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 (экс-участник и представитель Labcenter), Um, vgololobov, Worker и многих других.
Если кто-то считает, что его обошли вниманием в данном списке, а он претендует на авторство в ЧАВО, просьба сообщить мне при обсуждении темы, обязуюсь исправить в окончательном варианте, в ходе доработки ЧАВО список будет пополняться.

1.Краткие общие сведения о программном продукте PROTEUS.

1.1.Что такое Протеус.

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 трассировщика в последнем может использоваться вспомогательная программа ELECTRA Autorouter.

1.2.Сайт автора.

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

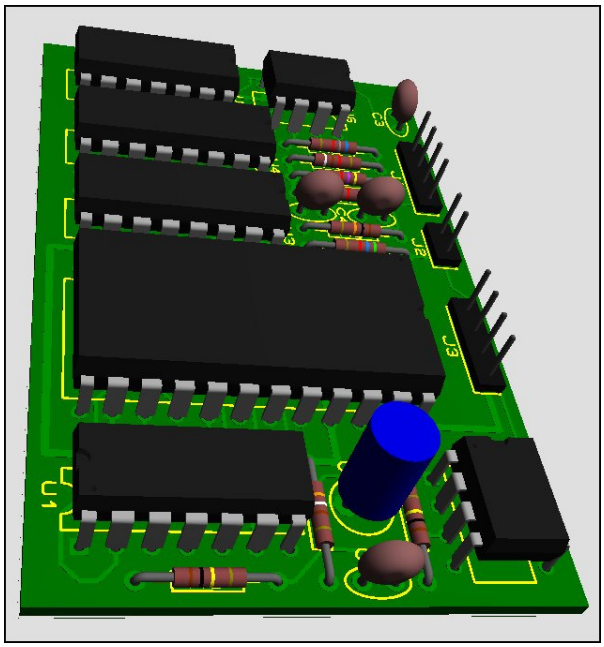
1.3. В чем отличие от других подобных программ.

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 МК: 8051, PIC (Microchip), AVR (Atmel), и др. Протеус имеет обширные библиотеки компонентов в том числе и периферийных устройств: индикации, температурных датчиков, интерактивных элементов ввода-вывода: кнопок, переключателей, виртуальных портов и виртуальных измерительных приборов, интерактивных графиков, которые не всегда присутствуют в других подобных программах.
К программному пакету прилагается большое количество примеров расположенных в папке **Samples** . В версиях 6 существовал HELP по Samples. К сожалению, в седьмых версиях Лабцентр посчитал это излишеством. Для англоязычной аудитории прилагаю HELP по примерам от версии 6.95 в оригинале. Для тех кто читает, но не переводит привожу краткий обзор примеров с указанием наиболее интересных.
Папка **Schematic&PCBLayout** - содержит примеры выполнения схем и печатных плат, т.е. почти каждый пример проекта в **ISIS** (.DSN) имеет одноименный пример проекта в **ARES** (.LYT). Наиболее интересны в этой папке:
Features.DSN - пример возможностей схемотехники в **ISIS** Пример не симулируется, а просто показывает варианты построения схем в **ISIS** . Обратите внимание - стереоусилитель в правом верхнем углу содержит дочерние листы модулей каналов.
PICDEM2.DSN - пример построения схемы с микропроцессором. Пример многолистовой, т.е. щелкнув правой лапкой мышки по свободному месту на листе можно выбрать один из трех листов во всплывающем меню внизу. Обратите внимание на грамотное построение - все интерактивные элементы (кнопки, индикация и т.п.) расположены на главном первом листе, сам микропроцессор на втором, блок питания на последнем.
Наиболее впечатляет окружающих - 3D визуализация в **ARES** (см. картинку). Для этого открыв любой проект из этой папки в **ARES** достаточно выбрать пункт **3D Visualization** в закладке **Output** верхнего меню. Далее удерживая нажатую левую кнопку мышки вы можете

вращать плату, шаркая мышкой по подстилке или удалять приближать колесом.
Папка **Interactive Simulation** - содержит примеры поддерживающие симуляцию в реальном времени. Для начинающих наиболее интересна подпапка **Animated Circuits** с анимированными простыми схемами. А в ней все дизайны **OSC** - генераторы и **TTLClock.DSN** - действующие часы на TTL логике. Остальные подпапки содержат примеры применения виртуальных инструментов и портов в Протеусе.
Graph Based Simulation - содержит примеры исследования схем с помощью графиков или графов. Им посвящен отдельный раздел ЧАВО на стр.4.
Tutorials - примеры создания собственных модулей и компонентов. Этому вопросу тоже посвящен раздел ЧАВО начиная со стр. 5.
Ну и ряд папок начинающихся с **VSM...** и далее характерный тип микроконтроллеров, примеры для которых помещены в одноименной папке. Наиболее полно представлены МК PIC фирмы MicroChip, поскольку Лабцентер плодотворно и давно сотрудничает с этим производителем. **Proteus** включен даже в качестве отладочного инструмента в MPLAB IDE и в папке **VSM MPLAB Viewer** имеются примеры совместного использования этих двух программных продуктов. В папках с примерами по микроконтроллерам вы найдете множество примеров применения, как правило в названии подпапки с примером можно понять что внутри: Clock - часы, Calculator - калькулятор, Chess - шахматы (действующие, можно поиграться!!!).
Кроме того, примеры проектов содержатся в одноименной ветке здесь на форуме **Kazus** .

1.4. Версии программного обеспечения и отличия в них.

Протеус активно развивается на протяжении 12 лет, начиная с ранних версий 4.xx и кончая последней на сегодняшний день версией 7.4.SP0. Из доступных в сети Интернет для «неофициального» использования последней является версия 7.2.SP6 Professional. Если не рассматривать ранние версии 4, 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 начинающим тяжело освоить. Это напоминает езду на британских автомобилях с правым рулем и при левостороннем движении.



1.5. Системные требования.
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 7.2.SP2 в Ubuntu 7.10 под Wine). С «пиратскими» версиями данных систем возможны проблемы устойчивой работы Протеуса.
1.6. 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://oproteus.narod.ru/PROTEUS начинающим на примерах – 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, сахара.ru и др.

2.Установка и первый запуск Proteus.
2.1.Где взять инсталляционный пакет Протеус.
На оффсайте Labcenter Electronics доступна последняя демо-версия на данный момент v.7.4.SP0. Она имеет ограничения: отсутствует опция сохранения проекта. Учитывая географическую удаленность «туманного Альбиона» и приличный размер инсталлятора – более 50 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. Какую версию ставить.
Это зависит от ваших потребностей и способностей. Чем старше номер версии, тем большее количество компонентов (а главное микроконтроллеров в ней присутствует). Однако, Протеус, как и многие другие программы-симуляторы не лишен наличия "багов". От версии к версии старые баги исправляются - новые плодятся. По мнению многих участников данного форума наиболее стабильные версии 6.7 и 7.2.SP2. Кроме того, для некоторых имеет значение размер закачки: версия 6.7.SP3 весит 21,4 Мбайт, 7.2.SP6 более 45 Мбайт. Если вы не собираетесь использовать последние разработки микропроцессоров PIC, не используете AVR, которые наболее широко представлены в версиях, начиная с 7.2, то можете выбирать любую, или установить несколько версий (об этом ниже). И еще два замечания:
Для создания собственных активных моделей с версиями до 6.3 прилагалась библиотека VSM SDK (папка INCLUDE), которая в более поздних версиях по прихоти разработчика отсутствует в инсталляторе.
В Интернет бытует мнение, что Протеус - это симулятор микроконтроллерных схем. После прочтения («с Протеусом в руках») вышеупомянутой книги В. Гололобова я готов «посыпать голову пеплом» и признать, что и я долгое был подвержен этому заблуждению. Аналоговая симуляция, пусть и не в RealTime, но работает прекрасно.

2.3.Установил – не запускается, отсутствует лицензия.
При попытке запуска ISIS или ARES появляется сообщение:
Cannot find a valid licence key for ISIS (ARES) on this computer.
Если вы устанавливаете версию Professional, уже в процессе установки Proteus запросит путь к файлу лицензии licence.lxk. При установке можно указать, что лицензия находится на сервере, однако потом ее все-равно необходимо установить. Для этого запускаем менеджер лицензий: ПУСК=>Все программы=>Proteus x Professional=>Licence Manager, нажимаем Browse For Key File для указания пути к файлу вручную, либо Find All Key File для автоматического поиска файлов лицензий на компьютере. Все имеющиеся файлы отобразятся в левом окне. Для установки нужной лицензии выбираем ее мышкой и жмем Install. Когда лицензия отобразится в правом окне, менеджер можно закрыть.

2.4. Установил – не запускается симуляция даже для прилагаемых Examples.
Основная ошибка всех начинающих работать с Протеусом – установка и запуск программы от имени пользователя компьютера, набранного кириллицей. Протеус не дружит с русскими буквами в путях к файлам. Поэтому, если у текущего пользователя имя «Вася», «Мария Ивановна» и т.п. вы рискуете при запуске симуляции получить следующее красное сообщение в окне log:
Cannot open 'C:\DOCUME~1\ТЕКПОЛЬЗ\Local Setting\Temp\LISAxxx.SDF'
Simulation FAILED due to fatal simulator errors
где вместо **ТЕКПОЛЬЗ** закорючки. Это объясняется тем, что при запуске симуляции Протеус пытается создать в данной папке свой файл **LISAxxx.SDF , но не понимает русские буквы в абсолютном пути.**
Есть два пути решения этой проблемы:
1) Изменить имя пользователя WINDOWS 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.

2.5.Симуляция запускается, но через несколько секунд (минут) программа закрывается. Симуляция работает только с некоторыми типами моделей.
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 применялось "хирургическое вмешательство".

2.6. Две (или несколько) версий Proteus на одном компьютере.
Совет от Kabron:
Разные версии устанавливаются последовательно. При этом для каждой версии нужно выполнить следующие действия:

a) установить программу в отдельную папку (например: Proteus Pro v6_7, Proteus Pro v7_2_SP2 и т.п.);
b) установить лицензию, настроить пути во вкладке программы System=>Set Paths…;
c) В папке \BIN каждой установленной версии создать пустую папку, например Regs
d) запустить любой редактор реестра (например штатный Пуск=>Выполнить=>regedit) и выполнить экспорт следующих веток реестра:
HKEY_CURRENT_USER\Software\Labcenter Electronics
HKEY_LOCAL_MACHINE\SOFTWARE\Labcenter Electronics
в соответствующую папку Regs. Пример:
HKCU_Labcenter_Electronics.reg
HKLM_Labcenter_Electronics.reg
После чего указанные ветки (перед установкой следующей версии) удалить.
Создать командный (текстовый в блокноте Notepad) файл следующего содержания:

Код:

```
reg import .\Regs\HKLM_Labcenter_Electronics.reg
reg import .\Regs\HKCU_Labcenter_Electronics.reg
isis.exe
reg export "HKLM\Software\Labcenter
Electronics".\Regs\HKLM_Labcenter_Electronics.reg
reg export "HKCU\Software\Labcenter Electronics"
.\Regs\HKU_Labcenter_Electronics.reg
```

сохранить его с расширением .cmd вместо .txt например Start.cmd и положить в каждую папку \BIN
e) для каждой версии создать на рабочем столе ярлык к этому файлу с оригинальным названием, например для Proteus Pro v6_7\BIN\ Start.cmd назовем Протеус_6 и т.д. Теперь запуск соответствующей версии осуществляем через нужный ярлык, при этом не надо закрывать окно командной строки до окончания работы с ISIS, чтобы сохранились изменения в реестре (см. последние две строки командного файла) после выхода из программы.

Немного упустил один нюанс и получил вопрос в личку. Исправляюсь:
Дело в том, что Протеусы из одной серии 6-й (или 7-й) при установке друг за другом выполняют апдэйт, т.е. обнаруживают предыдущую версию (например 7.4 обнаружит 7.2, но не 6.x) и выполняют обновление в этой папке. Поэтому, если вы установили 7.2.SP6 и хотите иметь отдельно 7.4, то после выполнения настроек и чистки реестра (как указано выше) переместите папку Proteus_7_2 куда-нибудь подальше из Program Files (лучше вообще на другой диск, а после установки новой версии в папку с другим именем (например Протеус_7_4), верните ту папку на место. Для меня это было очевидно, поэтому как-то упустил упомнить об этом раньше. Спасибо Zandy за своевременный вопрос.

Мой личный вариант нескольких версий:
если не использовать последние две строчки командного файла, а предыдущую строчку записать как start isis.exe, то окно DOS после старта ISIS закроется само. Как правило, при уже настроенном ISIS редко вносятся изменения, отражаемые в реестре. Поэтому если корректно выходить из программы с сохранением последних изменений в проекте, то нет необходимости при завершении работы лишний раз сохранять ветки реестра для данной

версии. Единственное, чем это грозит – не сохранится путь к последнему открытому проекту. Текст файла .cmd в этом случае выглядит так:

```
reg import .\Regs\HKLM_Labcenter_Electronics.reg
reg import .\Regs\HKCU_Labcenter_Electronics.reg
start isis.exe
```

И последнее замечание по данному вопросу. Рассмотренный метод применим и к ARES, если isis.exe заменить на ares.exe. Для внесения изменений в реестр Windows XP текущий пользователь должен обладать правами Администратора системы.

2.7. Русификация програмного интерфейса Протеус.

Русифицировать или нет программу – дело вкуса пользователя. В сети Интернет, да и здесь на сайте Kazus по ссылке: http://kazus.ru/programs/viewdownload/kz_0/cid_190.html можно найти варианты как бесплатных, так и платных вариантов русификации интерфейса программы. Однако, как уже подчеркивалось, программа имеет многоступенчатую защиту и, если после русификации некоторые функции будут работать некорректно, - причину в первую очередь ищите в стремлении «облегчить себе жизнь». Протеус имеет достаточно обширную справочную систему на английском языке, но с русифицированным интерфейсом Вам придется заниматься «обратным переводом», чтобы понять: для чего предназначена та или иная кнопка и функция в программе. Не проще ли выучить азы английского и технические термины на этом языке, тем более, что большинство документации по импортным компонентам в Интернет представлено только в английском варианте. Как показала практика, последнее утверждение уже проявило себя, особенно при создании собственных моделей.

3. Proteus ISIS - вопросы по симуляции (ошибки и электропитание).

3.1. Нарисовал самую простую схему, но симуляция не работает. В чем ошибка?

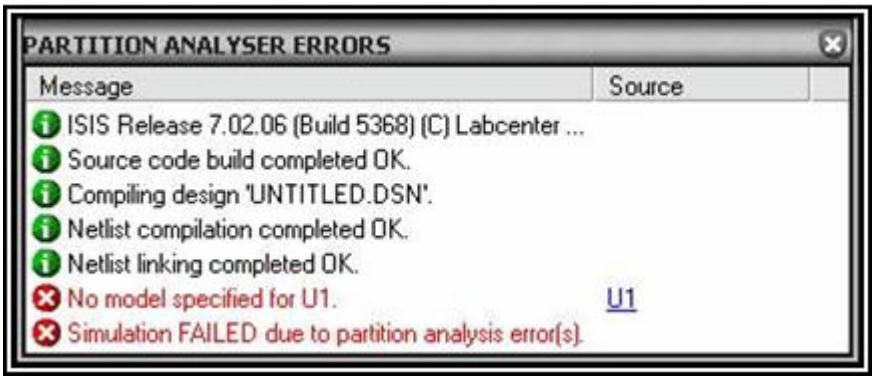
Самый частый вопрос новичков в Протеусе.
Обязательно обращайте внимание на сообщения в Simulation Log рядом с кнопкой СТОП симуляции. Окно Simulation Log доступно и после останова при щелчке по нему левой кнопкой мыши. Здесь действует принцип светофора: зеленые – сообщения об успешной компиляции, желтые – предупреждения, красные – ошибка симуляции. Англоязычные пользователи могут непосредственно узнать в чем их ошибка. Для тех, кто «читаю и перевожу со словарем» рассмотрим наиболее часто вылетающие сообщения.

Ошибка симуляции:

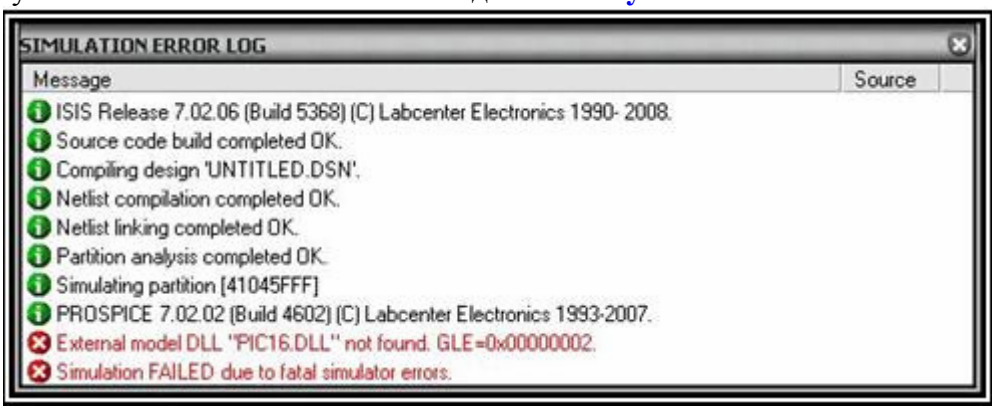
No model specified for (обозначение элемента)

Simulation FAILED due partition analysis error(s)

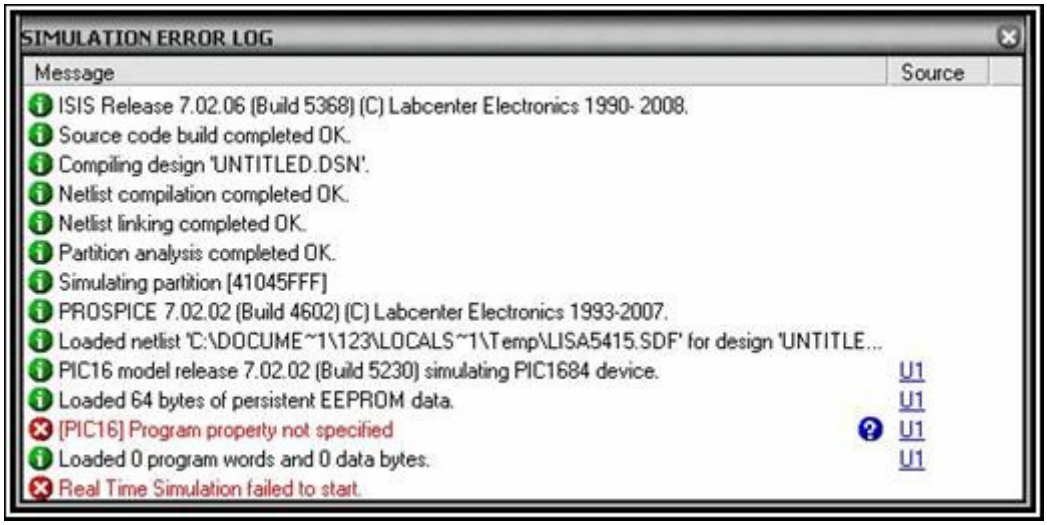
При выборе модели элемента из библиотеки обратите внимание на верхнее правое окно Schematic Preview, где над схемным изображением элемента появляется его характеристика. Если там стоит **No Simulator Model**, не пытайтесь добавлять его в схему, которую будете отлаживать в ISIS. Эти элементы (кстати к ним относятся почти все разъемы) предназначены для создания принципиальной схемы с передачей в ARES и последующей разработки печатной платы устройства. Все остальные модели, а это: Schematic Model, VSM DLL Model, SPICE Model, и различные Primitive предназначены для симуляции. Более подробно модели будут рассмотрены при рассмотрении вопроса создания собственных моделей.



Ошибка симуляции:
(External) Model DLL (обозначение элемента) not found
Simulation FAILED due partition analysis error(s)
Наиболее часто встречается при попытке запустить симуляцию проекта созданного в более поздней версии Протеуса. Вы скачали чей-то готовый проект, но в вашей версии отсутствует библиотека для этого элемента, либо автор применил свой собственный (External) элемент и «забыл» приложить к нему библиотеку. Иными словами в свойствах элемента прописана библиотека DLL, но она отсутствует и в папке с проектом и в папке **MODELS** Протеуса, в которой хранятся библиотеки симулируемых элементов. Может быть и так, что у вас не прописан путь к библиотекам Proteus в закладке меню **System=>Set Paths…**



Ошибка симуляции:
PROGRAM property not specified
Real Time Simulation failed to start
При симуляции схемы, содержащей микроконтроллер в его свойствах не указан (или неправильно указан) файл микропрограммы. В качестве такого файла могут использоваться бинарные с расширением **.hex**, а также файлы, создаваемые некоторыми компиляторами с языков высокого уровня с расширением **.cof, .elf.** Последний необходимо набрать непосредственно с клавиатуры, т.к. в окне выбора **Program File** в свойствах микроконтроллера он явно предлагаться не будет.



Предупреждение симуляции:
Simulation is not running in real time due to excessive CPU load
При этом обычно внизу фигурирует **CPU load 100%** . Симуляция не может выполнятся в режиме реального времени из-за стопроцентной загрузки процессора компьютера. Обычно проявляется на схемах перегруженных аналоговыми компонентами. Типичная ошибка у начинающих - слепое копирование принципиальной схемы устройства и попытка "оживить" ее в симуляторе в реальном времени. Подумайте: нужен ли, например, блокировочный конденсатор (типичный аналоговый элемент) на шине питания вашей схемы, если вы собираетесь ее отлаживать в симуляторе ISIS, который не будет иммитировать импульсные помехи по питанию. Если вам необходимо передавать схему в ARES, он необходим для установки на печатную плату, но его можно добавить и после отладки схемы. Еще хуже обстоит дело с гасящими помехи RC цепями и индуктивными элементами. Чтобы Протеус смог иммитировать их работу в реальном времени необходим сверхмощный компьютер.
А вот что по этому поводу гласит **Proteus VSM Help: ADVANCED TOPICS => How to make interactive simulations run faster** (Как ускорить интерактивную симуляцию).
Использование цифровых (Digital) моделей резисторов и диодов:
"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 достигнет 100%.
Для многих компонентов интуитивно можно определить: какое моделирование требуется - аналоговое или цифровое. Например, почти вся ТТЛ-логика и часть КМОП требуют цифрового моделирования, в то время, как аналоговые ИС: операционные усилители, компараторы требуют аналогового. Все компоненты представленные (имеется ввиду в библиотеках) стандартными SPICE моделями требуют только аналоговой симуляции
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"
Я нарочно привел почти дословный перевод раздела, чтобы показать, как полезно почитать входящий в стандартную поставку HELP. Перевести резистор или диод в режим Digital можно в свойствах этого компонента в окне Model Type.
Многие из последних проектов, которые авторы выкладывают на форуме с просьбой о

помощи, грешат именно этой ошибкой, поэтому я так подробно остановился на ней. Некоторые начинающие умудряются даже сетевой трансформатор с диодным мостом дорисовать в проект и пытаются запустить их симуляцию. Подумайте: а нужно ли оно там, если вы не проектируете конкретно блок питания? Добавлю сюда еще о кварцевом резонаторе - тот **CRYSTAL** , который в библиотеках Протеуса - это фильтр. Кто не верит - откройте одноименный проект в папке Протеуса **\SAMPLES\Graph Based Simulation**. Щелкните по кварцу правой лапкой мышки и выберите **Goto Child Sheet** (Перейти на дочерний лист) - там представлена схематичная модель этого кристалла. Поэтому, если вы вклеили в схему элементы для PSB (печатной платы в ARES) заранее, в свойствах этих элементов отметьте галочкой пункт **Exclude from Simulation** (Исключить из симуляции) - это избавит Вас от лишней головной боли.

Предупреждение симуляции:
[SPICE] TRAN: Timestep to small; timestep=(значение): trouble with node #значение#branch
Ключевым в данном сообщении является фраза: **Timestep to small** . Обычно этому предупреждению предшествуют еще несколько предупреждений Spice о DELMIN и GMIN. Вот наиболее подробный и толковый разбор этой проблемы от retro55:
Протеус ругается, что шаг по времени достиг минимальной величины. Если Протеус не может найти решение, то он начинает его искать все более мелкими шагами по времени, пока не достигнет минимальной разрешенной тобой величины TMIN. Подобные проблемы сходимости решаются следующим образом. Заходишь System->Set animation option->Spice option->Transient-TMIN=1E-25, NUMSTEPS=500. Затем на вкладке количества итераций для поиска решения Iteration начиная с третьего параметра увеличиваешь допустимое количество итераций в 10 раз, то есть везде дописываешь нолики.SRCSTEPS=1200, GMINSTEPS=1200 и так далее. Если решение не будет сходиться, ты разрешаешь Протеусу искать его большее количество шагов. Далее на вкладке tolerance ослабляешь требование к точности вычисления.
ABSTOL =1e-10, VNTOL=1e-5 CHGTOL=1E-10 GMIN=1E-10 RSHUNT=1e10 TRANSGMIN=1E-8 и так далее. Как они пишут очень важный параметр GMIN, если проводимость в какой либо цепи меньше этой величины, то такая цепь считается разорванной.
Дополнение от Worker:
Лечится увеличением до GMIN=1e-11, у меня большинство проблем по сходимости решалось именно таким образом! Нашел это решение в хэлпе по Протеусу.
И опять отошлю пытливых к **Proteus VSM Help** . Раздел **TROUBLESHOOTING** подробно описывает данную проблему. А попасть туда можно напрямую из окна Simulation Log если щелкнуть мышкой по знаку вопроса в сообщении.

Предупреждение симуляции:
Logic contention(s) detected on net #(номер цепи)
Типичная ошибка для начинающих. В цифровой цепи с номером **#(номер цепи)** обнаружен

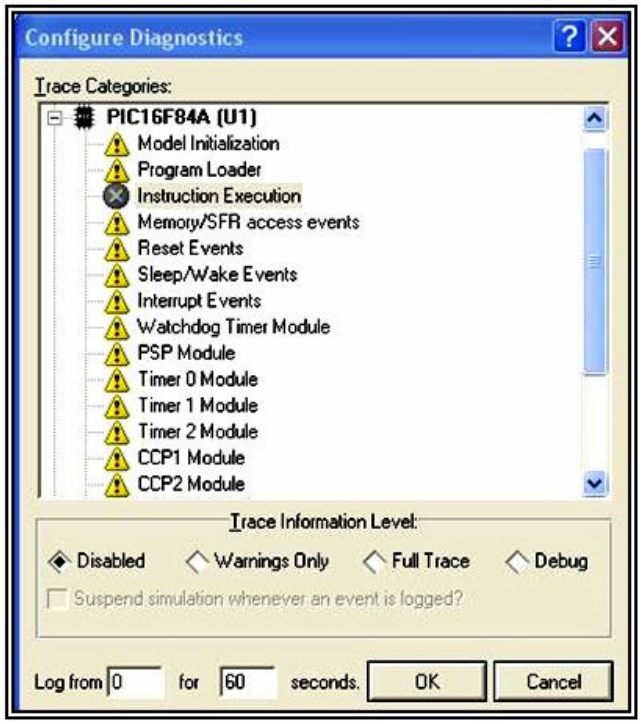
конфликт сигналов. Если щелкнуть мышкой по номеру откроется список цепи в которой обнаружен конфликт. Обычно, когда два или более выхода с различными состояниями 0 и 1 объединены в одну цепь в текущий момент времени. Часто вылетает, когда используется интерактивная кнопка, замыкающая вывод на землю или шину питания без дополнительного резистора, т.е. при симуляции вы ее замкнули, а в этот момент на выводе элемента появился противоположный сигнал. Еще одна причина в несогласованности по времени, например, двух микропроцессоров связанных между собой.

Предупреждение симуляции:

Иногда однотипные желтые предупреждения симулятора идут сплошным потоком и забивают окно **Simulation Log** , мешая творческому процессу отладки, хотя симуляция при этом идет успешно. В качестве примера приведу популярный компилятор **CCS PICC** . Для 14-разрядных МП (например: архипопулярного у начинающих PIC16F84A) **CCS PCM 14 bit** использует при компиляции архаичную команду TRIS, которая жутко не нравится Протеусу и он начинает каждый раз при изменении порта B выводит сообщения типа:

TRISB instruction deprecated for PIC1684

(TRISB нежелательная инструкция для PIC1684), хотя программа выполняется правильно. Если Вы уверены в своей правоте, то можно отключить это сообщение, равно, как и другие сообщения диагностики, войдя во вкладку: **Debug => (жучок) Configure Diagnostics...** для выбранного элемента, установив флажок **Disabled** для данного типа сообщений.



3.2. Вопросы касающиеся электропитания симулируемой схемы.

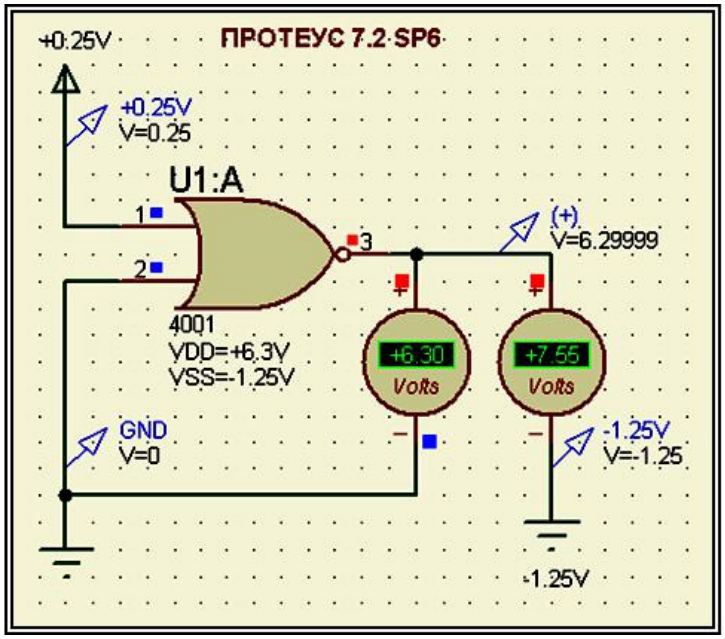
Начинающим необходимо сразу усвоить, что по умолчанию в **ISIS** при создании **New Design** автоматически конфигурируются три шины питания: **VCC/VDD** с потенциалом +5V; **GND** – 0V и **VEE** – -5V. Второй важный момент: для всех микросхем, имеющих скрытые (**HIDE**) выводы питания, а это все цифровые микросхемы, микроконтроллеры и ряд других при запуске симуляции их выводы питания автоматически подключены к упомянутым выше. Если вам необходимо иметь другие значения напряжений VCC/VDD GND или VEE, необходимо изменить их значения во вкладке **Design => Configure Power Rails…**

Если необходимы дополнительные шины питания, или земляная, отдельная от существующей по умолчанию поступаем следующим образом:

На листе проекта добавляем необходимый элемент POWER или GROUND, но в свойствах (**Edit Properties**) указываем конкретный потенциал обязательно таким образом: **+12** или **-0.5V** или **+0** т.е. начинается со знака и заканчивается цифрой или символом V. Тогда он будет автоматически добавлен в **Configure Power Rails…**Если вы дадите буквенное название (например VCC2), необходимо затем зайти в **Configure Power Rails…** и уже там создать (кнопкой New) шину питания с данным названием и необходимым потенциалом и добавить (кнопка **Add**) к ней созданную вами на листе цепь (она будет отображаться в окне **Unconnected Power Nets**).

Если необходимо питать конкретную микросхему со скрытыми выводами питания отличным от стандартного напряжением (наш пытливый русский народ всегда стремится отчебучить что-нибудь эдакое), заходим в свойства (**Edit Properties**) и щелкнув по кнопке **Hidden Pins** вводим вместо VDD и VSS нужное нам питание и потенциал земли как и выше.*(См.картинку ниже)* .

Если необходим двухполюсник питания не связанный с землей, то используем BATTERY или VSOURCE из библиотеки **Simulator Primitives => Sources** для источников постоянного или **ALTERNATOR, VSINE, V3PHASE** *(трехфазный!)* для переменного напряжения.

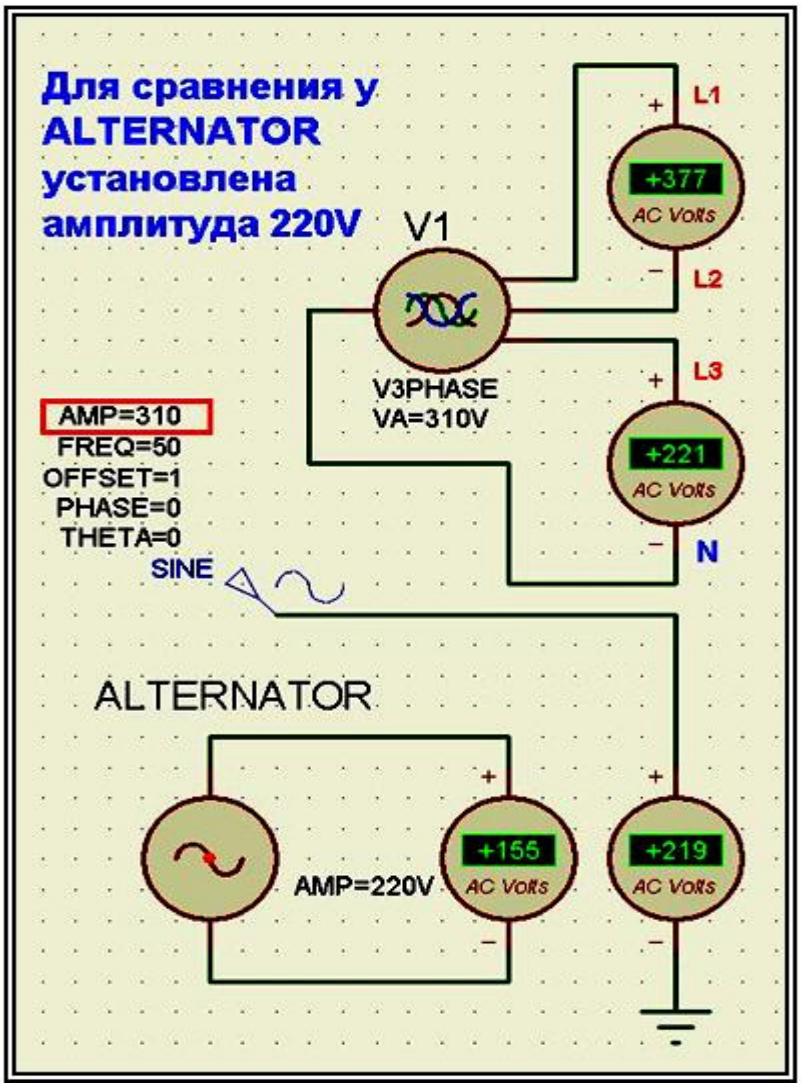


3.3. Вопрос касающийся генерации напряжения сети и применения синусоидальных генераторов.

Он плавно вытекает из упомянутых выше генераторов синусоидального напряжения. Здесь необходим ряд пояснений.

Если используется генератор из левого вертикального меню GENERATORS (однополюсники), то генерация сигнала осуществляется относительно земли (GND). Для аналоговых генераторов периодических колебаний (Sine) амплитудное значение напряжения в корень из 2 выше действующего. Это необходимо помнить при симуляции сети 220V (амплитудное значение устанавливаем 310V).

И еще одна особенность, а вероятнее всего глюк самого Протеуса – ну не хочет гореть лампочка (**Lamp - Active Model**) от синусоидальных генераторов кратных 10 Гц, т.е. 10, 20, 30…50…100 и т.д. А у трехфазного генератора это относится только к верхней фазе. Но достаточно установить частоту на 1 Гц отличающуюся от скажем 50 (например: 49 или 51) и модель лампочки имитирует свет! Имейте это ввиду те, кто пытается моделировать в ISIS различные светорегуляторы и т.п.



3.4. Визуализация выводов питания.

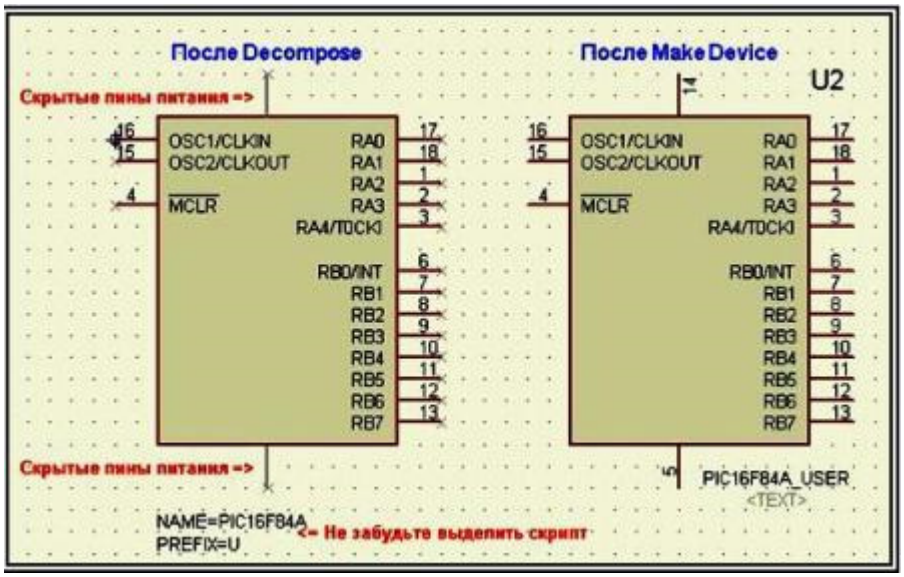
Иногда необходимо подвести провод питания к скрытому выводу (например VCC и VSS микропроцессора). Для этого придется немного поколдовать с моделью. Вся технология визуализации сводится к следующим действиям:

- Выбираем из библиотеки необходимый элемент (например PIC16F84A). Помещаем его на чистый лист, чтобы не зацепить что-нибудь лишнее при выделении. Щелкаем по нему правой лапой мыши или по соответствующему значку в верхнем меню и «разбиваем молотком» **Decompose** .
- После такой «варварской» разборки на составляющие мы и увидим сверху и снизу скрытые выводы питания бледно-серого цвета. Щелкаем по нужному выводу правой лапкой мыши и заходим в свойства (**Edit Properties**). Ставим нужные галочки: **Draw Body** (обязательно чтобы вывод стал видимым), остальные по вкусу (имя и номер и как их изображать горизонтально или вертикально). Убеждаемся, что вывод относится к **PP** (вывод питания) и давим **ОК** .
- Теперь выделяем левой лапкой мышки всю область с «разбомбленой» микросхемой, обязательно включая нижний текстовый скрипт. Все должно стать «как наши алые знамена». Щелкаем теперь уже правой лапой мыши внутри выделения и выбираем **Make Device** (можно соответствующей кнопкой в верхнем меню).

В первом появившемся окне обзываем наш девайс в **Device Name** по своему (чтобы не портить оригинал в библиотеке), например **PIC16F84A_USER** . Дальше **Next** -им окошки до последнего (кнопка **Next** станет не активной), убеждаемся, что наш девайс будет помещен в пользовательскую библиотеку **USERDVC** и давим **OK**. Убираем разбомбленный мусор с листа кнопкой Delete, а в левом окошке имеем наш микропроцессор с видимыми выводами питания, который и помещаем в свой проект. С аналогичным названием он будет фигурировать рядом с родным PIC16F84A в библиотеках Протеуса до тех пор, пока Вы не сподобитесь почистить пользовательскую библиотеку через **Library Manager** .

Если после разборки микросхемы серых выводов не обнаружится придется их дорисовать самостоятельно, присвоив им соответствующие имена и номера пинов и сконфигурировав как описано выше.

Любителей крутых Мег (от 64 и выше) вынужден пока расстроить: эти выводы в них реализованы только начиная с версии 7.3. Попытка дорисовать их ручками в 7.2.SP6 к желаемому результату не привела. В **Packaging Tool** они все равно остаются **Hidden** (скрытыми). Но выход из положения ищется.



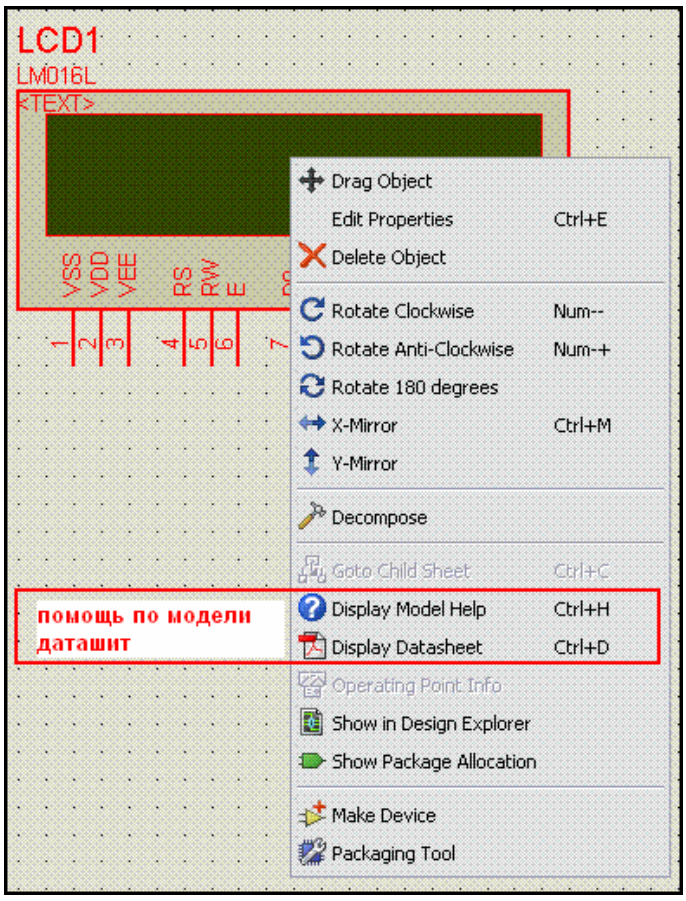
3.5. Вопросы по индикации и ндикаторам.

3.5.1 Алфавитно-цифровые ЖК дисплеи на основе HD44780.

В Протеусе эта группа сосредоточена в библиотеке **Optoelectronics => Alphanumeric LCDs** . А соответствующая им библиотека DLL носит название **LCDALPHA.DLL** . Большинство вопросов можно разделить на две группы: неправильно выводятся или не выводятся совсем символы и русский текст на дисплее.

Основная масса ошибок касающихся неправильного вывода символов возникает у тех, кто пишет программы на ассемблере, либо слепо использует чужие асм-коды и прошивки МК. Им следует помнить, что модель **HD44780** , используемого в Протеусе, написана строго по даташиту на данный контроллер, т.е. работает в соответствии с заложенными последовательностью и временами отработки команд и ответов на них. Некоторые авторы пренебрегают этим, т.к. реальные дисплеи допускают пропуски части команд в последовательности инициализации или отклонение времени их выполнения. Вот здесь чаще всего и возникают конфликты, причем «в железе» работает, а в симуляторе – нет. Рекомендация только одна – изучайте описание дисплея (Datasheet) и ищите ошибку. Тем более, что для LCD искать даташит по просторам Интернета и не потребуется. Достаточно при подключенном канале щелкнуть в ISIS правой кнопкой мыши по LCD помещенному в проекте и выбрать активный пункт **Display Datasheet** . Протеус сам предложит скачать даташит (406 кбайт английский PDF) и сохранит его для дальнейшего автономного использования в папке **Documents and Settings\All Users\Документы\Downloaded Data Sheets** . При щелчке правой кнопкой доступнен автономно и файл помощи – пункт **Display Model Help** , правда довольно краткий. Описанным способом можно скачать даташиты и на другие компоненты: графические дисплеи, микроконтроллеры, цифровые микросхемы, если активен пункт **Display Datasheet** . Причем легальный или нет у вас Протеус в данном случае не важно – регистрация не требуется – главное наличие подключенного канала Интернет.

Тем, кто использует для написания микропрограмм языки высокого уровня: Си, Бейсик, Паскаль повезло несколько больше – о них, как правило, позаботились авторы используемых компиляторов. В подавляющем большинстве компиляторов присутствуют готовые библиотеки для работы с LCD. Но и здесь есть «подводные камни». По умолчанию большинство компиляторов используют матрицу 5x7, в то время как для нормального отображения некоторых русских символов, например **Щ** (щука), требуется матрица 5x10, иначе нижний хвостик буквы будет усечен и ваш дисплей станет слегка «шепелявить». Здесь уже требуется хирургическое вмешательство в соответствующие библиотеки компилятора. Необходимо там найти команду установки режима дисплея (по даташиту это Function Set) и в ней поменять бит F (2-й бит) с нуля на единицу. Для наиболее распространенного 4-х битного режима с 2-х строчным дисплеем это выглядит как команда 0x28 (матрица 5x7) а необходима 0x2C (матрица 5x10).



3.5.2 Русский текст на экране LCD на основе HD44780 в Протеусе.

Теперь о русификации самой библиотеки **LCDALPHA.DLL** . Для ее перевода в русскоязычный режим в свое время одним из экс- (к сожалению) участников нашего форума и одновременно сотрудником Labcenter Electronics – Тенью была предложена утилита **charset.exe** . Утилита многократно проверена и работает. Суть ее действия заключается в следующем: вы берете указанную библиотеку из папки **MODELS** , извлекаете из нее ресурс в виде битмэп картинки, содержащей оригинальную таблицу символов и заменяете ее русифицированной таблицей, прилагаемой к утилите. Есть два замечания. Во-первых, необходимо использовать **LCDALPHA.DLL** именно от установленной у вас версии Протеуса, поскольку эта библиотека защищенная и меняется от версии к версии. Во-вторых, если вы не всегда используете русифицированные дисплеи, то не обязательно заменять оригинальную библиотеку модифицированной. Протеус имеет одну особенность, которой можно воспользоваться. Сначала он ищет доступные библиотеки и модели в папке с симулируемым проектом, а уже потом в своих библиотеках. Поэтому достаточно поместить русифицированную **LCDALPHA.DLL** в папку с вашим проектом в котором необходим русский текст на дисплее. Ну и последнее, при написании микропрограммы используются не русские буквы,слова и фразы, а ***массивы шестнадцатеричных кодов русских символов*** в тех местах, где необходим наш великий и могучий… На русском сайте компании ATMEL лежит бесплатная утилита для перекодировки русского текста в коды символов вот прямая ссылка на эту программу:
http://www.atmel.ru/Binary/HD-44780.exe
В прилагаемом ниже архиве содержится утилита **charset.exe** , русифицированная таблица **charset.bmp** и текстовый файл с инструкцией к применению.

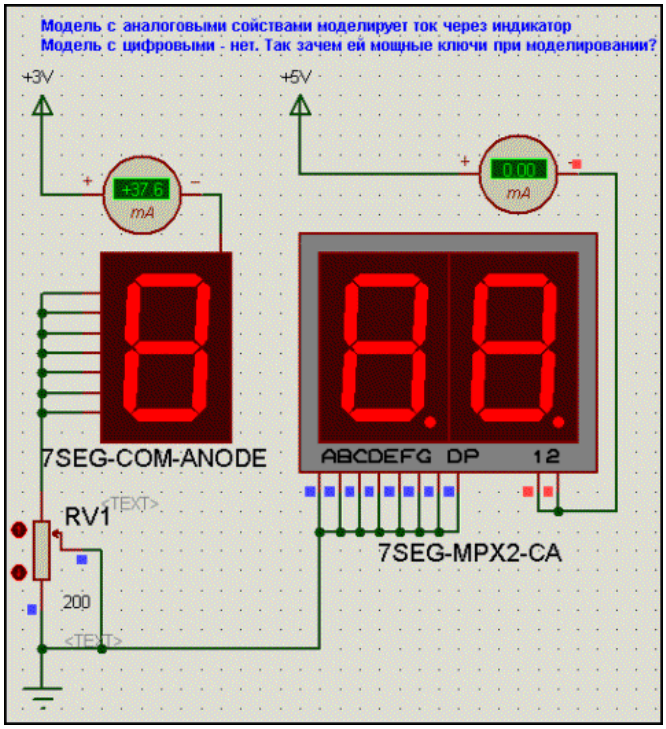
3.5.3 Динамическая индикация на семисегментных индикаторах в Протеусе.

Сразу же хочу обратить внимание всех, кто создает проекты с семисегментными индикаторами в Протеусе, что модели многоразрядных индикаторов чисто цифровые. Кто не верит – щелкните по индикатору правой кнопкой и выберите пункт **Display Model Help** (переведите первое предложение второго абзаца самостоятельно или посмотрите на приложенную картинку). Так что привешивание к ним в Протеусе всевозможных мощных транзисторных ключей, токоограничивающих резисторов и т.п. – это из разряда «аналоговых извращений» и здесь не рассматривается. Отмечу только, что это касается многоразрядных (2 и более) моделей, одноразрядные **Schematic** модели обладают аналоговыми свойствами, поэтому если их навешать достаточное количество – возникнет перегрузка ЦПУ компьютера. И еще, поскольку многоразрядные модели цифровые – яркость их свечения не меняется, чего не скажешь о реальных индикаторах. Поэтому при проектировании реального устройства необходимо учесть следующие азы:

1) человеческий глаз не замечает мерцания с частотой выше 25 Гц (вспомним про эффект 25 кадра), поэтому для исключения мерцания необходима минимальная частота обновления 25 x N (количество индикаторов);

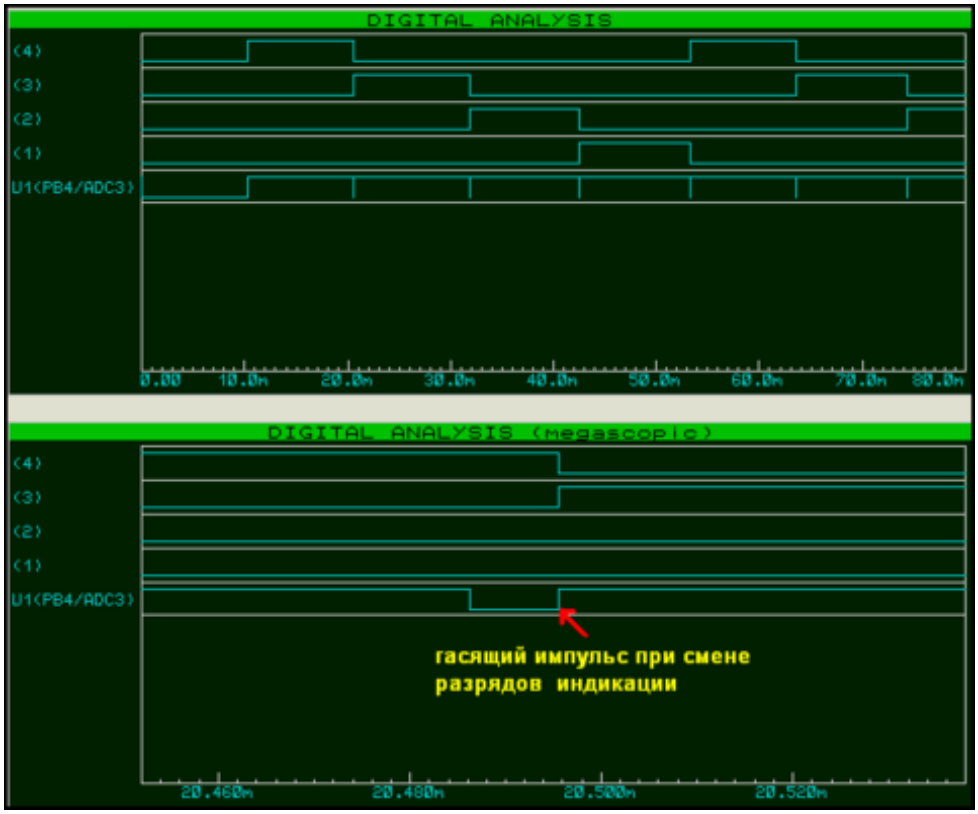
2) для того чтобы сохранить яркость свечения индикаторов в динамическом режиме – ведь средний ток через сегменты упадет – необходимо уменьшать токоограничивающие резисторы, а при большом количестве разрядов возможно и увеличивать напряжение питания индикаторов;

3) вот исходя из предыдущего пункта и вылезут на свет 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 и несколько меньше) т.е. получим от 192 до 640 мА!!! Здесь есть над чем поломать голову. Необходимо учесть и длительность импульса и реальные импульсные характеристики конкретного индикатора по его даташиту, чтоб его не спалить. Так что Протеус в данном случае может дать результат отличный от того, что вы увидите и увидите ли в реальном устройстве.



3.5.4 Анализ динамической индикации с помощью цифрового (DIGITAL) графа.

В примерах **(SAMPLES)**, прилагаемых к Протеус, я нашел только один, который демонстрирует характерные особенности использования динамической индикации – это **AVR Tiny15 Demo** . На нем и остановимся, т.к. в нем четко просматриваются те принципы, на которые в свое время в одной из веток форума указывал ***Dosikus*** , за что ему отдельное спасибо. До недавнего времени эта ветка существовала по ссылке:
http://kazus.ru/forum/topics/10434.html
Кроме того, материал из этой ветки с любезного разрешения автора размещен на сайте Каллиграфа:
http://www.kaligraf.narod.ru/nedodellki.html
Суть реализации метода заключается в том, что на момент смены разрядов индикация прерывается на короткий промежуток времени. Если этого не сделать – наблюдается наползание разряда на разряд и вместо желаемых цифр в лучшем случае будут отображаться восьмерки (все сегменты засвечены), а иногда и полная чехарда в виде отдельных горящих сегментов. Итак, копируем папку **AVR Tiny15 Demo** из **\SAMPLES\VSM for AVR** в удобоваримое место на диске, чтобы слегка модифицировать пример. Дело в том, что в исходном примере от версии 7.2 SP6 мне не удалось симулировать цифровой граф – Протеус ругался на модель семисегментного индикатора, а для рассмотрения особенностей динамической индикации он нам потребуется. Проблема решилась когда я удалил используемый индикатор из проекта и вновь добавил из библиотеки 7.2 SP6 (т.е. двойной щелчок по индикатору в проекте правой мышиной лапкой, затем в меню **Edit** по метле **Tidy** , ну а затем добавляем из библиотеки 7SEG-MPX4CA). Навешиваем **Voltage Probe** на выводы **1, 2, 3, 4** индикатора и на стробирующий выход МК - **PB4/ADC3** . Добавляем из **Graph Mode** в меню слева цифровой **(Digital)** граф в проект, добавляем в него по одному **Add Traces => Probe1** установленные пробники, ограничиваем в свойствах графа время **Stop Time** на 80m или 100m (мсек) и шлепаем по пробелу (или **Simulate Graph**). Я так подробно здесь рассматриваю использование графов, чтоб не описывать их использование отдельно по ним тоже часто задаются вопросы. На приведенных на рисунке графах хорошо видно что в конце засвечивания каждого знакоместа (высокий уровень **4,3,2,1**) проходит гасящий строб (**PB4/ADC3**) на нижнем графе это место увеличено. Делается это следующим образом: щелкаем по графу правой лапкой мыши выбираем **Maximaze** внизу в меню на раскрывшемся графе будут дополнительные кнопки прокрутки времени (стрелки) увеличения и уменьшения (4 различных лупы Zoom). Если необходимо поставить маркеры щелкаем по тому месту где он нужен (зеленый) и с нажатой **Ctrl** вторую точку (красный) внизу на черной полоске отразится их временное положение в цифровой форме и приращение (dx) между ними. Слева на оси Y отображается состояние сигнала для зеленого (основного) маркера. Итак, поигравшись с нашими графами, мы имеем следующее: суммарный период индикации 4 цифр составляет около 43 мсек (частота соответственно около 24 Гц), длительность сигнала на знакоместе около 10,5 мсек при этом в конце его следует гасящий стоб длительностью около 10 мксек.



Подведем итоги нашего исследования:

*** В классическом рабочем, прилагаемом с Протеусом, примере применен именно тот метод, который предложил в свое время ***Dosikus*** . И метод работает на все 100%.

*** Для того, чтобы избежать чехарды на индикаторах в Протеусе необходимо организовать гашение индикации на период смены разрядов. Время, на которое гасится индикация может быть достаточно малым, но оно должно быть. Как его организовать – это уже прихоть разработчика. Если у вас внешние буферные микросхемы (как в расмотренном примере) то можно стробировать их отдельным сигналом, если индикаторы подключены непосредственно к порту МК, можно, например, переводить выходы порта в высокоимпедансное состояние – фантазия тут безгранична – главное результат.

*** Использование одноразрядных аналоговых моделей индикаторов (на моем Атлоне 64 с частотой 3200 МГц свыше трех штук) приводит к 100% загрузке процессора компьютера и срыву симуляции в реальном времени.

*** Ну и наконец главное – моделирование динамической индикации в Протеусе не дает полной гарантии работоспособности реального устройства, т.е. потребуется еще и отработка на макете, но тем не менее позволит значительно сократить время разработки.

Список наиболее интересных на мой взгляд ресурсов и литературы, в которых рассматриваются вопросы динамической индикации:

Книги (ссылки ищите в соответствующей ветке форума):

*** 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 яркости http://arv.radioliga.com/inde...ent&task=view&id=101&Itemid=49*(интересная идея по ШИМ регулировке яркости индикаторов от Романа Абраша автора цикла статей по МК в журнале «Радиолюбитель»)*

3.6. Подробнее о графах (или графиках, если кому-то нравится такое название).

3.6.1. Что можно сделать с помощью графов?

Мой ответ однозначный - практически все, даже тогда, когда реал-тайм симуляция бессильна или у вас старенький "ПЕНЬ", а не суперкомпьютер. В частности HELPe авторами программы упоминаются такие исследования, как анализ слабых сигналов, шумов и параметрический анализ. Я не знаю, почему так незаслуженно обходится стороной их применение у русскоязычных пользователей Протеуса. Те, кто работает с этой программой не первый год, наверное давно раскусили возможности этой опции ISIS, но помалкивают о подробностях. Попробую приобщить начинающих к применению графов в своих разработках. Графы по своей сути очень близки к запоминающему осциллографу, плюс к тому в отличие от интерактивного осциллографа, который в седьмой версии не лишен некоторых "глюков", дают более реальную картинку происходящего процесса. К тому же в отличие от четырехканального осциллографа на графе можно расположить большее количество одновременно наблюдаемых сигналов. Мы уже проделали это пунктом раньше, расположив 5 наблюдаемых каналов в цифровом графе и это не предел! Итак, вопросу применения графов для анализа посвящен раздел **GRAPH BASED SIMULATION** в **ProSPICE HELP** (он же **PROTEUS VSM HELP**) - многое там понятно по карьинкам и без знания английского. Кроме того, в **Samples** к Протеусу есть отдельная папка **Graph Based Simulation** , полностью посвященная примерам с использованием графов. Заниматься подробным переводом HELP-а я не имею ни возможности, ни желания, но остановлюсь на некоторых ключевых моментах.

3.6.2. Какие графики можно симулировать в Протеусе? Примеры использования.

Эти вопросы рассмотрены в разделе **ANALYSIS TYPES** того же **ProSPICE HELP**. Остановлюсь на самых употребимых к остальным попытаюсь дать только ссылки на примеры в Протеусе.

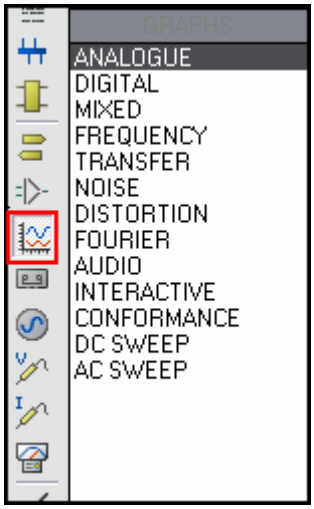
Щелкнув по соответствующему значку в левой панели, мы получим меню для выбора типа графика, размещаемого в проекте (всего их 13 типов). Наиболее употребимые в обиходе (с моей точки зрения) это: **ANALOGUE** (аналоговый), **DIGITAL** (цифровой), **MIXED** (смешанный). А в последнее время я все чаще стал их эксплуатировать **INTERACTIVE** - интерактивный граф и **CONFORMANCE** - график соответствия. На них чуть ниже остановимся особо. Остальные:

FREQUENCY - частотный анализ, **NOISE** - анализ шумов, **DISTOTION** - анализ искажений. Смотрите в примере 741.DSN. *При запуске анализа будут вылетать "горчичники" об отсутствии Pin1 и Pin2 на дочернем листе - это соответственно выводы коррекции 1 и 5, которые в дочернем листе не прорисованы. Для всех этих графов в свойствах (Properties) аналогично: Reference - источник, Start и Stop Frequency - начальная и конечная частоты исследования, Interval - шаг по оси X, No Steps/Interval - количество шагов исследования на интервал.*

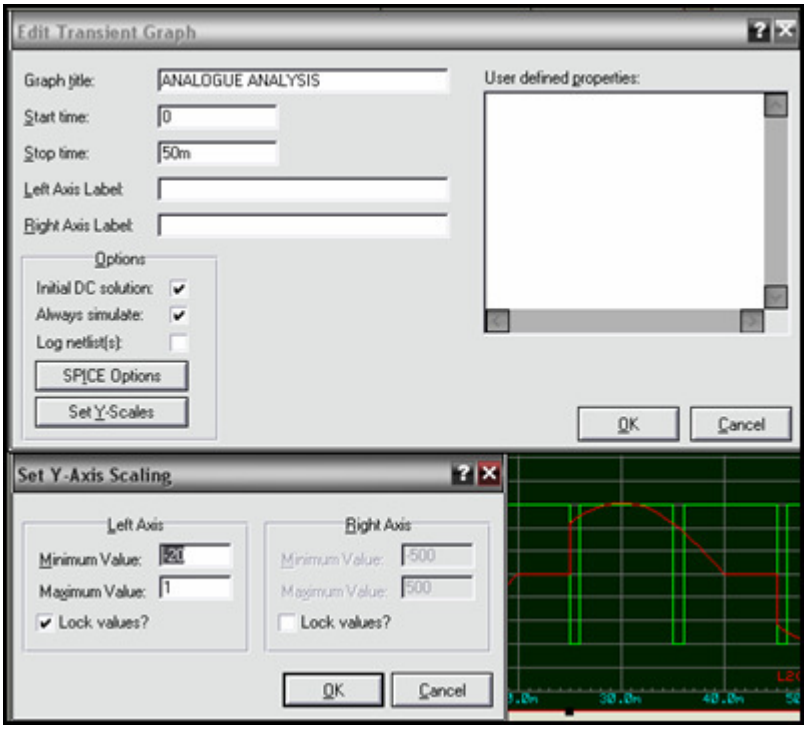
TRANSFER - анализ передаточных характеристик. Смотрим одноименный TRANSFER.DSN. *Ба, да это готовый справочник по импортным транзисторам. Да еще в стиле старых добрых советских справочников, только меняй тип транзистора из библиотеки. В современных этих графиков не найдешь. Я думаю, в назначениях свойств разберетесь самостоятельно, или с помощью старого справочника по транзисторам, где помещались подобные передаточные характеристики.*

DC SWEEP и **AC SWEEP** - создают соответственно график изменения напряжения (тока) и семейство частотных характеристик в зависимости от изменения какого либо параметра компонента. Смотрите в примере SWEEP.DSN. *Для DC показана зависимость изменения тока через диод от приложенного напряжения, а для AC семейство частотных характеристик RC-цепочки при изменении R. В свойствах графа также довольно легко понять что и как меняется.*

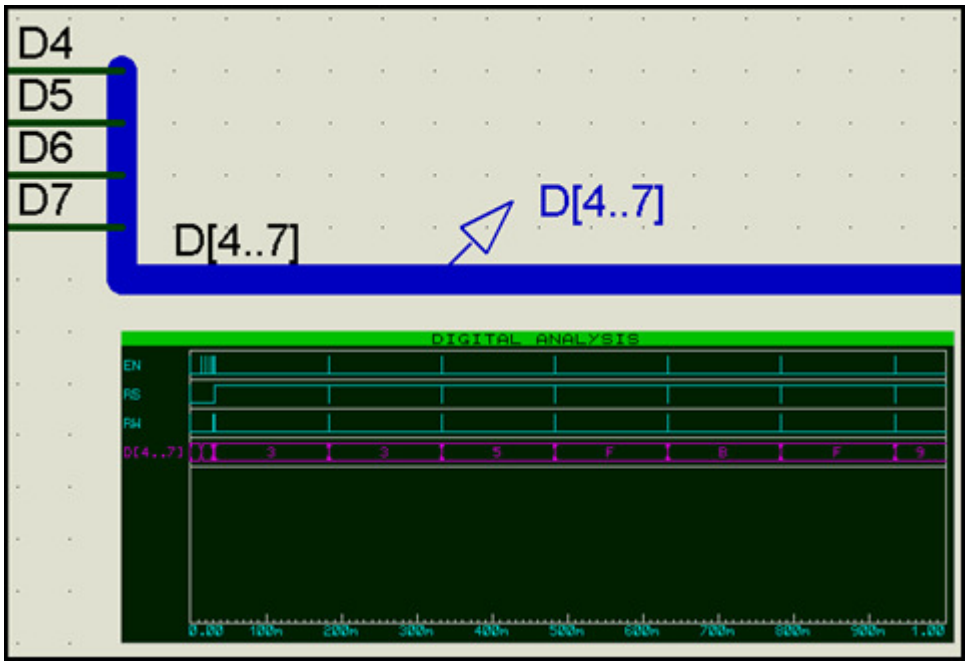
AUDIO и **FOURIER** - как и следует из названия соответственно анализируют звуковые сигналы и спектральный анализ сигнала по Фурье. Смотрите пример их применения в FOURIER.DSN. *Великолепный классический пример синтеза прямоугольного сигнала из ряда синусоидальных сигналов.*



Теперь подробно о моих излюбленных:
ANALOGUE - сродни обычному запоминающему осциллографу. На нем можно анализировать как аналоговые, так и цифровые сигналы. Графики различных сигналов будут различаться цветом, но при этом накладываться один на другой. В свойствах графа (см. картинку) можно задать начальное (**Start Time**) и конечное (**Stop Time**) время анализа. В **Left** и **Richt Axis Label** можно ввести названия которые будут показываться у соответствующих вертикальных шкал. Если щелкнуть по кнопке **Set Y-Scales** , можно установить фиксированные значения для верхнего и нижнего предела каждой шкалы, установив галочку **Lock Value** . Для цифрового графа эта функция будет неактивна.



DIGITAL - практически заменяет многоканальный логический анализатор. В отличие от последнего позволяет наблюдать сигнал непосредственно на многопроводных шинах. Для этого на шину устанавливается V-пробник с именем, соответствующим лэйблу шины. Естественно на шину предварительно как на провод вешается **Label**, но наименование у него должно быть несколько отличное от отдельных проводов. Например в шину входят 4 провода с именами от **D0** до **D3** . Тогда установленный **Label** должен иметь название **D[0..3]** и такое же значение присваивается пробнику. Обратите внимание диапазон охвата в квадратных скобках должен соответствовать количеству проводов в шине и между цифрами ставится не троеточие как принято у нас, а **ДВЕ** точки. При этом на графе в стабильные по состоянию промежутки времени высвечивается цифровое шестнадцатиричное значение шины (сиреневая трасса).



Следующие два типа предусматривают некоторое оперативное вмешательство исследователя в ход анализа.

INTERACTIVE - комбинированный тип исследования, т.е. стартует симуляция, а результат записывается в график. Это очень удобно, когда надо просмотреть реакцию устройства на изменение состояния какого либо управления (например кнопки). Устанавливаем время записи в графе достаточным, чтобы успеть "нажать" кнопку мышью на схеме (например: старт - 0s стоп - 10s) и запускаем граф и симуляцию одновременно клавишей пробел. Пока бежит прогресс-бар внизу (крестики) изменяем мышкой состояние кнопки. По окончании симуляции анализируем на графике результат.

CONFORMANCE - цифровой график совпадения (или соответствия). С этим типом графика я сам работаю недавно, но он представляет значительный интерес для разработчика. Суть его применения рассмотрим на конкретном примере:

Предположим вы разработали цифровое или микропроцессорное устройство в котором один из выходных сигналов должен находиться в строгих временных интервалах. В моем примере (Приложенная картинка) данное устройство имитирует CLOCK-генератор, который я обозвал **SIGNAL** и задал ему частоту 1,8 Гц. Мне необходимо изменить схему или микропрограмму, но при этом частота не должна меняться более чем на 0,2 Гц. Для контроля я поместил в проект два вспомогательных генератора одиночных импульсов и задал им параметры так, чтобы фронты моего **SIGNAL** находились в пределах действия

положительных импульсов и желательно ближе к середине. Эти два генератора я собрал через примитив ИЛИ и обозвал сигнал **CONTROL** . Затем вставил в проект **CONFORMANCE** -граф и первым поместил туда мой контрольный сигнал. Это всегда верхняя трасса желтого цвета. Затем вставил туда свой подопытный **SIGNAL** и просимулировал граф. В свойствах этого графа имеются две дополнительные кнопки: **Store Results** (Сохранить результаты) и **Reset Graph** (Сбросить граф) первой из которых я потом и воспользовался. При этом все трассы на графе стали серыми (запомнились). Затем я "изменил" что-то в схеме или микропрограмме (в моем примере я просто повысил частоту до 1,9 Гц) и для проверки, что параметры у меня не выскочили из заданной вилки, снова запустил симуляцию графа. Она благополучно закончилась, поскольку фронты сигнала сползли, но при этом остались в зоне совпадения (отсюда и название) с высокими уровнями на трассе **CONTROL** . Этот результат видно визуально по разнице между серой (сохраненной) трассой и новой голубой. Если же я задаю частоту 2,1 Гц, то при попытке симуляции получаю сообщение:
Data mismatch at time 250m in trace 'SIGNAL'.
Data signatures were 'L' and 'H'.
Conformance analysis FAILED
Даже не переводя нетрудно понять, что на трассе **SYGNAL** в районе 250мсек оюнаружено несовпадение сигналов, а на графе это можно будет посмотреть визуально передний фронт улетит за пределы вилки заданной первым контрольным импульсом. Таким образом можно контролировать поведение цифровой схемы при внесении изменений.

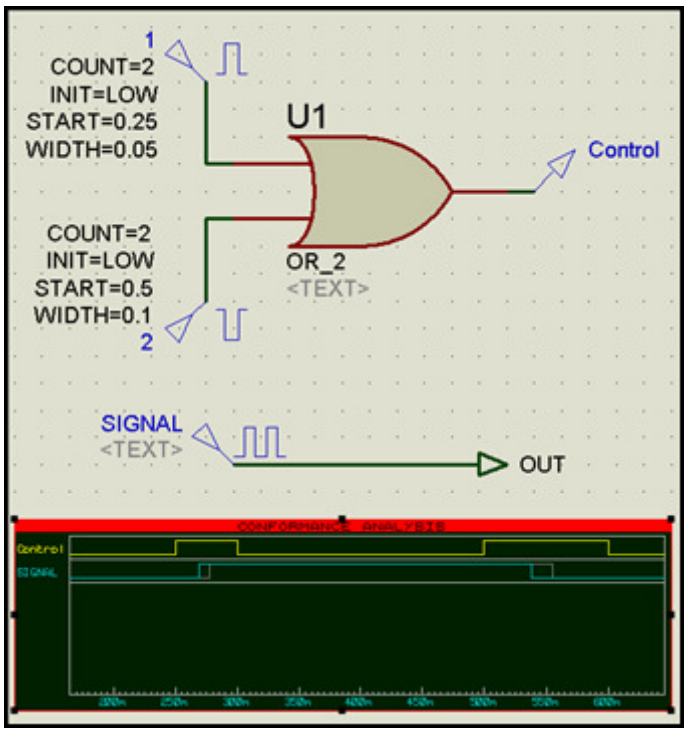
3.6.3. Немного о зондах (пробниках) и некоторые фичи с графами.

О пробниках особенно распространятся не стану их всего два: напряжение и ток. Для начинающих только замечу, что зонды напряжения в случае установки на цифровой провод показывают логический уровень (**SHI** -высокий, **SLO** -низкий), а не напряжение. Токовые зонды должны быть ориентированы по направлению протекания тока стрелкой в круге, за положительное направление принято от плюса к минусу. Вторая особенность токовых пробников - они работают только на аналоговых цепях так что на цифровые линии их лепить бесполезно. Оба типа пробников поддерживают запись исследуемого сигнала в файл (галочка **Load to File?**), который можно потом использовать для отладки другого каскада в другом месте. Пробник напряжения содержит еще свойство **Load to Ground** . На нем остановлюсь подробно. Пробник измеряет напряжение относительно потенциала земли (**GND**) и если вы установили зонд в точку, не имеющую гальванической связи с землей (например, имеется разделительный конденсатор), то он у вас измерять не будет. Вот эта опция и служит для создания связи с **GND** и, установив флажок, в соответствующей ячейке вы должны указать сопротивление нагрузки на землю, чтобы получить реальные показания. Теперь несколько фичей.
a) Для особо ленивых. Генераторы и зонды на граф можно запихивать прямо мышью. Первым щелчком левой лапки по генератору или пробнику "вгоняем его в краску". Затем цепляем левой лапкой за объект (он подсвечивается пунктирным прямоугольником) и, удерживая лапкой, тянем его на черное поле графа и там бросаем, отпустив лапку.
b) Максимизировав (**Maximize**) граф вы получаете два маркера зеленый при первом щелчке по полю графа, потом его можно перемещать, удерживая левой лапкой животного и красный - те же манипуляции при нажатой клавише **CTRL** . Цифровые значения внизу на ленте, желтым приращение по оси X, даже считать не надо.
c) На максимизированном файле доступными становятся кнопки зумов (увеличения уменьшения масштаба по оси X). Если вы увеличите масштаб, например кнопкой **Zoom To**

Area (Увеличить выделенную область, ее после щелчка надо обвести с нажатой левой мышиной лапой), то, закрыв граф щелчком потом по кресту в правом верхнем углу, получите в свернутом виде на графе в проекте масштабированное изображение.Обратный процесс через кнопку зума **Zoom to View Entire Sheet**.

d) Любой результат можно сохранить через **Export Graph Data** в графическом или векторном формате. Выбрав при этом черно-белый режим, получите картинку удобную для печати на принтере и с малым размером файла.

e) При использовании интерактивного графа или симуляции иногда требуется нажать две кнопки сразу. Забейте в отдельную строку в окошке **Other Properties** для этих кнопок (или переключателей) строку следующего содержания: **GANG=1** (для другой группы соответственно гонг 2 и т.д.). При нажатии одной кнопки вторая из этой группы будет переключаться синхронно.



4. Многостраничные проекты. Иерархия структуры проекта (дизайна) в ISIS.

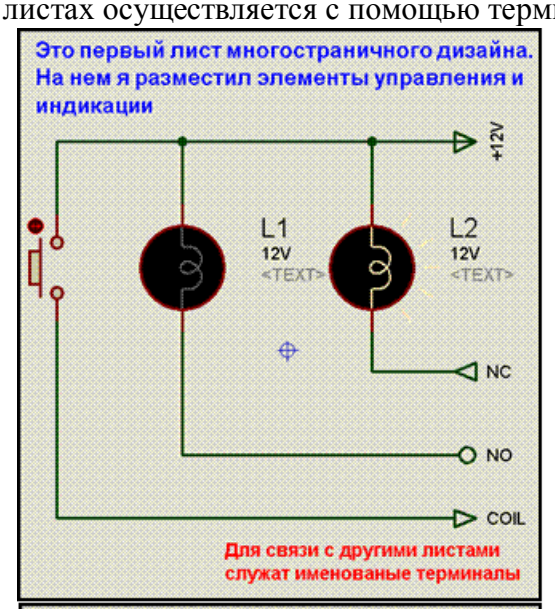
Прежде чем приступить к рассмотрению создания собственных моделей хочу остановиться на простом и эффективном способе, который зачастую избавит Вас от лишних затрат времени на проектирование. Поводом послужили несколько вопросов на форуме, которые можно свести к рассмотрению двух из раздела **ISIS HELP: MULTI-SHEET DESIGNS** (Многостраничные проекты).

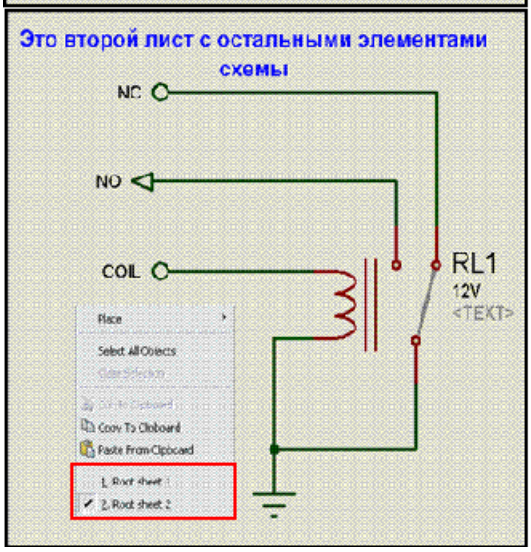
4.1. Многостраничные проекты.

Схема не помещается на листе. Как ее разместить на нескольких листах?

Тут все очень просто. В закладке **Design** щелкаем **NewSheet** и получаем второй, третий и т.д. Навигация между листами осуществляется клавишами **PageUp PageDown** , либо (для тех у кого правая рука приросла к животному) через всплывающее меню правой лапки хвостатого перестановкой флажка на нужный лист.

Через закладку **Design** можно также перемещатся между листами, отредактировать их имена (по умолчанию **ISIS** обзывает их **Root Sheet1, 2…**) или удалить листы. Кстати в **Edit Sheet Properties** для **Sheet Title** допустимо русское название. Ну и напоследок совет, который в свое время дал ***dosikus*** , и которым я тоже пользуюсь при разработке: размещайте все органы управления и индикацию на первом листе, а все, что не требует визуального контроля на последующих. Это удобно при симуляции - всегда управление под рукой. А связь между элементами на различных листах осуществляется с помощью терминалов.





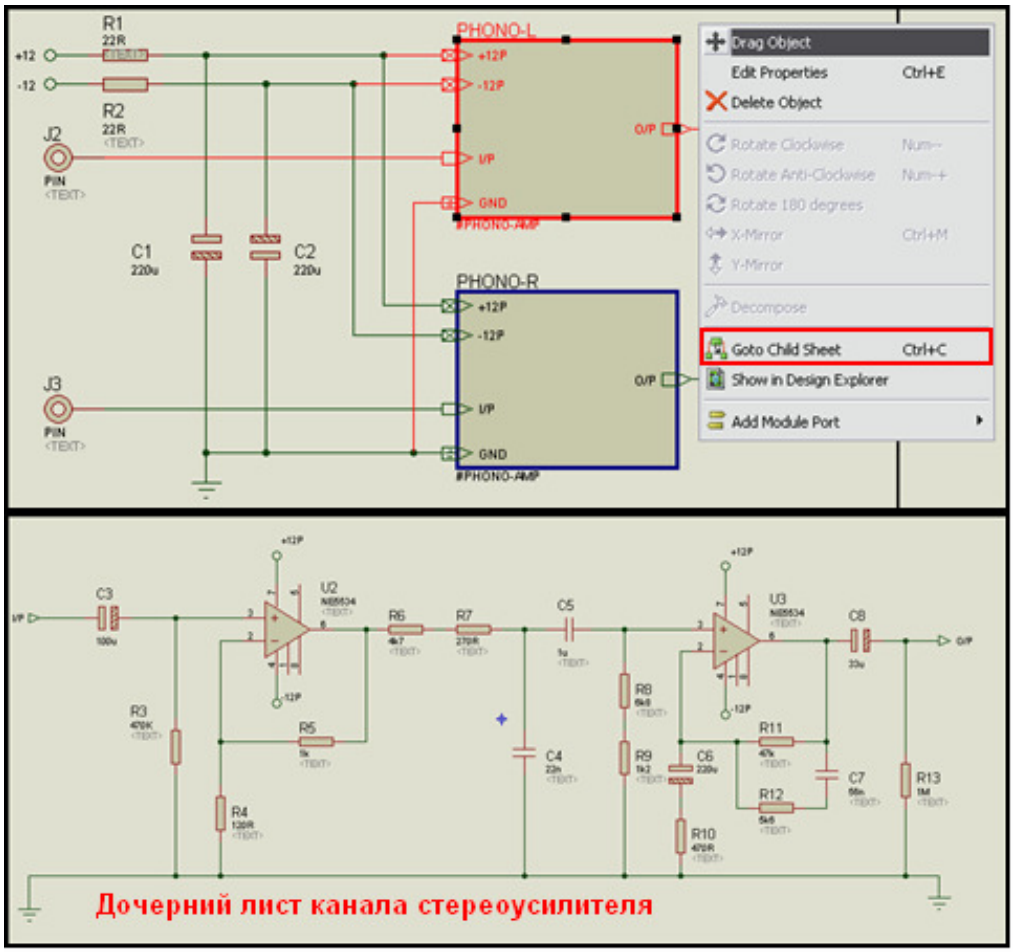
4.2. Иерархические проекты

Надоело Copy-Paste-ить из проекта в проект одну и ту-же часть схемы. Как создать свою "коробочку" (правильнее "черный ящик"), чтобы потом вставлять ее в последующий проект?

Начну с почти дословной цитаты из **HELP ISIS**: *"ISIS поддерживает иерархическую структуру проекта, который может содержать более полдюжины уровней. Как и в любом проекте самый высокий уровень может являться блок-схемой устройства и содержать как минимум один подлист с проектируемой схемой. Тот в свою очередь может содержать дополнительные дочерние листы содержащие субсхемы или модули и т.д. Таким образом можно спроектировать достаточно сложное устройство."*

4.3. Примеры иерархического построения проектов поставляемые с Протеусом.

Откройте в **ISIS** проект **EPE.DSN** из папки **SAMPLES\Schematic & PCB Layout** (программатор-эмулятор EPROM). Проект содержит три листа. В свою очередь на листах 2 и 3 находятся модули (субсхемы **ERAM** и **PPSU**). Они имеют синюю рамку. Если щелкнуть правой лапкой мыши по этой рамке становится активным пункт всплывающего меню **Goto Child Sheet** (переход в дочерний лист), который и содержит принципиальную схему модуля. В одном случае это модули памяти **ERAM** , в другом стабилизаторы напряжения **PPSU**
Вернуться назад можно также через правую кнопку хвостатой, выбрав пункт **Exit to Parent Sheet** (Возврат в родительский лист).
Другой характерный классический пример - два канала стереоусилителя в файле **Features.DSN** . В правом верхнем углу полотна проекта находится этот пример.
Прикладываю картинку на которой заодно и показан пункт перехода на дочерний лист



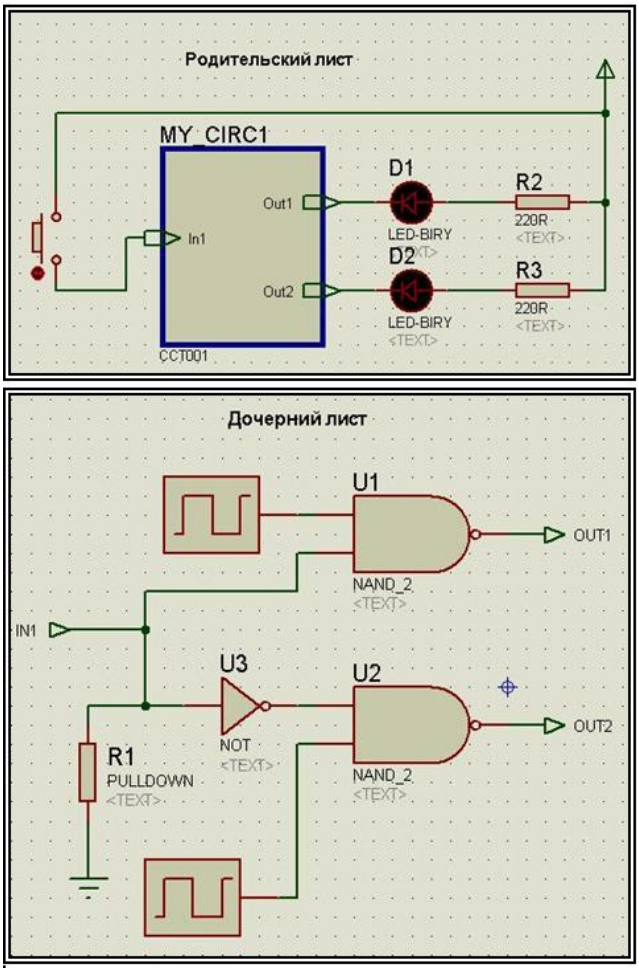
4.4. Создание модуля (субсхемы).

Создадим новый проект. 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.

Выбираем в левом меню режим **Terminals Mode** и, щелкая левой кнопкой мыши по периметру, расставляем нужное количество требуемых портов для связи с дочерним листом (общепринято: входы слева, выходы справа). То же самое можно проделать через правую лапку мыши пункт **Add Module Port** .

Даем самому модулю и его портам соответствующие названия и, как описывалось выше, переходим в дочерний лист. Пока он у нас пустой. Набираем из библиотек нужные компоненты и рисуем схему как обычно. Расставляем на входах выходах терминалы и выбираем им названия в соответствии с модулем на родительском листе. Я набросал самый простой пример, чтоб поместился на картинке. Рисуем, затем переходим на родительский лист, проверяем в симуляции что светодиоды моргают в зависимости от нажатой кнопки.

Из любого листа можно произвести **Export Section** через меню **File** для последующего **Import Section** в другой лист, однако пока у нас составляющие модуля и подсхема не связаны тесно между собой. Если произвести копирование и вставку модуля на родительском листе, то те компоненты второго дочернего листа, которые имеют авто-нумерацию получат персональные номера в **Design Explorer** . Но, в моем примере применены примитивы цифровых генераторов без префиксов, их придется перенумеровывать вручную. Изменения номиналов компонентов в одном сублисте не затрагивают другой. Это не очень удобно, однако, даже в таком виде с модулями работать удобнее, например, создавая стерео УНЧ. Позднее мы устраним эти неудобства.

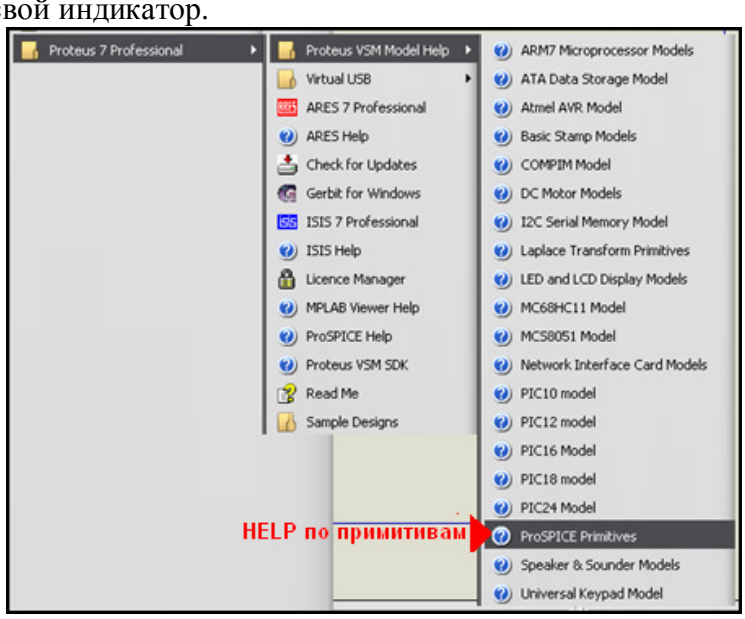


5. Обзор моделей компонентов используемых в Протеусе.

То как мы легко запихнули в "черный ящик" часть схемы пытливую часть пользователей Протеуса сразу натолкнет на мысль собрать таким же способом отсутсвующий в библиотеке компонент. И Протеус предоставляет Вам такую возможность. Но сначала для неискушенных рассмотрим: какими же моделями пользуется **ISIS** при симуляции работы схем. Далее материал будет излагаться на основе **VSMSDK.HLP** от версии 6.3, который в в более поздних версиях отсутствует. Поэтому желающим освоить этот материал настоятельно рекомендую скачать приложенный к этому посту архив с указанным **HELP** , чтобы иметь его под рукой в оригинале. Там же папка **INCLUDE** от версии 6.3, содержащая библиотеки для создания VSM моделей. Спасибо ***dosikus*** и ***as205*** за предоставленный материал.
Итак все модели, которые используются в Протеусе можно подразделить на две категории: электрические и графические. Их, в свою очередь, можно поделить на более мелкие группы. Об графических моделях чуть позже, а сейчас рассмотрим электрические, т.к. именно они обеспечивают возможность симуляции (имитации) работы устройства. Дальнейший материал базируется на разделе **TYPES OF MODEL** вышеуказанного **HELP** .

5.1. Модели-примитивы (Simulator Primitives).

Эта часть элементов встроена в **PROSPICE** симулятор как **SPICE3F5** для аналоговых или **DSIM** для цифровых компонентов. Эти элементы могут использоваться как непосредственно в симуляторе (такие как резисторы, конденсаторы, диоды, транзисторы, цифровые элементы), так и для создания более сложных устройств в качестве "набора-конструктора". (Вот последнее свойство для нас и важно!)
Примитивы определяются симулятором по их свойствам:
PRIMITIVE=ANALOG,NPN - аналоговый NPN-транзистор;
PRIMITIVE=DIGITAL,NAND_2 - цифровой элемент 2-И-НЕ;
Примитивы находятся соответственно в **ASIMMDLS** и **DSIMMDLS** библиотеках, а при выборе **Pick from Libraries** берутся из **Modelling Primitives** . Большинство моделей обладает дополнительными свойствами, которые редактируются в окне, вызываемом через **Edit Properties (Cntl+E)** . Описания свойств доступны через **ProSPICE Primitives** (см. картинку), часто туда можно попасть через пункт **Display Model HELP** при щелчке по ней правой кнопкой или через кнопку **HELP** в окне редактирования свойств. Ряд моделей - **Real Time Probes** предназначены для создания активных моделей. В частности на них я покажу позже как сделать свой индикатор.



5.2. Схематические модели (Schematic Models).

Большинство методов моделирования более сложных схем таких, как операционные усилители, сложные цифровые микросхемы основано именно на создании схематичной модели, обладающей эквивалентным поведением в симуляторе из простых примитивов. При этом эквивалентная электрическая схема (кстати, не всегда полностью повторяющая внутреннюю структуру компонента) создается на дочернем листе. Затем эта схема тестируется и, если она совпадает по электрическим свойствам с оригиналом, компилируется с помощью встроенного в **ISIS Model Compiler** в файл **MDF** (Model Description Format).
Чтобы закрепить **MDF** файл за определенной моделью используется свойство **MODFILE** .
Например, для ОУ 741: **MODFILE=OA_BIP**
Когда **ISIS** обнаруживает в схеме ОУ 741, он автоматически заменит этот компонентпри симуляции схемой и свойствами из соответствующего **MDF** файла. С помощью карты (**MAP**) свойств, размещенной на дочернем листе можно различным элементам назначить один **MDF** файл.
Так, например, Лабцентер поступил со всей логикой. И в самом деле, зачем плодить **MDF** файлы скажем для 7400, 74ALS00, 74HC00 и т.д., если все они содержат по четыре элемента 2И-НЕ, имеют одинаковую распиновку корпуса и все отличие в быстродействии. Достаточно создать карту, где указаны длительности фронта/спада и минимальная длительность импульса для каждого типа и скомпиллировать ее в тот же MDF, где схема. А при симуляции, в зависимости от того какой конкретно элемент стоит на схеме **ISIS** в соответствии с картой выберет для него нужные параметры.
MDF файлы в Протеусе объединены в библиотеки с расширением **LML** .

5.3. SPICE модели.

Формат **Беркли SPICE** фактически стал стандартным для большинства аналоговых устройств. Многие изготовители делают доступными **SPICE** модели для производимых ими компонентов. Для их поддержки **PROSPICE** снабжен несколькими тысячами таких моделей. Вы можете подключить сторонние SPICE-модели.
Так как SPICE-модели имитируются непосредственно ядром, для остальной части системы они представлены как примитивы, например:
PRIMITIVE=ANALOG,SUBCKT
SPICEMODEL=LM741/NS
SPICELIB=NATOA
SPISE-модели с определенными специфичными для группы компонентов свойствами объединены в библиотеки с расширением **SML** (SPICE Model Library).

5.4. VSM модели (VSM Models).

VSM модели аналогичны примитивам, за исключением того, что собраны в библиотеки **DLL** , а не встроены непосредственно в **PROSPICE** . Использование **DLL** представляет эффективную альтернативу схематическому моделированию особенно, когда моделируются очень сложные устройства такие, как микропроцессоры. Еще одним уникальным свойством **Proteus VSM** является то, что VSM модели предоставляют возможность использования графического интерфейса с пользователем. Связь **ISIS [b] и [b] ProSpice** с программной моделью устройства осуществляется через программный **C++** совместимый интерфейс **VSM API** .
Обычно свойства VSM-моделям назначаются так:
PRIMITIVE=DIGITAL,8052
MODDLL=8051.DLL
Здесь первая строка указывает **ISIS** , что это примитив, моделируется непосредственно в **PROSPICE** и не надо искать для него **MDF** файл, а вторая указывает библиотеку DLL, в

которой описан этот элемент.
От себя добавлю, что именно **VSM** позволяет имитировать в Протеусе такие сложные и быстрые устройства, как микроконтроллеры, но именно этот вид моделирования наиболее сложен и подвержен "глюкам", поскольку требует хорошего знания языка C++, функций **VSM API** (вот эти то библиотеки и содержатся в папке **INCLUDE**), а также наличие на компьютере **MS VisualC++ 6.0** или выше для компиляции исходников в библиотеки DLL.

6.1. Пример построения графической модели К176ИЕ12.

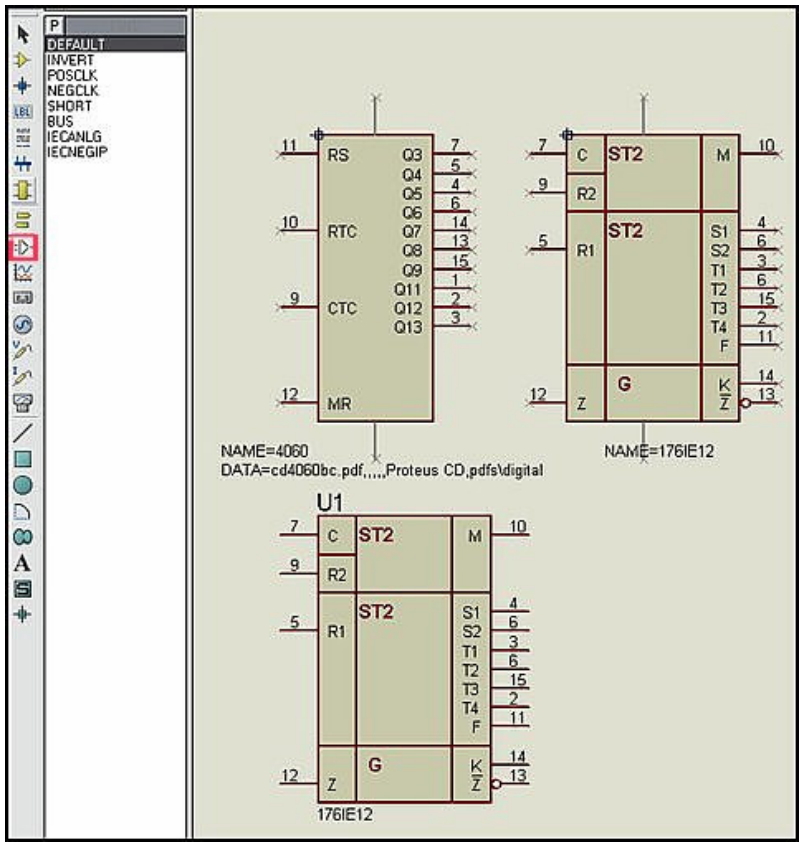
Итак, я решил создать модель старенькой часовой микросхемы К176ИЕ12. Наиболее близкой по назначению и строению оказалась CMOS модель микросхемы 4060. Для тех, кто с ней не сталкивался, поясню - это 14-разрядный двоичный счетчик-делитель со встроенной схемой генератора на инверторах, то есть практически первая половина нашей 176ИЕ12 (в ней первый счетчик 15-разрядный). Кладем микросхему 4060 на чистый лист, бьем молотком (**Decompose**) и из получившегося набора собираем свою - в этом варианте я решил слегка изменить графическое изображение (в таком виде оно лучше для пояснений принципа работы).

Текстовый скрипт, который находится под телом микросхемы для начала может содержать всего четыре строчки. Обязательно измените вторую **NAME** , иначе, при создании устройства вы рискуете затереть имеющуюся модель 4060:

```
{*DEVICE}
NAME=176IE12
PREFIX=U
{*PROPDEFS}
```

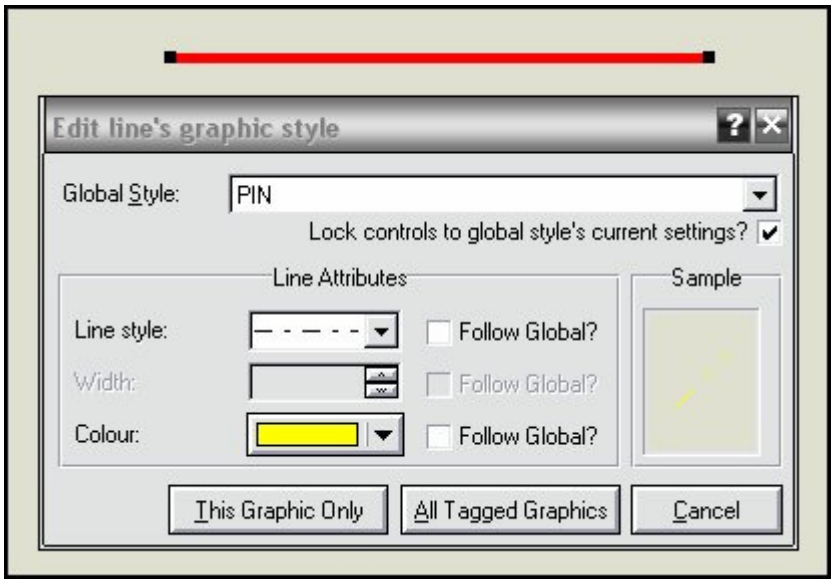
Теперь обводим, удерживая левую кнопку мыши, наше создание и щелкаем Make Device - создать устройство. Там всего пять вкладок. Все время давим **Next** , ничего не меняя. На пятой вкладке эта кнопка станет неактивной. В окна **Device Category** и **Device Manufacturer** прописываем через сопутствующие кнопки **New** соответственно **CMOS176** и **ExUSSR** . Это чтоб не потерять их потом в библиотеках Протеуса. Давим **ОК** . Ну вот мы и получили свою первую графическую модель.

Удаляем строительный мусор из проекта. Наша модель 176IE12 теперь существует в библиотеке Протеуса и мы всегда ее можем оттуда достать. Плюс к тому она появилась и в левой панели выбора девайсов. На картинке слева "разбитая" 4060, справа еще не "мэйкнутая" 176ИЕ12, а ниже уже созданная модель (обратите внимание: меню слева стоит в режиме добавления выводов-пинов).



6.2.Некоторые тонкости создания графики модели.

Как видно из рисунка выше я добавил линии и три надписи внутри тела компонента. Протеус имеет предустановки рисования, относящиеся ко всем элементам **2D Graphics** из левого меню (линия, прямоугольник, окружность, дуга, и т.д.). При выборе любого из этих режимов они высвечиваются в левом белом меню выбора стиля (так же как, на картинке выше в меню выбора стоит выбор типа добавляемого пина: DEFAULT, INVERT и т.д.). Я рисовал в режиме **COMPONENT** , поэтому линии и надписи соответствуют этому стилю (коричневые). Если бы я выбрал, например, **ACTUATOR** - получил бы красные, **WIRE** - темно-зеленые и .т.п. (примеры при выборе отображаются выше в сером окошке). Даже нарисованный элемент я в любой момент могу изменить, щелкнув по нему дважды левой лапкой мышки в режиме **Selection Mode** (Стрелка в меню слева) или через правую лапку мышки **Edit Properties** . Можно просто изменить **Global Style** , выбрав другой через развертывающийся список, а можно конкретно изменить что-то снимая галочки На картинке ниже я просто изменил стиль изображения **PIN** на штрих-пунктирный и цвет на желтый. Щелкаем кнопку[/b] This Graphics Only [/b] (Только для этого графического элемента). Кроме того для инверсного вывода 13 176ИЕ12 я применил стиль **INVERT** из **Device Pin Mode** (рисунок выше), а для того чтобы получить **Z** с верхним надчеркиванием в его свойствах ввел **Pin Name** , ограниченное знаком доллара с двух сторон: **Z**.



6.3. Получаем полноправную No Simulator модель для ARES.

Модель 176ИЕ12, которую мы получили, абсолютно бесполезна с точки зрения разработчика. Ведь мы не можем использовать ее в симуляции, поскольку в библиотеке напротив нашей модели стоит **No Simulator** , да и корпус в **PSB Preview** отсутствует. Пришла пора добавить хотя бы корпус, чтобы можно было ее использовать в **ARES** при создании печатных плат. Выделяем нашу микросхему в проекте, щелкнув по ней один раз левой лапкой, или открываем контекстное меню через правую лапку мышки. Ищем гаечный ключ: **Packaging Tool** (он везде рядом с **Make Device**), выбираем. При первом вызове всплывет еще напоминание о том, что вы можете изменить размеры окна **Packaging Tool** ОКеим его, а если Вам не надо больше напоминать об этом дополнительно ставим в нем галочку. В открывшемся окне **Package Device** у нас пока пусто. Жмем кнопку **Add** (добавить). В следующем появившемся окне выбираем **Category => Integrated Circuit** (интегральные схемы); **Type => Through Hole** (сквозь отверстия); **Sub-Category => Dual in Line** (в два ряда); **Results => DIL16** . Почему именно **DIL** ? Потому что **DIP16** у Протеуса относится к типу **Surface Mount** (односторонний монтаж) и не содержит отверстий. Но любители ставить микросхемы со стороны дорожек могут добавить его как второй вариант. Именно поэтому я не воспользовался поиском по ключевому слову **Keywords** в самом верхнем окне. Можно было сразу набрать там **DIP** , и Протеус нашел бы в своих библиотеках все корпуса, имеющие такое название. Жмем **ОК**, возвращаемся в предыдущее окно, где уже имеем корпус. Если надо еще один (тот же **DIP**) жмем опять **Add** . Но мне достаточно корпуса под "дырявый монтаж" и я просто жму **Assign Package(s)**. После этого Протеус предложит сохранить изменения в библиотеке **USERDVC** - подтверждаем. Вот и все, модель для **Ares** готова. Если бы в библиотеках Протеуса отсутствовал нужный нам корпус **PSB** , его бы предварительно надо было создать в **Ares** и сохранить в библиотеке корпусов, но это уже отдельная тема. При создании модели какого-либо разъема на этом можно было остановиться, но мы идем дальше к "оживлению" нашей модели.

6.4. Поближе знакомимся с вкладками Make Device

Первая вкладка **Device Properties** :

Ну с **Device Name** и **Reference Prefix** вроде можно догадаться из предыдущих наших действий. Если бы мы не оставили в скрипте те четыре строчки, то здесь нам бы пришлось вручную добавить Имя и Префикс с которым компонент будет вставляться в проект. Если в Префиксе пусто, то компонент не будет включаться в общий список компонентов проекта и соответственно передаваться в **ARES** . Это удобно для виртуальных приборов и блок-схем. В окно **External Module** помещается Имя внешнего файла, который вы хотите прикрепить к компоненту, когда он помещается в проект. Секция **Active Component Properties** относится к анимированным компонентам (индикаторы, кнопки и т.п.). Ее мы разберем подробно при создании собственного индикатора.

Вторая вкладка **Packaging** :

Ну она нам тоже теперь знакома (см. чуть выше). Здесь мы могли бы добавить корпус еще на этапе создания графического компонента, но я предпочитаю сразу не делать этого, т.к. те же скрытые ноги питания прикрепляются намертво и потом приходится заново проводить "мэйкование", если что-то не так прилепил.

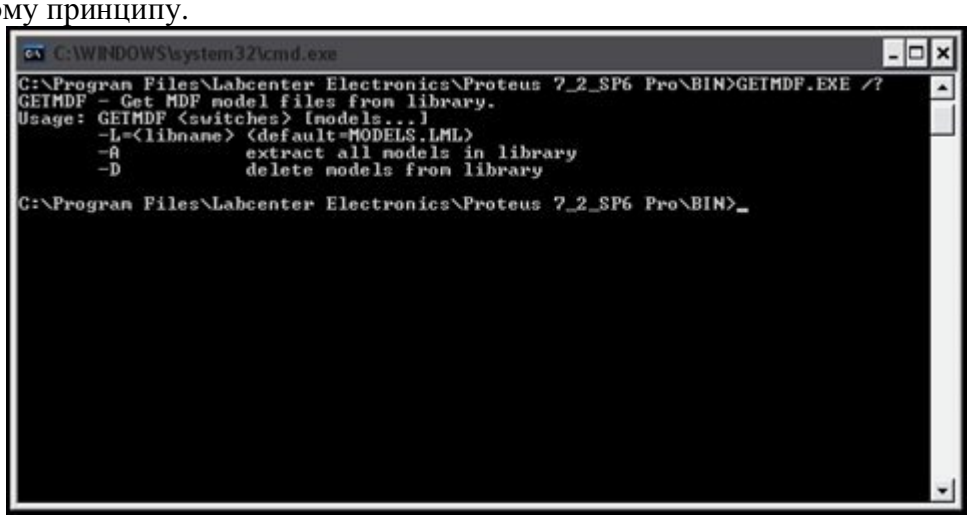
Третья вкладка **Component Properties & Definitions** :

На этой вкладке прописываются те свойства компонента, которые будут доступны при вызове окна **Edit Properties** при двойном щелчке по нему или через правую лапку мышки. Это могут быть стандартные , которые доступны через всплывающее меню кнопки **New** или собственные (после нажатия кнопки **New** выбирается **Blank Item**). При этом каждому свойству назначаются следующие определения: **Name** - имя, **Description** - описание, **Type** - тип значения (строка, целое число, Булево определение и т.п.), **Type** - тип отображения

(нормально, скрытое, только для чтения), а также значение по умолчанию (**Default Value**) и видимость (**Visibility**). Для стандартных свойств часть этих значений уже предопределена. При добавлении нового вы сначала выбираете его из всплывающего **New** , затем редактируете и добавляете его той же **New** . Если добавили лишнее можно удалить **Delete** .
Четвертая вкладка **Device Data Sheet & Help File** :
Здесь, как нетрудно догадаться, прописываются пути (сетевые и локальные, в т.ч. и на CD диске) к файлам даташита (технических данных компонента) и HELP, если таковые имеются. Для создания собственных компонентов эта закладка полезна только в том случае, если вы имеете PDF даташит компонента или умеете компилить свои хелпы CHM.
И, наконец, пятая последняя вкладка **Indexing and Library Selection** :
На ней мы определяем: как будет классифицироваться наш компонент в библиотеках Протеуса, и в какую библиотеку он будет помещен. Отмечу, что по умолчанию в Протеусе доступна для сохранения своих компонентов только библиотека **USRDVC** . Если она вас не устраивает, то необходимо заранее создать новую через верхнее меню: **Library => Library Manager =>** кнопка **Create Library** и далее набрать имя файла библиотеки (например: CMOS_176). После этого файл будет доступен в правом окне пятой вкладки. Что еще здесь прописывается: **Device Category** - категория компонентов в которой он будет показываться при выборе по **Pick Device. Device Sub-category** - принадлежность компонента к какому либо типу устройств (счетчики, дешифраторы и т.п.). **Device Manufacturer** - производитель компонента. **Device Description** - назначение компонента (например: counter for clock with dynamic indication - счетчик для часов с динамической индикацией). Device Notes - описание компонента. Галочка **Advanced Mode** убирает режим расширенного редактирования, поэтому ей лучше не пользоваться.

6.5. Утилиты Протеуса для работы с библиотеками.

Прежде чем мы начнем создание своей схематической модели 176ИЕ12, познакомимся с четырмя утилитами, входящими в состав Протеус. Две из них предназначены для работы с MDF/LML: **GETMDF.EXE** и **PUTMDF.EXE** . Аналогично для SDF/SML: **GETSPICE.EXE** и **PUTSPICE.EXE** . Расположены они в папке **BIN** . Соответственно те, которые GET извлекают, а PUT - вставляют файлы в соответствующие библиотеки. Обе утилиты автономны и работают в командной строке (т.е. в окне DOS). Чтобы получить помощь по использованию, как и для любых DOS-овских EXE вводим в командной строке полное имя программы и через пробел и слэш со знаком вопроса (например: **GETMDF.EXE /?**). На картинке вариант ответа программы.
Поскольку всю эту возню я затеял для того, чтобы добраться до файла **4060.MDF** - прототип нашей 176ИЕ12, попробуем GETMDF в действии. Остальные утилиты работают по аналогичному принципу.



6.6. Извлечение требуемого файла MDF из библиотеки LML.

Библиотек **LML** в папке **MODELS** не так уж и много. Логично предположить, что наша модель 4060, относящаяся к цифровым микросхемам, находится в библиотеке **Digital.lml** . Скопируем эту библиотеку и утилиту в отдельную папку, которую расположим в корневом каталоге диска, чтобы не набирать длинный путь в командной строке. Открываем через **Пуск => Все программы => Стандартные => Командная строка** окно MS DOS (у меня файлы расположены на диске **C:** в папке **ExMDF**) переходим в нужный нам каталог:
C: (чтобы попасть на диск С, если окно открылось с другим диском)
cd ExMDF (чтобы попасть в нашу папку)
(в следующей строкеперед каждым ключом начинающимся с дефиса пробел обязателен!)
GETMDF.EXE -L=Digital.lml -A
Поясню действие последней строки: извлечь все файлы **MDF** из библиотеки **Digital.lml** . В результате этого в нашей папке появится огромное количество файлов **mdf** , среди которых и нужный нам **4060.mdf** . Ненужное можно удалить, а этот файл нам потребуется для дальнейшего исследования.
Те, кто использует **Total Commander** , могут слегка облегчить себе жизнь. Во-первых, в коммандере легко можно вызвать командную консоль из верхнего меню, находясь уже в нужной папке, т.е. не надо "ручками" переходить в нужное место. Во-вторых в последних версиях в качестве текстового редактора по **F4** используется **AkePad** , которым можно заранее открыть библиотеку **LML** и убедиться, что нужная нам модель там присутствует. Достаточно просто запустить поиск нужного текста (например: 4060). После нескольких **Найти далее** редактор найдет текст: **Design: K:\PROLIBS\DIGITAL\CMOS\4060.DSN** - это начало нужного нам файла **mdf** в библиотеке.

6.7. Что внутри файла MDF?

Получив **4060.mdf** , откроем его спомощью стадартного блокнота **Notepad** . Давайте рассмотрим его структуру. Шапка файла содержит информацию об исходнике, авторе (в которой Лабцентр о себе почему-то скромно умалчивает), дате создания и модификации файла. Это все мало интересно. Далее идут более интересные сведения (текст из файла и мои пояснения):

```
*PROPERTIES,1
CLOCK=EXTERNAL
```
Одно свойство, тактовый генератор внешний.

```
*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
```
Две карты первая устанавливает задержки для трех разных напряжений питания, вторая - два типа параметров тактового генератора.

```
*MODELDEFS,0
```
Раз 0 значит отсутствует

```
*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>
```
Полный список используемых в модели примитивов (я привел не весь), с указанием назначаемых параметров для каждого.

```
*NETLIST,35
#00000,3
U1,OP,Q
U2,IP,D1
U4,IP,D1

#00001,2
U4,OP,Q
U11,IP,CLK

#00002,3
U5,IP,CLK
U13,IP,D
U13,OP,!Q
```

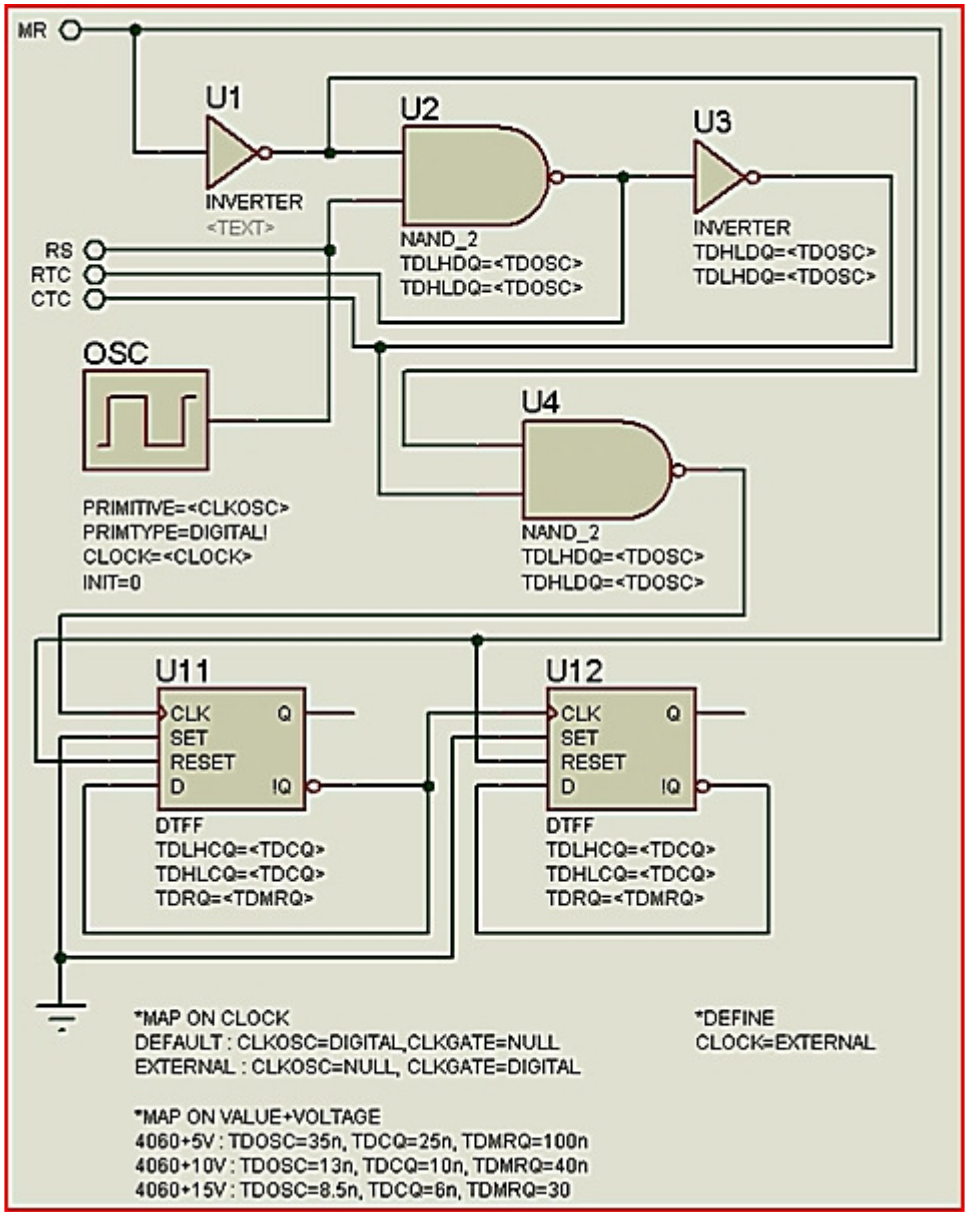
Коммутационный список цепей. Всего их 35. Я привел первые три. Кратенько поясню на примере одной что это:
#00002,3 -номер цепи 2 количество присоединений 3, далее что с чем:
U5,IP,CLK - элемент U5 - примитив D-триггера, IP (вход от англ.INPUT) CLK -обозначение
U13,IP,D - элемент U13 - примитив D-триггера, IP (вход от англ.INPUT) D -обозначение
U13,OP,!Q - элемент U13 - примитив D-триггера, OP (выход от англ.OUTPUT) !Q - инверсный
И так далее по всем пунктам мы можем полностью воссоздать исходник дочернего листа модели 4060.

6.8. Пробуем воссоздать внутреннюю схему из MDF.

Из полученного файла **4060.MDF** попробуем частично воссоздать внутреннюю структуру (дочерний лист) модели счетчика 4060. В этом нам помогут **PARTLIST** и **NETLIST** из файла **MDF** . В соответствии с первым набираем примитивы из библиотеки (генератор - **CLOCK** , инвертер - **INVERTER** , элемент 2И-НЕ - **NAND_2** , D-триггер - **DTFF**). В соответствии со вторым производим соединения набранных элементов. Я не стану здесь приводить полную структуру модели 4060, поскольку она достаточно громоздкая. Нас интересует в первую очередь воссоздание встроенного генератора и передача сигналов с него на счетчик из D-триггеров. Вот этот кусок дочернего листа я и прилагаю на картинке. К сожалению, мои продолжительные мучения с генератором **OSC** из файла **MDF** к желаемым результатам не привели. По иронии судьбы я умудрился выбрать единственную модель в библиотеке содержащую такой генератор. Управление им идет через два скрипта **MAP** и **PROPERTIES** . Реально на дочернем листе скрипт **MAP** должен выглядеть следующим образом:

```
*MAP ON CLOCK
DEFAULT : CLKOSC=DIGITAL,CLKGATE=NULL
```

EXTERNAL : CLKOSC=NULL, CLKGATE=DIGITAL
а скрипт **PROPERTIES** вот так:
***DEFINE**
CLOCK=EXTERNAL
Но на деле эта связка работать отказывается. Дело в том, что параметр **CLOCK** примитива генератора имеет значение **PNZ - POSITIVE, NOT ZERO** (положительное, не ноль). Если декомпознуть 4060, то там **POZ - POSITIVE, OR ZERO** (положительное, или ноль). Могу только предположить, что тот,кто делал модель 4060 использовал не этот стандартный **CLOCK** -генератор из примитивов, а свой модифицированный. А жаль, ведь в модели 4060 работают оба варианта: и внешние RC-цепи и, если задать параметр **Oscillator Frequency** (например 32768), то встроенный в модель. Ну чтож, отнесу пока это в область неразрешенных задач. А мы продвинемся дальше, т.к. пауза в создании модели из-за этого генератора слишком затянулась. Кстати в версии Протеуса 7.4 можно попробовать реализовать его через скрипт **HDL** .



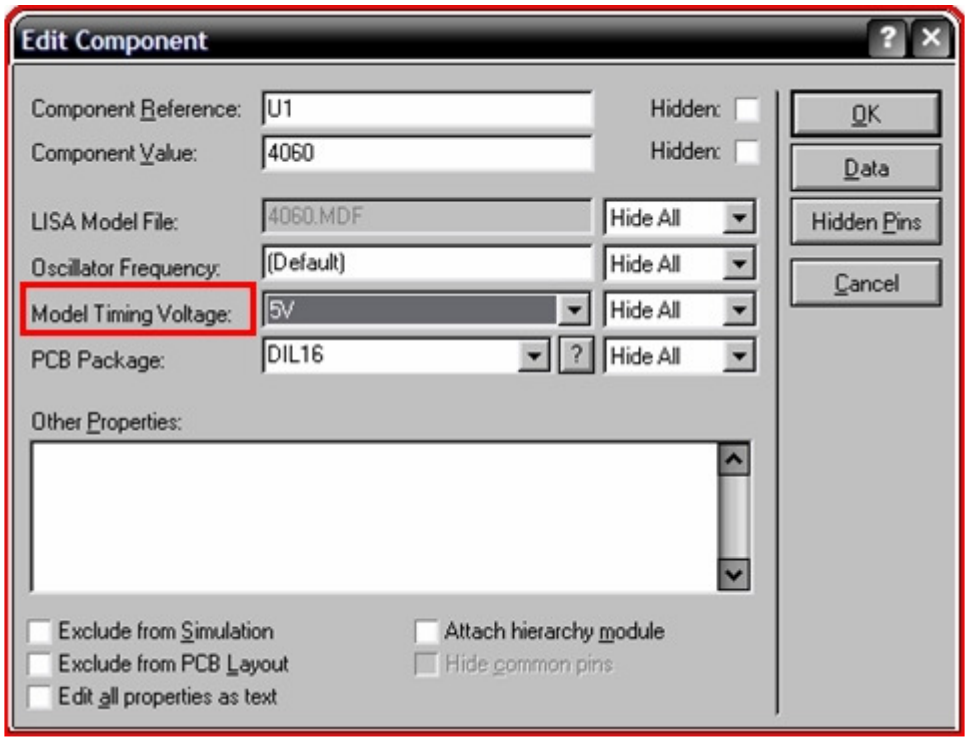
6.9. Подробнее о картах MAP и свойствах примитивов.

Как видно из предыдущей иллюстрации в модели применены две карты. Первая задает поведение внутреннего генератора. Вторая задает временные параметры фронтов в зависимости от выбранного напряжения. Обратите внимание, что в примитивах я в соответствии с **PARTLIST** в режиме редактирования свойств вставил строки типа **TDHLDQ=<TDOSC>**. Вот они то и задают длительности фронтов и спадов в зависимости от выбранного напряжения. То есть, если вы откроете свойства 4060, там этот выбор присутствует, и если вы там меняете вольтаж, изменяется не питающее напряжение микросхемы, а именно длительности фронтов при симуляции! Это свойство присутствует во всей CMOS серии в библиотеках Протеуса.

Что касается фронтов импульсов для примитивов, из которых смоделированы все цифровые логические микросхемы в Протеусе, то достаточно посмотреть хелп на соответствующий примитив - какое свойство за что отвечает, но можно запомнить и так:

TDLHDQ - low-high (переход с низкого на высокий) - передний фронт;

TDHLDQ - high-low (переход с высокого на низкий) - задний фронт.



. Создание схематичной модели счетчика К176ИЕ12.

7.1. Предпосылки и ограничения для создания 176ИЕ12.

Итак, первоначально мне хотелось создать универсальную модель, по аналогии с 4060, но после длительной и безуспешной борьбы со встроенным в нее генератором, придется упростить задачу. Поэтому для начала создадим модель, работающую только от внутреннего генератора. Внешние цепи, к которым в идеале подключается кварцевый резонатор, сохраним только для использования в **PCB** , т.е. в **ARES** . Учитывая, что модель кварца в Протеусе достаточно убога и ограничена, а 176ИЕ12 в основном используется с резонатором 32,768 кГц для получения точных временных интервалов для часов то это не существенное ограничение. Тем более, мы сохраним в свойствах возможность изменения частоты для получения других временных интервалов. Некоторые характеристики указанной микросхемы можно посмотреть например здесь:
http://lib.qrz.ru/?q=node/5376.
А вот здесь:
http://lib.qrz.ru/?q=node/5480
приведена схема RC генератора на 176ИЕ12. В общем то, все это взято из старых журналов Радио, но в Инете эту информацию оказалось найти проще, чем журналы почти тридцатилетней давности. Приведу заимствованную оттуда картинку структуры микросхемы. Желающие могут попробовать создать модель с внешними цепями (слева на картинке).

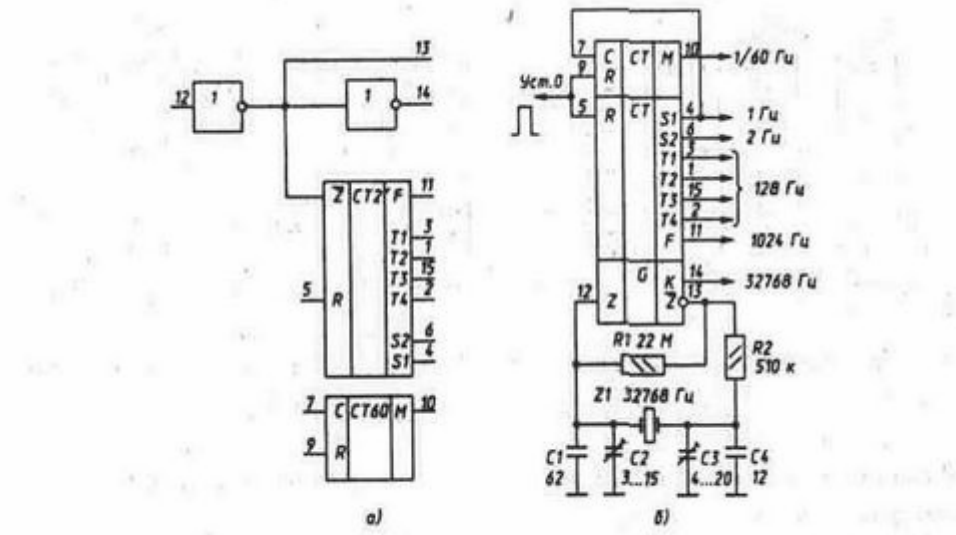


Рис. 203. Структура (а) и типовая схема включения (б) микросхемы К176ИЕ12

7.2. Временные диаграммы и питание 176ИЕ12.

В соответствии со следующей картинкой нам предстоит смоделировать внутреннюю структуру счетчиков **176ИЕ12**. Если проанализировать диаграмму, то мы видим, что необходимы выходы с частотой 1024 Гц (звук на будильник), четырехтактный генератор 128 Гц для динамической индикации, выходы секундных и полусекундных импульсов S1 и S2, достаточно хитрый сигнал минутного импульса - **M** . Кроме того, имеется выход K с тактовой частотой 32768 Гц (на диаграмме почему-то упущен).

Теперь обратимся к электропитанию. Стандартное напряжение для 176 серии +9V. 176-я серия достаточно "тихоходная", по справочным данным параметр задержки передачи импульса при напряжении составляет по разным данным около 250нсек. Предельная частота для 176ИЕ12 составляет 1,2 МГц. Это все сведения которые мне удалось найти по этой микросхеме. Теперь попробуем создать основные внутренние функциональные узлы 176ИЕ12.

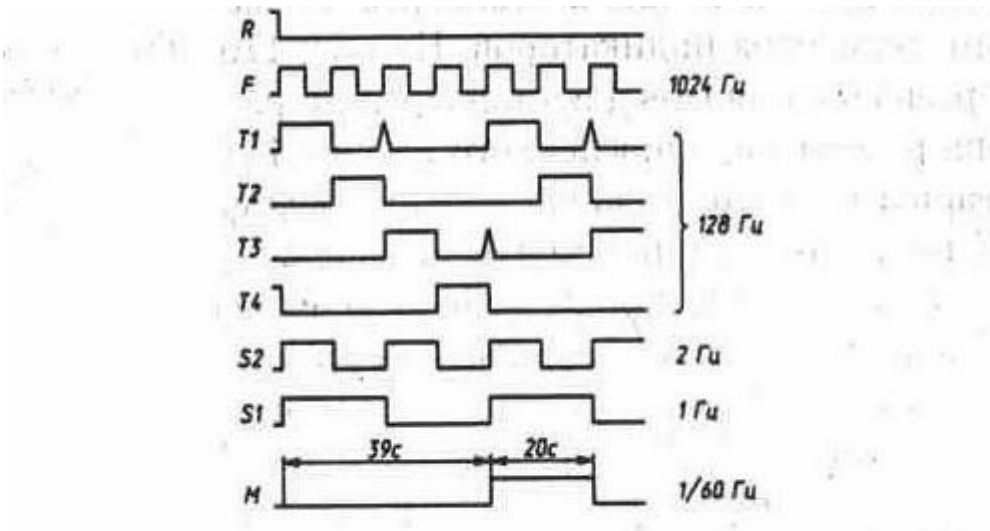
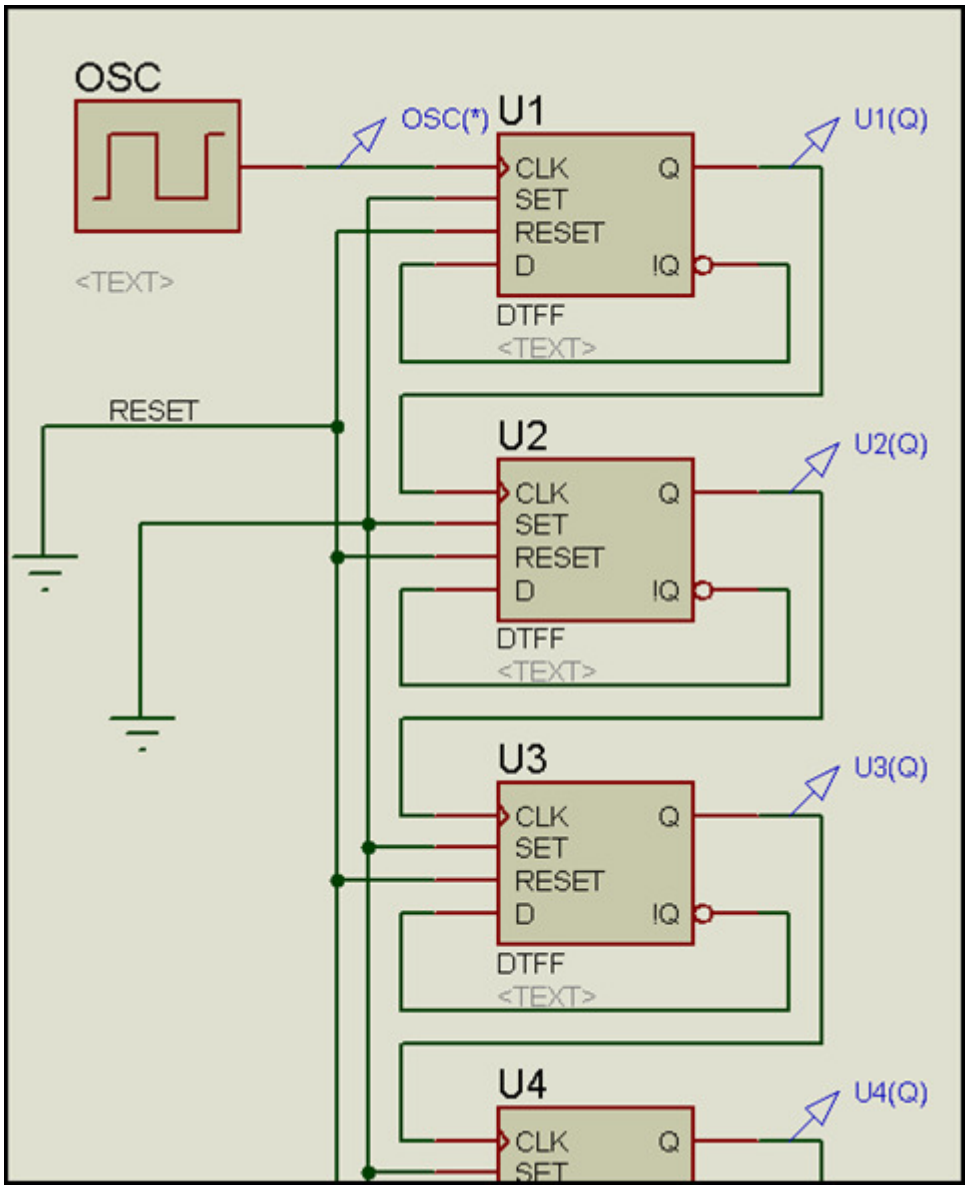


Рис. 204. Временная диаграмма работы микросхемы К176ИЕ12

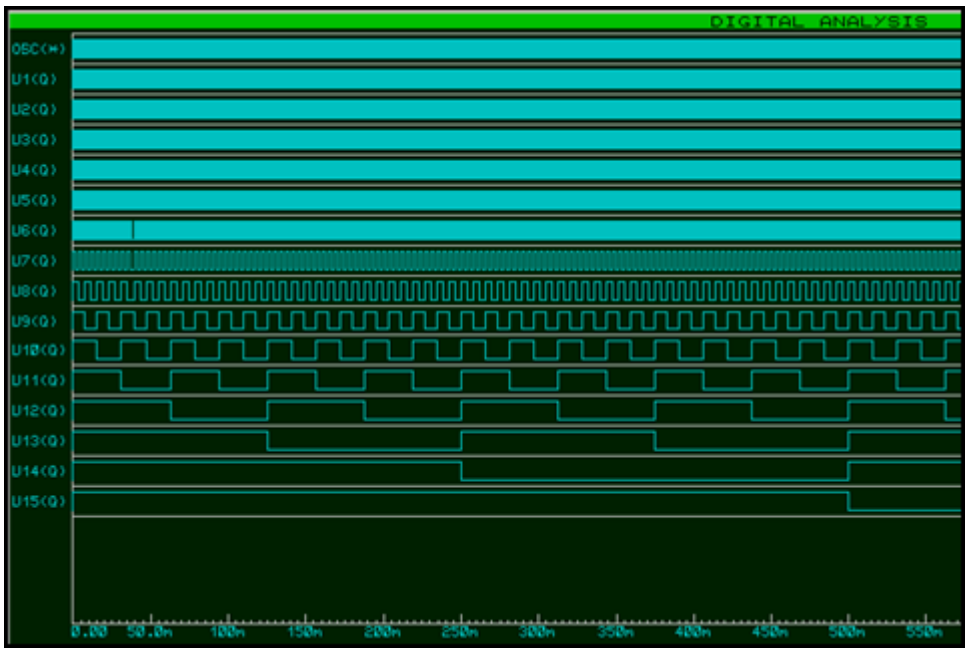
7.3. 15-ти разрядный счетчик-делитель.

Поскольку мы условились, что оставляем только внутренний генератор, то первое , что нам необходимо создать - это счетчик делитель на 15 рарядов для получения частоты 1 Гц из 32769 Гц (стандартный часовой кварц обычно подключаемый к этой микросхеме). Для создания счетчика воспользуемся аналогией с 4060 и наберем его из 15-ти триггеров **DTFF** . Использовать для этой цели готовые универсальные многоразрядные примитивы счетчиков не стоит, поскольку они имеют встроенную логику параллельного переноса и у нас не совпадут временные параметры. Я приведу на картинке только фрагмент, который показывает внутреннюю структуру, т.к. полная картинка не влезет на страничку или получится слишком мелкой. Остальные элементы соединяются аналогично. Частоту генератора **OSC** задаем 32768 Hz.



7.4. Тест 15-разрядного счетчика.

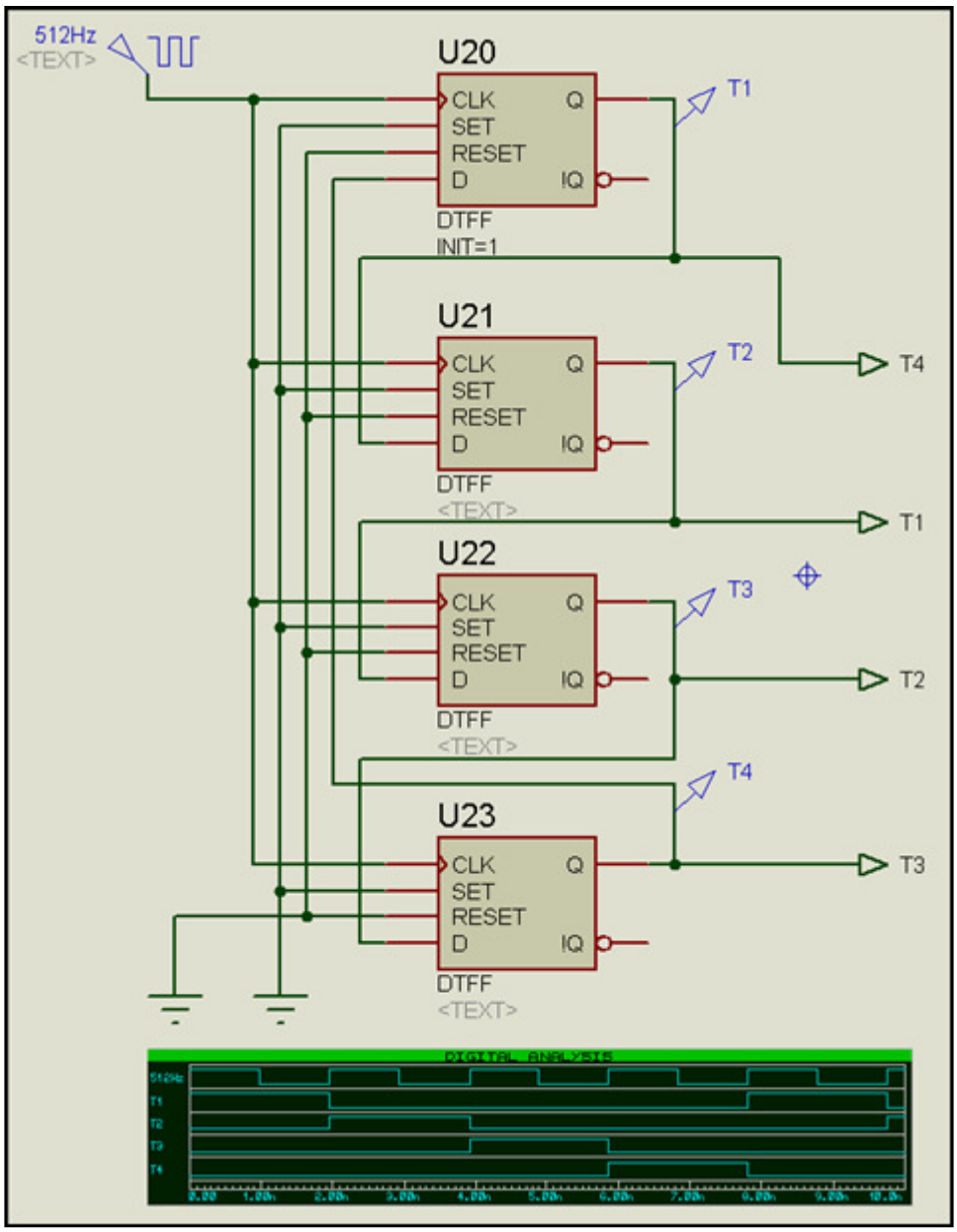
Для тестирования добавим цифровой (**DIGITAL**) граф и на него поместим все пробники с генератора и выходов триггеров. Запускаем пробелом граф и получаем соответствующую картинку. Как видим, на выходе триггера **U15** мы имеем импульсы с периодом 1 сек, что и необходимо было получить. Пока сохраняем этот проект и начинаем новый.



7.5. Четырехтактный формирователь сигнала динамической индикации 128 Гц.

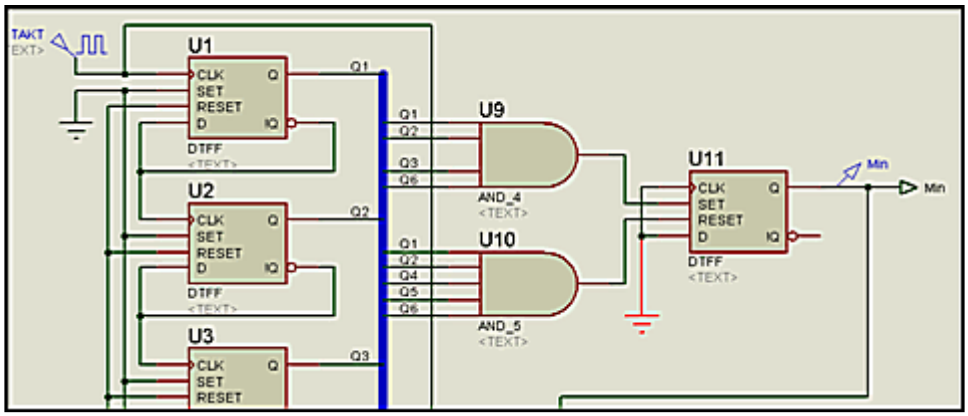
Реализуем его также на триггерах **DTFF** , но соединенных уже по схеме кольцевого регистра сдвига. Набираем четыре триггера, добавляем на вход генератор для тестирования и, как и раньше, проверяем через цифровой граф. Обратите внимание, что частота входного генератора для получения 128 Гц должна быть в четыре раза выше, т.е. 512 Гц. Кроме того, для первого триггера я использовал предустановку в единичное состояние: в свойствах в окне **Other Properties** введена строка **INT=1** . В цифровом графе параметр **Stop Time** установлен **10m** (т.е.10мсек). После тестирования можно приделать его через обычный метод копи-паст к 15-разрядному счетчику. Соответственно убираем из него генератор 512 Гц и объединяем входы **RESET** всех триггеров (на земле они у нас висят временно для тестирования). Вместо генератора приделываем вход **CLK** формирователя к прямому выходу триггера **U6** 15-разрядного счетчика (если вы нумеровали их последовательно, то именно там будет частота 512 Гц).

Нумерацию элементов в формирователе можно при желании изменить, чтобы продолжить последовательность из счетчика.



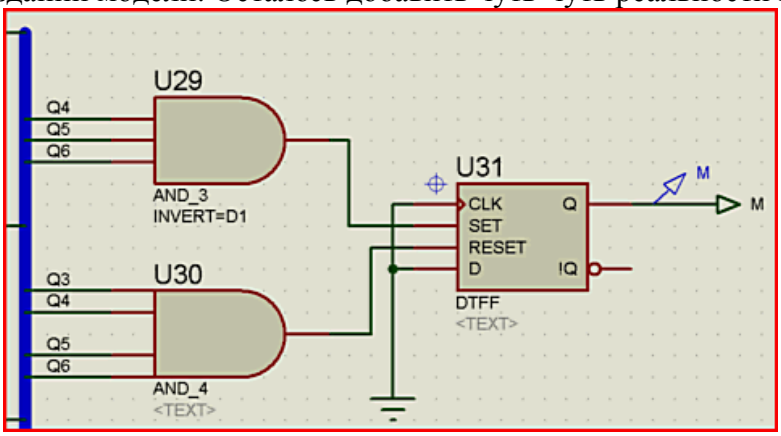
7.6. Формирователь минутного импульса.

Пожалуй, это наиболее сложная часть нашей модели, поскольку передний фронт этого импульса должен быть по прошествии 39 сек, а его длительность - 20 сек, т.е. после 59-й сек импульс должен быть сброшен в ноль. Поэтому также для начала собираем эту часть в новом проекте, но генератору для начала зададим частоту повыше и кратную 60, чтобы быстро измерить виртуальным частотомером при тестировании. Не будем мудрствовать и соберем эту часть все на тех же D-триггерах по схеме суммирующего счетчика. Десятичное число 60 эквивалентно двоичному **111100** , т.е. потребуется 6 триггеров. Кроме того, придется входы RESET раздвоить по 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 триггер DTFF, но уже как обычный RS-триггер. На вход S собираем через логический элемент И число 39 (двоичное **100111**), на вход R - число 59 (двоичное **111011**). В нашем случае это возможно, 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 логических 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 пяти-входового элемента И будет **D0** , а нижний **D4**). Поскольку схема достаточно громоздкая я приведу на картинке лишь ее фрагмент, а во вложенном файле проект в версии 7.2.SP6 для тестирования. Обратите внимание, что для логического элемента **U7** (формирователь сброса по достижении числа 60) установлены свойства **TDLHDQ=100n** и **TDHLDQ=100n** (нарастание и спад импульса по 100 наносекунд) иначе вы просто не увидите его на графе. Если запустить симуляцию, 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.



7.7. Полная внутренняя структура 176ИЕ12. Коррекция после теста.

Теперь собираем все что мы уже создали в единый проект: корректируем нумерацию элементов, добавляем недостающие связи и тестируем в полном объеме с помощью все тех же цифровых графиков. Конечно, поместить полную картинку здесь не удастся, поэтому в аттаче проект для версии **7.2.SP6** . В проекте два файла: **Vers1.dsn** и **Vers2fin.dsn** . Первый вариант - это то, что получилось после сборки. Однако, при тестировании выяснилось, что фронты минутного импульса сместились на одну секунду назад (т.е. 38-я и 58-я сек. Это видно если максимизировать нижний граф). Пришлось слегка подкорректировать формирователь минутного импульса (см. прилагаемую картинку). Передний фронт - установка триггера **U31** по двоичному **101000** (дес. 40), сброс по **111100** (дес. 60),. Обратите внимание, что для установки пришлось применить элемент **3-И** , с инверсией по второму сверху входу **(INVERT=D1)**. Это необходимо, поскольку иначе будет отсутствовать сброс минутного импульса (три старших разряда отличаются только нулем в середине). Кроме того, я уже заранее завесил на землю все входы, которые мы не будем симулировать. После тщательного тестирования уже можно сохранить последний вариант как секцию: **File=>Export Section** . Секцию потом удобнее добавлять на дочерний лист при окончательном создании модели. Осталось добавить чуть-чуть реальности и модель готова.



7.8. Добавляем реальные свойства к модели.

Сначала создадим проект для окончательного тестирования модели нашего счетчика по аналогии с помещенным в **\SAMPLES\Tutorials\ Dmodtut1.DSN** . Итак создаем новый проект. Помещаем на первый лист созданную ранее графическую модель 176ИЕ12. Входим в ее свойства двойным щелчком левой лапкой хвостатой и… ставим галку в окошечке **Attach hierarchy module** . Все, теперь нам доступен **Child Sheet** через правую кнопку мышки для нашей модели. "Goto -ем" туда и втаскиваем через **File => Import Section** нашу ранее сохраненную секцию. Правим связи терминалов на дочернем листе в соответствии с названиями ног модели на основном. Кроме того, нам придется добавить еще пару терминальчиков питания с лэйблами **VSS** и **VDD** , первый из которых заземляем, а второй просто соединяем с терминалом питания без названия (по умолчанию). Если этого не сделать при симуляции Протеус начинает грязно ругаться желтыми сообщениями об отсутствии данных связей на дочернем листе.

Получившаяся структура счетчика 176ИЕ12 вполне работоспособна и уже может быть использована в качестве схематической модели. Но давайте заглянем в **ПУСК => Все программы => Proteus 7 2 SP6 (или какой у Вас) => Proteus VSM Model Help => ProSPICE Primitives** и посмотрим там раздел **Digital Modelling Primitives** или просто щелкнув по любому примитиву в нашем проекте откроем **HELP** конкретно по нему. Ведь именно из этих примитивов мы собирали нашу модель. Оказалось, что по умолчанию (**Default**) параметры фронтов **TDLHDQ, TDHLDQ** и задержки прохождения **TGQ** равны нулю. Меня, да и Вас это конечно не радует. Ведь реальная КМОП микросхема имеет свои

параметры. Будем исправлять по аналогии с уже разобранным ранее счетчиком 4060. Для этого в свойствах каждого триггера в нашей структуре в окошке **Other Properties** добавим строки:

TDHLCQ=<TDCQ>
TDLHCQ=<TDCQ>
TDRQ=<TDMRQ>

Для логических элементов аналогично добавим строки:

TDHLDQ=<TDOSC>
TDLHDQ=<TDOSC>

Добавим на дочерний лист текстовый скрипт следующего содержания:

***MAP ON VALUE+VOLTAGE**
176IE12+5V : TDOSC=400n, TDCQ=500n, TDMRQ=500n
176IE12+9V : TDOSC=200n, TDCQ=250n, TDMRQ=250n

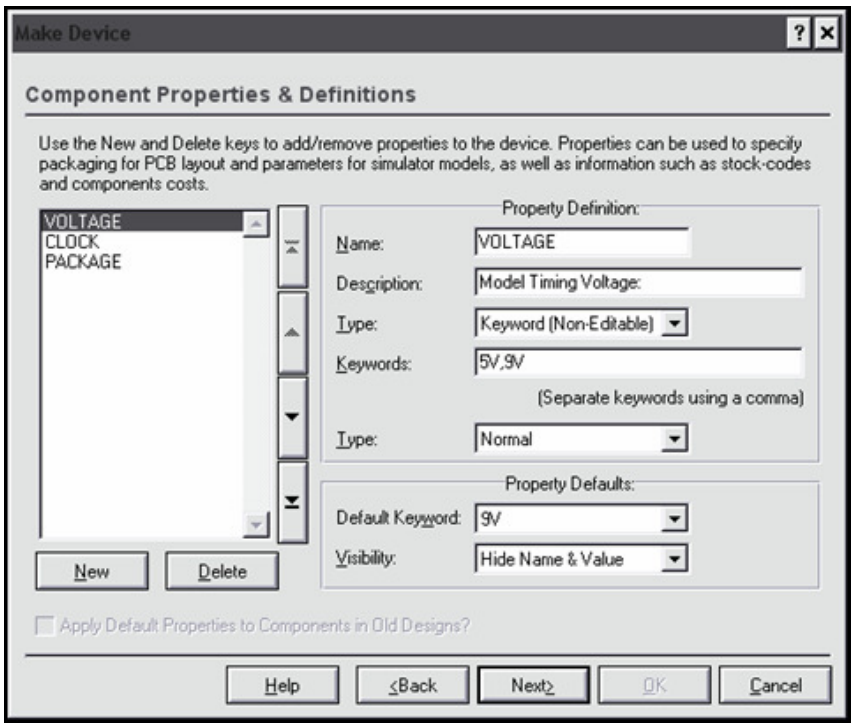
И еще один:

***DEFINE**
TGQ=?

Для простой логики я принял задержки 200 нсек, для триггеров 250 нсек. Здесь я не гарантирую полной точности, особенно для напряжения 5V, т.к. взял и увеличил времена грубо вдвое по аналогии с 561-й серией, для которой это справедливо. Но какое-то реальное сходство с физической микросхемой уже будет просматриваться.

Для того,чтобы это свойство "заработало" возвращаемся на основной лист и снова "мэйкаем" нашу модель. При этом ничего не меняем кроме добавления (через кнопку **New => Blank Item**)следующего свойства как на картинке.

Успешно пройдя по всем вкладкам, в конце соглашаемся перезаписать существующий девайс. Теперь у нас в Пропертях появилось свойство **Model Timing Voltage** , которое по умолчанию равно 9V (см. на картинке **Default Keyword**) и мы можем выбирать задержки для напряжения 5V. Можно попробовать симуляцию и двигаться дальше.



7.9. Добавляем свойство Clock Frequency (частота генератора) и компилируем модель.

Итак последний штрих - частота генератора. На дочернем листе в свойствах генератора OSC ставим галку **Edit all properties as text** и правим строчку:

CLOCK=<CLOCK>

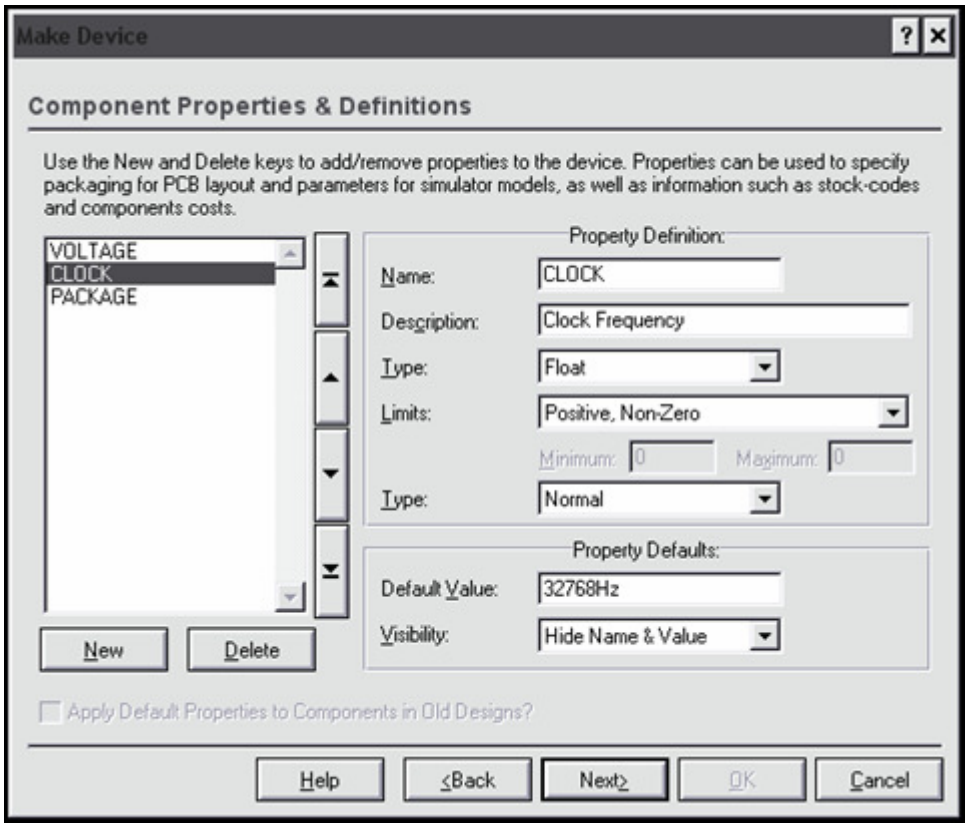
Далее опять уходим на основной лист и снова "мэйкаем" нашу модель. По аналогии с предыдущим создаем свойство **CLOCK**

Только теперь мы выбрали тип - с плавающей точкой; ограничение - положительное, не равно нулю; и значение по умолчанию 32768 Гц. Вот собственно и все. Тестируем нашу модель, проверяем, что появилось свойство **Clock Frequency** , пробуем его менять и убеждаемся, что все работает.

Теперь во вкладке **Tools** верхнего меню выбираем опцию **Model Compiler…** и создаем с помощью ее **MDF** файл с названием нашей модели. По умолчанию Протеус предлагает сохранить его в своей папке **MODELS** .

Теперь осталось в последний раз "мэйкнуть" модель и по аналогии с предыдущими добавлениями приклеить через **New => MODFILE** (этот пункт уже существует как стандартный в раскрывающемся меню) наш созданный файл **176IE12.MDF** , прописав его в окошке **Default Value** , и процедура создания модели полностью закончена.

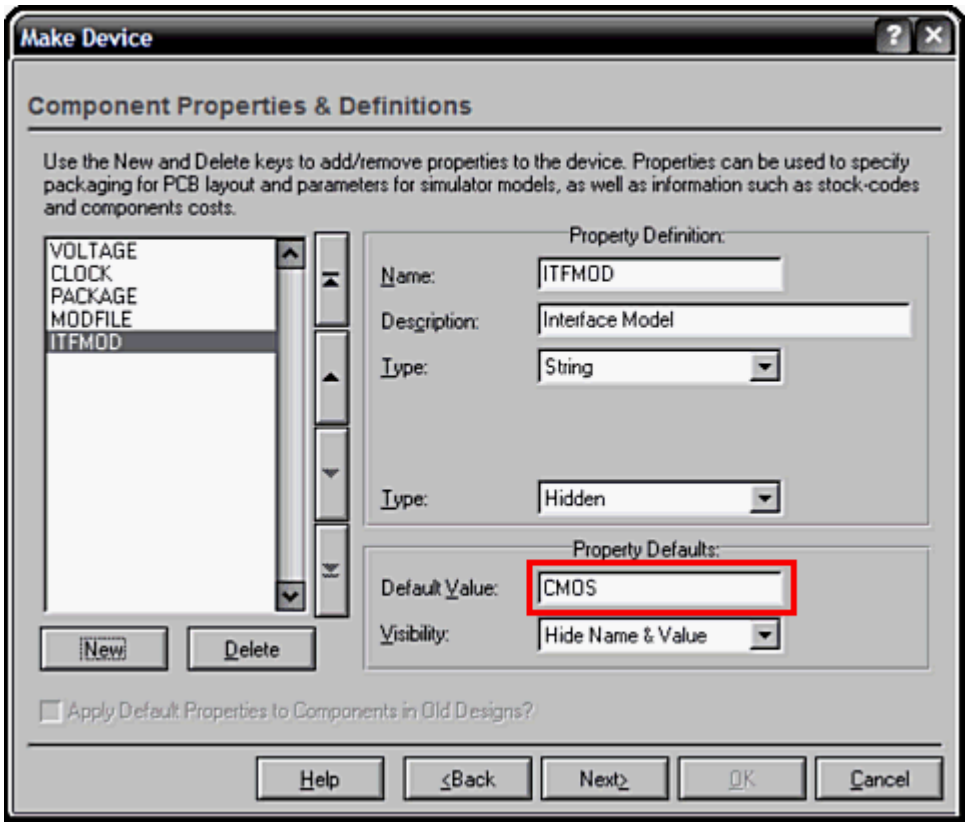
Если что-то непонятно, можно всегда посмотреть в любой модели цифровой микросхемы, например той же 4060. Так же щелкаем по ней " **Make Device** " и доходим до нужной вкладки. Главное в конце не сохранять изменения (не прозевать нажать Cancel), чтобы не испортить существующую модель.



7.10. Еще одно свойство для полного счастья.

В предыдущем пункте я слегка поторопился с компиляцией модели. Когда я в П.7.8 указал, что необходимы **VDD** и **VSS** на дочернем листе, модфайл MDF еще не создавался, а эти терминалы были необходимы для отладки. Поэтому после отладки их можно благополучно удалить, а 176 IE12 придется «мэйкнуть» еще раз. И все в том же окне **Component Properties & Definitions** добавить еще одно стандартное свойство из всплывающего меню кнопки New – **ITFMODE** (интерфейс модели). Ему достаточно просто по аналогии с 4060 указать значение по умолчанию **CMOS**. Вот тогда все наши VSS и VDD на дочернем листе становятся просто лишними. Если же скомпилировать MDF так, как я указал выше – модель работать будет, но при останове симуляции вылетает желтое сообщение: **Pin VDD not modeled**. Думаю лишняя головная боль от таких предупреждений симулятора никому не нужна.

В приложенном архиве два проекта: в одном (**V1_MDF.DSN**) – окончательный вариант модели еще с дочерним листом, с которого компилировался **176IE12.MDF**. В проекте **Test.DSN** дочерний лист уже недоступен, но требуется присутствие файла MDF. В нем же еще раз показано: как использовать микросхему со стандартным питанием 9V, не затрагивая при этом значение VDD=5V по умолчанию для всего проекта через **Configure Power Ralls**.

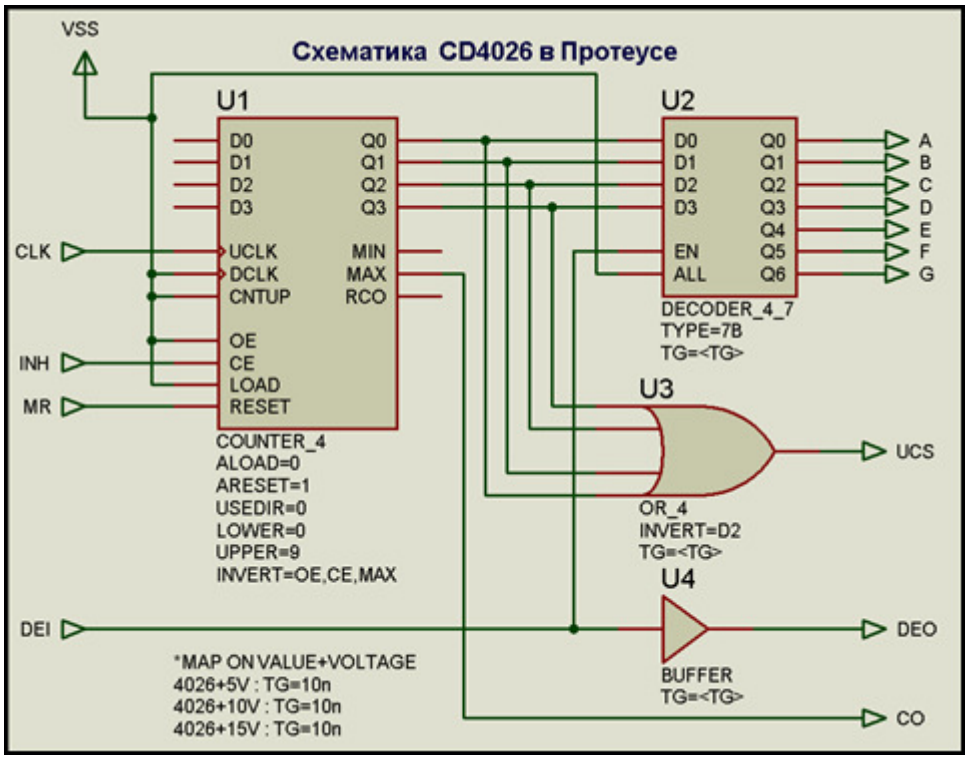


8. Создание моделей счетчиков 176ИЕ3 и 176ИЕ4.Создание своей библиотеки LML. Создание активной модели на примере вакуумно- люминисцентного индикатора.

8.1. Внутренняя структура исходной модели аналога CD4026.

Если досконально порыться в справочниках, то можно найти упоминание о том, что данные счетчики-дешифраторы созданы на основе счетчиков Джонсона. Для тех, кто подзабыл - напомню, что счетчик Джонсона представляет собой регистр сдвига на D-триггерах с замкнутой перекрестной связью с инверсного выхода последнего триггера на D-вход первого. У одного из этих счетчиков, а именно **176ИЕ4** имеется неполный аналог - **CD4026** , представленный в библиотеке Протеуса схематичной моделью.

Желающие посмотреть на действительную внутреннюю структуру **CD4026** могут скачать его даташит. Напомню, что для этого достаточно поместить его на лист проекта и при подключенном канале Интернет, выделив элемент через правую кнопку грызуна, выбрать опцию **Display Datasheet** , или просто воспользоваться сочетанием клавиш **Ctrl+D** . Файл модели **4026.MDF** , который понадобится для отечественных счетчиков, расположен в той же библиотеке, что и 4060 и добывается оттуда аналогично. Однако, если мы воспроизведем по нему структуру нас ждет неожиданный сюрприз. Дело в том, что ребята из Лабцентра подошли к процессу творчески и не стали воспроизводить внутреннюю структуру "дословно", а просто из готовых примитивов собрали свой функциональный аналог счетчика-дешифратора **CD4026** . И выглядит он намного компактнее и проще, чем реальная структура в даташите с дешифратором на логических элементах. Сравните картинку с той, что на странице 5 даташита.Даташит не такой уж и большой - приатачил на всякий случай.



8.2. Немного о свойствах примитивов модели 4026, и что предстоит изменить и дополнить, чтобы получить 176ИЕ4.

В приведенной выше структуре использованы новые для нас примитивы **COUNTER_4** и **DECODER_4_7** . Полностью список их свойств доступен на аглицком в **HELP** . Для тех кто "тормозит" с языком остановлюсь на важных для данного случая.

Универсальный счетчик:

ALOAD=0 - асинхронная загрузка по параллельным D0…D4 запрещена

ARESET=1 -асинхронный сброс разрешен

USEDIR=0 - вход CNTUP не используется

LOWER=0 -нижний предел счета равен 0

UPPER=9 - верхний предел счета равен 9 (а не 10!!!, хотя счетчик декадный счет то с 0 зафиксируйте себе в уме!!!)

INVERT=OE,CE,MAX -это уже знакомо входы проинвертированы, т.е. активный сигнал для них 0.

Декодер:

TYPE=7B - тип декодирования в семисегментный код 6 и 9 с горизонтальными хвостами

TG=<TG> - задержка распространения со входа на выход определяется в MAP (почему такая сокращенная запись, ведь обычно TGQ непонятно, видно Протеус проглатывает и этот вариант)

Теперь посмотрим, что необходимо изменить для нашего родного счетчика. Во первых ИЕ4 переключается по спаду импульсов на тактовом входе, значит в **INVERT** через запятую надо добавить **UCLK** . Во вторых наш счетчик должен формировать дополнительный импульс (выход **UCS**) не по счету 2, а по счету 4 - изменяем для **U3** на **INVERT=D1** . В третьих, если посмотреть диаграмму, наш счетчик формирует положительные импульсы на выходах аналогичных **UCS** и **CO** (перенос) в отличие от импортного. Значит, в **U1** убираем из **INVERT MAX** , для **U3** добавляем через запятую **Q** (выход). В четвертых **INH, DEI, DEO** в нашем отсутствуют. Ликвидируем эти цепи и **U4** , входы **CE** и **EN U1** и **U2** завешиваем на **VSS** . Ну и, наконец, пятое и самое важное. **176ИЕ4** имеет вход **S** , который используется для изменения полярности выходного кода. В нашем случае для этой цели проще всего применить на выходе дешифратора двухвходовые исключающие ИЛИ, один из входов которых объединить и обозвать **S** . Вот и вся реконструкция. Счетчик **176ИЕ3** отличается от **ИЕ4** только коэффициентом пересчета - 6, и вместо четвертого импульса формирует второй. Ставим для **U1 UPPER=5** (!!! Предупреждал выше), а вот для **U3** становится актуальным **INVERT=D2** . Я создам две модели, но для "особо одаренных" Подскажу, что можно поместить эти параметры в **MAP** , зависимое от **VALUE** и там прописать разницу для двух счетчиков. Тогда можно дя них создать единый **MDF** .

Выглядеть это будет примерно так:

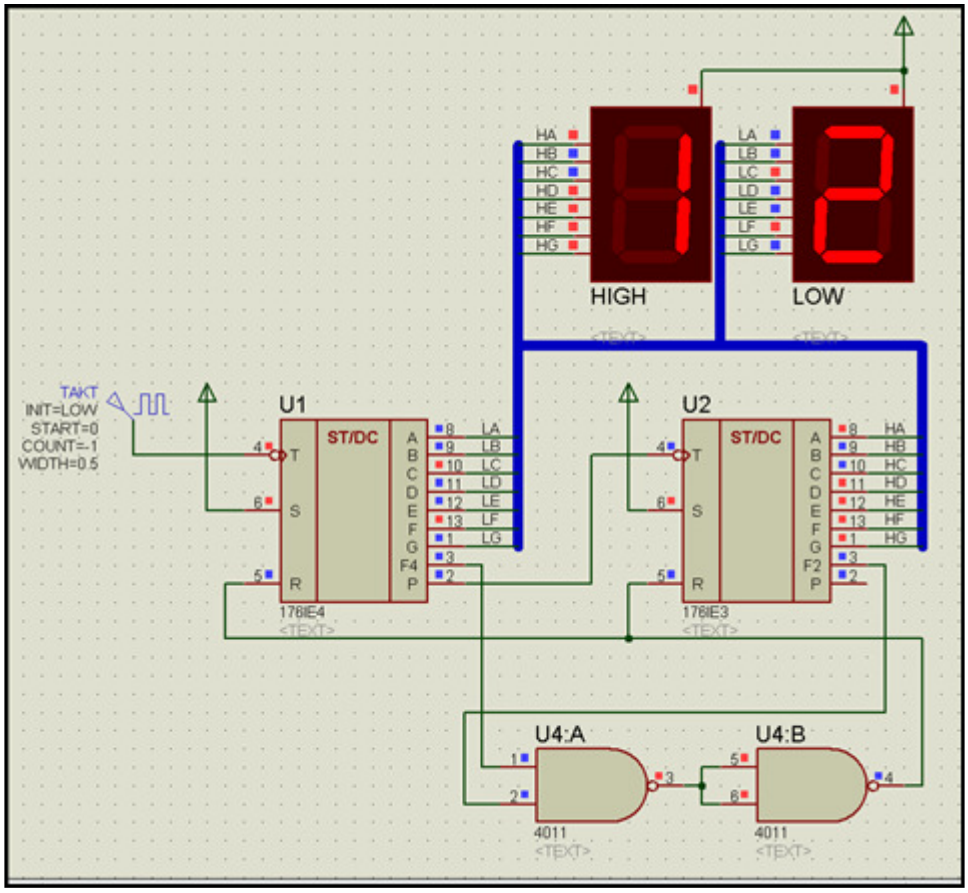
В свойствах U1 **UPPER=(вот засада! здесь UPPER в угловых скобках, но движок форума ловит его как тег и не отображает)** , в свойствах U3 отдельной строкой **INVERT =<Dx>** . И дополнительный скрипт:

```
*MAP ON VALUE
176IE3 : UPPER=5, Dx=D2
176IE4 : UPPER=9, Dx=D1
```

Но на эксперименты у меня нет времени, хочу закончить эту тему до нового года, так что дерзайте сами.

8.3. Тестовая схема для проверки моделей.

В свете вышесказанного я создал графические модели счетчиков и приделал к ним дочерние листы, на которых и воссоздал нужные мне структуры. В качестве проверки правильной работы на основном листе пришлось собрать **Test Jig** (как ее обозвал Лабцентер), т.е. тестовую имитирующую схему. В качестве таковой я собрал как-бы секцию индикации часов от реальных электронных часов на микросхемах серии 176. Это удобно тем, что одновременно проверятся выходы переноса, 4-й и 2-й импульсы для соответствующих счетчиков и их входы сброса R при достижении 24 часов (т.е. сброс по совпадению в 00 должен произойти после 24-го секундного импульса на входе, а максимальное число индикации - 23). В качестве **176ЛА7** использована **CD4011** - полный аналог нашей. Результаты на картинке, а проект во вложении. Оба счетчика имеют дочерние листы с внутренней структурой, которые также сохранены и как секции. Осталось только сгенерировать MDF-файлы с дочерних листов, а затем приклеить их к соответствующим моделям через **Make Device** , как и в случае с моделью **176ИЕ12.**



8.4. Тестируем все. Собираем библиотеку CMOS176.LML.

После создания MDF-файлов попробуем наши модели в действии. Я "спаял" простейшие часы на 176-й серии в файле проекта **TEST2.DSN** . Не могу удержаться от сарказма в сторону противников Протеуса. Ну, господа присяжные заседатели, что быстрее - сбегать в магазин за микросхемами и потом целый вечер варганить сей опус на макетке, вдыхая вожделенный, очень "полезный" для здоровья свинцовый "аромат", или повозить животное 15 минут по коврику и получить тот-же результат? Да, я затратил больше времени на создание самих моделей, но, пройдя сей путь единожды, получил "неограниченное" количество микросхем и могу "палить" их пачками без потерь в семейном или производственном бюджете. А на следующую модель, например: **176ИЕ5** или **176ИЕ8** я затрачу максимум 1 час. Но это так, лирическое отступление - двигаемся дальше.
Соберем все наши полученные **MDF** в одну папочку желательно в корне диска **C:** (так легче писать путь) и туда же поместим из папки **BIN** Протеуса утилиту **PUTMDF.EXE** . А теперь подсчитаем количество наших файлов **MDF** . Поясню для чего последнее - при создании библиотеки **LML** нам потребуется указать количество помещаемых в нее файлов. Конечно, можно указать заведомо большее, например, штук эдак 100, но под них резервируется объем в файле и он будет занимать больше места на диске. А мы и создаем нашу библиотеку, чтобы прекратить бардак в папке **MODELS** . Так зачем разводить его в другом месте? Поэтому если вы не планируете пополнять эту библиотеку, то надо указать именно то количество файлов, которое мы в нее поместим.
Итак, у меня три файла. Открываю командную консоль, вписываю туда путь к папке и набираю следующую команду (там где у меня подчеркивания должны быть пробелы):

```
PUTMDF.EXE _ -L=CMOS176.LML _-C=3
```

После этой операции в папке появится файл с именем **CMOS176.LML** . Затем вводим следующую строку

```
PUTMDF.EXE _ -L=CMOS176.LML _*.MDF
```

Должен появиться отчет о добавлении (**Storring**) всех наших файлов. И размер файла **LML** возрастет. Можно провести эту операцию и в одной строке, а если добавить ключ **-D** , то исходные MDF-файлы будут удалены. Если не указывать имя библиотеки, то по умолчанию создается **MODELS.LML** , а переименовать ее можно потом, а если в ключе **-C** забыть указать количество файлов, то будет добавлен только один. Вот однострочный вариант создания библиотеки на 100 "посадочных мест" с удалением исходных файлов:

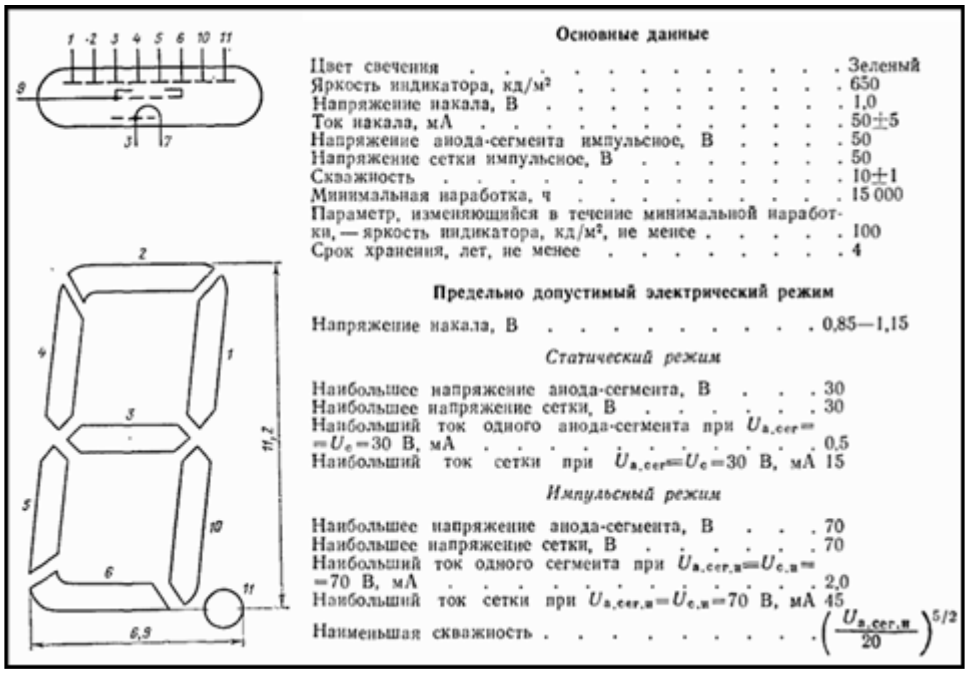
```
PUTMDF.EXE _ -L=CMOS176.LML _-C=100_ -D_*.MDF
```

После этого помещаем созданную **CMOS176.LML** в папку **MODELS** Протеуса (соответсвующие MDF-файлы там уже больше не нужны) и проверяем работу наших моделей.
Во вложении тестовый проект часов с файлами **MDF** , а в архиве **CMOS176** - новогодний подарок для "ленивых". Скопируйте из соответствующих папок архива файлы **CMOS176.LIB** и **CMOS176.LML** в одноименные папки Протеуса и все, что написано выше можно не читать и не делать. Проверены в версиях **7.2.SP6** и **7.4.SP3** .

8.5. Исходные данные для моделирования индикатора ИВ-6.

Библиотеки Протеуса содержат достаточно большое количество разнообразных элементов индикации в разделе **Optoelectonics** . Однако, такой тип индикатора, как вакуумно-люминисцентный в них отсутствует. В то же время иногда возникает необходимость применить именно такой индикатор. Попробуем восполнить этот пробел и создать простенькую модель пусть и устаревшего, но иногда еще используемого радиолюбителями индикатора ИВ-6. Обратимся к справочнику Н.Вуколова и А.Михайлова "Знакосинтезирующие индикаторы"(см. картинку)

Исходя из этих данных, определимся: что мы хотим получить от нашей модели и какие она будет иметь ограничения. Мне хотелось бы, чтобы при напряжении накала ниже 0,85V индикатор был погашен, а при напряжении 1V ток накала составлял 50mA Согласно закону Ома при этом сопротивление нити составит 20 Ом. Аналогично для сетки при напряжении 30V и токе 15mA - сопротивление равно 2кОм, а для одного сегмента при 0,5mA - 60кОм. Эти данные я использую при создании модели. Ну и конечно цоколевка - соответствие выводов - сегментам. Сразу же оговорю еще одно ограничение-допущение, которое будет в модели. Аналоговыми свойствами с точки зрения нагрузки она будет обладать только для окружающей периферии. Я не планирую плавно изменять яркость свечения индикации, поскольку подозреваю, что это вызовет чрезмерное усложнение модели и соответственно нагрузку на процессор компьютера и на мои мозги тоже. Поэтому я пошел практически тем же путем, что и Лабцентровцы, а в качестве исходного материала для моделирования выбрал из раздела **Optoelectonics** одиночный семисегментный индикатор **7SEG-COM-AN-GRN** (его **MDF** расположен в файле **DISPLAY.LML** - можно извлечь и посмотреть).



Основные данные

Цвет свечения	Зеленый
Яркость индикатора, кд/м²	650
Напряжение накала, В	1,0
Ток накала, мА	50±5
Напряжение анода-сегмента импульсное, В	50
Напряжение сетки импульсное, В	50
Скважность	10±1
Минимальная наработка, ч	15 000
Параметр, изменяющийся в течение минимальной наработки, — яркость индикатора, кд/м², не менее	100
Срок хранения, лет, не менее	4

Предельно допустимый электрический режим

Напряжение накала, В	0,85—1,15

Статический режим

Наибольшее напряжение анода-сегмента, В	30
Наибольшее напряжение сетки, В	30
Наибольший ток одного анода-сегмента при $U_{а.сег}=U_{с}=30$ В, мА	0,5
Наибольший ток сетки при $U_{а.сег}=U_{с}=30$ В, мА	15

Импульсный режим

Наибольшее напряжение анода-сегмента, В	70
Наибольшее напряжение сетки, В	70
Наибольший ток одного сегмента при $U_{а.сег.и}=U_{с.и}=70$ В, мА	2,0
Наибольший ток сетки при $U_{а.сег.и}=U_{с.и}=70$ В, мА	45
Наименьшая скважность	$\left(\frac{U_{а.сег.и}}{20}\right)^{5/2}$

8.6. Джентльменский набор для моделирования индикатора.

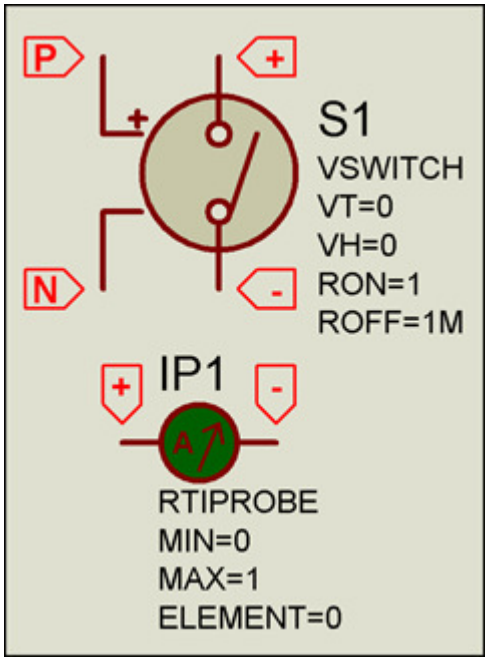
Аналогично **7SEG-COM-AN-GRN.MDF** для моделирования потребуются примитивы **RTIPROBE** - токовый пробник-индикатор реального времени и **VSWITCH** - управляемый напряжением включатель (практически аналог электромагнитного реле). Ну и конечно, для иммитации нагрузки примитив резистора. Напомню, что все это берется из библиотеки **Modelling Primitives** , а не где-нибудь еще.
Рассмотрим подробнее те свойства незнакомых примитивов, которые нам потребуются.

RTIPROBE:

MIN и MAX - соответственно минимальное и максимальное значение тока по которому определяется индицируемое состояние STATE в соответствии с формулой приведенной в HELP для модели:
STATE=(NUMSTATES-1)* ((CURRENT-MIN)/(MAX-MIN))
Где CURRENT - текущее значение тока, NUMSTATES - число состояний.
Если ток меньше MIN, то состояние принимается как MIN, если больше MAX - то максимум. Внутри диапазона дробное значение округляется к ближайшему целому.
ELEMENT - номер графического элемента (2D GRAPHIC SYMBOL), который индицирует состояния данного индикатора. Подробно применение этого свойства рассмотрено на примере **BITWISE indicator** (он находится в **VSMSDK.HLP** , который я ранее рекомендовал скачать). Чуть ниже мы им воспользуемся практически.

VSWITCH:

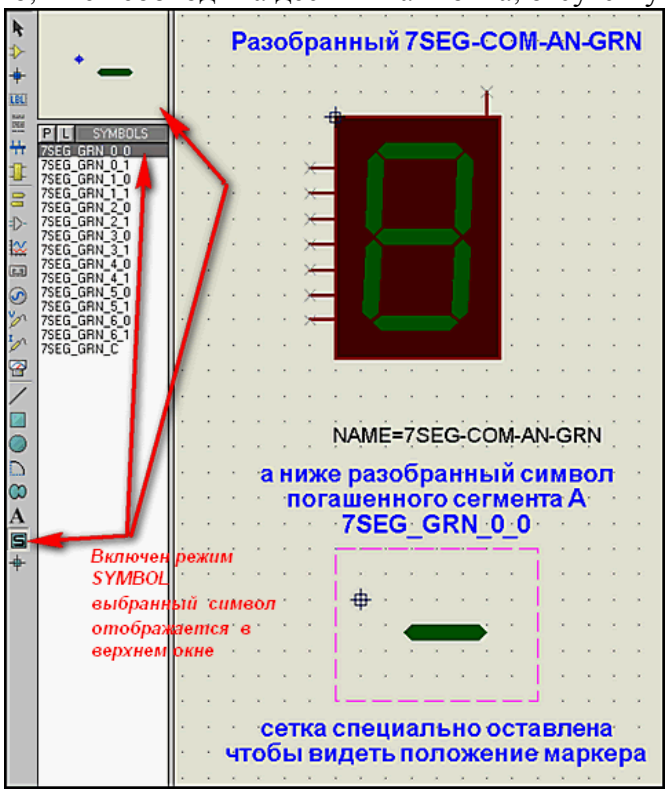
VT - Threshold Voltage - напряжение срабатывания
VH - Hysteresis Voltage - напряжение гистерезиса (если не 0, то включение при VT+VH/2, выключение при VT-VH/2, по умолчанию равно 0)
RON - ON Resistance - сопротивление контактов в замкнутом состоянии (по умолчанию 1 Ом)
ROFF - OFF Resistance - сопротивление контактов в разомкнутом состоянии состоянии (по умолчанию 1 МОм)
Расположение и фигурирующие в **MDF** названия выводов этих примитивов на картинке ниже:



8.7. Готовим исходный материал для графической модели ИВ-6.

Чтобы не начинать создание графики с нуля я воспользуюсь вышеупомянутой моделью **7SEG-COM-AN-GRN**. Как и прежде втаскиваем ее на лист и доблестно расшибаем молотком (**Decompose**). В отличие от обычных схематичных моделей у активных имеются в составе еще и графические символы, которые после "удара молотком" доступны для обозрения и использования при нажатии кнопки **S** на зеленом фоне (**2D Graphics Symbols Mode**) в левом меню **ISIS** (см. картинку ниже). Здесь необходимо несколько пояснений. Символы - это те графические элементы, которыми при симуляции заменяются на изображении активного компонента в зависимости от значения **RealTime** пробника определенные области изображения. Они жестко привязаны по расположению относительно маркера (это тот прицел, что этажом ниже под S в левом меню. Вот теперь, я надеюсь Вам до конца понятно назначение этого маркера в графическом элементе. Для примера на картинке вытащен на поле и также "декомпозирован" погашенный верхний сегмент индикатора. Маркер станет виден только после воздействия на символ все тем же молотком. Символам при создании присваиваются названия и номера в зависимости от их принадлежности. Например в разобранном нами индикаторе имя группы - **7SEG_GRN** . Оно общее для всех символов. Дальше через символ подчеркивания следует номер в группе начиная с нулевого: **_0** , **_1** , **_2** и т.д. Именно он и является для пробника параметром **ELEMENT** (см. картинку выше). Для тела компонента это значение: **_C**. И следующий номер также через символ подчеркивания - это порядковый номер изображения для данного символа. В разобранном **7SEG_GRN** их всего два: **_0** - для погашенных сегментов и **_1** - для засвеченных. В качестве эксперимента я рекомендую на досуге "расколотить" активный светодиод - вот там в зависимости от яркости вторичная нумерация богата. Именно эти номера и определяют яркость изображения в зависимости от параметра **STATE** пробника (формула в предыдущем пункте).

Итак, для чистоты эксперимента я вытащу все изображения символов на проект, разобью их и, слегка изменив, создам свои с другим цветом затененных символов (ведь на реальном ИВ-6 они бледно-серые) и заменю цвет тела компонента на грязно-желтый (тоже поближе к реальности). Кроме того, мне необходима десятичная точка, отсутствующая в прототипе.



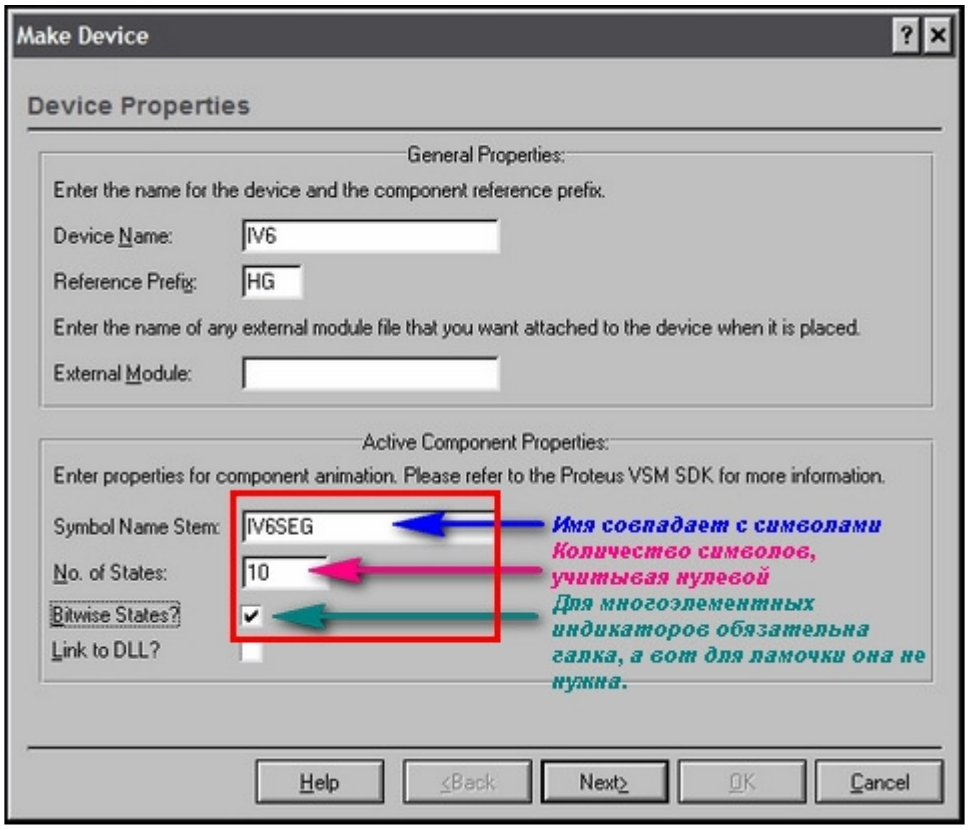
8.8. Создание собственных символов.

В качестве примера я добавлю активные (подсвечиваемые) символы нити накала и сетки, чтобы не забыть подать на них напряжение при симуляции. Это придаст нашему индикатору своеобразную "изюминку" по сравнению с остальными семисегментными моделями Протеуса. На что здесь необходимо обратить внимание:
Положение активного (засвечивающего или затемняющего) элемента относительно маркера для всех символов имеющих одинаковые номера (**ELEMENT**) должны быть строго выровнены по сетке. При необходимости можно изменить шаг сетки на более мелкий через верхнее меню **View => Snap**. Я поступаю следующим образом: Выделяю область, где расположен базовое изображение (в нашем случае то что мы "разбили" можно вместе со скриптом). Затем нажимаю в верхнем меню кнопку **Block Copy** (Два зеленых прямоугольника с жирной красной вертикальной стрелкой вниз), которая становится активной после выделения области на проекте. После этого раставляю по полю проекта левой кнопкой хвостатой нужное мне количество изображений. А потом на каждом двойным щелчком правой кнопкой удаляю ненужные мне элементы: выводы, тело и т.п. не затрагивая при этом маркер и нужный в данном символе элемент. Для самих сегментов не забудьте еще через меню **Edit** сделать **Bring To Front** (на передний план), чтобы символ не был загорожен телом при симуляции. А для тела соответсвенно: **Send To Back**. Теперь как это сохранять. Если мы выделим графический элемент совместно с маркером, то через правую лапку животного нам будет доступна активная опция **Make Symbol**. В открывшемся окне вводим имя со всеми номерами через подчеркивания (например: **IV6SEG_0_0** , а для светящегося - **IV6SEG_0_1** , чтобы не засорять системную библиотеку символов выбираем в окне **Library => USERSYM**, а **Type** оставляем **Graphic**. Жмем ОК, после чего в селекторе (см. картинку выше) в списке символов появится сохраненный нами. Я рекомендую перед сохранением первого символа сразу же скопировать в буфер обмена его имя (это для особо ленивых вроде меня) из окна **Name**, а потом только вставлять (Paste) его и менять цифры. Это предохранит Вас от недоразумений при симуляции (потеря символов), так как имена, за исключением номеров у всей группы должны быть строго идентичны.Все эти этапы создания графики я свел в один проект на два листа. Художник из меня никакой, так что за убогую графику приношу извинения. При создании символов обратите внимание, что вместе со светящимся элементом должен быть захвачен выделением и маркер (прицел), но больше ничего. Сиреневые линии обводки я ввел только чтобы разграничить отдельные символы их ни в коем случае нельзя захватывать. Не забудьте про символ "тела" с индексом _C.

8.9. Создаем модель. Добавляем дочерний лист и редактируем его свойства.

Внимание! В справочнике Вуколова очепятка. Левый по схеме вывод накала, он же соединен физически с катодом должен иметь номер 8. Правильное расположение выводов у индикатора ИВ-3А. Устраните это недоразумение в предыдущем вложении.
Итак мы нарисовали графику, приклеили к модели выводы, пронумеровали их - пора создавать модель. Технология мало чем отличается от изложенной для 176ИЕ12. Есть одно существенное НО на первой вкладке **Make Device** . Что там добавляется ясно из картинки ниже. Поясню особенности:
*** Simbol Name Stem** - имя, которое мы присвоили нашим символам при создании. НО без подчеркиваний и номеров в конце! Эти символы при создании модели должны быть доступны при нажатии **S** (см. картинку в предыдущем пункте)
*** No of States** - количество состояний. Для бит зависимых (наш сегментный индикатор) ставим число созданных символов, учитывая, что нумерация с нуля. "Тело" не в счет. Если

меняется только один элемент, то ниже нет галки, а число должно совпадать с числом его градаций яркости (например лампочка в HELP **VSMSDK**).
*** Bitwise States** - галочка ставится для многоэлементных индикаторов, как наш. Если бы мы лепили светодиод или лампочку, где меняется только один элемент по яркости - она не нужна.
Создали нашу модель пора приделать к ней дочерний лист. И здесь тоже есть особенность, с которой я долго бился. После создания дочернего листа (**Attach hierarchy module**), заходим в него и через верхнее меню **Design => Edit Sheet Properties** обязательно добавляем галку в окошко **External .MOD file** , иначе наша модель при отладке работать не будет. Лист должен иметь свое имя, чтобы файл именовался в соответствии с ним. На этом все подготовительные процедуры закончены и мы можем на дочернем листе создавать внутреннюю структуру модели, используя наш "джентльменский набор" и другие примитивы. При этом у **RTIPROBE** или **RTVPROBE** в качестве **ELEMENT** указываем номер (**ИМЕННО НОМЕР А НЕ ПОЛНОЕ НАЗВАНИЕ**) соответствующего символа, т.е. для нашего случая от **0** до **9**.



8.10. Создаем внутреннюю структуру индикатора и тестируем ее.

Для создания модели, поскольку она учебная я постарался запихать на дочерний лист как говорится "всяко-разно, лишь бы не заразно". Кроме упомянутых выше RTIPROBE и VSWITCH в новый джентльменский набор вошли:

ACCVS: управляемый током источник напряжения напряжение на выходе вычисляется по формуле, которая указана в свойствах - по умолчанию: **U=1,0 * I(A,B)**. Где **I** - ток. Т.е, если ток через левые выводы равен 1 А, то напряжение на выходе справа 1V. Поскольку ток у меня в mA, чтобы получить приличное напряжение на выходе и не валандаться с дробями я применил множитель 1000 вместо единицы.

CSWITCH: аналогичен **VSWITCH** и обладает теми же пропертями, только по входу управляется не напряжением, а током.

RTVPROBE: аналогичен **RTIPROBE** , но контролирует не ток, а напряжение. Свойсва и формула в хелпе похожи, но вместо **CURRENT** (текущее значение тока) стоит **VOLTAGE** (напряжение). У **RTVPROBE** есть еще одно дополнительное свойство: **LOAD** (нагрузка), которое по умолчанию отсутствует. Но я применил его для одного из пробников, чтобы нагрузить выход переключателя **CSWITCH** хотя бы на 1МОм, иначе даже при **ROFF=1000МОм** у него на выходе будет висеть напряжение.

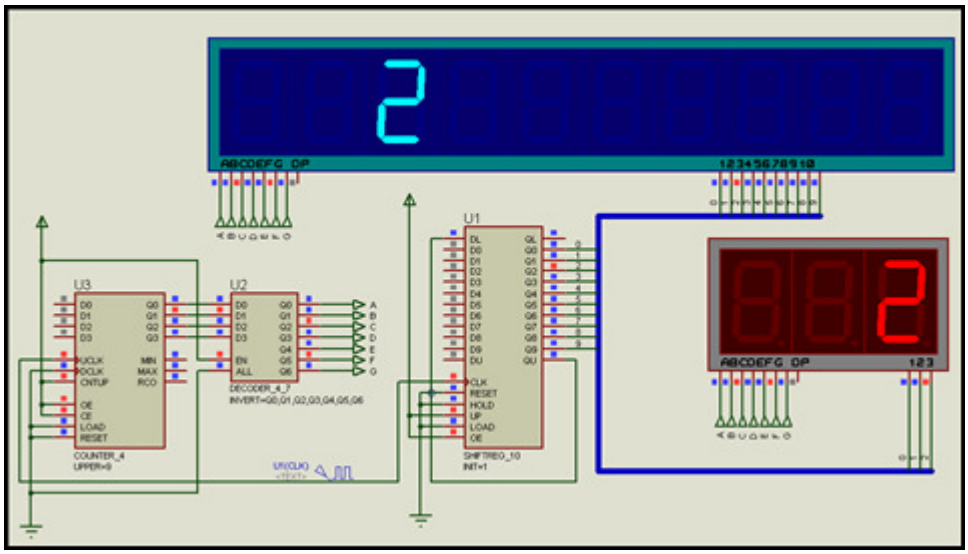
Внутренняя структура получилась достаточно корявая, но я стремился к тому, чтобы показать как все это делается, а не к максимальной простоте и быстродействию. Отсюда и вылезли дополнительный однополупериодный выпрямитель и **ACCVS** на накальных выводах - стремление к достоверности, т.е. к тому, чтобы запитать накал переменным напряжением, ну прямо как у настоящего ИВ6. Я не стал полностью изолировать цепь накала от остальной схемы, хотя если покумекать, то можно это сделать. Опять сошлюсь на то, что вывод накала 8 у реального индикатора связан с катодом и позволяет питать накал постоянным током, при этом к этому выводу должен подводиться минус источника. Как видите даже это в моей модели реализовано. Во вложении проект с прикрепленным к модели дочерним листом и **MOD** - файлом (не путайте с **MDF**), который создался из дочернего листа при установке флага **External .MOD file** (см. выше). Модель полностью жужжит, желающие могут поиграться с внутренней структурой на дочернем листе и с проверкой работы по внешним выводам на основном. Кому не терпится может сразу скомпилить **MDF** также как и для модели 176ИЕ12, тут никаких "подводных камней" не зарыто и в дальнейшем использовать ее. Однако не забудьте все же про **Make Device** , чтобы модель появилась в Ваших библиотеках Протеуса, т.к. сам по себе **MDF** - это еще не модель. Кому не ясно - вспомните, что для счетчиков 176 серии я прилагал кроме **LML** еще и файл **.LIB** , который помещается в папке **LIBRARY** . Это прописывается на последней вкладке при **Make Device** .

9.1. Манипуляции с LED или программная модель без программирования.

"А русские прилетали - с молотком, зубилом и какой-то матерью за два часа починились" (из анекдота)

В ходе создания активной модели ИВ-6 однажды меня осенила туповатая, но как оказалось верная по существу мысль. А именно: если есть в Протеусе модели двух-, четырех-, шестиразрядных 7-сегментных дисплеев, то почему не быть трехразрядным или, допустим, - десятиразрядным, тем более что у меня давно лежит парочка трехразрядников в хламе. Взял и проверил эту идею. Так что про молоток я тут не зря упомнил. Процедура старая - добываем, например, четырехразрядный индикатор с ОА, колотим его, вытираем один разряд из графики, подрезаем (смещаем границу) "тела" и выводы, чтоб не торчало лишнее и убираем четвертую ногу. Дополнительно редактируем надпись (это просто графическая надпись и никак не связана с количеством выводов) 1234 переделываем в 123. Вот и вся процедура. Обводим все это выделением и … волшебное **Make Device** . Только правим имя

устройства (ведь у нас уже 3, а не 4 разряда) и далее по всем пунктам. На последней вкладке не забудьте внести аналогичную поправку в окошке **Device Description** . Модель будет создана по умолчанию в библиотеке **USERDVC** . Вот так в пять минут не написав ни строчки кода мы слепили первую программную модель. Чуть больше я потратил на десятиразрядник. Особо пытливые при создании девайса на вкладке **Component Properties & Definitions** могут посмотреть в свойстве **INVERT** различия у моделей с ОА и ОК.
ВНИМАНИЕ! Начиная с этого поста я прилагаю примеры в версии Proteus 7.4SP3! Для импорта в предыдущие версии прилагается **Section** .



9.2. Создаем свою клавиатуру (Keypad).

В стандартной поставке Протеуса есть три вида Keypad калькуляторные и телефонная клавиатуры. Однако универсальная модель **Keypad.DLL** позволяет создавать собственные варианты, чем мы и займемся. Все это подробно описано в **HELP**, доступном для вызова при установке любой из этих моделей в проект. Я взял за основу телефонную клаву. Процедуры все те же. Колошматим, создаем свою графику и мэйкаем. Некоторые замечания из **HELP** для не владеющих аглицким.
Графика: расставляя кнопки на своей клавиатуре необходимо зафиксировать их координаты относительно верхнего левого угла "тела", чтобы они потом фигурировали в свойствах для указания положения конкретной кнопки.На "разобранной" модели на этом месте стоит маркер **ORIGIN** (прицел) Наведите курсор на его центр и нажмите клавишу буквы[**O**]. При этом зафиксируется псевдоначало координат (в правом нижнем углу окна ISIS позиция курсора станет **0, 0**). Теперь наводя курсор на центры наших кнопок мы получим их относительные координаты и зафиксируем их на бумажке.
Выводы строк (слева) именуем латиницей **A, B, C, D** и т.д. Выводы столбцов цифрами **1, 2, 3, 4** и т.д. Позиция конкретной кнопки записывается как **B4** или **D2, E9** и т.п. Внутри кнопок можно смело вставлять надписи на русском.

Параметры: кнопки могут быть:
-квадратными - (SQUARE,координата X, координата Y, ширина (она же высота))
-прямоугольными - (OBLONG, координата X, координата Y, ширина, высота)
-круглыми - (ROUND, координата X, координата Y, ширина, высота)
Пример записи для последней:

{A1=ROUND,300,-300,300,1}
или
{D3=ROUND,1100,-1500,300,#5}

Здесь нас особо интересует последнее значение перед закрытием фигурной скобки - это клавиша основной клавиатуры компьютера, которая назначена нашей кнопке. В данных примерах это **1** основной клавы и **5** цифровой дополнительной.
Назначать можно основную клавиатуру (вот здесь используем только латынь), Дополнительную цифровую клавиатуру справа. Пример: **#1** или **#6**.
Сочетания с **CTRL**. Пример: **^D**, **^R**.
Спец. клавиши Например: **INSERT, DELETE, HOME, END, PGUP, PGDN** (задавать как здесь!).
Ну и наконец глобальное назначение **@5** или **@R**. О нем особо. Дело в том, что при симуляции кнопки становятся активными для ввода с клавиатуры только при наведении курсора хвостатой на область модели кейпада в проекте или непосредственном нажатии их левой лапой грызуна. Вот если вы хотите дать передышку мелкому зверьку - используйте такое назначение с "собакой". Тогда наводить курсор на Keypad не надо можно сразу пользоваться назначенными клавишами компа после запуска симуляции.
Ну вот вкратце все по этой модели и файл с примерами и опять с Section для предыдущих версий.