задана земля, иначе размещение пробника на участке цепи не имеет смысла, так как напряжение на этом участке должно определяться в терминах опорной точки. Фактически есть ещё требование, чтобы все части схемы имели путь по постоянному току к земле, поскольку зондирование точки на «плавающем» участке цепи, бессмысленно. Например, на схеме ниже измерение напряжения в VB бессмысленно,



поскольку вторая сторона — плавающая. Теоретически, мы можем предложить двух выводные пробники (наподобие реального мультиметра), но возникают серьёзные математические трудности при расчёте схемы без земли, и, главное, Berkeley University не обращается к ним в SPICE3F5.

Ключевым, таким образом, будет то, что все секции вашей схемы должны иметь путь по постоянному току к земле. Хорошая новость в том, что ProSPICE проверяет это для вас и сообщит в предупреждении обо всех цепях, которые не отвечают этому критерию. В большинстве случаев симуляция на этом прекращается.

Другая конфигурация схемы, которая могла иметь трудности в прошлом, показана ниже:



Здесь наличие разделительного конденсатора C1 означает, что пробник на выходе не имеет пути для постоянного тока к земле. Следовательно, симулятор не может рассчитать рабочую точку для выхода, поскольку при вычислении рабочей точки все конденсаторы выбрасываются из цепи. Мы выбрали решение для этой проблемы в использовании конденсаторов с очень маленькой утечкой. Поэтому, в отсутствии любых других путей для постоянного тока, рабочая точка на выходе вычисляется с полностью разряженным C1.

Конденсаторы без утечки определяются свойством управления симуляцией GLEAK, которое имеет значение по умолчанию 1E-12Mho. Установка этого значения в нуль устраняет утечку конденсатора, как в традиционных SPICE симуляторах.

Исключая симуляцию внутренней схемы DRAM памяти, мы не видим каких-то проблем с этой схемой, и она предохраняет начинающих от множества странных сообщений об ошибках. В любом случае, реальные конденсаторы обычно имеют утечку значительно большую, чем миллион мегаом.

Цепь земли в схеме может быть определена либо явно, либо скрыто. Явная земля определяется размещением не именованного *Ground* контакта, как на нижнем конце RF2 на рисунке выше. Вы также можете обозначить провод текстом GND, если места мало.

Неявная земля может быть создана с помощью питающей шины, генератора с единственным выводом, нагрузочного пробника или цифрового выхода, как, впрочем, и использованием модели, которая имеет внутренний заземлённый узел.

Шины питания

PROSPICE распознаёт некоторое количество объектов, как шины питания (power rails); правила следующие:

- Любая сеть, названная GND или VSS, обосновывается как опорная земля. А насколько это затрагивает PROSPICE, GND и VSS означают то же самое.
- Цепи, названные VCC и VDD, считаются логическими шинами питания, и будут трактоваться симулятором DSIM как логическая «1». Как и с GND и VSS, PROTEUS VSM принимает, что эти два имени относятся к той же цепи, если вы действительно хотите отделить логическое питание, вы должны выбрать разные имена для цепей.
- Контакты питания с именами в виде +5, -5 или +10V, -10V принимаются как предопределённые фиксированные шины питания с опорой на землю. Символы «+» и «-» критичны, а этикетки, как 5.0V нет.

  Соединение двух таких этикеток с разными значениями с одной и той же цепью — это ошибка.

  Создание шины питания также неявно задаёт землю.

Дополнительная возможность, добавляемая как часть схемы для симуляции смешанного режима, такова — модели логических микросхем могут рассматриваться как уже включённые. Это позволяет вам рисовать схему, как показанная ниже, и получать ощутимые результаты без прорисовки явных цепей питания.

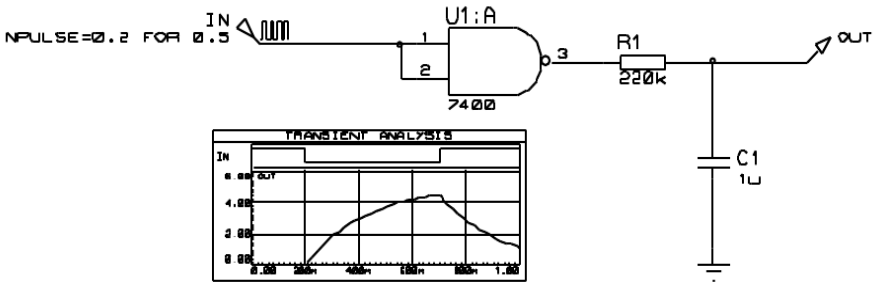


Схема работает, поскольку Interface Model, определённая для 7400, включает свойство VOLTAGE. Что подразумевает создание 5V батарейки между скрытыми выводами питания 7400 (VCC и GND), таким образом, цепь VCC приобретает потенциал 5V. Остальные элементы, соединённые с VCC, включая DAC объект на выходе 7400, будут видеть это напряжение.

Дальнейшее обсуждение Interface Models (модели интерфейса) вы найдёте в соответствующем разделе.

НАЧАЛЬНЫЕ УСЛОВИЯ

Обзор

Какой бы тип симуляции не был выбран, первое, что делает PROSPICE, это рассчитывает рабочую точку схемы — условия устойчивого состояния, предшествующие появлению сигналов на входе. Есть два отдельных режима операций, ассоциированных с расчётом этих значений:

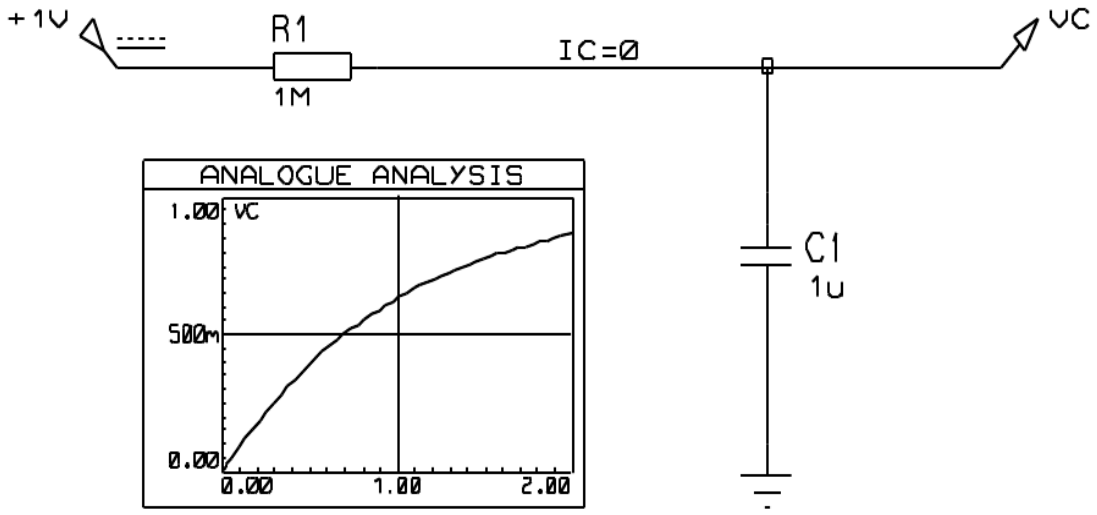
- При поиске рабочей точки считается, что все конденсаторы заряжены, а через индуктивности протекает ток. В этом случае Transient Analysis будет показывать поведение схемы, как если бы питание было подано непосредственно до времени начала симуляции, так что при старте симуляции всё уже успокоилось. Это режим операций по умолчанию, который даёт возможность установить начальные условия для отдельных компонентов и/или узловых напряжений, подходящие для многих целей.
- При поиске рабочей точки все конденсаторы считаются закороченными, а индуктивности оборванными. В этом случае Transient Analysis покажет поведение схемы таким, как если бы питание было подано в нулевой момент времени. Этот режим операций может быть выбран сбросом флажка *Compute Operating Point* на графике.

В любом случае возможно задать начальные условия для отдельных компонентов или узловых напряжений. Это особенно полезно для схем осцилляторов или таких, где работа схемы зависит от того, чтобы отдельные конденсаторы были разряжены до начала испытания. Действительно, концепция устойчивого состояния рабочей точки не имеет

смысла для осцилляторов, а симуляция может полностью остановиться, если некоторые начальные условия не заданы.

Задание начальных условий для цепей

Самый лёгкий способ задать начальные условия, чаще всего, обозначить начальное напряжение для отдельной цепи. На схеме ниже это выполнено добавлением этикеток к проводам с текстом IC=0 для испытуемой цепи. Без этого присваивания PROSPICE будет вычислять значение устойчивого состояния напряжения на C1, то есть, 1 вольт, и график будет показывать VC, как горизонтальную прямую линию.



Для цепей, которые имеют внутреннее соединение только цифровых компонентов, вы должны использовать логические состояния для начальных условий; то есть, 1,0,H,L,HIGH,LOW,SHI,WHI,SLO,WLO или FLT, и присваивать им BS свойство (Boot State, состояние загрузки). Для смешанных цепей вы должны задавать начальное напряжение — это будет автоматически распространяться как логический уровень на цифровые компоненты.

Задание начальных условий компонентов

Эта опция доступна только тогда, когда PROSPICE не вычисляет начальную рабочую точку, то есть, когда флажок Initial DC Solution на графике сброшен. В этом случае все узловые напряжения будут нулевыми в нулевой момент времени, исключая цепи с заданным свойством «начальные условия», как это описано выше. В этом случае можно задать начальные условия для отдельных компонентов. Например, вы можете добавить свойство

IC=1

для C1 в предыдущем примере, так что C1 начнёт работу со значением устойчивого состояния в 1 вольт.

Детали свойства IC поддерживаются разными SPICE примитивами, и даны в разделе о моделировании.

Некоторым образом неприятно, что эта опция не доступна при вычислении рабочей точки, но это то, как SPICE3F5 было кодировано в Berkeley. Однако мы добавили свойство

PRECHARGE в качестве лекарства от этой болезни.

Свойство NS (NODESET, установка узла)

Иногда при отказе симуляции на стадии поиска рабочей точки может быть полезно дать SPICE «подсказку» в виде начальных значений для отдельных цепей. Это отличается от установки начальных условий в том, что значения даются только для использования в первой итерации, а затем цепь становится «плавающей» до получения значения, к которому сходятся уравнения матриц. Это не более, чем помощь в сходимости, и не скажется на определении реальной рабочей точки.

Такая подсказка для сходимости может быть задана при использовании свойства цепи NS, так что размещая этикетку провода с текстом NS=10, вы задаёте начальное значение в 10 вольт для этой цепи.

Свойство PRECHARGE

Ещё одна опция для задания начального условия — это свойство PRECHARGE (предзаряд). Оно может присваиваться любому конденсатору или индуктивности в цепи и задавать либо напряжение на элементе, либо ток, протекающий через него, соответственно.

Свойство PRECHARGE — это специфическое добавление Labcenter к SPICE, и отличается от свойства IC в том, что оно применимо в зависимости от того, установлен ли флажок *Initial DC Solution*.

МОДЕЛИРОВАНИЕ ТЕМПЕРАТУРЫ

Ядро SPICE3F5 предлагает широкую поддержку для моделирования температурных эффектов. Схема работает следующим образом:

- Есть глобальное свойство TEMP, которое, если ничего другого не сделано, будет отнесено ко всем компонентам схемы.
- Температура индивидуальных компонентов может быть задана через их собственное свойство TEMP.
- Когда параметры модели устройства меняются под действием температуры, отличной от той, при которых они были измерены, эта температура может быть задана глобально и/или индивидуально через свойство TNOM.

Моделирование температуры поддерживается у резисторов, диодов, JFET, MESFET, BJT (полевые и биполярные транзисторы), и уровня 1, 2 и 3 MOSFET. BSIM модели (ожидаемо) были созданы для заданной температуры. В цифровые примитивы температурная зависимость не встраивалась.

Дополнительно, очень важно уяснить, что большинство эквивалентных моделей схем IC не имеют моделей корректных температурных эффектов. Причина этого в том, что эти модели используют идеальные управляемые источники и другие макро-модельные примитивы, а не построены из точных копий внутренних компонентов. А когда такие примитивы используются, маловероятно, что модели покажут корректную температурную зависимость поведения, пока кто-то не возьмёт на себя труд сделать это.

ПАРАДИГМА ЦИФРОВОЙ СИМУЛЯЦИИ

Модель девяти состояний

Вы можете думать, что цифровая симуляция будет моделировать только высокое и низкое состояние, но, фактически, DSIM моделирует всего девять отдельных состояний:

Тип состояния	Ключевое слово	Описание
Питающее высокое	**PHI**	Питающая шина логической 1.
Сильное высокое	**SHI**	Активный выход логической 1.
Слабое высокое	**WHI**	Пассивный выход логической 1.
Плавающее	**FLT**	Плавающий выход — высокий импеданс.
Неопределённое	**WUD**	Среднее напряжение от аналогового источника.
Спорное	**CON**	Среднее напряжение от цифрового конфликта.
Слабое низкое	**WLO**	Пассивный выход логического 0.
Сильное низкое	**SLO**	Активный выход логического 0.
Питающее низкое	**PLO**	Питающая шина логического 0.

В сущности, данные состояния содержат информацию об их полярности: high, low или mid-way, — и их силе. Сила измеряется величиной выходного тока как источника или для потребителя, и становится существенной, когда два или более выхода соединяются с той же цепью.

Например, если выход с открытым коллектором присоединяется через резистор к VCC, тогда при переходе выхода в низкое состояние и Слабое высокое (Weak High), и сильное низкое (Strong Low) состояния применимы к цепи. Сильное низкое состояние победит, и цепь перейдёт в низкое состояние. С другой стороны, если два выхода с трёхстабильным состоянием становятся активы в цепи, а переходят в противоположные состояния, ни один из выходов не «побеждает», а результатом будет состояние Спорное (Contention).

Эта схема позволяет DSIM симулировать выходные цепи с открытым коллектором или открытым эмиттером и подтягивающими резисторами, а также цепи, в которых трёхстабильные выходы противостоят друг другу через резисторы — разновидность мультиплексора бедняков, если вам так нравится. Однако важно помнить, что DSIM — это только цифровой симулятор и не может моделировать поведение, которое становится бесспорно аналоговым. Например, подключение избыточно большого резистора к TTL входам будет работать хорошо в DSIM, но откажет на практике, из-за недостаточного тока подтяжки от входов.

Неопределённое состояние

Когда вход цифровой модели не определён, это распространяется через модель согласно тому, что может быть описано как правила здравого смысла. Например, если вентиль И имеет

низкий уровень, тогда выход будет в низком состоянии, но если все, кроме одного, входы в высоком состоянии, и этот вход не определён, тогда выход будет неопределённым.

- Фронт переключающегося устройства требует перехода от положительно определённого логического 0 к логической 1 (или наоборот), чтобы фронт был обнаружен. Переход от 0 к 1 не определён и не возвращается для вычисления фронта.
- Более сложные последовательные логические устройства (счётчики, защёлки и т.п.) будут часто иметь не определённые входы ни к логическому 0, ни к логической 1, согласно устройству их внутренней логики. Это не похоже на реальную жизнь!

Поведение плавающих входов

Обычная практика, если нет иного опыта, следует доверять тому факту, что свободные TTL входы ведут себя так, как если бы были подключены к логической 1. Эта ситуация может возникнуть и как результат пропуска соединения, и в том случае, когда вход присоединён к неактивному выходу с тремя состояниями. DSIM должен что-то предпринять в подобной ситуации, поскольку внутренние модели принимают истинное логическое поведение со входами, находящимися исключительно в высоком или низком состоянии.

Это учитывается через свойство FLOAT. Оно может присваиваться либо непосредственно компоненту, либо Interface Model. В частности, модель интерфейса (interface model) для элементов TTL имеет присваивание:

```
FLOAT=HIGH
```

которое гласит, плавающий вход должен интерпретироваться как уровень логической 1.

Чтобы задать, что плавающие входы должны быть логическим 0, используйте

```
FLOAT=LOW
```

Иначе плавающий вход будет считаться в неопределённом (Undefined) состоянии (см. выше).

Поддержка проблем

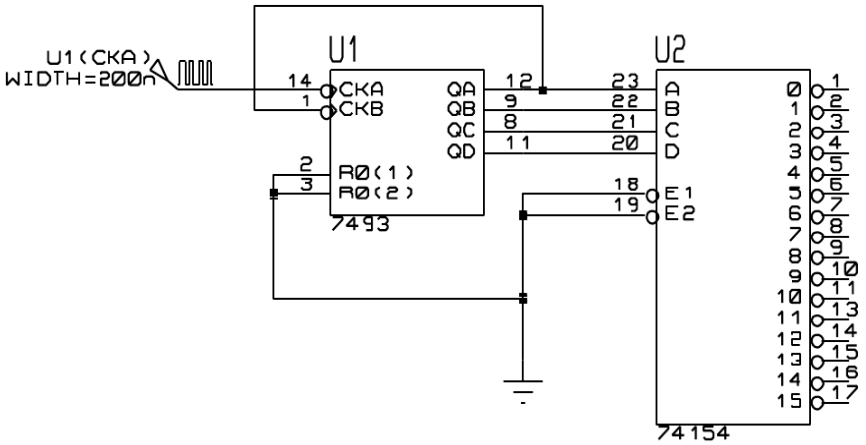
При разработке DSIM мы долго обсуждали, как поддерживать симуляцию моделей, подверженных очень коротким импульсам. Фундаментальная проблема в том, что при этих условиях основное положение парадигмы DSIM — модели ведут себя как чисто цифровые — начинает проваливаться. Например, реальная модель 7400 под действием входного импульса в 5ns генерирует на выходе какого-то рода импульс, но не тот, который встречается в спецификации логических уровней для TTL. Будет ли такой выходной импульс приводить в действие счётчик, это зависит от очень многих аналоговых явлений.

Самое лучшее, что можно предложить для учёта крайностей, это:

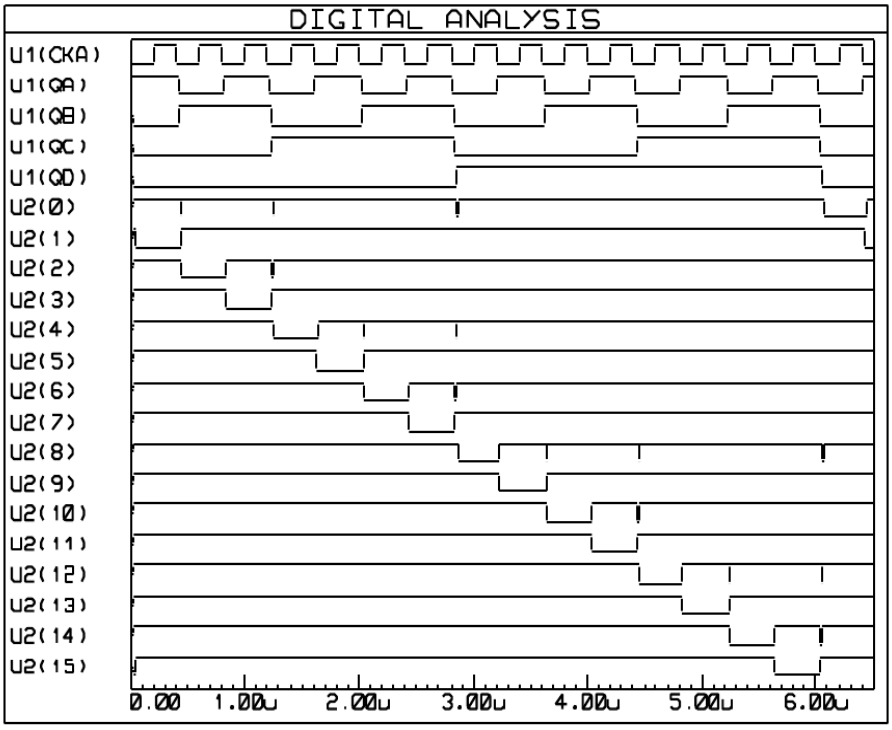
- Входной импульс в 1ns не передавать совсем.
- 20ns импульс передавать хорошо.

Где-то между ними вентиль перестанет передавать импульсы правильно и, скажем, будет сдерживать проблему. Это приводит нас к концепции *Glitch Threshold Time* (пороговое время проблемы), которая может быть дополнительным свойством модели наряду с обычными TDLH и TDHL.

Другая тонкая штука — будет ли проблема касаться входа или выхода модели. Чтобы как-то решить это, обратимся к дешифратору 4-16 счётчика сквозного переноса, как показано ниже.



Выходы счётчика сквозного переноса «расположены уступами», следовательно, возникает возможность того, что дешифратор будет генерировать фальшивые импульсы, когда входы проходят через промежуточные состояния. Эта ситуация показана на следующей диаграмме:



The above graph was produced with TGQ=0 for the 74154

Рассмотрим первую проблему примера на событии U1(QA); спад вначале побеждает подъём U1(QB) и промежуточное состояние входа проходит в дешифратор, примерно, в течение 10ns. Вопрос в том, что может ли дешифратор действительно отреагировать на это или нет, и даже более того, что случится, если колебания на входе продлятся только 1ns или 1ps? Ясно, в последних двух случаях реальное устройство не будет реагировать, и это говорит нам, что мы должны поддерживать проблему на выходе, а не на входе, поскольку в примере выше

входные импульсы относительно долгие и не будут рассматриваться проблемными при любых разумных критериях. Некоторые конкурирующие продукты могут внести в это путаницу, можно прогнозировать отклик даже в ситуации с 1ps!

Действительно интересная часть этой истории в том, что если вы соберёте схему, приведённую выше, она, возможно, не будет иметь проблем. Это очень плохая, естественно, разработка, но TDLH и TDHL модели '154 приблизительно 22ns, и это делает условия отклика на входное воздействие в 10ns порядком выше её возможностей. С индивидуальными компонентами мы пробовали, нет выходных импульсов, иных чем, возможно, лёгкие «просадки» питающего напряжения, насколько это измеримо.

Для поддержания управления поддержкой проблем все DSIM примитивы предлагается использовать с определяемым свойством *Glitch Threshold Time*, названным *TGxx*, где xx — это имя соответствующего выхода. Наши TTL модели определены так, чтобы эти свойства могли быть отменены для TTL компонентов, а значения затем переопределены с тем, чтобы *Glitch Threshold Times* усреднялось по основным задержкам распространения «low-high и high-low». Задание нулевого *Glitch Threshold Times* позволит выявить все проблемы, если вам удобнее такое поведение. График, показанный выше, был создан именно так, присвоением `TGQ=0` модели 74154.

И, наконец, важно уяснить, что если *Glitch Threshold Time* больше, чем либо low-high, либо high-low задержка распространения, тогда *Glitch Threshold Time* будет игнорироваться. Это для того, чтобы после входного фронта по истечении значимого времени задержки выход вентиля изменил своё значение — он не может «заглянуть» в будущее и увидеть, произойдёт ли событие на другом входе, которое может сбросить выход. Рассмотрите симметричный вентиль с временем задержки распространения в10ns и *Glitch Threshold Time* равным 20ns. В момент t=0ns вход переходит в высокое состояние, а при t=15ns переходит в низкое. Вы можете исключить это из распространения, переведя выход в высокое состояние при t=10ns и переходом в низкое при t=25ns, так что произведённый импульс будет 15ns шириной, и был бы подавлен, поскольку меньше, чем *Glitch Threshold Time*. Причина, по которой этого не произойдёт, в том, что при t=10ns выход должен перейти в высокое состояние, и он не может стать низким в ближайшие 20ns без шансов (как в нашем примере), чтобы второй фронт, приходящий при этом, произвёл выходной импульс, он должен быть подавлен! Когда же выход становится высоким при t=10ns, тогда второй фронт (при t=25ns) свободно может сбросить его. Вы должны хорошо это всё обдумать, чтобы понять это.

МОДЕЛИ ИНТЕРФЕЙСОВ СМЕШАННОГО РЕЖИМА (ITFMOD)

Обзор

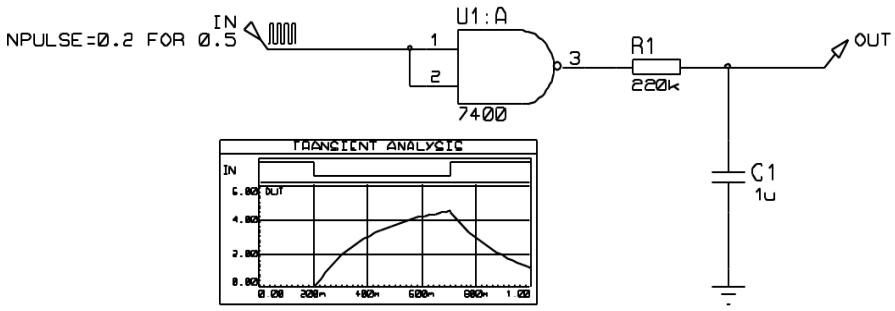
При разработке нашей схемы для симуляции смешанного режима в PROSPICE мы уделили много внимания проблеме того, как задать аналоговые характеристики устройствам цифровых семейств. Характеристики включают:

- Входной и выходной импеданс устройства.
- Логические пороги для входов устройств.
- Уровни напряжения для высокого и низкого состояний выхода.

- Времена подъёма и спада для выходов устройств.
- Определённое логическое состояние «плавающих» входов.

Схема, которая учитывает задание всех этих параметров для каждого из устройств в библиотеке TTL, скажем, была бы крайне громоздка.

Вдобавок, возникает важная проблема (для начинающих, во всяком случае) в задании питающего напряжения — есть тенденция выкладывать схемы, подобные той, что ниже, и ожидать разумных результатов. Здесь проблема, конечно, в неявном соглашении, что 7400 имеет 5В питающее напряжение, подводимое к его скрытым выводам, которые соединены с VCC/GND.



Все эти проблемы решаются введением в свойство компонента ITFMOD. Это очень похоже на свойство MODEL в том, что поддерживает ссылку на набор значений свойств, но также активирует специальный механизм в компиляторе netlist. По существу это работает следующим образом:

- Для любого устройства, которое имеет свойство ITFMOD, дополнительное определение модели вызывается в процессе создания netlist, которое задаёт управляемые параметры для ADC и DAC объектов, и также имена положительного и отрицательного выводов подключения питания. В схеме выше U1:A будет иметь ITFMOD=TTL.
- Получив имена выводов питания (VCC, GND в данном случае), ISIS создаёт специальный примитив и соединяет его с выводами питания. ISIS имена этого объекта похожи на те, что у объектов производных листов или моделей, так что схеме выше питание объекта будет вызвано U1:A_#P.
- Когда PROSPICE симулирует схему смешанного режима, он создаёт ADC и DAC объекты и подразумевает, что они «принадлежат» объектам, к которым они подключены. В случае вышеприведённой схемы DAC объект будет создан с именем U1:A_DAC#0000, поскольку он формирует интерфейс для выхода U1:A.

  Смысл в том, что при выполнении этого также просматриваются объекты интерфейса питания с тем же основанием имени, то есть, U1:A, и ищется U1:A_#P. Затем приходит инструкция U1:A_DAC#0000 — взять свойство из U1:A_#P, которое в свою очередь наследует свойства от заданной модели в оригинальном присваивании ITFMOD. Так объект DAC оперирует с параметрами, определёнными для семейства TTL логики.

- Каждый объект интерфейса питания также содержит батарею, которая назначается свойству VOLTAGE, данному в определении модели интерфейса. Модель TTL интерфейса имеет заданное VOLTAGE=5V.

  Это означает, что в вышеприведённой схеме батарея 5V включена между VCC и GND, поскольку это цепь, обозначенная как выводы питания для устройства 7400.

- Батареи имеют маленький внутренний импеданс (1 миллиом). А это означает, что если вы назначаете реальные шины питания VCC/VDD (разместив контакты питания или источник напряжения), тогда этим будет отменён уровень, определённый внутренними батареями — в мире симуляции большой ток через батареи ничего не значит!

Использование свойств ITFMOD

Существующие модели интерфейсов определены следующим образом:

TTL	Стандартная TTL серия (74 серия)
TTLLS	Низковольтная Шотки TTL (74LS серия)
TTLS	Стандартная Шотки TTL (74S серия)
TTLHC	Высокоскоростная CMOS TTL (74HC серия)
TTLHCT	Высокоскоростная CMOS TTL с TTL выходами (74HCT)
CMOS	4000 серия CMOS
NMOS	Микропроцессорного типа MOS схемы
PLD	PLD типа MOS схемы

Из этого следует, что любая новая цифровая модель может быть назначена фамилии устройств добавлением свойства, такого как

```
ITFMOD=TTL
```

Определения фамилии поддерживаются файлом ITFMOD.MDF, который хранится в директории моделей.

Каждое определение может содержать любое или все свойства, определённые для интерфейсов примитивов ADC и DAC. Дополнительно может быть задано следующее:

V+	-	Имя вывода плюса питания.
V-	-	Имя вывода минуса питания.
VOLTAGE	5V	Заданное по умолчанию рабочее напряжение.
RINT	1mΩ	Заданный импеданс внутренней батареи. Нулевое значение отключает батарею.
FLOAT	-	Задаёт HIGH или LOW значение для плавающего входа.

И, наконец, важно уточнить, что любое заданное свойство, например, TRISE, может быть отменено в родительском устройстве, так что, если вы хотите симулировать микросхему 4000 серии с большим временем переднего фронта, вы должны добавить TRISE=10u непосредственно в её список свойств.

ПОСТОЯННЫЕ ДАННЫЕ МОДЕЛИ

Некоторые модели симулятора, частично те, что для EPROMS и микропроцессоров, содержащие EEPROM или флэш память, способны «запомнить» данные между запусками симуляции, подобно их реальным прототипам. Это действие облегчается свойством MODDATA. Постоянные блоки данных модели сохраняются в файле DSN, и не удерживаются между сессиями PROTEUS, пока вы не сохраните проект.

Чтобы сбросить постоянные данные модели до начального состояния, используйте команду *Reset Persistent Model Data* из раздела *Debug* основного меню.

ЗАПИСЬ И РАЗБИЕНИЕ

Обзор

Уникальная особенность системы VSM в её способности делить большой проект на одну или более секций или частей и симулировать каждую из них индивидуально.

Есть два принципиальных преимущества, связанных с этим:

- В полностью интегрированных CAD схема всего проекта будет содержать несколько секций, которые вам не захочется или вы не сможете симулировать. Для предотвращения необходимости дробить проект требуется некий механизм для определения, какие компоненты в проекте действительно влияют на данный эксперимент при симуляции.

  ISIS делает это с учётом точек измерения и, в дальнейшем отступая от проекта, точек, управляемых тестовыми источниками и/или шинами питания.

- Если проект состоит из нескольких этапов, будет общим требованием видеть, как более поздние этапы выполняются, когда управляются выходами ранних этапов. Хотя это может быть получено симуляцией всех этапов вместе, но будет осуществляться неоправданно медленно.

  Разбиение позволяет результаты симуляции предыдущих этапов записать на «плёнку» (tapes) и воспроизвести как входной сигнал на следующих этапах.

  ISIS включает логику, чтобы сделать это либо с ручным управлением, либо автоматически. В последнем случае ISIS определяет, когда схема с заданным разбиением изменилась и новой симуляции повергается только то, что изменилось, или то, на чём скажутся произведённые изменения.

Операции с единственной частью

Многие эксперименты при симуляции будут состоять из тестирования единственной части проекта, изолированной от остального. Это обычно требует сигнала или сигналов на входе (или входах) этой секции, и затем наблюдения за происходящим в разных её точках.

Должно быть ясно, что сигналы могут добавляться размещением генераторов на подходящих входных проводах, и что работа схемы может наблюдаться путём размещения пробников. Однако эти действия сами по себе не изолируют тестируемую секцию от остальной части проекта.

Чтобы сделать это, вам нужно установить флажки *Isolate Before* на генераторах и флажки *Isolate After* на пробниках. Когда это сделано, только секция между точками изоляции будет компилирована в netlist и будет симулироваться.

Чтобы выполнить симуляцию секции, ISIS просматривает пробники, которые связаны с графиком, и проходит по компонентам и проводам (включая контакты внутренних соединений) наружу, пока не встретит одно из следующих условий:

- Больше нет компонентов для процесса.
- Достигнуты изолированные пробники или генераторы.
- Достигнуты записывающих устройства (tape) на входе или выходе.
- Достигнуты шины питания. Цепь превращается в питающую, если присоединена к контактам POWER или GROUND. GND или VCC этикетки на проводах НЕ делают этого.

Объекты записи

Объекты записи (Tape objects) имеют два отдельных применения в системе:

- Чтобы определить точки, в которых есть смысл игнорировать схему правее, когда схема левее симулируется. Такие точки обычно там, где низкий импеданс выходов управляет высоким импедансом входов. Это может быть также достигнуто изоляцией пробников.
- Чтобы поддержать средства для захвата состояния выходов одного из этапов проекта и использовать эти состояния для управления следующим этапом без предварительной симуляции первого этапа.

Чтобы разместить запись (Tape):

1. Выберите иконку *Tape Recorder Mode*.
2. Используйте иконки поворота и отражения для придания нужного положения записывающему устройству.
3. Переместите мышку в окно *редактирования* и нажмите левую клавишу мышки, перетащите записывающее устройство в нужное место и нажмите клавишу ещё раз.

   Вы можете разместить записывающее устройство (tape) непосредственно на

существующем проводе, разместив его так, чтобы точка соединения касалась провода. В качестве альтернативы вы можете разместить несколько записывающих устройств на свободном месте и соединить со схемой позже.

Жизненно важно размещать записывающее устройство в чувствительных местах, там где низкий импеданс управляет высоким. Если вы разместите записывающее устройство в другом месте, вы можете изменить свойства вашей схемы и испортить результаты, которые будут произведены.

Режимы записи

Чтобы дать максимальные возможности для пользовательского управления, записывающее устройство имеет три режима: AUTO, PLAY и RECORD. В последующем рассказе мы используем термины left и righ в смысле логической зависимости от структуры разбиения, чтения проекта от входа к выходу, слева-направо.

Режим AUTO

Это схема по умолчанию и определяет режим операций, при котором ISIS решает, какие части следует пересимулировать, а для каких можно использовать прежде полученные данные. Для большинства симуляций, которые требуют записи, режим AUTO оказывается наиболее полезен.

Это автоматическое определение основано на учёте всего текста, обнаруженного в sub-netlist, относящегося к секции проекта. Из чего следует, что если любые имена частей, значения, свойства, соединения и т.п. в этой секции менялись, будет выполнена пересимуляция, если только файл раздела уже не содержит этот набор информации.

Заметьте, что ВСЕ модели, скрипты, глобальные свойства проекта и т.д. включены в sub-netlist (под-спецификацию), так что изменение любого такого объекта вызовет пересимуляцию всех автоматически управляемых секций. ISIS не задаётся вопросом, будет ли отдельная модель, скрипт и т.п. использоваться компонентами в отдельной секции. Если вам нужно обойти это, вы можете использовать ручное управление в режимах RECORD и PLAY.

Заметьте, что объект TAPE в автоматическом режиме будет удалён из схемы, если выполняется интерактивная симуляция.

Режим PLAY

Этот режим даёт вам возможность проиграть именованный файл, который вы прежде записали либо с помощью записывающего устройства, либо добавлением свойства RECORD пробнику. Имя файла данных для проигрывания должно быть введено в поле *Filename* записывающего устройства; вы не можете использовать режим PLAY пока не введёте имя файла.

Когда записывающее устройство в режиме PLAY, схема слева отключена и игнорируется, если только не включает пробники, которые также включены в текущий график.

Не забывайте, что вы можете проигрывать только данные, которые были записаны для тех типов анализа, которые выполняются в данный момент. Попытка выполнения других типов вызовет ошибку.

Режим RECORD

Этот режим приведёт к тому, что данные, присутствующие на входе записывающего устройства, будут записаны в файле с именем в поле *Filename* устройства; вы не можете использовать режим RECORD, пока имя файла не введено.

Другой эффект режима записи в форсировании пересимуляции секции схемы слева от записывающего устройства, независимо от того, будет ли она меняться или нет. Если есть часть схемы справа от устройства записи, которая тестируется, тогда она тоже будет пересимулирована, поскольку она зависит от той части, что слева.

Режим RECORD записывающего устройства наиболее полезен при симуляции, в которой вы хотите записать сигнал, а затем использовать его как входной в дальнейших экспериментах.

СВОЙСТВА УПРАВЛЕНИЯ СИМУЛЯТОРОМ

Обзор

Есть огромное количество параметров, которые сказываются на деталях выполнения симуляции. Это включает такие элементы, как максимальное количество итераций, разрешённых для нахождения рабочей точки, точность, которая определяет момент сходимости, используемый метод интеграции и т.д.

Эти опции, общие для всех типов анализа, могут настраиваться отдельно для каждого графика с помощью его редактирования и выбора кнопки **SPICE Options**.

Свойства точности (Tolerance)

Эта группа параметров определяет, как точно SPICE будет вычислять решение. Чем выше точность, тем, обычно, длительнее время симуляции, а в некоторых обстоятельствах схема может вовсе не симулироваться из-за расходимости, если вы выбрали слишком высокий набор параметров точности.

Наиболее полезное значение здесь — это *Truncation Error Estimation* фактор, или TRTOL в традиционной номенклатуре SPICE. Если вы получили результаты, которые излишне «изрезаны» или страдают отклонениями, вы должны попробовать уменьшить это значение.

Значение минимальной электропроводности, GMIN, определяет утечку обратно включённых полупроводниковых переходов или других теоретических точек бесконечного импеданса. Вырезание этого значения может помочь выполнить сходимость для цепи, которая не смогла симулироваться, хотя это может уменьшить точность симуляции. См. раздел далее, где больше сказано о проблемах сходимости.

Утечка проводимости, GLEAK, похожа на то, что мы описывали, когда говорили о решении для схем с блокированием конденсаторов по постоянному току. Она определяет утечку по постоянному току конденсаторов и может, обычно, оставаться в стороне, пока вы не симулируете что-то похожее на ячейки CMOS памяти или нечто в этом роде.

Свойства Mosfet

SPICE симуляция MOSFET (канальный полевой униполярный МОП-транзистор) основана на допущении, что вы выполняете IC проект и, следовательно, реализуете схему для

масштабируемой геометрии. На практике это означает, что есть некоторое количество параметров, которые определяют предопределённые физические размеры элементов устройства MOSFET. Это значения, определяемые здесь.

Дополнительно были созданы некоторые модели, которые зависят от поведения старых версий SPICE, и это поведение MOSFET можно включать и выключать здесь, если вы используете старые MOSFET модели.

Свойства итерации

Свойства на закладке *Iteration* определяют, как SPICE обходится с цепями, которые плохо сходятся.

Метод интеграции может быть либо *Gear*, либо *Trapezoidal*. Последний поддерживается в основном для обратной совместимости с предыдущими версиями SPICE, хотя метод интеграции *Gear* обычно даёт более точные результаты для заданного количества временных шагов. При интеграции по Gear возможны порядки более второго; это заставляет SPICE использовать больше истории проходов точки времени, чтобы предсказать, что случится в следующей временной точке.

Когда SPICE терпит неудачу при поиске сходящегося решения для рабочей точки, программа делает попытки в двух приближениях: Gmin Stepping и Source Stepping. Количество шагов попыток в каждом методе может быть задано здесь.

Три следующих опции определяют, какое максимальное количество итераций будет использовано для каждой из рабочих точек: шагов в *Transfer* анализе, временных точек в *Transient* анализе. Увеличение этого может помочь в получении результата для сложных или близких к нестабильности схем.

Наконец, две опции дают возможное ускорение симуляций. LTRA уплотняет использование только для схем, использующих линии передач с потерями (LOSSYLINE модель). Идея в том, что близкие к одинаковым значения в данных канала отбрасываются так, что эти точки данных проходятся. Прохождение не изменяющихся элементов в общем оптимизируется, что предохраняет SPICE от пересчёта значений полупроводниковых устройств, чьи узловые напряжения не менялись с последней оценки.

Температурные свойства

Есть два глобальных температурных свойства: TEMP — предопределённая рабочая температура, и TNOM — параметр измеряемой температуры. TEMP определяет актуальную температуру схемы, тогда как TNOM — это значение, при котором зависящие от температуры параметры устройства берутся для выполнения измерения. Более детальное обсуждение моделирования температуры в PROSPICE смотрите в соответствующем разделе.

Свойства цифрового симулятора

TDSCALE, TDSEED, TDLOWER и TDUPPER

Переменная TDSCALE используется для управления масштабированием всех временных свойств, используемых моделями при запуске симуляции, если модель не была явно

отмечена, как не масштабируемая. Переменной TDSCALE может быть присвоено либо значение постоянной с плавающей точкой, либо ключевое слово RANDOM.

Если задано постоянное значение, тогда все временные свойства, определяемые для моделей, используемых при симуляции, умножаются на это значение; значение меньше, чем 1.0 уменьшают временные свойства, а значения больше 1.0 увеличивают их. Например, присвоение переменной:

```
TDSCALE = 1.1
```

даёт эффект удлинения всех временных свойств, используемых при симуляции на 10%. Значение по умолчанию для TDSCALE — это постоянное значение 1.0; это сказывается на всех временных свойствах так, что они не модифицируются.

Если переменной TDSCALE присвоить ключевое слово RANDOM, DSIM будет случайным образом масштабировать каждое временное свойство, умножая его на случайное значение с плавающей точкой. Диапазон значений временного масштабирования, выбираемый симулятором, может быть ограничен присвоением переменным TDLOWER и TDUPPER; этим определяются допустимые наинизшее и наивысшее случайные значения соответственно. По умолчанию значения TDLOWER и TDUPPER 0.9 и 1.1, что ограничивает случайное масштабирование значениями ±10%.

Последовательность генерируемых случайных значений, скажем, псевдослучайная — каждое последующее сгенерированное значение, которое должно быть случайным, фактически определяется предыдущим значением (и сложной формулой). Из этого следует, что любая последовательность случайных чисел определяется начальным значением, и что для заданного начального значения генерируемые последовательности случайных чисел всегда одинаковы. Свойство TDSEED позволяет вам задать начальное значение для генерации случайных значений временного масштабирования и гарантирует, что последовательность, выбранная для одной симуляции, при другом запуске будет всегда той же. Это устраняет проблему ошибок с временами в проекте, обусловленных случайными задержками распространения, которые появляются, но исчезают при последующих запусках симуляции.

Переменная TDSEED может получать только положительные целые значения в диапазоне 1-32767. Предопределённое значение само по себе случайное число (основанное на дате и времени), и этим поддерживается метод генерации случайных чисел от одной симуляции к следующей.

Например, присвоение:

```
TDSCALE = RANDOM
TDLOWER = 2.00
TDUPPER = 3.00
TDSEED = 723
```

даст эффект случайного увеличения всех временных свойств примерно на 200-300% со случайной последовательностью значений масштабирования. Начальное значение 723 приведёт к тому, что та же последовательность псевдослучайных чисел будет генерироваться при каждом запуске симуляции.

INITSEED

Свойство INITSEED используется для начальной случайной инициализации значения генератора, используемого моделями примитивов DSIM, чьё свойство инициализации было присвоено ключевому слову RANDOM.

Как и со свойствами TDSCALE и TDSEED, описанными выше, если последовательность значений, сгенерированная случайным значением инициализации, случайна, последовательность в целом конечна и определена. Свойство INITSEED поддерживает метод выбора, которым последовательность случайных значений используется симулятором DSIM.

Свойство INITSEED может принимать только положительные целые значения в диапазоне 1-32767. Предопределённое значение для INITSEED само по себе случайное число (основанное на дате и времени), и этим поддерживается метод генерации случайных чисел от одной симуляции к следующей.

ТИПЫ МОДЕЛЕЙ СИМУЛЯТОРА

Как сказать, имеет ли компонент модель

PROTEUS приходит с более, чем 8000 элементами библиотеки, из которых около 6000 имеют модели симулирования. Устройства, которые не имеют моделей, очень существенны для использования при разводке печатной платы (PCB design), и идея, что каждый элемент должен иметь модель, абсолютно несостоятельна. Это предполагало бы, между прочим, что мы должны создать модель для 68020 процессора — задача совсем не тривиальная.

Итак, для целей симуляции схем, вам нужно определиться, имеет ли компонент, который вы используете, модель симулирования. Иначе попытка симуляции может окончиться неудачей, если модели нет, и обычно с сообщением похожим на следующее:

```
ERROR [PSM] : No model specified for 'U1'.
```

Ошибка обнаруживается на этапе разбиения (PSM), поскольку до этого может получиться так, что компонент без модели окажется вне части схемы, которая должна симулироваться.

Иногда вы получите:

```
ERROR [U1] :
Value '74F00' of VALUE not found in parameter mapping table.
```

Это означает, что есть файл модели для устройства, но вы должны изменить значение типа элемента, который не моделируется MDF файлом. В примере выше мы должны изменить значение вентиля на 74LS00, который моделируется, для вентиля 74F00, который не моделируется.

Тип модели симулятора (если есть), доступный для компонента, отображается слева вверху *окна предварительного просмотра* в обозревателе устройств библиотеки. Например, если вы открываете обозреватель библиотеки и выбираете устройство 74LS00 в библиотеке 74LS, вы увидите

```
Schematic Model [74NAND.MDF]
```

Дальнейшая информация о разных типах моделей дана в следующих разделах.

Модели примитивов

Значительная часть базовых типов компонентов построены непосредственно в PROSPICE. Эти типы устройств названы Primitives и включают резисторы, конденсаторы, диоды, транзисторы, вентили, счётчики, память и много другое.

Модели примитивов не требуют внешних файлов для симуляции, и они не определяются наличием единственного свойства PRIMITIVE. Дополнительные свойства задаются непосредственно в компоненте и передаются в PROSPICE через netlist. Например, резистор будет иметь:

```
PRIMITIVE=ANALOG,RESISTOR
```

Это определяет элемент, как SPICE Resistor примитив.

Стандартный набор примитивов симулятора можно найти в библиотеках ASIMMDLS и DSIMMDLS. Для этих элементов есть контекстно чувствительная подсказка (help) их свойств, а примеры их использования можно найти в документации VSM SDK.

Схемные модели

Когда симулируются более сложные устройства, общий подход таков — нарисовать схему, которая подражает нужным действиям, используя примитивы симуляции. Эта схема может быть составлена из актуальных внутренних электронных компонентов устройства, но более общим случаем было бы использование идеальных источников тока, напряжения и переключателей для ускорения процесса.

Схемные модели задаются свойством MODFILE, которое по соглашению мы задаём со свойством «только для чтения». Например, операционному усилителю 741 присваивается:

```
MODFILE=OA_BIP
```

Вы могли заметить, что файл модели доступен для нескольких моделей устройств — фокус в использовании *Parameter Mapping Table*.

Схемные модели создаются вычерчиванием схемы в ISIS с последующей компиляцией её *Model Compiler* для производства MDF файла. Дальнейшие детали процесса описаны в документации к VSM SDK.

VSM модели

VSM модели в действительности модели примитивов, которые реализованы во внешних DLL, а не в самом PROSPICE. Они поддерживают функциональность симулируемых устройств, используя язык программирования по вашему выбору, хотя обычно лучше использовать C++.

VSM модель будет иметь и свойство PRIMITIVE (поскольку и ISIS, и PROSPICE трактуют их как примитивы), и свойство MODDLL, которое задаёт имя DLL файла, в котором находится код модели.

Например, модель 8052 имеет:

```
PRIMITIVE=DIGITAL,8052
MODDLL=MCS8051
```

Заметьте, что DLL модели может реализовать более одного типа примитива — MCS8051.DLL реализует несколько вариантов 8051.

VSM модели могут также реализовать функциональность, которая относится к анимации, так что электрические и графические аспекты операций компонента могут комбинироваться весьма спокойным образом. Модель LCD дисплея прекрасное тому подтверждение.

Создание VSM моделей вращается вокруг нескольких классов C++ Interface (похожих на COM). Всё это задокументировано в руководстве к VSM SDK.

SPICE модели

Поскольку PROSPICE основан на Berkeley SPICE3F5, он непосредственно совместим со стандартными SPICE моделями, и множество компонентов в библиотеках PROTEUS моделируются, используя SPICE файлы, полученные от производителей компонентов. SPICE модели могут быть заданы либо блоком SUBCKT, либо набором параметров в записи MODEL. SUBCKT модели будут иметь присваивание свойств, как следующие:

```
PRIMITIVE=ANALOG,SUBCKT
SPICEMODEL=CA3140
```

где примитив модели SPICE для транзистора может получить следующие свойства:

```
PRIMITIVE=ANALOG,NPN
SPICEMODEL=BC108
```

Собственно модель может хранится либо в ASCII (текстовом) файле, либо в библиотеке *SPICE Model*. Имена этих файлов задаются либо свойством SPICEFILE, либо SPICELIB.

Различные детали того, как использовать SPICE модели от производителей, описаны в разделе «Использование SPICE моделей».

НЕПОЛАДКИ

ЖУРНАЛ (LOG) СИМУЛЯЦИИ

При каждой симуляции создаётся запись в файле журнала (Simulation Log). Файл отчёта содержит всю информацию, предупреждения и сообщения об ошибках и самого симулятора, и индивидуальных моделей. Когда сообщение создаётся моделью, оно имеет префикс ссылки модели компонента в квадратных скобках, так что вы можете увидеть:

```
[U1] Loaded 26 files from PROGRAM.HEX
```

В процессе интерактивной симуляции содержимое этого файла может быть отображено во всплывающем окне из раздела *Debug* основного меню, тогда как для симуляции, основанной на графиках, файл может быть просмотрен, если указать на график и нажать **CTRL+V**.

В случае, когда симуляция прекращается полностью, файл отчёта будет отображаться автоматически, чтобы вы могли видеть причину возникающей проблемы немедленно.

ОШИБКИ NETLIST

Эти ошибки обнаруживаются, как результат проблем попыток ISIS создать netlist схемы — вы также столкнётесь с этим, если попытаетесь экспортировать схему в ARES для разводки печатной платы. Общим будет:

- Наличие двух компонентов с одинаковыми именами или компонентов без имени, например, двух резисторов с этикеткой R?.
- Плохо сформированные файлы скриптов, как MAP ON таблицы и т.д. Обратитесь к определениям синтаксиса в руководстве ISIS, если вы не можете выявить проблему немедленно.

ОШИБКИ КОМПОНОВКИ (LINKING)

Компоновка модели — это процесс в котором ISIS вызывает файлы MDF для компонентов, которые моделируются эквивалентными схемами. Безоговорочно, наиболее общей проблемой служит ситуация, когда заданный файл модели отсутствует. Файл модели должен быть в текущей директории или в *Module Path*, как указано в диалоге *Set Paths*.

Другие общие ошибки компоновки включают:

- Значение не найдено в таблице карты параметров. Это означает, что тип элемента — скажем, CA3140, не включён в список в таблице карты заданного файла модели. Файлы модели, такие как OA_MOS.MDF разработаны для моделей нескольких разных компонентов, использующих ту же схему, но с разными значениями. Эта ошибка означает, что заданный файл модели не имеет параметров для типа устройства, которое вы используете — вы можете открыть MDF файл с помощью текстового редактора, чтобы увидеть, какие устройства моделируются.
- Неразрешённый вывод модуля. Это, скорее, вызовет предупреждение, а не сообщение об ошибке, и означает, что корпус родительского компонента имеет вывод, который отсутствует в модели. Часто это не препятствие — например, большинство моделей

операционных усилителей не моделируют отвод от нулевого вывода, но это может быть ошибкой, и, если новая модель не работает, в этом общем случае появляется предупреждение «No DC path to Ground».

ОШИБКИ РАЗБИЕНИЯ

Разбиение (Partitioning) — это механизм, с помощью которого ISIS решает, какая часть(и) схемы нуждаются в симулировании. Проблемы, которые могут возникнуть, это:

- Обнаруживается циклическая зависимость. Это означает, что расположение записывающих устройств (tapes) таково, что секции за и перед записывающим устройством взаимно зависимы. При условии, что вы корректно задали все *Isolate Before* и *Isolate After* флажки на пробниках и генераторах, ваше простейшее действие в этом случае будет, возможно, таким — удалить все объекты записывающих устройств и симулировать всю схему за один заход.
- Не задана модель — это означает, что обнаружен компонент, который не имеет свойств PRIMITIVE, MODFILE, и появился в части схемы, которая нуждается в симуляции. Если устройство неподходящее (например, коннектор), тогда вы должны задать PRIMTIVE=NULL, иначе модель вам нужна!

В руководстве есть раздел, где больше сказано о типах моделей симулятора. Обратитесь к нему.

ОШИБКИ СИМУЛЯЦИИ

Ошибки симуляции генерируются PROSPICE, а не ISIS, и поэтому обнаруживаются после того, как файл netlist был успешно сгенерирован. Общие проблемы, которые при этом обнаруживаются, включают:

- Тип устройства не распознаётся. Это означает, что вы задали тип примитива, который не поддерживается, или что файл модели и уже использовался.
- Нет пути постоянного тока к земле. Это обсуждается в разделе «Шины питания и земли».
- Не находится пробник — вы пытаетесь обратиться к пробнику или генератору напряжения, который не существует. Помните, что вы должны использовать объект IPROBE из ASIMMDLS.LIB — вы не можете ссылаться на гаджет токового пробника.
- Не открывается исходный SPICE файл. Исходный файл, заданный свойством SPICEMODEL, не может быть локализован. Он должен быть в текущей директории или задан в *Module Path* из диалоговой формы *Set Paths*.
- Не находится библиотечная модель. SUBCKT или MODEL, которые вы задали, не существуют в указанной библиотеке или на диске.
- Не находится DLL модель. Заданная VSM модель DLL не может быть локализована. Она должна быть в текущей директории или задана в *Module Path* из диалоговой формы *Set Paths*.

ПРОБЛЕМЫ СХОДИМОСТИ

Этот последний набор ошибок относится к тому, что случается, если SPICE сам терпит неудачу при симуляции. Есть три основных сообщения об ошибке, которые говорят об этом:

- Сингулярная матрица. Похоже, неизвестных больше, чем уравнений, чаще всего это относится к схемам, которые «недорисованы», или в которых некоторые начальные условия нуждаются в том, чтобы давались в порядке, определяющем стартовое состояние.

  Эта ошибка часто предваряется предупреждением «No DC path to ground», и вам нужно проверить соединения вокруг выводов, обозначенных в списке после этого предупреждения. Если часть вашей схемы не заземлена, симулятор не может определиться с напряжением относительно земли — это столь же просто, как это.

- Слишком много итераций без схождения. Это означает, что решение схемы нестабильно. Цепи с примитивами VSWITCH или CSWITCH могут легко создавать такие условия, но любые цепи, передаточные функции которых разрывны, могут создавать серьёзные проблемы для SPICE.

- Временной шаг (Timestep) слишком мал. Это означает, что схема переключается таким образом, что прохождение по времени даже при очень маленьких значениях (обычно 1E-18 сек) все ещё не производит сколь-нибудь заметных малых изменений напряжения в цепи.

  Часто это связано с плохо разработанной моделью, или с отсутствием задания подходящих параметров для моделей диодов или транзисторов. На практике, если ёмкость перехода не выбрана правильно, такое устройство будет показывать нулевое время переключения, которое может непосредственно вызвать это сообщение об ошибке.

Большинство ошибок схождения происходят из-за плохо нарисованной схемы или плохих моделей — время от времени мы получаем схемы, присланные по причине «не симулируется», но только для того, чтобы обнаружилось что-то, что не было соединено. Пожалуйста, проверьте отчёт о симуляции на ошибки, перепроверьте вашу схему, прежде чем делать заключение о неработоспособности PROSPICE. *В тех случаях, когда используются SPICE или VSM модели других производителей, мы не можем тратить время на их отладку, пока не сможем предоставить простую схему, демонстрирующую проблему.*

Остерегайтесь также использования SPICE моделей других производителей, которые не поддерживают стандарт SPICE 2 или SPICE 3. Модели разработанные для PSPICE™ могут включать и элементы, и синтаксические конструкции, которые не являются стандартом SPICE.

Осцилляторы могут стать причиной особых проблем, поскольку начальное решение для рабочей точки потерпит неудачу. В конце концов, осцилляторы не имеют устойчивого состояния! Используйте IC, NS или OFF свойства для определения начального состояния, как это обсуждалось в разделе «Начальные условия».

Если проблема в действительности в числовом расхождении, вы можете попробовать следующие тактики:

- Увеличьте значение GMIN. Это сопротивление утечки для обратно смещённого перехода полупроводника, и наименьшее значение сделает цепь более похожей на цепочку резисторов (которые всегда решаются). Но это ухудшает точность. По умолчанию, это 1E-12; значения больше 1E-9 дадут довольно бессмысленные результаты.

  Заметьте, что в любом случае SPICE3F5 попробует то, что называется продвижением GMIN, если вначале схема не сходится. Это означает, что большое значение GMIN используется для нахождения начального решения, а значение затем постепенно возвращается к его оригинальному значению, чтобы сохранить точность.

- Увеличьте значение ABSTOL и/или RELTOL. Эти значения управляют точностью, которая требуется для симуляции, чтобы она считалась сходящейся. Однако, чем больше вы делаете допуск, тем меньше точные получаете результаты.

- Если схема использует операционные усилители, попробуйте задать MODFILE=OA_IDEAL вместо специфического типа устройства — эту модель намного легче симулировать.

- Уменьшите значение TRTOL. Это заставит SPICE использовать меньшие временные шаги, так что уменьшит вероятность «потерять» сходящееся решение, но это может увеличить время симуляции. Это сработает только тогда, когда симуляция отказывает в части анализа переходного процесса.

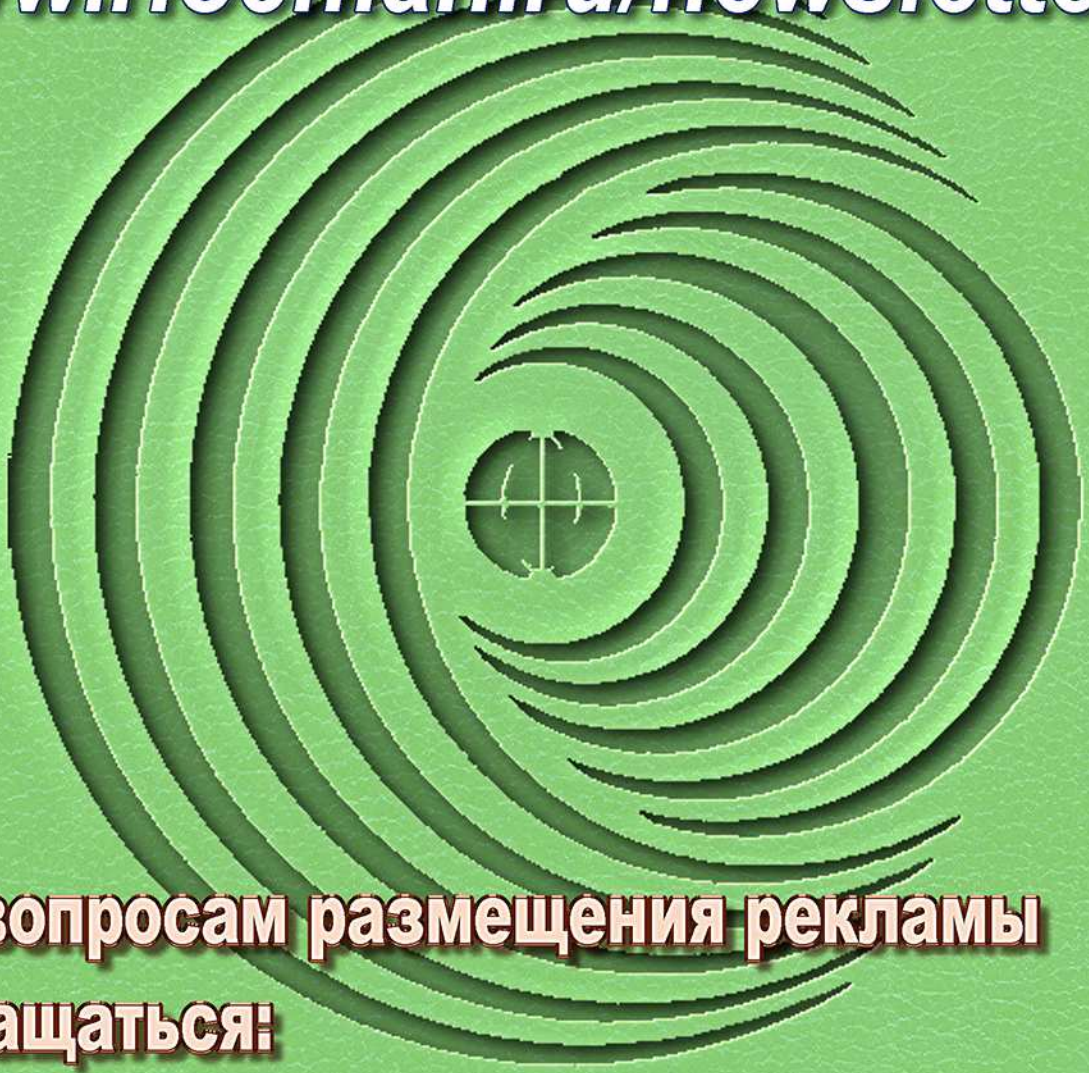
  Вы также должны попробовать уменьшить TRTOL, если нарисованные кривые выглядят «изрезанными» или содержат математический шум. Это часто проявляется в виде колебаний значения после быстрой смены уровня.

Уведомление о новых
выпусках "Радиоежегодника"
на основной ленте новостей

www.rlocman.ru

и выпусках почтовой рассылки

www.rlocman.ru/newsletter



По вопросам размещения рекламы
обращаться:

rlocman@rlocman.ru

radioyearbook@gmail.com