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

Содержание.

3. Виды симуляции и типы моделей в ISIS.

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

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

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

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

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

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

стоит подумать об использовании графиков для анализа поведения схемы.

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

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

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

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

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

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

3.2. Типы моделей в ISIS.

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

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

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

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

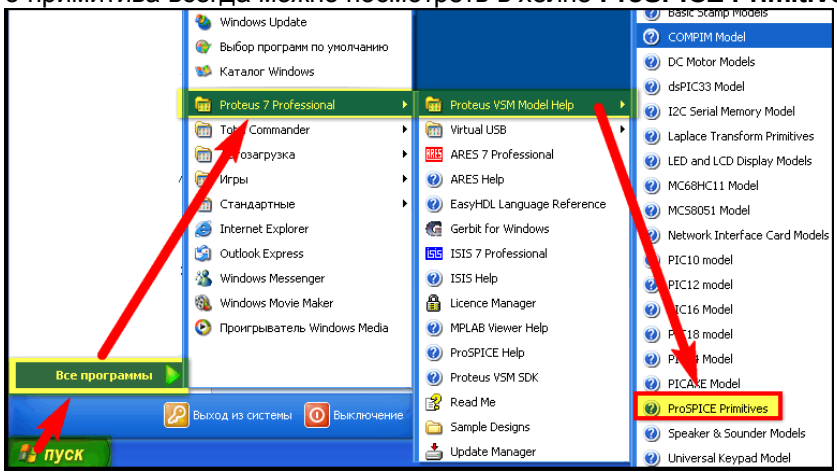


Рис.1

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

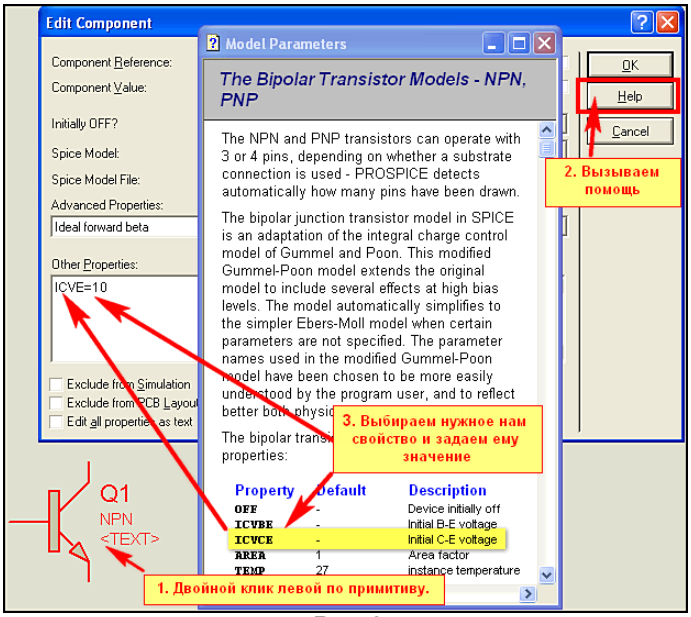


Рис.2

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

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

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

3.3. Взаимосвязи симулятора Proteus VSM.

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

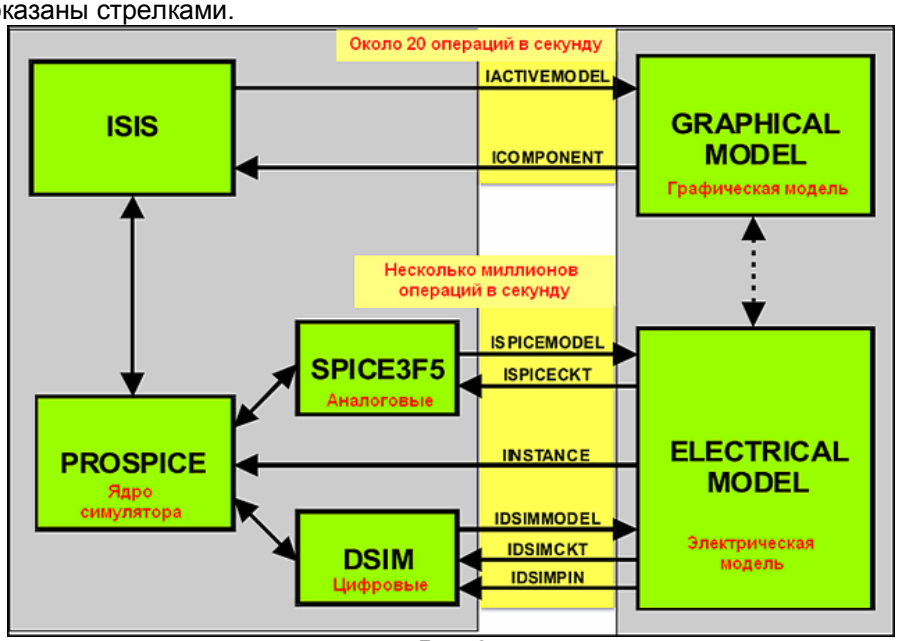


Рис.3

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

3.4. Параметры анимации.

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

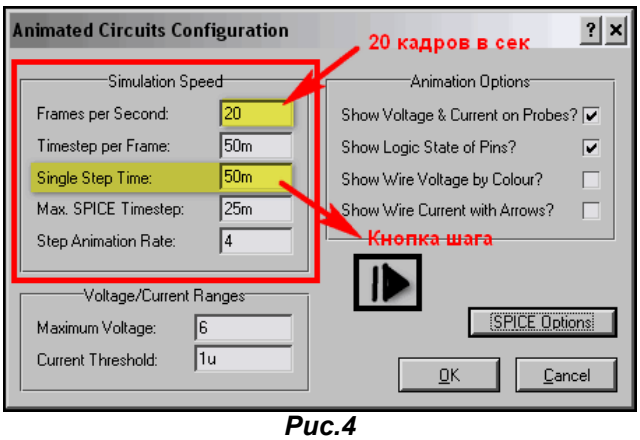


Рис.4

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

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

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

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

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

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

3.5. Параметры симуляции.

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

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

- Вкладка **Tolerances** *(Рис.5).* На этой вкладке представлены параметры, определяющие с какой точностью **ProSPICE** производит вычисление решений. При очень высокой точности может потребоваться больше времени для моделирования и в ряде случаев решение может не иметь сходимости вообще.

  ABSTOL – абсолютная ошибка расчета токов;

  VNTOL – допустимая ошибка расчета напряжений;

  CHGTOL – допустимая ошибка расчета зарядов;

  RELTOL – допустимая относительная ошибка расчета напряжений и токов;

  PIVTOL и **PIVREL** – абсолютная и относительная величины элемента строки матрицы узловых проводимостей;

  GMIN – минимальная проводимость ветви цепи – определяет утечки обратно смещенных полупроводниковых переходов и других точек схемы с высоким импедансом. В ряде случаев уменьшение этого значения может помочь получить сходимость для схем, которые при стандартных параметрах по умолчанию не имеют решений, однако при этом понизится

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

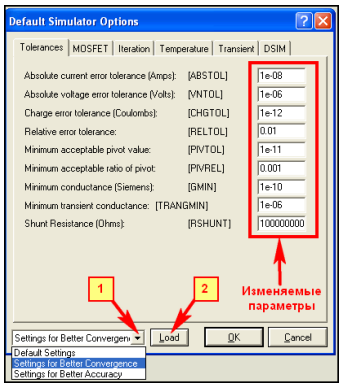


Рис.5

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

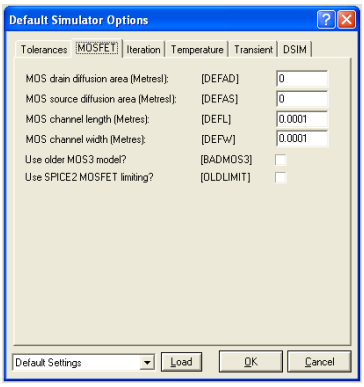


Рис.6

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

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

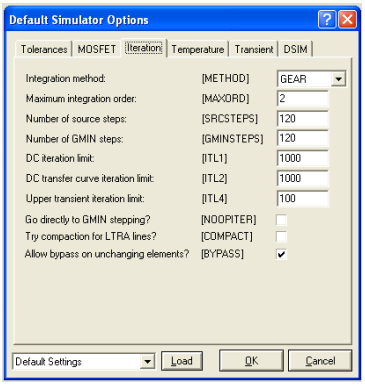


Рис.7

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

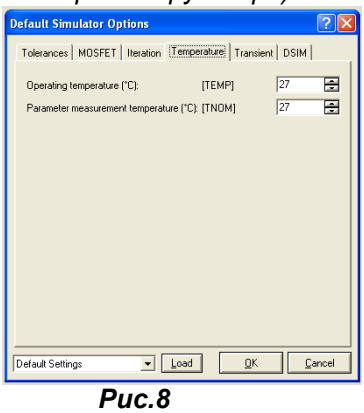


Рис.8

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

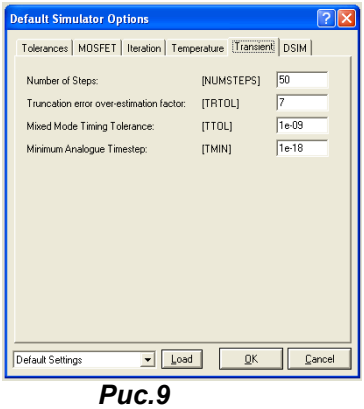


Рис.9

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

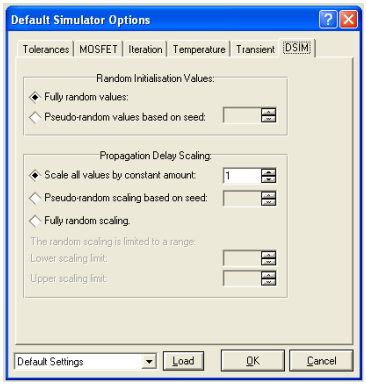


Рис.10

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

Разевиг В.Д. «Схемотехническое моделирование с помощью программы Micro-Cap» 2003 г.
К сожалению, безвременно ушедший Виктор Данилович больше не порадует нас своими шедеврами. Если я когда-нибудь и напишу что-либо подобное, то буду считать, что жизнь прожита не зря. В любой его книге по косточкам разобран описываемый предмет. Например, из этой я частично «списал» описания параметров симуляции SPICE (стр. 62-65).

Короновский А.А., Храмов А.Е. «Применение Electronics Workbench для моделирования электронных схем» Учебно–методическое пособие. Саратовский Госуниверситет, 2004.
Как видите, даже старенький Electronics Workbench перекликается с Протеусом, а все благодаря SPICE. На стр. 17-18 те же описания параметров Spice, как впрочем, и Multisim:

Хернитер М. «Multisim 7 Современная система компьютерного моделирования и анализа схем электронных устройств». 2006 г.

Хайнеман. Р. PSPICE «Моделирование работы электронных схем». 2005 г.

Петраков. О М. Создание аналоговых PSPICE моделей радиоэлементов. 2004 г.
Последней книги нет в сети, я пользуюсь бумажным вариантом. Но аналогичный по названию цикл статей выходил в журналах «Схемотехника» в 2002 г. В них же в районе 2005 г. был и цикл по цифровым моделям. Для тех, кто решил

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

4. Создание моделей компонентов в ISIS.

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

4.1. Создание графических моделей компонентов.

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

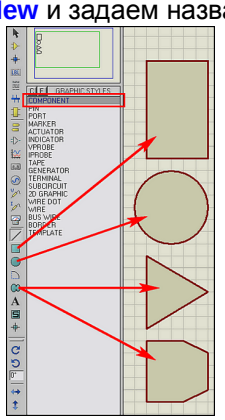


Рис.11

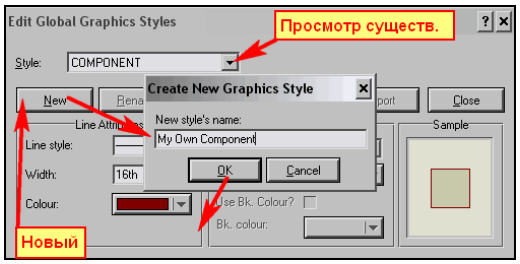


Рис.12

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

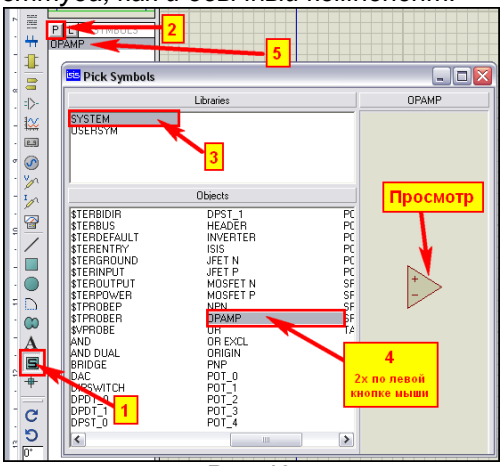


Рис.13

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

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

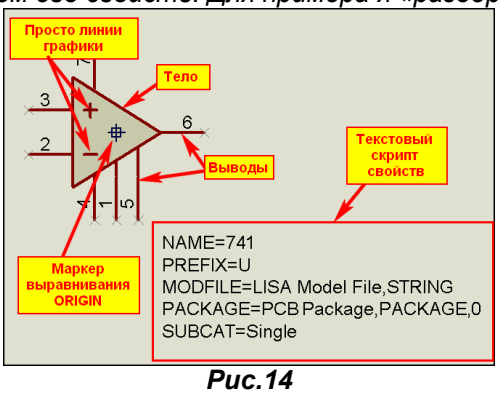


Рис.14

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

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

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

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

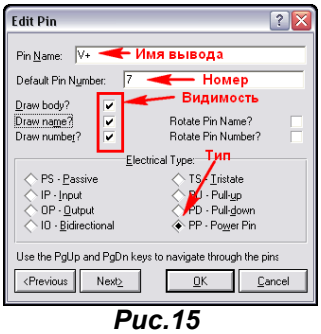


Рис.15

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

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

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

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

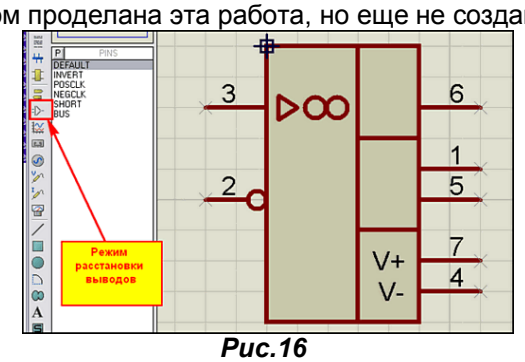


Рис.16

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

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

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

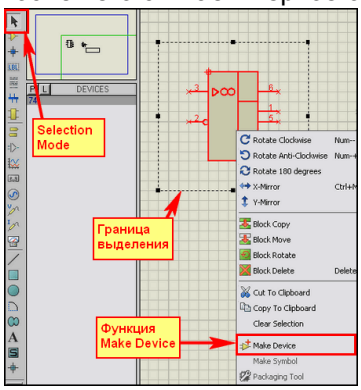


Рис.17

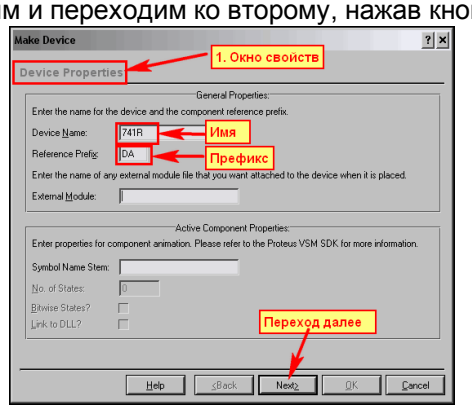


Рис. 18

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

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

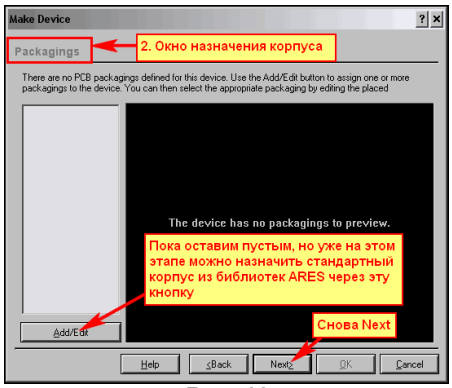


Рис.19

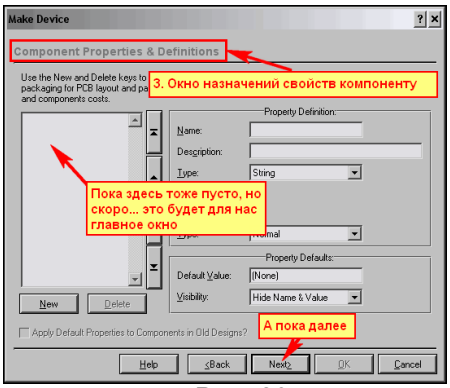


Рис. 20

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

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

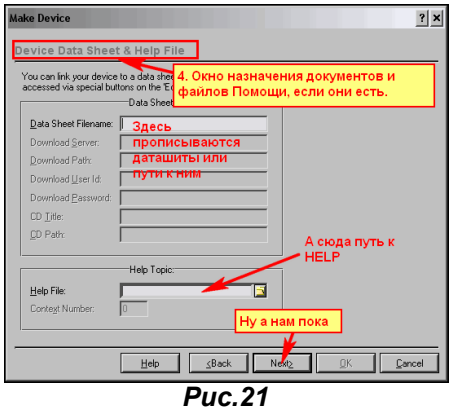


Рис.21

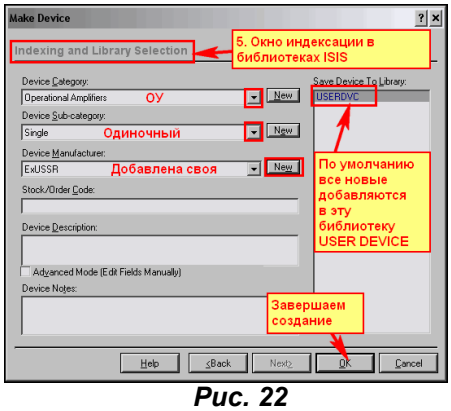


Рис. 22

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

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

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

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

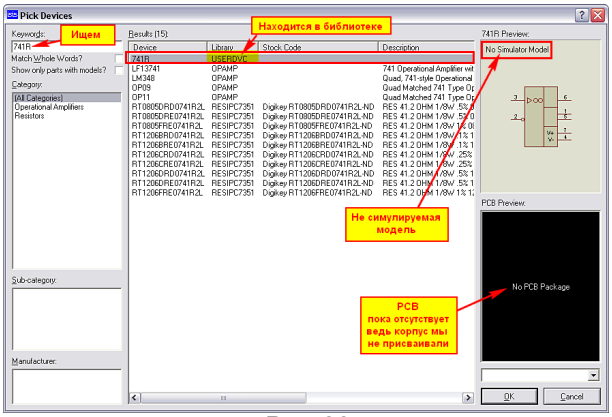


Рис.23

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

No model specified for DA1

Simulation FAILED due to partition analysis error(s)

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

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

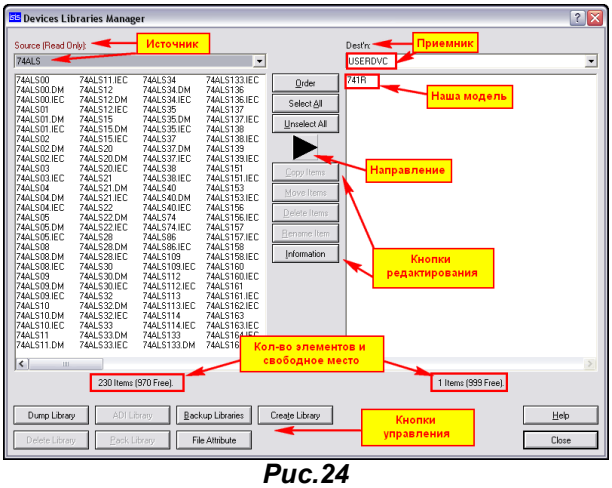


Рис.24

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

Кнопки редактирования между панелями:

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

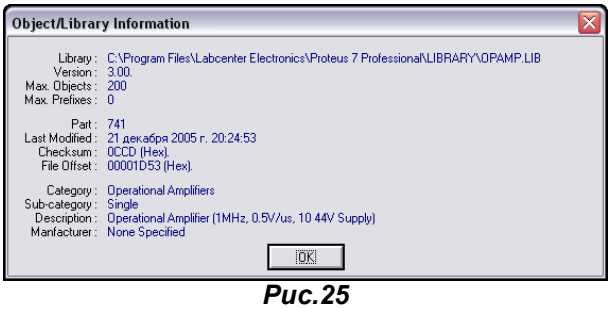


Рис.25

Теперь перейдем к рассмотрению кнопок управления библиотеками в нижней части:

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

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

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

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

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

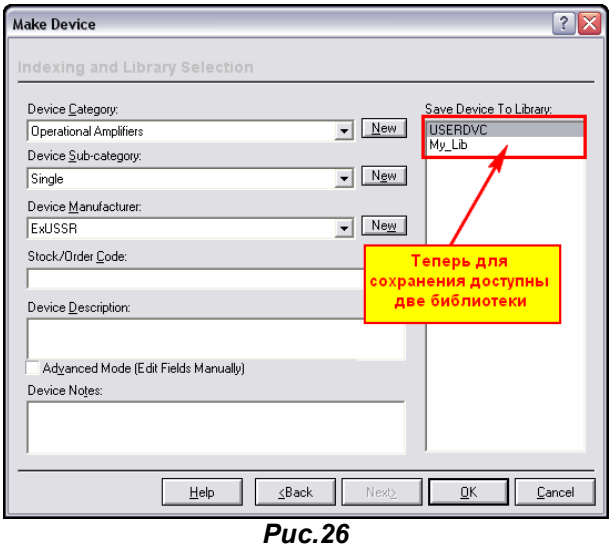


Рис.26

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

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

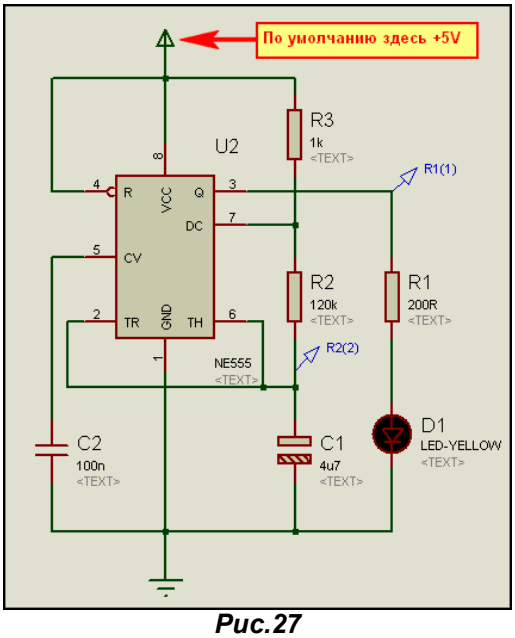


Рис.27

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

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

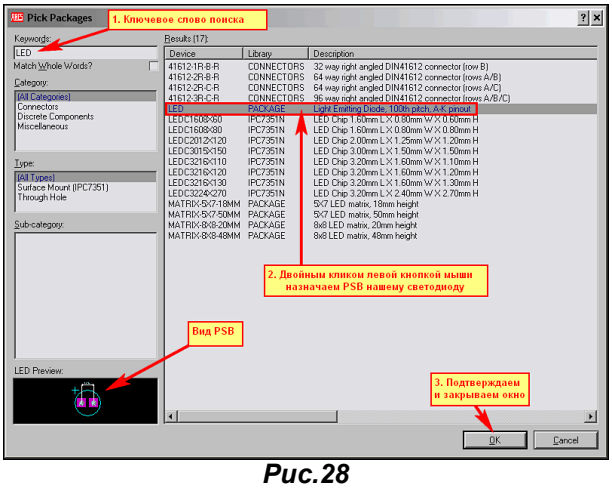


Рис.28

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

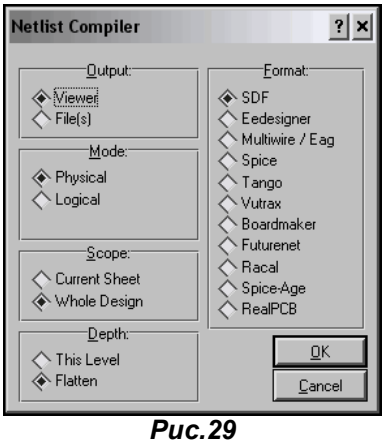


Рис.29

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

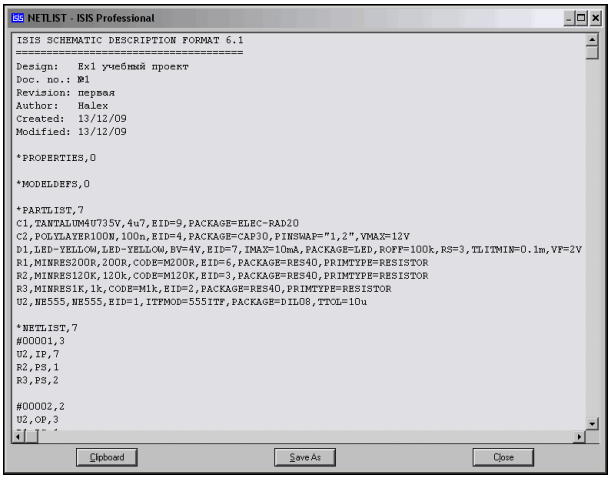


Рис.30

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

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

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

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

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

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

*PROPERTIES,0

*MODELDEFS,0

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

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

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

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

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

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

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

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

*GATES,0
```

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

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

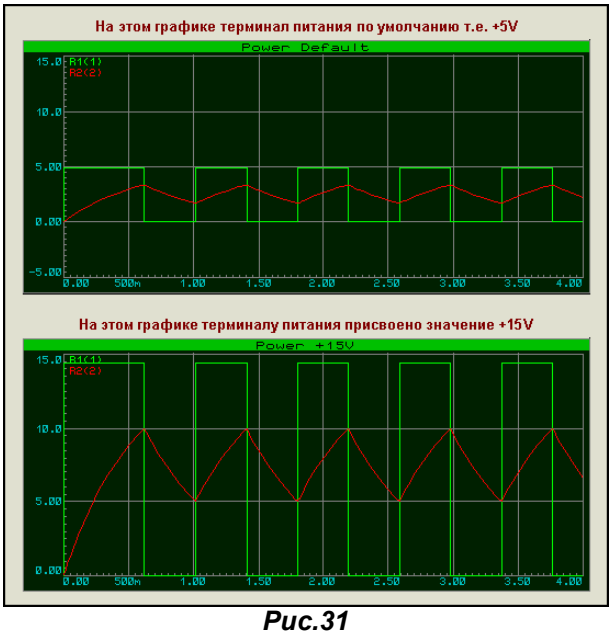


Рис.31

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

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

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

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

*GATES,0
```

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

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

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

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

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

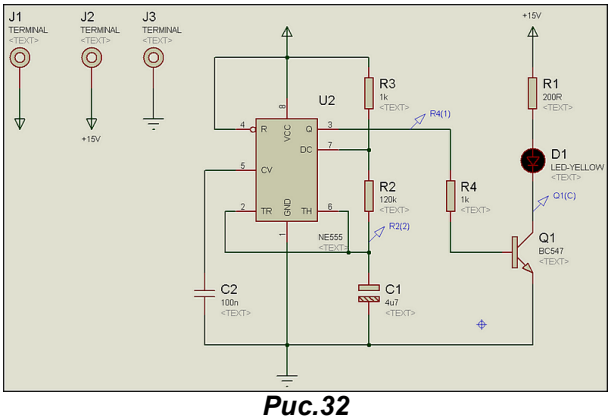


Рис.32

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

В левой части окна расположены четыре переключателя.

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

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

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

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

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

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

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

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

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

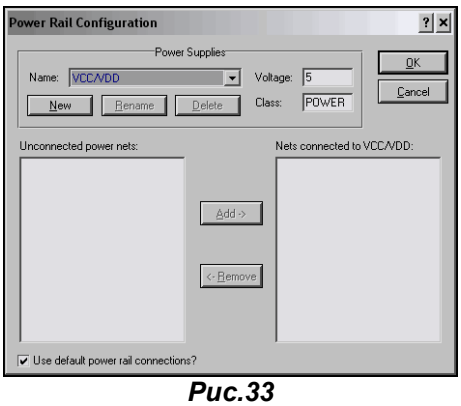


Рис.33

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

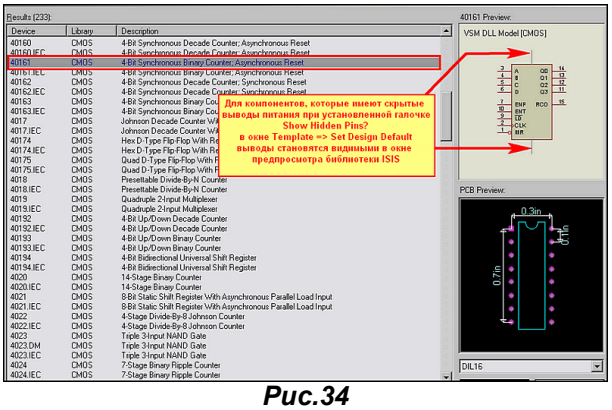


Рис.34

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

Для начала три не очень «лирических» замечания.

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

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

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

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

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

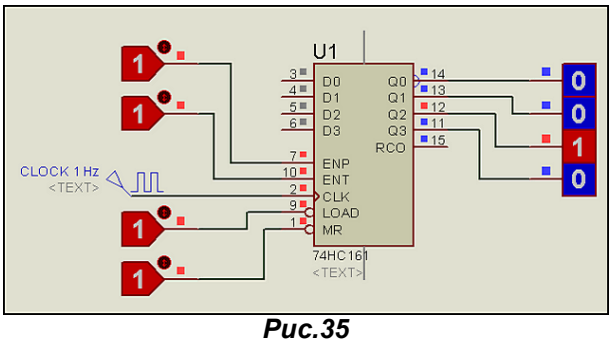


Рис.35

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

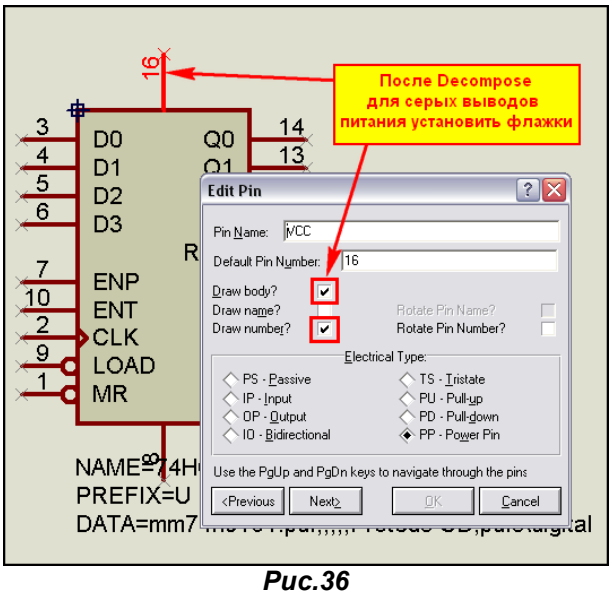


Рис.36

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

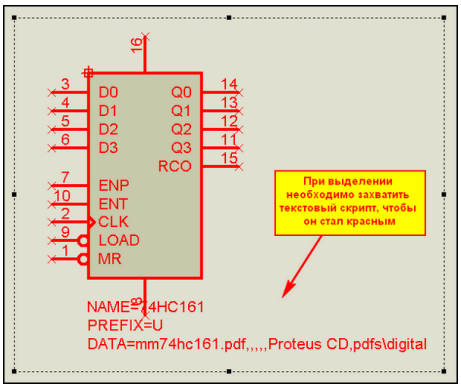


Рис.37

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

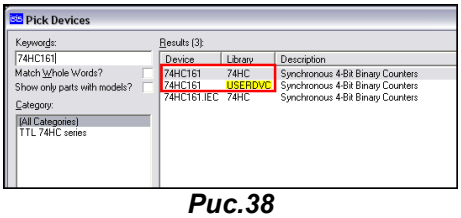


Рис.38

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

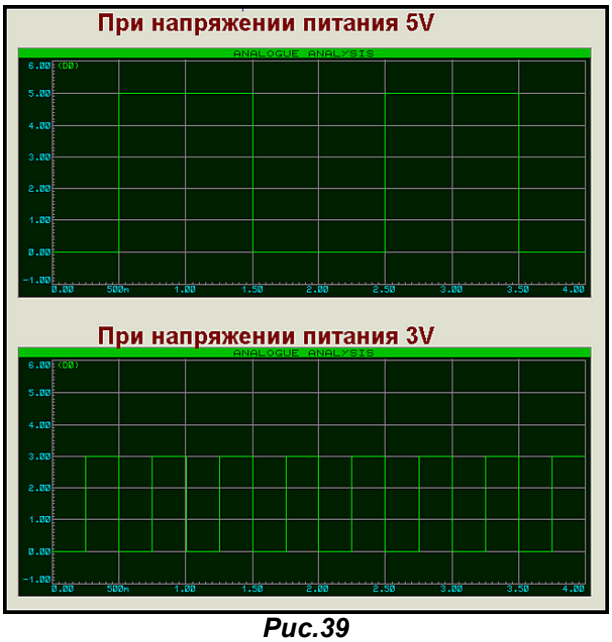


Рис.39

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

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

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

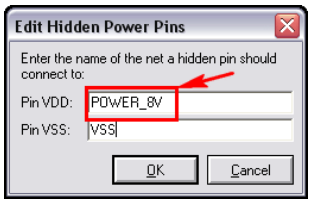


Рис.40

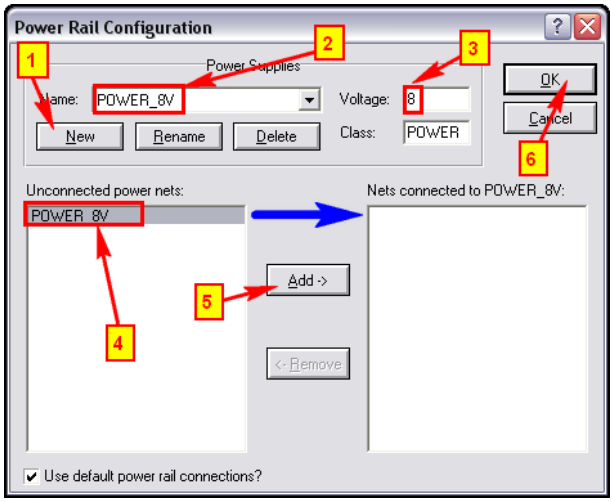


Рис. 41

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

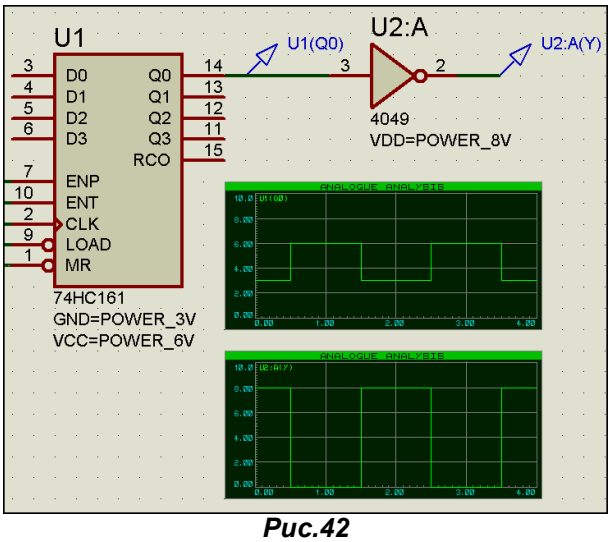


Рис.42

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

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

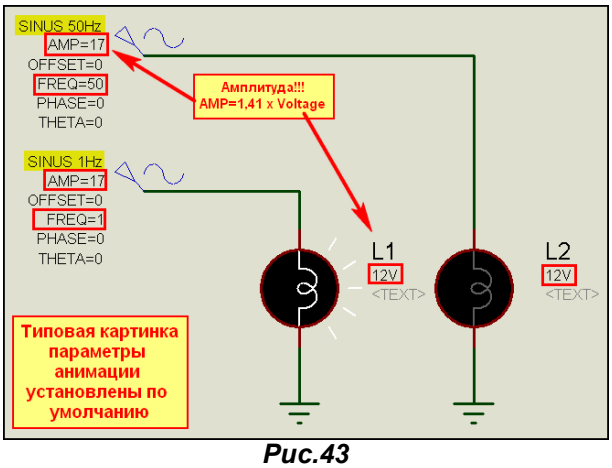


Рис.43

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

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

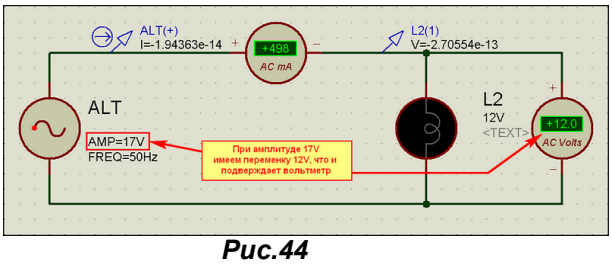


Рис.44

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

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

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

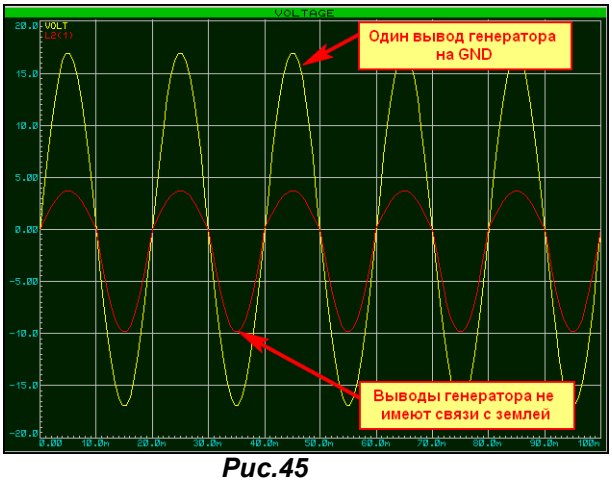


Рис.45

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

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

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

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

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

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

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

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

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

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

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

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

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

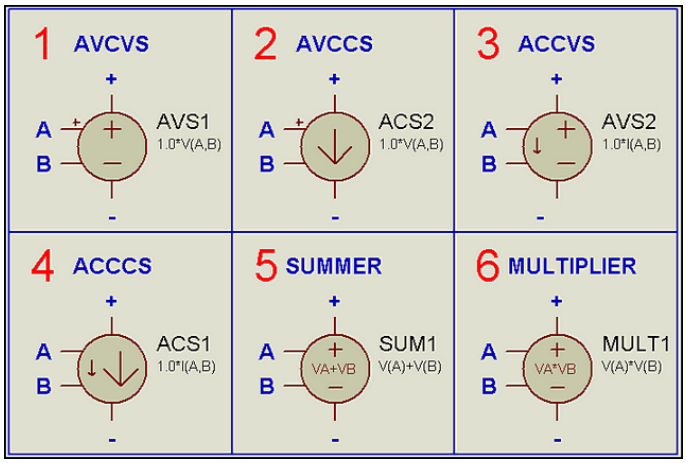


Рис.46

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

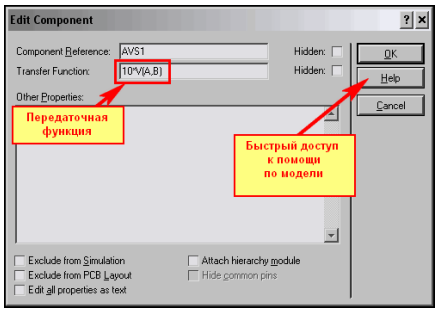


Рис.47

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

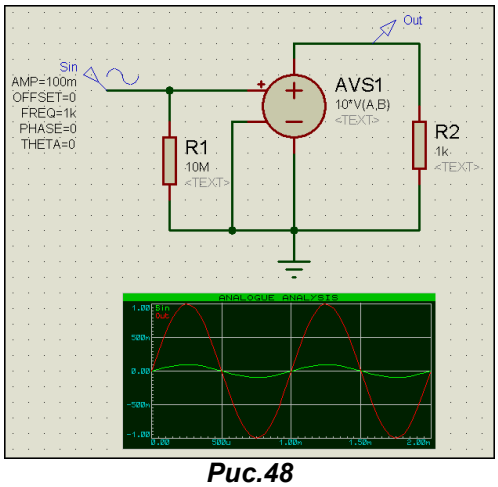


Рис.48

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

Допустимы следующие операторы математических действий:

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

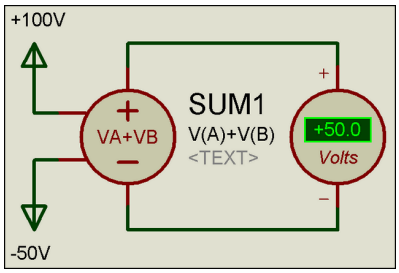


Рис.49

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

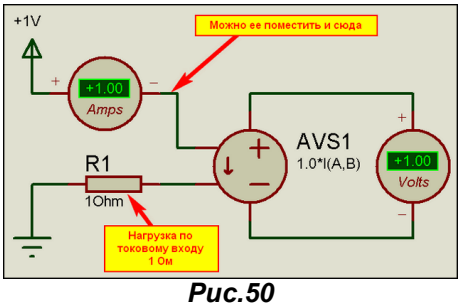


Рис.50

Теперь рассмотрим допустимые функции:

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

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

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

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

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

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

Рис.51

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

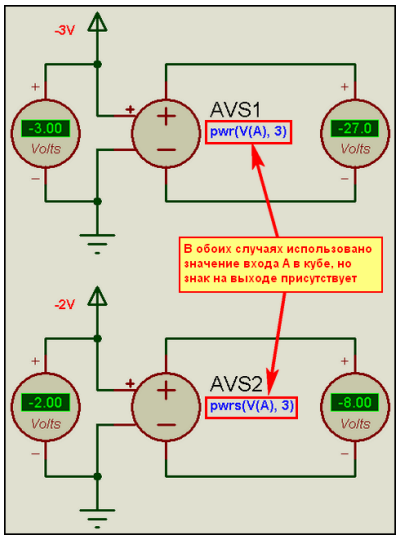


Рис.52

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

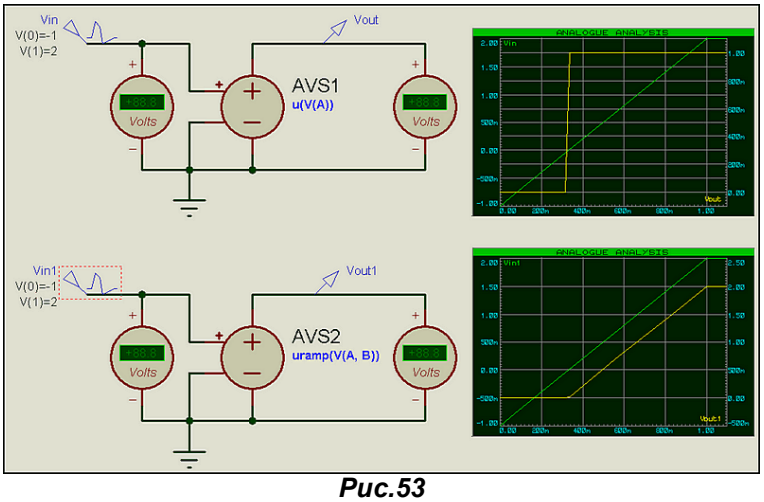


Рис.53

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

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

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

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

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

4.8. Аналоговые примитивы. Линейные управляемые источники сигналов.

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

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

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

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

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

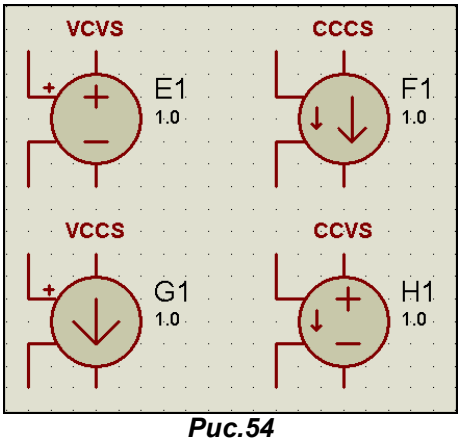


Рис.54

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

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

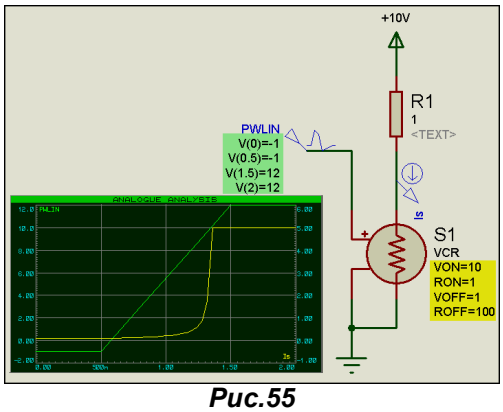


Рис.55

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

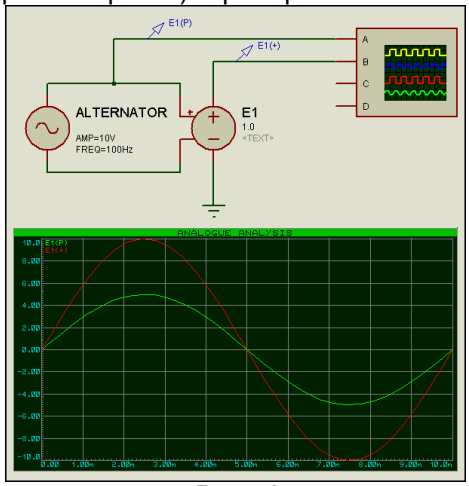


Рис.56

4.9. Применение Analogue и Mixed Graph для исследования сигналов.

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

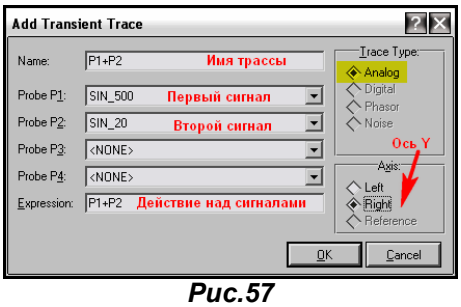


Рис.57

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

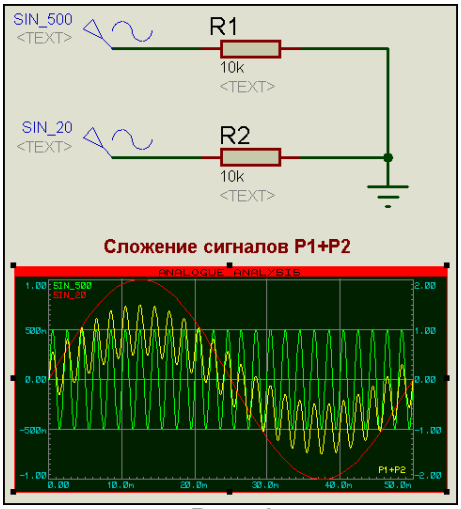


Рис.58

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

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

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

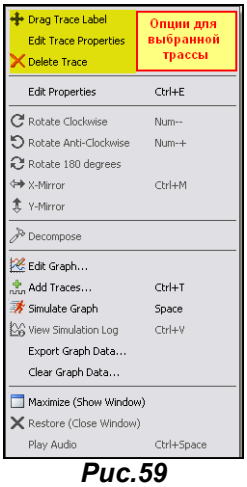


Рис.59

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

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

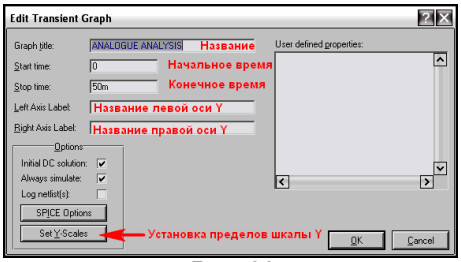


Рис.60

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

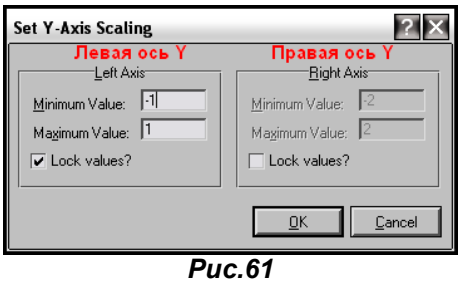


Рис.61

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

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

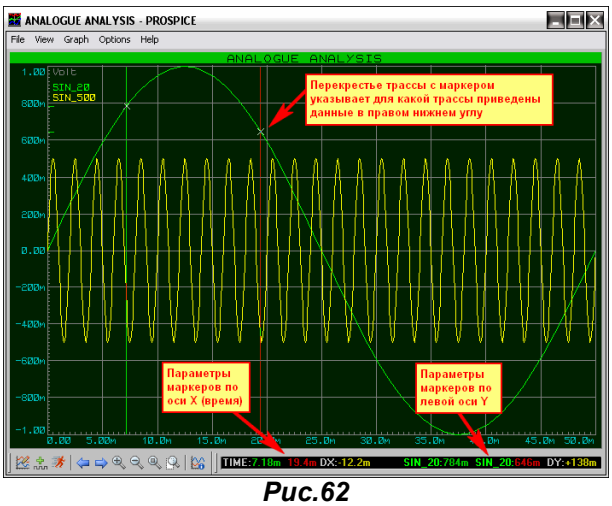


Рис.62

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

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

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

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

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

Rt=R+A*dt+B*dt2** где **dt=t-25**

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

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

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

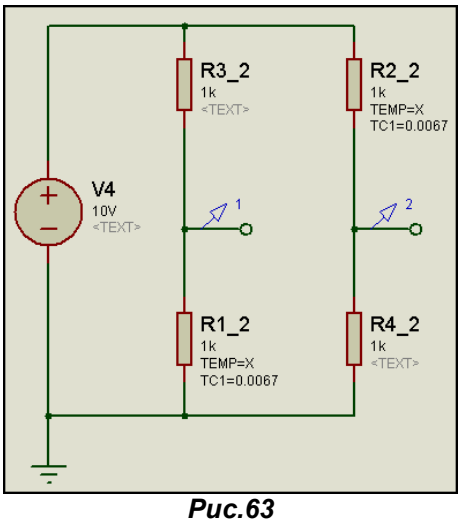


Рис.63

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

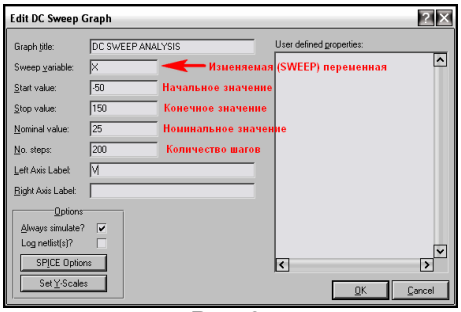


Рис.64

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

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

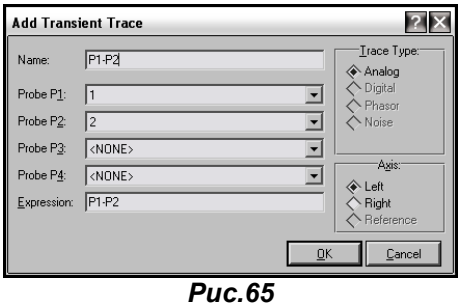


Рис.65

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

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

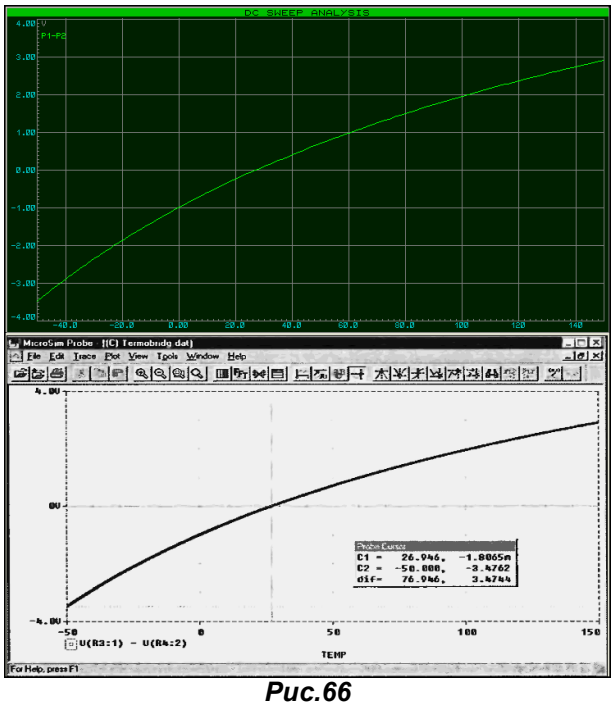


Рис.66

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

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

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

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

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

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

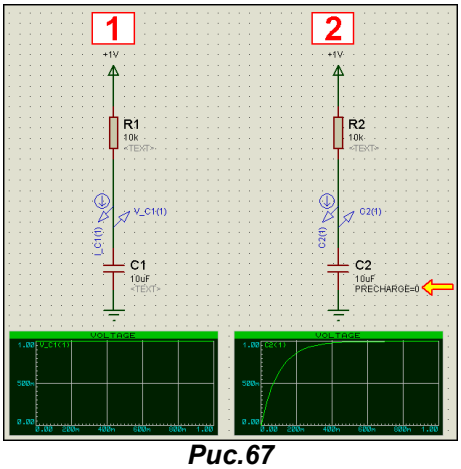


Рис.67

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

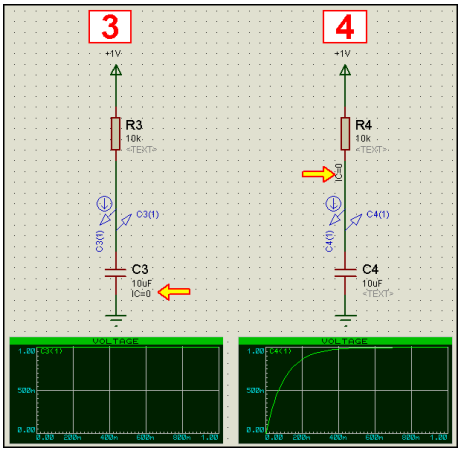


Рис.68

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

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

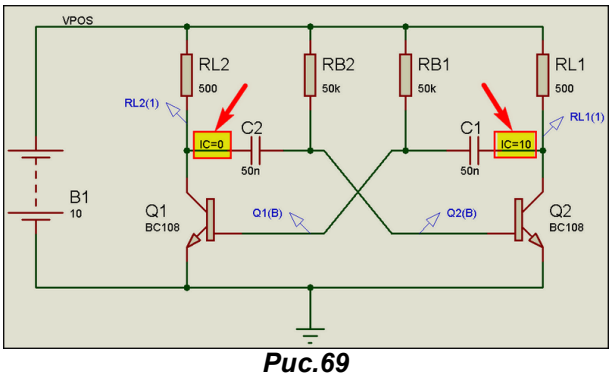


Рис.69

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

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

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

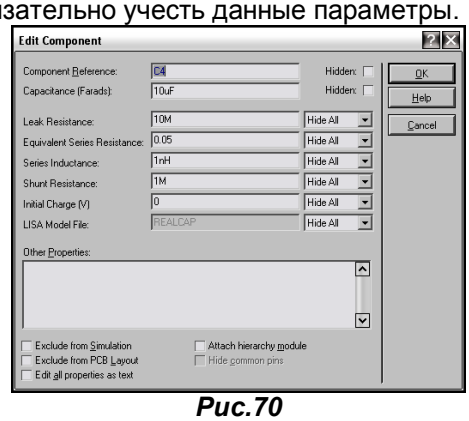


Рис.70

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

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

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

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

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

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

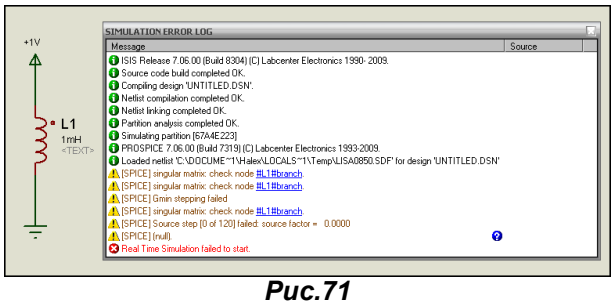


Рис.71

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

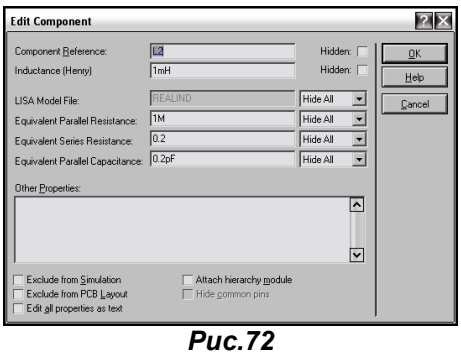


Рис.72

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

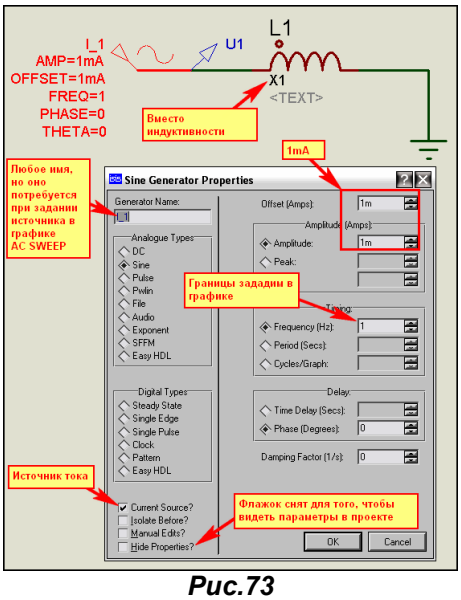


Рис.73

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

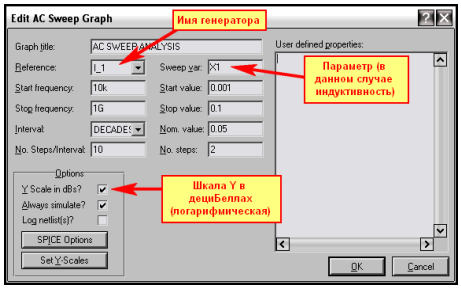


Рис.74

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

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

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

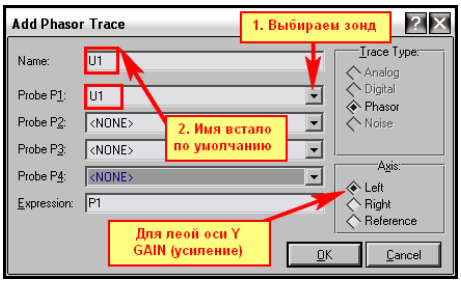


Рис.75

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

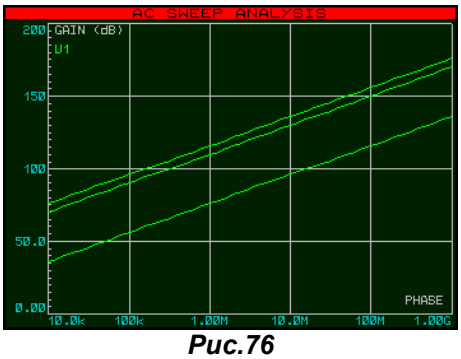


Рис.76

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

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

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

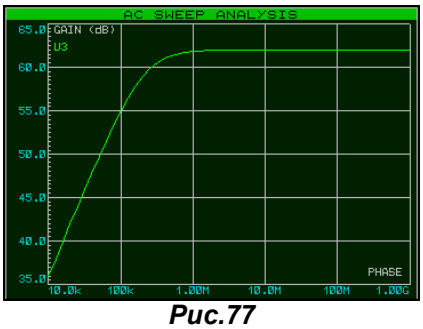


Рис.77

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

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

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

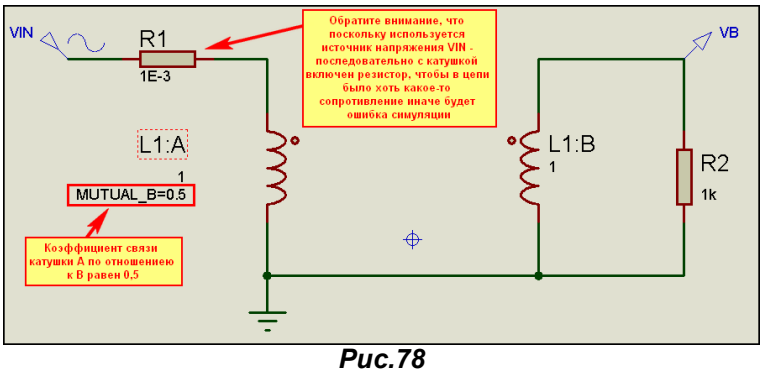


Рис.78

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

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

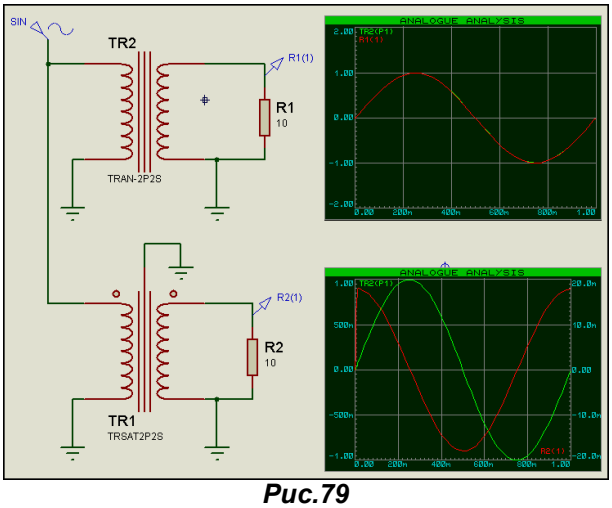


Рис.79

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

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

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

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

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

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

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

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

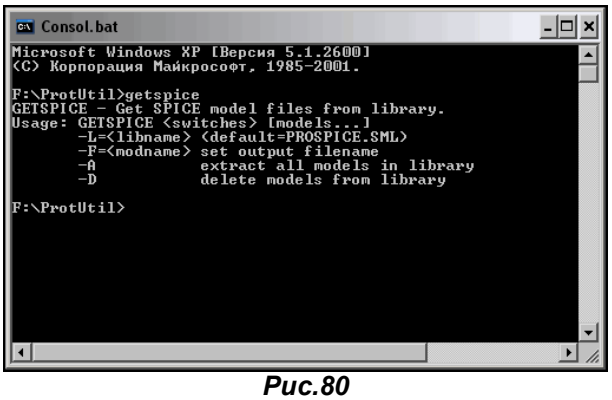


Рис.80

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

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

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

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

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

-A – извлекает все модели из указанной библиотеки.

-D – стирает модели в библиотеке.

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

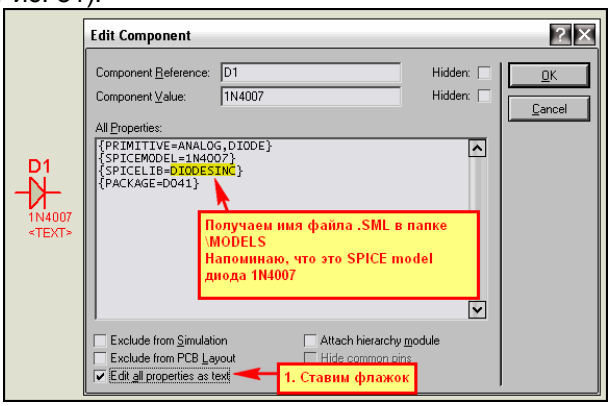


Рис.81

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

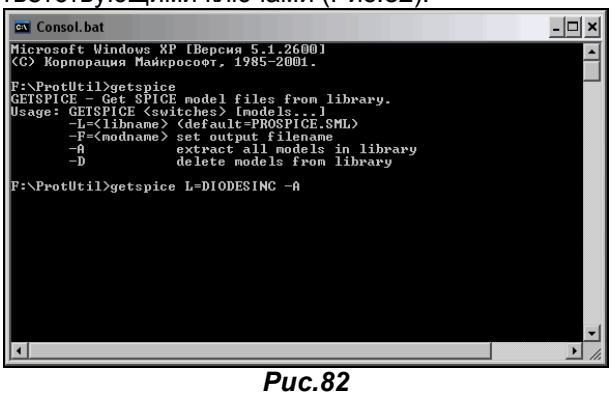


Рис.82

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

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

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

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

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

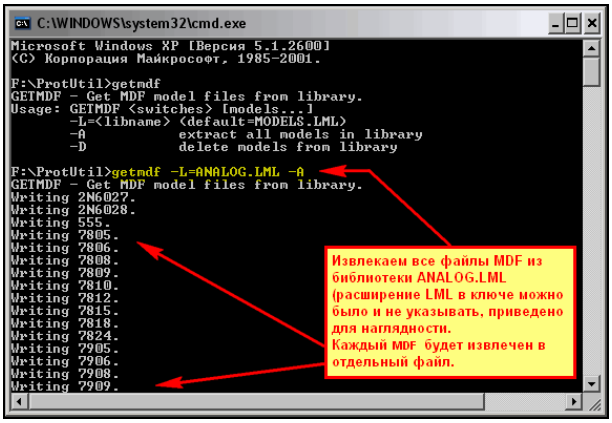


Рис.83

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

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

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

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

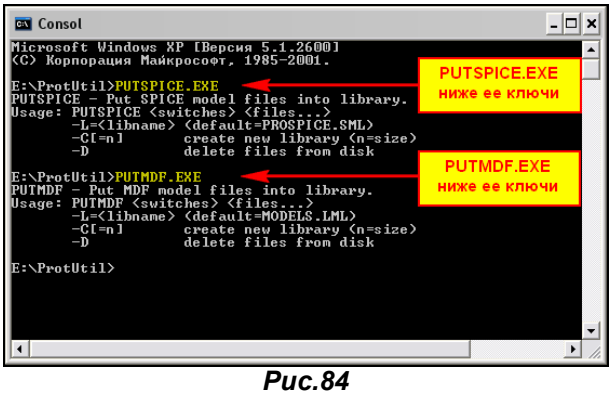


Рис.84

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

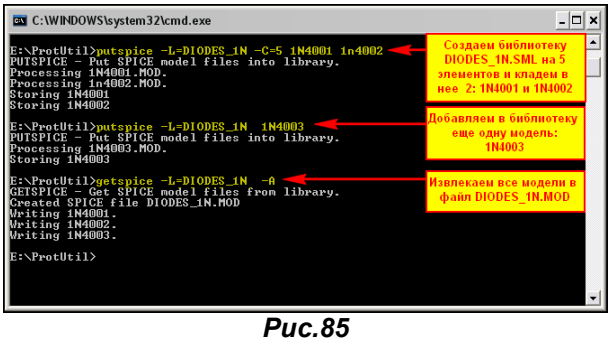


Рис.85

4.13. Примитивы управляемых ключей. Генераторы-двухполюсники.

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

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

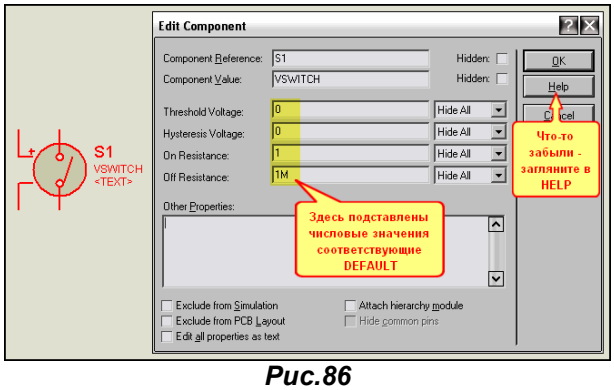


Рис.86

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

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

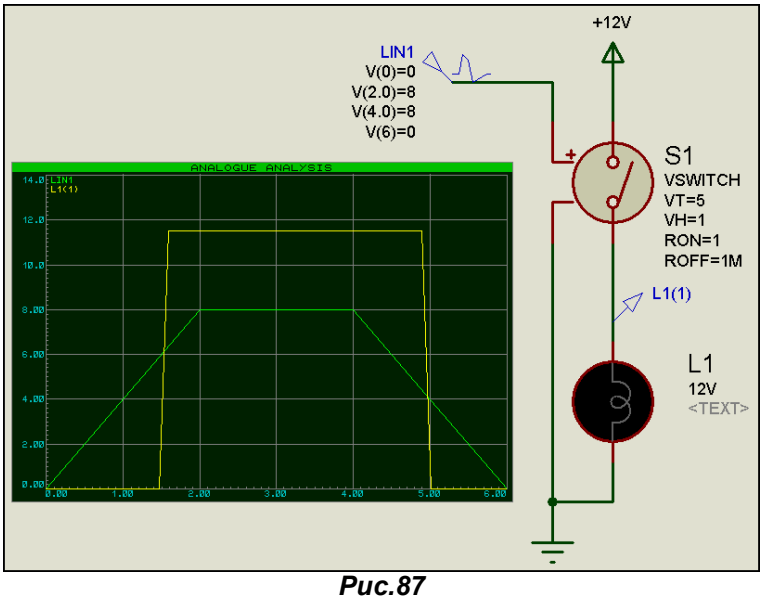


Рис.87

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

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

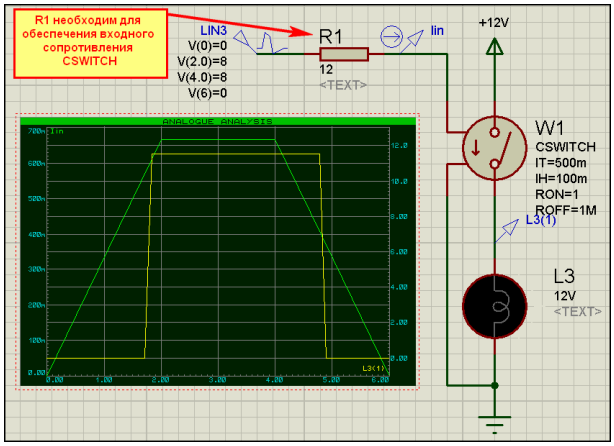


Рис.88

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

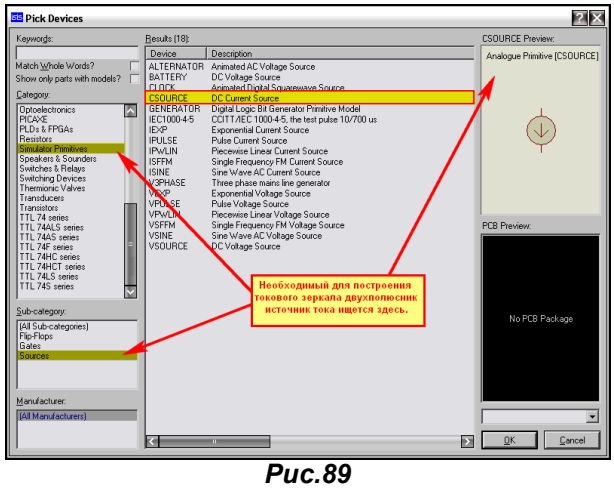


Рис.89

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

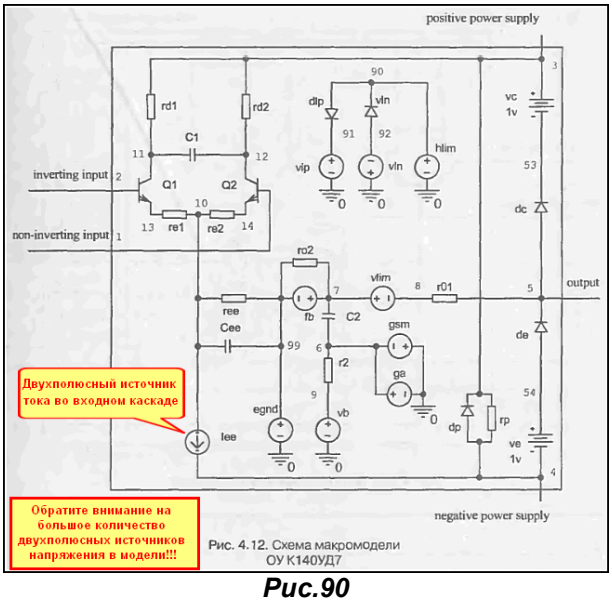


Рис.90

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

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

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

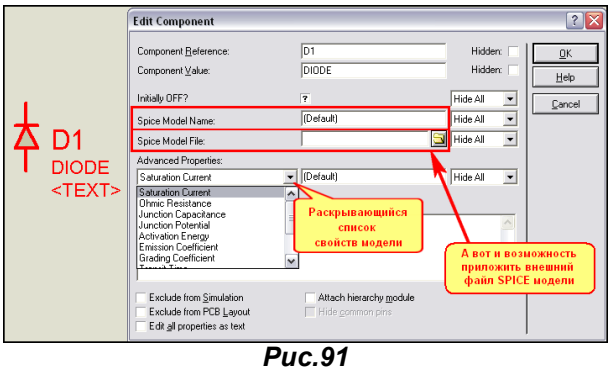


Рис.91

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

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

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

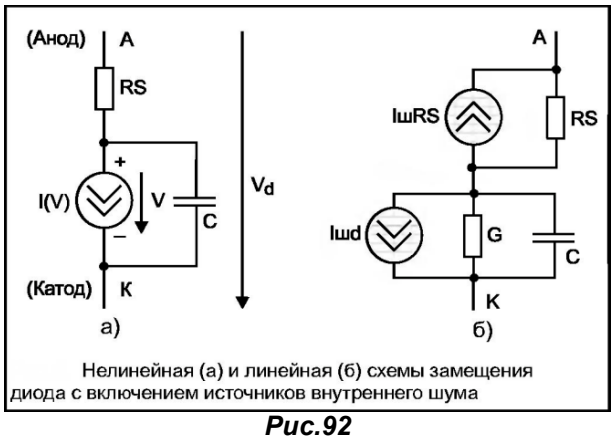
- **Initial device voltage** **IC** **(-)** Начальное напряжение *(можно задать только вручную, в списке его нет, т.к. это фишка Протеуса – еще одна трактовка IC)*.
- **Instance temperature** **TEMP** **(27)** Реальная температура диода [ºC] *(изменяется при температурном моделировании, как и для остальных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

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

Параметры из раскрывающегося списка:

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

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



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

Рис.92

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

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

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

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

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

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

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

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

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

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

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

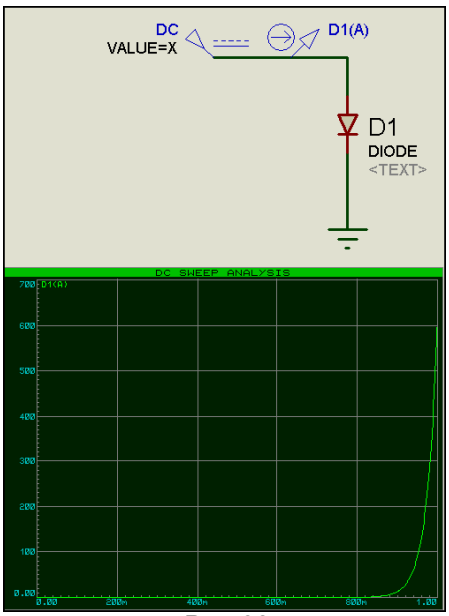


Рис.93

Здесь уместны несколько замечаний:

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

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

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

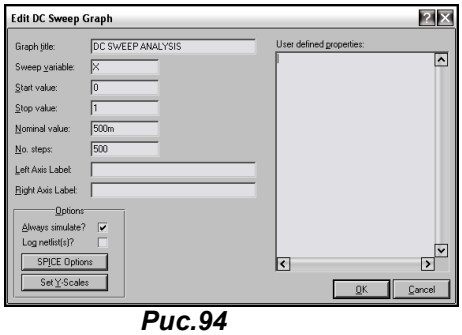


Рис.94

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

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

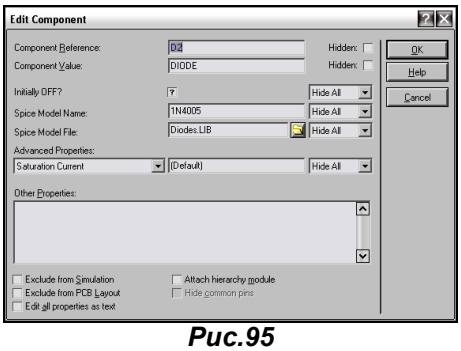


Рис.95

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

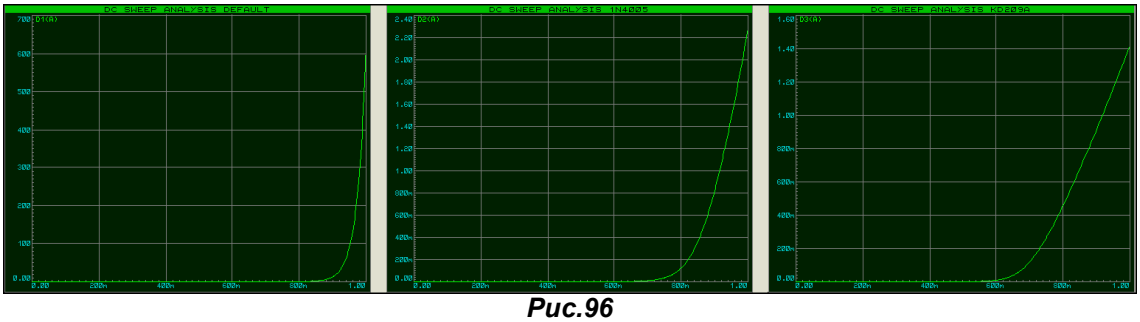


Рис.96

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

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

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

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

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

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

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

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

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

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

Отсюда получим формулу для емкости.

$$Сд = \frac{Iд}{10^7}$$

или окончательно

Сд(пф) = 0,1Iд (мкА)

Рис.97

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

Ну а теперь подведем итог изложенному материалу:

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

Ну и конечно материал от О. Петракова:

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

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

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

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

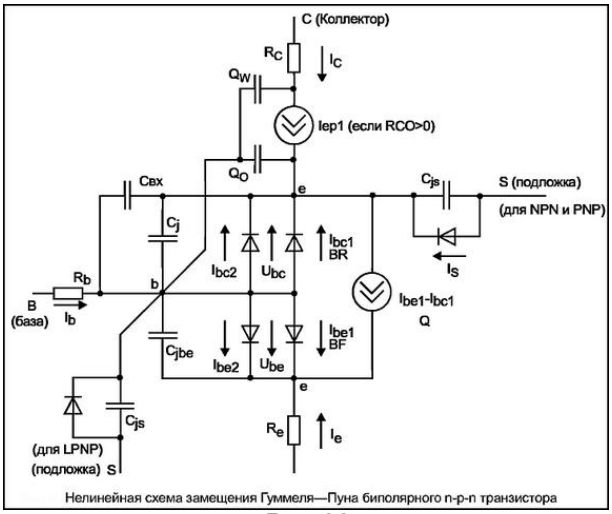


Рис.98

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

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

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

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

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

Ну и два, уже известных нам температурных коэффициента.

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

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

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

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

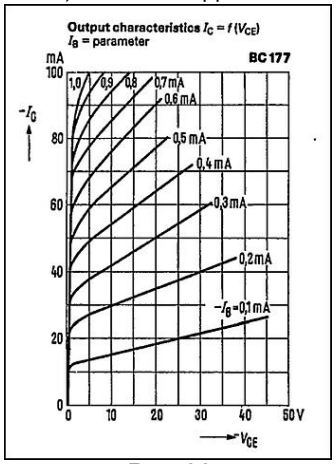


Рис.99

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

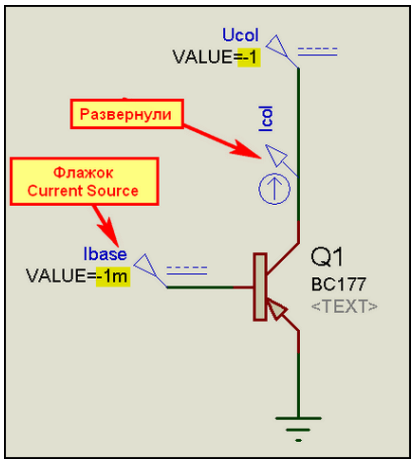


Рис.100

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

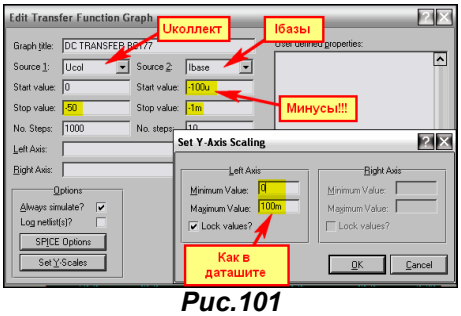


Рис.101

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

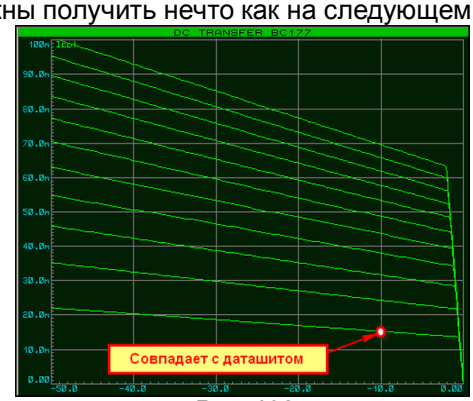


Рис.102

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

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

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

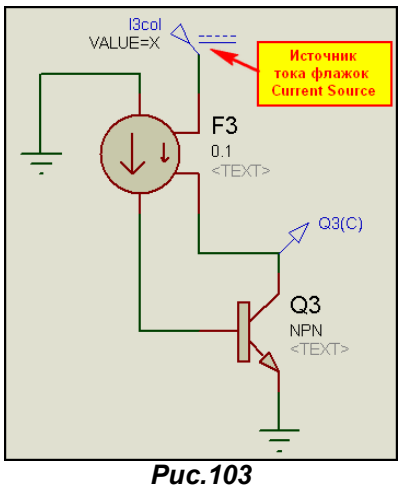


Рис.103

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

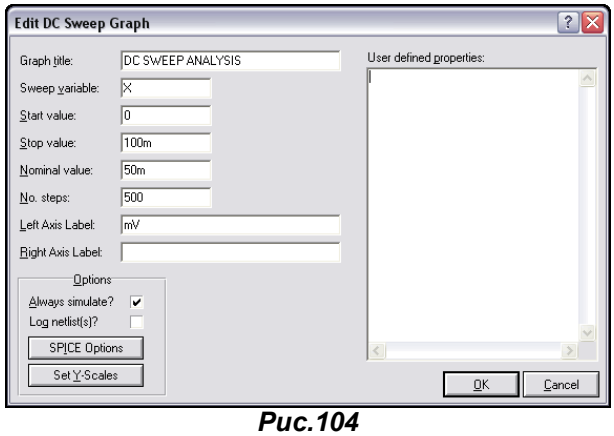


Рис.104

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

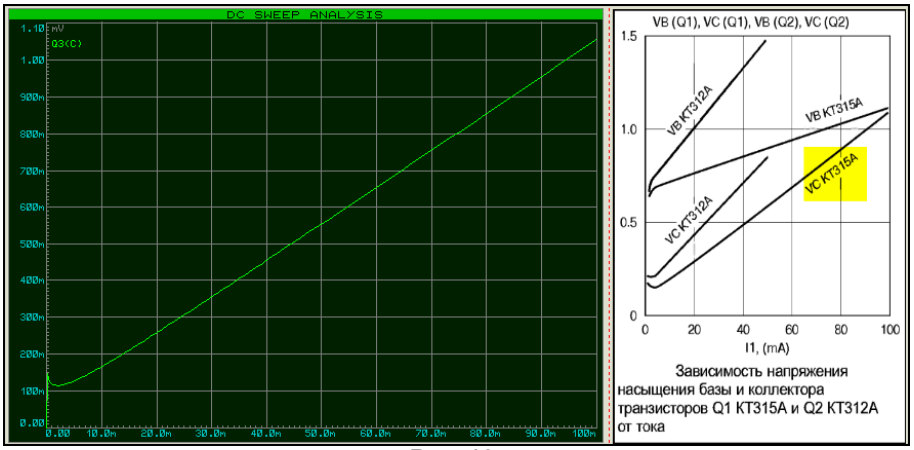


Рис.105

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

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

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

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

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

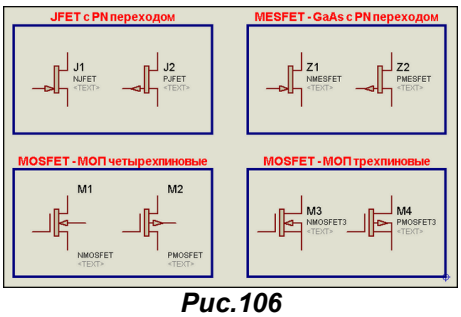


Рис.106

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

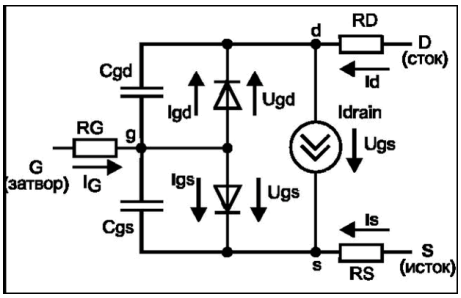


Рис.107

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

Параметры характерные только для ISIS:

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

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

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

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

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

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

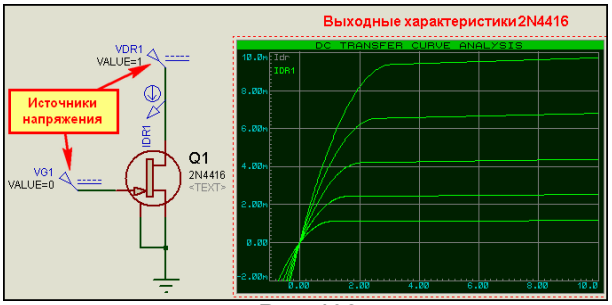


Рис.108

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

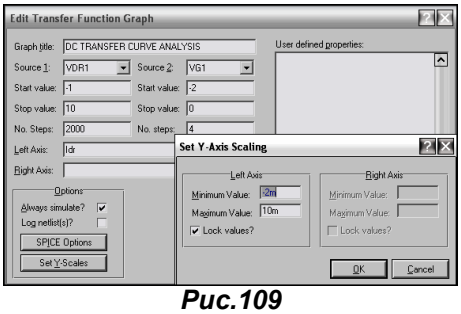


Рис.109

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

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

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

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

PRIMITIVE=ANALOG, NMOSFET
LEVEL=5

или:

PRIMITIVE=ANALOG, NBSIM2

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

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

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

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

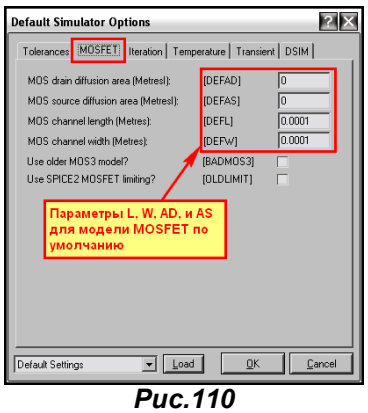


Рис.110

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

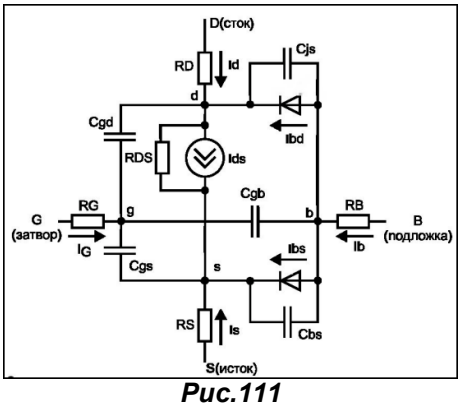


Рис.111

Параметры, выделенные в отдельные строки:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

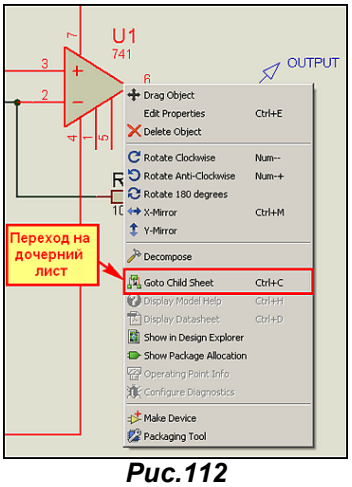


Рис.112

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

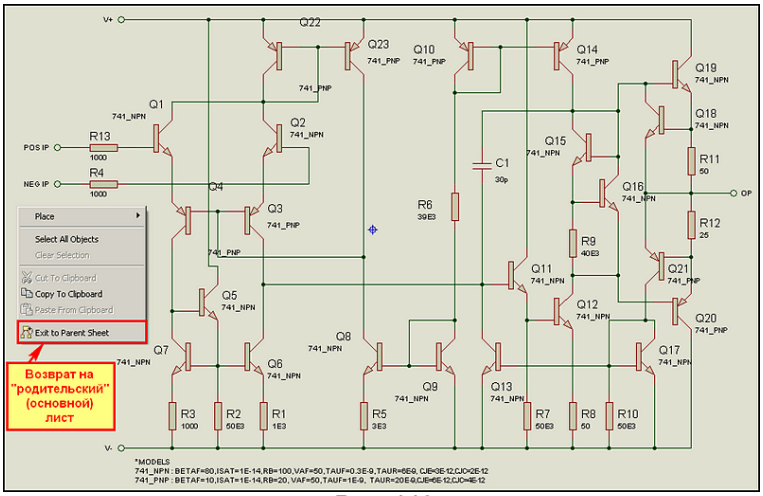


Рис.113

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

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

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

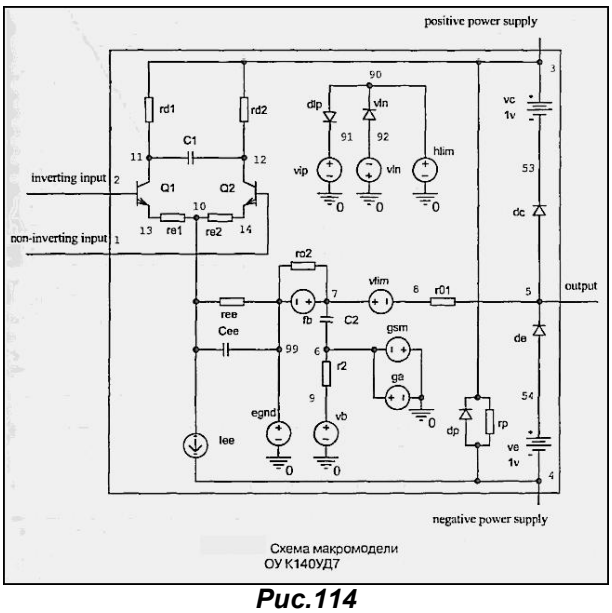


Рис.114

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

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

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

Например:

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

или

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

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

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

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

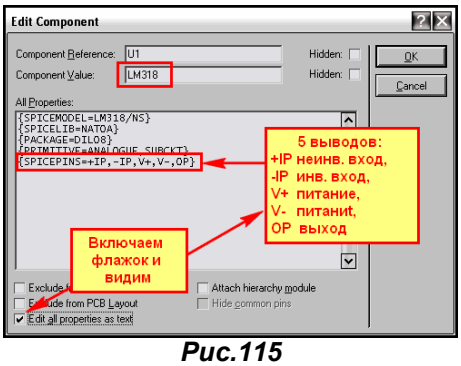


Рис.115

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

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

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

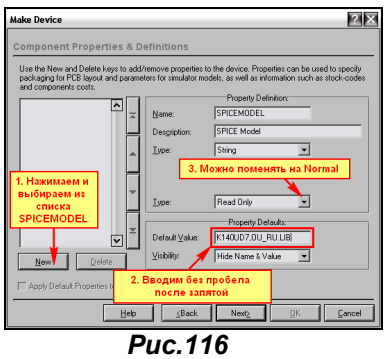


Рис.116

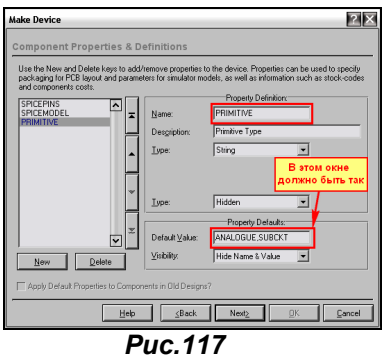


Рис.117

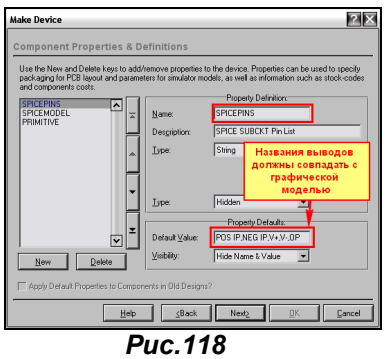


Рис.118

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

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

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

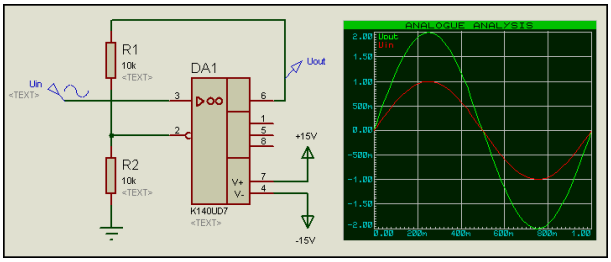


Рис.119

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

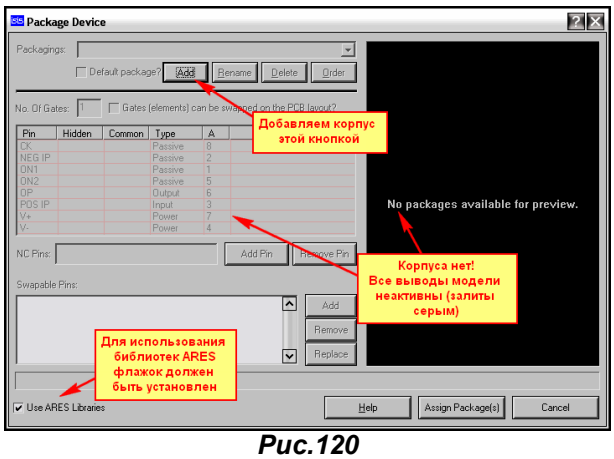


Рис.120

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

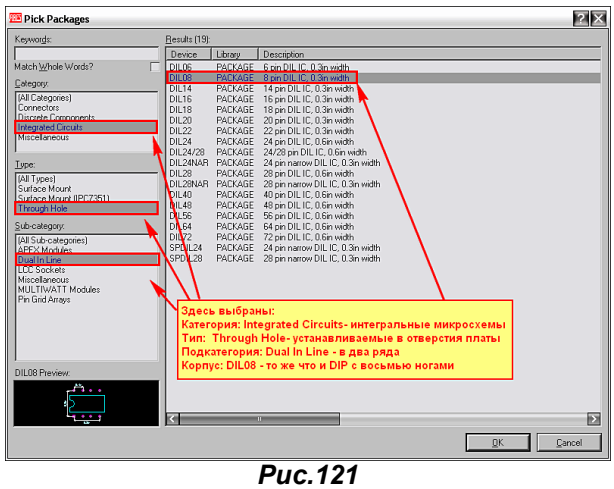


Рис.121

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

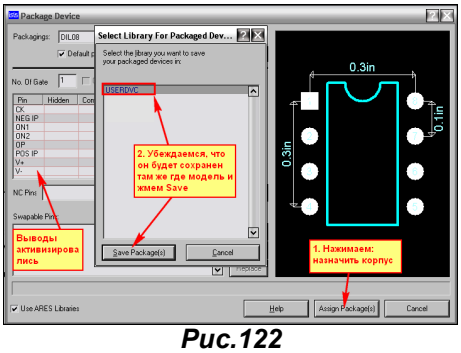


Рис.122

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

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

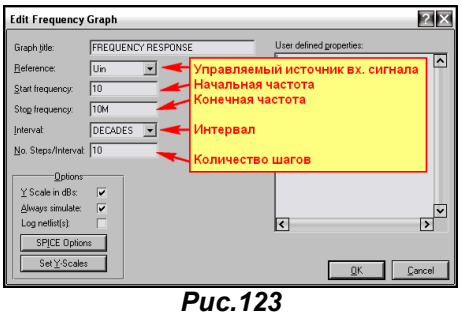


Рис.123

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

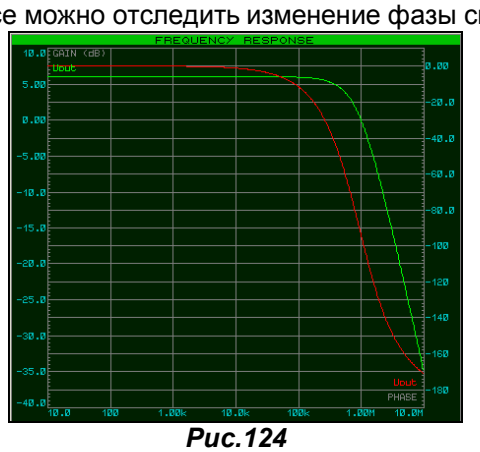


Рис.124

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

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

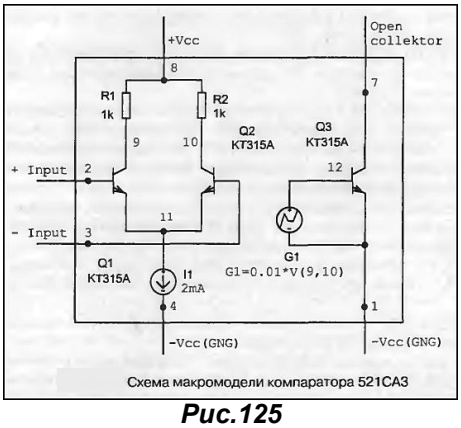


Рис.125

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

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

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

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

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

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

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

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

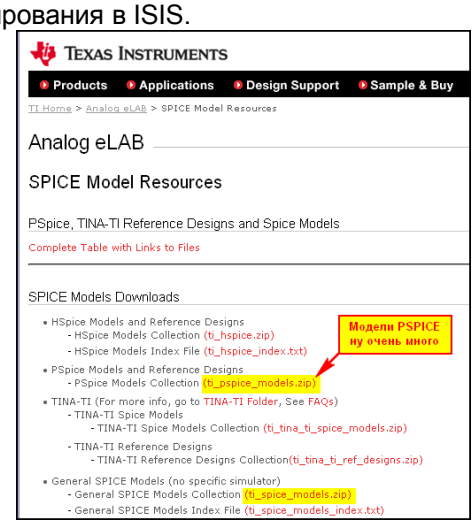


Рис.126

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

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

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

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

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

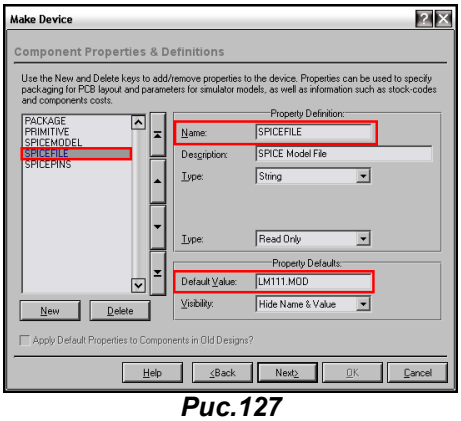


Рис.127

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

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

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

PUTSPICE –L=TEX301_1.LML –C=200

После чего другой командой:

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

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

Рис.128

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

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

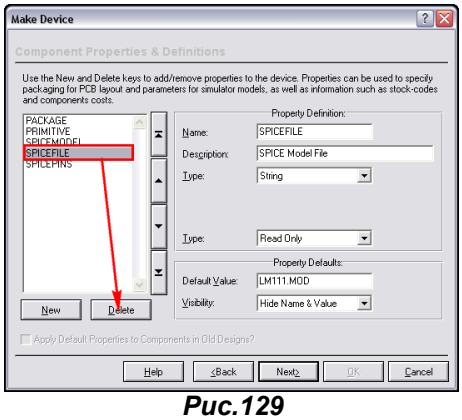


Рис.129

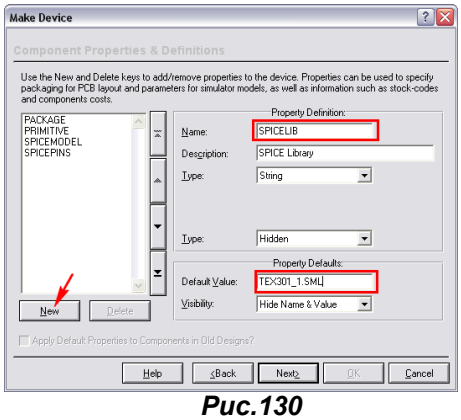


Рис.130

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

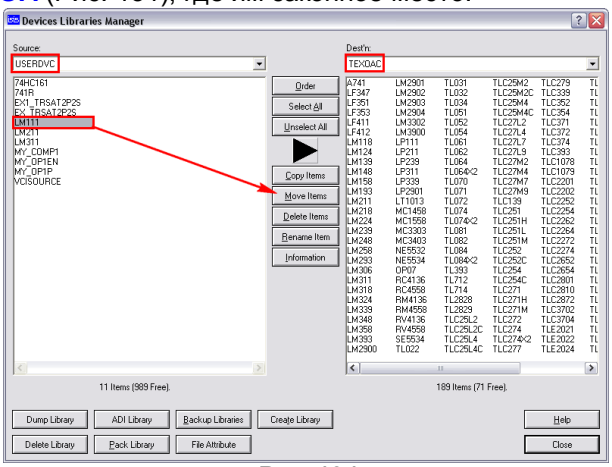


Рис.131

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

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

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

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

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

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

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

****

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