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

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

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

С уважением, Halex07.

1. Краткие общие сведения о программном продукте PROTEUS.

1.1. Протеус – что это такое?

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

1.2. Сайт автора программы.

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

1.3. В чем отличие от других подобных программ.

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

1.4. Системные требования. Различия в версиях программы.

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

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

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

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

2. Установка и запуск Proteus. Интерфейс программы ISIS.

2.1. Где взять инсталляционный пакет Протеус.

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

2.2. Установка программы на компьютер.

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

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

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

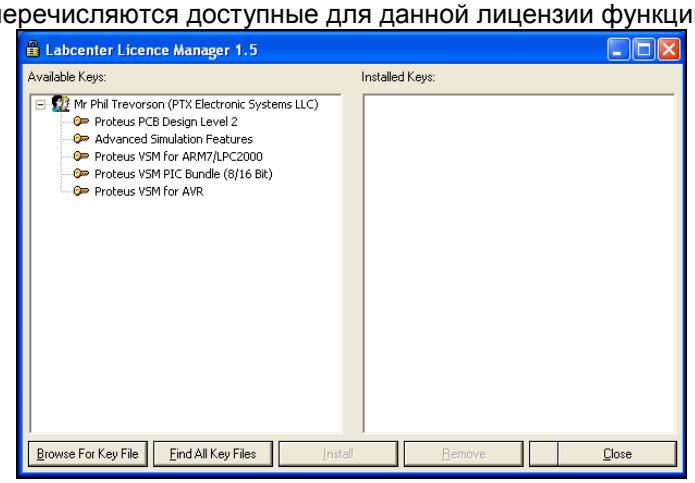

Рис.1

2.3. Первый запуск и первые проблемы.

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

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

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

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

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

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

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

Есть два пути решения этой проблемы:

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

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

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

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

2.4. Интерфейс программы ISIS.

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

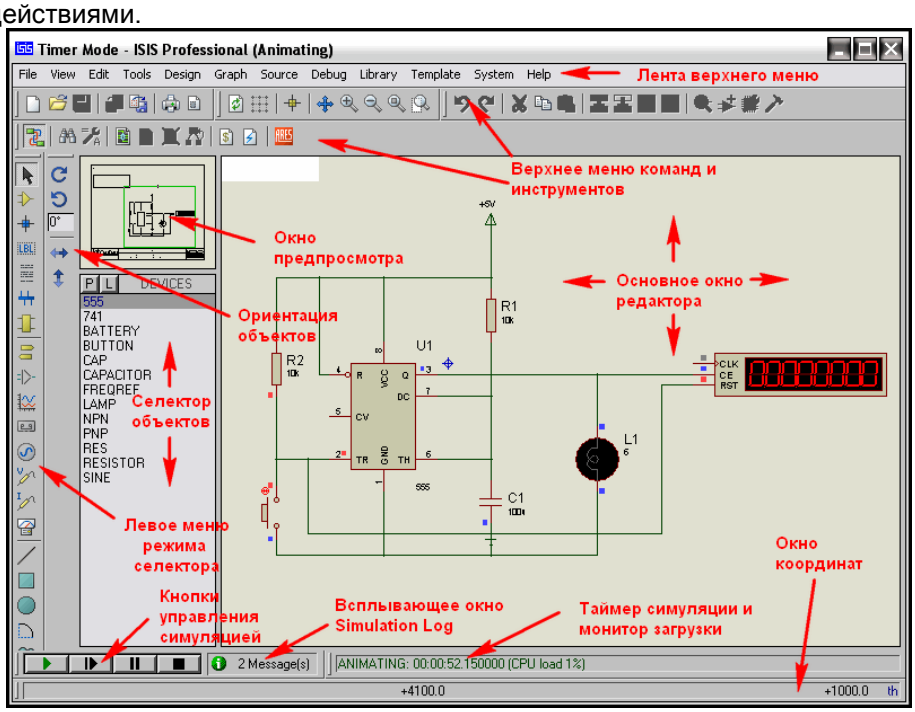

Рис.2

2.5. Папка Samples - кладезь примеров проектов для начинающих.

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

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

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

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

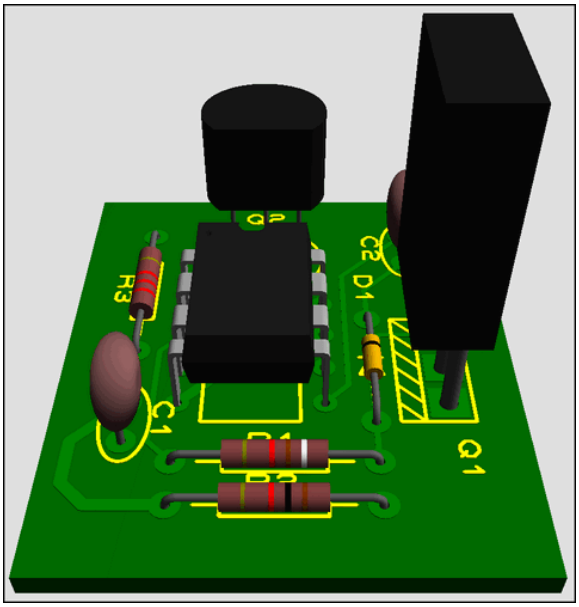

Рис.3

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

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

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

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

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

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

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

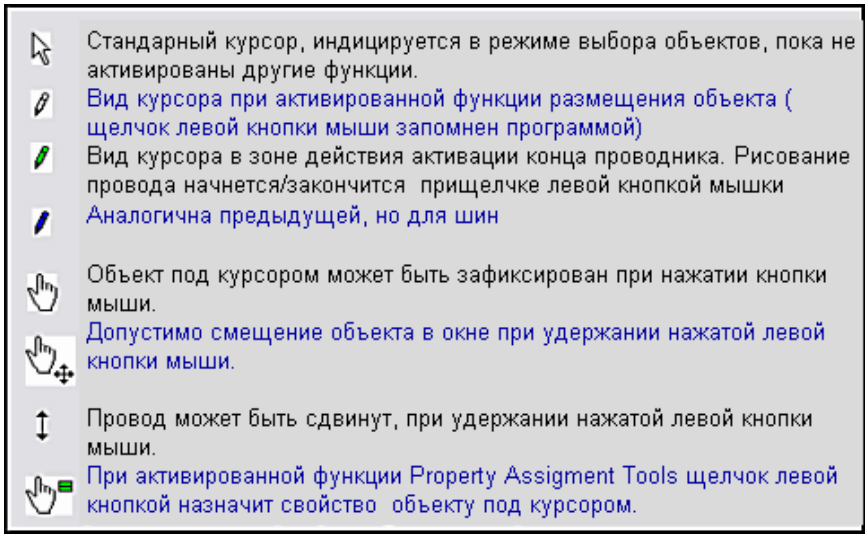

Рис.4

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

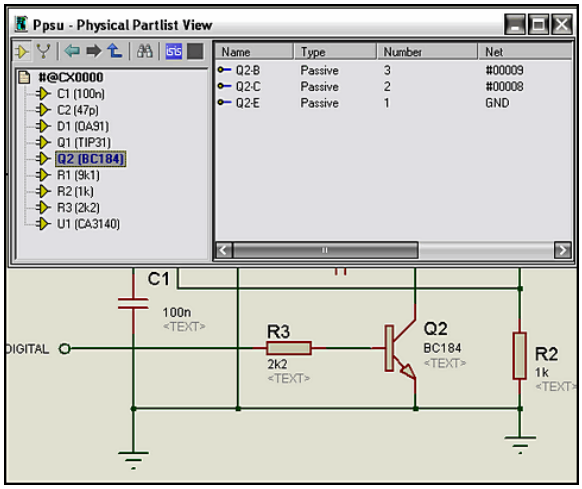

Рис.5

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

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

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

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

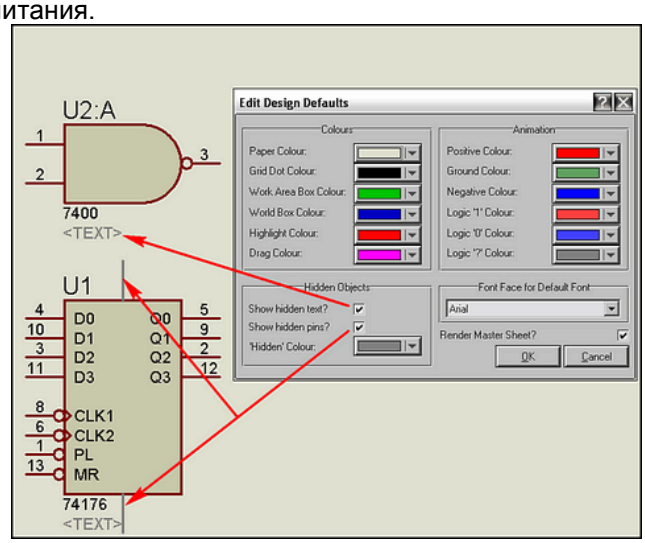

Рис.6

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

Еще одна полезная функция меню **System** – **Set Sheet Size…** (установить размер листа). Типичная ситуация: Вы рисуете свой проект, увлеклись и с ужасом обнаруживаете, что схема не помещается в синих границах листа (по умолчанию А4 альбомный). Через эту функцию вы выбираете больший, например А3. При этом то, что вы уже набросали на схеме, автоматически центрируется в рамках нового размера листа.

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

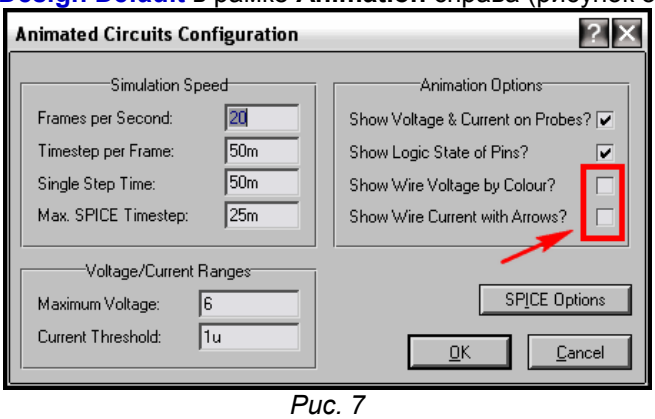

Рис. 7

2.7. Верхние (подключаемые) тулбары.

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

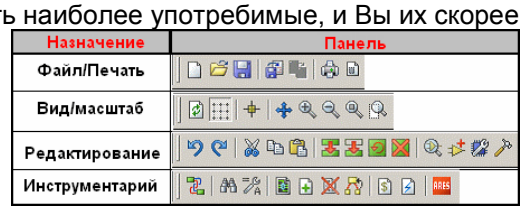

Рис. 8.

Вид/Масштаб (слева – направо):

Redraw Display (**R**- здесь и далее в скобках кнопки клавиатуры) – обновляет окно программы. *Круговые зеленые стрелки*

Toggle Grid (**G**) – включает/выключает показ сетки. *Точечная сетка.*

Toggle False Origin (**O**) – включает/выключает мнимую точку начала координат. Потребуется при создании графических моделей. *Прицел.*

Center At Cursor (**F5**) – центрирует изображение на экране по указателю курсора (щелчок левой кнопкой мышки). *Четыре голубые растягиваюшие стрелки.*

Zoom In (**F6**) – Увеличить масштаб. *Лупа с плюсом.*

Zoom Out (**F7**) – Уменьшить масштаб. *Лупа с минусом.*

Zoom To View Entire Sheet (**F8**) – разместить на экране лист целиком. В меню **View** опция **Zoom All…** *Лупа с мелким квадратом.*

Zoom To Area – разместить на экране выделенный регион (щелчок левой кнопки мыши первая точка, повторный вторая по диагонали выделяемого региона). *Лупа с белым прямоугольником.*

Редактирование (слева – направо):

Undo **(Ctrl+Z)** –отмена последнего действия. По умолчанию допустимо откатить до 20 шагов. Меню **System => Set Environment…** *Голубая стрелка против часовой.*

Redo **(Ctrl+Y**) – возврат последнего действия. Активна только после **Undo**. *Голубая стрелка по часовой.*

Три стандартные кнопки, начиная с ножниц, **Cut, Copy, Paste** - вырезать, копировать в буфер, вставить из буфера в пояснении не нуждаются. Как и четыре последующих кнопки операций с блоками, становятся активными только при выделении элемента или участка схемы левой кнопкой мыши. Однако, в отличие от того же **MS Word**, не имеют стандартных сочетаний клавиатуры типа **Ctrl+X** и т.д.

Block Copy – копировать блок (*два зеленых прямоугольника с вертикальной красной стрелкой вниз*). **Block Move** – переместить блок *(кнопка аналогична предыдущей, только верхний прямоугольник прозрачный).*

Block Rotate – позволяет через всплывающее окно повернуть выделенный блок (элементы 2D графики) на заданный угол или отразить его (галочка **Mirror**) по оси **X** или **Y** *(зеленый прямоугольник с круговой стрелкой против часовой)*. Для этого проще использовать всплывающее меню правой кнопки мыши. Причем в нем автоматически изменяются опции для элемента и блока (участка схемы). Еще один нюанс, поворот на заданный угол доступен только для элементов 2D графики, а например резистор или диод установить можно только горизонтально или вертикально. Через меню правой кнопки отдельный элемент вертится только на 90 градусов. **Block Delete** **(Del)** – стирает выделенный блок/элемент *(зеленый прямоугольник с иксом)*.

Pick Parts From Library – выбрать объект из библиотеки Протеуса *(лупа с мнемоникой ОУ – треугольник внутри)*. В отличие от аналогичной по действию кнопки вверху селектора объектов – **P**, всегда переносит нас в библиотеку электро-радиокомпонентов. Кнопка **P** – изменяет свое действие при выборе режима селектора. Мы это рассмотрим позже.

Make Device – создание нового устройства (компонента) из выделения (*мнемоника ОУ с символом плюс)*. Рассмотрим подробно при создании собственных моделей.

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

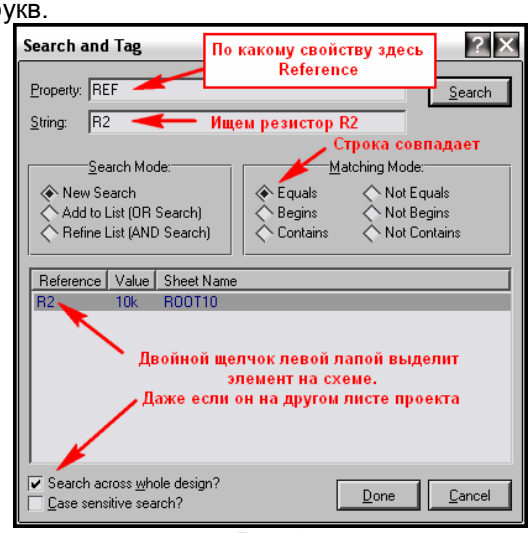

Рис. 9

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

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

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

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

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

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

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

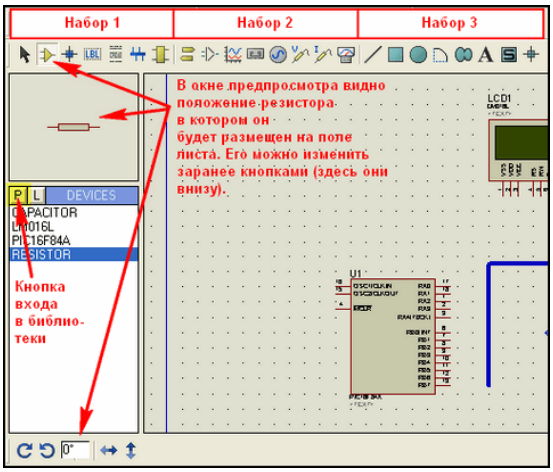

Рис. 10

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

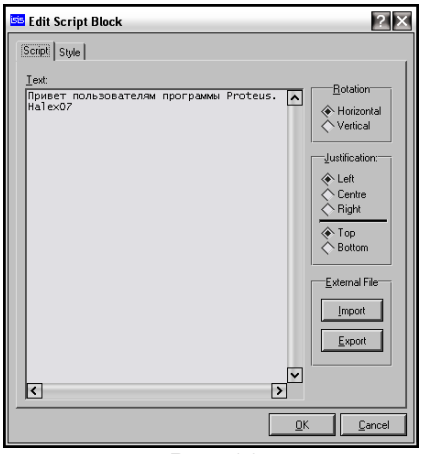

Рис. 11

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

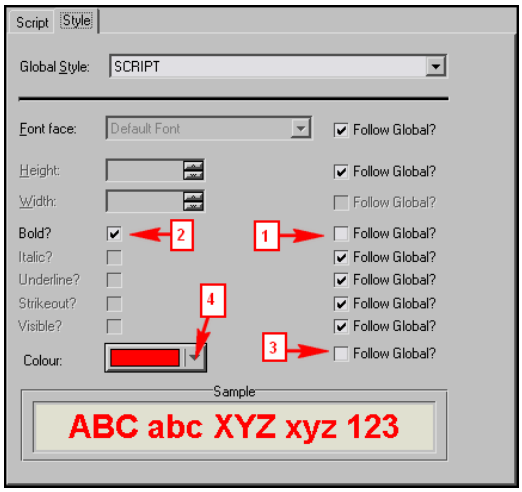

Рис. 12

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

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

Набор 2.

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

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

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

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

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

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

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

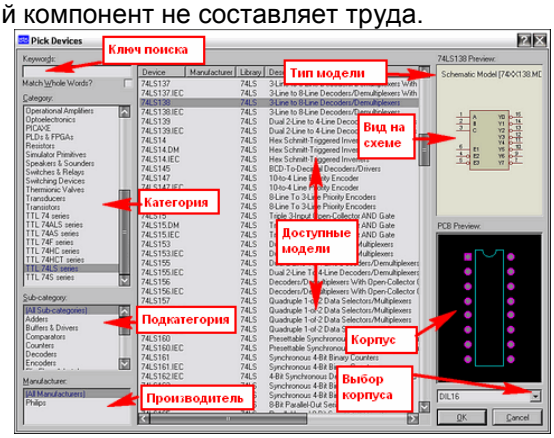

Рис. 13

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

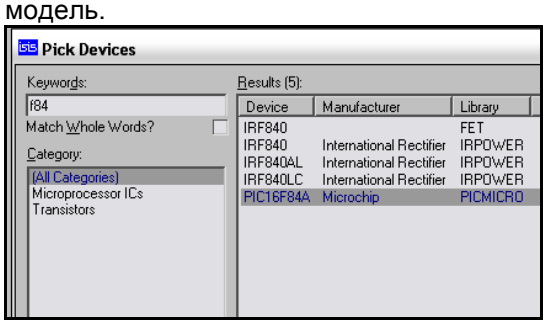

Рис. 14

2.10. Подбираем компоненты, расставляем их в проект.

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

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

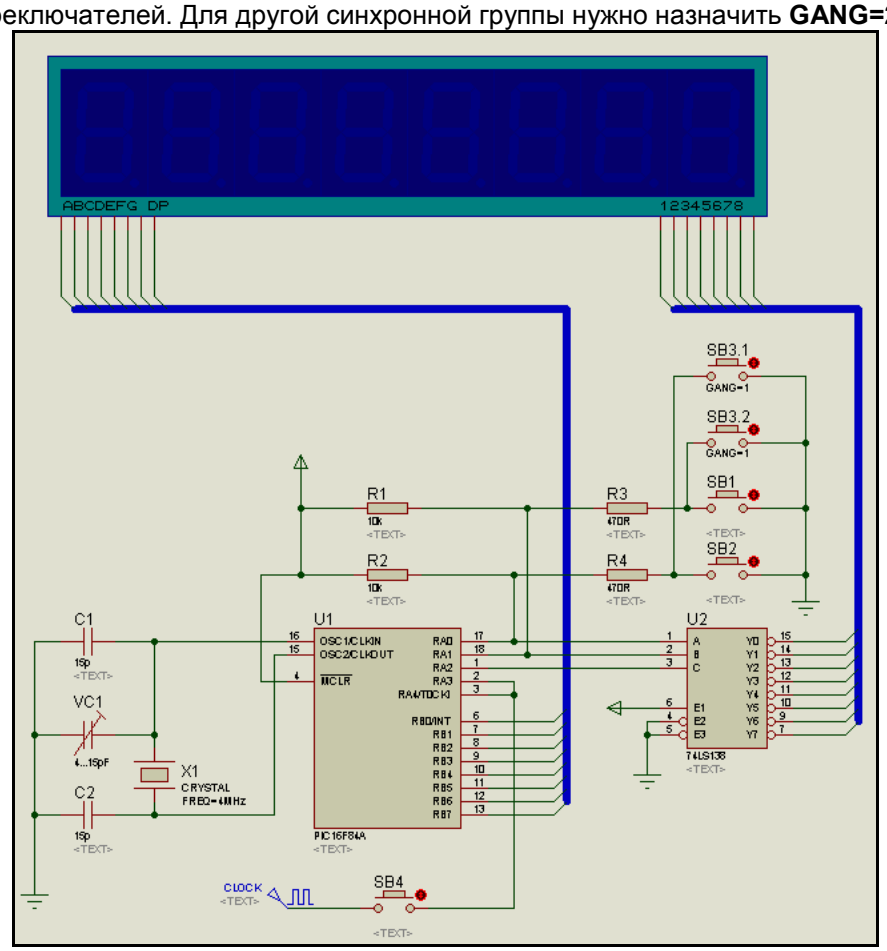

Рис. 15

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

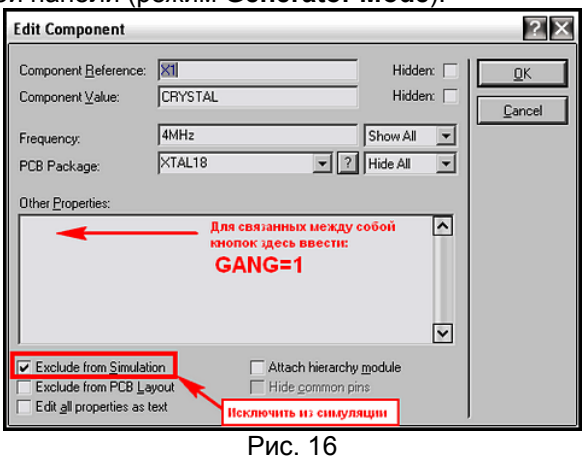

Рис. 16

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

2.11. Приемы быстрого редактирования. Разводка проводов и шин.

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

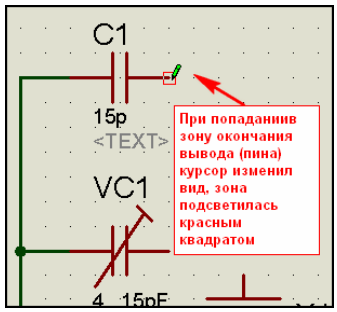

Рис. 17

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

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

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

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

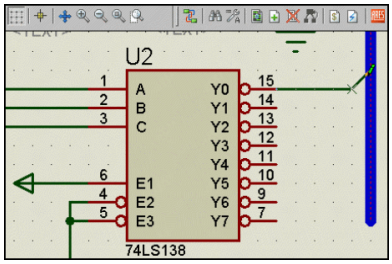

Рис. 18

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

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

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

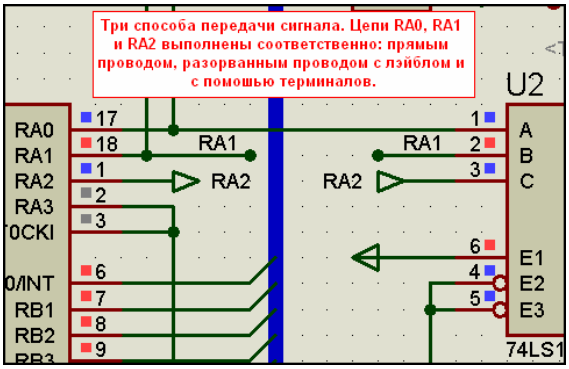

Рис. 19

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

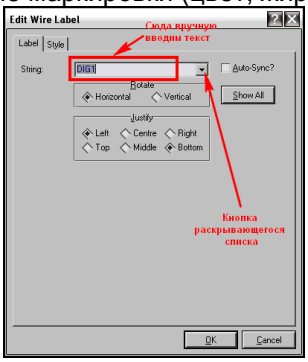

Рис. 20

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

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

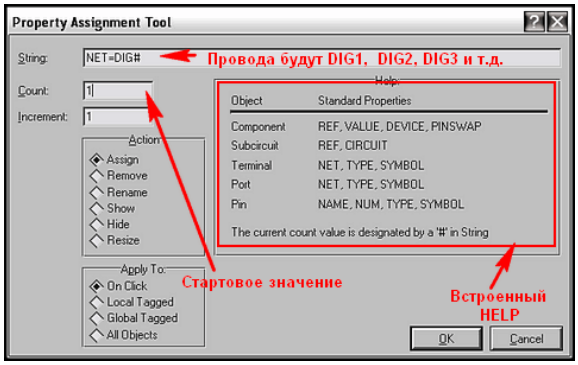

Рис. 21

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

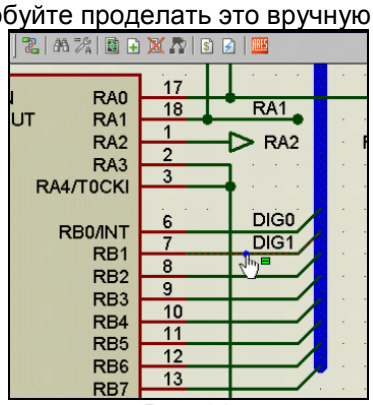

Рис. 22

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

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

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

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

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

Explanation of Warning Messages – разъяснение предупреждающих сообщений симулятора.

Run Time Disassembler – пояснение по использованию встроенного дизассемблера.

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

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

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

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

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

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

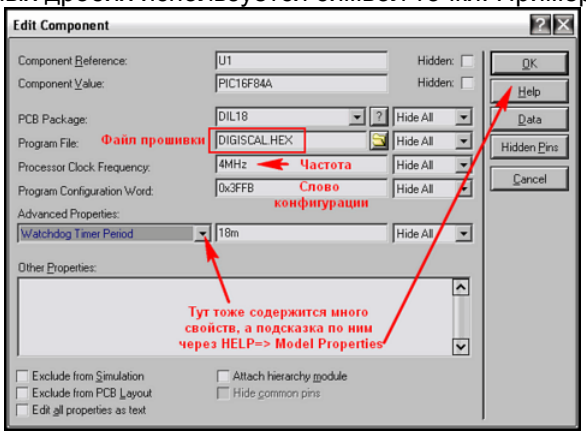

Рис. 23

2.14. «Прошивка» микроконтроллера в ISIS.

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

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

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

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

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

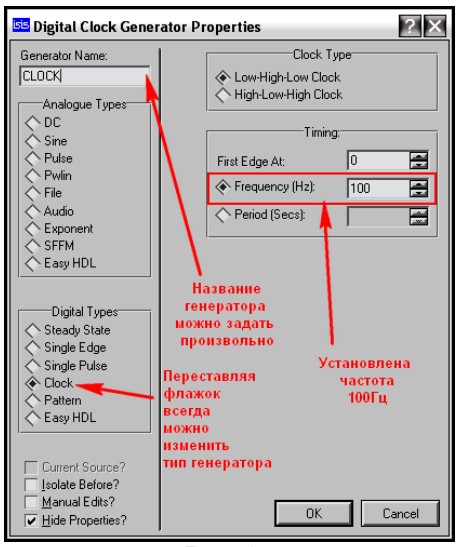

Рис. 24

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

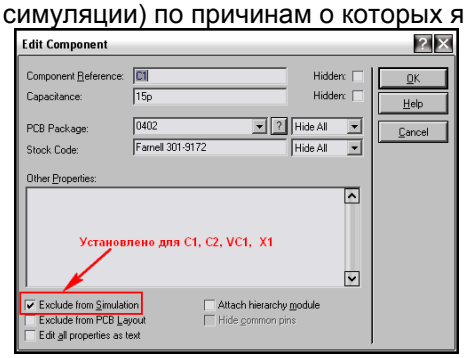

Рис. 25

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

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

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

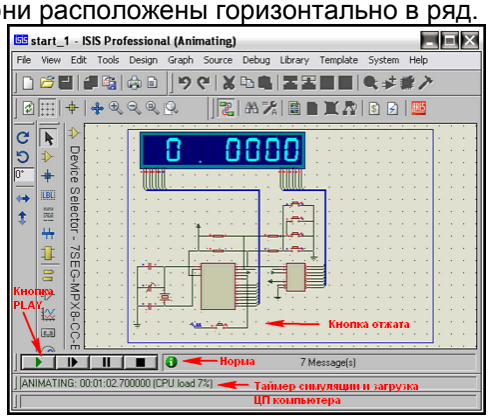

Рис. 26

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

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

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

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

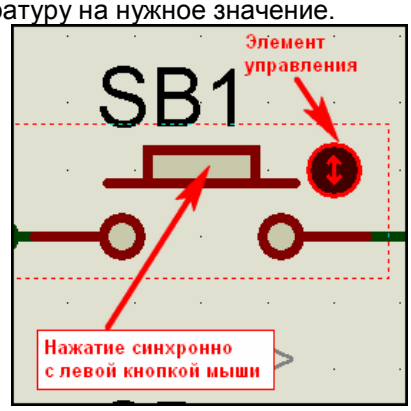

Рис. 27

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

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

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

2.16. Зонды-пробники в Протеусе.

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

В ISIS можно использовать два типа зондов: напряжения **V** и тока **I**. Соответствующий режим выбирается в левом вертикальном меню (Рис.27). На рисунке показан зонд, установленный на шину сегментов индикатора.

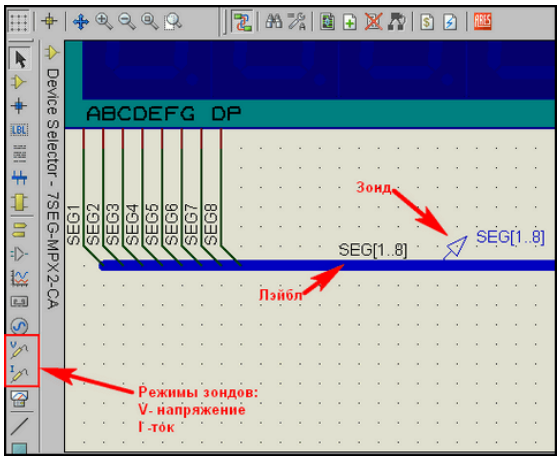

Рис. 28

Прежде чем его установить необходимо присвоить шине маркировку (лэйбл). В данном случае маркировка ставится вручную, т.е. выбираем режим **LBL**, наводим курсор в нужное место шины (при этом подсветится красный пунктир внутри ее а под карандашом курсора появиться белый X) и щелкаем левой кнопкой. Я присвоил шине имя **SEG[1..8]** (еще раз напомню- точек две!!!), потому что в данном пробнике мне необходимо контролировать все восемь сигналов сегментов. Совет на будущее – если сделать отвод от шины и ввести для лэйбла только сигналы сегментов **SEG[1..7]**, то код на графике этой шины будет совпадать с семисегментным кодом символа, записанным в программе микроконтроллера. Иногда такое полезно при отладке.

После этого переходим в режим расстановки пробников напряжения (кнопка в левом меню с логотипом щупа и буквой V) и в непосредственной близости от лэйбла ставим пробник. При этом курсор ведет себя также (карандаш с X). Пробник автоматически поймает имя ближайшей маркировки, т. е. **SEG[1..8]**.

Аналогично я установил пробник и на другую шину с именем **DIG[1..8]**. Но здесь нам понадобятся отдельные пробники по разрядам. Поскольку проект нарисован довольно компактно, если ставить их на провода, то получится «каша» из зондов и их названий, они будут наползать друг на друга. Поэтому здесь я применил другую тактику. На свободном месте разместил восемь зондов напряжения (неподключенные зонды получают имя: **?**), а затем подвел их к шине, используя функцию автоповторения провода. Если на шине уже стоит лэйбл **DIG[1..8]**, соотвествующее имя получат и все зонды. Потом, используя функцию **Property Assignmebt Tools** (**PAT**), назначаем проводам нужные лэйблы (пусть Вас не пугает, что пробники не переименовываются автоматически мгновенно – достаточно один раз толкнуть симуляцию и все встанет на свои места) . Весь процесс

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

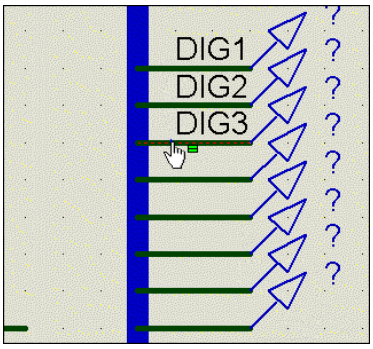

Рис. 29

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

2.17. Полезные свойства пробников.

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

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

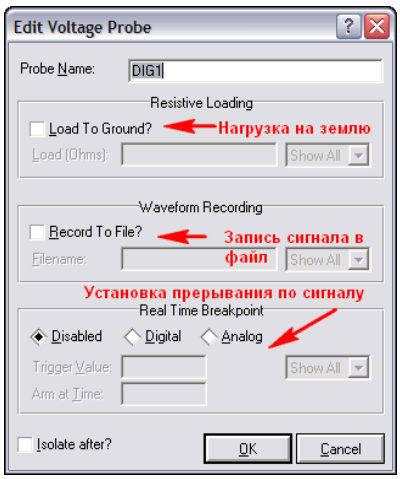

Рис. 30

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

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

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

Рис. 31

2.18. Digital Graph – применяем на практике.

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

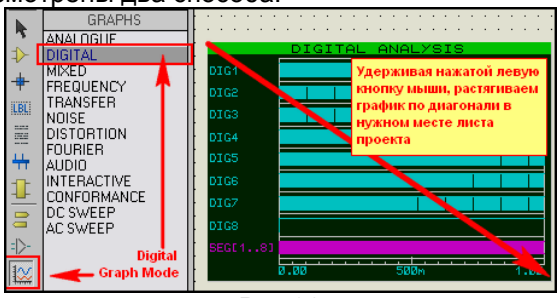

Рис.32

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

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

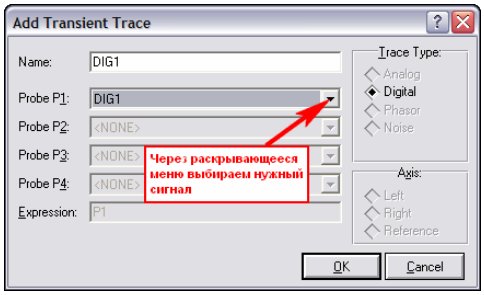

Рис. 33.

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

2.19. Свойства цифрового графика.

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

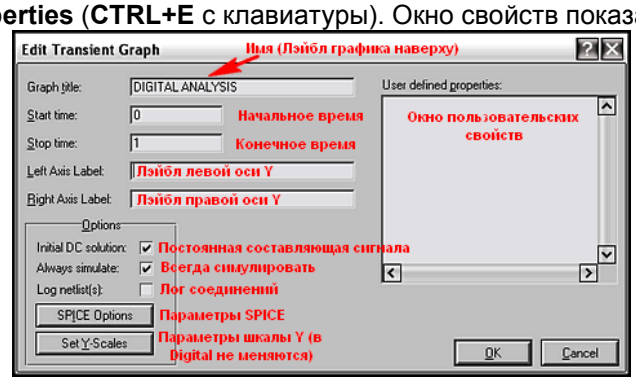

Рис. 34.

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

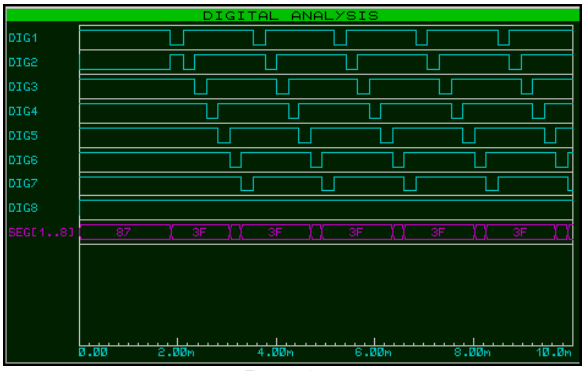

Рис. 35.

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

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

2.20. Дополнительные возможности анализа графика при максимизации окна.

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

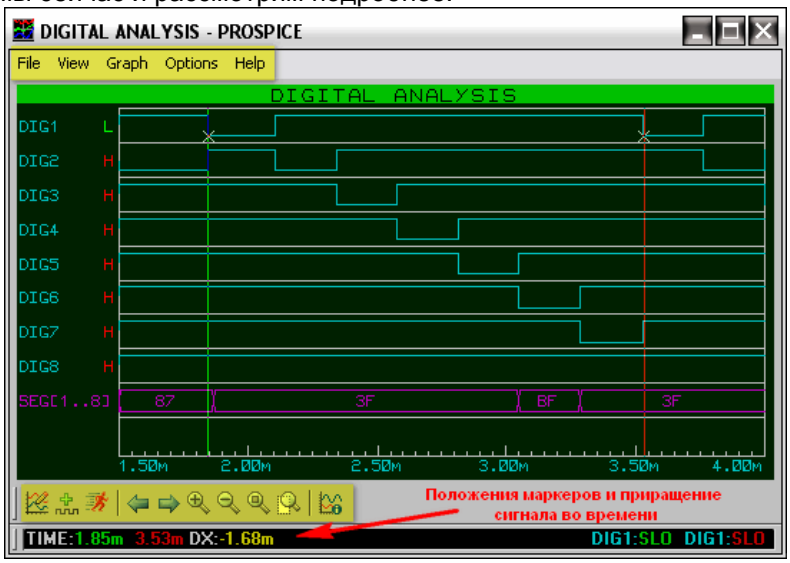

Рис. 36.

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

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

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

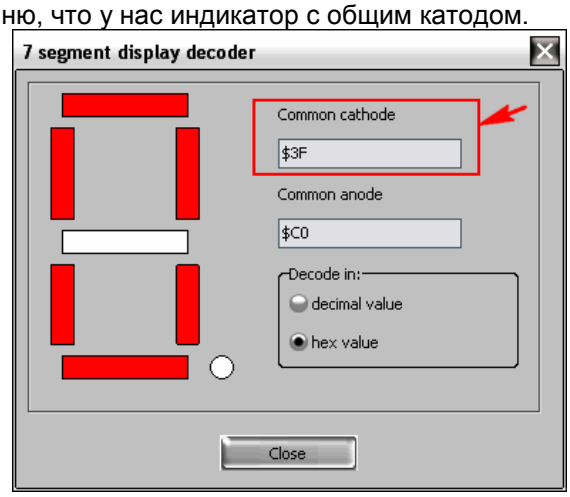

Рис. 37.

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

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

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

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

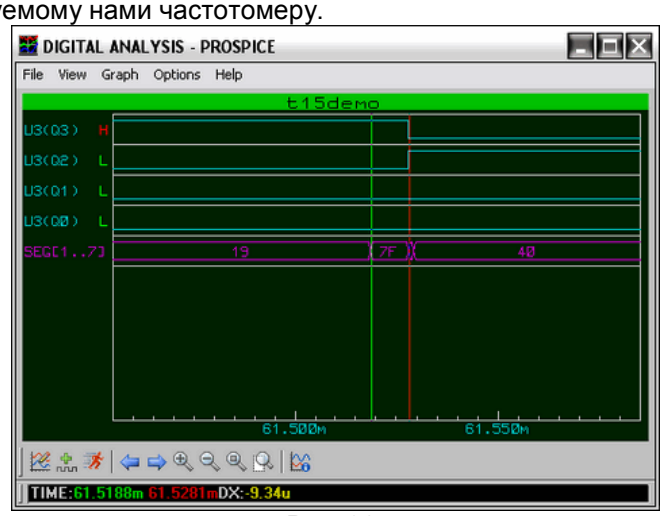

Рис. 38.

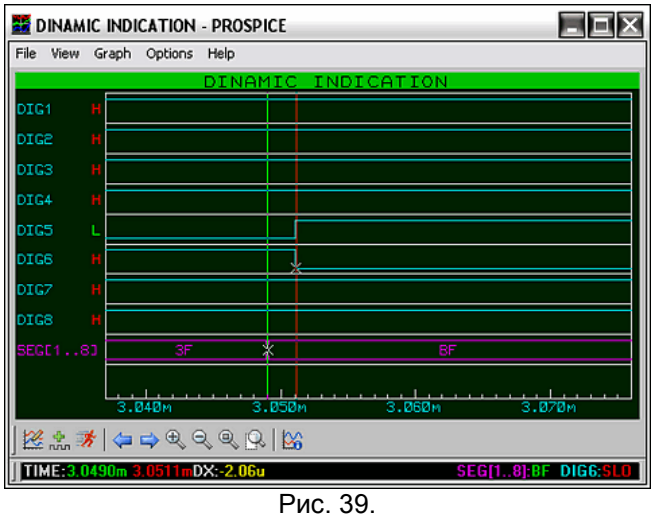

Рис. 39.

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

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

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

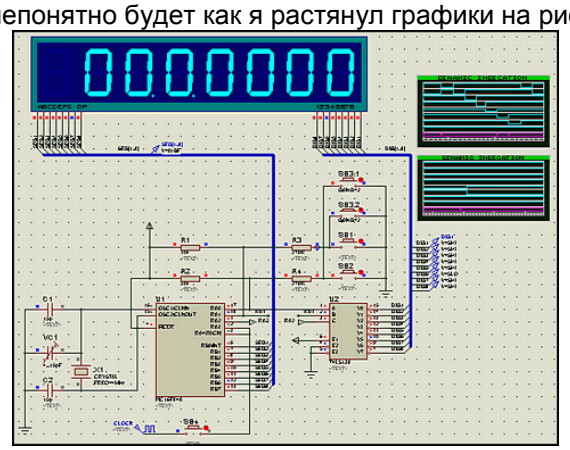

Рис. 40.

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

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

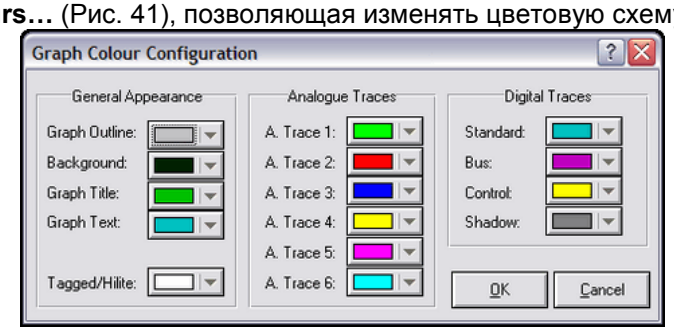

Рис. 41.

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

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

2.23. Подключаем файл микропрограммы для пошаговой отладки.

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

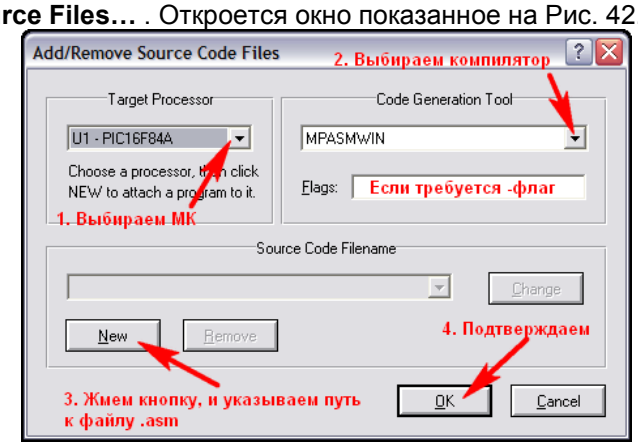

Рис. 42.

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

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

ASEM51 для МК серии MCS-51;

ASM11 для МК Motorola MC68HC11;

AVRASM2 для МК Atmel AVR;

MPASM и **MPASMWIN** для МК Microchip PIC.

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

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

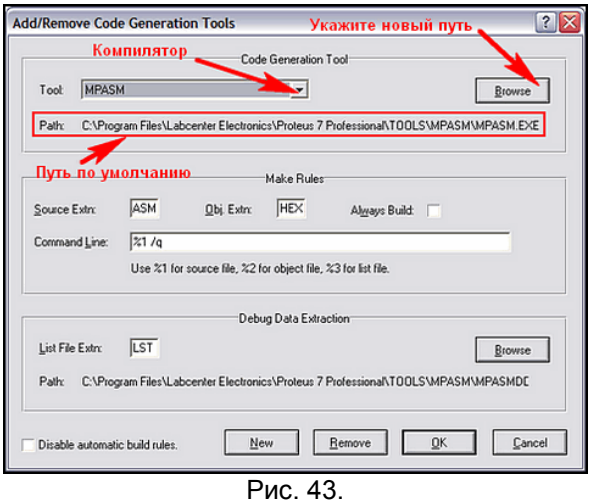

Рис. 43.

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

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

2.24. Режим пошаговой отладки программы в ISIS.

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

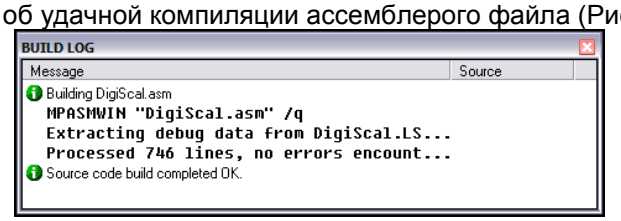

Рис. 44.

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

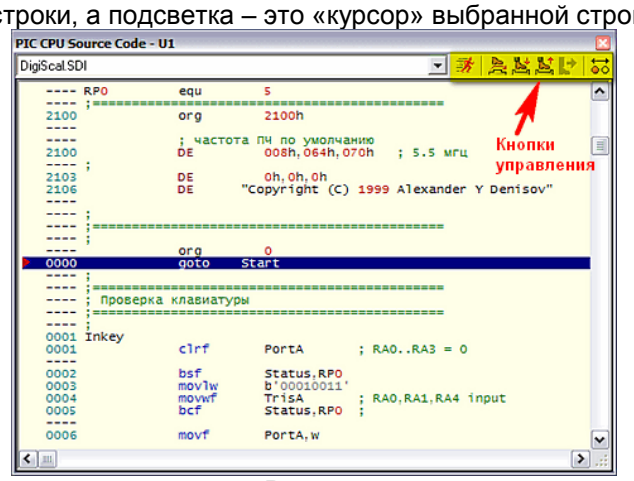

Рис. 45.

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

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

2.25. Контекстное меню окна пошаговой отладки.

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

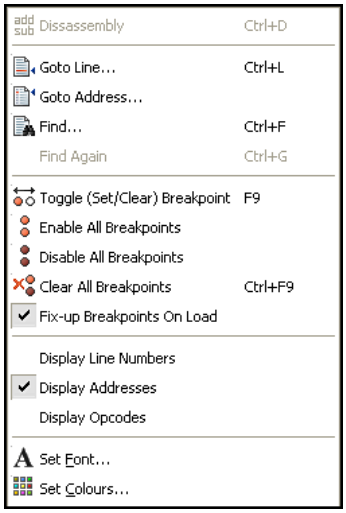

Рис. 46.

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

2.26. Меню Debug в развернутом виде.

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

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

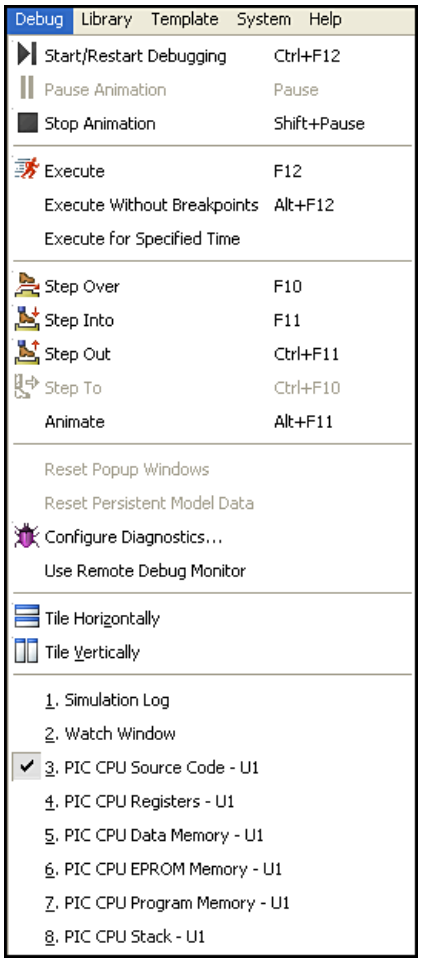

Рис. 47.

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

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

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

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

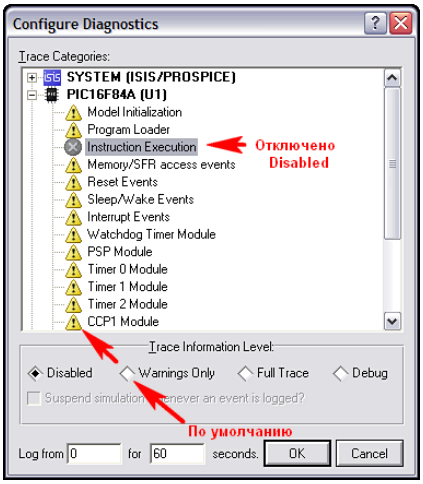

Рис. 48.

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

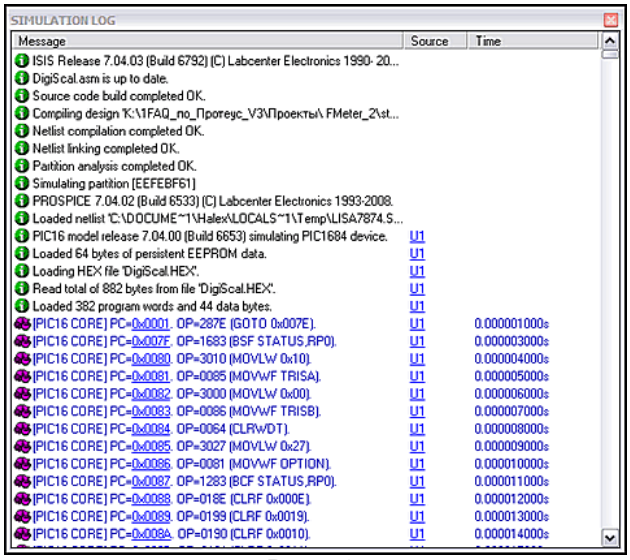

Рис. 49.

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

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

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

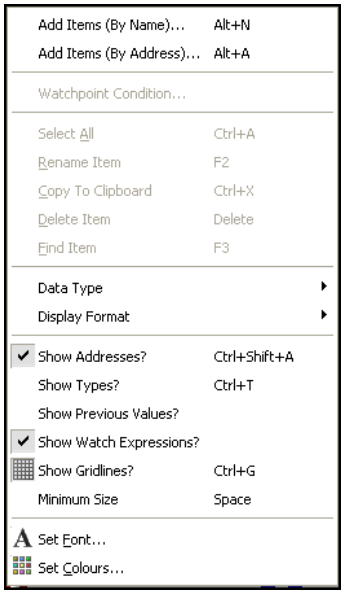

Рис. 50.

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

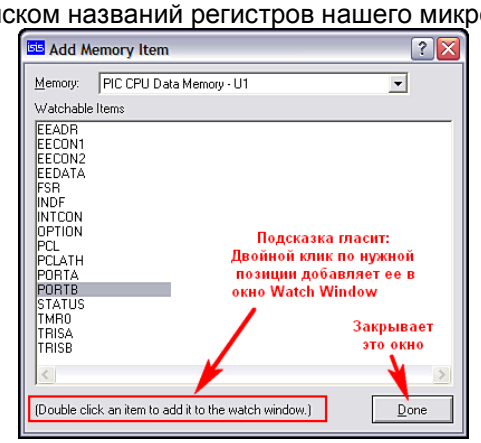

Рис. 51.

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

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

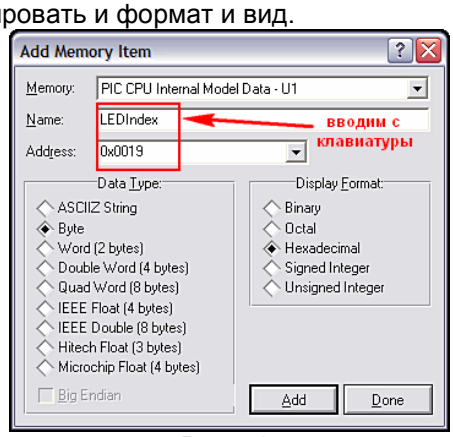

Рис. 52.

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

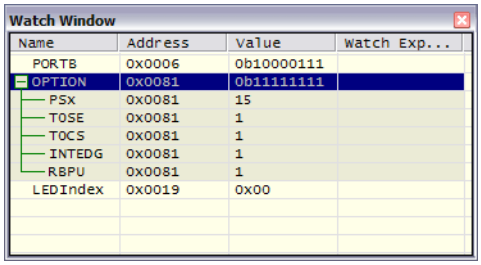

Рис. 53.

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

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

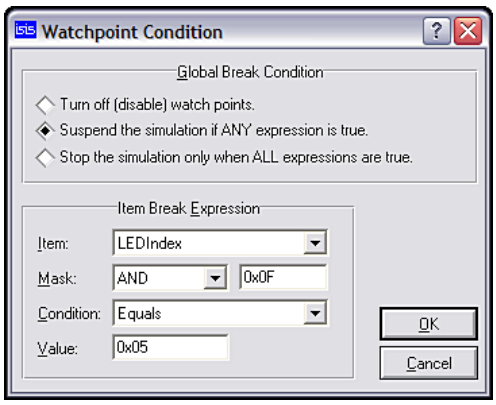

Рис. 54.

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

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

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

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

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

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

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

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

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

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

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

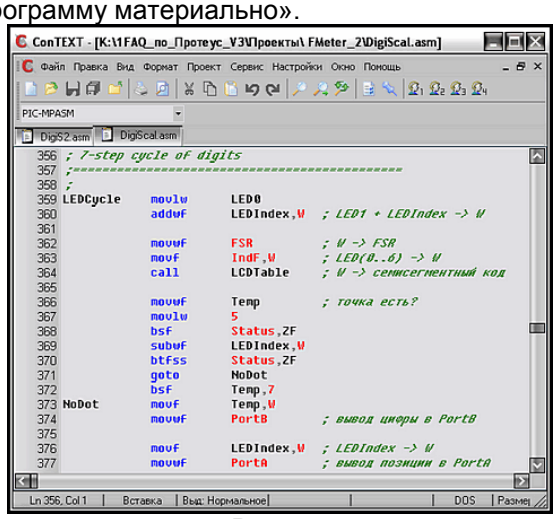

Рис. 55.

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

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

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

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

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

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

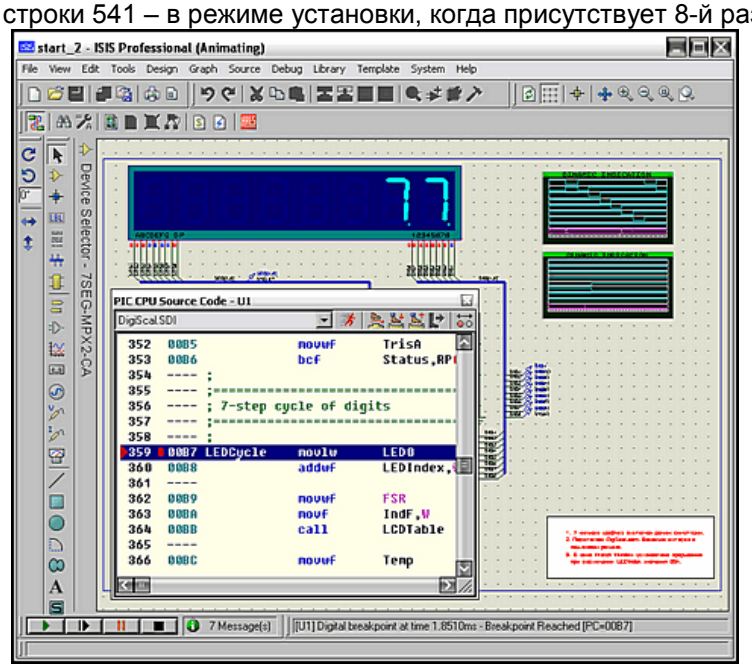

Рис. 56.

2.30. Реальные показания индикации в окне Watch Window.

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

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

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

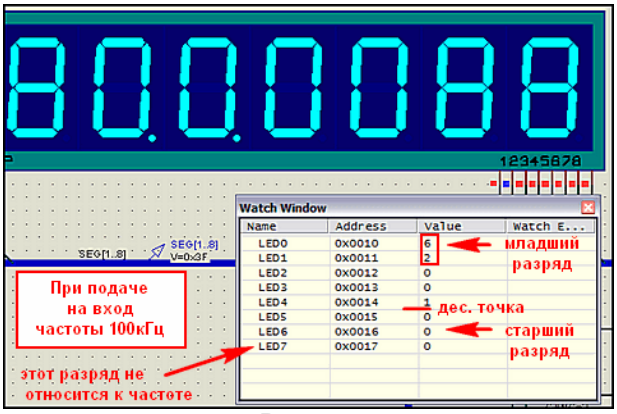

Рис. 57.

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

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

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

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

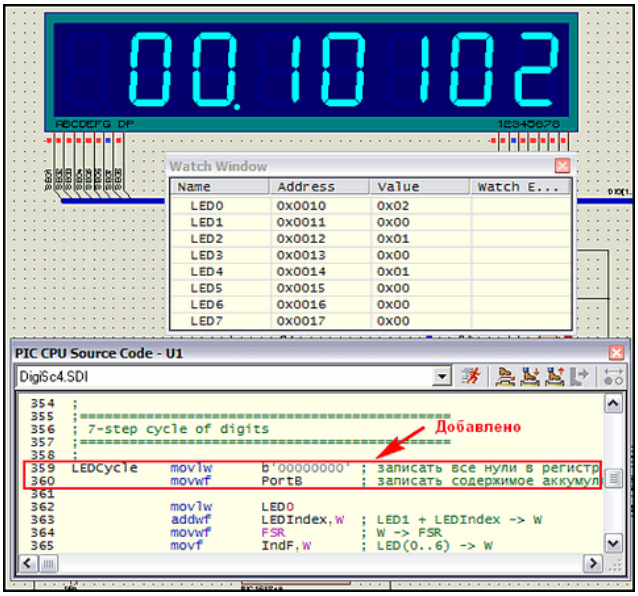

Рис. 58.

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

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

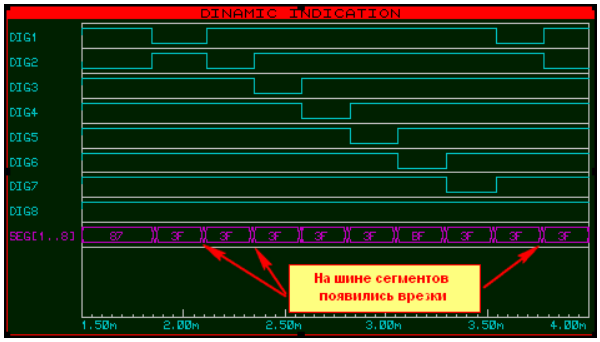

Рис. 59.

2.32. Финальный вариант проекта с рабочей индикацией.

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

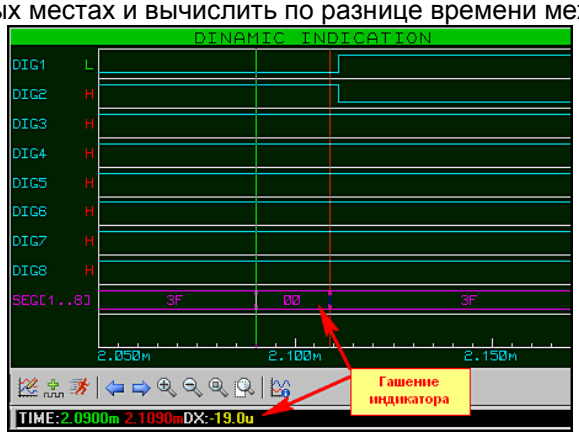

Рис. 60.

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

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

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

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

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

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

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

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

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

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

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

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

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

печатной платы в ARES – сделайте отдельную копию дизайна с транзисторными ключами и прочей аналоговой бижутерией.
Во-вторых: все попытки изменить яркость многоразрядных сегментных индикаторов ISIS обернутся полной неудачей. Сегменты индикаторов имеют только два реальных «цифровых» состояния: включено и выключено. При небольшом количестве индикаторов можно попробовать применить схематичные (**Schematic**) одноразрядные модели индикаторов, но и они, если не переделать на свой лад (о чем поговорим позже) тоже имеют только два состояния. Единственное, что они Вам дадут – это реальную токовую нагрузку, поскольку для их создания были применены аналоговые примитивы.
В-третьих: для того, чтобы динамическая индикация при симуляции выглядела реально, нам необходимо предусмотреть кратковременные паузы при смене разрядов, исключающие наползание разрядов индикации. Паузы могут быть достаточно короткими (в районе 10 микросекунд), но их присутствие необходимо. Первым на эту особенность обратил внимание **dosikus** и материал по этой теме был в одной из веток форума Kazus:
http://kazus.ru/forum/topics/10434.html ,
а также размещен на сайте Каллиграфа:
http://www.kaligraf.narod.ru/nedodellki.html
Дополнительно реальность показаний корректируется свойством **Minimum Trigger Time** индикатора.
Нужны ли эти паузы в реальной индикации – решать Вам. Здесь хочу обратить ваше внимание только на один нюанс, который мы обнаружили и устраняли при изучении первого проекта. Вспомните про перекрывающиеся импульсы разрядов индикатора. А теперь представьте, что мы зажгли в двух соседних разрядах все сегменты – восьмерка. Может быть в реальности глазом подсветка и не будет заметна, но ведь сегменты подключены непосредственно к портам и на момент перекрытия импульсов разрядов нагрузка на порты подскочит вдвое!!! Порт может и выдержит, но опять-таки представим, что наш девайс питается от батарейки. Нужны ли нам эти абсолютно бесполезные броски по току? А если учесть то, что изложено чуть ниже в пунктах **b** и **c**, то можно запросто и порт МК спалить.
Теперь остановимся на некоторых особенностях реальной индикации:

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

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

Книги:

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

Он-лайн ресурсы:

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

2.34. Заключение к первой части.

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

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