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

Содержание:

5. Иерархия проектов Протеуса.

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

До сих пор мы в основном строили дизайны в ISIS на одном листе проекта. Также мы знаем, что всегда можно добавить дополнительные листы с помощью меню **Design** -> **New Sheet**. При этом последующие листы могут иметь отличные от первого форматы. Например, первый лист имеет формат А4, второй – А3, третий – А4, а четвертый – А1. Это так называемый «плоский» проект, поскольку все листы принадлежат к одному уровню. Если рассматривать бумажный аналог, то это как бы один том многотомника. Мы можем листать страницы и переходить с первой на вторую, пятую десятую и назад. В пределах плоского проекта действует глобальная нумерация элементов и цепей (проводов) и шин питания. Все это хорошо, когда наш проект содержит десяток-другой элементов. Но, представим себе, что мы разрабатываем многополосный фильтр или даже просто стереофонический усилитель с двумя идентичными каналами. И что? Будем рисовать одно и то же дважды, трижды и т.д. Нет, конечно. Как и в обычных бумажных вариантах, мы можем структурировать наш проект. Например, основной лист будет содержать только блок схему, с которой будут ссылки на другие листы, содержащие отдельные принципиальные схемы блоков (модулей). Мало того, мы можем применить какие-нибудь нестандартные (пока назовем их так) компоненты, например – датчики, которые тоже будут иметь собственные принципиальные схемы. Программа **ISIS** позволяет связать все это в единое целое, да еще и заставить работать в виртуальном виде. Количество вложенных уровней иерархии может достигать восьми. Графически это можно представить так, как на рисунке 1.

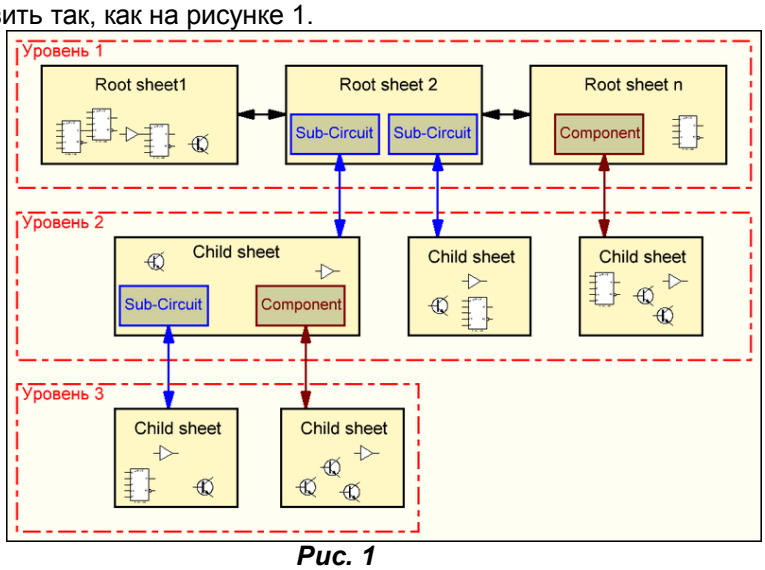


Рис. 1

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

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

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

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

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

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

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

Graph Based Simulation\DAC0808.DSN – микросхема U1 также обладает дочерним листом, но здесь уже смешанное цифро-аналоговое поведенческое (т.е. структура воспроизведена не тотально) моделирование.

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

Возврат к содержанию

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

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

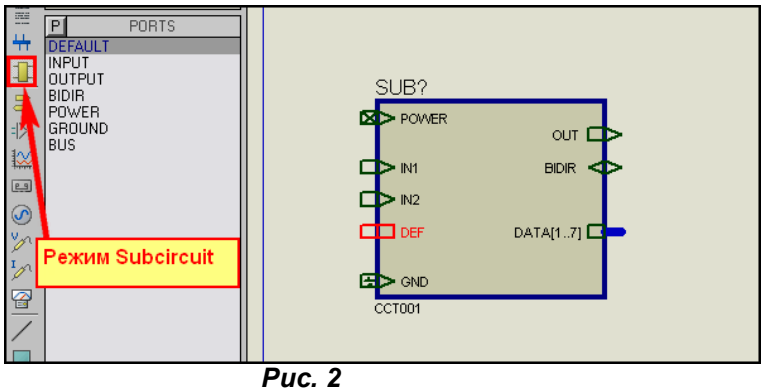


Рис. 2

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

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

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

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

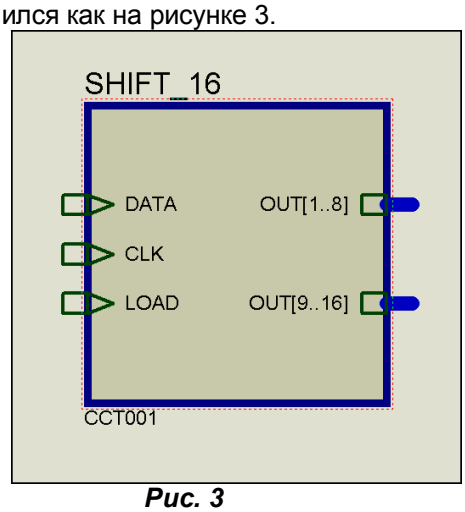


Рис. 3

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

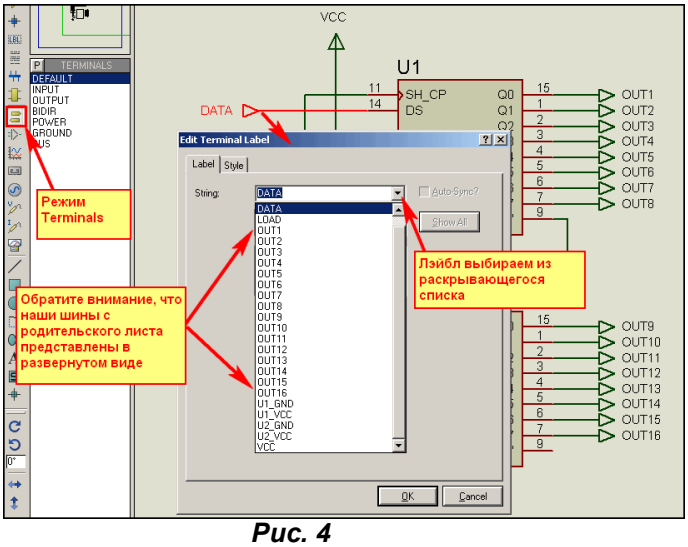


Рис. 4

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

Рис. 5

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

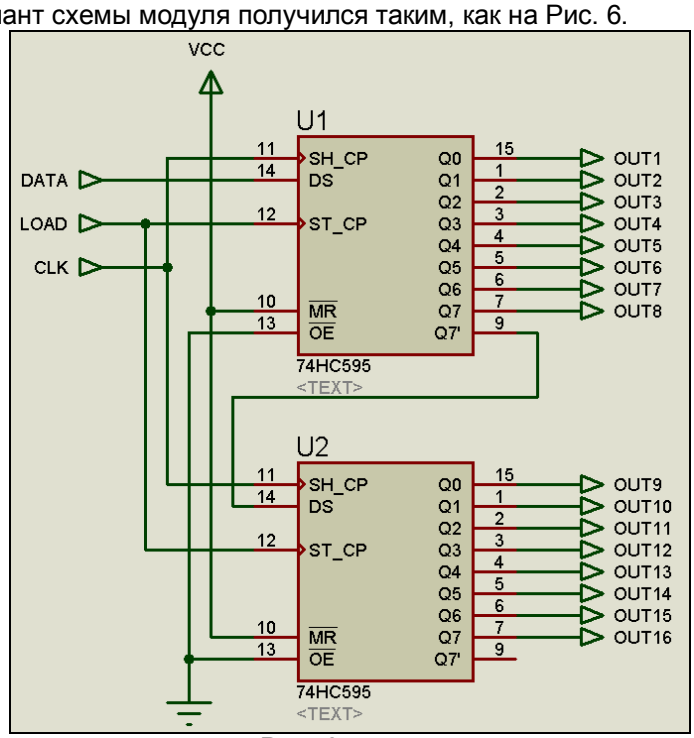


Рис. 6

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

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

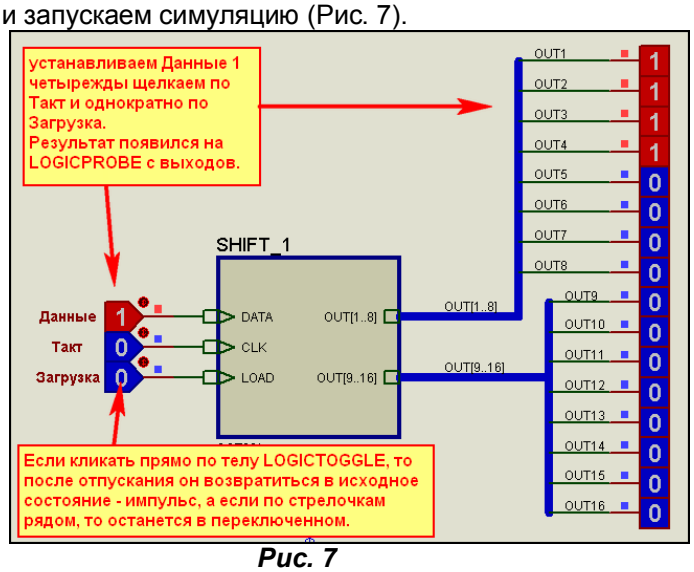


Рис. 7

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

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

Возврат к содержанию

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

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

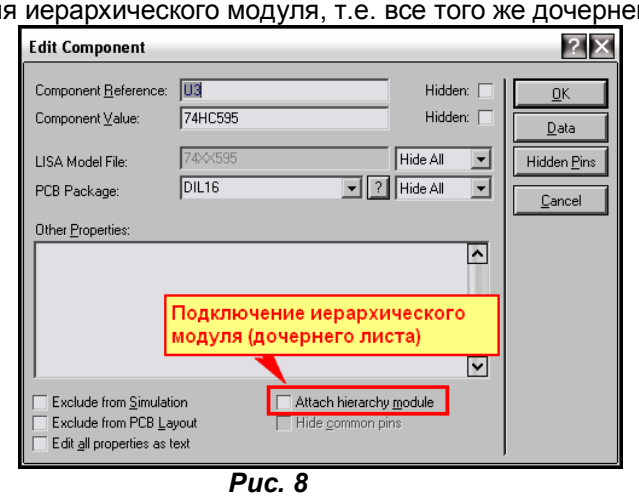


Рис. 8

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

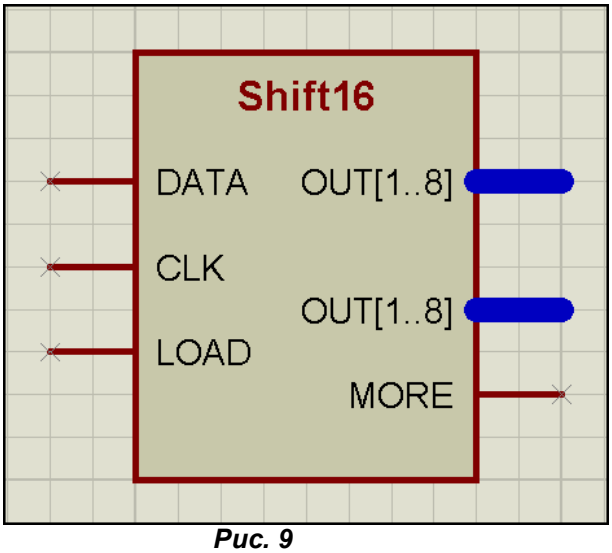


Рис. 9

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

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

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

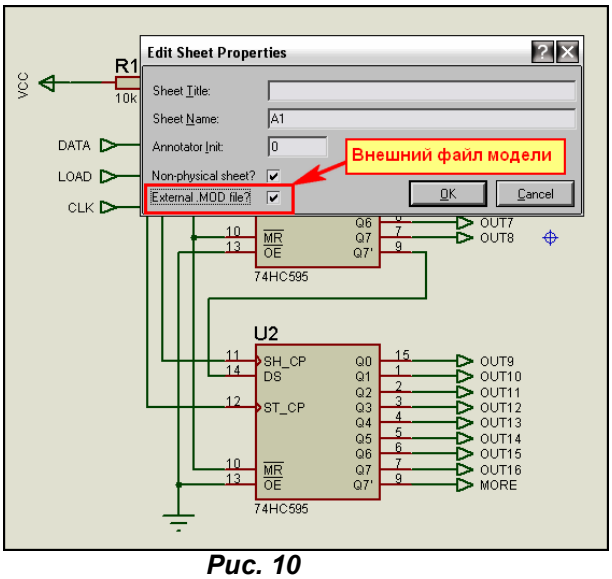


Рис. 10

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

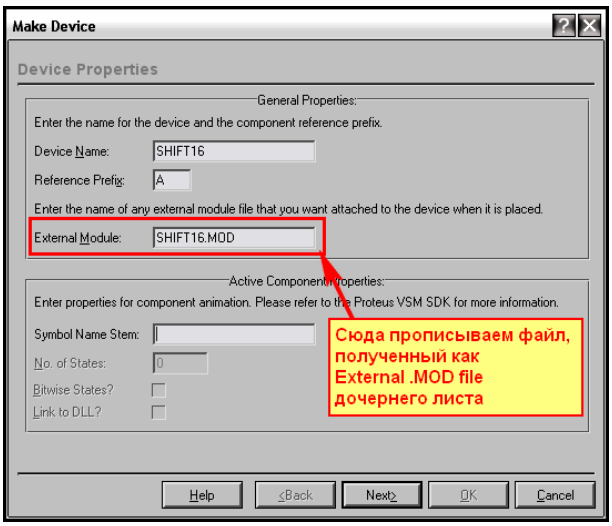


Рис. 11

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

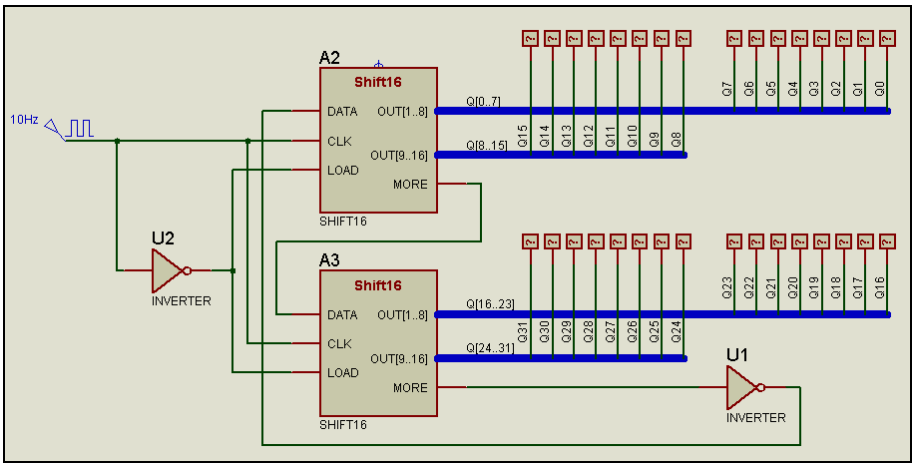


Рис. 12

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

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

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

Возврат к содержанию

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

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

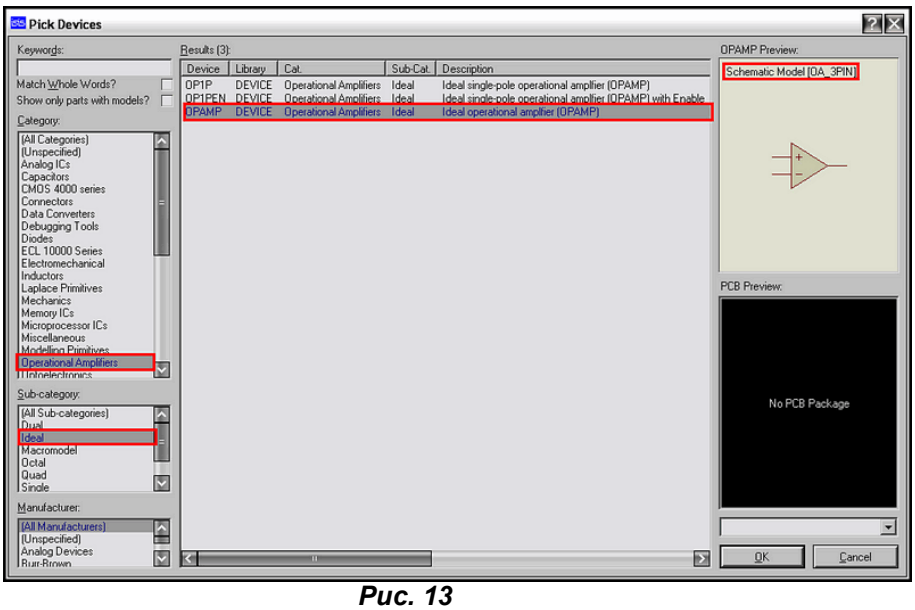


Рис. 13

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

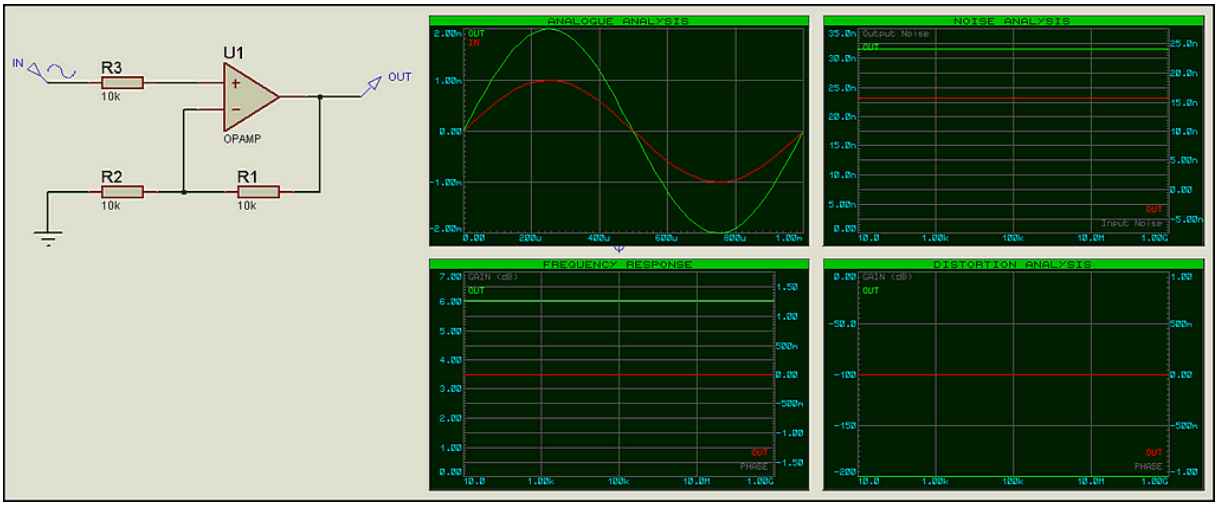


Рис. 14

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

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

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

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

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

GETMDF.EXE –L=ANALOG.LML –A

Можно и не указывать расширения:

GETMDF –L=ANALOG –A

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

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

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

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

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

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

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

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

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

На что в списке компонентов хотелось бы обратить внимание.

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

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

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

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

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

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

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

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

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

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

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

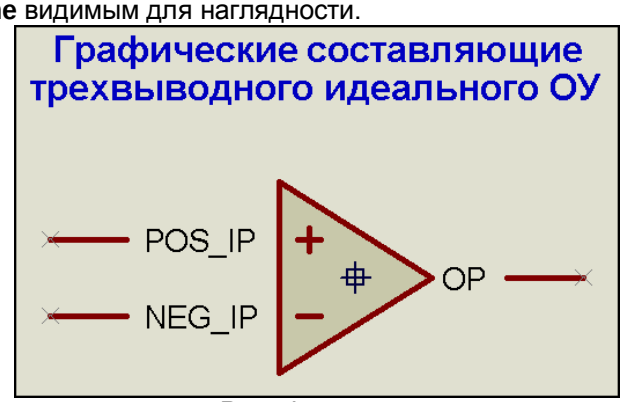


Рис. 15

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

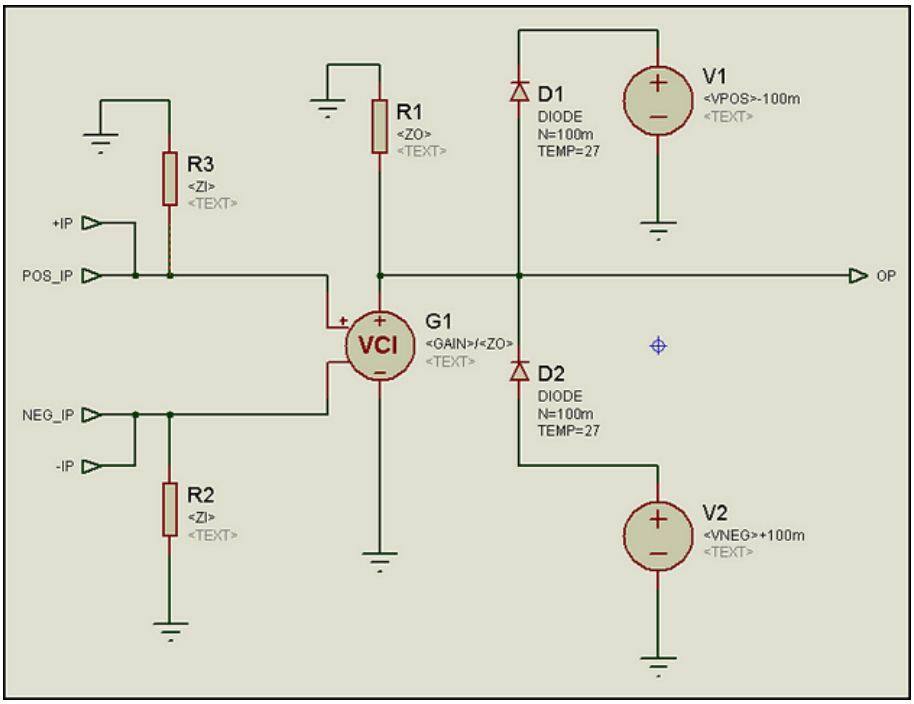


Рис16.

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

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

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

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

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

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

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

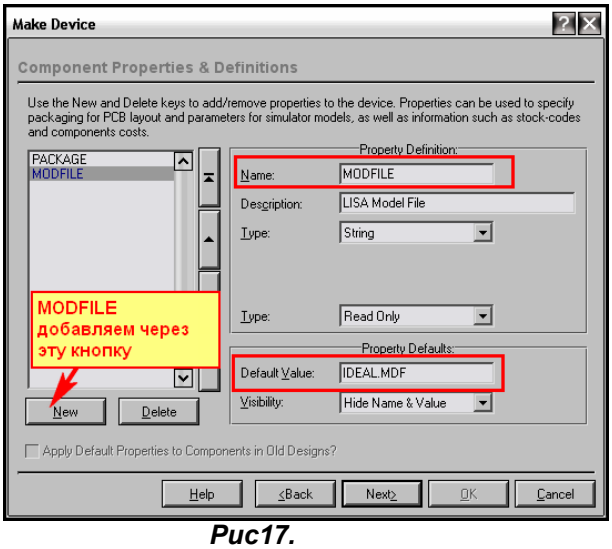


Рис17.

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

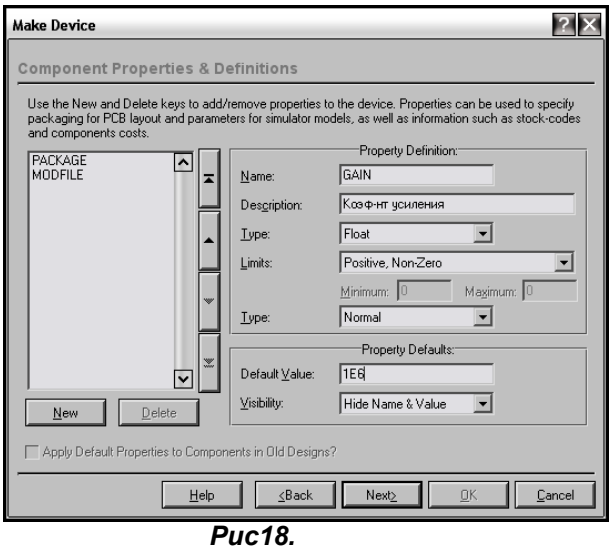


Рис18.

Итак, рассмотрим подробно.

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

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

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

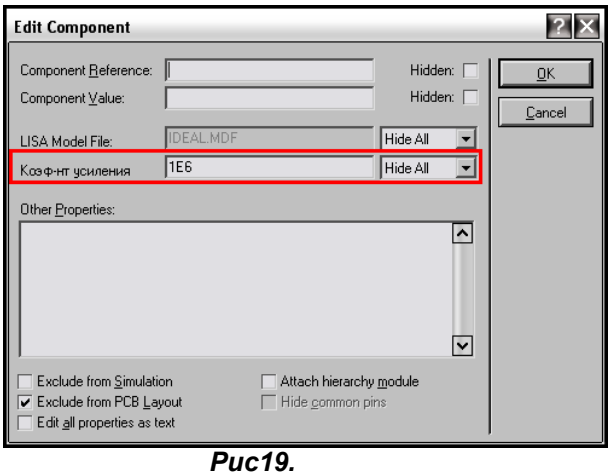


Рис19.

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

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

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

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

Возврат к содержанию

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

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

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

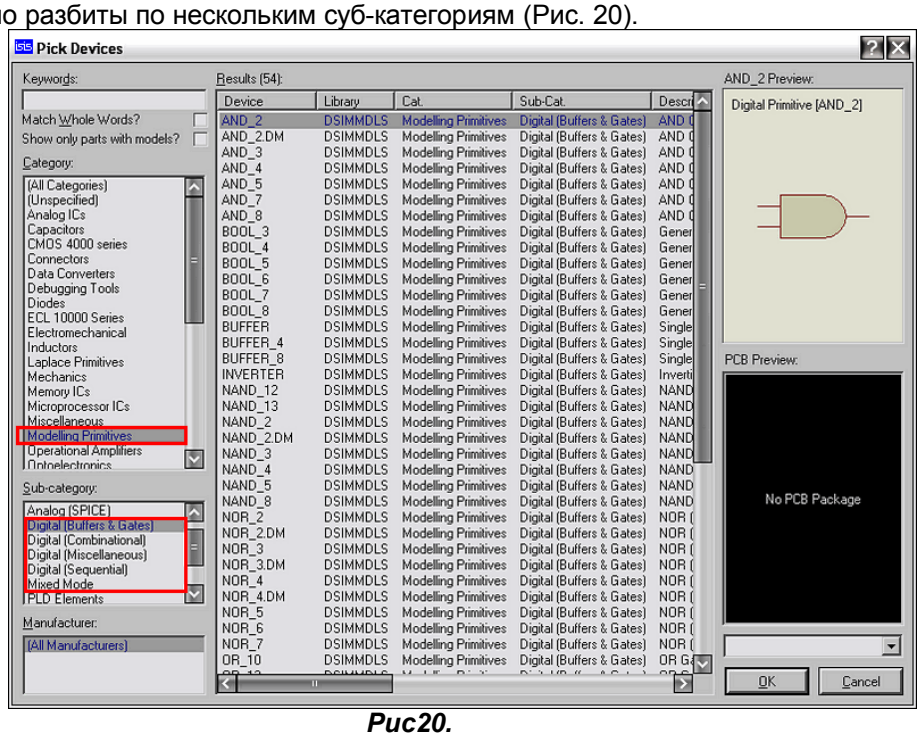


Рис20.

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

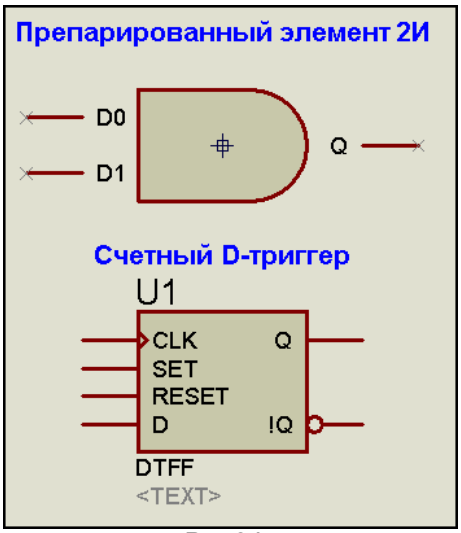


Рис21.

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

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

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

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

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

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

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

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

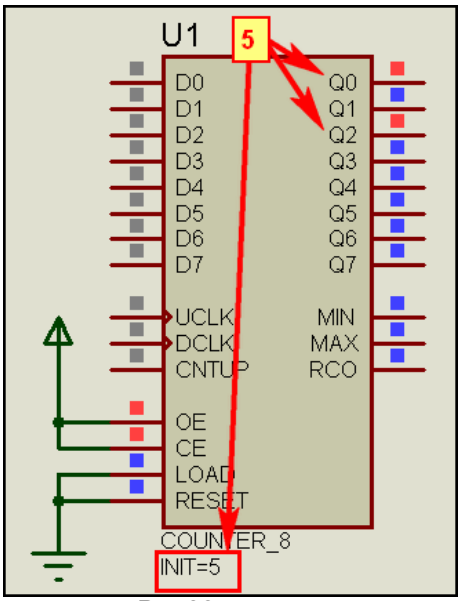


Рис22.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

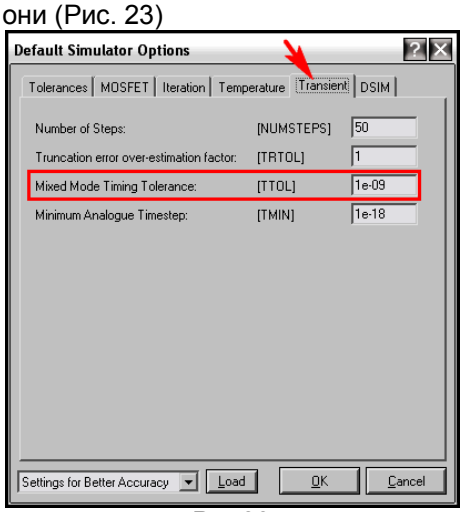


Рис23.

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

Возврат к содержанию

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

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

Рассмотрим для примера первую строчку CMOS, описывающую КМОП логику:

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

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

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

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

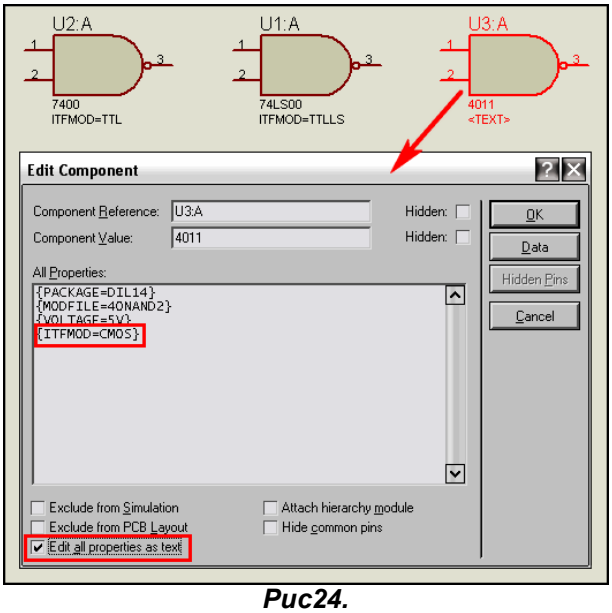


Рис24.

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

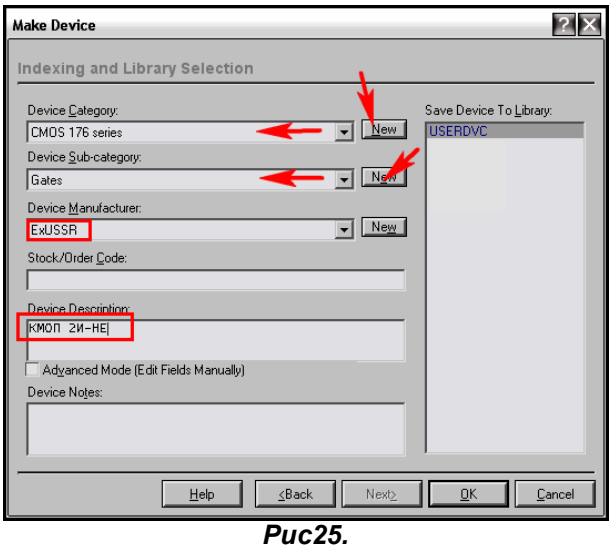


Рис25.

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

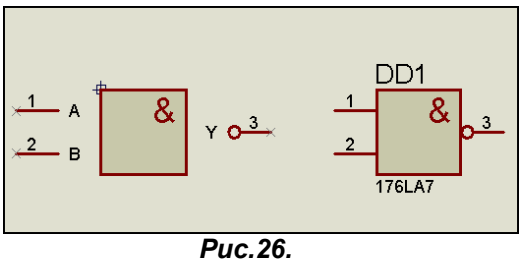


Рис.26.

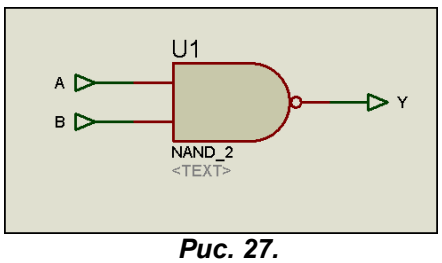


Рис. 27.

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

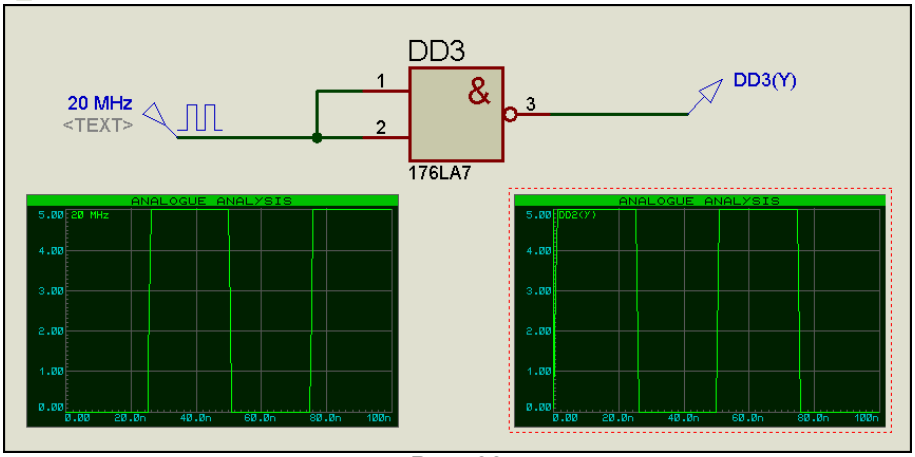


Рис. 28.

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

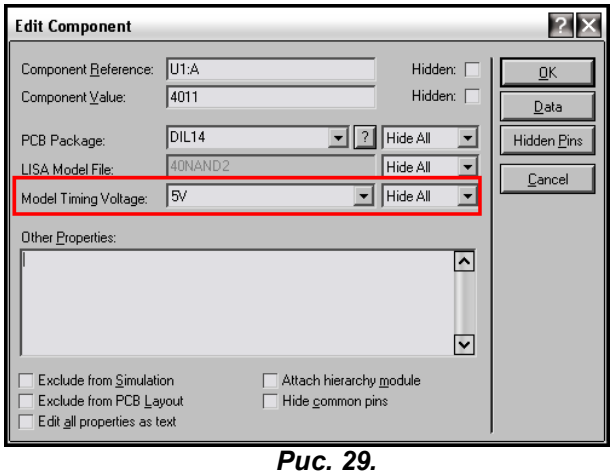


Рис. 29.

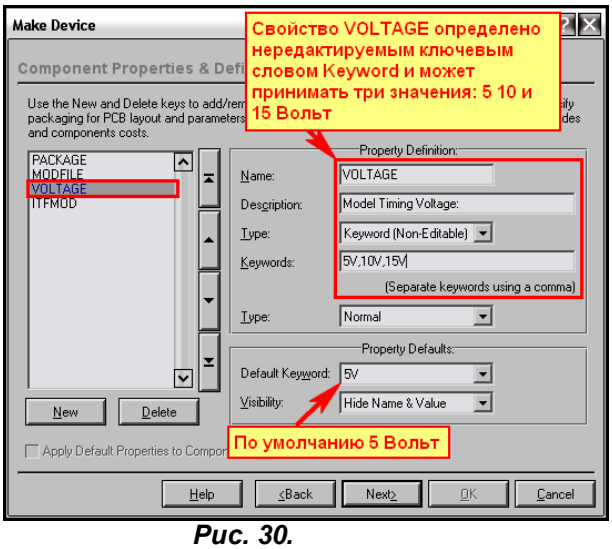


Рис. 30.

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

```
*PROPERTIES,1
TGQ=?

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

*MODELDEFS,0

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

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

A,2
A,IT
U1,IP,D0

B,2
B,IT
U1,IP,D1

*GATES,0
```

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

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

Теперь остановимся на задержках фронтов. Числовые значения прописаны именно в **MAPPINGS** и зависят от того, что мы (подчеркиваю – мы, а не сам симулятор) выберем в окне **Model Timing Voltage**. По умолчанию используются максимальные задержки, соответствующие приведенным в справочных данных для напряжения питания 5 Вольт. Надеюсь, теперь вам стало понятно – как использовать этот параметр на практике. Ну а мы далее рассмотрим – как применить это к нашей модели.

В разделе ***PARTLIST** сиротливо притулился один элемент **NAND_2** – все, как и у нас на дочернем листе **176LA7**. Вот только в этой строке прописаны еще и параметры не с конкретными значениями, а находящимися в угловых скобках, т.е. переменными. И подставляются они туда симулятором как раз из **MAPPINGS**. Вот он – момент истины! Но «не все так просто в доме…» Лабцентра. Помните чудесное превращение скрипта ***DEFINE** на дочернем листа в ***PROPERTIES** файла MDF при компиляции идеального ОУ. Тут нас ожидает такой же сюрприз. Скрипт на дочернем листе для таблицы соответствий должен начинаться с оператора **MAP ON**. Поскольку это оператор, как и **DEFINE**, перед ним должна стоять звездочка. Заходим на дочерний лист нашей модели **176LA7** и в свойствах **NAND_2** прописываем аналогично тому, что мы видели в MDF для **4011** задержки и свойство **SCHMITT**. Проще всего это сделать все же руками в окне **Advanced Properties**, поставив (да можно и не ставить) флажок **Edit all Properties as text** (Рис. 31).

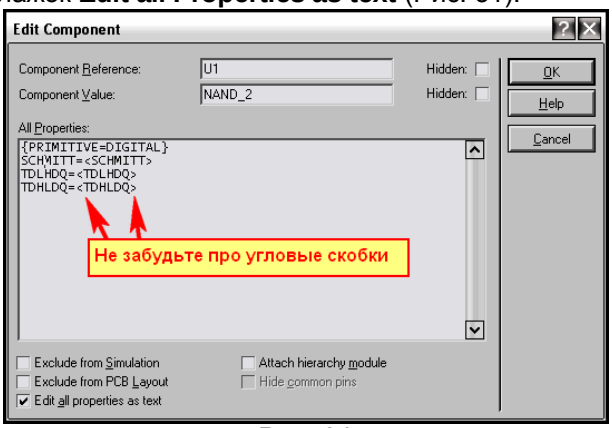


Рис. 31.

Совет: *Обратите внимание на фигурные скобки в верхней строчке на Рис. 31 – свойство Hidden (скрытое), т.е. не показывается под элементом на месте серого <TEXT> в проекте. SCHMITT и задержки не имеют фигурных скобок по краям, значит будут видимыми. Когда свойств и компонентов на листе много, это начинает мешать. Заключите их в фигурные скобки, и они скроются с глаз долой. Надо, чтоб опять стали видимыми – скобки убираем.*

Еще на дочернем листе необходимо поместить скрипт следующего содержания:

```
*DEFINE
TGQ=?
*MAP ON VOLTAGE
10V : SCHMITT=[NULL],  TDHLDQ=250n, TDLHDQ=250n
5V : SCHMITT=[NULL],  TDHLDQ=400n, TDLHDQ=400n
```

Предвижу возникающие вопросы и поясняю. Ну, что во что превратится при компиляции – ясно. Свойство Шмитта я сохранил на будущее, плюс к тому его необходимо будет включать, если мы захотим сделать генератор на нашей модели. Я не стал включать в таблицу стандартное для 176-й серии питание 9V, хотя справочные задержки взяты именно для него. Данная серия прекрасно работает и при 12V, поэтому я и поставил 10 – просто округлил. Ну и для питания 5V задержки фронтов явно увеличатся, это неоспоримый факт. Когда-то мне попадалось, что около 400 наносек, но сейчас не помню, где это было – поставил по памяти. Ну и поскольку у нас одна серия и один компонент, то необходимость в **VALUE** отпала, и оно в **MAP ON** не фигурирует.

Будем последовательными, вернемся на главный лист и снова запустим **Make Device**, чтобы добавить на третьей вкладке свойство **VOLTAGE** (Рис. 32). Конечно же, его нет в стандартных, т.е. добавляем через **New => Blank Item**.

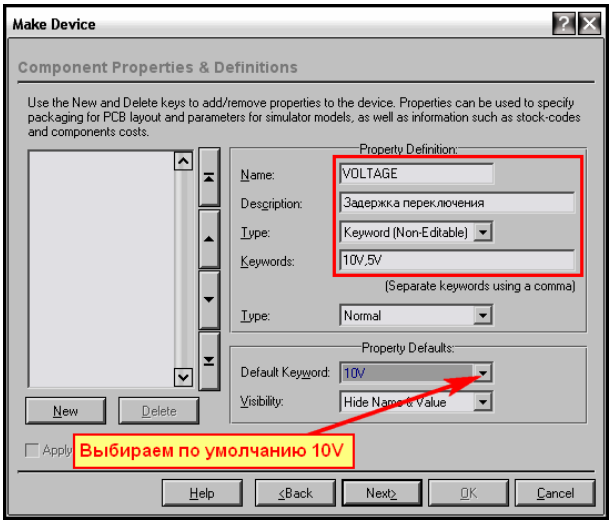


Рис. 32.

Прогоняем тестирование компонента. Теперь уже о 20 МГц нечего и мечтать, даже на 1 МГц видны задержки уже и на цифровом графике. Ну, так оно и должно быть, ведь 176-я серия разрабатывалась как тихоход для часов. Пример со скриптом на дочернем листе в папке **LA7_with_MAP** вложения.

И осталось нам заскочить на дочерний лист и сформировать MDF, а затем подключить его на третьей вкладке **Make Device**. Этот вариант в папке **LA7_with_MDF** вложения.

Ну а когда же Протеус начнет ругаться? Да он бы уже нас давно обложил трехэтажным английским, но мы его обманули в самом начале. Выводы питания то мы не приделывали и корпус не назначали. Ну, выводы питания мне приклеивать лень, а корпус придется назначить, например DIL14, хотя по большому счету шаг выводов у 176-й серии метрический 2,5 мм и немного отличается от 2,54 мм, принятого в импортных DIP (DIL). Ну, где там наш **Make Device** – пошли на вторую вкладку **Packagings**. Там щелкаем кнопку **New** и в раскрывшейся библиотеке корпусов в окне **Keywords** набираем **DIL**, чтобы не рыться по всем библиотекам. Корпус быстро нашелся по совпадению с ключевым словом (Рис. 33).

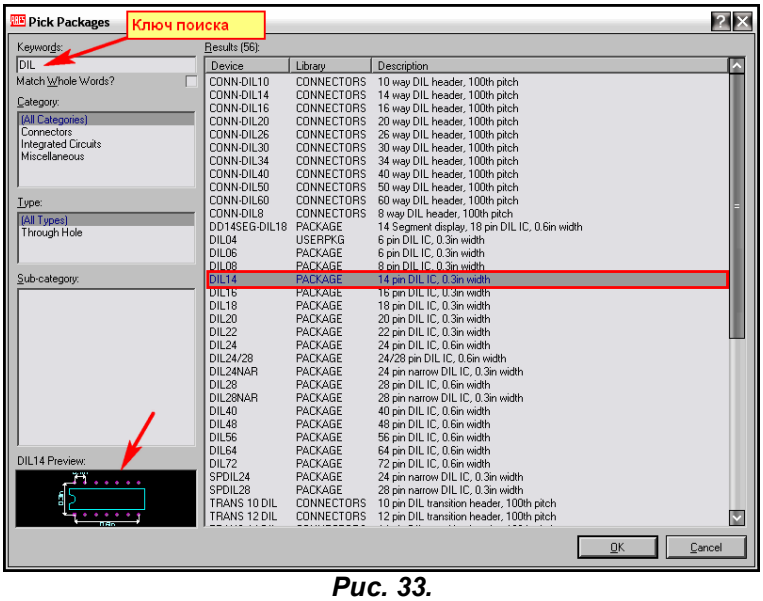


Рис. 33.

Далее назначаем его нашей микросхеме в таком порядке (Рис. 34). Сначала в окне **No of Gate** ставим 4 – ведь в корпусе 4 элемента 2И-НЕ. После этого переходим к назначению выводов и тут же у нас появляются колонки B, C, D и становится доступным установка флажка **Gates (elements) can be swapped on the PCB layout**. Этот флажок позволяет **ARES** менять местами выводы и целиком элементы A,B,C,D для улучшения трассировки печатной платы, поэтому его лучше включить. Проходим последовательно назначение выводов для этих элементов. Назначаемый в данный момент в таблице подсвечивается желтым, свободные (неназначенные) выводы на корпусе в окне справа – лиловым.

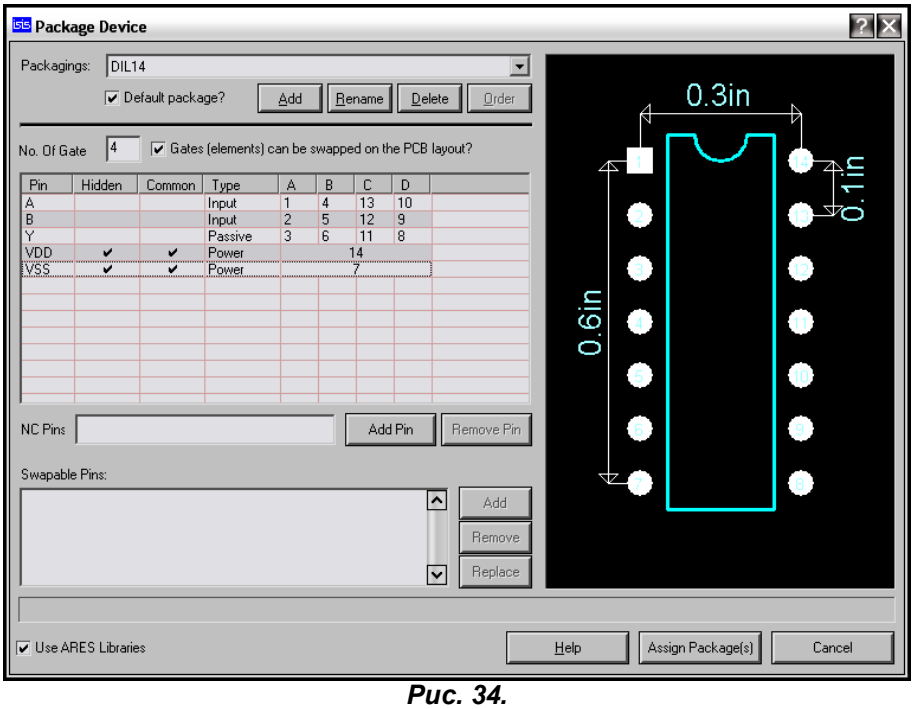


Рис. 34.

Первоначально у нас не будет строк VDD и VSS, ведь у графической модели эти выводы отсутствовали. Поэтому, когда у нас останутся неназначенными последние два вывода 7 и 14, давим кнопку Add Pin и прописываем их имена именно так, как на картинке **VDD** и **VSS**. Протеус автоматически решит, что это Power и поможет нам в этом. Достаточно только назначить номера выводов. Завершаем назначение корпуса кнопкой **Assign Package(s)**. Вот и все, проходим до конца процедуру **Make Device**. Теперь у нас на третьей вкладке будут уже три свойства: **VOLTAGE**, **MODFILE** и **PACKAGE**. Вот теперь и запустим еще раз симулятор. Сначала, вроде, как и ничего, но при остановке получим горчичники типа: **Pin 'VSS' is not modeled** и **Pin 'VDD' is not modeled**. Специально в таком виде лежит во вложении, в папке **LA7_with_PCB**.

Дождались!!! Сейчас мы их…. Пробегаем **Make Device** до третьей вкладки и добавляем через New из списка **ITFMOD** со значением по умолчанию **CMOS** (Рис. 35). Проходим до конца и сохраняем. Этот вариант в папке **LA7_FINAL**. Тестируем – никаких горчичников, потому что, какой вывод питания куда подключен – **ISIS** извлек из файла **ITFMOD.MDF** для строчки **CMOS**.

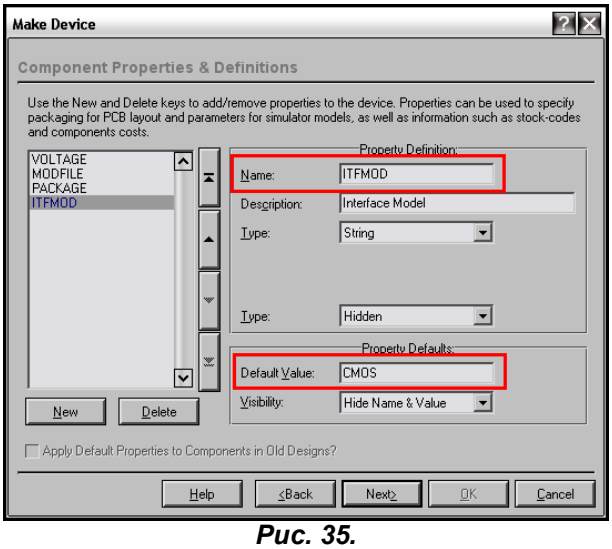


Рис. 35.

Ну вот, материал получился очень объемным, но, надеюсь, и очень полезным. В завершение только добавлю, что все примеры из этого параграфа лучше рассматривать в той последовательности, как они встречаются в тексте. Иначе, сохранив более продвинутую модель через **Make Device** заранее, вы не сможете увидеть некоторые эффекты, например, те же горчичники.

Возврат к содержанию

6.3. Генераторы на RC и LC цепях в Протеусе и несколько способов их запуска. Извечные русские вопросы: «Что делать? » и «Кто виноват? ».

Очень многих начинающих пользователей программы эти два вынесенных в заголовок вопроса начинают терзать уже на первом этапе знакомства с **ISIS** при попытке моделировать генераторы с задающей RC цепочкой. Я уже столько раз отвечал по этому поводу на форуме, что сбился со счета. Пора положить этому достойный конец и разобрать вопрос досконально. Тем более что частично этот материал нам понадобится при моделировании счетчика К176ИЕ12 буквально в следующем параграфе. Почему то на второй из извечных вопросов большинство пользователей отвечает однозначно: «Глюк программы, виноваты разработчики». Особенно прельщает частая добавка: «Да вот я в Мультисиме проверял – там сразу все работает, а в Протеусе никак». Ну, так каждому свое. Нравится – моделируйте там, ну а мы грешные уж как-нибудь справимся и здесь. Ну а теперь к делу…

Основная ошибка при попытке моделирования генераторов – это «тупое» (другого слова, извините, подобрать не могу) перерисовывание схемы генератора и попытка в лоб запустить его. Автор схемы или учебник по электронике гласит, что работает – значит так и должно быть. Ну, что ж «нарисуем» классический мультивибратор на транзисторах и попробуем его запустить (Рис. 36).

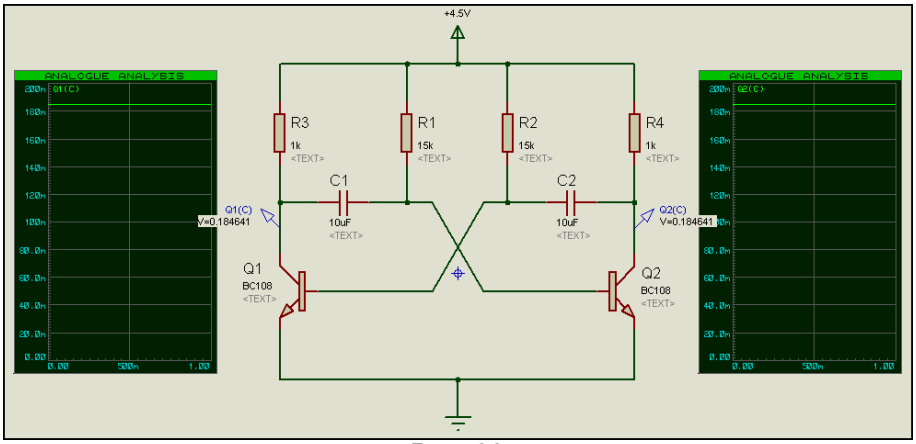


Рис. 36.

Как видим, и в графическом режиме, и в интерактивном никакой генерации не наблюдается, оба транзистора полностью открыты – напряжения на коллекторах 0,18 Вольт (пример во вложении **MULT_TRANS/ Default_mult.DSN**). Все, – Протеус в помойку, разработчики – «кАзлы». Или все-же мы слегка закозлили? Вспомним теорию – режим генерации в таких мультивибраторах возникает из-за несимметричности плеч, обусловленных разбросом параметров реальных компонентов схемы. При этом один из конденсаторов зарядится быстрее и возникнет генерация. Ну и где она у нас разброс параметров у виртуальных моделей? Значит, напрасно мы виним разработчиков. Основным времязадающим элементом здесь является конденсатор. Давайте зарядим один из них. Для предварительной зарядки (вспомним свойства примитива конденсатора) используется свойство **PRECHARGE**. Зададим для **C1** мультивибратора **PRECHARGE=2** (т.е. зарядим хотя бы до 2V, а можно и больше). Опля, заработало!!! (Рис. 37). В той же папке вложения - **Precharge_mult.DSN**.

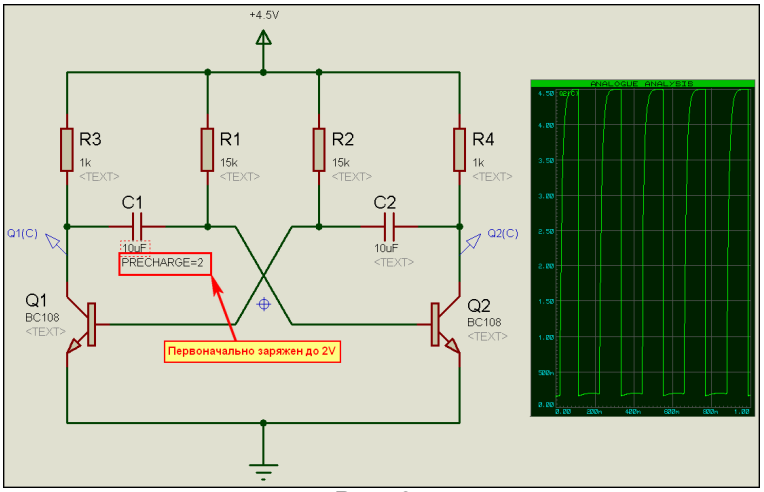


Рис. 37.

Да, «ларчик просто открывался». Но это не единственный способ заставить наш мультик работать, как положено. Вернемся к исходному варианту и уберем у конденсатора первоначальный заряд. Вспомним, что существует еще одно замечательное свойство – **Initial Condition** (первоначальное состояние). По отношению к проводникам оно означает начальное напряжение на проводе при

старте симуляции и назначается в режиме меток **LABEL** из левого меню. Давайте присвоим проводнику от базы транзистора Q1 (он же подключен и к выводу конденсатора C2) метку **IC=0**. Запустим график и … - тоже работает (Рис. 38). Вот и еще один способ обнаружился.

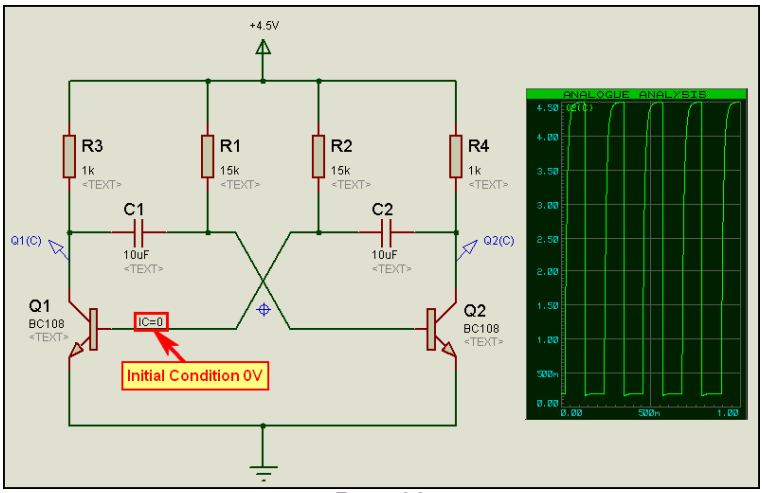


Рис. 38.

Ну а теперь несколько экзотичный, хотя и работающий в некоторых случаях способ. Иногда достаточно просто поменять конденсатор на их анимированную модель, лежащую в библиотеке **Capacitors=>Animated**. Есть у меня большое подозрение, что в этом случае срабатывает стоящее по умолчанию для этой модели в ее **MDF** свойство **PRECHARGE=0**. Правда, этот способ хорош, когда времязадающий конденсатор в единственном числе.

Как мы видим из всех приведенных примеров, в основном устойчивый запуск генераторов связан именно с проблемами конденсаторов, а уж если быть совсем точным, то с начальным условием старта симуляции. Вот этот фактор и надо учитывать при моделировании генераторов в ISIS.

Недавно на форуме всплывал вопрос по генератору трехточке Колпитца, навеянный материалом, расположенным по этой ссылке:

http://logic-bratsk.ru/radio/ewb/ewb2/CHAPTER2/2-8/2-8-1/2-8-1.htm

Автор вопроса поступил именно так, как я описывал вначале, т.е. просто перенес схему от **Electronic Workbench** в **ISIS**. Естественно, результат был плачевный. Но теперь мы «вооружены и очень опасны». Поступим так же, как и в предыдущих случаях (Рис. 39).

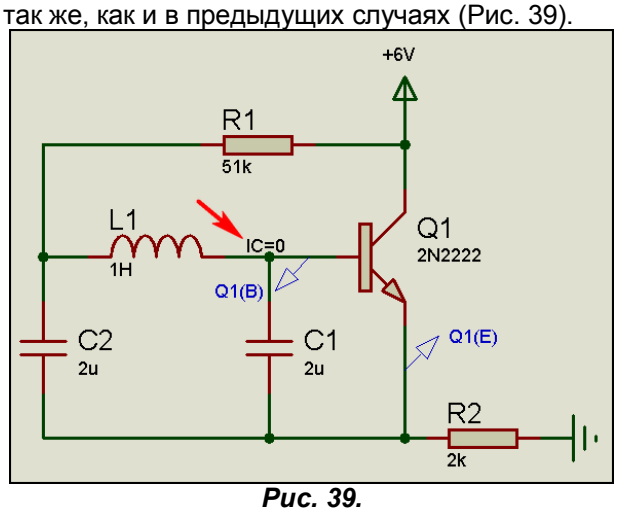


Рис. 39.

Простым добавлением **IC=0** в базовую цепь легко удалось запустить генератор. Более того, в ISIS результат симуляции именно этого варианта оказался более приемлемым. Если у этой схемы в **EWB** согласно приведенной ссылке период оказался равным 7,34 мс при теоретическом — 6,28 мс, то в ISIS период равен 6,63 мс (Рис. 40). Сравните, чей ближе. Этот вариант в приложенном файле **3_POINT\Colpitts1_IC.DSN**. В той же папке вариант с емкостной связью **Colpitts2_Precharge.DSN**, в котором использовано свойство конденсаторов **PRECHARGE**. Чтобы уж окончательно развеять мифы, я взял и грубо, не вдаваясь в расчеты, набросал схему с индуктивной связью – генератор Мейснера (или Майснера). И воспользовавшись свойством **Initial Condition**, в несколько секунд добился его работы. Этот вариант я также приложил в той же папке под именем **Meisner1_IC.DSN**. Думаю, достаточно материала по чисто аналоговым генераторам, чтобы убедиться в их работоспособности. Мы же далее немного остановимся на RC-генераторах на логических элементах, с которыми тоже у начинающих возникают большие проблемы.

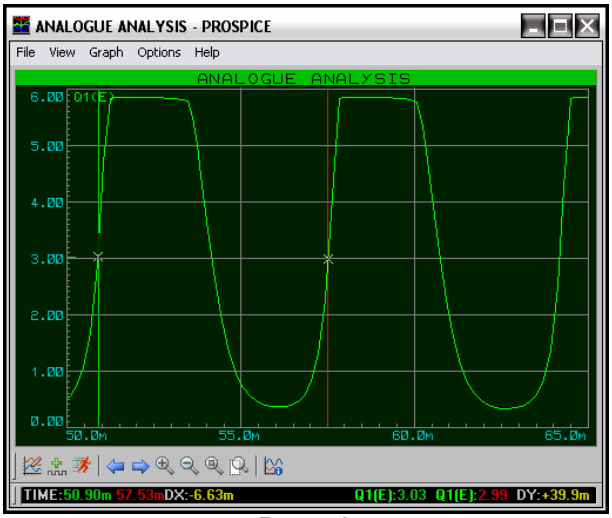


Рис. 40.

Я уже ранее указывал на свойство **SCHMITT**, которое в большинстве случаев позволяет запустить генератор на логических элементах. Но это большинство тоже строго ограничено. Как правило, все пытаются моделировать генератор на элементах 2И-НЕ микросхемы **4011** – наши аналоги ЛА7 в сериях 176, 561, 564 и т.п. Мы уже рассмотрели ее модель выше и обратили внимание, что свойство **SCHMITT** прописано в **MDF**. Быстренько набросаем типовой генератор, пропишем первому логическому элементу по входам это свойство и запустим симуляцию (Рис. 41).

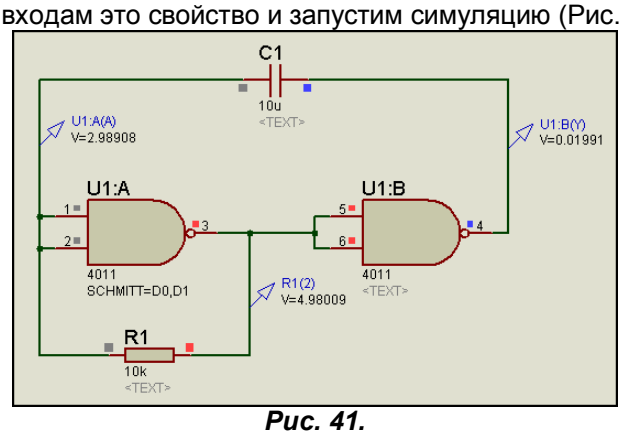


Рис. 41.

Я на рисунке 42 приведу аналоговые графики в характерных точках схемы. Обратите внимание, что на четвертом – **DIGITAL** первый пробник выглядит достаточно «коряво». Именно поэтому в данном случае я и использовал аналоговые графики для анализа этого генератора. Этот вариант во вложении **LOGIC_SCHMITT\ NAND_2.DSN**.

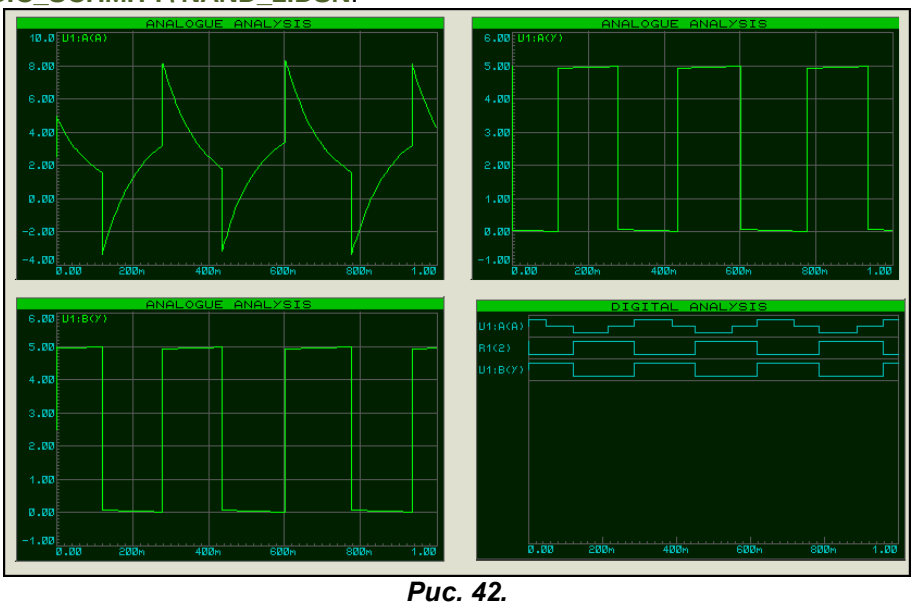


Рис. 42.

В той же папке лежит схема генератора на элементах 3И-НЕ микросхемы **4023**. И так же, как и в предыдущем случае для логического элемента стоящего первым задано **SCHMITT=D0,D1,D2**. Ну, вроде все, как учили, но при попытке запустить симуляцию мы видим унылые серые квадратики вместо веселого перемигивания и строго половинчатые уровни напряжения по всем входам/выходам (Рис. 43). Вроде кому то пора что-то набить - это я имею в виду себя. Но, я и не говорил, что это свойство будет работать всегда.

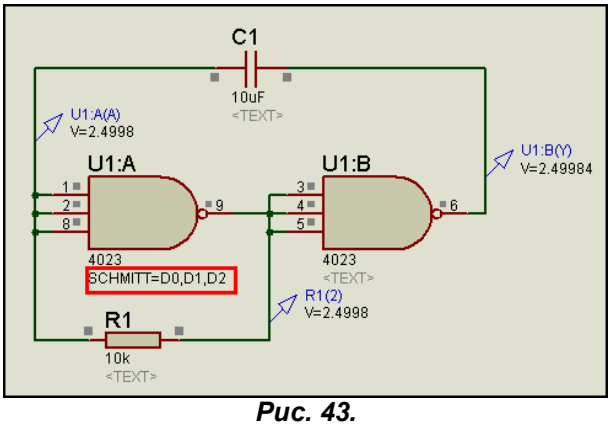


Рис. 43.

А вот теперь открою секрет из «загашника» - почему не работает. Сходите в файл **40NAND3.MDF**, назначенный для **4023** и посмотрите в его начало. Раньше просто посылать вас туда было бесполезно - вы б его и не нашли, но теперь уже пора. Ниже я приведу значимый кусок из этого файла:

```
*MAPPINGS,3,VALUE+VOLTAGE
4073+5V : TDLHDQ=45n, TDHLDQ=55n, TGQ=?
4073+10V : TDLHDQ=20n, TDHLDQ=25n, TGQ=?
4073+15V : TDLHDQ=15n, TDHLDQ=20n, TGQ=?

*MODELDEFS,0

*PARTLIST,1
U?,AND_3,AND_3,PRIMITIVE=DIGITAL,TDHLDQ=<TDHLDQ>,TDLHDQ=<TDLHDQ>,TGQ=<TGQ>
```

Вы где-нибудь видите упоминание **SCHMITT**? Представьте, я тоже не вижу. Ну и что тогда здесь будет работать? Делаем вывод – это свойство будет работать только в тех случаях, если прописано в MDF для данного компонента. А универсальными моделями, содержащими его, являются те, которые предназначены для моделирования триггеров Шмитта. Так что не надо даже лазить по всем MDF, а достаточно заглянуть в библиотеку. Вводим ключевое слово **Schmitt**, выбираем серию, например 4000 и видим, что наряду с рассмотренным в предыдущем параграфе **40NAND2**, которое назначено для 4011 и двухвходового триггера Шмитта 4093, это свойство будет еще в модели инверторов **40INV.MDF** (Рис. 44).

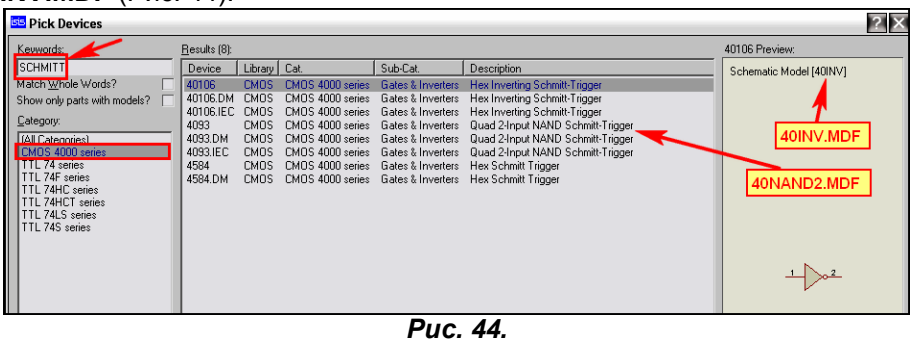


Рис. 44.

Заглянем в этот MDF и увидим, что его можно присвоить следующим инверторам: **4009**, **4049**, **4069** (конечно же, для **40106**, **4093** и **4584** оно уже присвоено). Присваиваться оно будет строчкой **SCHMITT=D**, поскольку у инверторов только один вход, и он не нумеруется. Файл **40INV.MDF** есть во вложении.

Ну а как быть, если мне приспичило использовать, например, элементы 2ИЛИ-НЕ нашей К561ЛЕ5 или ее аналога CD4001 – в Протеусе просто **4001**. Там Шмитт не катит…. Итак, второй «кролик из цилиндра» - я сегодня как факир. Подумайте логически, о чем я тут распинался вначале. Ну а теперь посмотрите на Рис. 45 и загляните в папку **LOGIC_GEN** вложения. Там представлены в двух дизайнах генераторы на двух и трех логических элементах. От комментариев воздержусь, по-моему, и так всем все ясно.

Ну, вот такой небольшой, но весьма познавательный материал по моделированию генераторов в **ISIS**. Надеюсь, что теперь со страниц форума надолго пропадут заголовки типа: «не работает генератор в Протеусе». А если начнут появляться в мое отсутствие, то каждый теперь знает, куда послать страждущего…

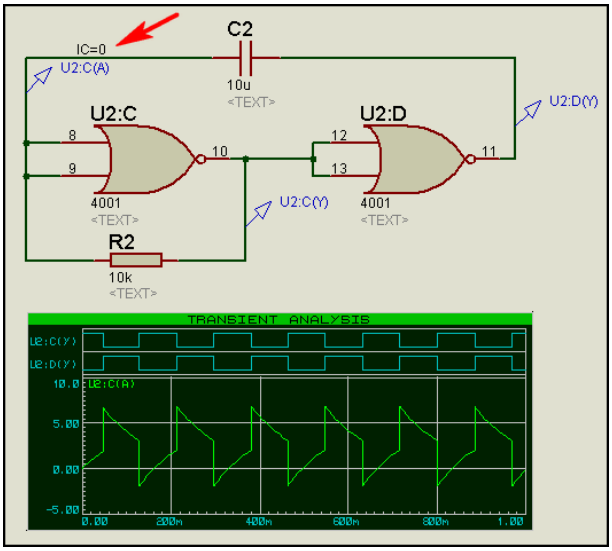


Рис. 45.

****

Материал был уже готов и выложен на форуме, когда в параллельной ветке форума:
http://kazus.ru/forums/showthread.php?t=13612
возник вопрос о генераторах на триггерах в Протеусе. Естественно, я не мог пройти мимо, так как этот вопрос напрямую связан с симуляцией генераторов, рассматриваемой в этом пункте. Чтобы не нарушать нумерацию прилагаемых рисунков, в этом дополнении их приводить не буду. Но в дополнительном вложении **Trigger_Gens** в папке **Examples_Gens** добавлю парочку вариантов генераторов на RS-триггерах, устойчиво работающих в Протеусе.

А заинтересовал меня этот вопрос еще и потому, что при исследовании модели **4042**, на которой сделана попытка просимулировать генератор по приведенной выше ссылке, обнаружена ошибка. Суть ошибки в том, что в реальном триггере **CD4042** происходит асинхронный перенос данных с входов **Dx** на выходы **Qx** при одинаковых уровнях сигнала на управляющих входах **CLK** и **POL**. Т.е. если **CLK=POL=0** или **CLK=POL=1**, то сигнал с входа D должен переноситься на соответствующий ему выход Q асинхронно. Внутренняя структура модели **4042** в ISIS выполнена на D-триггерах и сигнал переносится только по изменению фронтов на управляющих входах, что не соответствует действительности. Я не поленился, и проверил поведение аналогичной модели в **Multisim 11**, там она ведет себя вполне адекватно и генератор с использованием данной модели устойчиво запускается. Пришлось заняться исправлением модели. Что у меня получилось в результате приложено в папке вложения **New_model_4042**. В папке **Structure_Vers_1** приложен тестовый проект одного из четырех каналов микросхемы, созданного на основе даташита. Однако мне этот вариант не очень понравился и другой, более компактный приложен в папке **Structure_Vers_2**. С этим вариантом был протестирован генератор. Результат в проекте из папки **Test_Generator_With_Child**. Там схематичная модель присоединена в качестве дочернего листа к графической модели. С этого листа скомпилирован файл **4042B.MDF**. Тест модели с этим файлом приложен в папке **Test_Generator_With_MDF**. Можно просто скопировать его в папку **MODELS** Протеуса, а над моделью из этого проекта произвести Make Device и сохранить вариант **4042B** для дальнейшего использования. Если есть желание заменить существующую модель **4042**, то можно над ней проделать Make Device и на третьей вкладке поправить ссылку для **MODFILE** на **4042B.MDF**. Для желающих полностью заменить модель в библиотеке **DIGITAL.LML** рекомендую воспользоваться методикой из **п.6.17** этой части FAQ. С этой целью в папке New_4042_MDF приложен MDF-файл, в разделе ***MAPPINGS** литера B на конце имени модели отсутствует, как и в оригинальном файле модели из поставки Протеуса.

****

Возврат к содержанию

6.4. Полезные опыты с цифровым элементом в ISIS. Заключительный материал об IC, NS, PRECHARGE и SCHMITT.

Этот дополнительный материал призван поставить финальную точку в разборке ситуации с цифровыми элементами в **ISIS**. Разбирая в предыдущем параграфе генераторы, я уже привел характерные особенности их запуска в симуляторе. Но, все же, немного поразмыслив, решил более подробно разобрать – чем же характерны цифровые элементы в Протеусе, ну и попутно ознакомить вас с еще одним очень полезным свойством – **NS** (NODESET).

Внимание! Я пошел на поводу у ребят из Лабцентра и допустил серьезную «очепятку» вслед за ними. В **п.6.1** для аналого-цифрового примитива свойство гистерезиса для нижнего порога записывается как **VHL**, а не **VLH**. И в хелпе Протеуса и у меня ранее ошибка. Приношу извинения за себя и Labcenter Electronics и восстанавливаю «статус-кво».

Для начала рассмотрим – как ведет себя типовой цифровой примитив буфера или инвертора при подаче на его вход аналогового сигнала. Для этого подадим на его вход линейно изменяющееся

напряжение от генератора (треугольный сигнал). Для этой цели я воспользуюсь генератором **Pulse** из левого меню **Generator Mode** с несколько «заумными» настройками (Рис. 46).

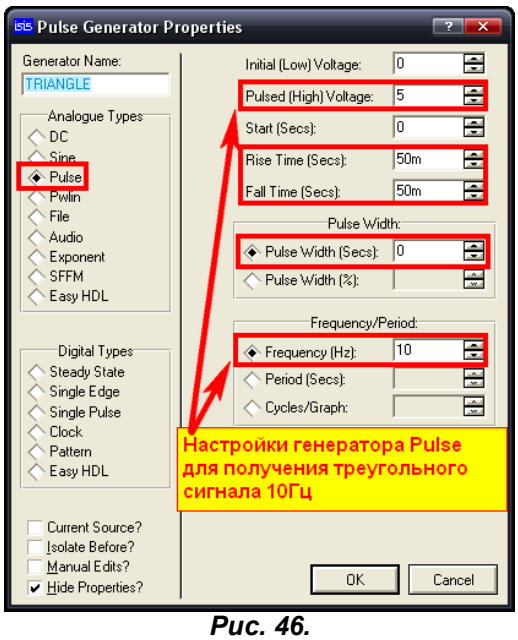


Рис. 46.

Допустим, мы приняли длительность фронта (**Rise**) и спада (**Fall**) импульса равными половине его периода, а ширину верхней полки **Width** равной нулю. Получится, что наш импульс состоит только из линейно повышающегося и понижающегося фронтов, т.е. фактически треугольного сигнала. Если частота равна 1 Гц, то период 1 сек, а фронт и спад по 500мсек. Я взял для своих экспериментов сигнал частотой 10Гц и амплитудой 5 Вольт. Соответственно фронт и спад будут в 10 раз меньше. Теперь подадим этот сигнал на вход цифрового примитива **BUFFER** из библиотеки **Modelling Primitives**. На выход повесим зонд напряжения и проанализируем с помощью аналогового графика выходной сигнал в зависимости от напряжения на входе цифрового буфера. Эта ситуация с выделенными характерными точками представлена на рисунке 47.

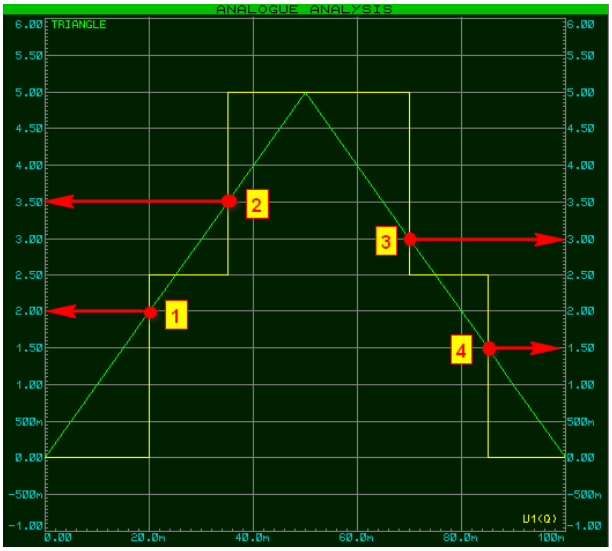


Рис. 47.

Наш цифровой буфер ведет себя как трехпороговый элемент с уровнями 0В, 2,5В (половина питания) и 5В (полное питание). Для того чтобы представить себе откуда взялись эти точки, вспомним приведенные в **п.6.1** свойства аналого-цифрового примитива по умолчанию. Вычислим абсолютные значения:

- Порог переключения на высокий уровень – **VTH=70%** – 5*0,7=**3,5В** (точка 2);
- Порог переключения на низкий уровень – **VTL=30%** – 5*0.3=**1,5В** (точка 4);
- Гистерезисы переключения – **VHH=VHL=10%** – 5*0,1=0,5В. Переход в неопределенную зону (2,5В) происходит от нуля – **VTL+VHL=2.0В** (точка 1) и от единицы – **VTH-VHH=3,0В** (точка 3).

Ну, вот мы и просчитали все наши выделенные точки. И теперь можем точно предсказать – как поведет себя цифровой элемент с этими свойствами по умолчанию при подаче на его вход сигнала определенным напряжением. Для примера я занизил амплитуду треугольника до 3В – ниже порога переключения **VTH**. Результат симуляции графика на рисунке 48, и он вполне предсказуем.

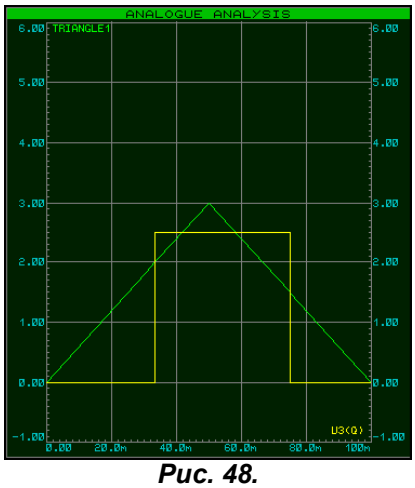


Рис. 48.

Пойдем дальше и рассмотрим – что дает нам добавление в примитиве свойства **SCHMITT**. График для примитива буфера приведен на рисунке 49. Сравнивая с Рис. 46, делаем вывод – исчезли гистерезисы. Остались только две точки переключения – переход на высокий уровень при 3,5В и возврат на низкий при 2,0В. Вот она физическая сущность этого «загадочного свойства». Но стоит еще раз заметить, что воздействует оно только на исходные цифровые примитивы, из которых построена модель реального логического элемента. Вспомните о том, что я писал в предыдущем параграфе про элементы 3И-НЕ микросхемы **4023**. Но и это еще не все сюрпризы, связанные с цифровыми примитивами **ISIS**. Дотошные пользователи решат – да вот мы обнулим гистерезисы и получим тот же **SCHMITT**. И вот тут всплывает один неприятный нюанс из серии «индейской национальной избы», как обозвал ее пес Шарик в мультике про Простоквашино. Попытки изменить аналого-цифровые свойства примитива посредством переназначения **VTH**, **VTL**, **VHH** и **VHL** к желаемому результату не приведут. Можете поэкспериментировать самостоятельно. График будет «стоять как вкопанный». И не важно, в чем вы их будете назначать – в абсолютных значениях или в процентах. Во всяком случае, я лично реального результата не получил. Результаты моих исследований для примитива буфера и инвертора, который, как и положено, ведет себя с точностью до наоборот, приведены в соответствующих проектах **Buffer.DSN** и **Inverter.DSN** вложения. Может кому-то повезет больше.

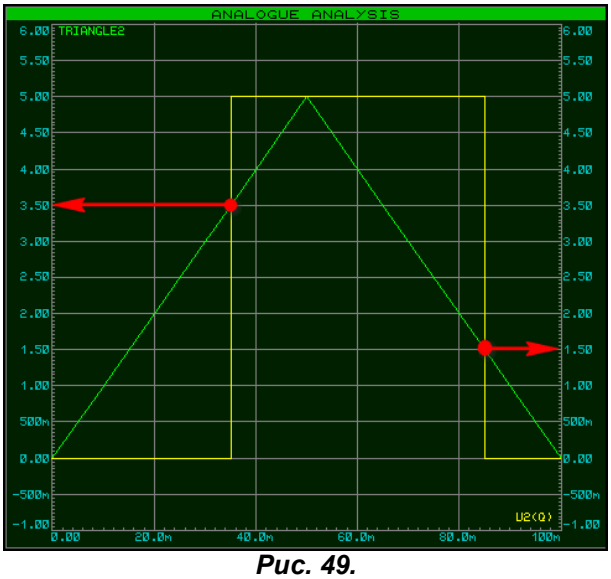


Рис. 49.

Зато картина существенно меняется, если мы задействуем свойство **ITFMOD** и причислим наш примитив к какой либо конкретной серии цифровой логики. На рисунке 50 показан вариант с присвоением примитиву **ITFMOD=CMOS**. Сравните уровни с графиком рисунка 49 – полное совпадение.

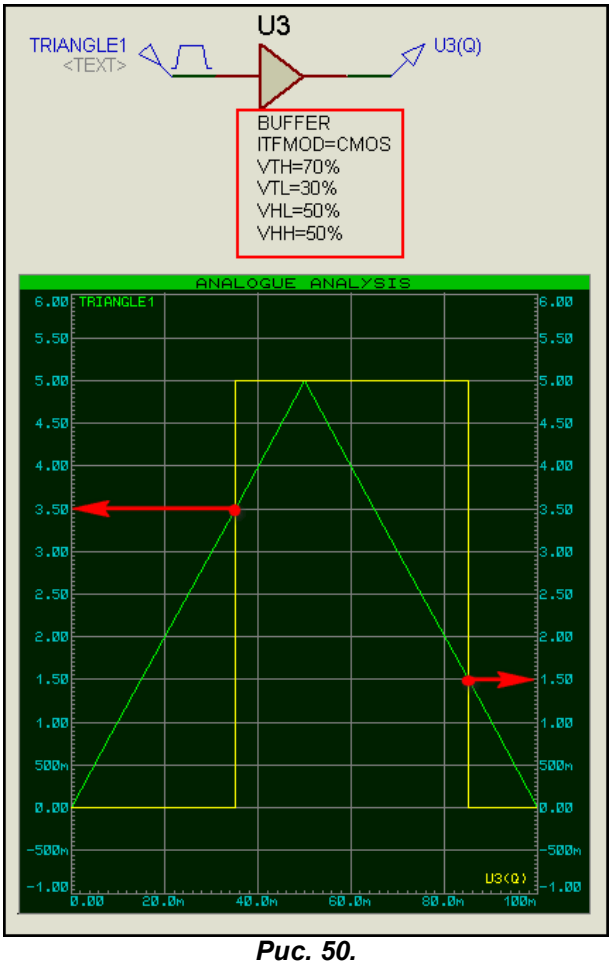


Рис. 50.

Практически мы получили тот же эффект, что и при воздействии свойства **SCHMITT**, только другим путем. В данном примере нам достаточно было задать гистерезис 40%. Простая арифметика: 40% от 5В составит 2В. При этом **VTL+VHL**=1,5+2=3,5В т.е. совпадет с точкой перехода на уровень логической единицы **VTH**. Таким образом, мы убрали ступеньку на переднем фронте импульса. Далее **VTH-VHH**=3,5-2=1,5В – совпадает с точкой перехода на низкий уровень **VTL**. Этот вариант убирает ступеньку на заднем фронте импульса. Когда я задал **VHL**=**VHH**=50%, что составит 2,5В я сдвинул точки перехода так, что ранее срабатывает **VTH** или **VTL**. При любом значении одного из гистерезисов менее 40% будет появляться соответствующая ступенька с уровнем 2,5В как на графике рисунка 47. Варьируя этими четырьмя значениями для логического элемента в проекте, мы можем добиваться нужных нам результатов. Следует помнить, что единицы измерения для них должны совпадать – или все в Вольтах или все в процентах. Последнее принято по умолчанию. Поэтому если вы поставите какой-либо элемент 4000 серии в схему и зададите ему в свойствах только **VTL** и **VTH**, но в абсолютных единицах, например **VTL=1.5** и **VTH=3.5**, то гистерезисы, которые остались по умолчанию в процентах работать не будут и эффект будет тот же, что и на Рисунке 50. Различные варианты с графиками для CMOS приведены в **Buffer_CMOS.DSN**. Все вышесказанное касается и других серий цифровых микросхем, только и уровни там будут несколько другие в соответствии с заданными в **ITFMOD.MDF**. Пример для TTL во вложении **Buffer_TTL.DSN**.

Теперь вернемся на секунду к нашему генератору на инверторах и с точки зрения новых познаний рассмотрим, почему он не запускается без лишних телодвижений с нашей стороны. Я еще раз повторил картинку генератора на двух элементах 2И-НЕ с установленными зондами на входах-выходах элементов (Рис. 51).

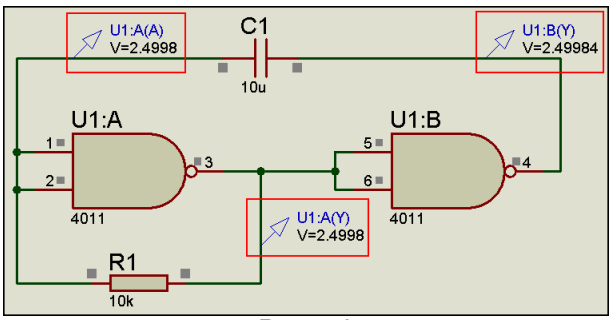


Рис. 51.

Обратите внимание, что при запуске симуляции напряжение во всех контрольных точках практически одинаковое и равно половине питания или тому пресловутому порогу 2,5В на предыдущих графиках. Значит, для того чтобы его сдвинуть хотя бы в одной из точек и нужны дополнительные меры. Это можно сделать даже кнопкой **BUTTON**, посадив кратковременно на землю, например, входы первого логического элемента при запущенной симуляции. Но мы уже знаем, что есть такое интересное свойство **IC** – **Initial Condition** для цепей, которым мы воспользовались в предыдущем параграфе. Однако и здесь тоже кроется несколько подводных камней. Главная особенность состоит в том, что данное свойство можно присваивать только аналоговым цепям. Если Вы попытаетесь назначить **IC** цепи, не содержащей аналоговых компонентов – Протеус незамедлительно при старте симуляции вывесит красное сообщение об ошибке. Для цепей, которые соединяют только цифровые компоненты, аналогичное свойство носит название **BS** (**Boot State**) – состояние при загрузке. Свойству **BS** можно назначать состояние в любом виде, которое поддерживает Протеус для цифровых цепей. Это: **1, 0, H, L, HIGH, LOW, SHI, WHI, SLO, WLO** или **FLT**. Второй особенностью **IC** является то, что это состояние принимается для цепи до начала первой итерации, т.е. до начала расчета симулятором **ProSPICE** операционных точек схемы. Для задания начальных условий по постоянному току при нулевой итерации служит другая директива **NS** (NODESET) – установка узла. Она назначается, как и **IC** через лэйбл, присваиваемый нужному проводу. Например, **NS=10** означает, что на начальной стадии расчета точки в данном узле цепи устанавливается значение 10 Вольт. Это свойство наиболее полезно, когда **ProSPICE** не может точно рассчитать потенциал узла на начальной стадии и является как бы подсказкой симулятору – каким принять значение потенциала в первый момент расчета. Оно так же полезно, когда симулируется достаточно сложная схема, поскольку позволяет симулятору сократить время на расчет операционных точек.

Обе директивы **IC** и **NODESET** являются стандартными для всех программ, базирующихся на ядре **SPICE**. Это и OrCAD, и Multisim и т.д. Открыв любую книжку, посвященную описанию этих программ, вы сможете найти эти директивы в главах, посвященных заданию начальных условий моделирования. В тоже время свойство **PRECHARGE** для конденсаторов, которое мы тоже использовали для запуска генераторов, является фирменной добавкой Лабцентра к SPICE-симулятору. Ну, и чтобы окончательно поставить точку в этом вопросе рассмотрю еще один пример применения данных свойств. Очень часто разработчики используют для сброса цифровых счетчиков, регистров, да и микроконтроллеров интегрирующую RC-цепочку. Не будьте так уверены в ее непогрешимой работе, просто прилепив ее к соответствующей цепи. Взгляните на рисунок 52. На левом графике RC-цепочка стоит с параметрами по умолчанию, на среднем – с применением **Initial Condition**, а на правом – **PRECHARGE**. Надеюсь, для вас не составит большого труда сделать вывод, – в каком случае цепочка обеспечит вам сброс, а в каком нет. Данный пример во вложении носит имя **RC_Reset.DSN**.

Ну и в заключение данного материала сошлюсь на раздел HELP Протеуса, где описан материал, посвященный начальным условиям. ProSPICE Help => ADVANCED TOPICS => INITIAL CONDITIONS. Желающие прочитать в оригинале, могут заглянуть туда.

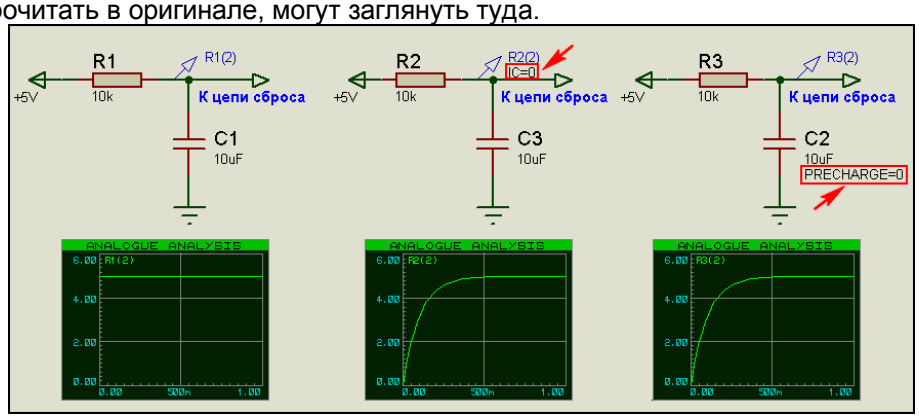


Рис. 52.

Возврат к содержанию

6.5. «Я его слепила из того, что было…». Анатомия CD4060 – прообраза будущей К176ИЕ12.

Когда я достаточно давно задумал сделать модель К176ИЕ12, то естественно подумал – а что можно взять за основу для будущей модели. Основательно перерыв все библиотеки тогдашней версии Протеуса, да и в нынешней там мало что изменилось, я остановил свое внимание на модели 4060. Микросхема представляет собой 14-разрядный счетчик со встроенной схемой генератора. Частота генератора может задаваться как RC цепочкой, так и кварцевым резонатором. Почему-то у нас в России эта микросхема не пользуется особой популярностью. Подозреваю, все дело в том, что она не имеет отечественного аналога. Ближайшее, что когда-то предлагалось в журнале «Радио» №6 за 2006 год, - это аналог на двух м/сх: К561ИЕ16 и К561ЛА7. Да и публикаций всевозможных устройств на этой микросхеме не так уж много. Правда в свое время «Мастер-Килька» отозвался выпуском набора NF239 для сборки таймера: http://www.masterkit.ru/main/set.php?code_id=129679, но и тот уже на данный момент снят с продаж. Больше всего публикаций самоделок на данной микросхеме встречается в журнале «Радиоконструктор», авторы которого до сих пор нет-нет, да и опубликуют что-то свежее с

использованием CD4060. В тоже время, данная микросхема до сих пор выпускается многими зарубежными фирмами под несколько различающимися названиями. Это и **CD4060BC** у Fairchild и **MC14060B** у On Semiconductor и **HEF4060B** у Philips (ныне NXP). Наиболее подробные даташиты на эти микросхемы у двух последних из перечисленных. Я и сам, грешен, пару лет назад использовал ее для создания 12-ти часового таймера заряда резервного аккумулятора в одном девайсе, который благополучно пашет до сих пор на одном из подведомственных объектов. Для тех, кто заинтересуется применением микросхемы всерьез, но не дружит с английским, рекомендую ознакомиться со статьей «Анатомия таймера на ИС CD4060 из набора “МАСТЕР_КИТ» в №3 за 2006 год журнала «Радиосхема» http://publ.lib.ru/ARCHIVES/R/"Radioshema"/_"Radioshema".html .

Ну а мы бегло познакомимся с особенностями ее модели в Протеусе, чтобы иметь подходящую материальную базу для создания К176ИЕ12. Если заглянуть в свойства модели 4060, то помимо типовых для всей этой серии мы обнаружим параметр **Oscillator Frequency** (Частота генератора) со значением по умолчанию **Default** (**Рис. 53**).

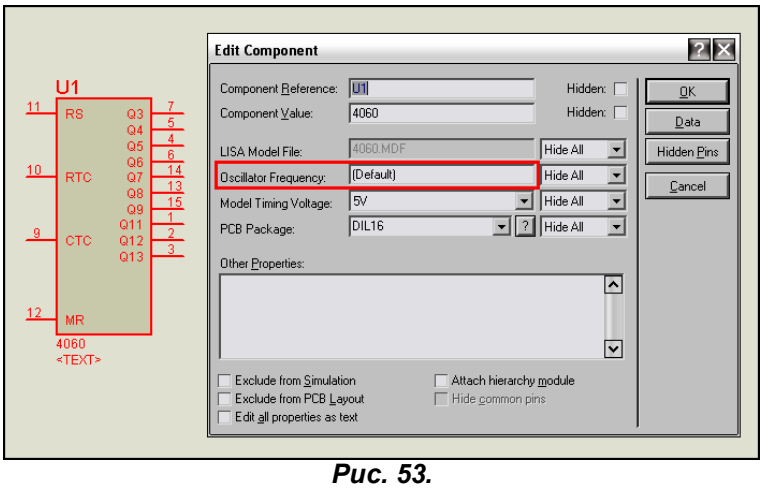


Рис. 53.

На третьей вкладке **Make Device** обнаруживается, что **Default Value** для свойства **CLOCK**, каковым и является частота генератора равно **None**, т.е. генератор отключен, а Limits для него установлен Positive or Zero (положительное или ноль). Для того чтобы привести микросхему в чувство введем вместо Default (**Рис. 53**) числовое значение, например **10** (Гц), а также завесим на землю вход сброса счетчика **MR**. Запускаем симуляцию, и наш счетчик начинает благополучно моргать выходами и входами времязадающих цепей **RS**, **RTC**, **CTC**. Прицепим к ним осциллограф, а заодно и частотомер и убедимся, что там прямоугольные импульсы с заданной нами частотой 10 Гц, т.е. работает встроенный в модель генератор (**Рис 54**). Этот пример во вложении **Internal_Osc.DSN**.

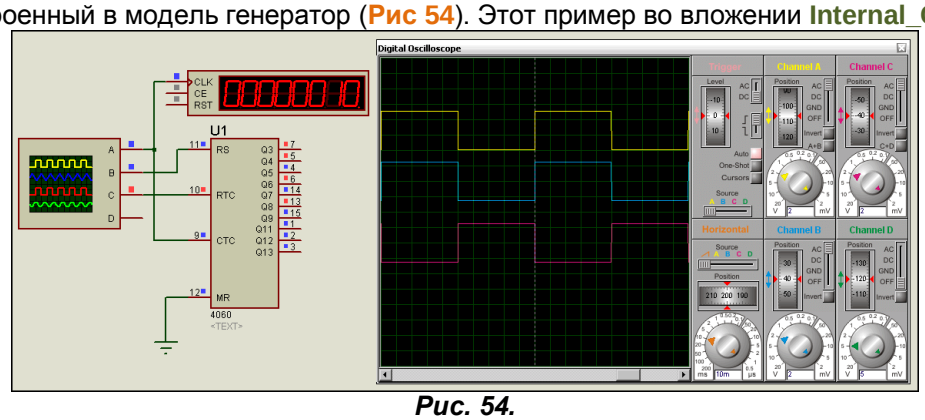


Рис. 54.

Для того, чтобы понять какое еще кроме цифровых значение можно вписать в **Oscillator Frequency**, нам придется добыть из библиотеки **DIGITAL.LML** и исследовать файл модели **4060.MDF**. Я приложил его для тех, кому лень воспользоваться утилитой **GETMDF.EXE**. Ниже приведены первые разделы файла.

```
*PROPERTIES,1
CLOCK=EXTERNAL

*MAPPINGS,3,VALUE+VOLTAGE
4060+5V : TDOSC=35n, TDCQ=25n, TDMRQ=100n
4060+10V : TDOSC=13n, TDCQ=10n, TDMRQ=40n
4060+15V : TDOSC=8.5n, TDCQ=6n, TDMRQ=30n

*MAPPINGS,2,CLOCK
DEFAULT : CLKOSC=DIGITAL,CLKGATE=NULL
EXTERNAL : CLKOSC=NULL, CLKGATE=DIGITAL

*MODELDEFS,0
```

Внимательно посмотрев на разделы ***PROPERTIES,1** и ***MAPPINGS,2,CLOCK** можно сделать вывод, что микросхема должна адекватно реагировать на значение **EXTERNAL** в окне **Oscillator Frequency**. Попробуем этот вариант. Заменим значение **10,** словом **EXTERNAL** и вновь запустим симуляцию. Микросхема тихо стоит, но симулятор не ругается и ошибок не выдает, значит, наше предположение верное. Далее придется обратиться к даташиту на сей девайс, чтобы посмотреть – как подключается времязадающая RC цепочка. Можно просто в режиме подключения к Интернету щелкнуть в свойствах кнопку **Data** (или по меню правой кнопки **Display Datasheet**) и скачать файрчилдовский даташит. Я же приведу на **Рис. 55** вариант от **MC14060B**, поскольку в нем наиболее крупно расписана формула для частотозадающей цепи. Небольшое пояснение – левый по схеме вывод **Rs** в других даташитах соединен с выводом **11** микросхемы, который здесь обозначен как **CLOCK**.

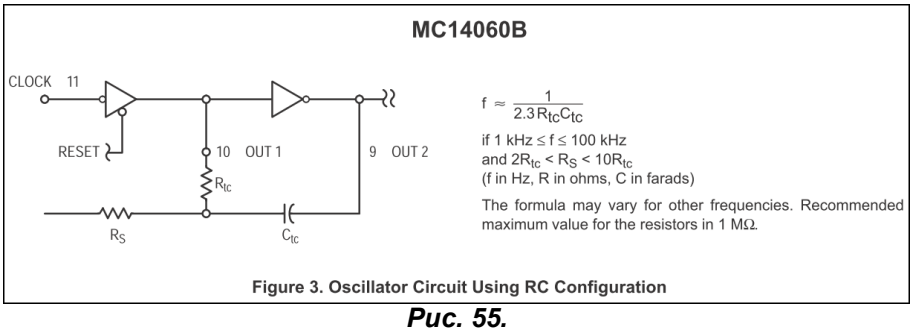


Figure 3. Oscillator Circuit Using RC Configuration

Рис. 55.

В соответствии с формулой рассчитаем для той же частоты 10 Гц сопротивление резистора **Rtc** для емкости **Ctc** в 1мкФ. Получим 43,2кОм – берем ближайшее 43к. В соответствии с упоминавшейся статьей **Rs** выбирается в пределах **2…10Rtc**. Я поставил 200кОм. Ну и чтобы закрепить материал предыдущего параграфа я приложил три варианта: **External_Animal_C.DSN** (на **Рис. 56**) – с анимированным конденсатором; **External_Precharge_C.DSN** – используется **PRECHARGE** и **External_IC_wire.DSN** – используется **Initial Condition**. Открыв соответствующие проекты во вложении и, запустив симуляцию, вы можете лично убедиться, что все они имеют право на существование. Попадание в частоту оказалось не очень, но все же удачным. Вместо расчетных 10 Гц имеем 15, но для RC генератора, да еще в программе, а не в железе – это достаточно неплохо. Желающие могут подобрать резистор, увеличив сопротивление для получения точных 10 Гц. У меня получилось при R1=68кОм – этот вариант оставлен в **External_IC_wire.DSN**.

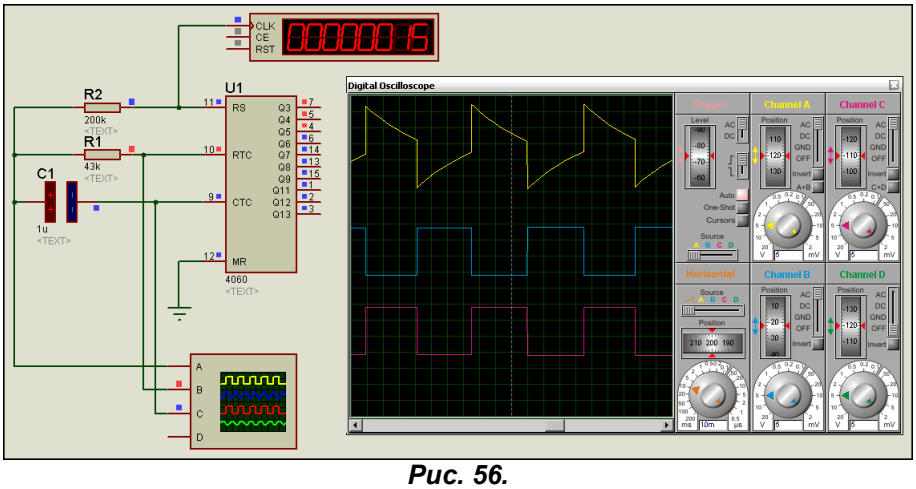


Рис. 56.

Теперь приведу еще одну картинку из даташита с очень важным в дальнейшем пояснением. На **Рис. 57** приведена упрощенная схема структуры **HEF4060B**. Я опять воспользовался тем, что в этом даташите нужный мне элемент уже заранее прорисован покрупнее. Обратите внимание, что в том месте, где подключается конденсатор **CTC** – вывод 11 микросхемы, используется триггер Шмитта! Это важно для нас в дальнейшем. Мы попытаемся пойти дальше разработчиков модели 4060 и воспользоваться этим моментом.

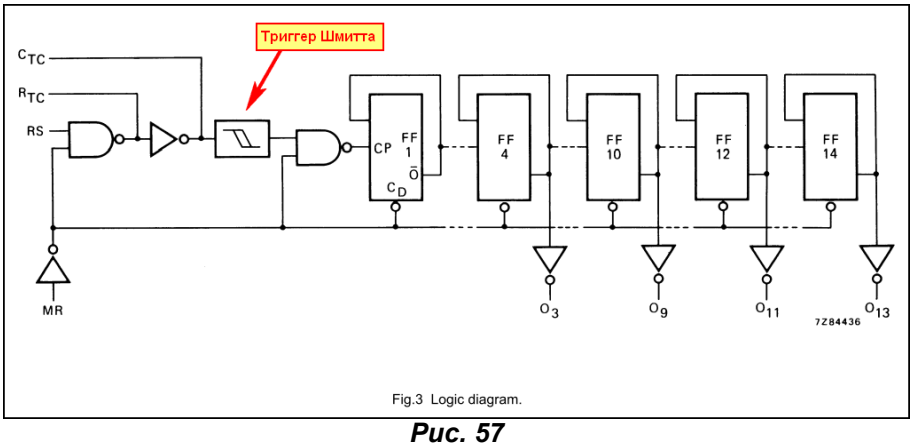


Fig.3 Logic diagram.

Рис. 57

Налюбовавшись вдоволь на структуру, представленную на **Рис. 57** вновь обратимся к файлу **4060.MDF**. Теперь нас интересует **PARTLIST**, приведенный ниже:

```
*PARTLIST,19
OSC,CLOCK,,CLOCK=<CLOCK>,INIT=0,PRIMITIVE=<CLKOSC>,PRIMTYPE=DIGITAL!
U1,INVERTER,INVERTER,PRIMITIVE=DIGITAL
U2,NAND_2,NAND_2,PRIMITIVE=DIGITAL,TDHLDQ=<TDOSC>,TDLHDQ=<TDOSC>
U3,INVERTER,INVERTER,PRIMITIVE=DIGITAL,TDHLDQ=<TDOSC>,TDLHDQ=<TDOSC>
U4,NAND_2,NAND_2,PRIMITIVE=DIGITAL,TDHLDQ=<TDOSC>,TDLHDQ=<TDOSC>
U5,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U6,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U7,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U8,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U9,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U10,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U11,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U12,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U13,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U14,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U15,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U16,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U17,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
U18,DTFF,DTFF,PRIMITIVE=DIGITAL,TDHLCQ=<TDCQ>,TDLHCQ=<TDCQ>,TDRQ=<TDMRQ>
```

Мы можем констатировать, что помимо внутреннего генератора **OSC** и элементов входной логики **U1…U4**, модель содержит 14 счетных DTFF триггеров – элементы **U5…U18**. Можно сделать вывод, что разработчик модели полностью воспроизвел внутреннюю структуру, т.е. имеет место полное схематичное моделирование. В случаях с цифровыми моделями это вполне приемлемо, поскольку симуляция в отличие от аналоговых проходит намного быстрее и не «съедает» ресурсы компьютера. На временных параметрах, заданных примитивам останавливаться не буду, поскольку мы их назначение достаточно подробно разбирали раньше. Также, не стану подробно разбирать и **NETLIST**, а просто приведу кусочек восстановленной структуры модели **4060** на **Рис. 58.** Полностью структуру рисовать не имеет смысла, т.к. соединения всех 14 счетных триггеров одинаковы и повторяют друг друга.

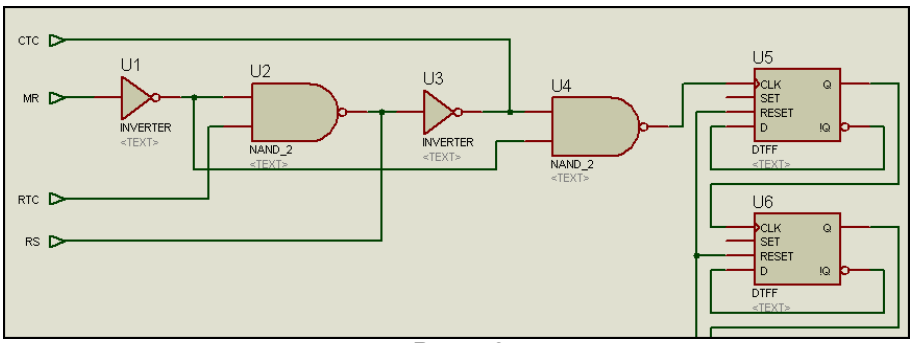


Рис. 58

Тем, кто с элементами цифровой логики знаком только понаслышке, и собирается дальше осваивать представленный здесь материал, могу порекомендовать найти и ознакомиться с одной из нижеперечисленных книг:

Угрюмов Е.П. «Цифровая схемотехника», БХВ-Петербург, - *есть два издания 2000 и 2004 года, годится любое, мне лично больше нравится старое.*

Уилкинсон Барри «Основы проектирования цифровых схем», М., ИД «Вильямс», 2004.

Точчи Рональд Дж,, Уидмер Нил С., «Цифровые системы. Теория и практика» , М., ИД «Вильямс», 2004.

Годится и любой другой учебник по основам цифровой схемотехники. Желательно иметь понятия о работе JK и D триггеров, последовательных и параллельных счетчиков, в т.ч. Джонсона и сдвиговых регистров. Само собой разумеется, что работа логических элементов И, ИЛИ, исключающее ИЛИ и

их комбинаций тоже не должна ставить вас в тупик. В последующем материале я буду свободно оперировать этими понятиями, а разжевывать работу этих элементов я не ставлю себе в задачи. Моя забота – научить вас созданию моделей в Протеусе, но никак не основам цифровой техники, иначе мы увязнем в этом материале на год, а нас ждет еще масса полезных сюрпризов из приемов работы с моделями в этой программе.

Возврат к содержанию

6.6. Пример создания полной схематичной модели счетчика К176ИЕ12. Часть 1 – подготовительные работы.

Те, кто пользовался ранними версиями FAQ по Протеусу, могут благополучно пропустить несколько последующих параграфов, потому что они мало чем будут отличаться от аналогичных старого варианта FAQ.

Итак, ищем исходные данные по микросхеме **К176ИЕ12**. Если есть старые справочники, можно воспользоваться ими. Я же приведу две ссылки в сети, где можно найти информацию по этой микросхеме:

http://lib.qrz.ru/?q=node/5376
http://lib.qrz.ru/?q=node/5480

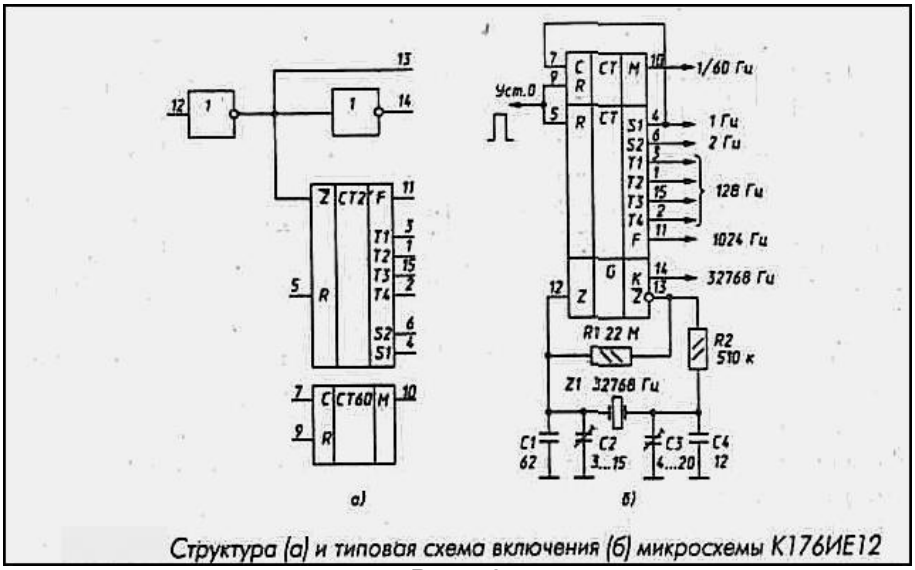
По первой ссылке есть описание ее работы и диаграммы, а по второй есть вариант генератора на **К176ИЕ12** с использованием задающей RC цепочки. Вообще, вся эта информация взята из книги: **Бирюков С.А. «Применение цифровых микросхем серий ТТЛ и КМОП»**, М.; ДМК, 2000.

Любители порыться в архивах могут воспользоваться циклом статей С. Алексеева **«Применение микросхем серии К176»**. Журналы «Радио» №№4…6 за 1984 г.

Давайте кое-что из данной информации перенесем сюда, чтобы было удобнее ориентироваться. Я преднамеренно изменил номера рисунков в тексте, взятом по верхней ссылке, чтобы они совпадали с принятой у меня нумерацией. Остальное оставлено как есть.

Микросхема К176ИЕ12 предназначена для использования в электронных часах **(рис. 59)**. В ее состав входят кварцевый генератор G с внешним кварцевым резонатором на частоту 32768 Гц и два делителя частоты: СТ2 на 32768 и СТ60 на 60. При подключении к микросхеме кварцевого резонатора по схеме **рис. 59** (б) она обеспечивает получение частот 32768, 1024, 128, 2, 1, 1/60 Гц. Импульсы с частотой 128 Гц формируются на выходах микросхемы Т1 - Т4, их скважность равна 4, сдвинуты они между собой на четверть периода. Эти импульсы предназначены для коммутации знакомест индикатора часов при динамической индикации. Импульсы с частотой 1/60 Гц подаются на счетчик минут, импульсы с частотой 1 Гц могут использоваться для подачи на счетчик секунд и для обеспечения мигания разделительной точки, для установки показаний часов могут использоваться импульсы с частотой 2 Гц. Частота 1024 Гц предназначена для звукового сигнала будильника и для опроса разрядов счетчиков при динамической индикации, выход частоты 32768 Гц - контрольный. Фазовые соотношения колебаний различных частот относительно момента снятия сигнала сброса продемонстрированы на **рис. 60**, временные масштабы различных диаграмм на этом рисунке различны. При использовании импульсов с выходов Т1 - Т4 для других целей следует обратить внимание на наличие коротких ложных импульсов на этих выходах.

Особенностью микросхемы является то, что первый спад на выходе минутных импульсов М появляется спустя 59 с после снятия сигнала установки 0 с входа R. Это заставляет при пуске часов отпускать кнопку, формирующую сигнал установки 0, спустя одну секунду после шестого сигнала поверки времени. Фронты и спады сигналов на выходе М синхронны со спадами импульсов отрицательной полярности на входе С.



Структура (а) и типовая схема включения (б) микросхемы К176ИЕ12

Рис. 59.

Подытожим найденную информацию. Если проанализировать структуру и диаграмму, то мы видим, что необходимы следующие входы/выходы модели:

F – с частотой 1024 Гц (11-й вывод – звук на будильник);

T1, **T2**, **T3**, **T4** – четырехтактный генератор 128 Гц для динамической индикации (выводы 3, 1, 15, 2);

S1 и **S2** – выходы секундных и полусекундных импульсов (выводы 4 и 6 соответственно);

C – вход счетчика-формирователя минутного импульса (7-й вывод).;

M – достаточно хитрый сигнал минутного импульса (10-й вывод);

K – выход с тактовой частотой 32768 Гц (14-й вывод) на диаграмме почему-то отсутствует);

Z и **Z**инв – для подключения кварцевого резонатора (выводы 12 и 13);

R – цепи сброса счетчика-делителя на 32768 (вывод 5) и счетчика-формирователя минутного импульса (вывод 9). Они разнесены и, в отличие от **CD4060**, не блокируют входной тактовый генератор на двух инверторах.

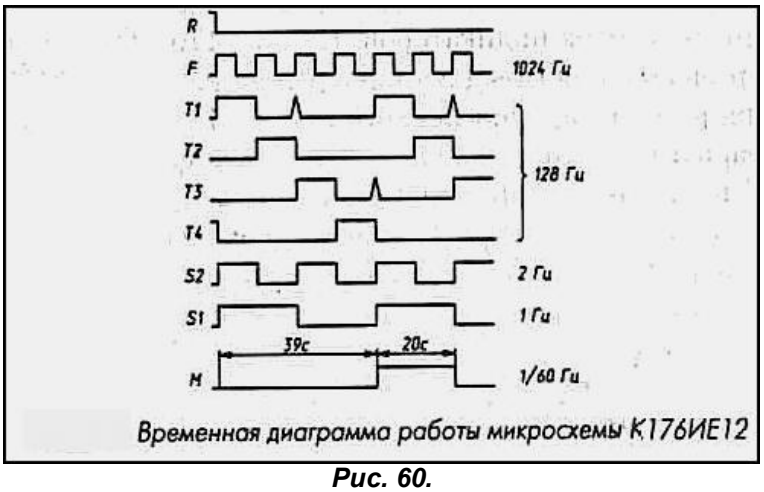


Рис. 60.

Еще одно отличие от **CD4060** в счетчике-делителе тактовой частоты. Для того чтобы поделить частоту 32768Гц до 1Гц нам потребуется на один разряд (счетный триггер) больше. Вы можете убедиться в этом, открыв приложенный в папке **CD4060_PLUS** проект **15_counter_gen.DSN**. Там для CD4060 задана нужная нам исходная частота и с помощью дополнительного к 14 разрядам микросхемы триггера мы получаем нужный нам сигнал 1 Гц. Обратите также внимание, что необходимые нам в соответствии с исходными данными сигналы 1024Гц и 2 Гц будут находиться соответственно на выходах 4-го и 14-го разрядов счетчика.

На основании этой информации и, руководствуясь типовой схемой включения, строим нашу графическую модель (файл вложения **Graphic_model.DSN** в одноименной папке). Здесь необходимо отметить, что входы сброса мне пришлось дополнительно пометить цифрами, чтобы не сбивать симулятор одноименными выводами модели. Вид получившейся графической модели представлен на Рис. 61. Тут есть два нюанса. Чтобы инверсный выход **Z** отображался с верхним надчеркиванием необходимо в имя вывода в его свойствах с двух сторон ограничить знаком доллара, т.е. в окне **Name** для вывода 13 должно быть так: **Z**. Кроме того, в модели присутствуют два скрытых вывода питания – **VDD** (сверху) и **VSS** (снизу).

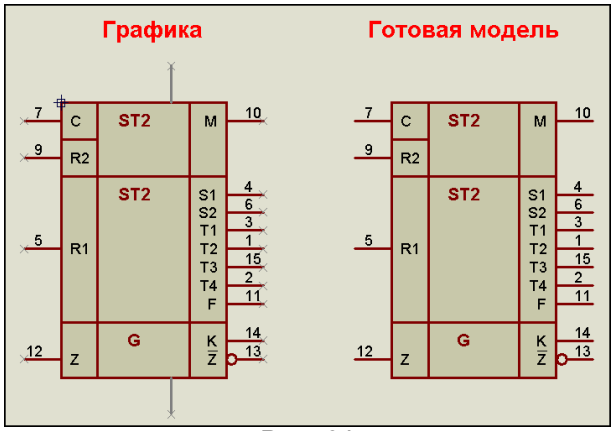


Рис. 61.

Обратите внимание, что выводы питания присутствуют только в графике, а после создания модели они исчезли (скрытые). На Рис. 62 на примере **VDD** показано – что необходимо установить в свойствах вывода перед созданием модели, чтобы он стал скрытым выводом питания.

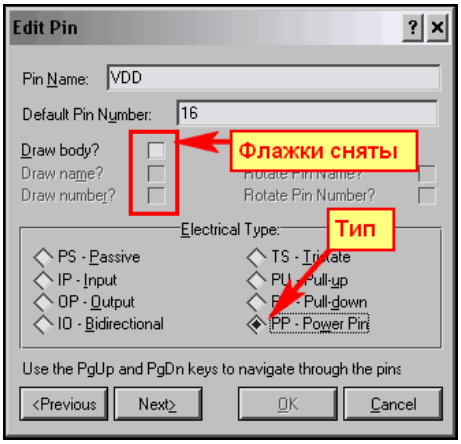


Рис. 62.

Но пока мы ничего с нашей графической моделью больше делать не будем. Отложим ее до лучших времен, а все наши промежуточные тестирования будем проводить с помощью **Subcircuit**, т.е. модулей. Почему? Да потому, что мне уже надоело то и дело писать, а вам – читать: **Make Device**. И крутимся мы без конца по этим пяти окнам вкладкам…. С модулями такие операции проводить не надо, но в тоже время они позволяют вести отладку в пределах одного проекта так же, как и в случае создания схематичной модели. Прогуляемся по второй из ссылок выше и найдем там схему RC-генератора на микросхеме **К176ИЕ12** (Рис. 63). Я ее поместил сюда, поскольку она нам тоже понадобится.

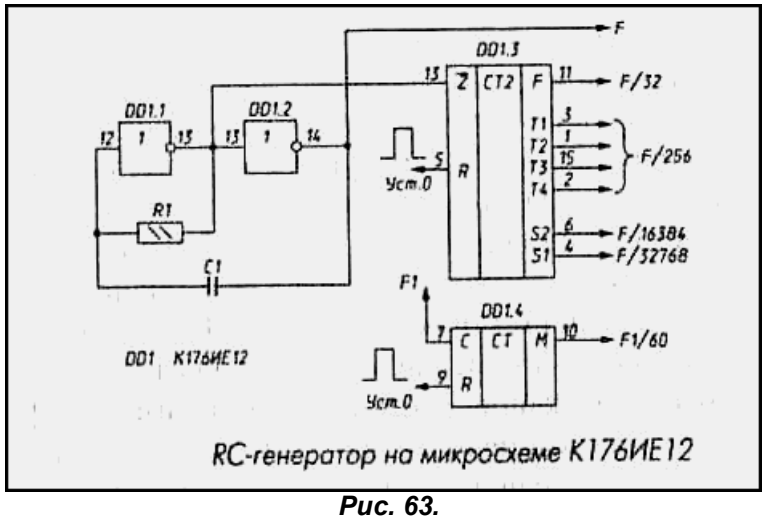


Рис. 63.

Для начала нам предстоит разобраться со встроенным генератором, ну и попутно приклеим к нему часть делителя на D-триггерах, чтобы проверить, как это будет работать в модели.

Возврат к содержанию

6.7. Пример создания полной схематичной модели счетчика К176ИЕ12. Часть 2 – встроенный генератор и делитель тактовой частоты.

Итак, первоначально воспроизведем входную цепь микросхемы, допустим первые четыре триггера делителя тактовой частоты. Я сначала делаю это обычно в отдельном дизайне, на главном листе, чтобы не париться с «прыжками» по листам туда-сюда. Нам потребуются цифровые примитивы **INVERTER** и **DTFF** из библиотеки **Modelling Primitives**. Кроме того, сразу же можно отыскать и втащить в проект цифровой генератор **CLOCK** из библиотеки **Simulator Primitives**. Мы его используем для тестирования, а в дальнейшем в модели для внутреннего генератора по аналогии с **CD4060**. Первый вариант для тестирования представлен на Рис. 64.

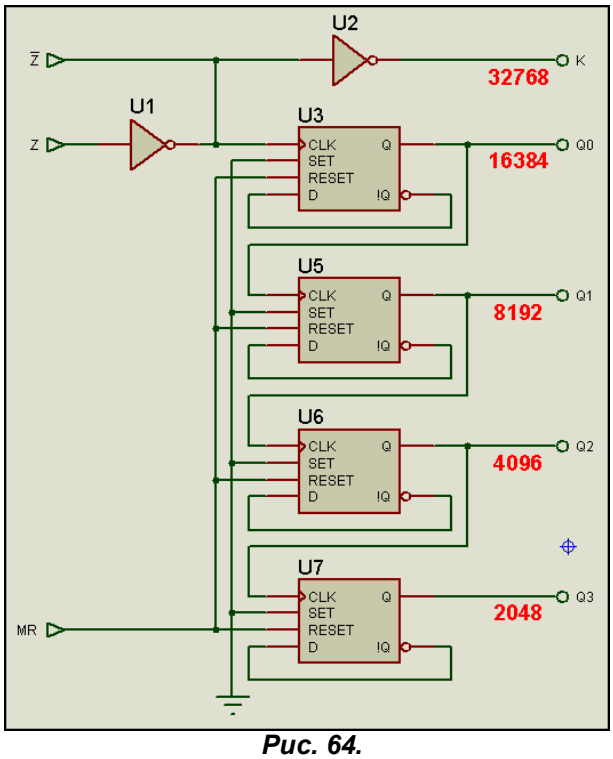


Рис. 64.

Я преднамеренно поставил у элементов в свойствах флажок **Hidden** напротив **Component Value**, воспользовавшись **Property Assignment Tools** (**PAT**). Для этого в окне **String** PAT набираем **VALUE**, в переключателе **Action** устанавливаем **Hide**, а ниже выбираем **On Click** и пробегаем щелчками левой кнопки по всем установленным компонентам. Кто еще не выучил PAT как «Отче наш…» - привыкайте. Сейчас у нас на схеме всего 7 элементов, а будет в несколько раз больше. И задавать каждому свойства вручную – каторжный труд. Еще я отключил опцию **Show Hidden Text** в меню **Temlate** => **Set Design Default**. Это позволило мне сэкономить пространство проекта, поскольку картинки становятся уже достаточно объемными и лишняя информация на них нам ни к чему. Итак, из литературы по ссылкам в предыдущем параграфе, а кто и по памяти, мы знаем, что каждый из D-триггеров в таком включении поделит тактовую частоту на 2. Т.е., подав на вход 32768Гц, мы получим сетку частот, выделенную красным на Рис. 64. Проверяем это в приложенном проекте **GEN_IE12_st_1.DSN**, а в проекте **GEN_IE12_st_2.DSN** проверяем функционирование нашего генератора в соответствии с Рис. 63 при внешней задающей RC-цепи. Оба проекта в папке **TEST1**. Во втором случае я воспользовался **IC=0** на инверсной цепи **Z** для запуска генератора. Теперь, когда мы убедились в успешной работе тестовых проектов, создаем модуль, а количество разрядов делителя на дочернем листе увеличиваем до требуемых пятнадцати. Как выглядит наш модуль на основном листе в действии показано на рисунке 65.

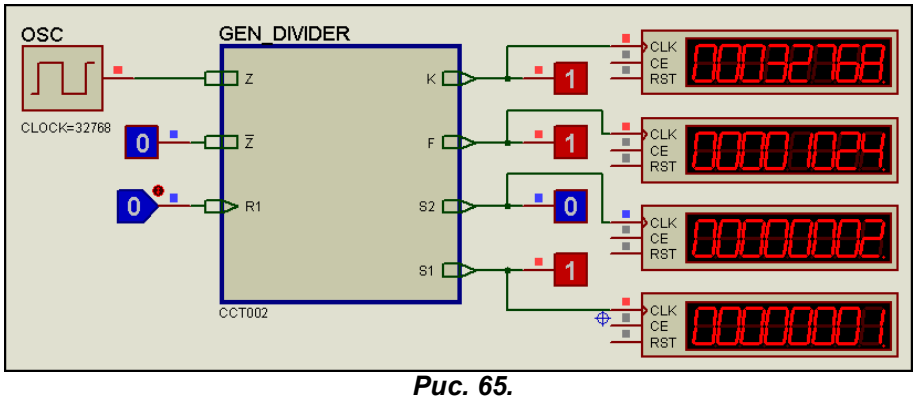


Рис. 65.

Пока наш примитив генератора **OSC** находится на основном листе. Мы получили требуемую сетку частот в соответствии с описанием микросхемы. На рисунке 66 приведены три цифровых графика работы модуля с различными временными интервалами.

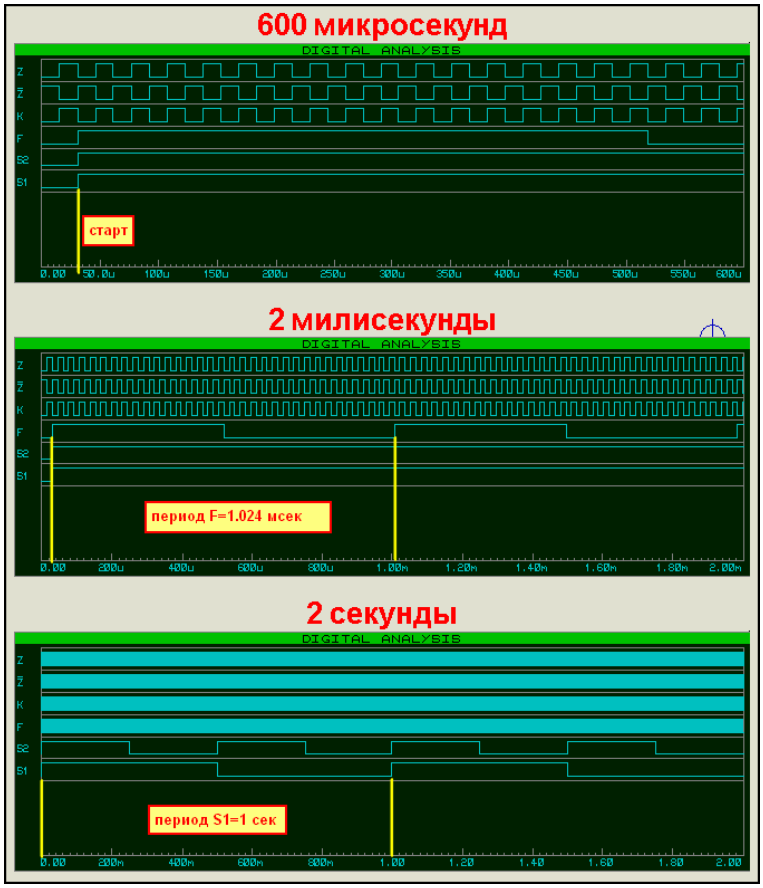


Рис. 66.

Для чего я привел эти графики? Очень важный момент, при создании сложных многофункциональных моделей – это совпадение фаз сигналов с их реальными прототипами. Если на секунду вернуться к временной диаграмме на Рис. 60, можно заметить, что передний фронт импульсов на выходах **F**, **S2** и **S1** должен совпадать, что я и проконтролировал по верхнему графику. Ну и попутно проверили периоды следования импульсов.

Я не буду здесь приводить дочерний лист с пятнадцатиразрядным счетчиком, потому что он займет слишком много места. Желающие посмотрят его самостоятельно в тестовом проекте **Modul_Gen.DSN** в папке **MODUL1** вложения. Мы же, убедившись в работоспособности делителя, идем дальше и начинаем процесс создания внутреннего генератора модели. Для этого помещаем наш примитив **CLOCK** с именем **OSC** на дочерний лист и подключаем его на вход инвертора **U1**. В свойствах генератора ставим галочку **Edit all properties as text** и вводим вручную следующие строки:

PRIMITIVE=<CLKOSC>
PRIMTYPE=DIGITAL!
CLOCK=<CLOCK>

Строка **INIT=0** там уже присутствует и в ней я просто убрал фигурные скобки, чтоб она стала видимой (Рис. 67).

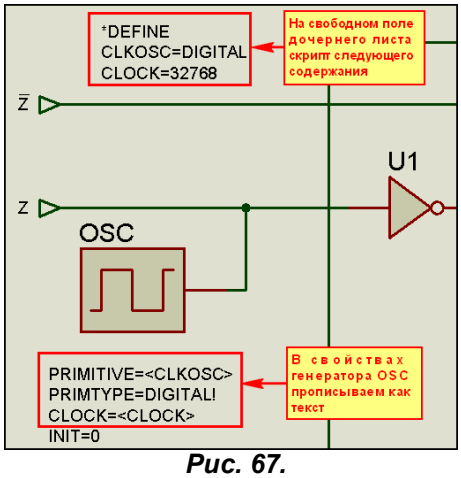


Рис. 67.

Кроме того, на свободном поле дочернего листа размещаем текстовый скрипт следующего содержания:

***DEFINE**
CLKOSC=DIGITAL
CLOCK=32768

Этот скрипт назначает нашему генератору свойства, принятые по умолчанию. Частоту я взял для стандартно используемого с К176ИЕ12 кварцевого резонатора 32768Гц. Теперь возвращаемся на основной лист и запускаем симуляцию. Если все сделано правильно, то наш модуль будет функционировать так же, как и с внешним генератором. Отдублируем модуль на этом же листе и зададим ему другое имя, чтобы не было совпадения и ошибки, например **GEN_DEVIDER_1**. Кроме того у копии модуля нам придется проследовать на дочерний лист и вручную назначить имя генератору тоже отличное от первого варианта, например **OSC1**. Теперь в свойствах модуля **GEN_DEVIDER_1** впишем строку **CLOCK=16386** и вновь запустим симуляцию (Рис. 68).

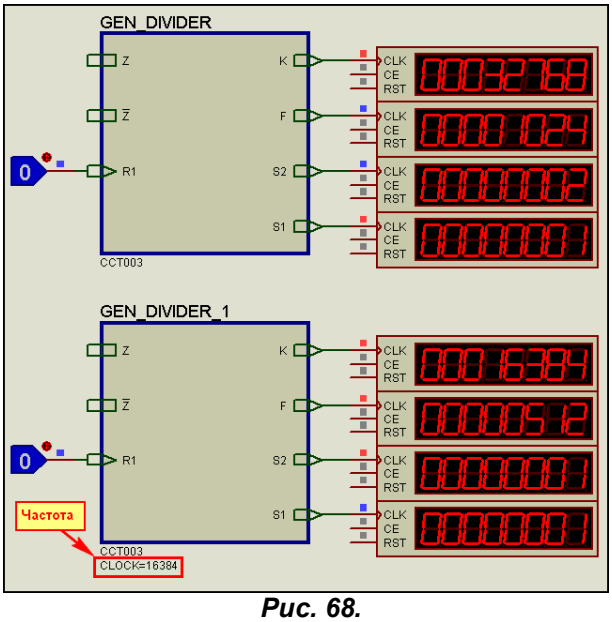


Рис. 68.

На всех выходах свежеиспеченного модуля частота упала вдвое. Ну, вот мы и получили управление частотой генератора модели с основного листа. Для чистоты эксперимента зайдем на дочерний лист первого модуля и выполним компиляцию MDF с этого листа. Я приложил получившийся файл **Modul_Gen_2.MDF** в папку **MODUL1_1** вместе с этим тестовым проектом **Modul_Gen_2.DSN**. Ниже приведено начало этого файла:

```
*PROPERTIES,2
CLKOSC=DIGITAL
CLOCK=32768

*MODELDEFS,0

*PARTLIST,18
OSC,CLOCK,,CLOCK=<CLOCK>,INIT=0,PRIMITIVE=<CLKOSC>,PRIMTYPE=DIGITAL!
U1,INVERTER,INVERTER,PRIMITIVE=DIGITAL
U2,INVERTER,INVERTER,PRIMITIVE=DIGITAL
```

Как и следовало ожидать, скрипт ***DEFINE** превратился в ***PROPERTIES**. Но самое главное, обратите внимание на первую строчку ***PARTLIST**. Сравните ее с аналогичной для генератора **OSC** модели **CD4060** и убедитесь, что они полностью совпадают. Мы на верном пути!
Теперь займемся свойством **CLKOSK**, которое определяет тип модели генератора **OSC**. По умолчанию оно равно **DIGITAL** - цифровой. Мы же на основном листе в свойствах модуля впишем ему значение **NULL** - пусто. Пробуем симуляцию и видим, что внутренний генератор перестал работать, что нам и нужно. Теперь попробуем запустить наш модуль с внешней RC цепочкой, задав свойство **IC=0** одной из внешних цепей. Наш модуль опять запустился и частота импульсов на выходе определяется внешними компонентами (Рис. 69). Этот пример приложен в проекте **Modul_Gen_2.DSN**, который расположен в папке **MODUL1_1**.
Таким образом, мы пришли к варианту построения генератора аналогичного модели **CD4060**. Нам осталось только расписать таблицу ***MAP ON** и заставить ее работать. Давайте окончательно воспроизведем вариант CD4060. Для этого изменим скрипт ***DEFINE** дочернего листа следующим образом:

```
*DEFINE
CLOCK=DEFAULT
```

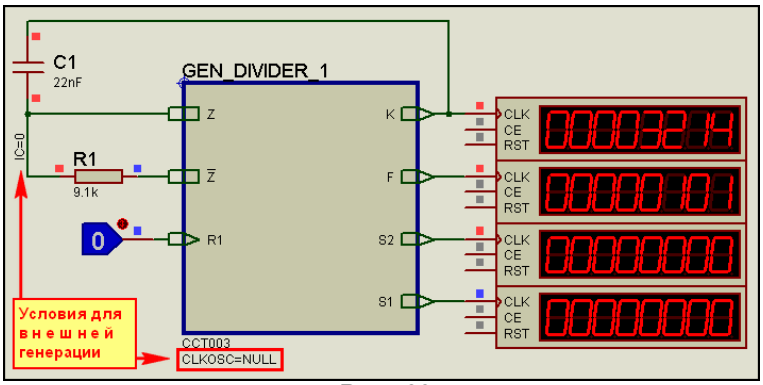


Рис. 69.

Где-нибудь на свободном поле листа поместим еще один скрипт:

```
*MAP ON CLOCK
DEFAULT : CLKOSC=DIGITAL
EXTERNAL : CLKOSC=NULL
```

Ну, а в свойствах модуля на основном листе впишем строку **CLOCK=32768**. Запустим симуляцию – работает. Для дубля модуля зададим **CLOCK=EXTERNAL** и навесим внешние RC элементы, не забыв задать **IC=0**. Тоже работает (Рис. 70). Вот теперь у нас почти полностью аналогичный генератор. Если мы задаем цифровое значение свойству **CLOCK** на основном листе, то включается внутренний генератор, а если задать ему значение **EXTERNAL** – то необходима внешняя RC-цепь или внешний генератор на вход **Z**. Таким образом, с помощью одного свойства компонента мы управляем как частотой внутреннего генератора, так и режимом его работы.

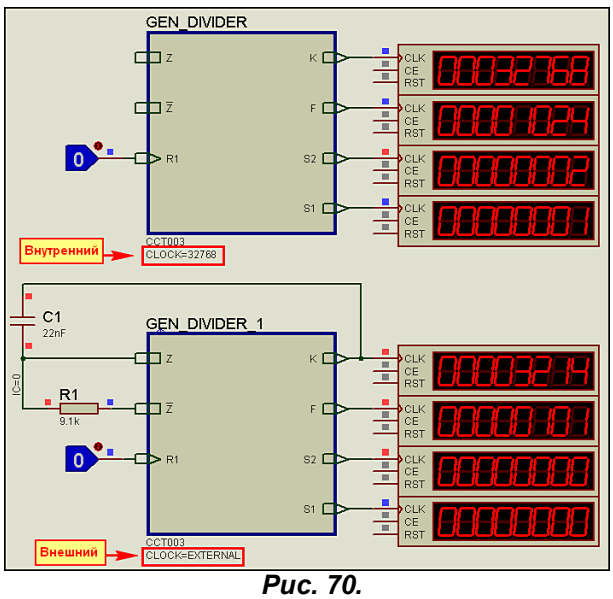


Рис. 70.

Этот пример в папке **MODUL1_2** под именем **Modul_Gen_4.DSN**. Ну и в заключение проверяем работу модуля при подаче сигнала от внешнего генератора – пример **Modul_Gen_5.DSN**. Вот и весь материал, который я хотел довести до вас в этом параграфе. Но наиболее внимательные уже, наверное, заметили, что я тихо замолчал про параметр **CLKGATE**, который имеется в **MAP ON CLOCK** для модели **4060**. Честно говоря, я так и не понял до конца – зачем он там присутствует, поскольку упоминается только в таблице и далее нигде не встречается. Наличие или отсутствие его особого значения на работу модели не оказывает, в чем мы убедились, «забыв» включить его в карту для модуля **GEN_DIVIDER**. Есть у меня подозрение, что **CLKGATE=NULL** команда симулятору **PROSPICE** не создавать виртуальный генератор для модели при использовании режима внешнего. Возможно, таким образом, автор модели **4060** преследовал цель снизить нагрузку на ЦП компьютера в режиме внешнего генератора модели. Если попадет под руку какой-нибудь доходяга Пентиум III, попробую проверить, но на тех монстрах, которые сейчас в наличии эффектов не заметно. Наиболее сложная в понимании часть будущей модели **К176ИЕ12** на этом завершена. Нам осталось сформировать генератор для динамической индикации и формирователь минутного импульса. Об этом далее…

Возврат к содержанию

6.8. Пример создания полной схематичной модели счетчика К176ИЕ12. Часть 3 – формирование сигналов динамической индикации и минутного импульса.

Четырехтактный формирователь сигналов управления динамической индикацией, пожалуй, самый простой узел внутренней структуры **К176ИЕ12**. Если проанализировать временную диаграмму на рисунке 60, мы увидим, что это последовательная цепочка импульсов с частотой 128Гц на выходах **T1**, **T2**, **T3**, **T4** микросхемы. Из литературы нам известно, что для таких целей наиболее часто используют кольцевые счетчики 1 из n на регистрах сдвига, замкнутых в кольцо. Так мы и поступим. Поскольку это модель, а не реальное устройство схема самовосстановления режима работы такого счетчика будет излишеством, а для того чтобы обеспечить предустановку одного из триггеров в единичное состояние при старте симуляции воспользуемся стандартным для триггеров свойством **INIT**. Кто забыл – смотрите HELP для примитива D-триггера. Для первого из четырех триггеров установим **INIT=1**. Ну и еще один нюанс. Нам нужна частота импульсов 128Гц, триггеров – четыре. Включаем в голове математику, а кто не изучал – арифметику. Тактовая частота для регистра сдвига должна быть в 4 раза выше, т.е. 512Гц. Проверяем наш регистр в отдельном проекте – **128Hz.DSN** (вложение папка **4Takt**). Схема формирователя и график его переключения представлены на Рис. 71.

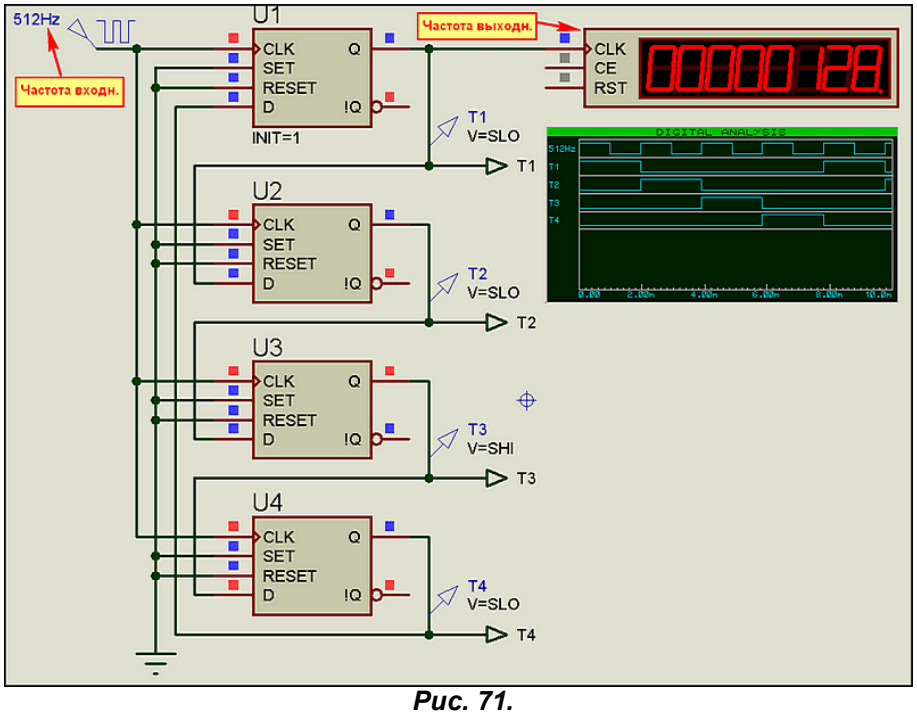


Рис. 71.

Нам осталось только поместить данный формирователь на дочерний лист модуля, созданного ранее, присоединить его к выходу делителя с частотой 512Гц – это выход элемента **U8** в делителе и привести в порядок нумерацию элементов, чтобы не было конфликта. Кроме того, добавим на основном листе нашему модулю соответствующие выходы **T1…T4**. Чтобы добавить реальности при тестировании, введем начальный сброс модуля. Для этого на его вход **R1** я подал цифровой сигнал перепада с 1 на 0 с помощью генератора **Digital Edge** (**DEDGE**) из левого меню **Generator Mode** с параметрами, представленными на Рис. 72.

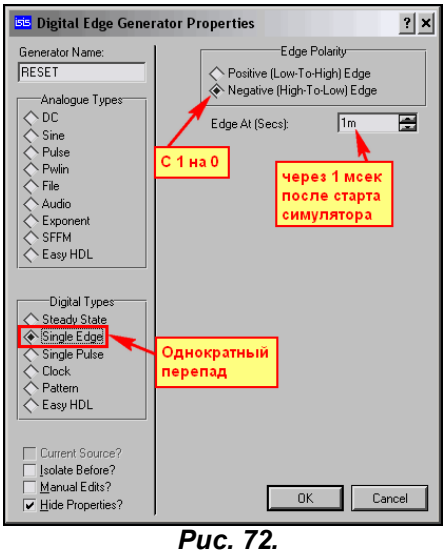


Рис. 72.

Это позволит нам более четко отслеживать поведение нашей модели. Тестируем новый вариант модуля в интерактивном режиме – все нормально, на выходе **T1** присутствуют требуемые 128Гц. Теперь проведем тест с помощью цифрового графика (Рис. 73), и тут же обнаруживается несоответствие временной диаграмме из рисунка 60.

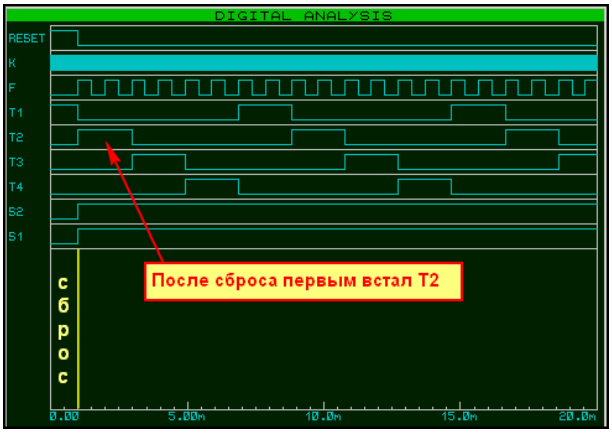


Рис. 73.

Я нарочно оставил это проект под названием **Modul_2_bad.DSN** во вложении **MODUL2**, чтобы вы могли сравнить его с хорошим **Modul_2_good.DSN**. Разберем причину моего косяка и быстро устраним. Согласно рисунку 71 я поставил предустановку **INIT=1** для первого триггера, т.е. в момент времени 0 – он установлен в единицу и по первому же положительному перепаду на счетных входах **CLK** нашего регистра она уедет во второй триггер с выходом **T2**, что мы и получили на графике рисунка 73. Ошибка настолько очевидна, что правится в шесть секунд. Переносим **INIT=1** из первого триггера формирователя в четвертый и снова тестируем (Рис. 74). Статус-кво восстановлен, и все соответствует документации.

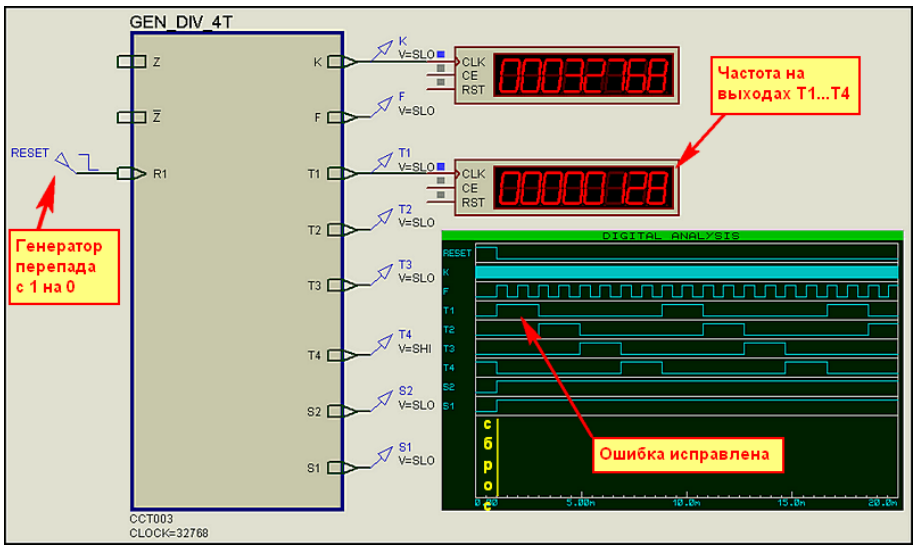


Рис. 74.

Фактически, нам осталось сформировать минутный импульс, и наша модель будет почти готова. Тут тоже придется немного «попотеть», поскольку импульс довольно хитрый. Согласно временной диаграмме передний его фронт возникает по прошествии 39 сек, а длительность составляет 20 сек, т.е. после 59 сек мы должны иметь на выходе **M** опять ноль. Эту часть структуры **К176ИЕ12** для проверки мы также соберем сначала в отдельном проекте. Не будем мудрствовать и сформируем эту часть все на тех же D-триггерах по схеме суммирующего счетчика. Десятичное число **60** эквивалентно двоичному – **111100**, т.е. потребуется 6 разрядов (триггеров). Кроме того, придется входы **RESET** раздвоить по **OR** (ИЛИ), так как нам необходимо иметь сброс и от внешнего сигнала и при достижении отсчета 60 секундных импульсов (1 минуты). В качестве формирователя длительности импульса используем все тот же примитив **DTFF**, но уже как обычный RS-триггер. На вход **S** собираем через примитив **AND** (И) число 39 (двоичное значение **100111**), на вход **R** - число 59 (двоичное – **111011**). В нашем случае это допустимо, т.к. указанные числа имеют значительный разброс значащих единиц, но в некоторых случаях такая логика работы входов невозможна и приходится использовать инверсию незначащих нулей. В цифровых примитивах Протеуса это достигается добавлением для соответствующего входа строки **INVERT=Dx** в окне свойств, где **x** – номер входа (счет ведется с нуля, т.е. верхний вход пятивходового элемента **AND** будет **D0**, а нижний **D4**). Схема достаточно громоздкая, но я ее поджал за счет применения шины и приведу полностью на Рис. 75 . Во вложенном в папку **MODUL3** файле **Counter60.DSN** представлен тестовый проект.

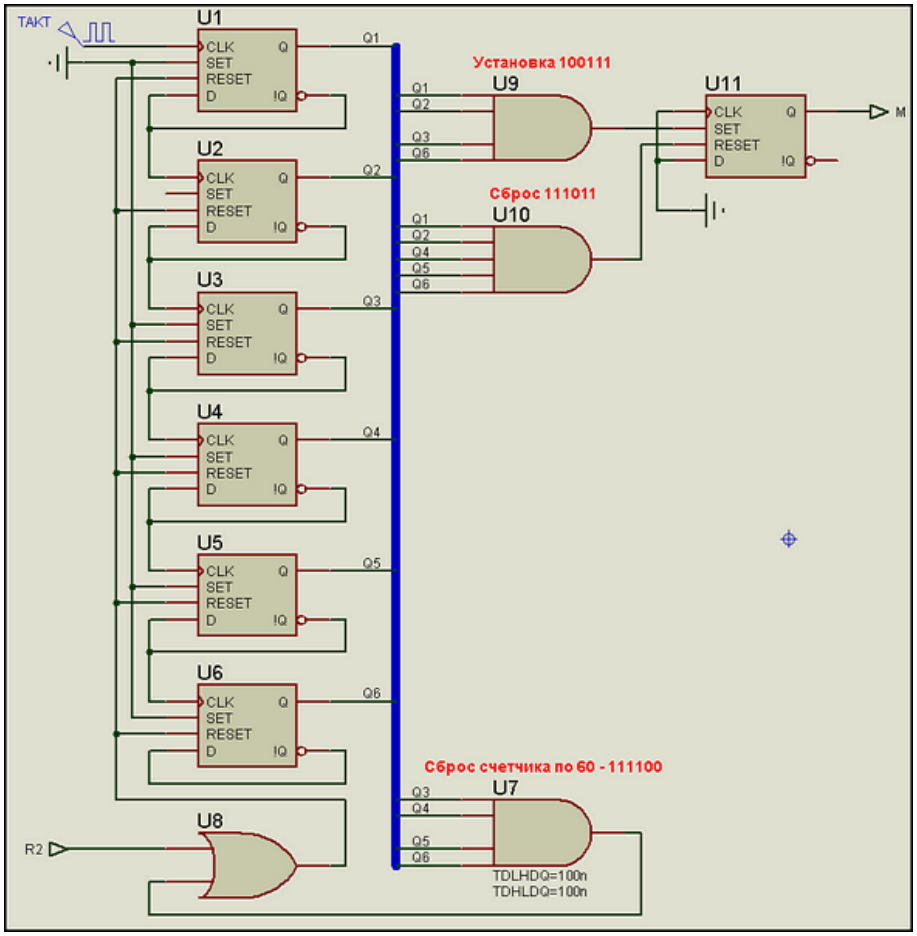


Рис. 75.

Обратите внимание, что для логического элемента **U7 (**AND_4), формирующего импульс сброса счетчика по достижении числа 60 установлены свойства **TDLHDQ=100n** и **TDHLDQ=100n** (нарастание и спад импульса по 100 наносекунд). Иначе он просто не будет виден на графике, поскольку слишком короткий. График работы схемы представлен на рисунке 76.

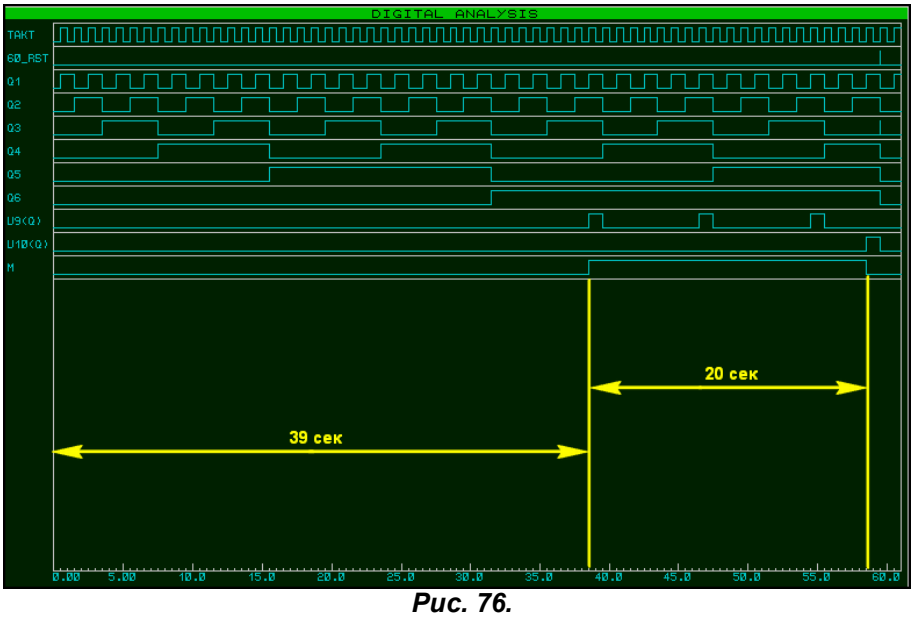


Рис. 76.

Если запустить симуляцию тестового проекта, то, манипулируя кнопкой **Gate Polarity** прибора **Counter Timer** можно измерить длительность и паузу минутного импульса. Генератор я уже поставил на 1 Гц для имитации реальных условий. Правда, занятие это в реальном времени утомительное, приходится честно высиживать минуту, наблюдая за таймером.
Нам осталось перенести формирователь минутного импульса в дочерний лист модуля и добавить вход **R2** и выход **M** к нему на основном листе. Я проделал эту манипуляцию в проекте **Modul_3.DSN** соответствующей папки вложения. Там же представлены графики с различными временными интервалами, на которых видны те или иные выходные сигналы модуля.
Нам же осталось для придания нашей будущей модели более реальных характеристик на дочернем листе для каждого элемента задать соответствующие временные задержки фронтов. Вот тут-то и вспомним добрым словом разработчиков программы Протеус. В финале структура содержит 32 цифровых элемента. Нам предстоит каждому задать **TDLHDQ**, **TDHLDQ** и т.д. Как работенка – впечатляет? Но на помощь опять приходит **PAT**. Поскольку и здесь мы воспользуемся аналогией с моделью **4060**, то в конечном итоге сведем все это к таблице **MAP ON**. Начнем с того, что зададим всем триггерам **TDHLCQ=<TDCQ>** и **TDLHCQ=<TDCQ>**. Вызываем окно **Property Assignment Tools** и набираем в окне **String** строку **TDHLCQ=<TDCQ>** (Рис. 77).

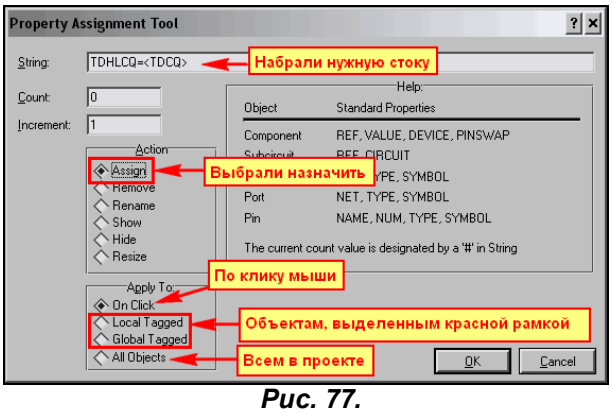


Рис. 77.

Если не хотите, чтобы это значение подсвечивалось под каждым элементом, поставьте по краям фигурные скобки. Но мне в данном случае это не нужно, я хочу себя контролировать. В рамке **Action** нам предстоит действие назначить, т.е. **Assign**. Дальше у нас три пути:

- Можно стандартно выбрать **On Click** и честно прощелкать мышью все нужные компоненты. Процедура, которую многие уже наверняка освоили. Наводим до появления «указательного перста» с зеленым прямоугольником справа и щелкаем левой кнопкой мыши. 26 триггеров – столько же щелчков. Шурик в «Кавказской пленнице» пожалел птичку, ну а мне – мышку жалко…
- Можно заранее выделить (обвести, удерживая нажатой кнопку мыши) район с нужными элементами, а лишь потом вызвать PAT. Причем он сразу предложит в рамке **Aply To** действие **Global Tagged**, т.е. для всех выделенных красным компонентов. Кстати, я так и не разобрался, чем отличается **Local Tagged** – все равно присваивается выделенным. Это уже более прогрессивный метод и за один раз при правильном построении схемы (похвалюсь – как у меня в данном случае) можно зацепить достаточно много объектов.

- Ну, и наконец, можно выбрать **All Objects**, но тут надо быть предельно осторожным. В частности, при использовании на дочернем листе модуля мы присвоим это свойство и всем объектам на основном листе – оно нам надо? Такие вещи хороши, когда питание переназначаешь, вроде **VCC** или **VSS**, но зачем примитиву генератора **TDHLCQ**? А ведь назначит, я проверял. Еще удобно, когда надо скрыть/показать что-то у всех объектов – этим мы воспользуемся позже.

Ну, в общем, в данном случае мне идеально подходит для назначения второй вариант. Я выделяю часть триггеров, стоящих в одной колонке и вызываю **PAT**. Ввожу строку как на рисунке 77 и давлю **OK**. Под соответствующими триггерами сразу появляется видимая строка (Рис. 78). Вся операция прошла в три действия, причем в последнем участвовал только один триггер минутного импульса. Последняя введенная строка сохраняется в **PAT**, и набирать мне ее пришлось только один раз.

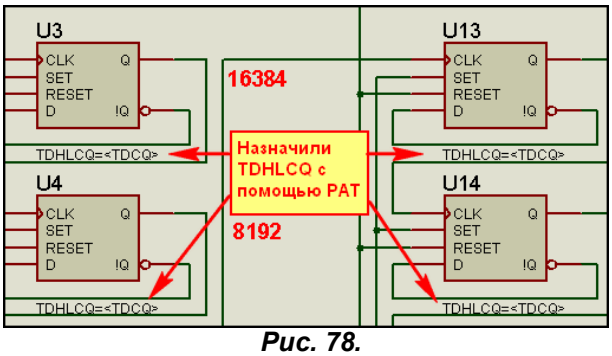


Рис. 78.

Но присмотритесь внимательно – мне ведь еще пару свойств прописывать – а куда? Со свободным местом в проекте – полный напряг. Но я не зря затеял практический повтор материала по **PAT**. Цитирую опять, пусть и не дословно, классику комедии – «Кавказскую пленницу»: «Тот, кто нам мешает – тот и поможет…», т.е. **PAT**. Для начала надо визуально убедиться, что я назначил **TDHLCQ** всем триггерам. Ну а больше мне на него любоваться незачем. Вызываю окно PAT, в **String** оставляю только аббревиатуру **TDHLCQ**, а в **Action** выбираю **Hide** – скрыть. И вот теперь в рамке **Aply To** выбираю **All Objects** – пригодилось! Все, зарыл **TDHLCQ**. Но в свойствах триггеров, если выбрать соответствующий пункт оно видно (Рис. 79). Видно его и в режиме **Edit all properties as text**.

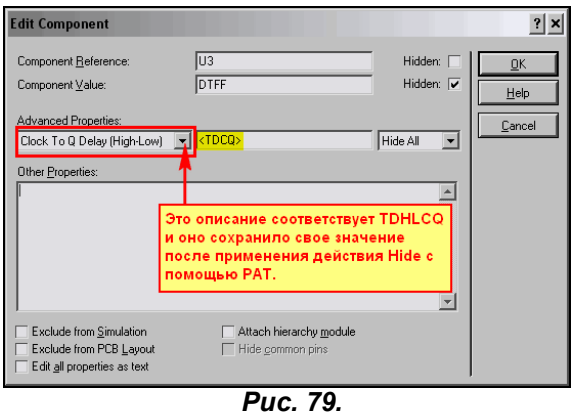


Рис. 79.

А то, что я применил **Hide** ко всем объектам в данном случае не криминально, поскольку это свойство есть только у счетчиков и триггеров и сработало оно только для них. Аналогичным образом всем триггерам назначаем и **TDLHCQ=<TDCQ>**, а затем и **TDRQ=<TDMRQ>**. Ну уж будем последовательными и всем логическим элементам в проекте назначим **TDHLDQ=<TDOSC>** и **TDLHDQ=<TDOSC>**.

Ну и, закончив все вышеуказанные операции, осталось на свободном поле дочернего листа поместить скрипт следующего содержания:

```
*MAP ON VOLTAGE
5V : TDOSC=35n, TDCQ=25n, TDMRQ=100n
10V : TDOSC=13n, TDCQ=10n, TDMRQ=40n
```

Пока я нахально слизал задержки из модели **4060**, чтоб протестировать нашу модель, но позже подумаем, как и сделать более реальными. Снова тестируем наш модуль. Опля, что-то не то. Ну да, забыл в свойствах модуля на основном листе прописать напряжение в соответствии с только что написанной картой. Добавляем там строчку **VOLTAGE=10V**, и все становится на свои места. Этот пример в файле **Modul_4.DSN** папки **MODUL4** вложения. Ну вот, теперь у нас все подготовлено и протестировано, осталось только закончить модель. Об этом в следующем параграфе.

Возврат к содержанию

6.9. Пример создания полной схематичной модели счетчика К176ИЕ12. Часть 4 – создаем MDF и законченную Schematic модель.

Наступила пора извлечь из небытия нашу графическую модель **К176ИЕ12**, созданную в **п.6.6**. Помещаем ее в поле проекта и в свойствах устанавливаем флажок **Attach Hierarchy Module**. Заходим на дочерний лист и втаскиваем туда (меню **File=>Import Section**) секцию, которую создали с дочернего листа предыдущего проекта **Modul_4.DSN**.

Да, давно это было… склероз, однако. Придется кое-что еще поправить на дочернем листе. В тестовом проекте **Modul_4** вход формирователя минутного импульса напрямую соединен с выходом **S** делителя, а в реальности это отдельный вход **C** у графической модели. Ну, это поправимо и сейчас – разрываем и добавляем на дочернем листе терминал **C** на вход минутного формирователя (Рис. 80).

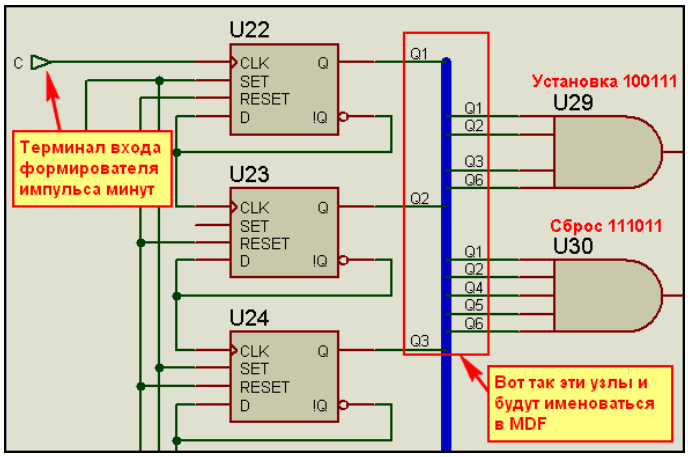


Рис. 80.

Возвращаемся на родительский лист и заходим в свойства модели. Там в первую очередь нам необходимо прописать вручную частоту и напряжение так, как мы это делали в предыдущем проекте для модуля. Иначе, при запуске симуляции вылетит «красная птичка». Также, если мы ранее не назначали нашей модели префикс и **VALUE**, то тоже пора дописать (Рис. 81), хотя пока это и не обязательное требование.

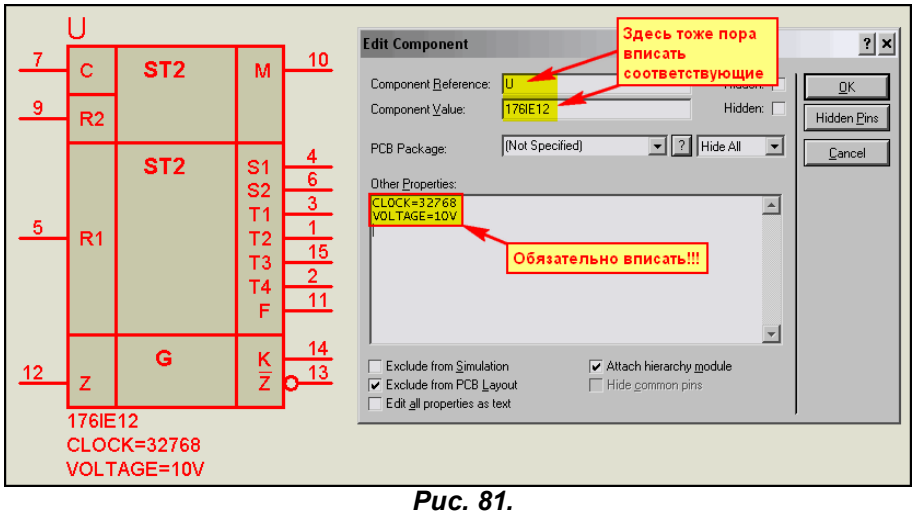


Рис. 81.

Теперь соединяем **S1** с **C** и заземляем входы сброса **R1** и **R2**, на выходы вешаем зонды и запускаем симуляцию. Если ранее нигде не накосячили, все прекрасно работает. Останавливаемся и. … А вот и горчичнички (Рис. 82).

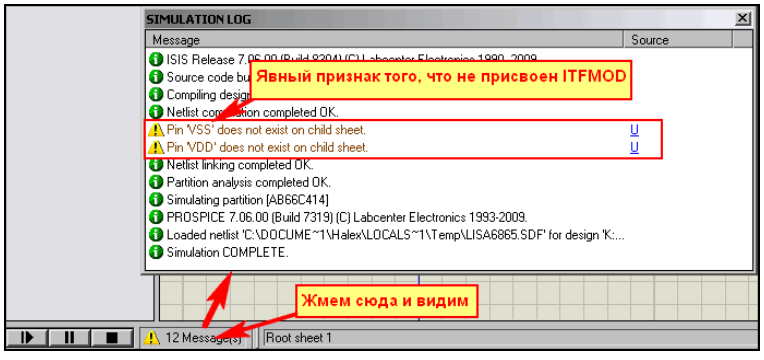


Рис. 82.

Ну конечно, про **ITFMOD** мы забыли. А ведь при создании графики мы прилепили скрытые пины питания **VSS/VDD**. Вот симулятор нас и предупредил, что они отсутствуют на дочернем листе. Но мы их и не собираемся там рисовать, а просто добавим в свойствах одну строку: **ITFMOD=CMOS**. Пробуем еще раз – все, проблема со специей (**SPICE**), т.е. горчицей разрешилась (как-бы каламбур). Я оставлю эти два примера **Graphic_with_Child** с окончаниями **bad** (плохой – без ITFMOD) и **good** (хороший – с ITFMOD) в папке **Graphic_and_Hierarchy** вложения. Результаты теста на рисунке 83.

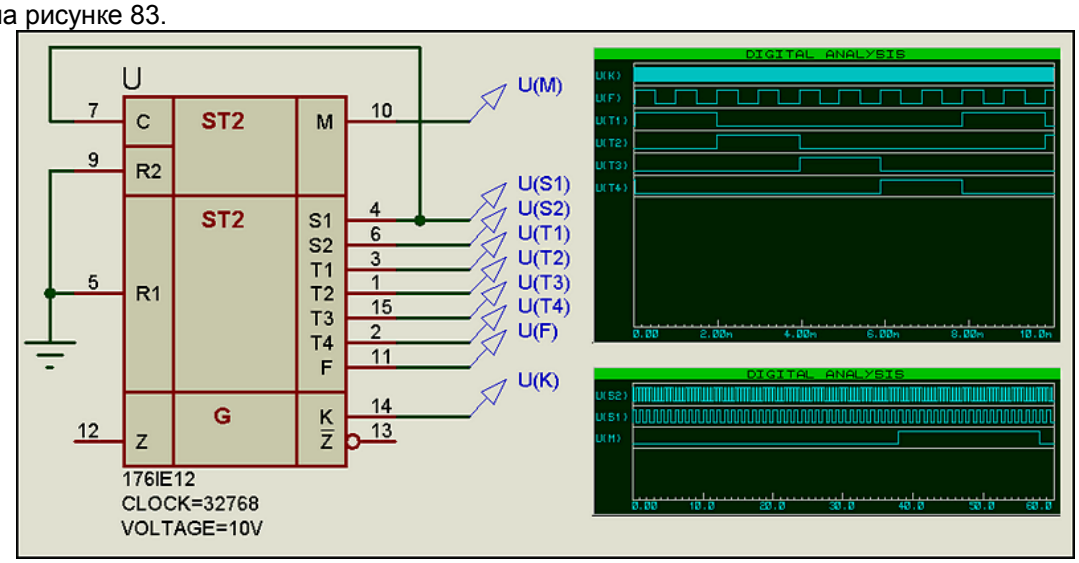


Рис. 83.

Поскольку тест прошел удачно, и все наши графики приближены к реальности, можно приступить к формированию MDF. Но прежде необходимо до конца разобраться с задержками, ведь у нас по-прежнему они используются от модели **4060**. Когда в **п.6.2** мы пробовали создать модель **176ЛА7**, я упоминал, что стандартные задержки распространения для этой серии при питании 9В равны 250 наносек. Это все, что мне удалось найти в справочниках. Ну что, давайте отталкиваться от этого. Заменим в карте **MAP ON VOLTAGE** на дочернем листе задержки следующим образом:

```
*MAP ON VOLTAGE
5V : TDOSC=400n, TDCQ=520n, TDMRQ=1200n
10V : TDOSC=250n, TDCQ=325n, TDMRQ=750n
```

Чтоб не возникало кривотолков – поясню. Для 10(9)V я взял стандартную задержку распространения для логических элементов - в нашей модели это **TDOSC**. Для 5V она увеличится и где-то мне попадалось значение 400 наносек. Значения для триггеров **TDCQ** и **TDMRQ** я просто взял и просчитал на калькуляторе пропорционально от модели 4060. Понимаю, что кривлю душой, но садиться с осциллом замерять реальные – мне, честно говоря, ну ни в жилу. Да и нет у меня пачки микросхем, а с единичных экземпляров – все равно будет лажа. Так что, примите – как есть. В общем, тестируем с этими задержками и убеждаемся, что никаких излишних эксцессов не вылезает. Протестировать придется и при **VOLTAGE=5V**, чтобы быть до конца уверенными в себе. Этот проект **Model_with_child.DSN** представлен в папке **Model_and_MDF** вложения.

Теперь у нас все готово к компиляции MDF, но прежде одно маленькое замечание, которое предшествует небольшому сюрпризу для вас. Обратимся снова к рисунку 80. Помните, я для ужимания схемы прибегнул к применению шины? Соответственно проводникам, присоединенным к шине, пришлось назначить соответствующие метки (лейблы). Так вот, при компиляции MDF эти метки и превратятся в узлы для этих цепей. Те цепи, которые не помечены и не присоединены к входным/выходным терминалам будут просто автоматически пронумерованы компилятором типа **#00001**, **#00002** и т.д. Цепи, входящие в шину и помеченные с помощью лейблов в **NETLIST** будут выглядеть так:

```
Q1,4
Q1,LBL
U30,IP,D0
U29,IP,D0
U22,OP,Q
```

Это пример узла с меткой **Q1** – выход триггера **U22**. На работоспособность модели это не повлияет, но если кого-то смущает данный момент – придется перерисовать дочерний лист, исключив из него шину, а все соединения сделать отдельными проводниками.

Заходим на дочерний лист и вызываем компилятор через меню **Tools=>Model Compiler**. Указываем путь сохранения к нашей текущей папке проекта, а также меняем имя файла, чтоб совпадало с именем модели – **176IE12**, хотя это и необязательно, но так меньше путаницы (Рис. 84).

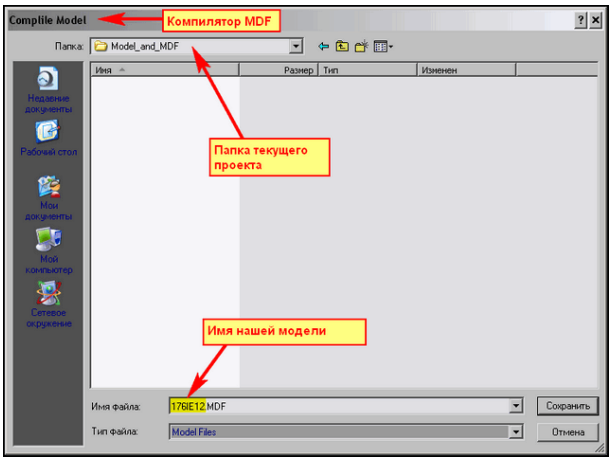


Рис. 84.

Готовый файл **176IE12.MDF** лежит в папке **Model_and_MDF** вложения. Теперь настала пора присоединить его к графической модели. Я не стану для этой цели использовать уже находящуюся в поле проекта графику с присоединенным **Child Sheet**. Причин несколько. Во-первых, я хочу оставить ее вам для сравнения. Во-вторых, там уже прописаны вручную свойства CLOCK, VOLTAGE и ITFMOD, а я хочу показать их присоединение с нуля. Ну и, в-третьих, эти уже присоединенные свойства все-равно пришлось бы править на третьей вкладке **Make Device**. Поэтому, мы начнем с голой графики, а для этого используем еще одну графическую модель на этом же листе проекта. Выделяем ее и запускаем незаслуженно подзабытую нами **Make Device**.

На первой вкладке, если еще не прописано, вставляем **Device Name** – **176IE12** и **Reference Prefix** – **U**. Я не стал здесь оригинальничать и оставил префикс, принятый в **ISIS** по умолчанию для микросхем. Лишний раз напомню, что здесь мы можем применить только английский язык. Ни какого русского языка Протеус здесь не потерпит (Рис 85). Вторую вкладку можно спокойно пропустить, там нам исправлять и добавлять нечего. На третьей вкладке нам предстоит добавить сразу несколько свойств. Начнем с **CLOCK**. Выбираем через кнопку **New** опцию **Blank Item** – (чистая позиция) и заполняем ее в соответствии с рисунком 86. **Name** должно быть только **CLOCK** и никаким другим. Именно так оно прописано у нас в скриптах на дочернем листе модели, а теперь уже и в файле **MDF**. В графе **Description** (описание) допустим русский язык. Я ввел фразу: **Частота (Внешн.=External)**. Именно так она будет отображаться в окне свойств. Тип переменной Type я оставил по умолчанию, т.е. **String**. При этом, отсутствуют какие либо ограничения по вводу значений в эту строку. В прототипе **4060** используется тип **Float** и ограничение **Positive, Non-Zero**, т.е. действует защита от ввода отрицательного и нулевого значения. Возможно, и есть смысл сделать аналогично, но я надеюсь, что в России хотя бы одной бедой когда-то станет меньше, и отрицательные значения частоты туда писать не будут. Следующий **Type**, относящийся к окошку ввода оставляем **Normal**, т.к. нам необходимо будет окно доступное для записи значения частоты. А вот **Default Value** (по умолчанию) я в последний момент решил поменять на **External**. Причина все та же, не очень надеюсь, что наши пользователи запомнят, как пишется это слово. И, хотя я и ввел подсказку в описании, но так надежнее, да и числовое значение вбить на это место всегда быстрее, даже полному чайнику в расположении символов на клавиатуре.

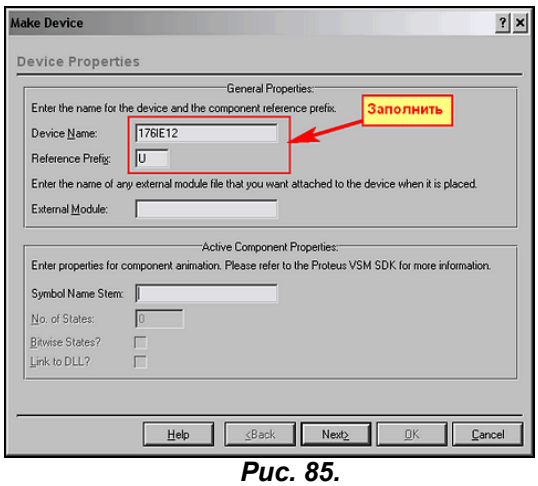


Рис. 85.

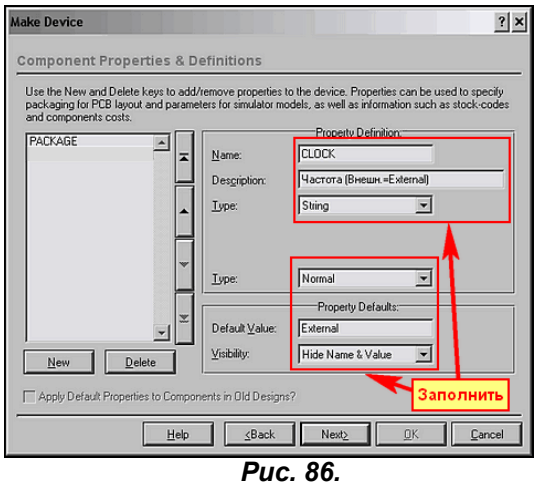


Рис. 86.

Снова давим кнопочку **New => Blank Item**. При этом свойство **CLOCK** запомнится в окне свойств модели, а нам предстанет чистое окно для заполнения следующего с именем **VOLTAGE** (Рис. 87). Заполняем его аналогично. В описании я ввел фразу **Задержка для питания**. Тип переменной в данном случае **Keyword (Non-Editable)** (ключевое слово – нередактируемое). Здесь уместна «защита от дураков», ведь упорно будут пытаться вписать 9V – пусть помучаются. В появившемся окне Keywords вводим наши возможные значения (те, что в **MAP ON VOLTAGE** в начале строк до двоеточия). Значения разделяем запятой без последующего пробела. Ну и по умолчанию **Default Value** выставляем 10V (наиболее близкое к стандартному питанию).
И опять щелкаем **New**, но на этот раз выбираем стандартное свойство из раскрывающегося списка ITFMOD. Здесь просто в **Default Value** задаем ему значение **CMOS**, а все остальное уже заранее заполнено в шаблоне (Рис. 88).

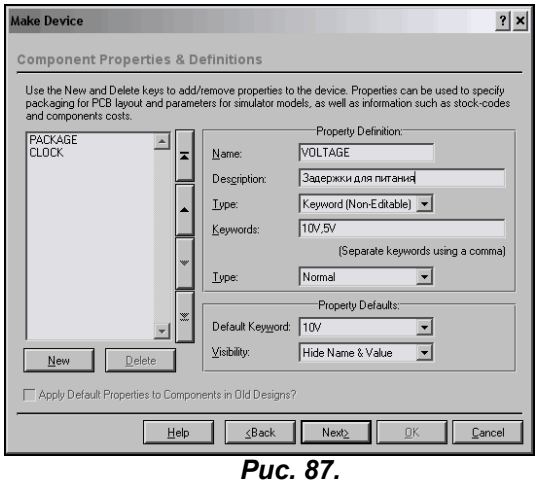


Рис. 87.

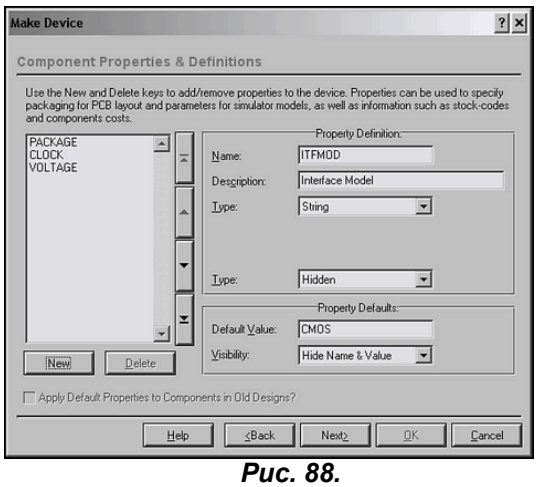


Рис. 88.

И в последний раз кликаем **New** и выбираем из списка **MODFILE**. Тут в окошке **Default Value** вводим имя нашего файла **176IE12.MDF** (Рис. 89) и на этот раз давим кнопку **Next**. Со свойствами мы покончили. На последней вкладке вводим данные так, как нам хочется (Рис. 90) и сохраняем нашу модель пока в библиотеке **USRDVC**.

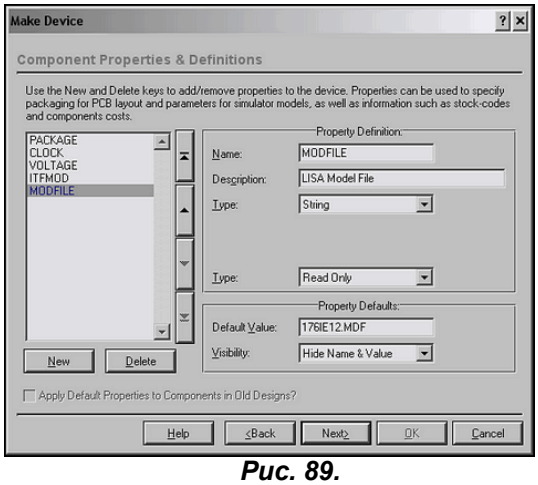


Рис. 89.

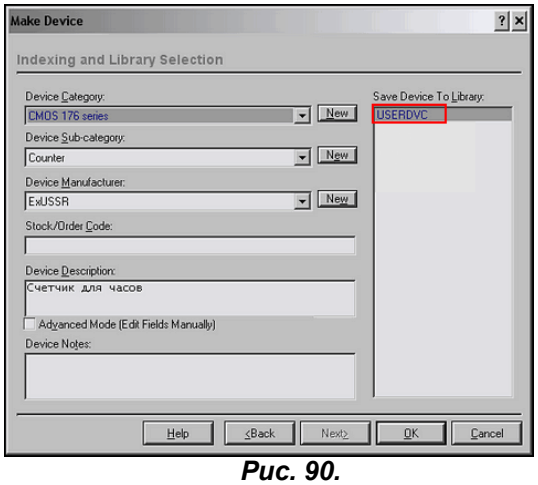


Рис. 90.

Ну, вот и все, наша модель готова к использованию. Теперь необходимо поместить файл **176IE12.MDF** в папку **MODELS** Протеуса, чтобы он был доступен из любого нового проекта. Создаем тестовые проекты и проверяем работу микросхемы **176IE12**. У меня во вложении это: **TEST_176IE12_int.DSN** для внутреннего и **TEST_176IE12_RC.DSN** для RC-генератора в папке **Model_and_MDF**. На этом наши мучения с созданием модели счетчика **К176ИЕ12** закончены.
В заключение несколько слов о путях дальнейшего развития этой модели. Первоначально у меня были замыслы использовать свойство **SCHMITT** для элемента **U2** по схеме дочернего листа для автоматического запуска генератора при наличии внешних цепей и переводе в режим **External**. Однако если вы внимательно изучали материал, посвященный генераторам в **п.6.3** и **6.4**, то обратили внимание, что при использовании этого свойства частота получаемого RC-генератора значительно ниже, чем при **IC** и **PRECHARGE**. Причем она настолько ниже, что вообще не стыкуется с расчетными данными, полученными по формулам из справочников по цифровым устройствам. Поэтому от этой затеи я отказался. Попытка втащить свойство **IC** внутрь модели тоже была обречена на провал. Наличие IC предусматривает наличие аналоговой цепи внутри модели – хоть пресловутый резистор на 1 миллиОм, а надо врезать в эту цепь. При этом сразу падает быстродействие модели, а оно у нас и так невелико – при задирании частоты внутреннего

генератора выше 600кГц или 1,2 -1,3 кГц с RC мой Core Quad с 2500ГГц тактовой и 4(3,2) Гбайт мозгов благополучно грузится на все 100% и начинает бросаться «китайскими парашютистами» в логе. Ну а про **PRECHARGE** вообще можно забыть – конденсатор у нас снаружи. Но все-же покажу, как сделать со свойством **SCHMITT** на примере **Graphic_with_Child_SCHMITT.DSN**, лежащем в папке **Graphic_and_Hierarchy**. Для элемента **U2** в свойствах вписываем **SCHMITT=< SCHMITT >**, а скрипт для генератора дописываем следующим образом:

```
*MAP ON CLOCK
DEFAULT : CLKOSC=DIGITAL, SCHMITT=NULL
EXTERNAL : CLKOSC=NULL, SCHMITT=D
```

Вот и вся модификация. Если кому то нравится этот вариант, то скомпилируйте **176IE12.MDF** с дочернего листа модуля из проекта **Graphic_with_Child_SCHMITT.DSN** самостоятельно. Я там уже заранее поставил и задержки, которые мы использовали в конечном варианте. Можно даже сразу откомпилировать в папку **MODELS** Протеуса. Она при старте компилятора предлагается автоматически.
Еще, учитывая наличие второй Российской достопримечательности после дорог, можно создать и скомпилировать (сторонней программой, а не Протеусом!!!) файл помощи типа **xxx.HLP**. Файл помещаем в папку **HELP** Протеуса, а на четвертой вкладке **Make Device** (которую пока мы просто проскакиваем) в графе **Help File** указываем путь к файлу помощи. Возможно, я так и поступлю, но сделаю единый **xxx.HELP** для всей 176-й серии позже (да проклянут меня владельцы Windows 7!!!).
Ну и последнее замечание. Если все же до конца покривить душой и оставить задержки как у модели 4060, то модель 176ИЕ12 будет меньше грузить ЦП компьютера, так что это тоже вариант модификации. А мы далее переходим к созданию счетчиков неизменно сопутствующих применению **К176ИЕ12** в часах.

Возврат к содержанию

6.10. Структура модели счетчика 4026 – основы для будущей К176ИЕ4. Полезные сведения о примитивах счетчиков и дешифраторов в ISIS.

В 176-й серии микросхем есть два счетчика-дешифратора неизменно сопутствующих **К176ИЕ12** при создании схем электронных часов с сегментными индикаторами. Это декадный счетчик **К176ИЕ4** с выходами в коде семисегментного индикатора и **К176ИЕ3** – аналогичный предыдущему счетчик с коэффициентом пересчета 6. Почему-то в таблицах аналогов микросхем нашему **К176ИЕ4** принято подставлять в аналог импортный **CD4026**. Но редко кто при этом делает ссылку на то, что аналог неполный. Я бы рискнул подчеркнуть, что аналогия наблюдается только в функции десятичного пересчета и выходов в семисегментном коде и не более того. Но, тем не менее, воспользовавшись моделью для **4026** как основой, можно попробовать создать **176ИЕ4**. Оставим пока в покое наши счетчики, и попробуем посмотреть, как выглядит модель **4026**. Файл **4026.MDF** извлекается аналогично модели **4060** и расположен в той же библиотеке **DIGITAL.LML**. Я прикреплю его к вложению для тех, кому лень лишний раз поработать с **GETMDF**. Там же вы найдете даташит на **CD4026B** от фирмы Texas Instruments. Он самый маленький по размеру из имеющихся у меня в наличии, поэтому не сильно увеличит объем вложения. А вот внутреннюю структуру на Рис. 91 я взял от **HCF4026** – этот даташит с самыми четкими картинками, но и весит в три раза больше.

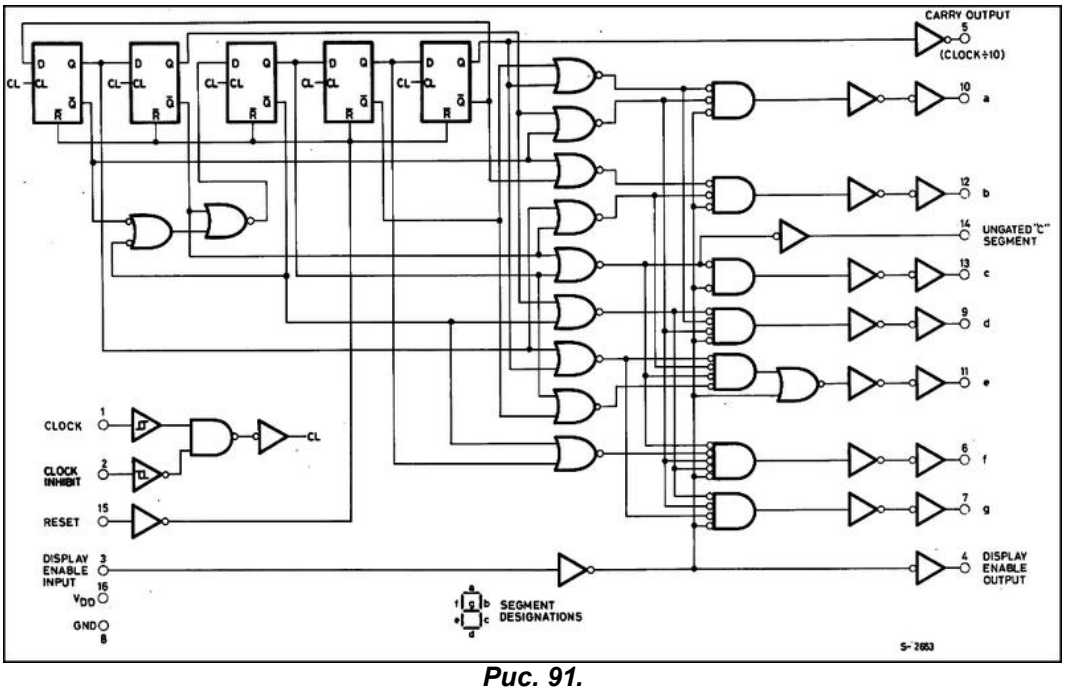


Рис. 91.

Если проанализировать структуру счетчика, то можно сделать вывод, что это пятиразрядный счетчик Джонсона на D-триггерах с дешифрацией кода в семисегментный на логических элементах. В принципе, мы могли бы, как и в случае с **4060** воспроизвести эту структуру на цифровых примитивах и получить вполне нормальную, рабочую модель счетчика. Но давайте заглянем в **4026.MDF**. Ниже приведен раздел, содержащий список компонентов модели:

```
*PARTLIST,4
U1,COUNTER_4,COUNTER_4,ALOAD=0,ARESET=1,INVERT="OE,CE,MAX",LOWER=0,PRIMITIVE=DIGITAL,UPPER=9,USEDIR=0
U2,DECODER_4_7,DECODER_4_7,PRIMITIVE=DIGITAL,TG=<TG>,TYPE=7B
U3,OR_4,OR_4,INVERT=D2,PRIMITIVE=DIGITAL,TG=<TG>
U4,BUFFER,BUFFER,PRIMITIVE=DIGITAL,TG=<TG>
```

Приятная неожиданность, не правда ли? Весь список – 4 примитива. Ни триггеров, ни многовходовых логических элементов. Остается констатировать факт – в данном случае применена поведенческая модель, т.е. функции реального компонента имитированы с помощью подходящих по назначению цифровых примитивов, путем задания им специфических свойств. Как я и обещал, в процессе создания моделей мы познакомимся с цифровыми примитивами поближе. В данном случае присутствуют два заслуживающих особого внимания компонента. Это **COUNTER_4** и **DECODER_4_7**. Логические элементы, надеюсь, не вызывают каких-либо затруднений в использовании. Кстати, изучив специфические свойства четырехразрядного счетчика и декодера из 4 разрядов в семисегментный код, мы сможем без труда использовать и остальные примитивы счетчиков и дешифраторов. Итак, приступим.

Четырехразрядный универсальный счетчик **COUNTER_4** с обвеской для тестирования представлен на Рис. 92, а тестовый проект в папке **COUNTER_TEST** вложения. Для визуальной индикации состояния его выходов я использовал модель семисегментного индикатора **7SEG-BCD**,

позволяющую в цифровом виде просматривать двоичный код. Обратите внимание, что у этой модели вывод младшего разряда находится справа. Имена выводов не индицируются, поэтому просто запомните на будущее.

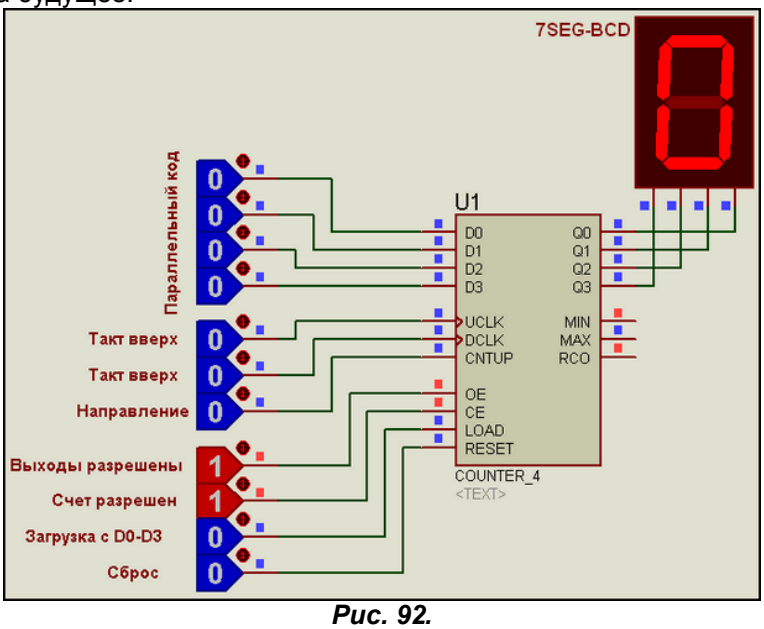


Рис. 92.

Для начала познакомимся с назначением выводов (пинов) модели.
Входы:
D0…D4 – (**Load data**) входы данных параллельной загрузки счетчика.
UCLK – (**Count-up clock**) счетный вход инкремента (приращения) по положительному фронту импульса.
DCLK – (**Count-down clock**) счетный вход декремента (уменьшения) по положительному фронту импульса.
CNTUP – (**Count up/down direction control**) вход направления счета. По умолчанию отключен. Подключается флажком в свойствах: лог.0 – счет вниз, лог.1 – счет вверх.
OE – (**Output-enable control**) разрешение выходов. Лог.1 разрешает сигнал на выходах Q0…Q4.
CE – (**Count-enable control**) вход разрешения счета. Лог.1 разрешает счет.
LOAD – (**Load input**) вход разрешения параллельной загрузки. По умолчанию синхронный, т.е. при установке в лог. 1 загрузка с входов D0…D4 осуществляется по переднему фронту UCLK или DCLK. При установке в асинхронный режим флажком в свойствах загрузка осуществляется по переднему фронту импульса на нем независимо от счетных входов.
RESET – (**Reset input**) сброс счетчика по умолчанию синхронный по положительному фронту UCLK или DCLK. Перевод в асинхронный режим флажком в свойствах.
Выходы:
Q0…Q4 – (**Count output**) выходы счетчика.
MIN – (**Minimum count output**) выход минимального значения. Сигнал на нем формируется при направлении счета вниз и установленном значении **Minimum count value** в **Advanced** свойствах.
MAX – (**Maximum count output**) выход максимального значения. Сигнал на нем формируется при направлении счета вниз и установленном значении **Maximum count value** в **Advanced** свойствах.
RCO – (**Ripple-carry output**) выход достижения верхнего или нижнего порога счета. Сигнал на нем формируется как по верхнему **Minimum count value**, так и по нижнему **Maximum count value** пределам счета.
Окно свойств счетчика представлено на Рис. 93.

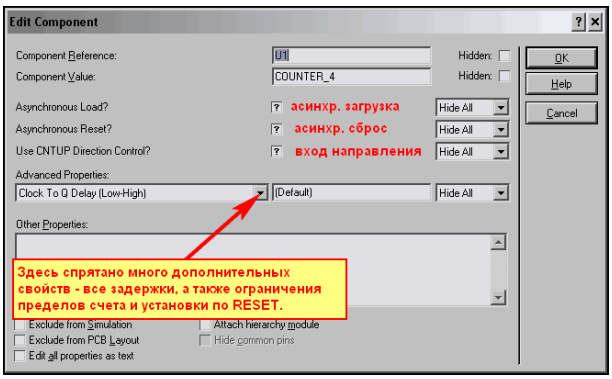


Рис. 93.

Из основного окна свойств непосредственно доступны только три флажка: **Asynchronous Load** (**ALOAD**), **Asynchronous Reset** (**ARESET**) и **Use CNTUP Direction Control** (**USEDIR**). О них я упоминал в описании выводов модели. По умолчанию они не определены – **None** (символ знака вопроса). Для установки необходимо добиться щелчком мыши галочки в окошке, для снятия – пустого окна. Все остальные свойства спрятаны в раскрывающемся списке **Advanced Properties**. В основном это временные свойства задержек. Я не думаю, что стоит приводить здесь весь список времянок с переводом Протеусной «фени» на наш великий и могучий. Как расшифровывать эту временную абракадабру я уже описывал, ничего принципиально нового здесь нет. Остановлюсь только на трех параметрах, которые нам будут интересны в данный момент. Все они сосредоточены в конце раскрывающегося списка и по умолчанию не определены, т.е. **None**.

Lower Count Value (**LOWER**) – нижнее значение предела счетчика.

Upper Count Value (**UPPER**) – верхнее значение предела счетчика.

Output Value On Reset (**RESET**) – состояние выходов счетчика после воздействия сигнала сброса по входу RESET.

Все эти три значения задаются десятичными числами. Следует помнить, что исходное состояние выходов 0000 – тоже значимое состояние. Поэтому, при задании пределов для счетчиков-делителей его необходимо учитывать и задавать предел на единицу меньше. Например, для декадного счетчика с коэффициентом пересчета 10 необходимо задать верхний предел **UPPER=9**. При этом **LOWER** и **RESET** можно и не задавать, если счетчик только суммирующий. Однако если предполагается реверсивный счет, то лучше его обозначить. Если вы хотите получить реверсивный счетчик с одним тактовым входом, то просто объедините **UCLK** и **DCLK**, включите флажок **USEDIR** и управляйте направлением счета с помощью входа **CNTUP**. Интересно назначение последнего из трех вышеуказанных свойств – **RESET**. Если задать ему ненулевое числовое значение ниже верхнего предела, то при воздействии входного сигнала **RESET** счетчик будет сбрасываться не в ноль, а в это значение, при условии, что в данный момент значение на выходах больше данного числа. Для предустановки счетчика при запуске симуляции как и у триггеров, используется свойство **INIT**, введенное вручную в окне дополнительных свойств. Если вы введете **INIT= 3**, то при старте симуляции встанут в лог. 1 выходы **Q0** и **Q1** (двоичное 0011). Также как и с другими цифровыми примитивами здесь применимо свойство **INVERT** как к входам, так и к выходам. Например, по умолчанию счет происходит по переднему фронту тактового импульса на **UCLK**. Если ввести **INVERT=UCLCK**, то счетчик будет перекидываться по заднему фронту этого импульса. В INVERT можно записывать одновременно несколько выводов модели, разделяя их имена запятыми без дополнительных пробелов. Например: **INVERT=D0,D1,D2,D3,RESET** проинвертирует сигналы на всех входах данных и входе сброса, т.е. счетчик сбрасываться будет лог. 0.

Как вы уже, наверное, догадались из рисунка 92, чтобы активировать модель счетчика – получить сигналы на выходах и разрешить счет необходимо подать на **OE** и **CE** сигналы лог. 1. Чтобы вы могли самостоятельно изучить возможности универсальной модели счетчика, я приложил два проекта **Manual_TEST.DSN** и **Misc_TEST.DSN** в папке **COUNTER_TEST**. В первом из них тестирование с различными предустановками осуществляется путем ручного изменения состояния логических переключателей на входах счетчиков. Во втором на входы поданы сигналы генераторов, тестирование по частотомерам в интерактивном режиме и с помощью графиков.

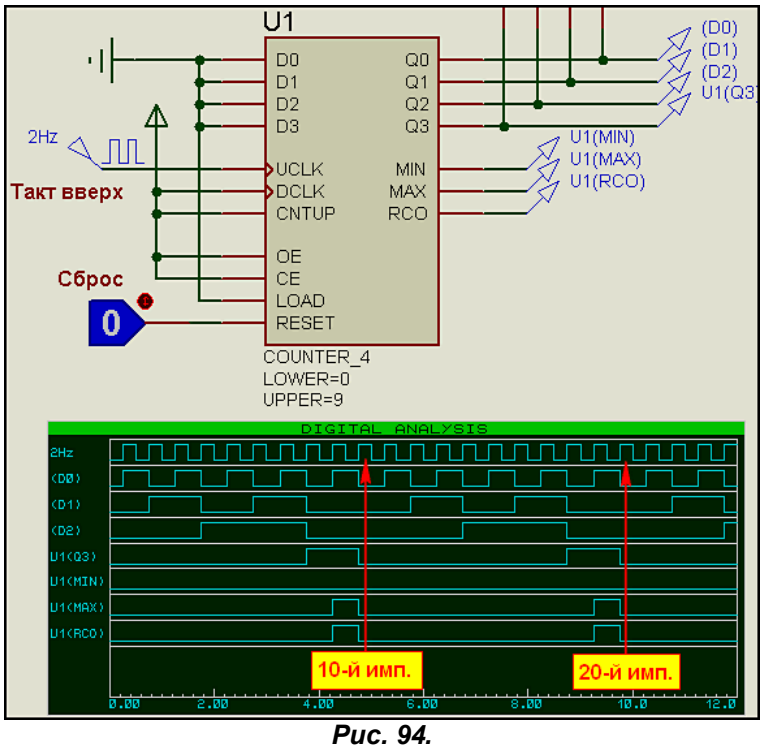


Рис. 94.

Мы тестировали четырехразрядную модель универсального счетчика, но в арсенале цифровых примитивов имеются различные варианты от трех до 14 разрядов. Свойства у всех у них аналогичны рассмотренному нами.
Следующий не менее интересный примитив дешифратор **DECODER_4_7**. Свойств у него будет поменьше, но, тем не менее, знание особенностей этой модели позволит нам успешно применять ее и другие дешифраторы для своих целей.
Вид декодера включенного в двоичный режим с обвеской для ручного теста показан на Рис. 95.

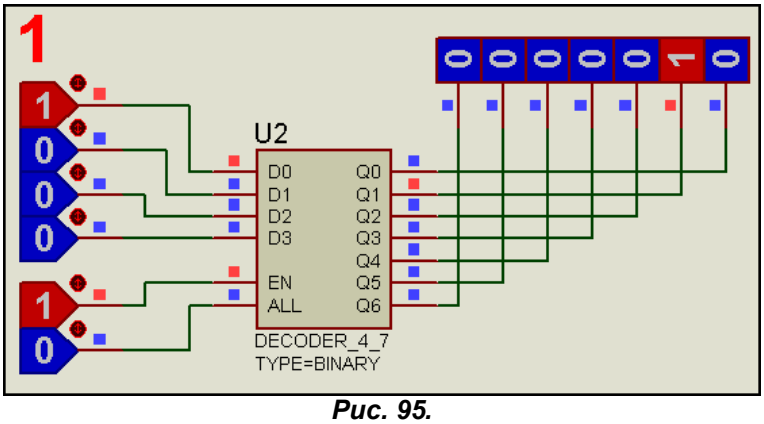


Рис. 95.

Назначение выводов модели.
Входы:
D0…D4 – (**Data input to be decoded**) входы данных для декодирования.
EN – (**Enable**) сигнал на выходах разрешен.
ALL – (**All outputs active**) се выходы в активное состояние в соответствии с выбранным в окне свойств уровнем LOW или HIGH для **All sets all outputs** (ALL). По умолчанию не выбрано – **None**.
Выходы:
Q0…Q6 – (**Decoded output value**) декодированное выходное значение.
Теперь рассмотрим окно свойств (Рис. 96).

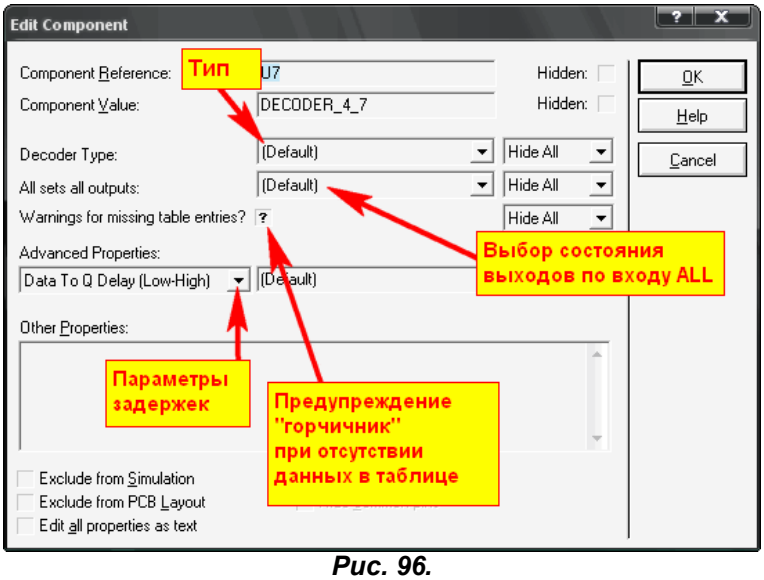


Рис. 96.

Decoder Type (**TYPE**) – тип декодированного сигнала. На этом свойстве остановимся особо чуть ниже.
All sets all outputs (**ALL**) – установка всех выходов по сигналу на входе **ALL**. Описано выше.
Warnings for missing table entries (**WARN**) – желтое предупреждение в логе в случае отсутствия данных в таблице для присутствующей на входах кодовой комбинации.
В раскрывающемся списке **Advanced Properties** прописаны все типы задержек, присущие данной модели. На них я также как и со счетчиком останавливаться не буду.
При наличии на входе **EN** высокого логического уровня данные на выходах модели устанавливаются в соответствии с двоичной кодовой комбинацией на входах и типом дешифратора, выбранным в окне свойств. Вот на этом моменте остановимся подробнее.
Возможны пять типов дешифрации входного сигнала, выбираемых из раскрывающегося списка:
BINARY – в этом режиме двоичному коду на входах **D0…D4** модели соответствует единственный активный выход в соответствии с заданным входным значением, т.е. **0000** активирует выход **D0**, **0001** – выход **D1** (на Рис. 95) … **1001** – выход **D9** … 1111 – выход **D15**. Конечно, последние два из перечисленных вариантов в нашем случае отсутствуют, но имеются, например, в модели **DECODER_4_16**.

BCD – двоично-десятичный декодер. Этот вариант до цифры 9 – код **1001** полностью совпадает с двоичным, однако последующие комбинации четырехразрядного кода не рассматриваются, и состояния выходов не меняют. Это вытекает из концепции BCD кодирования, где каждый разряд десятичного числа описывается двоичной тетрадой, т.е. числу **123** будет соответствовать в коде BCD кодовая комбинация **0001 0010 0011** (сотни-десятки-единицы).
Две кодовых комбинации для семисегментного кода – **7A** и **7B**. Варианты отображения сегментов для кода на входах представлены на Рис. 97 и всегда доступны в файле помощи по кнопке **HELP** в окне свойств. Если кто-то затрудняется найти отличие – сравните цифры 6 и 9.

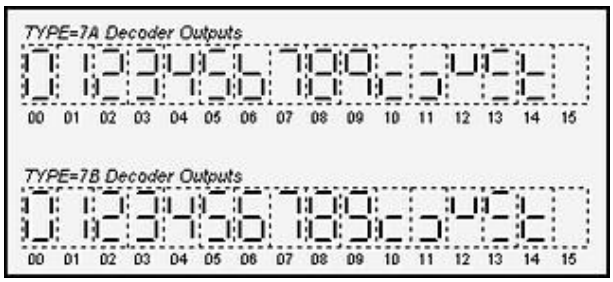


Рис. 97.

Ну и, наконец, пятый, самый интересный тип **TABLE** – дающий нам возможность проявить собственные фантазии. Этот тип туманно описан в **HELP**, поэтому я восполню пробелы в информации. Как и следует из перевода, это табличный тип представления соответствия числа на входе состоянию выходов. С ним неразрывно связано свойство **LENGTH** (длина таблицы), которое вручную надо задать в окне свойств, выбрав этого тип дешифратора. В качестве примера использования этого типа в папке **DECODER** вложения приложен проект TABLE.DSN, в котором с помощью таблицы я задал декодирование семисегментного кода, где комбинациям с 10 по 15 соответствуют выходные сигналы мнемоники букв **A, b, C, d, E, F** для индикатора с общим катодом. Ниже приведен код, заданный в свойствах:

```
LENGTH=16
TABLE0=%0111111
TABLE1=%0000110
TABLE2=%1011011
TABLE3=%1001111
TABLE4=%1100110
TABLE5=%1101101
TABLE6=%1111101
TABLE7=%0000111
TABLE8=%1111111
TABLE9=%1101111
TABLE10=%1110111
TABLE11=%1111100
TABLE12=%0111001
TABLE13=%1011110
TABLE14=%1111001
TABLE15=%1110001
```

Рассмотрим, как он сформирован. **LENGTH=16** задает количество строк таблицы, включая **TABLE0**. Далее, в каждой строке, начинающейся со слова **TABLE**, назначен двоичный код на выходах **Q** соответствующий десятичному значению после слова **TABLE** на входах **D**. Знак процента после равенства означает, что далее следует двоичное число. В двоичном значении, как и обычно, младший разряд справа, т.е. в нашем случае для шести выходов они расположены как **Q6,Q5…Q0**. Используя другие типы примитивов дешифраторов с меньшим или большим числом входов/выходов, можно таблично задать любую задуманную заранее комбинацию. Обращаю только внимание на то, что заданная длина таблицы и количество знаков в двоичном числе должны совпадать соответственно с числом строк и числом выходов модели, иначе получите «горчичник» в логе. Различные примеры для тестирования модели дешифратора приложены в проектах **BIN_BCD.DSN** и **7A_7B.DSN** папки **DECODER** вложения. Из названий видно, какие типы, где тестируются. Для вариантов **7A** и **7B** я показал, как с помощью свойства **INVERT** для выходов можно использовать индикаторы с общим катодом и общим анодом.
На этом пока закончим знакомство с примитивами и вернемся к нашей модели **4026**. В папке **CD4026_model** я привел проект, в котором по файлу **MDF** реконструирована внутренняя структура модели счетчика. На рисунке 98 она приведена полностью. Наверное, теперь, когда ясны те или иные назначения свойств использованных примитивов, излишние комментарии по поводу работы этой структуры становятся неуместными. Но все-же немного поясню, чтобы снять ненужные вопросы. Посредством назначения верхнего и нижнего пределов и запрещения параллельной загрузки примитив **U1** превращен в обычный десятичный счетчик. Посредством **INVERT** изменен активный уровень по входу **CE**, чтобы совпадал с реальным. Все ненужное завешено на **VSS**, т.е. на нулевой вывод питания, причем вход **OE** для этого проинвертирован, чтобы активным разрешением по нему стал лог. 0. Проинвертирован также и выход **MAX**, чтобы создать перепад с нуля на

единицу на выходе **CO** по достижении десятого импульса. Кстати, здесь есть небольшое расхождение с реальной микросхемой по временному формированию этого сигнала не влияющее на работу при последовательном соединении счетчиков. На элементе **U2** реализовано декодирование в сигнал семисегментного кода по таблице 7В (шестерка и девятка с «хвостиками»). На четырехвходовом ИЛИ - элементе **U3** реализовано формирование импульса по счету 2. Здесь также применена инверсия к третьему сверху входу. Ну и цифровой буфер **U4** применен чисто для «гальванического» разделения входа **DEI** и выхода **DEO**.

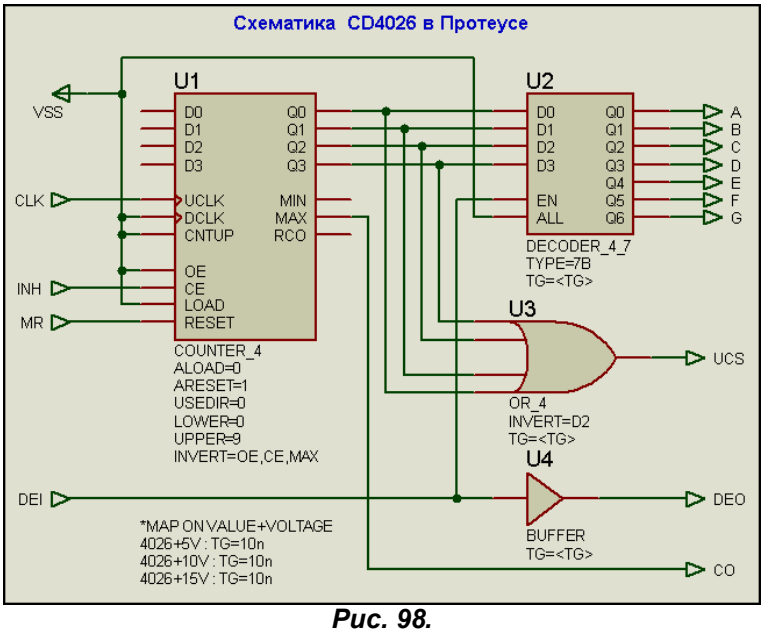


Рис. 98.

Вот эту структуру мы и возьмем за основу для формирования моделей наших **К176ИЕ3** и **К176ИЕ4**, прибегнув к некоторым корректировкам в сторону их особенностей. Об этом в следующем материале.

Возврат к содержанию

6.11. Поведенческие модели К176ИЕ4 и К176ИЕ3 для Протеуса на основе примитивов универсальных счетчиков.

Для начала сформируем графические модели этих счетчиков в соответствии с Рис. 99.

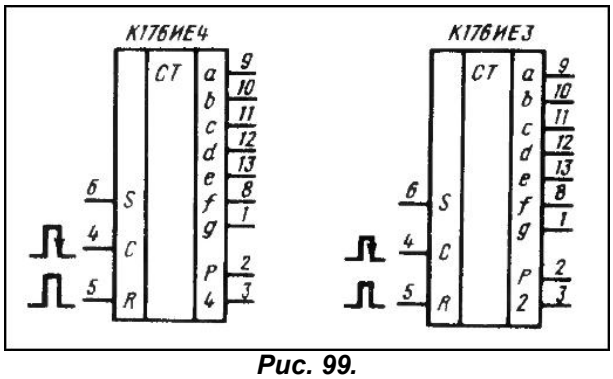


Рис. 99.

Надеюсь, что мне не стоит в очередной раз повторять процесс создания графики, я просто приложил проект **Graphic_model.DSN**, в котором есть и несформированная в модели графика и готовые модели. Обратите внимание, что моделям назначен корпус **DIL14** в отличие от шестнадцативыводной **К176ИЕ12**. Базовая графика показана на рисунке 100.

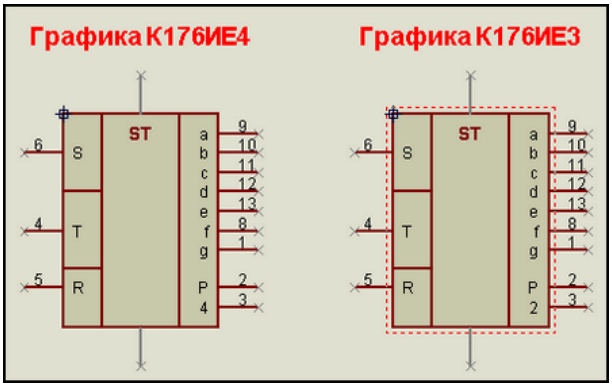


Рис. 100.

Обратите внимание, что вместо стандартного обозначения счетного вывода я применил к нему символ “**T**”. Протеус не приучен различать прописные и строчные буквы, а латинским символом “**c**” у нас уже обозначен третий выход на семисегментный индикатор. Поэтому пришлось пойти на такую замену. Порывшись в старых справочниках, в книге В. Л. Шило «Популярные цифровые микросхемы» М., Радио и связь, 1989 мне удалось найти временные диаграммы формирования сигналов на выходах **P** и **4**(**2**) этих микросхем. Диаграмма приведена на Рис. 101. Соответственно на ней **U2** – выходной сигнал на выводе 2 (**P**), а **U3** – выходной сигнал на выводах 3 счетчиков.

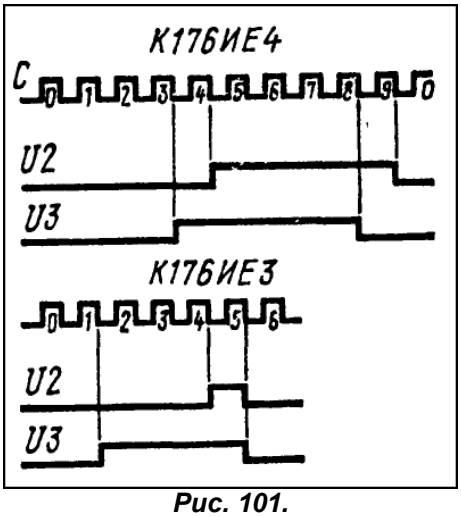


Рис. 101.

Приведу еще несколько выдержек из различных справочных материалов, посвященных этим счетчикам, которые пригодятся нам в дальнейшем. Установка триггеров счетчиков в 0 происходит подачей лог. 1 на вход **R**, переключение (счет) по спаду импульса положительной полярности на входе **C** (в моем случае **T**). Сигнал на входе **S** служит для изменения полярности выходных семисегментных кодовых комбинаций. В случае подачи на вход **S** положительного сигнала лог. 1 получаем на выходе код для индикаторов с общим анодом, а в случае лог. 0 – код для индикаторов с общим катодом. При использовании вакуумно-люминисцентных индикаторов на этот вход подается модулирующие импульсы, например 32 кГц счетчиков **К176ИЕ5** или **К176ИЕ12**. При использовании жидкокристалических индикаторов также подаются модулирующие импульсы, но с частотой на порядок ниже 30-100Гц, которые одновременно подаются и на подложку индикатора. Импульсы на выходах **P** и **4** (**2**) данных счетчиков позволяют при последовательном соединении организовать разряды индикации минут от 0 до 59 и часов от 0 до 23. Спады положительных импульсов на соответствующих выходах служат в качестве тактовых для переключения старших разрядов. Мы рассмотрим в дальнейшем организацию структуры электронных часов с использованием данных счетчиков. А пока, отталкиваясь от этих данных и приняв структуру **4026** как базовую, попробуем создать на ее основе модели данных счетчиков.

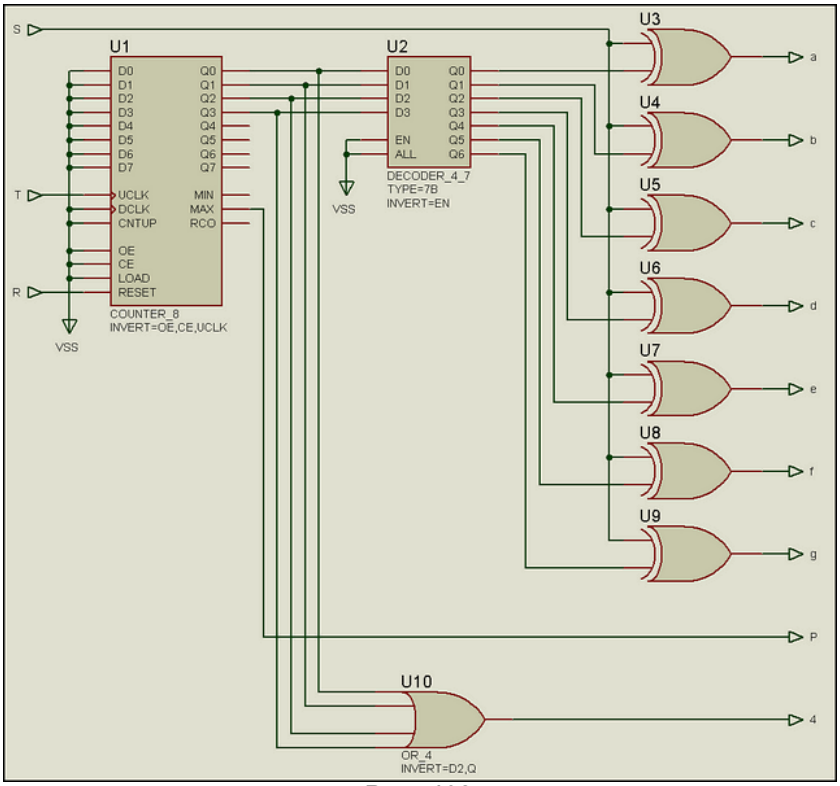


Рис. 102.

Основным отличием **К176ИЕ4** от **CD4026** является наличие входа **S**, позволяющего инвертировать выходной код. Достаточно добавить к исходной структуре дополнительную логику, реализующую данную функцию и модель приобретет все необходимые нам свойства. Проще всего реализовать этот момент с помощью двухвходовых элементов **XOR** (исключающее ИЛИ). Кроме того, нам потребуется изменить полярность сигналов на выходах **P** и **4**. Это легко делается с помощью свойства **INVERT**. Получившаяся в результате данных действий структура показана на Рис. 102.

В отличие от модели **4026** в данном случае не инвертируется сигнал **MAX** со счетчика **U1**, но зато инвертируется выход **D** элемента 4ИЛИ **U10**. На выходе дешифратора включены элементы исключающее ИЛИ, которые с помощью изменения полярности сигнала на входе **S** позволяют инвертировать выходной код. На Рис. 103 приведена временная диаграмма для данного варианта.

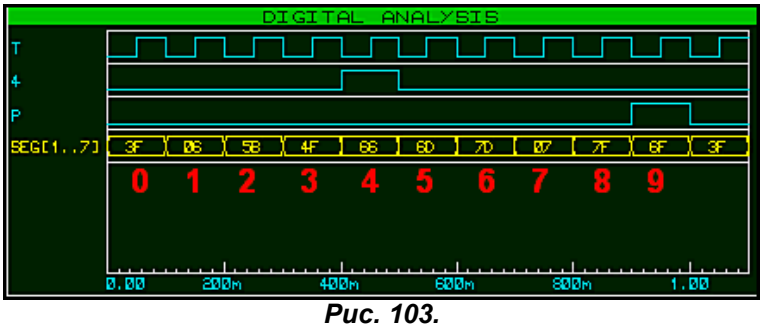


Рис. 103.

Для лучшего восприятия диаграммы я включил в нее в виде семиразрядной шины сигнал на выходах **a…g**. В данном случае на входе **S** присутствует лог. 0 и код шины соответствует сигналам для индикатора с общим катодом. Все отличие от **4026** в инверсии сигналов на выходах **P** и **4**. Данный вариант модели имеет право на существование, и даже будет корректно работать, поскольку нужные перепады уровней с лог. 0 на лог. 1 после цифры 3 и с лог. 1 на лог. 0 после цифры 9 совпадают во времени с приведенными на диаграмме Рис. 101 сигналами для счетчика **К176ИЕ4**. Но вот длительности этих импульсов оставляют желать…. Попробуем с помощью дополнительных элементов устранить данное несоответствие. Я не стал сильно мудрить и посредством двух дополнительных элементов 2И и триггеров достиг желаемого (Рис. 104).

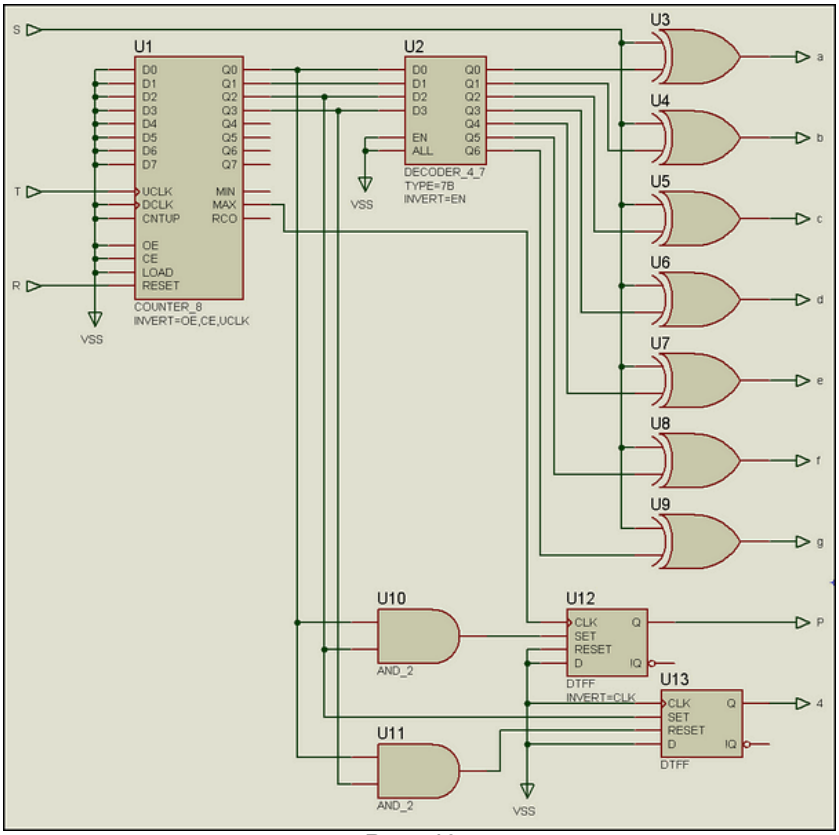


Рис. 104.

В результате проведенных коррекций картинка на диаграмме (Рис. 105) стала полностью совпадать с приведенной ранее на рисунке 101. Данные примеры приложены в папке вложения **176IE4_structure**. Проект **IE4_var_1.DSN** – с укороченными импульсами и проект **IE4_var_2.DSN** с дополнительными триггерами и импульсами соответствующими диаграмме рисунка 105. Аналогичные примеры для счетчика **К176ИЕ3** приведены в папке **176IE3_structure**. Они отличаются только логикой формирования импульсов на выходах **P** и **2**, поскольку счетчик имеет коэффициент пересчета 6. Я не стану здесь приводить лишние картинки, желающие посмотрят их в соответствующих проектах.

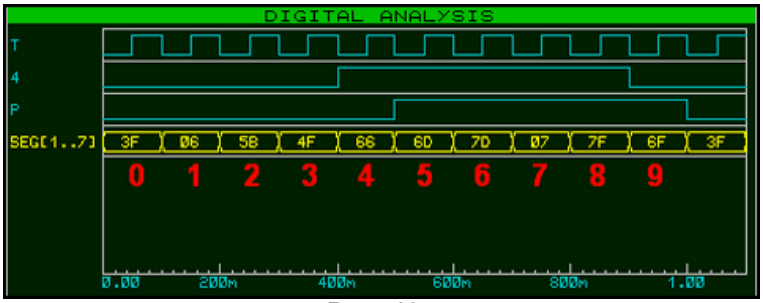


Рис. 105.

В проектах **Test_176IE3_IE4_V1.DSN** и **Test_176IE3_IE4_V2.DSN** данные структуры присоединены в виде дочерних листов к соответствующим графическим моделям, и из них построена схема счетных секций минут и часов для проверки моделей в действии. Папка с проектами называется **Test_counters**.

Но готовых файлов MDF во вложении вы не найдете. Дело в том, что в процессе построения моделей этих счетчиков меня слегка «торкнуло» и я нашел более красивое и близкое к реальности решение для этих моделей. Заодно «пристрелю еще одного ушастого» - познакомлю вас с цифровым примитивом универсального регистра сдвига. Вот из тех моделей мы и сформируем MDF для последующего использования.

Возврат к содержанию

6.12. SHIFTREG - примитив универсального регистра сдвига и модели К176ИЕ4 и К176ИЕ3 для Протеуса на его основе.

Если вспомнить описание внутренней структуры счетчика **4026**, то изначально я упоминал, что он построен на базе пятиразрядного счетчика Джонсона. Не являются исключением и наши **К176ИЕ3**, **К176ИЕ4**. Типично счетчик Джонсона выглядит как замкнутый в кольцо сдвиговый регистр, у которого инверсный выход последнего триггера соединен с входом **D** первого. Давайте попробуем построить максимально приближенную к реальности модель на базе универсального сдвигового регистра. Для начала познакомимся с примитивом **SHIFTREG** и его свойствами. Поскольку нам потребуется пятиразрядный регистр сдвига, я буду рассматривать именно **SHIFTREG_5**, у остальных свойства те же. Для ручного тестирования я обвешал примитив переключателями и пробниками из **Debugging Tools** (Рис. 106) и поместил в проект **ShiftReg5_Manual_test.DSN** в папке **SHIFTREG_TEST** вложения.

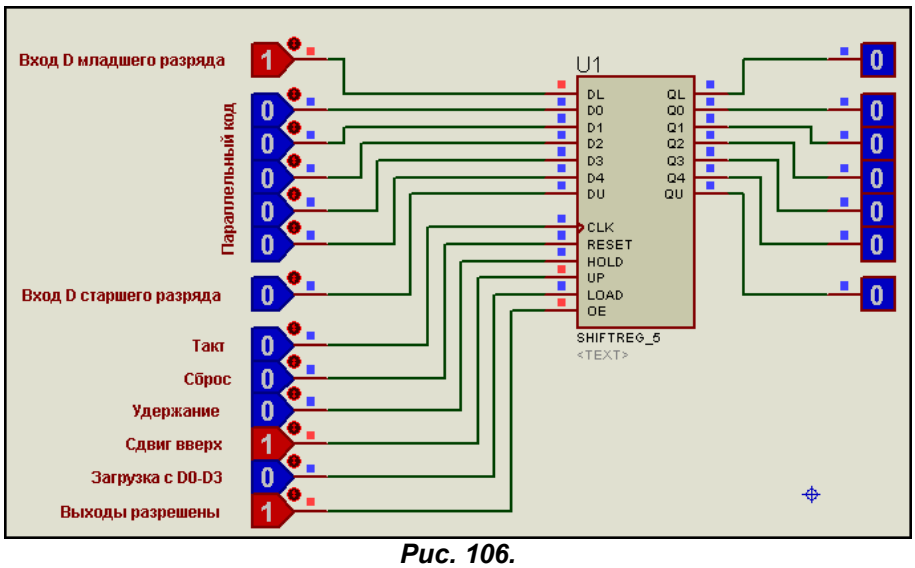


Рис. 106.

Для начала познакомимся с назначением входов и выходов примитива.
Входы:
DL – **(New lower data)** вход данных младшего разряда регистра, по сути вход D триггера младшего разряда.
D0…D4 – (**Parallel load data**) входы данных для параллельной загрузки.
DU – **(New upper data)** вход данных старшего разряда регистра, по сути вход D триггера старшего разряда.
CLK – (**Clock**) тактовый вход. По умолчанию сдвиг (а также загрузка и сброс, если не установлены флажки в свойствах) происходит по переднему фронту импульса на нем.
RESET – (**Data reset**) сброс данных. По умолчанию при лог. 1 на этом входе по переднему фронту CLK. Для асинхронного сброса только этим сигналом необходимо установить флажок в свойствах (Рис. 107).
HOLD – (**Shift-hold**) удержание данных. Лог. 1 на этом входе запрещает сдвиг данных по тактовому сигналу на входе CLK.
UP – (**Direction control**) направление сдвига. При лог. 1 на этом входе сдвиг происходит с входа **DL** по направлению вверх, т.е. **Q0=>Q1…Q4**. При лог. 0 на этом входе сдвиг происходит с входа **DU** по направлению вниз, т.е. **Q4=>Q3…Q0**.
LOAD – (**Data load**) сигнал параллельной загрузки со входов **D0…D4**. По умолчанию при лог. 1 на этом входе по переднему фронту CLK. Для асинхронной загрузки только этим сигналом необходимо установить флажок в свойствах (Рис. 107).
OE – (**Output-enable**) сигнал, разрешающий работу выходов данных **Q0…Q4**. Обратите внимание, что сигналы на выходах переноса **QL** и **QU** он не блокирует.
Выходы:
QL и **QU** – (**Lower Q and Upper Q**) выходы соответственно переноса вниз и вверх. Фактически сигнал на этих выходах дублирует соответственно Q0 и Q4, но как бы гальванически развязан от них.
Q0…Q4 – (**Data output**) выходы данных соответствующих разрядов регистра.
Теперь обратимся к окну свойств **SHIFTREG** (Рис. 107). По сути, я уже почти все описал, пока рассматривал назначение выводов. Осталось указать, что **Initial Output Value** (**INIT**) определяет состояние триггеров регистра при старте симуляции. По умолчанию оно не определено, но если ввести в него десятичное значение, например 3, то при старте симуляции соответственно выходы (триггеры) **Q0** и **Q1** встанут в состояние лог.1. Фактически, у счетчиков было то же самое. Ну и

конечно по раскрывающемуся списку **Advanced Properties** запрятаны все возможные задержки распространения сигнала для данного примитива.

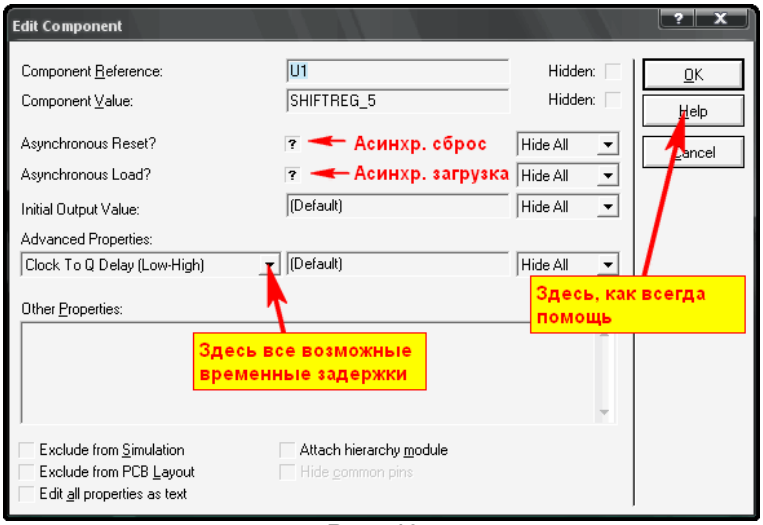


Рис. 107.

В папке **SHIFTREG_TEST** имеется проект **ShiftReg5_auto_test.DSN**, в котором приведены характерные комбинации сигналов и свойств для применения примитива в качестве сдвигового регистра. Последний пример в этом проекте показывает организацию счетчика Джонсона, который нам необходим для создания моделей наших счетчиков. На рисунке 108 приведена диаграмма его работы. Мы видим, что полный цикл от состояния от **00000** до **11111** счетчик проходит за 10 тактовых импульсов.

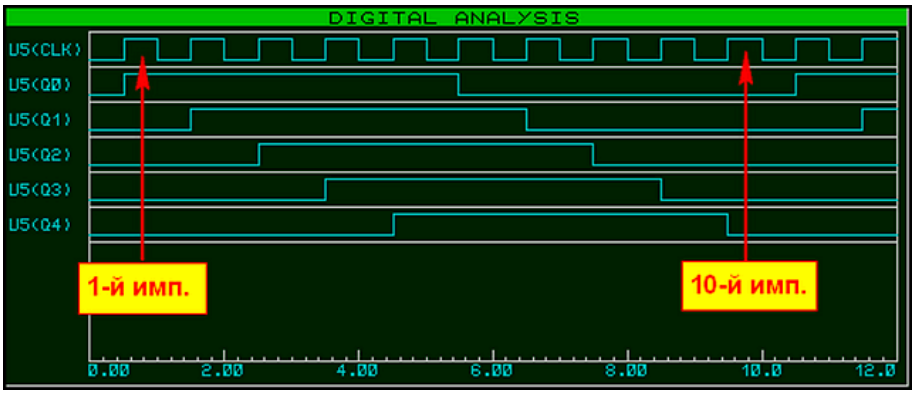


Рис. 108.

Фактически, это то, что нам нужно для построения моделей. Достаточно проинвертировать тактовый вход, чтобы срабатывание происходило по спаду тактового импульса и половина модели готова. Причем импульсы на выходах переполнения **P** (выход **Q4**) и **4** (выход **Q3**) уже будут иметь необходимую длительность. Если кому интересен первоисточник информации, рекомендую обратиться к книге: Ковалев В. Г. Лебедев О. Н. «Электронные часы на микросхемах» М.: Радио и связь, 1985. Там построение счетчиков ИЕ3, ИЕ4 рассмотрено в главе 3 «Микросхемы серии 176». Ну, что, полдела сделано, но очень не хочется крутить «хитрый» дешифратор на мелкой логике для преобразования кода Джонсона в семисегментный. И тут на помощь нам снова приходит примитив дешифратора, только на этот раз **DECODER_5_8**. Поступим просто: пять входов этого дешифратора заводим на выходы счетчика, а первые семь выходов описываем таблицей для семисегментного индикатора. Правда таблица получится достаточно длинной, 2 в степени 5, т.е. 32 строки, потому что неиспользуемые комбинации также пришлось включить с нулевыми значениями выходов. Если этого не сделать, **ISIS** будет плеваться горчичниками на отсутствующие комбинации.

Результирующий код получился следующим:

```
{LENGTH=32
TABLE0=%00111111
TABLE1=%00000110
TABLE2=0
TABLE3=%01011011
TABLE4=0
TABLE5=0
TABLE6=0
TABLE7=%01001111
TABLE8=0
TABLE9=0
TABLE10=0
TABLE11=0
TABLE12=0
TABLE13=0
TABLE14=0
TABLE15=%01100110
TABLE16=%01101111
TABLE17=0
TABLE18=0
TABLE19=0
TABLE20=0
TABLE21=0
TABLE22=0
TABLE23=0
TABLE24=%01111111
TABLE25=0
TABLE26=0
TABLE27=0
TABLE28=%00000111
TABLE29=0
TABLE30=%01111101
TABLE31=%01101101}
```

Еще раз напомню, что фигурные скобки в начале и конце кода включают **Hide** - скрытие кода, чтобы он не загромождал проект. Я привел только значимые коды (всего их 10) таблицы в двоичном виде, чтобы было нагляднее видно расположение разрядов в коде, а остальное в десятичном. Симулятор Протеуса обучен воспринимать и другие варианты. Значимые строки тоже можно было бы записать в другом виде, например в шестнадцатеричном. Тогда строка для нуля имела бы вид - **TABLE0=$3F**, а для девятки **TABLE16=$6F**. Знак доллара перед числом означает шестнадцатеричное значение. Но до знака равенства, т.е. после слова **TABLE** можно использовать только десятичное значение. Еще обратите внимание, что семиразрядные коды следуют в таблице не по порядку, поскольку код Джонсона до половины максимального значения (для пятиразрядного счетчика это 10) нарастает, а потом спадает в зависимости от количества поступивших на вход счетчика импульсов. Поэтому числу 5 соответствует **TABLE31**, а числу 9 – **TABLE16**. Кто «не въезжает» – в «Основы цифровой техники», читать про счетчик Джонсона. По этому поводу предупреждал.

Ну и еще один полезный совет тем, кто будет аналогичные операции проводить самостоятельно. Поскольку можно использовать шестнадцатеричные значения, то чтобы «не парить себе мозги», можно воспользоваться утилитами генераторов кода для семисегментных индикаторов, например, встроенными в компиляторы фирмы **Mikroelektronika**.

Оставляем выходы модели счетчика в том виде, в котором они и были, т.е. с использованием исключающего **ИЛИ** (проще тут ничего не придумаешь) и формируем наши модели. Для счетчика **К176ИЕ4** мы практически уже все сформировали, и мне остается только привести получившуюся внутреннюю структуру на Рис. 109.

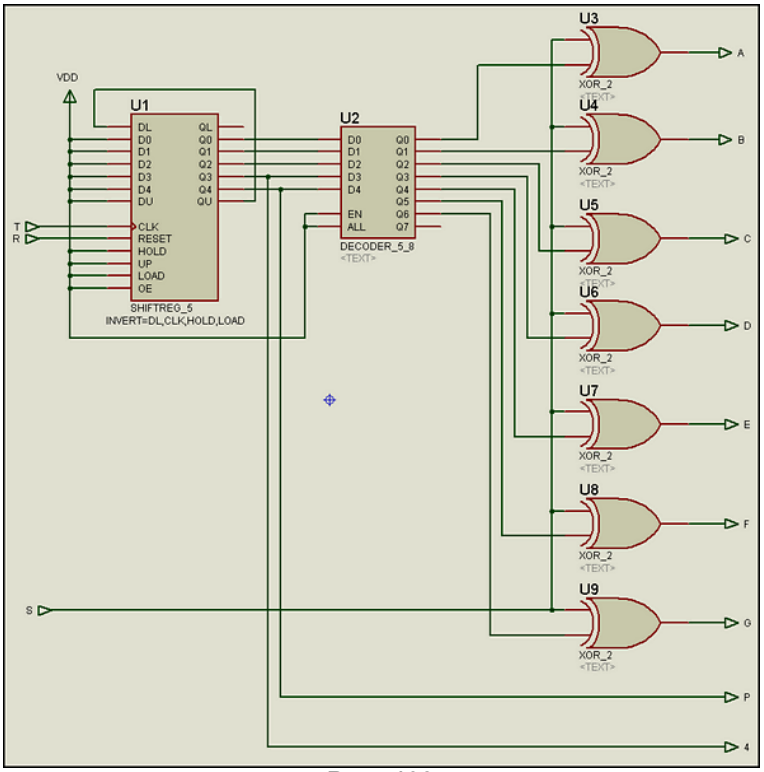


Рис. 109.

Как видно из рисунка, схема получилась даже несколько проще, чем с универсальным счетчиком. Аналогично сформируем и счетчик **К176ИЕ3**. Правда здесь придется уже немного усложнить схему. Дело в том, что менять разрядность входного регистра сдвига мы не можем, он должен быть пятиразрядным, а счетчик должен иметь коэффициент пересчета – 6, и при этом еще и сохранить возможность внешнего сброса – вход R. Поэтому пришлось применить еще один логический элемент двухвходового **ИЛИ**, а для формирования импульса сброса по достижении числа 6 можно воспользоваться незадействованным выходом дешифратора. Достаточно было прописать в нужной строке таблицы единичку в старшем разряде **TABLE30=%11111101**, и мы без всяких лишних хлопот получили нужный нам коэффициент пересчета. Структура приведена на Рис. 110.

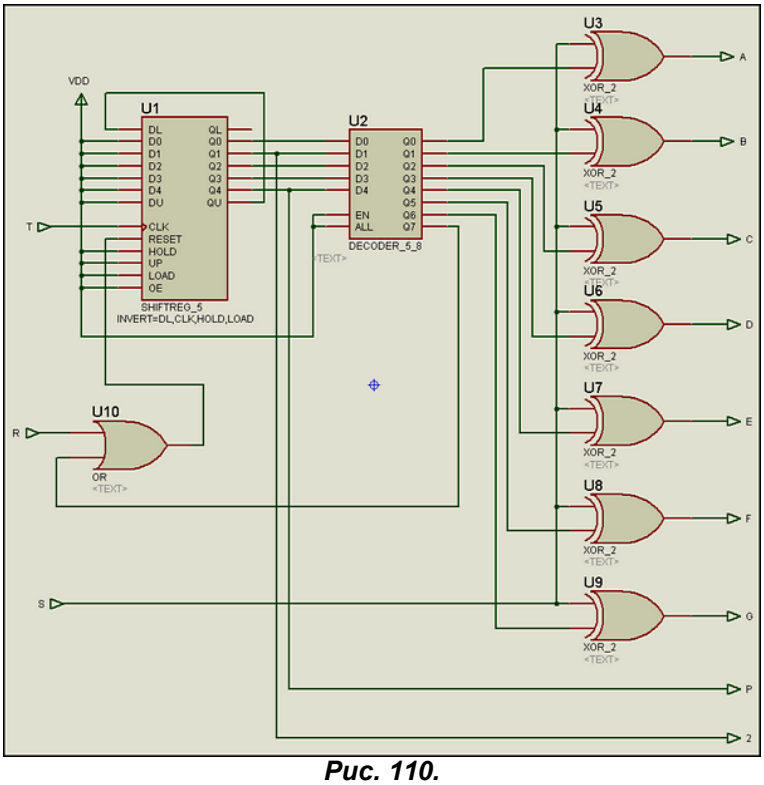


Рис. 110.

Тесты структур для каждого счетчика приведены в папке вложения **COUNTER_STRUCTURE_TEST**, а общий тест счетных декад на их основе в папке **MODELS_WITH_CHILD_LIST**. В ней модели еще имеют дочерний лист и можно туда перейти. Вот с этих листов и скомпилируем **MDF** файлы соответствующих счетчиков. Как это делается, я уже объяснял в примере со счетчиком ИЕ12, но

прежде чем компилировать добавим, также по аналогии с ИЕ12 временные задержки к элементам структуры счетчиков ИЕ3 и ИЕ4. В папке **MODELS_WITH_CHILD_LIST** они уже с добавленными свойствами, поэтому на дочерних листах присутствует скрипт **MAP ON VOLTAGE**. Поскольку у нас модель поведенческая, я добавил временные параметры только к тем элементам, которые непосредственно задействованы в передаче сигналов с входов на выходы микросхемы, т. е. сдвиговому регистру и выходной логике на исключающих **ИЛИ**. Да и у регистра прописаны только те «времянки», которые связаны со счетным входом и входом сброса.
Ну и окончательный тест моделей счетчиков с прописанными в свойствах файлами MDF приложен в папке **MODELS_WITH_MDF_TEST**. Естественно, в этой же папке лежат и сами файлы **176IE3.MDF** и **176IE4.MDF**. Тест показывает, что модели ведут себя вполне адекватно. Итак, у нас уже три собственных работающих схематичных модели 176-й серии. Пришел черед научиться формировать из своих моделей библиотеки.

Возврат к содержанию

6.13. Объединение MDF в библиотеку LML.

Итак, наши цифровые модели серии 176 успешно прирастают в количестве. Желающие могут продолжить создание моделей этой серии и далее, например, по аналогии с **176ИЕ12** создать модель **176ИЕ5**. Можно пополнить модели 176-й серии и совпадающими по назначению аналогами из **CMOS 4000**, сохранив их под новыми именами, соответственно изменив задержки при различных напряжениях питания. Мы уже проделывали такой фокус ранее, получив из **4011** модель **176ЛА7**. Я же на этом закончу рассмотрение процесса моделирования цифровой логики и напоследок покажу – как отдельные схематичные модели с файлами **MDF** объединить в единую библиотеку. Сразу оговорюсь, что совсем не обязательно, чтобы все модели в библиотеке **LML** совпадали по назначению или были только цифровыми, только аналоговыми и т.п. В родных **LML** библиотеках Протеуса полно примеров, когда и «мухи» и «котлеты» благополучно лежат в одной «тарелке». Здесь скорее наоборот, идет принцип разделения по производителю, а не по назначению. Именно поэтому в папке **MODELS** есть библиотеки **FAIRCHLD.LML**, **MAXIM.LML**, **TEXAS.LML** и т.д. Но есть и объединенные «по половому» признаку, например, **MEMORY.LML** – сразу ясно, что внутри модели микросхем памяти. По большому счету, можно и не объединять **MDF**, а просто держать их разрозненно в папке **MODELS**. Но это хорошо, когда у Вас две три своих модели, а когда счет перевалит за пару десятков.… Помимо того, что и самому становится трудно копаться в папке с большим количеством отдельных файлов, Протеусу тоже приходится проделывать эти операции. И что-то мне интуиция подсказывает, что процесс открытия нескольких отдельных файлов и поиск нужных моделей в каждом займет чуть больше времени, чем поиск их в одном открытом файле. Поэтому, для меня лично ответ на вопрос, объединять или нет, однозначен. Итак, приступим.
Для начала соберем все наши модели в единую библиотеку **LIB** в папке **LIBRARY**. Для этого воспользуемся опцией **Library Manager** из верхнего меню **Library**. Напомню, что в одноименных файлах с расширениями **LIB** и **IDX** содержатся сведения о графическом изображении компонентов, в том числе и не имеющих исполняемых моделей (**No Simulation**). Когда мы создаем новый, то по умолчанию **ISIS** предлагает сохранить его в **USRDVC.LIB**. Я эту библиотеку использую как «промежуточную» и периодически провожу там «зачистку» от накопившегося мусора – всяких недоделок и тестовых вариантов. Я решил назвать новую библиотеку **CMOS_RUS** и оставил ей количество позиций 100, предлагаемое Протеусом по умолчанию (Рис. 111).

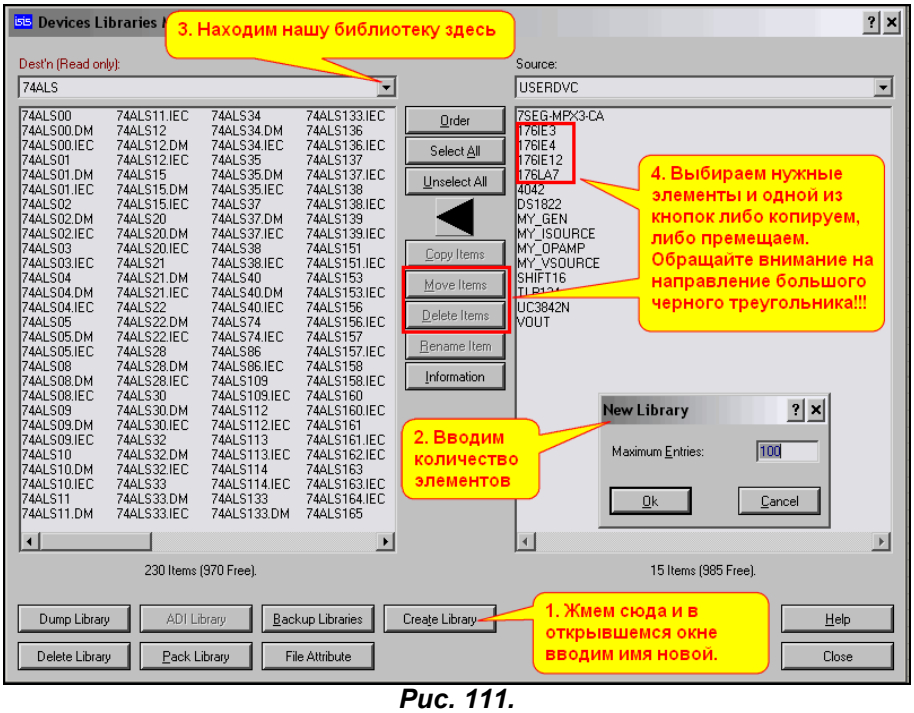


Рис. 111.

После того, как библиотека создана, находим ее через раскрывающийся список (третья операция на рисунке 111). Затем щелкаем по первой из нужных моделей левой кнопкой, зажимаем клавишу Shift и вторым щелчком по последней из нужных – выделяем группу. Я приношу извинения за столь «водянистые» подробности, но последнее время на форуме появились «индивидуумы», которые даже этих типовых операций Винды не знают. Такое «обсасывание костей» персонально для них. Давим кнопку **Move Items** и в появившемся окне подтверждаем еще раз выполнение операции. После этого наше окно примет вид, показанный на рисунке 112. Естественно, в библиотеке **USRDVC** у Вас на компьютере будут находиться совсем другие модели (те, что когда то сохранялись через операцию **Make Device**), а 176-е там будут присутствовать, только если Вы применяли эту функцию к графическим моделям, которые я прилагал в предыдущих примерах.

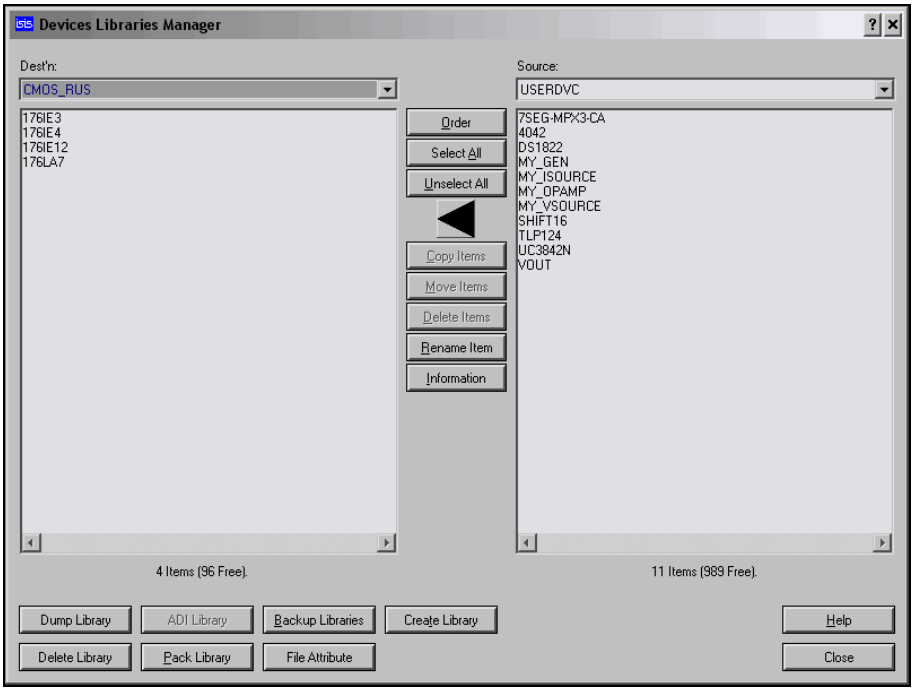


Рис. 112.

После того, как мы создали **CMOS_RUS** и перенесли в нее нужные модели, это окно нам больше не нужно и его можно **Close** соответствующей кнопкой внизу справа. На этом с папкой LIBRARY покончено. Теперь у нас там должны присутствовать **CMOS_RUS.IDX** и **CMOS_RUS.LIB**.
Наступил черед создания **LML**. Для этого в отдельную папку, лучше в корне диска копируем все нужные файлы моделей с расширением **MDF**, которые мы создали ранее и туда же копируем (а не перемещаем!!!) утилиту **PUTMDF.EXE** из папки **BIN** установленного Протеуса. Запускаем утилиту командной строки либо через **Пуск=>Все программы=>Стандартные**, либо **Пуск=>Выполнить**, набрать в появившемся окне **cmd** и нажать **ОК**.
Весь процесс создания **CMOS_RUS.LML** показан на рисунке 113. В данном случае все файлы были помещены во вновь созданную папку **MakeLML** в корне диска **C:**. Сначала я перешел в эту папку, затем вызвал **PUTMDF** без ключей, чтобы увидеть подсказку (помощь), потом создал **CMOS_RUS.LML** на 100 элементов и в последней операции поместил туда четыре ранее созданных **MDF**. Для тех, кто дружит с компьютером, а не «забивает им гвозди» – операции 3 и 4 можно объединить. В этом случае строчка будет выглядеть так:

PUTMDF –L=CMOS_RUS –C=100 176IE3 176IE4 176IE12 176LA7

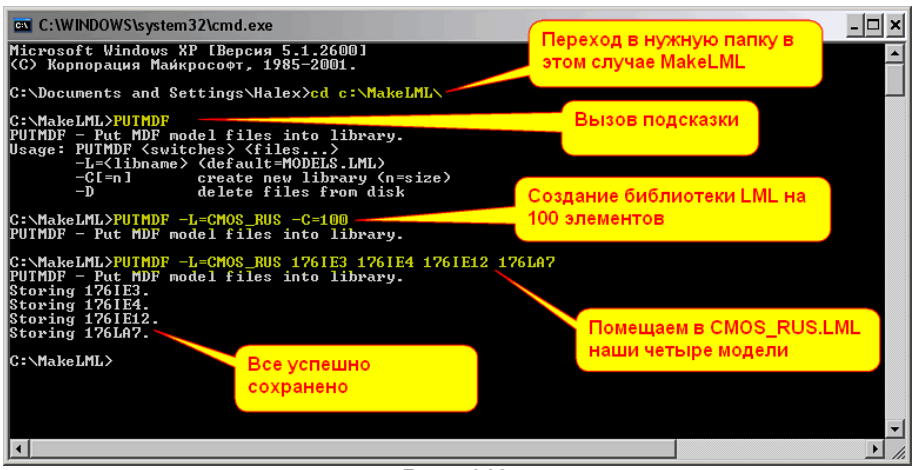


Рис. 113.

В конечном итоге в папке **MakeLML** должен появиться файл **CMOS_RUS.LML**, а после добавления в него моделей его размер увеличится. Этот файл перемещаем в папку **LIBRARY** Протеуса. Если там остались **MDF** тех моделей, которые мы поместили в нашу **LML**, то их удаляем. В дальнейшем, в эту библиотеку **LML** можно будет добавить еще **MDF** общим количеством до 100.
И еще одно замечание. Я в обоих случаях использовал одинаковое название и для **LIB** и для **LML**. Это совсем не обязательно, тут дело вашего вкуса.
Ну вот, получился такой короткий, но полезный материал. Теперь немного остановимся на **MIXED** (смешанных аналого-цифровых) моделях и перейдем к самому интересному – активным моделям. Для тех, кому «с трудом и словарем живого великорусского языка» поддается данный материал, во вложении прилагаются готовые библиотеки **CMOS_RUS**. Копируем файлы из папок архива в одноименные Протеуса и … наслаждаемся.

Возврат к содержанию

6.14. MIXED примитивы для аналого-цифровых и цифро-аналоговых преобразований.

Ранее, в **п.6.1**, мы уже познакомились с некоторыми свойствами элементарных однобитных АЦП и ЦАП и успешно применяли их при исследовании цифровых моделей. Остановимся еще раз на них подробнее и познакомимся с N-битными примитивами АЦП и ЦАП, которые нам потребуются в дальнейшем материале. Все они расположены в библиотеке **Modelling Primitives => Mixed Mode**.
Начнем со свойств элементарного АЦП (Рис. 114). Модель имеет один аналоговый вход (**A**) и один цифровой выход (**D**).

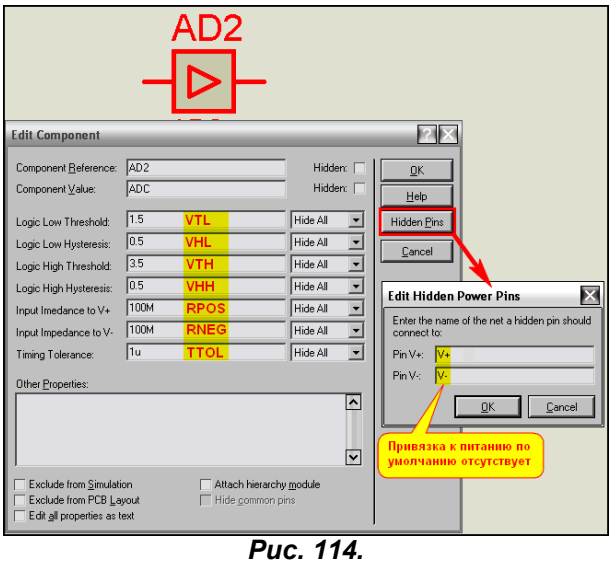


Рис. 114.

Я не буду еще раз подробно останавливаться на параметрах порогов (**VTL** и **VTH**) и гистерезисов (**VHL** и **VHH**), поскольку мы их подробно рассмотрели ранее, в **п.6.4**. Там мы рассматривали поведение примитивов цифровых буферов и инверторов и использовали только эти параметры АЦП, поскольку все остальные были уже условно привязаны через файл **ITFMOD.MDF**. Однако теперь, если мы хотим использовать при моделировании элементарный АЦП, они для нас имеют важное значение.
RPOS и **RNEG** – соответственно входной импеданс к положительному **V+** и отрицательному **V-** питающим выводам аналогового входа АЦП. Как видим, и к отрицательному и к положительному питанию сопротивление составляет по умолчанию 100 МегаОм и практически не оказывает влияние на подключаемые по входу АЦП цепи.
Выводы питания **V+ (**положительное) и **V-** (отрицательное) являются скрытыми и доступны для редактирования через кнопку **Hidden Pins** (Рис. 114). Значения по умолчанию **V+** и **V-** не являются стандартными для **Power Rails** и поэтому, если вы просто поместите в проект элементарный АЦП и запустите симуляцию, получите сообщение об ошибке. Вы вольны по своему усмотрению привязать их к любым питающим шинам и любым из рассмотренных ранее способов. Это можно проделать и через меню **Design=>Configure Power Rails** или просто добавить в окошке **Edit Hidden Power Pins** (Рис. 114) вместо **V+** и **V-** ранее сконфигурированные или существующие (например, вместо **V+** назначить **VCC**, а вместо **V-** – **GND**) питающие шины. Можно просто на листе проекта связать попарно терминалы питания **V+** и к примеру **+12V** и **V-** и **-12V**, соответственно при этом вход будет подтянут через резисторы **RPOS** и **RNEG** не к **VCC/GND**, а к этим потенциалам.
Еще одно существенное замечание состоит в том, что выход элементарного АЦП полностью оправдывает свое назначение, т.е. чисто цифровой. Он не проявляет аналоговых свойств, даже если к нему нацеплять аналоговых компонентов, соответственно он симулируется только на цифровом графике и может оказывать влияние только на вход последующего цифрового компонента. Забегая вперед, замечу, что для элементарного ЦАП по входу та же картина, но с точностью до наоборот. Для нас это означает, что если мы хотим после АЦП связать сигнал с каким либо аналоговым компонентом (даже с резистором), то необходимо на выход АЦП прилепить еще и

цифровой примитив **BUFFER**, а уж с его выхода уходить на транзисторы, резисторы, и пр. Ну и соответственно для ЦАП, при подаче на его вход сигналов с аналоговых примитивов по входу потребуется цифровой **BUFFER**. Это необходимо учитывать при создании собственных моделей с применением этих примитивов.
Ну и еще один параметр, который существенен для элементарного АЦП – **TTOL** – время переключения. По умолчанию оно составляет 1 микросекунду.
Теперь рассмотрим свойства элементарного ЦАП (Рис. 115). Модель имеет один цифровой вход (**D**) и один аналоговый выход (**A**).

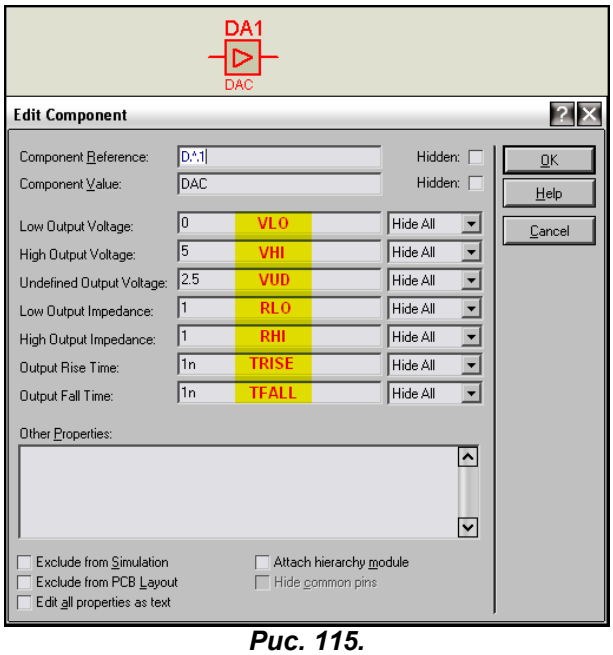


Рис. 115.

С ним мы раньше не знакомились, поэтому опишу поподробнее. Этот примитив в отличие от ADC не требует привязки каких либо напряжений к питанию, поскольку привязка выходов уже прописана в свойствах.
VLO, **VHI**, **VUD** – три выходных напряжения. Соответственно низкого уровня (**Low Voltage Output**), высокого уровня (**High Voltage Output**) и неопределенного уровня (**Undefined Voltage Output**). Ну, и их значения по умолчанию равны соответственно 0, 5 и 2,5 Вольт.
RLO и **RHI** – выходные импедансы (нагрузочные сопротивления) соответственно к нижнему (**VLO**) и верхнему (**VHI**) выходным напряжениям. По умолчанию и то и другое по 1 Ому.
TRISE и **TFAIL** – длительность соответственно переднего и заднего фронтов выходного сигнала при переключении (по умолчанию 1 наносек).
Еще один MIXED примитив, с которым мы не сталкивались раньше – это **DSWITCH** (Dual Mode Switch) – основа всех моделей аналоговых коммутаторов. Модель (Рис. 116) имеет два аналоговых, равнозначных по назначению входа/выхода – **A** (слева) и **B** (справа) и управляющий цифровой вход **EN** (сверху). Все свойства этого входа доступны через раскрывающийся список и составляют стандартный набор всевозможных задержек переключения, как и у цифровых примитивов. Поэтому на них останавливаться не будем, а рассмотрим только свойства аналогового переключателя.

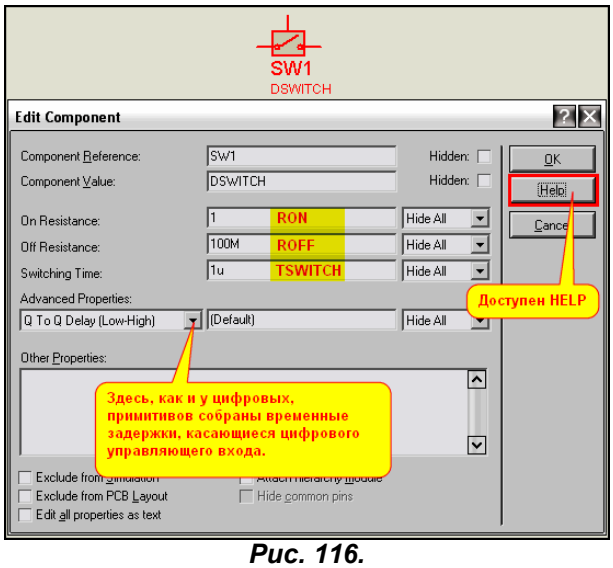


Рис. 116.

RON (1 Ом) и **ROFF** (100 МОм) – соответственно сопротивление между входами A и B во включенном и выключенном состоянии.
TSWITCH – время переключения аналогового переключателя – по умолчанию 1 микросек. Обращаю ваше внимание, что в **HELP** указаны отдельно **TON** (время включения) и **TOFF** (время выключения). Если необходимо сделать время включения и выключения различным, можно вручную задать их в окне **Other Properties**. Однако при этом **TSWITCH** надо определить как **(Default)**, иначе оно будет преобладать. Указать надо именно так, в скобках, а не цифру 0 или что-то еще. Если кто-то боится ошибиться в написании – скопируйте из окна любой из задержек и вставьте в окне **Switching Time**.
Ну, вот вроде вкратце все об элементарных смешанных моделях. Немного полезных опытов с ними сведены в проекты, представленные в папке **Mixed_Prim/OneBitPrimitives** вложения. По названию проектов **ADC**, **DAC**, **DSWITCH** – можно понять что внутри.
А мы переходим к многоразрядным АЦП и ЦАП. В отличие от элементарных, кнопка помощи для этих примитивов не доступна, но **HELP** по ним все-таки есть. Открыть его можно, выбрав соответствующие разделы через **ПУСК => Все программы => Proteus Professional=> Proteus VSM Model Help => ProSPICE Primitives** (Рис. 117).

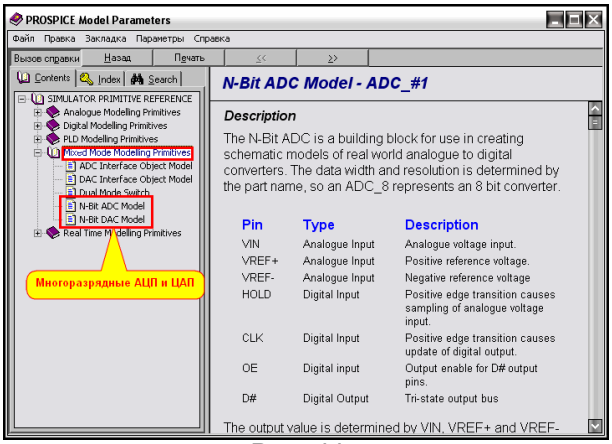


Рис. 117.

Начнем знакомство с аналого-цифровых примитивов N-bit ADC. В папке **Modelling Primitives => Mixed Mode** библиотек Протеуса представлены 8, 10, 12 и 16-ти разрядные ADC. Поскольку все параметры у них одинаковы, рассмотрим самый простой – 8-разрядный. Остальные отличаются лишь тем, что для экономии места их выходные сигналы сведены в соответствующую N-разрядную шину, а у **ADC_8** они выведены на отдельные пины. На рисунке 118 представлены вид модели в окне проекта и окно параметров по умолчанию.

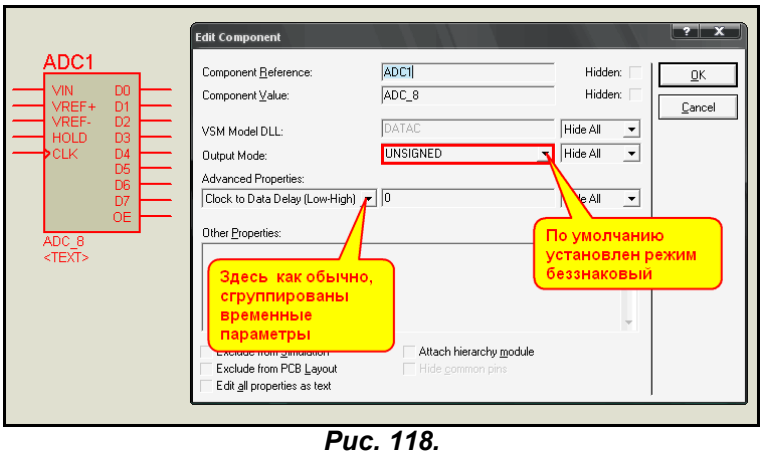


Рис. 118.

Для начала познакомимся с назначением входов/выходов модели.
Входы аналоговые:
VIN – вход аналогового сигнала АЦП.
VREF+ и **VREF-** – соответственно положительный и отрицательный входы опорного напряжения.
Входы цифровые:
HOLD – вход разрешения фиксации уровня аналогового сигнала в АЦП. По умолчанию сигнал фиксируется по положительному перепаду 0-1 на этом входе.
CLK – тактовый вход вывода сигнала на цифровые выходы. По умолчанию сигнал выводится по положительному перепаду 0-1 на этом входе.
OE – вход разрешения вывода сигнала на цифровые выходы. При лог. 1 на этом входе вывод разрешен, при лог. 0 – выходы находятся в высокоимпедансном состоянии.

Цифровые выходы:
D0…Dn – на этих выходах результат преобразования выводится в цифровом виде в соответствии с выбранным в свойствах режимом **Output Mode**.
Теперь заглянем в окно свойств. Помимо обычного набора всяческих задержек упрятанных в раскрывающийся список и по умолчанию равных 0, здесь присутствует только один важный для нас параметр **Output Mode** – режим вывода данных.
По умолчанию он принят **Unsigned**, т.е. беззнаковый. Это означает, что код на выходе АЦП задействует все разряды и равен отношению **Vin/Vref**, представленному в двоичной форме. Соответственно для **Vin=Vref** мы получим на выходе восьмиразрядного АЦП шестнадцатеричный код **0xFF**.
Если выбран режим **SIGNMAGNITUDE** – знакозависимый, то старший цифровой разряд выбранной модели используется под знак для нуля и положительных значений в этом разряде 0, а отрицательных – 1. В остальных разрядах представлен прямой двоичный код. Нулю соответствует половина **Vref**. Так, например, для 8-ми разрядного АЦП при **Vref=+5B** напряжению **Vin=+2,5B** будет соответствовать код **0x00**, **Vin=+2,51B** – код **0x01**, а **Vin=+2,49B** – код **0x81**. Соответственно необходимо помнить, что раз у нас один разряд задействован под знак, то полному положительному размаху уже будет соответствовать код с меньшей разрядностью, т.е. для **Vin=Vref** код будет **0x7F**, а для **Vin=-Vref** – код **0xFF**.
Ну и наконец, последний режим – **TWOSCOMPLEMENT**. Он аналогичен знакозависимому, но значения **Vin** ниже **Vref/2** при этом представлены в дополнительном коде. Применительно к рассмотренному выше примеру значению **Vin=+2,49B** будет соответствовать код **0xFE**, а для **Vin=-Vref** код будет **0x80**. Этот вариант наиболее часто встречается в реальности, поскольку облегчает вычисления в двоичном виде.
Если кому-то принципы преобразований непонятны – обратитесь к соответствующей литературе. В частности именно по этому вопросу рекомендую прочитать гл. 2.1. Кодирование и квантование в книге **«Аналого-цифровое преобразование» под ред. У. Кестера, М., «Техносфера, 2007**.
Ну и еще парочка практических замечаний по моделям **ADC** в Протеусе. Как Вы наверное уже догадались, для того чтобы использовать только однополярный сигнал **Vref** достаточно вход **VREF-** завесить на землю. Несколько сложнее, если необходимы аналоговые **VREF** и **VIN** никак не связанные с **GND**. Но и в этом случае все решается достаточно просто. На входы АЦП добавляются примитивы **AVCVS** или по-русски ИНУН (источник напряжения управляемый напряжением) с коэффициентом передачи **1,0** в качестве гальванических разделителей, как показано на Рис. 119. При этом не забудьте, что они имеют бесконечное входное сопротивление, поэтому при имитации реальных устройств потребуются входные резисторы с нужным сопротивлением.

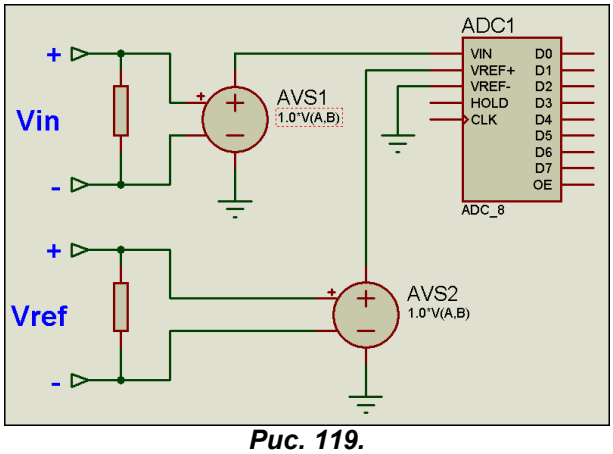


Рис. 119.

Параметры N-разрядных **DAC** (ЦАП) аналогичны параметрам АЦП, если принять во внимание, что входы/выходы здесь поменялись местами (Рис. 120).

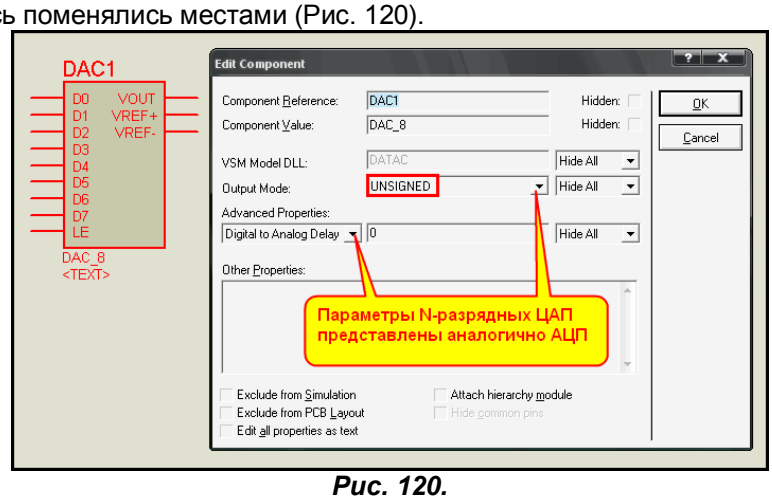


Рис. 120.

Высокий уровень на входе **LE** (**Load Enable**) позволяет запись цифрового кода со входов **D0…Dn** во внутренние регистры ЦАП. После возвращения **LE** в ноль они защелкиваются там до следующего высокого уровня на **LE**. Как видим, управление здесь еще проще, чем в ЦАП, да и времянок всего две, так что есть смысл их описать.
TDDA – **Digital To Analog Delay** – задержка преобразования из цифрового кода в аналоговый сигнал. По умолчанию она нулевая.
SLEWRATE – **Slew Rate (Volts/Sec)** – скорость нарастания выходного аналогового напряжения. По умолчанию она равна **1000000 В/сек**.
Раскрывающееся меню режима **Output Mode** в параметрах ЦАП полностью аналогично рассмотренному выше для АЦП и содержит те же опции.
Примеры применения примитивов ADC и DAC для создания схематичных моделей есть в стандартной поставке Протеуса. В папке **SAMPLES\Graph Based Simulation** находятся два примера для АЦП: **ADC0808.DSN** и **ADC0831.DSN** и пример ЦАП: **DAC0808.DSN**. Во всех примерах при щелчке правой кнопкой по изображению ADC или DAC возможен переход на дочерний лист, где и представлена соответствующая модель «вид изнутри». Разбирать эти стандартные примеры здесь не имеет особого смысла, поскольку базируются они на тех примитивах, которые рассмотрены выше.
В качестве тестовых примеров для изучения поведения 8-ми битных АЦП и ЦАП во вложении **Mixed_Prim** приложены соответствующие проекты в папках **ADC_8** и **DAC_8**. Некоторые комментарии даны прямо в проектах. Ниже мы практически познакомимся с моделированием 12-ти разрядного реального АЦП **MAX1241**, где будет задействован примитив N-битного ADC. Почему именно **MAX1241**? Да просто мне понадобилась эта модель для собственных нужд, ну и выяснилось, что модель даже в последних версиях приложена Протеуса не без ошибок. Вот мы и займемся «работой над ошибками», а заодно и поучимся моделировать реальные АЦП. Но прежде нам придется освоить еще один очень полезный примитив, описание которого отсутствует в HELP Протеуса. Это **SPISLAVE** – модель подчиненного последовательного интерфейса. Именно по **SPI** АЦП **MAX1241** общается с внешним миром.

Возврат к содержанию

6.15. Примитив SPISLAVE. Исследуем поведение последовательного интерфейса с помощью различных цифровых генераторов.

Наряду с обычными аналоговыми, цифровыми и смешанными примитивами, описание которых хоть как то представлено в HELP, в библиотеках Протеуса существует ряд моделей, на основе которых создаются схематичные компоненты, но описания их свойств полностью отсутствуют. К таковым как раз и относятся расположенные в папке **Modelling Primitives/Digital(Miscellaneous)** примитивы различных подчиненных интерфейсов **SPISLAVE** (интерфейс SPI), **I2SSLAVE** (интерфейс I2S), **D1WSLAVE** (интерфейс 1-wire). Для воссоздания MAX1241 нам потребуется SPI. Вот его-то мы и рассмотрим сейчас. Примитив **SPISLAVE**, как и АЦП/ЦАП изначально представлен для 8-ми, 12-ти и 16-тибитного интерфейсов. Как и ранее рассмотрим только 8-ми битный, остальные аналогичны. Для начала познакомимся с назначением выводов модели (Рис. 121).

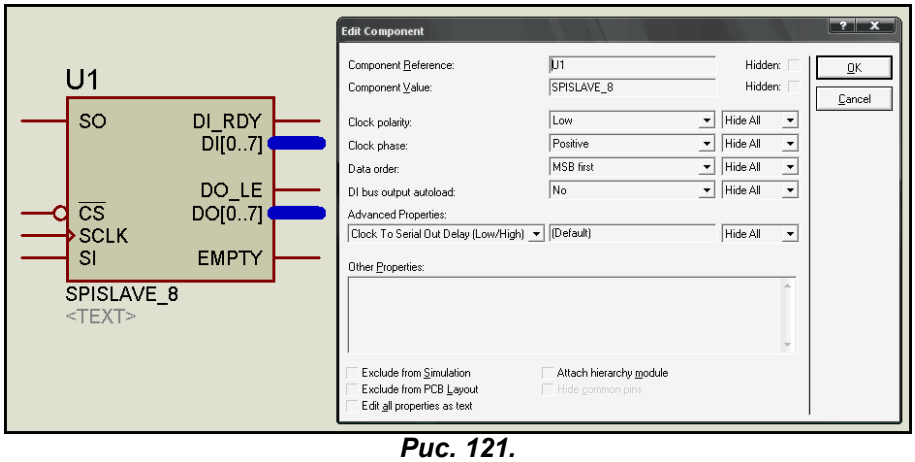


Рис. 121.

- В левой части модели расположены выводы характерные для внешнего последовательного интерфейса. К ним относятся:

SO – **SPI Output** – цифровой последовательный выход данных на другие компоненты.
SI – **SPI Input** – цифровой последовательный вход данных от других компонентов.
CS – **Crystal Selector** – цифровой вход. По умолчанию низкий уровень на нем служит для выбора (активации) модели.
SCLK – **SPI Clock** – тактовый вход. На него подаются импульсы с частотой обмена информацией по интерфейсу.
В правой части модели расположены выводы и 8-ми разрядные параллельные шины для компоновки внутренней структуры модели какого-либо компонента. Сюда входят:

DI_RDY – **Data Input Ready** – цифровой выход выдачи сигналов принятых со входа **SI** на параллельную шину **DI[0..7]**. По умолчанию вырабатывает положительный перепад 0-1 по окончанию (перепаду 0-1) входного сигнала **CS**. При этом принятые данные выводятся на шину. Поведение выхода зависит от параметра **DI bus output autoload** (см. ниже).
DO_LE – **Data Output Load Enable** – цифровой вход для выгрузки данных с параллельной шины **DO[0..7]** на выход **SO**. Для правильной (неискаженной) выдачи кода на **SO** положительный перепад 0-1 на этом входе должен предшествовать первому тактовому импульсу (по умолчанию положительному фронту) на входе **SCLK** после активизации сигнала **CS**.
EMPTY – цифровой выход. Сигнал на нем зависит от параметра **DI bus output autoload** (см. ниже).

- Теперь рассмотрим изменяемые в окне свойств параметры модели:

Два параметра: **Clock polarity** (по умолчанию **Low** – низкий уровень) и **Clock phase** (по умолчанию **Positive** – положительный 0-1) описывают требуемый входной тактовый сигнал **SCLK**. При необходимости можно поменять на **High** и **Negative** соответственно, чтобы подобрать нужный фронт тактирования модели.
Data order – порядок выдачи (приема) данных в последовательном интерфейсе. По умолчанию принят **MSB first**, т.е. старший бит данных (для 8-ми разрядной модели это **D7**) выдается (принимается) первым. Если поменять на **LSB first**, то первым в последовательности будет **D0**.
DI bus output autoload – этот параметр определяет логику работы (автозагрузку) шины данных **DI[0..7]**. По умолчанию **[Default]** эквивалентно **No** автозагрузка запрещена. Это означает, что модель не делает подсчет входных тактовых импульсов, а производит выгрузку данных на внутреннюю шину **DI** по окончании действия сигнала на входе **CS**. Чем это чревато рассмотрим сразу, не откладывая в долгий ящик.
Предположим, мы загружаем в восьмиразрядный **SPISLAVE** по входу **SI** кодовую последовательность **0x91 (b10010001)**. При этом сигнал выборки **CS** у нас значительно длиннее, чем длительность восьми тактовых импульсов, а тактовые импульсы следуют непрерывно, т.е. их тоже пройдет больше восьми. Фактически универсальная программная модель **SPISLAVE** представляет собой сдвиговый регистр, а точнее два – один внутрь на шину **DI**, другой наружу – на выход **SO**. Разрядность регистров определил программист Лабцентра при создании модели, и она нам неизвестна. Хотя, учитывая некоторый мой опыт по исследованию моделей, могу предположить, что там заложено не менее 32 разрядов. Допустим, что **CS** завершится у нас после 10-го тактового импульса. При этом два лишних импульса протолкнут данные на два разряда влево, добавив два нуля в младшие разряды. Если первым следовал **MSB**, то разряды **D7** и **D6** уйдут влево за пределы байта, на место **D6** встанет единица из бывшего **D4**, а на **D2** единица из бывшего **D0**. Проверим это на практике. На две модели подадим одинаковые сигналы, но у первой включим режим **DI bus output autoload** (**Yes**), а у второй оставим по умолчанию (Рис. 122).

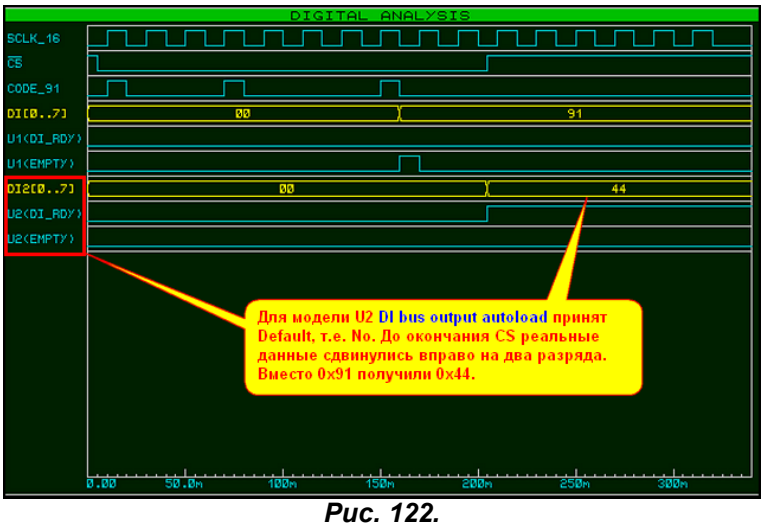


Рис. 122.

Из графика видно, что предположения полностью подтвердились, т.е. для второй модели (**U2**) мы имеем на выходе код **0x44** (**b01000100**). Справа в этом байте появились два нуля, обусловленные двумя лишними импульсами. Для модели **U1**, у которой включен режим **DI bus output autoload** данные были занесены в **DI** правильно. Я прошу вас обратить особое внимание на поведение выходов **DI_RDY** и **EMPTY** при включенном (**U1**) и выключенном (**U2**) режиме автозагрузки. В первом случае по окончании 8-го импульса (именно для **SPISLAVE_8**, а для **SPISLAVE_12** будет по окончании 12-го) на выходе **EMPTY** появился импульс и в этот момент появились правильные данные на шине **DI[0..7]**, в то время как **DI_RDY** вообще не принимал участия в обмене и оставался строго низким. Во втором случае **EMPTY** был безучастным, но зато на **DI_RDY** появился положительный перепад по окончании сигнала **CS** и именно в этот момент были выданы данные, проехавшие на пару разрядов вперед на шину **DI[0..7]**. Это может быть несколько сложный для сиюминутного восприятия, но очень важный момент поведения модели **SPISLAVE** при приеме данных и о нем нельзя забывать при использовании примитива для разработки собственных моделей компонентов.

Во временных параметрах модели **SPISLAVE** присутствуют всего две задержки, по умолчанию нулевые. Они описывают задержку выдачи сигнала в последовательный интерфейс относительно тактового **Clock** одна по перепаду 0-1, другая по перепаду 1-0.
Чтобы несколько подробнее познакомиться с поведением модели я приложил несколько простых примеров приема, передачи и одновременно приема/передачи (поскольку каналы **SI** и **SO** если их не объединить специально абсолютно не мешают друг другу, то возможно и такое) с использованием различных цифровых генераторов из левого меню **Generator Mode**. Конечно, можно было бы воспользоваться и подручной моделью микроконтроллера, написать программу обмена и т.п. Но я хочу попутно приучить вас использовать «подручные средства» для достижения своих целей. Цифровые генераторы из меню **Generator Mode** наиболее подходящие для этих целей, поскольку практически не загружают ЦП компьютера и позволяют получить нужный результат. Элементарный пример – для первоначального сброса МК многие пририсовывают в проект обычную RC-цепочку. И свято верят, что это обязательно нужно при моделировании и работает. Кроме бесполезной траты времени компьютера на вычисление начальной точки аналоговой цепи данный прием ничего хорошего не дает. В реальности она работать будет, в симуляции необходимо, по крайней мере, задать начальные условия (**IC**) для точки соединения R, C и входа сброса МК, ну или задать нулевой начальный заряд конденсатора (**PRECHARGE**). В то же время, воспользовавшись генератором перепада (**DEDGE**) расшифровывается как **Digital Edge** (цифровой перепад) можно на сколь угодно долго задержать старт МК или другого цифрового компонента, имеющего вход сброса. На рисунке 123 пример «безграмотного» и грамотного применений стартовой задержки. Как видите, гарантированные 250 миллисекунд дает только генератор цифрового перепада. И совсем не обязательно применять такой генератор именно для коротких задержек. Вы можете задать ему время перепада и через **3600s** (1 час), если требуется имитировать включение какого либо устройства.

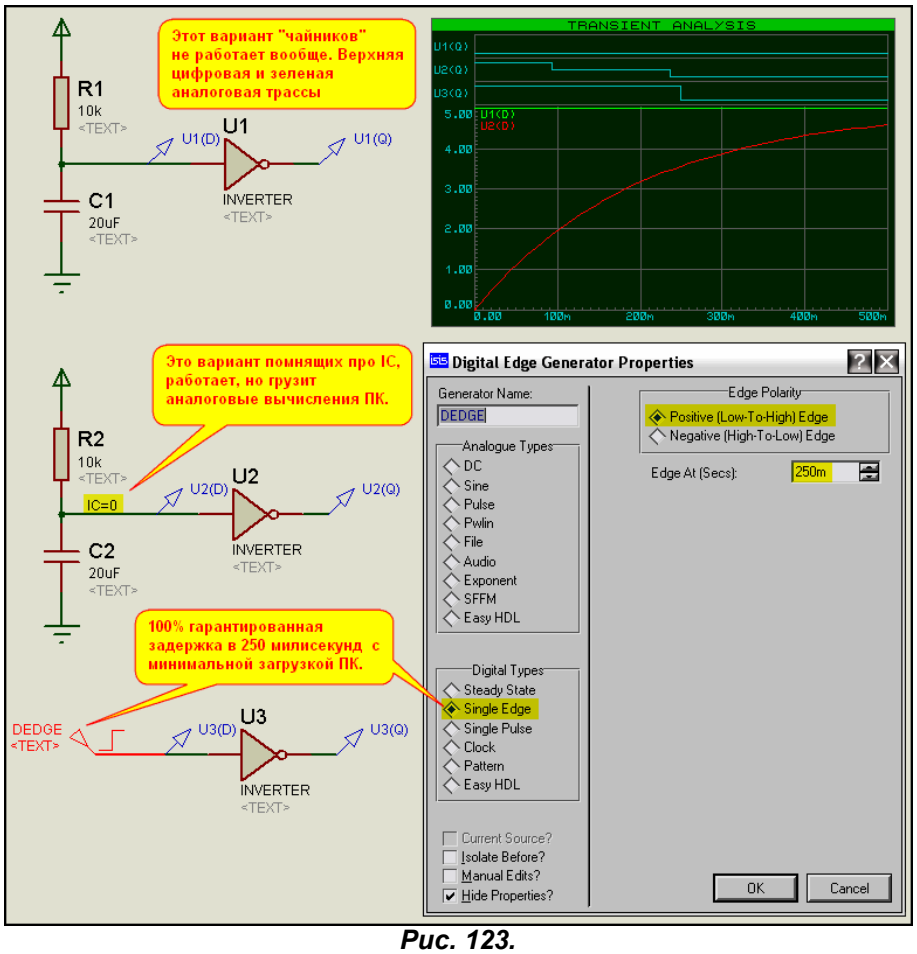


Рис. 123.

Аналогично можно задать и не единичный перепад, а единичный импульс **Single Pulse** или непрерывную тактовую **Clock** именно из генераторов **Digital Types**. Только параметров там уже можно задать больше. Почему я так настойчиво упираю на применение для цифровых устройств именно этих типов? По-прежнему на форуме вылезают кривые проекты со 100% загрузкой ЦП. Открываешь проверять – вот оно, вместо **Digital Clock** автор кривого проекта пихнул **Pulse** из расположенного выше окна **Analogue Types**. В проекте и так напихано аналоговых элементов, так еще и генератор аналоговый. Эти два селектора не зря разделены. Все что аналоговое – оно и работает как аналоговое, со всеми вытекающими при этом дополнительными вычислениями: токи, импеданс и т.п. – отсюда и лишние тормоза. Я в последний раз заостряю на этом внимание, просто устал повторяться. А затеял я разговор о цифровых генераторах по поводу генератора **Pattern**, который мы сейчас и применим для исследования нашего **SPISLAVE**. Будут там и **Single Edge** и

Single Pulse, но главным инструментом будет этот. Давайте познакомимся с назначением его свойств (Рис. 124) и применим нужный нам режим, а мне надо получать последовательность из определенного количества импульсов для входа **SCLK** и синхронизированные с **SCLK** во времени последовательные кодовые комбинации для входа **SI**.

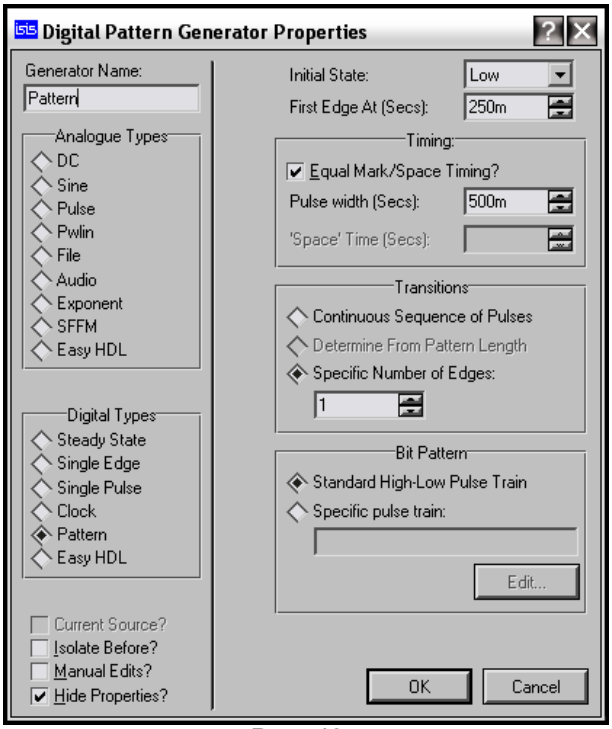


Рис. 124.

В верхней части окна свойств расположены:
Initial State (стартовое состояние) по умолчанию **Low** (лог. 0) – меня это устраивает и мы оставляем как есть.
First Edge At (Sec) – первый перепад сигнала через **250m** (миллисек) – слишком долго ждать, я поставлю **10m**.
В следующей группе **Timing** с помощью флажка **Equal Mark/Space Timing** определяется, будут ли длительности импульса и паузы одинаковыми. Если флажок убрать, то можно задать отдельно длительность импульса **Pulse width** и паузы **'Space' Time**. Я флажок оставлю, но длительности подрежу с 500 до тех же **10m**.
Следующая группа **Transitions** определяет, как будут выдаваться импульсы. Если переключатель стоит так, как на рисунке 124 **Specific Number of Edges**, то будет выдано определенное в окне ниже число импульсов (в данном случае 1). Если переключатель установить в **Continuous Sequence of Pulses**, то будет выдаваться бесконечно повторяющаяся последовательность импульсов (по сути, получим цифровой **Clock**). Третье положение **Determine From Pattern Lengh** – определяемое продолжительностью схемы пока не доступно для включения (серое). Но именно оно меня и интересует.
Для того чтобы оно стало активным переведем переключатель в группе **Bit Pattern** из положения **Standard High-Low Pulse Train** – стандартная последовательность нулей и единиц в положение **Specific pulse train** – заданная последовательность импульсов. Вот теперь в наборе выше нам стали доступны все положения. Там я ставлю **Determine From Pattern Length**, а в группе Bit Pattern щелкаю по кнопке **Edit**, Которая стала активной. В результате откроется окно редактирования последовательности импульсов **Edit Pattern** (Рис. 125).
В окне **Edit Pattern** с помощью мышки можно набрать любую требуемую нам комбинацию импульсов, причем с различной длительностью и тремя доступными уровнями: **High** (лог. 1), **Float** (неопред. уровень) и **Low** (лог. 0). Всего доступно 48 временных интервалов – горизонтальная шкала. Длительность каждого интервала определяется заданным в группе **Timing** значением **Pulse width**. Таким образом, если я задал там значение **10m**, каждая горизонтальная клетка равна 10 миллисекундам. Установка уровня в нужном месте осуществляется левой кнопкой мышки. Если необходимо заполнить несколько временных интервалов одним уровнем – просто проводим по нужному уровню с зажатой левой кнопкой мыши. Если все 48 интервалов не нужны, то незадействованные не заполняем. Они будут индицироваться пунктирной линией, и воспроизводиться при симуляции не будут. Ну и если случайно вы заполнили в конце лишние интервалы их можно затереть (пунктиром), используя правую кнопку мыши.
Ну и последний нюанс – мне в данный момент нужен режим **Determine From Pattern Lengh**, но совсем не обязательно использовать именно его. Если поставить **Continuous Sequence of Pulses**, то набранная в окне редактирования схема выдачи импульсов при запуске симуляции будет повторяться бесконечно.

Для сигнала **SCLK** на рисунке 125 я задал 16 импульсов с равными длительностью и паузой. В принципе, для 8-ми разрядного **SPI** хватило бы и 8-ми, но надо же посмотреть – как поведет себя модель, когда на вход будут поступать лишние импульсы.

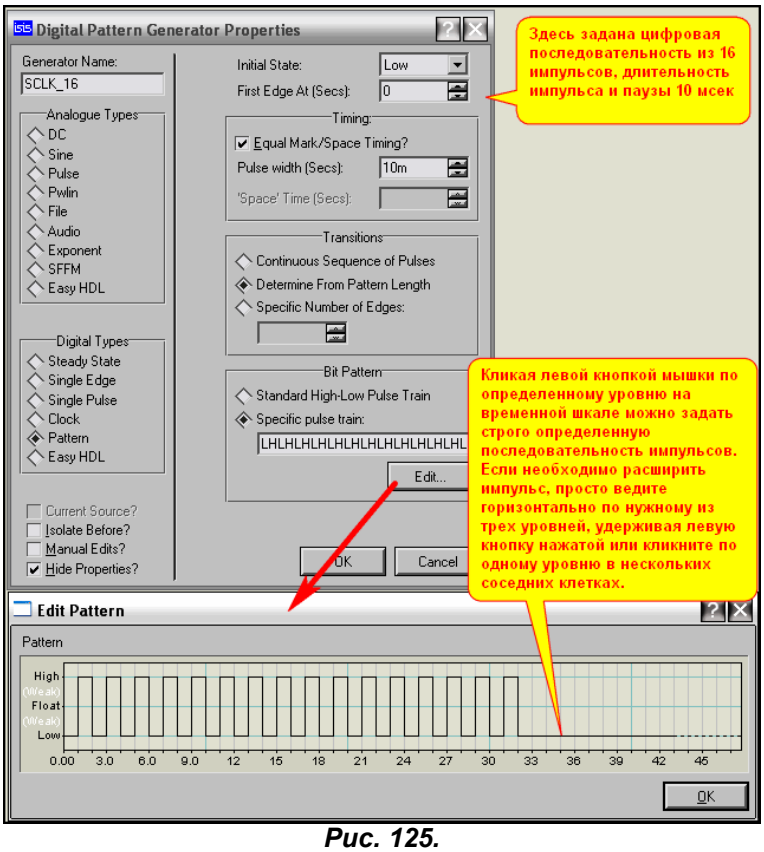


Рис. 125.

Для того чтобы подать на вход **SI** определенную кодовую комбинацию я просто через **Block Copy** скопирую в проекте этот генератор, подключу копию к входу **SI**, а затем подредактирую нужную мне комбинацию **Pattern**, убрав ненужные импульсы. На рисунке 126 показано окно **Edit Pattern** для подачи на вход **SI** кодовой комбинации **0x91**.

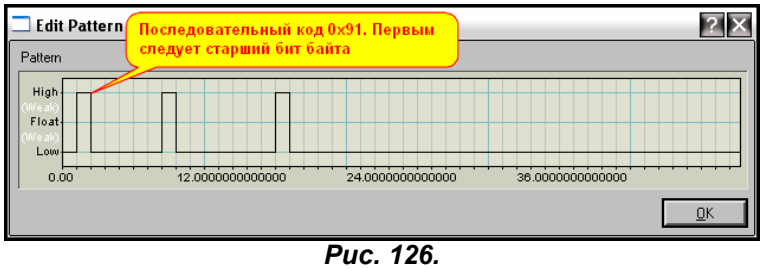


Рис. 126.

Ну, вот вроде и все об использовании генератора **Pattern**. Теперь о том, что во вложении **SPISLAVE.RAR** к этому разделу. В примере **SPI_8_autoload.DSN** показано влияние свойства **DI bus output autoload** на прием информации с входа **SI** на внутреннюю шину **DI**. В примере **SPI_8_Out.DSN** показана передача информации на вывод **SO** с внутренней шины **DO**. Ну и наконец, в примере **SPI_8_2direction.DSN** показан одновременный прием и передача информации по последовательным выводам **SI** и **SO**. Дополнительные комментарии вы найдете непосредственно в проектах.

Возврат к содержанию

6.16. 12-ти разрядный АЦП со SPI интерфейсом MAX1241. Анализируем поведение модели в Протеусе.

Для того чтобы понять – правильно ли работает модель АЦП **MAX1241** необходимо первоначально обратиться к даташиту на данный АЦП. Можно скачать непосредственно из Протеуса, поместив модель в поле проекта и нажав на кнопку **Data** в свойствах (естественно, канал Интернета при этом должен быть подключен). Но при этом вы заполучите даташит ревизии 2 от 1998 года. Более свежий вариант Rev. 5 от 2010 года доступен на сайте фирмы Maxim: http://www.maxim-ic.com
Даташит единый на **MAX1240/MAX1241**. Отличаются эти микросхемы только тем, что **MAX1240** имеет встроенный источник опорного напряжения 2,5 Вольта. Для тех, у кого с английским так и не заладилось, могу порекомендовать найти книжку **П. Гелль «Как превратить персональный компьютер в измерительный комплекс», М., ДМК,1999**. Правда, там **MAX1241** касаются только

боком, но подробно рассмотрен **MAX1243** – его десятиразрядный «собрат». Некоторые, необходимые нам в дальнейшей работе выдержки из даташита 2010 года в моем персональном, может и чуть «корявом» переводе, будут размещены и здесь. Чтобы оценить «качество перевода», эти фрагменты я буду выделять бирюзовым цветом, некоторые мои пояснения будут вставлены курсивом. Итак, познакомимся для начала с самой микросхемой. Функциональная диаграмма **MAX1241** приведена на рисунке 127.

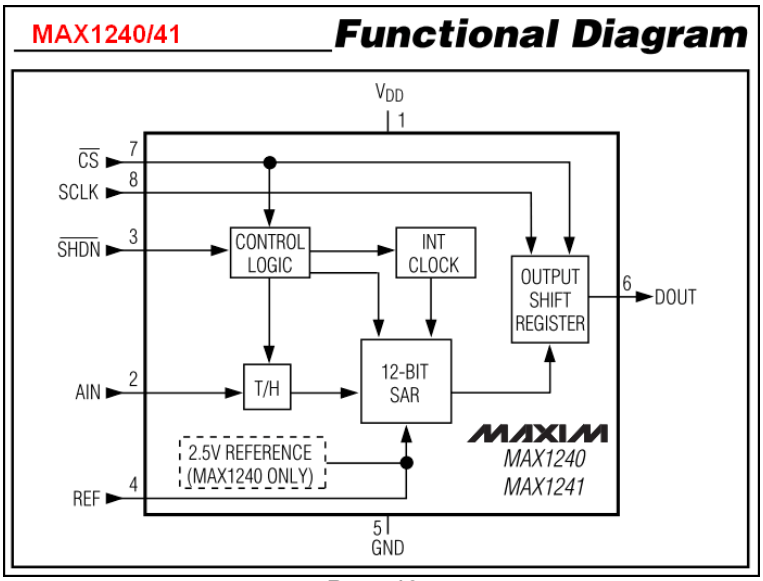


Рис. 127.

Рассмотрим назначение выводов микросхемы (***Pin Description стр.7***):

1 – **VDD** – Плюс питания: 2,7-3,6V (MAX1240); 2,7-5,25V (MAX1241).

2 – **AIN** – Вход аналогового сигнала в диапазоне от 0 до Vref.

3 – **SHDN** – Трехуровневый вход **Shutdown** (отключение). Подача на вход низкого уровня переводит MAX1240/MAX1241 в режим пониженного энергопотребления (около 15мкА). Обе микросхемы полностью работоспособны при высоком уровне или неподключенном выводе **SHDN**. Для MAX1240 подача на **SHDN** высокого уровня подключает внутренний источник опорного напряжения, если оставить **SHDN** неподключенным, подразумевается использование внешнего опорного напряжения.

4 – **REF** – Опорное напряжение для АЦП. Внутренний источник 2,5V для MAX1240 шунтируется на землю конденсатором 4,7мкФ. Вход внешнего опорного напряжения для MAX1241 и MAX1240 (при отключенном внутреннем источнике) шунтируется на землю конденсатором как минимум 0,1мкФ.

5 – **GND** – вывод цифровой и аналоговой земли.

6 – **DOUT** – выход последовательного интерфейса. Данные изменяются по спаду импульса на входе **SCLK**. При высоком уровне на входе **CS** выход **DOUT** находится в высокоимпедансном состоянии.

7 – **CS** – Низкий уровень на этом входе активизирует выбор кристалла. Когда на **CS** высокий уровень, **DOUT** находится в высокоимпедансном состоянии.

8 – **SCLK** – Вход тактовой частоты. Частота импульсов на этом входе может быть в диапазоне до 2,1МГц.

Теперь рассмотрим, как происходит синхронизация и управление **MAX1241** (***Timing and Control стр.10***)**.**

Старт преобразования и операция считывания данных контролируются с помощью сигналов по входам **CS** и **SCLK**. Временные диаграммы иллюстрирующие данные операции приведены на рисунках 8 и 9 даташита. Первый из них, со своими комментариями я приведу на Рис. 128, поскольку он играет важную роль в понимании дальнейшего материала.

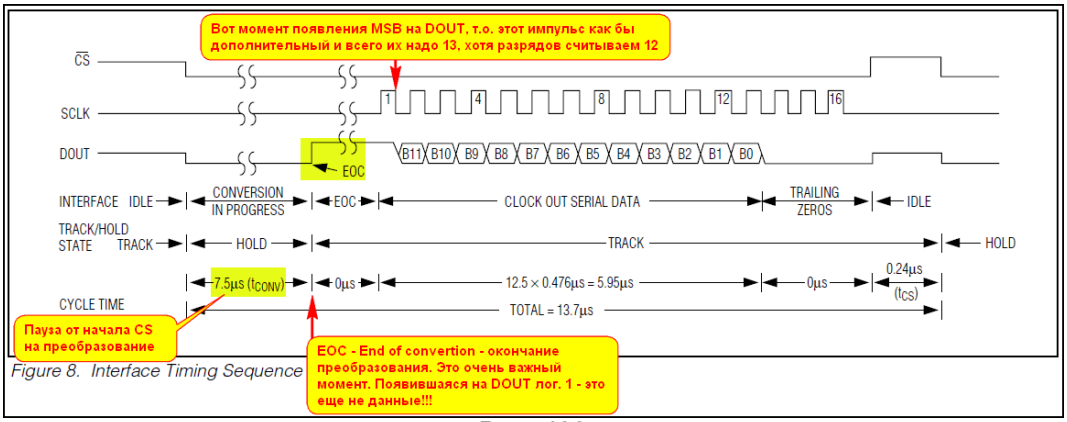


Рис. 128.

Спадающий сигнал на входе **CS** инициирует очередное преобразование: **T/H** запоминает входное напряжение (на внутреннем конденсаторе), АЦП начинает преобразование, а на выходе **DOUT** появляется логический ноль. В течение всего времени преобразования (7,5 мксек) **SCLK** должен удерживаться низким уровнем. Внутренний регистр АЦП в это время запоминает информацию.
Окончание преобразования (**EOC**) отмечается появлением высокого уровня на выходе **DOUT**. Передний фронт **DOUT** может быть использован в качестве сигнала прерывания. Каждым импульсом **SCLK** после окончания преобразования данные сдвигаются из внутреннего регистра АЦП. На **DOUT** они появляются по спаду **SCLK**. Первый спад **SCLK** инициирует появление на выходе **DOUT** бита **MSB**, следующий спад – следующий бит. Таким образом, для вывода всех 12 бит преобразования и одного предшествующего высокого уровня необходимы 13 импульсов. Последующие лишние импульсы до ближайшего изменения **CS** к высокому уровню вызывают появление на **DOUT** дополнительных нулей и не оказывают влияния на операцию преобразования.
Минимальное время цикла получения информации от **ADC** достигается с использованием переднего фронта изменения сигнала на **DOUT** в качестве сигнала **EOC**. Оно составит 12,5 циклов тактового сигнала на максимальной скорости. По окончании считывания младшего бита **LSB** необходимо поднять сигнал **CS** в логическую единицу. Выждав определенное минимальное время (**tcs**=0,24мксек), спадом сигнала **CS** можно инициировать следующее преобразование.
Ну и, пожалуй, последняя нужная нам выдержка из даташита на **MAX1240/MAX1241**, касающаяся подключения к стандартным последовательным интерфейсам (***Connection to Standard Interfaces стр. 11).***
Интерфейс **MAX1240/MAX1241** полностью совместим со стандартными интерфейсами **SPI/QSPI** и **MICROWIRE**. Рисунок 11 в даташите (Рис. 129 здесь).

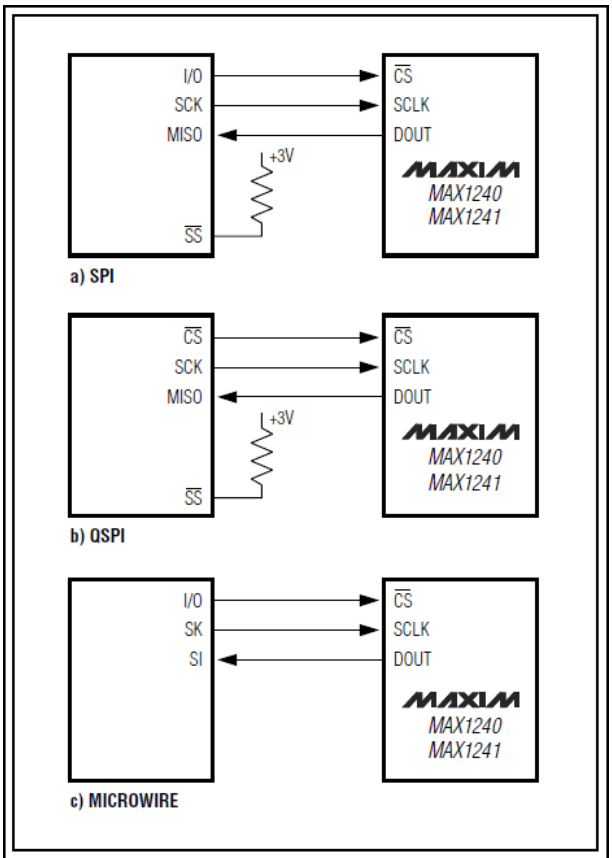


Figure 11. Common Serial-Interface Connections to the MAX1241

Рис. 129.

Если последовательный интерфейс имеется в наличии у вашего ЦПУ *(микроконтроллера - МК)*, установите его в режим «мастер» для генерации тактового сигнала. Выберите частоту тактового сигнала до 2,1 МГц.

1) Используйте стандартные команды ввода/вывода ЦПУ для изменения сигнала **CS** в логический ноль. Сохраняйте при этом **SCLK** низким.
2) Выждите максимально необходимое время преобразования перед активацией сигнала **SCLK**. Альтернативно для определения окончания преобразования можно отслеживать изменение сигнала в логическую единицу на линии **DOUT**.
3) Активируйте **SCLK** минимум на 13 тактов. Первый спад импульса выводит **MSB** цифрового результата преобразования на линию **DOUT**. Изменения выходных данных на **DOUT** происходят по заднему фронту импульсов **SCLK** в стандартном формате **MSB-first** *(старший разряд – первый)*. Соблюдайте при этом допустимые временные характеристики

SCLK. Данные могут быть синхронизированы в МК по переднему фронту *(следующего)* импульса S**CLK**.

4) Установите **CS** высоким уровнем после прохождения 13-ти задних фронтов импульсов на **SCLK**. Если **CS** остается низким, дополнительные нули на **DOUT** будут считаны по каждому лишнему импульсу **SCLK**.
5) Когда **CS=1** выждите минимально необходимое время **tcs** перед началом следующего преобразования (**CS=>0**). Если преобразование было прервано установкой **CS=1** до завершения предыдущего, выждите минимально необходимое время **tacq** перед началом следующего преобразования.

Сохраняйте **CS** низким до тех пор, пока все данные не будут считаны. Данные могут быть считаны двумя байтами или непрерывной последовательностью, как показано на рисунке 8 (даташит, 128-здесь). При считывании двумя байтами данные содержат одну стартовую лишнюю единицу в начале и дополнительные нули *(три нуля)* в конце.

Вот этот последний момент для нас и важен.

Ну, еще маленькие выдержки из даташита по стандартным последовательным интерфейсам. Для всех трех типов интерфейса устанавливается CPOL = CPHA = 0. В отличие от стандартного SPI, требующего считывания 2 отдельных байтов, QSPI использует минимально необходимое количество тактовых импульсов для считывания. На этом, пожалуй, цитаты можно закончить и перейти к практике.

Далеко заходить не будем, возьмем стандартный пример из **CodeVision AVR** для работы с **MAX1241**. Те, кто использует **CodeVision**, могут найти его в папке **Examles\MAX1241**. Для более старых версий он с использованием **AT90S8515**. Я использую версию 1.25.9 – там **ATMEGA8515**. Конечно же, кроме самого проекта **CodeVision**, который необходимо откомпилировать, нам потребуется тестовый проект в Протеусе (Рис. 130). Я опять применил на аналоговых входах собственные модели источников напряжения, просто с ними удобнее здесь работать – все видно наглядно.

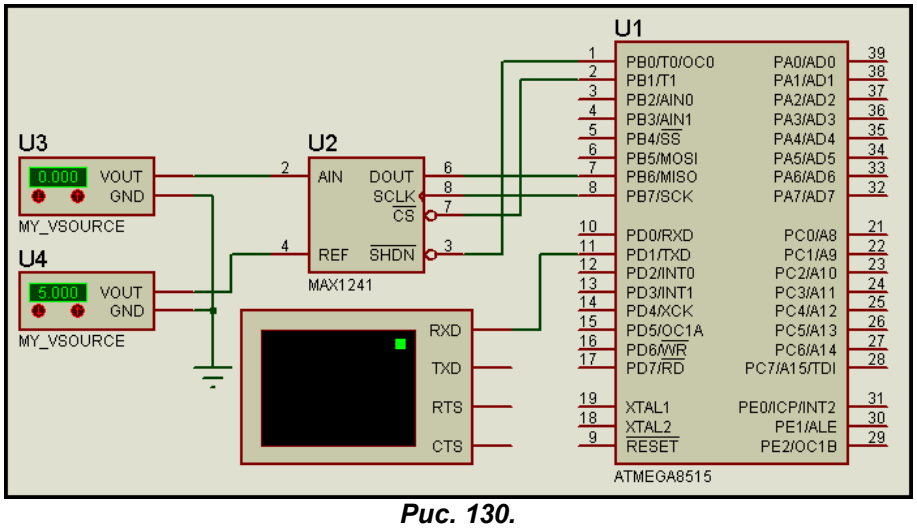


Рис. 130.

Желающие, могут запустить данный проект и убедиться, что нормальное считывание информации наблюдается только при нулевом напряжении на входе **AIN**, дальше начинается полный бред. Проект находится во вложении в папке **CV/MAX1241**. На Рис. 131 график считывания информации из этого проекта. Растянут второй цикл чтения, поскольку первый считает просто нули.

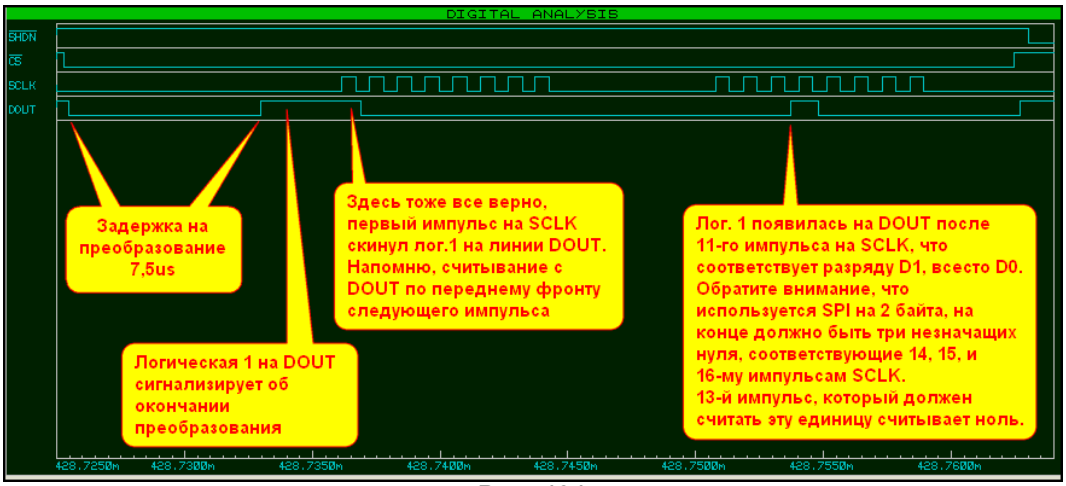


Рис. 131.

Из графика видно, что используется чистый протокол SPI, считывания 2-х байтов. Если заглянуть в проект **CodeVision** – так оно и есть, вначале имеется директива подключения библиотеки SPI:

```
#include <spi.h>
```

А в самом начале программы находится сама процедура считывания:

```
unsigned int max1241_read(void)
{
union adcu adc_data;
// exit MAX1241 from shutdown
NSHDN=1;
// wait 5us for the MAX1241 to wake up
delay_us(5);
// now select the chip to start the conversion
NCS=0;
// wait the conversion to complete
// DOUT will be 0 during conversion
while (DOUT==0);
// DOUT=1 -> conversion completed
// read MSB
adc_data.byte[1]=spi(0);
// read LSB
adc_data.byte[0]=spi(0);
// deselect the chip
NCS=1;
// enter shutdown
NSHDN=0;
// now format the result and return it
return (adc_data.word>>3)&0xfff;
}
```

Обратите внимание на последний оператор возврата. Если его записать в виде:

```
return (adc_data.word>>4)&0xfff;
```

Т.е. сдвинуть лишний раз данные вправо, то все будет работать правильно. Но это в Протеусе, а в «железе» мы получим обратный эффект – все исказится.

Чтобы было нагляднее, я несколько усложнил проект из **CodeVision** и заставил его выводить информацию еще и в двоичном коде. При этом наглядно видно, что при изменении напряжения на **AIN** на **0,001V** вместо разряда **D0** (самый правый) меняется **D1** (второй справа). Таким образом, получается, что данные как бы смещены влево на 1 разряд. Этот пример во вложении называется **MAX1241BIN**.

Ну и чтобы окончательно убедиться, что это не **CodeVision** нам портит картину, а именно модель – считаем информацию с помощью обычных генераторов, как мы это делали с примитивом **SPISLAVE**. Подадим, как велит даташит, 13 импульсов чтения на **SCLK** и посмотрим результат. Этот вариант в примере вложения **GEN_READ**. График из этого примера при чтении с **AIN** напряжения 1mV и VREF=5V приведен на рисунке 132.

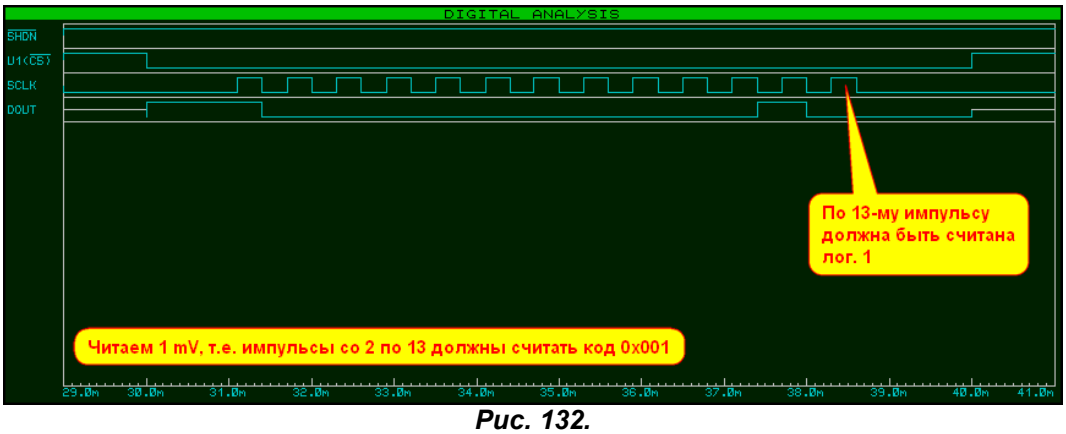


Рис. 132.

Поскольку чтение происходит с тем же дефектом, можно констатировать окончательно и бесповоротно, модель **MAX1241** глючит, но дело поправимое, т.к. организована она чисто с помощью подсхемы, с которой мы познакомимся далее и попробуем поправить положение.

Возврат к содержанию

6.17. MAX1241 Schematic Model – взгляд изнутри. Ищем и исправляем ошибку моделей MAX1241 и MAX1240.

Пора перейти к рассмотрению непосредственно модели **MAX1241**. Для этого нам потребуется извлеченный, как и ранее с помощью **GETMDF** файл **MAX1241.MDF**. Расположен он в **MODELS\MAXIM.LML**. Процедура извлечения мной уже не раз описывалась, останавливаться не буду. Для особо ленивых просто приложу извлеченный файл во вложении. Далее идем стандартным путем и по этому файлу начинаем восстанавливать изначальную схему, с которой

формировался MDF. И тут выясняется «приятная» неожиданность в разделе **PARTLIST** мы встречаем следующую строчку:

I1,SPIIO,SPISLAVE_12,MODDLL=SPIIO.DLL,PRIMITIVE=DIGITAL,SPI_AUTOLOAD=0,SPI_CPHA=0,SPI_CPOL=0,SPI_DORD=0

Но в библиотеке нет двенадцатиразрядного **SPISLAVE_12** – что делать? Ответ прост – создать. Все дело в том, что когда программист Лабцентра писал библиотеку **DLL** для модели **SPISLAVE**, то, конечно же, сделал ее универсальной. Посмотрите модели **SPISLAVE_8** и **SPISLAVE_16**. Обе они привязаны к **SPIO.DLL**. Представим себя тоже немного англичанами. Причем конкретно проживающими на Бейкер-стрит, курящими трубку и играющими по вечерам на скрипке. Раз есть 8 и 16, то почему не быть 10, 12 или 14, да можно даже 11 и 13. Проверив все свойства уже существующих примитивов, можно «дедуктивным методом» прийти к выводу – все отличия только в разрядности шин, ну и еще в имени модели – это обязательное условие. Пробуем применить на практике. Берем любую из существующих, например **SPISLAVE_8**, втаскиваем в проект и … молотком (**Decompose**). Получился набор «Сделай сам» (Рис. 133).

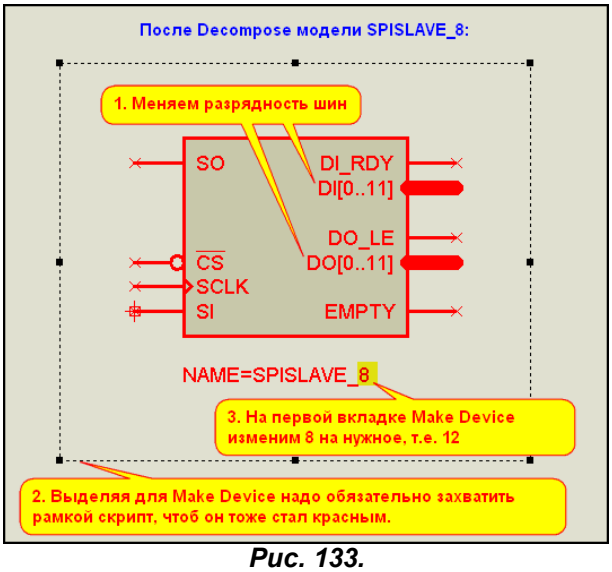


Рис. 133.

С этим набором поступаем следующим образом: меняем разрядность шин на нужную (для 12 - это с 0..11); выделяем, удерживая нажатой левую кнопку мыши, всю графику и в том числе текстовый скрипт, начинающийся с **NAME=**; нажимаем в меню **Make Device** и на первой вкладке меняем в графе **Device Name** имя **SPISLAVE_8** на **SPISLAVE_12**. После проходим процедуру создания модели до конца и сохраняем ее в библиотеке моделей, можно даже в **USRDVC**, так как нужна она нам будет временно, только для создания модели. Аналогично можно создать и 13-ти и 14-ти разрядный **SPISLAVE**. Как протестировать получившуюся модель я уже показал в **п.6.15**. Кстати, такие метаморфозы можно проделывать не только со **SPIO.DLL**, но и с некоторыми другими программными моделями, например, с теми же **ADC** и **DAC**. В этом мы убедимся чуть позже.

Ну а теперь у нас в распоряжении полный набор примитивов для воссоздания структуры модели АЦП. Процесс воссоздания ничуть не отличается от того, что мы делали раньше. Набираем в соответствии с разделом **PARTLIST** файла MDF нужные примитивы и соединяем их между собой в соответствии с **NETLIST**. Поскольку в данном случае MDF достаточно объемный, **NETLIST** содержит 53 цепи, процедура воссоздания длительная и требует внимательности. Могу порекомендовать простой способ, которым я пользуюсь в таких случаях. Текстовое содержание MDF копируем в MS WORD или любой другой редактор, поддерживающий расцветку текста. Сразу же определяемся с тем, что разводить не надо и подсвечиваем каким либо цветом. В данном случае это будут цепи, содержащие только один вывод примитивов. Такие цепи я подсветил зеленым, чтобы в процессе восстановления схемы не обращать на них внимание. Остальные цепи, по мере их прорисовывания, я постепенно подсвечиваю красным. Это удобно еще и тем, что если Вас оторвали от этого занятия, на определенном этапе можно все сохранить и продолжить в другом месте и в другое время. Впрочем, это уже из серии «бесполезных советов». Вернемся к структуре **MAX1241**. Восстановленная структура находится в проекте вложения **MAX1241_Part2\Structures\Structure_MAX1241.DSN**. Там же лежат оригинальный **MAX1241.MDF** и «расцвеченный» **MAX1241_MDF.DOC** (по которому она восстанавливалась). Я не стал помещать на лист только скрипт:

```
*MODELS
MAX1241 : RHI=100,RLO=10,VUD=2,VTL=0.8,VHL=0.2,VTH=2.5,VHH=0.2,V+=VDD,V-=GND
```

Он при компиляции превращается в раздел ***MODELDEFS**. Пока он нам не нужен, но при создании нового «рабочего» MDF понадобится. Результат моего творчества представлен на Рис. 134. Пришлось поместить его полностью и с достаточно большим разрешением, поскольку схема требует некоторых комментариев. Итак, что тут к чему относится и для чего служит. Приемы моделирования пригодятся вам в дальнейшем.

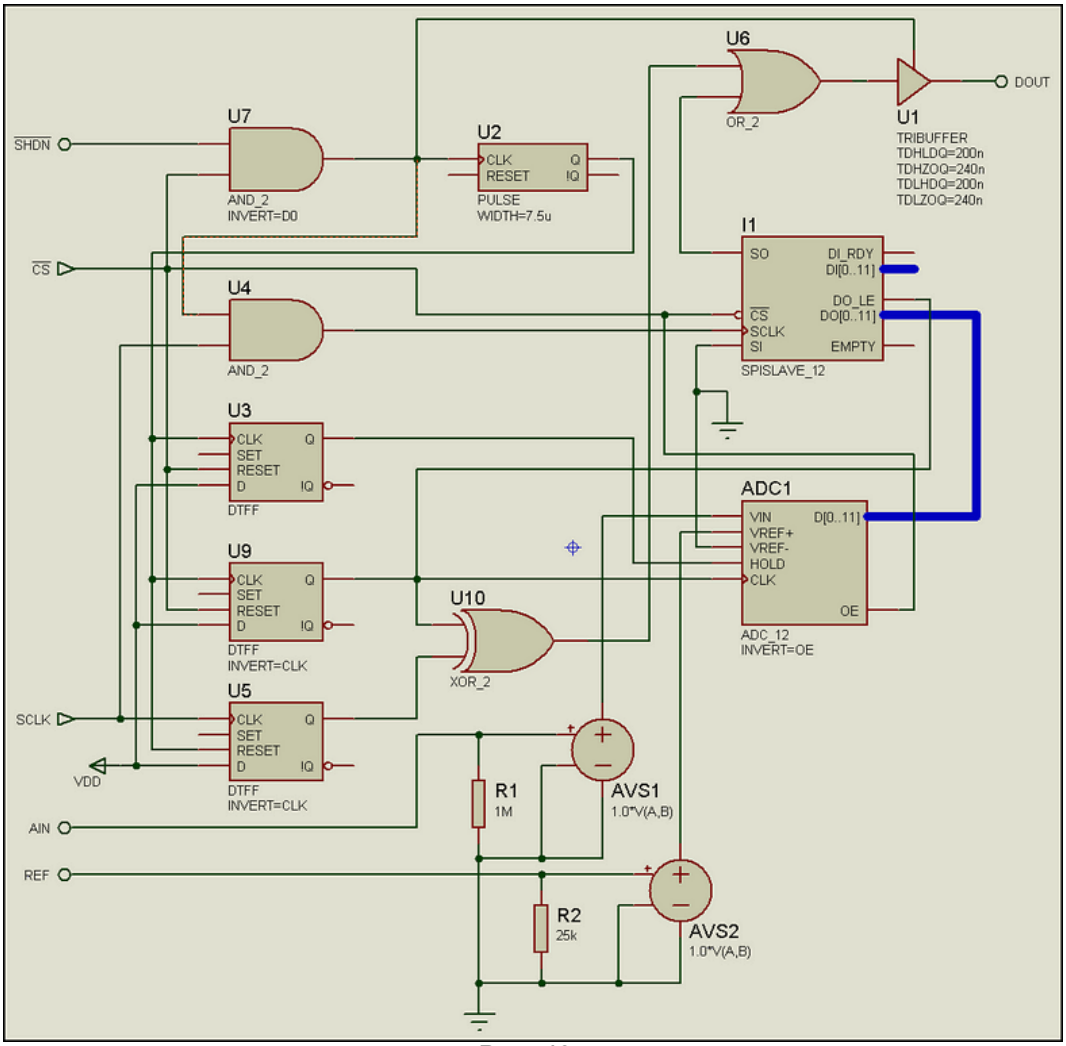


Рис. 134.

Начнем с аналоговой части. Для гальванического разделения по входам **AIN** и **REF** применены примитивы AVCVS **AVS1** и **AVS2** с коэффициентом передачи единица. На их входах помещены соответствующие резисторы, обеспечивающие имитацию входных сопротивлений по этим входам. С выходов **+** AVCVS аналоговые сигналы поступают на аналоговые входы двенадцатиразрядного АЦП **ADC1**, вход **VREF-** которого заземлен. На этом аналоговая часть кончается.

Сигнал с выходной шины **D[0..11]** элемента **ADC1** напрямую соединен с шиной **DO[0..11]** двенадцатиразрядного (**!!!**) примитива **I1**. Такое присоединение (без дополнительных меток и т.п.) ISIS интерпретирует как – «разряд в разряд». Т.е. **D0 ADC1** соединен с **DO0 I1**, **D1** с **DO1** и т.д. до **D11**–**DO11**. Это очень важно, и этим мы воспользуемся для исправления.

Далее сигнал с выхода **SO** SPISLAVE_12 в последовательном коде через элемент **U6** и буфер с тремя состояниями выхода **U1** отправляется на выход модели **DOUT.** Таков путь преобразования входного аналогового сигнала в цифровую форму в модели **MAX1241**.

Теперь о назначении вспомогательных узлов и элементов. Элемент **U7** обеспечивает запуск одновибратора **U2**, формирующего задержку 7,5 мксек при появлении на входах **SHDN** и **CS** разрешающих сигналов. Эта задержка через элемент **U4** блокирует прохождение тактового сигнала **SCLK** на вход **I1**, имитируя стандартную задержку на процесс преобразования **tconv** (см. рис. 128).

Триггер **U3** обеспечивает удержание преобразованного сигнала в цифровой форме на выходе **ADC1** в процессе одного цикла обращения к микросхеме (активность **CS**).

На триггерах **U9**, **U5** и элементе **U10** собран формирователь единичного импульса на время окончания преобразования и первый тактовый импульс (**!!!**) на выходе **DOUT**. Вот здесь и «зарыта» ошибка разработчика модели.

Я скомпоновал структуру **MAX1241** в виде модуля и разместил в стандартный проект CodeVision, который мы рассматривали ранее. Этот пример во вложении:

MAX1241_Part2\Bad_Test_Module1241\TEST_Structure_12razr.DSN

Растянутый график второго цикла преобразования (первый дает нулевой результат) представлен на рисунке 135. Обратите внимание на широкий единичный импульс – сигнал окончания преобразования на выходе **DOUT**. Он перекрывает первый тактовый импульс **SCLK**. Но первый же тактовый импульс проходит и на тактовый вход **SPISLAVE_12** – трасса **U4(Q).** Поскольку у нас модель **I1** двенадцатиразрядная, этот импульс соответствует старшему биту MSB (**D11**) интерфейса. Как говорят врачи реаниматоры: «мы теряем его…». Это нетрудно доказать. Если подать на вход **AIN** половинное напряжение от **REF** должен оказаться заполненным единицей старший разряд **D11** АЦП, т.е. двоичный код выглядит так: **1000 0000 0000**. Но на деле, в тестовом

проекте, упомянутом выше, на терминал будет выводиться нулевое напряжение – реаниматоры оказались правы.

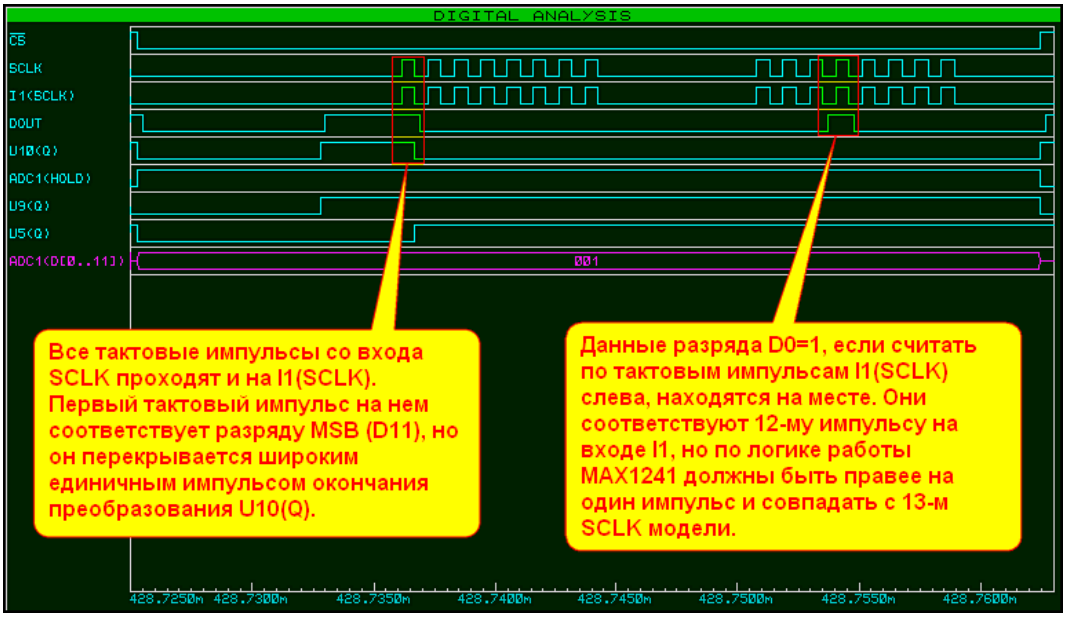


Рис. 135.

Ну что же, причина ясна. Пора приступать к хирургическому вмешательству. И тут помогает простейшая аптечная «свинцовая примочка». Помните, чуть выше я упоминал, что SPISLAVE может быть и 13-ти разрядным, а при описании структуры указал, что данные по шине передаются «разряд в разряд». Конечно, если с 12-ти разрядного АЦП передать данные в 13-ти разрядный SPI, то старший 13-й разряд последнего окажется незаполненным. Но он нам особенно и не нужен, его благополучно «скушает» импульс окончания преобразования, как он проделывал это с 12-м разрядом ранее. И, как в старом бородатом анекдоте, «пусть хомяк подавится». Проверяем на деле. В проекте вложения:
Good_Test_Module1241\TEST_Structure_13razr.DSN
представлена данная замена. Надеюсь, повторяться о том, как сделать **SPISLAVE_13** не надо. Тестируем проект и убеждаемся, что все встало на свои места (Рис. 136).

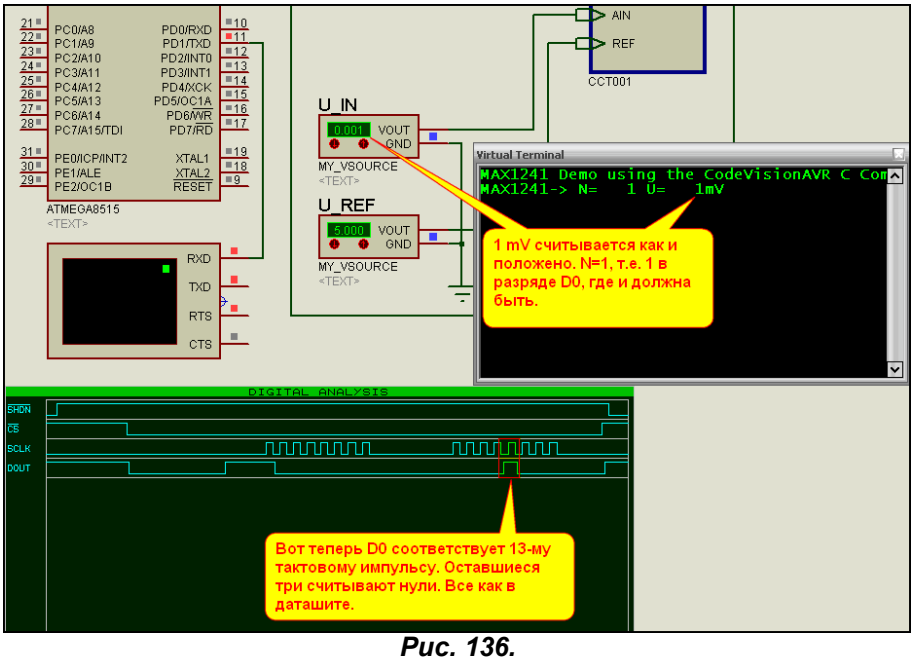


Рис. 136.

Теперь осталось восстановить «статус-кво» в самом Протеусе. Создаем проект с **MAX1241**, привязываем к МАХу дочерний лист (в свойствах ставим галочку **Attach Hierarchy Module**), устанавливаем дополнительно галочку **Edit all properties as text** и временно удаляем строчку:
{MODFILE=MAX1241.MDF}
Процесс представлен на рисунке 137.

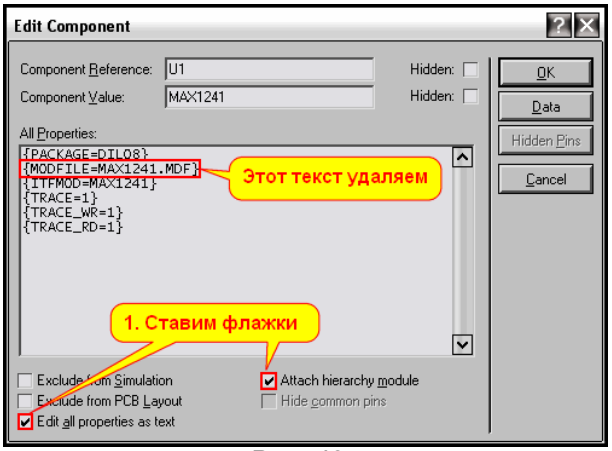


Рис. 137.

На дочернем листе располагаем восстановленную структуру (Рис.134), скопированную с дочернего листа **TEST_Structure_13razr.DSN**. Там у нас уже 13-ти разрядный SPI. Не забудьте туда же поместить тестовый скрипт ***MODELS**, который приведен ранее.

Есть еще один нюанс, который прояснился в последний момент. Вероятно, модели **MAX1240/MAX1241** разрабатывались не сотрудниками Лабцентра, а сторонним пользователем. Дело в том, что «шапка» MDF полностью заполнена, а в графе **Author** стоит **EA**. Схематичные модели, разработанные в самом Лабцентре, как правило, в шапке имеют только дату. Так вот этот самый **EA** в графической модели обозначил инвертированные выводы знаком доллара **$** не с двух сторон, как я привык и объяснял где-то вначале FAQ, а только спереди, т.е. **$SHDN** и **$CS**. Поэтому на дочернем листе одноименные терминалы надо привести в соответствие, иначе полезут ошибки. Перед компиляцией нового MDF модель можно протестировать. Вариант с дочерним листом во вложении **New_Model_MAX1241\With_Child\ 1241_Wtith_Child.DSN**.

После того, как мы убедились, что все работает и при подаче на вход напряжения 1mV импульс разряда **D0** в графике встал, где положено, т.е. 13-м по счету с дочернего листа через меню **Tools=>Model Compiler** компилируем новый **MAX1241.MDF**. Тест с новым MDF во вложении **New_Model_MAX1241\With_New_MDF\ 1241BIN_new.DSN**.

Далее можно пойти тремя путями:

- Первый и самый простой вариант. Новый файл MDF переименовываем, например, в **MAX1241N.MDF** и помещаем в папку **MODELS** Протеуса. Предварительно сняв защиту от записи с помощью **Library=>Library Manager** (Рис. 138), помещаем в проект **MAX1241** и запускаем для него **Make Device**. На третьей вкладке для **MODFILE** в графе **Default Value** задаем наш новый MDF и в последней вкладке сохраняем его не **USRDVC**, а в **MAXIM**, который теперь доступен для записи. После этого можно снова защитить библиотеку, повторив процедуру через **Library Manager**.

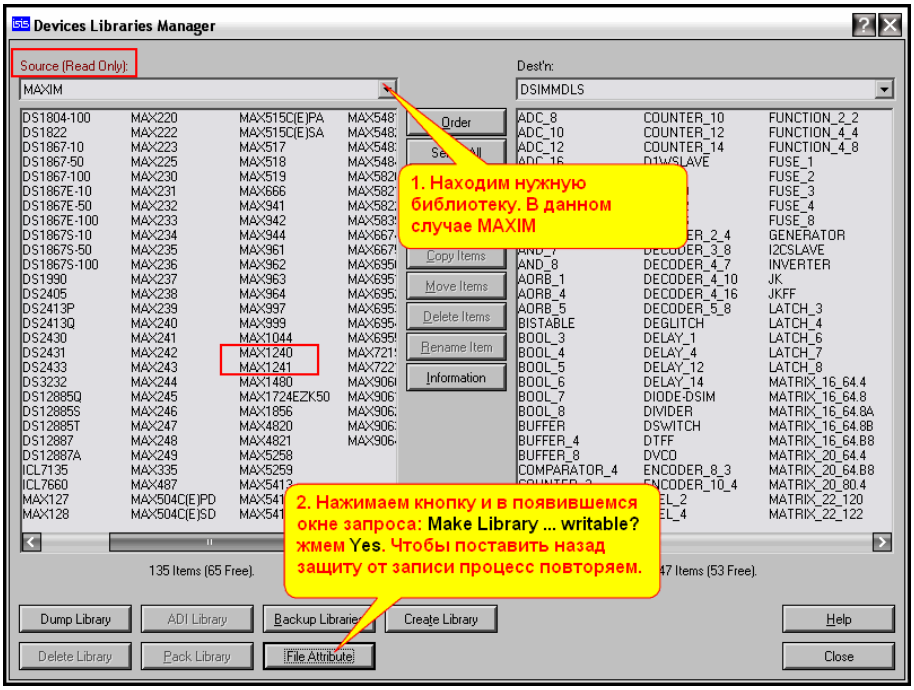


Рис. 138.

- Второй вариант – более сложный, но корректный с точки зрения Протеуса. Файл **MAXIM.LML** (не забудьте сохранить резервную копию на всякий пожарный случай!) из **MODELS** Протеуса копируем в отдельную папку и туда же помещаем утилиты **GETMDF.EXE** и **PUTMDF.EXE** из папки **BIN**. Запускаем командную консоль для этой папки и выполняем:
  GETMDF.EXE –L=MAXIM.LML –D MAX1241
  При этом ключом **–D** (delete) модель стирается из библиотеки и дополнительно извлекается в нашу папку в виде файла **MAX1241.MDF**. Его нужно удалить – это старый вариант. Правда, термин «стирается» – это слишком громко сказано. На самом деле, стирается только в бинарном заголовке файла LML. Но, если запустить поиск по ключу MAX1241, то сам текст старого MDF в библиотеке LML мы обнаружим. Впрочем, нам он не мешает, пусть живет. Затем помещаем новый файл (я приложу их в папке **New_MDF_MAX1240_MAX1241** вложения) в нашу папку и выполняем в командной консоли следующую процедуру:
  PUTMDF.EXE –L=MAXIM.LML MAX1241.MDF
  Наша новая MDF модель приплюсуется в конце библиотеки, а в заголовке LML появится ссылка уже на нее. Теперь можно **MAXIM.LML** вернуть на старое место в **MODELS**. Протеус будет работать уже с новой моделью. Корректное выполнение процедуры показано на рисунке 139.

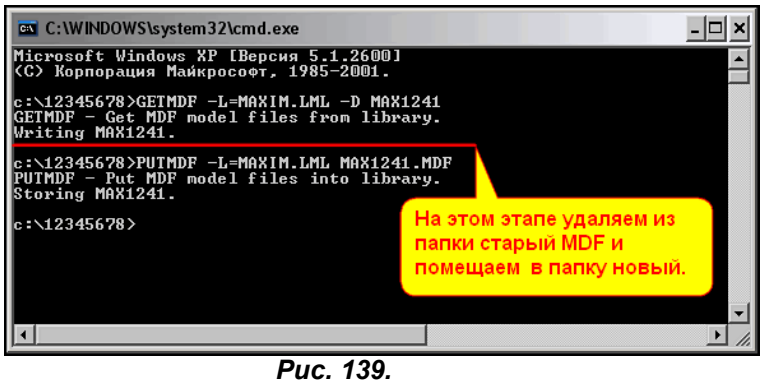


Рис. 139.

- Наконец, третий вариант – самый трудоемкий. Полная перекомпиляция библиотеки **MAXIM.LML** с заменой моделей **MAX1241** и **MAX1240** (о ней ниже) на корректные новые. Для этого придется активно поработать с утилитами командной строки, поскольку библиотека содержит 84 модели. Процесс схож с предыдущим вариантом и заключается в следующем. Сначала с помощью **GETMDF** извлекаются все MDF из библиотеки командой:
  GETMDF –L=MAXIM.LML –A
  Затем убираем из папки саму библиотеку, заменяем **MAX1241.MDF** и **MAX1240.MDF** новыми и создаем новую библиотеку **MAXIM.LML**, например, на 100 элементов командой:
  PUTMDF –L=MAXIM.LML –C=100
  Далее через **PUTMDF** постепенно (порциями по несколько штук, чтоб не запутаться) добавляем в нее все присутствующие в папке MDF. При этой операции удобно пользоваться ключом **–D**, который при добавлении моделей в библиотеку будет одновременно и удалять их MDF из папки. Допустим, мы добавляем в библиотеку четыре модели: **DG417.MDF**, **DG418.MDF**, **DG419.MDF** и **DG508.MDF**. Тогда строка будет выглядеть так:
  PUTMDF –L=MAXIM.LML –D DG417.MDF DG418.MDF DG419.MDF DG508.MDF
  На рисунке 140 показан процесс создания **MAXIM.LML** на 100 «посадочных мест» и процесс добавления моделей по 4 штуки в строке. Напомню, что в окне командной консоли можно повторно выбирать предыдущие выполненные команды с помощью клавиши навигации «стрелка вверх». Просто вызываем каждый раз предыдущую команду и в ней перебиваем список добавляемых моделей, после чего давим **Enter**. Ну и еще, как вы поняли, и я упоминал ранее – расширения файлов можно не набирать, достаточно только имен.

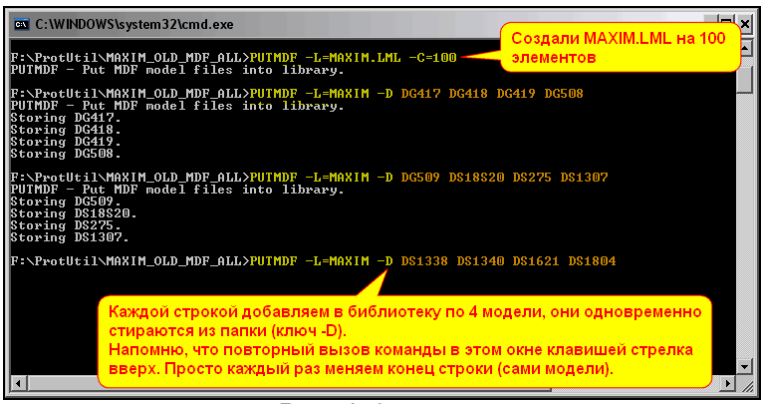


Рис. 140.

Теперь немного о **MAX1240**. Эта модель имеет схожую структуру, но отличается тем, что в ней используется внутренний источник опорного напряжения 2,5V, который подключается при условии, что на входе **SHDN** присутствует высокий уровень, либо он не подключен. Естественно, в модели присутствует и та же ошибка, что в **MAX1241**. Восстановленная с MDF структура **MAX1240** находится во вложении в файле **Structures\Structure_MAX1240.DSN**. Подробно я останавливаться на этой модели не стану, только добавлю, что в папке **Good_Test_Module1240** находится тест с уже 13-ти разрядным SPISLAVE (соответственно изменена программа CV под 2,5V), а в папке **New_Model_MAX1240\With_Child** проект, с дочернего листа которого скомпилирована новая модель. Сам новый **MAX1240.MDF** лежит в папке **New_MDF_MAX1240_MAX1241**.
В папке **New_LML_lib** лежат полностью пересобранные по третьему способу **MAXIM.LML** для версий 7.6 и 7.7 с исправленными моделями **MAX1240** и **MAX1241**.

Возврат к содержанию

6.18. Создаем модель АЦП ADS1286 от Burr-Brown, или LTC1286 от Linear Technology.

Разбираться в чужих моделях и искать ошибки дело конечно нужное, но иногда хочется и творческого полета собственной мысли. Так вот, разборка с **MAX1241** натолкнула меня на идею сваять другую, не менее популярную модель 12-ти разрядного АЦП **ADS1286**. Тем более что он полностью совместим с **LTC1286**, так что «пристрелим двух ушастых» одним выстрелом. Это послужит некоторым закреплением пройденного материала, ну и лишний АЦП в хозяйстве пригодится. Анализ протокола обмена из даташита на **ADS1286** показал, что он незначительно отличается от **MAX1241**, т.е. часть структуры последнего можно «принять за основу», как модно было выражаться на различных собраниях в эпоху недоразвитого социализма. На рисунке 141 приведена диаграмма обмена по последовательному интерфейсу.

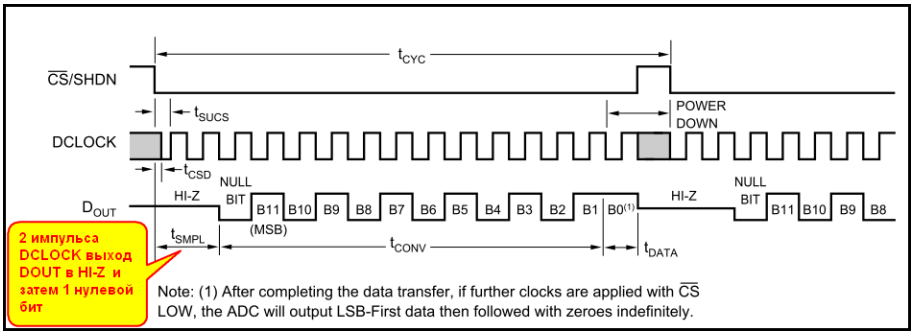


Рис. 141.

Рассмотрим основные отличия от **MAX1241**. После появления низкого уровня на входе выбора кристалла **CS** два тактовых импульса (время $\mathbf{t_{SMPL}}$) выход **Dout** находится в высокоимпедансном состоянии. Затем следует один нулевой бит и далее 12 бит оцифрованного сигнала, где, как и у **MAX1241** MSB следует первым. Это и будет для нас основным фактором отличия. По большому счету вторая диаграмма (**FIGURE 1** даташита) показывает, что если **CS** и далее будет оставаться низким, и будут следовать тактовые импульсы, то возможно считывание данных в обратном порядке, пока АЦП не «заснет», но та же оговорка в сноске предупреждает, что там могут быть, и считаны и нули. Нам это не так важно в нашей модели. Поскольку играться с всякими неопределенностями себе дороже, практической ценности то, что будет правее бита **B0,** для нас не представляет, и эту информацию лучше не использовать. Структура АЦП приведена на Рис. 142.

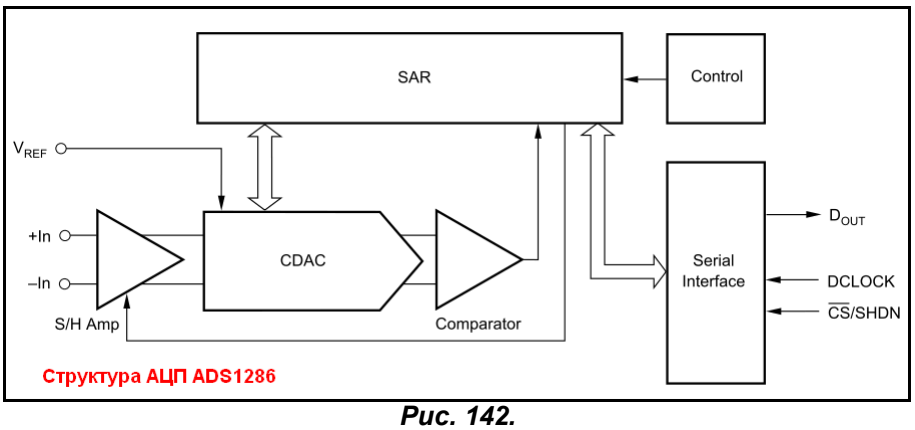


Рис. 142.

Давайте попробуем спроектировать структуру нашей модели. Связку **ADC_12** и **SPISLAVE_13** пока оставим в неприкосновенности. Останется и **TRIBUFFER**, так как нам необходим выход **Dout** с тремя состояниями. А вот всю остальную входную логику, формирующую управление будем менять. Нам необходимо отсечь первые два тактовых импульса, которые не должны изменять состояние

выхода АЦП и уж тем более влиять на сдвиг данных в последовательном интерфейсе. В этом поможет обычный счетчик на двух D-триггерах. Потребуется и несколько логических элементов совпадения по И. А вот с тем стартовым нулевым битом можно применить трюк, который мы использовали в **MAX1241**. Вспомните, модель **ADC** – 12-ти битная, а **SPISLAVE** – 13-ти. При этом старший, незадействованный 13-й бит будет читаться нулем, если мы специально не закинем в него единицу. Вот это нам и надо. Именно поэтому в данном случае оставляем модель **SPISLAVE_13**. Кроме того, подвергнется небольшой переделке и та часть модели, которая относится к аналоговым входам. Мы видим, что V_{REF} в данном случае подается относительно земли, зато входной сигнал **+In** и **–In** является дифференциальным. На рисунке 143 приведена схема получившейся у меня модели.

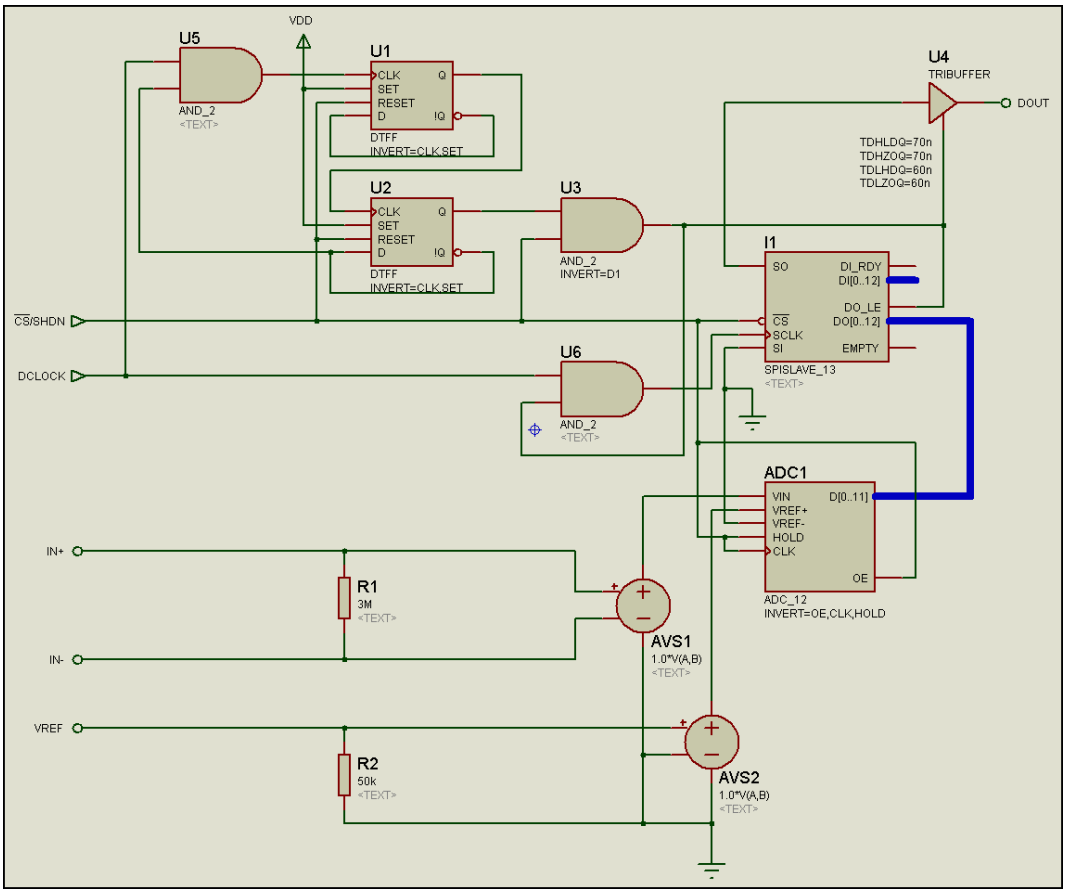


Рис. 143.

Рассмотрим ее несколько подробнее. Появление низкого уровня на входе **CS/SHDN** по входам CLK, HOLD и OE «запирает» результат аналого-цифрового преобразования на выходной шине **ADC1**. На триггерах **U1** и **U2** собран счетчик, который отсекает первые два тактовых импульса. Установка их идет по заднему фронту (в свойствах **INVERT=CLK**). После установки триггера **U2** через элемент 2И **U5** дальнейший счет блокируется до следующего появления на входе **CS/SHDN** высокого уровня, который сбросит триггера счетчика. Элементом 2И **U3** при этом активируется выходной буфер **U4** (снимается третье состояние), а через элемент **U6** разрешается прохождение тактовых импульсов на вход SCLK интерфейса **I1**. Поскольку интерфейс 13-ти разрядный, первым проследует старший разряд, в который ничего не заносилось – получится требуемый нулевой стартовый бит, а затем 12 разрядов с выходной шины **ADC1** в порядке MSB - первый. Как видите, с этой точки зрения структура получилась даже проще, чем у **MAX1241**. В аналоговой части по-прежнему стоят два гальванических разделителя **AVS1** и **AVS2**. Только теперь у **AVS2** минусовой вход заземлен (VREF подается относительно GND). Входное сопротивление по этому входу имитируется R2=50кОм. Переход в высокоимпедансное состояние при неактивном («спящем») АЦП я не стал имитировать, т.к. это неоправданно усложнило бы модель. Входы же аналогового сигнала **+In** и **–In** связаны через резистор R1=3МОм. Он рассчитан, исходя из входного тока около 1,5мкА при 5В (см. даташит раздел: **RC INPUT FILTERING**).

Теперь немного о том, что во вложении. В папке **Test_structure_module** приложены два варианта тестирования полученной структуры. С помощью обычных генераторов – **Generators** и проект с CodeVision в одноименной папке. В последнем проекте для того, чтобы считанное **N** совпадало с расчетным принято директивой **#define VREF 4096** в начале файла на Си и для вывода применен LCD 2x16. В папке **Test_with_child** уже графическая модель **ADS1286**, но с дочерним листом – с него потом скомпилирован **ADS1286.MDF**. Ну, и наконец, в папке **Test_with_MDF** тестовый проект с готовым MDF, он лежит в этой же папке.

Для использования в собственных проектах достаточно переложить **ADS1286.MDF** в папку **MODELS** Протеуса, а из проекта **Test_ADS1286_MDF.DSN** запустить и пройти до конца **Make Device** для модели **ADS1286**. На последней вкладке определитесь с библиотекой, в которой она будет

храниться. Аналогично можно сделать и модель **LTC1286** (не путайте с **LTC1298**, даташит у них единый, но они отличаются), просто переименовав на первой вкладке имя модели, а на последней изменить производителя. Поскольку они по логике работы полностью совпадают, файл MDF для них используется один и тот же. Ну и наконец, в папке **DATASHEETS** вложения даташиты на **ADS1286** и **LTC1286**, которыми я руководствовался при создании модели.

Возврат к содержанию

Заключение к части III

Материала в этой части получилось достаточно много, но надеюсь, что он вызвал определенный интерес не только у новичков в освоении Протеуса, но и у давних пользователей этого программного продукта. Я сознательно не стал углубляться в подробное изложение основ SPICE-моделирования, поскольку этой теме посвящено достаточно много литературы других авторов. Но применение SPICE-моделей, заимствованных из других программ и у производителей компонентов было рассмотрено выше. Не знаю, насколько мне это удалось, но главное, чего я добивался при написании этой части, чтобы пользователи освоили создание пусть примитивных, но своих собственных моделей с помощью применения подсхем, а также больше внимания уделяли свойствам моделей. Конечно, материал не претендует на стопроцентное рассмотрение всех особенностей Протеуса, но по мере изложения я стараюсь в примерах вводить инструментарий и приемы, которых не касался в изложенном материале. Вдумчивый пользователь самостоятельно сможет использовать это в своих разработках, ну а кто привык «с шашкой наголо»… извините. Заранее предупреждаю, что «секретной кнопочки» - нажал, и все само заработало, здесь нет. Моделирование – процесс творческий, и порой времени на то, чтобы на экране компьютера все заработало так, как и в реальном устройстве тратится больше, чем для отладки реального устройства. Причем, если там иногда достаточно только познаний в области электроники и не совсем кривых рук, то при компьютерном моделировании надо быть еще и программистом по призванию и в совершенстве знать особенности поведения конкретной программы в том или ином случае, а также грамотно оперировать свойствами моделей. Именно это я и стремился довести в вышеизложенном материале. В следующей части мы рассмотрим активные модели – наиболее интересную «изюминку» Протеуса, а также немного поучимся программировать модели на встроенном языке EASYHDL. Но, заранее предупреждаю, без навыков создания графических моделей, которые я неоднократным повторением в этой части старался «вдолбить» до уровня подсознательных действий, освоение следующего материала фактически невозможно.