МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ РАДИОТЕХНИКИ,
ЭЛЕКТРОНИКИ И АВТОМАТИКИ»

АППАРАТНЫЕ И ПРОГРАММНЫЕ СРЕДСТВА СИСТЕМ УПРАВЛЕНИЯ ИНТЕЛЛЕКТУАЛЬНЫХ МОБИЛЬНЫХ РОБОТОВ

Методические указания
по выполнению лабораторных работ

МОСКВА 2011

Составитель: А.А. Алешин, В.В. Герасимов

Редактор: В.М. Лохин

Методические указания по выполнению лабораторного практикума по курсу "Аппаратные и программные средства систем управления интеллектуальных мобильных роботов" предназначены для магистрантов направления 221000.68 "Мехатроника и робототехника" программы "Интеллектуальные мобильные роботы".

Лабораторные работы выполняются на компьютере с использованием программы Proteus, предназначенной для симуляции электрических принципиальных схем с уклоном в цифровую электронику, а так же IDE CodeVisionAVR, предназначенной для написания программ управления микроконтроллерами AVR. Лабораторные работы построены таким образом, что они не предполагают углубленного знания студентами программ Proteus и CodeVisionAVR.

В ходе выполнения лабораторных работ студенты исследуют основные свойства и функциональные возможности программ, позволяющих без специальных аппаратных средств: написать и отладить программы управления микропроцессорами, освоить приёмы моделирования реальных устройств.

Лабораторный практикум может быть изменен по согласованию с научным руководителем и заведующим кафедрой выполнением работ в ходе научно-исследовательской практики.

ЛАБОРАТОРНАЯ РАБОТА №1

Знакомство с интерфейсом симулятора Proteus, написание программного кода в среде CodeVisionAVR

Цель работы: Ознакомиться с программными средами Proteus и CodeVisionAVR; изучить методы управления портами ввода–вывода микроконтроллеров семейства AVR.

Время выполнения работы – 4 учебных часа.

Задание на лабораторную работу*:*

1. Изучить описание интерфейса среды моделирования Proteus.
2. Изучить методику написания программы для микропроцессоров AVR в среде CodeVisionAVR.
3. Изучить методы работы с портами ввода–вывода микропроцессора.
4. Написать программу управления светодиодом.
5. Создать схему и программу, имитирующие светофор, или эффект «рыцаря дорог».

Методика выполнения работы:

Отличие Proteus от аналогичных по назначению пакетов программ, например, Electronics Workbench Multisim, MicroCap, Tina TI и т.п. заключается в развитой системе симуляции (интерактивной отладки в режиме реального времени и пошаговой) для различных семейств микроконтроллеров: 8051, PIC (Microchip), AVR (Atmel), и др. Proteus 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.

Упрощенно, работа в среде моделирования Proteus состоит

из следующих пунктов:

1. Размещение на чертежном поле необходимых элементов электрической принципиальной схемы (ЭПС) и задание их параметров функционирования.
2. Выполнение соединений элементов ЭПС.
3. Размещение виртуальных приборов там, где это необходимо, выбор режимов их работы.
4. Симуляция или проведение специализированного анализа.
5. Выполнение отладки программ микроконтроллеров, если это необходимо.

Для того чтобы получить представление о том, на что способен Proteus как среда моделирования, следует открыть некоторые файлы примеров проектов.

Например, посмотреть на процессы, происходящие при двухполупериодном выпрямлении можно, открыв файл:

\Labcenter Electronics\Proteus 7 Professional\SAMPLES\Interactive Simulation\Animated Circuits\Diode08.dsn (рис. 1.1).

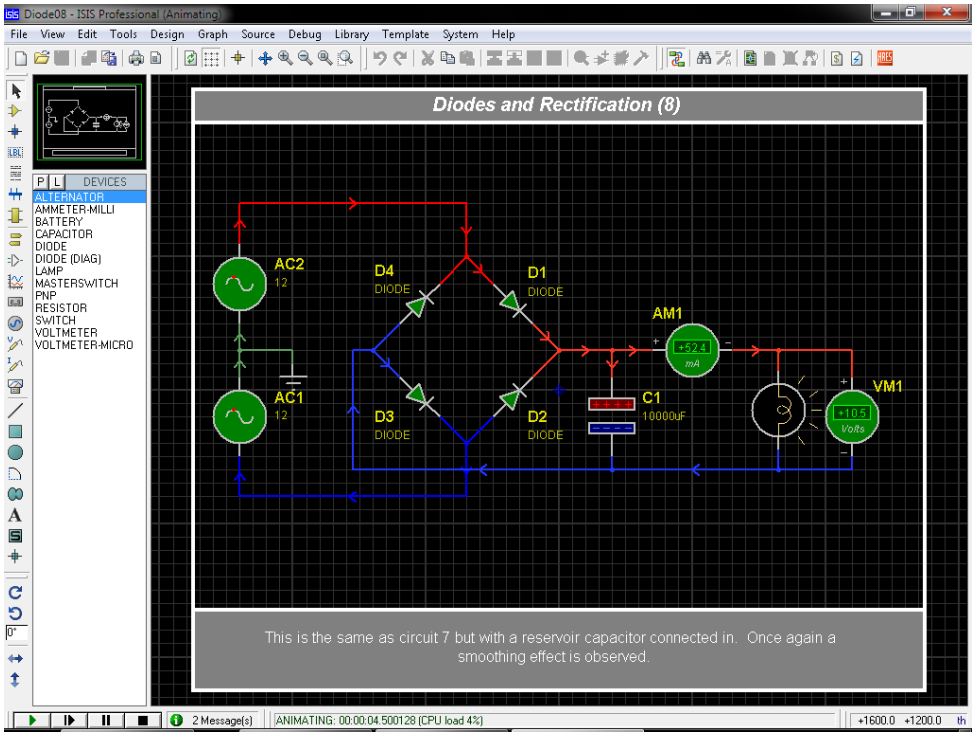

Рис. 1.1. Пример проекта по симуляции двухполупериодного выпрямления

Proteus может симулировать и более сложные ЭПС, как, например, происходит в проекте:

\Labcenter Electronics\Proteus 7 Professional\SAMPLES\Interactive Simulation\Motor Examples\SERVO.dsn (рис. 1.2).

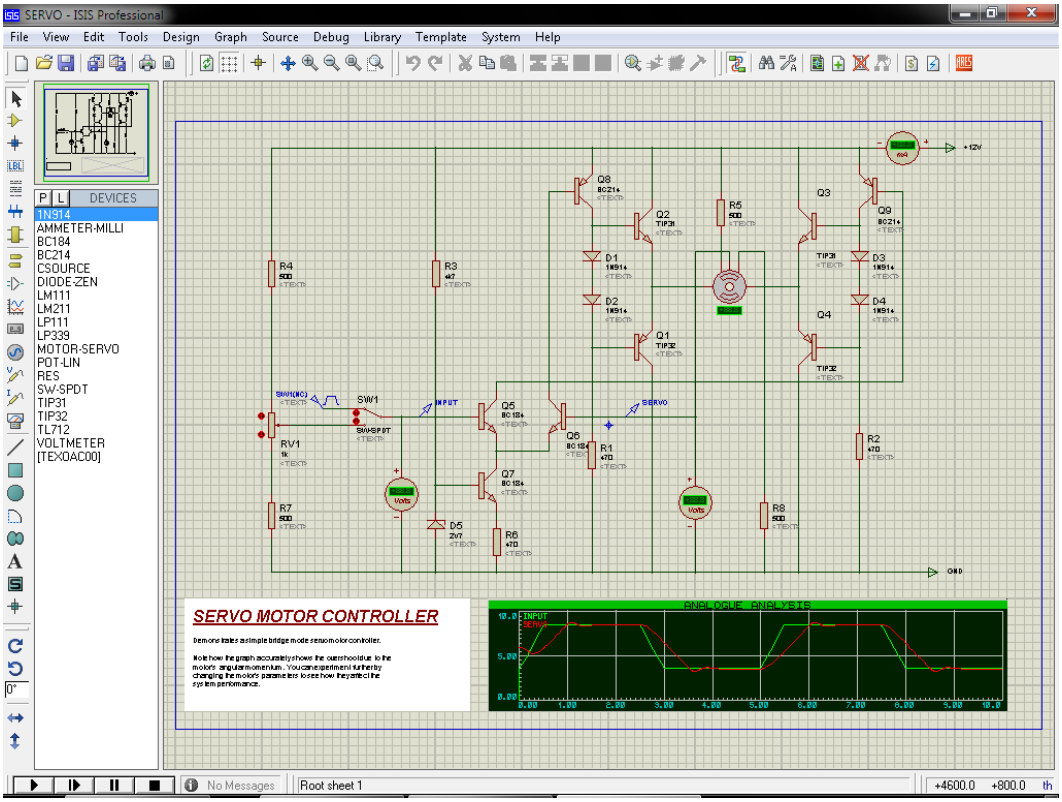

Рис. 1.2. Симуляция мостового управления двигателем

И, как с венцом интерактивной симуляции, можно познакомиться с одним из проектов из папки:

\Labcenter Electronics\Proteus 7 Professional\SAMPLES\VSM Chess\Tiny Chess

Например, открыв проект avrchess.dsn и запустив симуляцию, можно поиграть в шахматы с компьютером (рис. 3).

К сожалению, реализованный там ИИ не слишком силен и ему вполне можно поставить детский мат.

(Поскольку симулятор находится в состоянии активной разработки/доработки, возможна ситуация, когда приведенные примеры находятся в других папках. Если не получается найти указанные примеры – следует поискать их в соседних папках.)

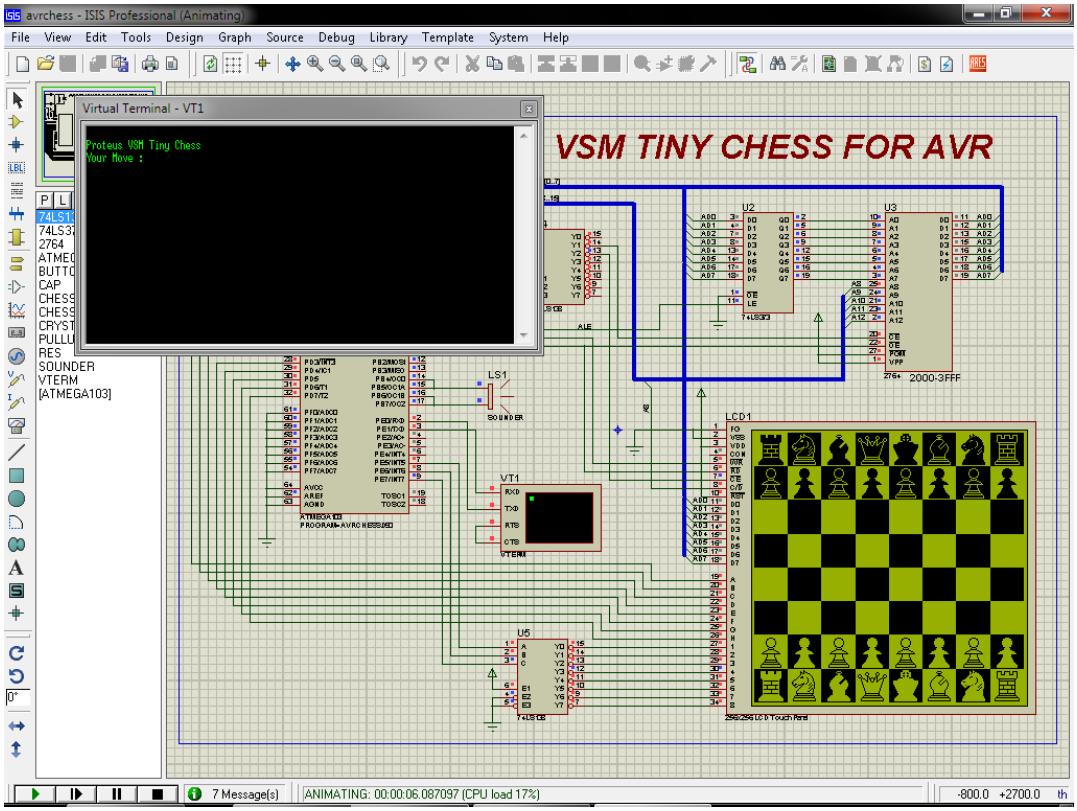

Рис. 1.3. Симуляция игры в шахматы

Создадим новый проект в симуляторе. Все настройки можно оставить по умолчанию. Внешний вид главного окна программы показан на рис. 1.4.

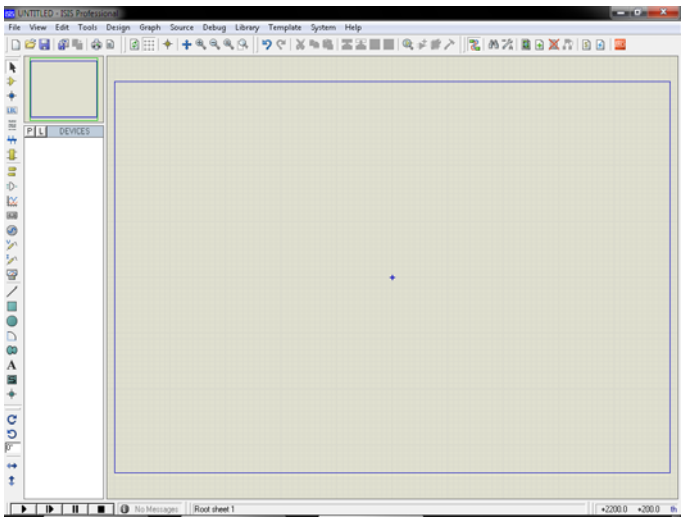

Рис. 1.4. Главное окно программы Proteus

Меню File, View и т.д. интуитивно понятны, также интуитивно понятны значения пиктограмм верхней строки (Создать новый файл, открыть дизайн и т.д.). Для дальнейшего понимания и продуктивной работы в Proteus следует познакомиться с назначением пиктограмм левой панели.

В зависимости от размеров окна программы, положение основных панелей инструментов может немного изменяться. Например, таким образом – рис.1.5.

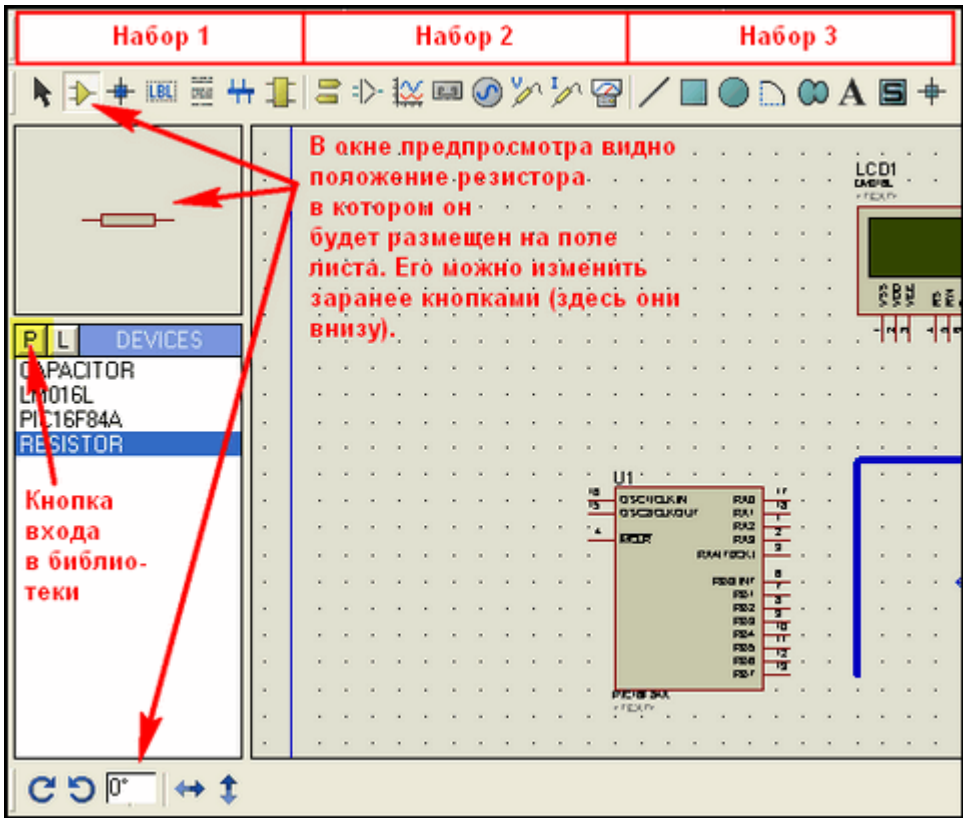

Рис. 1.5. Элементы управления симулятора

Четкого разделения по функциональному назначению в панели инструментов нет, поэтому условно её можно разделить так, как проведены границы в Proteus на три набора. Однако это сделано лишь для удобства описания. К тому же по умолчанию там расположены кнопки поворота/отражения объекта. Одно основное свойство левой панели инструментов – кнопки не имеют дублирующего вызова функций с клавиатуры. Все

действия здесь возможны только мышкой. В скобках за названием кнопки размещено описание её вида в меню.

Набор 1

Selection Mode *(жирная черная косая стрелка–указатель)* – режим выбора. В этом режиме в окне редактирования единичным щелчком левой кнопки мыши (ЛКМ) по объекту (компоненту, проводу, шине, графическому элементу) можно выделить его – он становится 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 их помощью можно выбрать в каком положении будет размещаться объект на поле в окне редактирования. В окне предпросмотра это положение будет отражено. В режиме **Component Mode** первый щелчок ЛКМ 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. Расстановка как и в предыдущем режиме на два щелчка ЛКМ: подсветка, установка.

Wire Lebel mode *(кнопка LBL)* – режим текстовой маркировки проводов в проекте (и шин тоже). При наведении курсора на маркируемый провод/шину под изображением карандаша появляется косое перекрестие, после чего щелчок ЛКМ 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.**

Text Script Mode *(горизонтальные пунктиры, изображающие текст)* – режим размещения текстовых скриптов (простых многострочных текстов). Щелчок ЛКМ по свободному полю в проекте вызывает всплывающее окно встроенного редактора текста **Edit Script Block** (рис 1.7.**)**. В окне **Text** набирается текстовый блок. Допустим импорт текста из текстовых файлов или наоборот экспорт через соответствующие кнопки внизу справа. Переключателями **Rotation, Justification** выбирается расположение/ориентация текста в проекте (а не здесь в окне **Text –** не следует путать). Если **Wire Autorouter** в верхнем меню **(**рис. 1.6**)** выключен, шины можно протягивать не только под прямым углом, но и наискось.

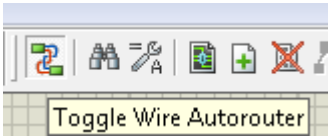

Рис. 1.6. Включение/отключение автоматического проведения проводников

На рис. 1.8 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. 1.5), но набор функций немного другой. Последовательность действий везде одна и та же: сначала снять флажок, затем изменить параметр, который при этом становится доступным для изменения.

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

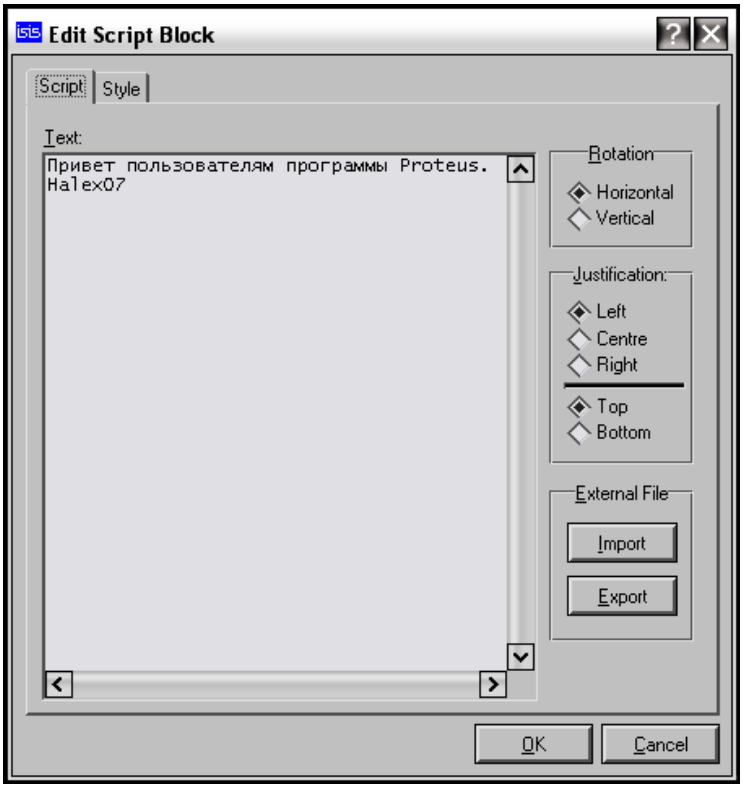

Рис. 1.7. Встроенный редактор текста

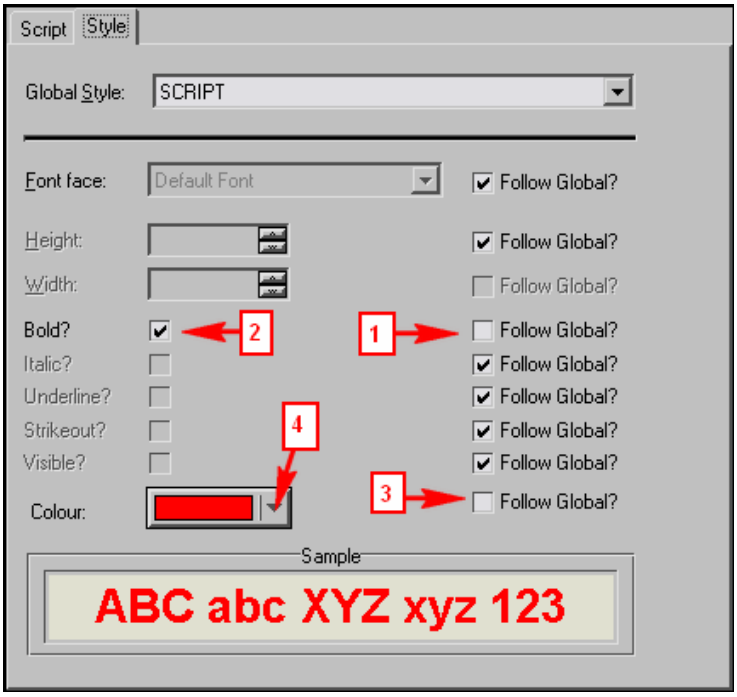

Рис.1.8. Последовательность действий по изменению параметров шрифта

Subcircuit Mode *(желтый прямоугольник с выводами справа и слева)* – режим размещения субмодулей. Модули – прямоугольники с толстой синей окантовкой и заливкой цветом компонентов – позволяют в Proteus вынести функционально законченные узлы на отдельные листы **(Child Sheet** – дочерний лист модуля). Рисуется модуль удержанием нажатой ЛКМ 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 (ПКМ) 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 следует полагать, что если на выходе микросхемы установлен терминал **Output,** то назад в микросхему он сигнал не пропустит. 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. Следует обратить внимание на то, 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]** получим 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. можно развести по компонентам. Если же мы разместим на отводе **A[0..15]** терминал **A[8..11],** то поместив терминал с таким же именем в другом месте, получим на нем только четыре выделенных разряда A8, A9, A10 и A11, которые и можем растащить дальше одноименными проводами.

Device Pins Mode *(расчлененное изображение ОУ на сером фоне)* – режим расстановки выводов модели устройства (компонента) при его создании.

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 ЛКМ диагонально.

Tape Recorder Mode (*изображение магнитофонной кассеты*) – режим установки магнитофона. Сигналы, генерируемые разработанным устройством, в процессе симуляции можно записать в файл с последующим использованием их в другом проекте. Для этого и служит этот виртуальный магнитофон.

Generator Mode *(синусоида в окружности)* – режим расстановки виртуальных генераторов. В этом режиме в окне селектора доступны для выбора и размещения в схеме виртуальные генераторы сигналов. Условно их можно разделить на цифровые (все начинающиеся с букы D, за исключением DC – постоянный потенциал) и аналоговые (все оставшиеся). Отдельно отметим **SCRIPTABLE** – програмный генератор, для которого предварительно надо написать скрипт – программу на встроенном в Proteus языке **Easy HDL.** Два генератора **FILE** и **AUDIO** используют для генерации сигналов файлы, предварительно записанные на жесткий диск. 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 доступные поворот и отражение. При последующем подключении к выводу компонента или проводу генератор автоматически изменит свое имя на имя соответствующего ближайшего вывода. Если оно по каким-то причинам не подходит, 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 стрелочного прибора)* – режим выбора и размещения виртуальных инструментов в проекте. В Proteus, как и во многих других программах-симуляторах, имеется обширный инструментарий виртуальных приборов: вольтметры, амперметры, четырехканальный осциллограф, счетчик/частотомер, генератор сигналов и другие специфичные приборы. В этом режиме осуществляется их выбор и размещение в проекте. Мы будем обращаться к ним по мере изучения ISIS, а пока отметим, что большинство из них, за исключением вольтметров/амперметров имеют однополюсное подключение. Это означает, что измерение осуществляется ОТНОСИТЕЛЬНО земляного провода. Об этом не стоит забывать, чтобы не получить нереальные значения измерений. В примерах Proteus есть проекты с использованием виртуальных приборов и всегда можно посмотреть: как они действуют в реальности. Единственное замечание – там использована двухканальная модель осциллографа из старых версий Proteus. Также следует отметить, что виртуальные приборы активно используют определенное количество ресурсов компьютера и в сложных проектах могут стать причиной тормоза симуляции в режиме реального времени.

Набор 3

Все кнопки данного набора относятся к режиму 2D (двухмерной) графики. Поскольку лабораторные работы не предполагают создание собственных компонентов, рассматривать эти наборы кнопок мы не будем.

Proteus – замечательная среда симуляции. Однако стоит всё время помнить, что, как и любой симулятор, он обладает рядом недостатков. Одним из них является некоторая излишняя упрощенность происходящих процессов. Рассмотрим эту ситуацию на примере светодиода (рис. 1.9).

Разместим на чертежном поле светодиод, резистор, источник питания, земляную клемму и измерительные приборы.

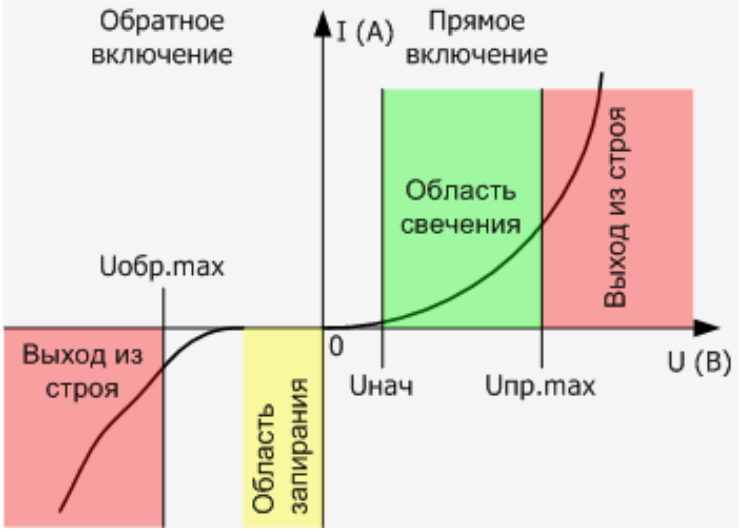

Рис. 1.9. ВАХ светодиода

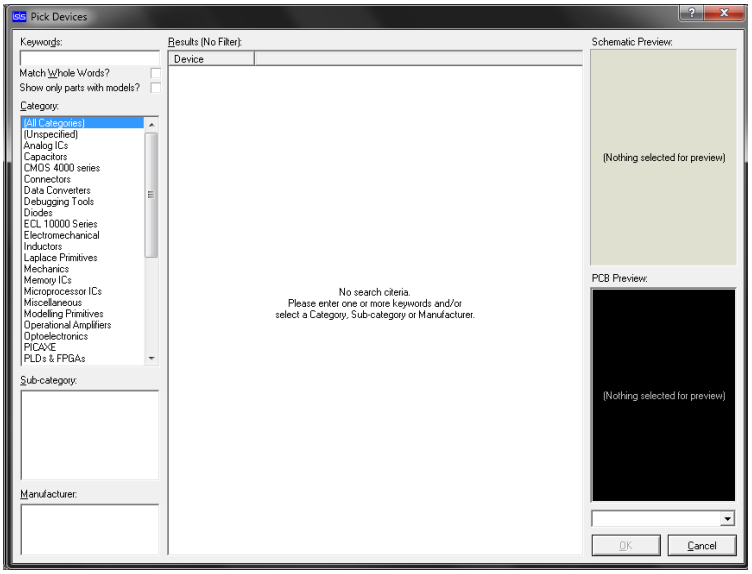

Рис. 1.10. Библиотечное окно

Для этого сделаем следующее:

1. Нажмем кнопку входа в библиотеку (P). Откроется библиотечное окно (рис. 10).

2. В поле «Keywords» введем слово LED. В списке ниже выберем раздел Optoelectronics. В большом правом поле Results следует выбрать один из предлагаемых светодиодов. Выберем, например, анимированную модель под названием LED-RED (рис. 1.11). Нажмем кнопку OK и поместим светодиод на чертежное поле.

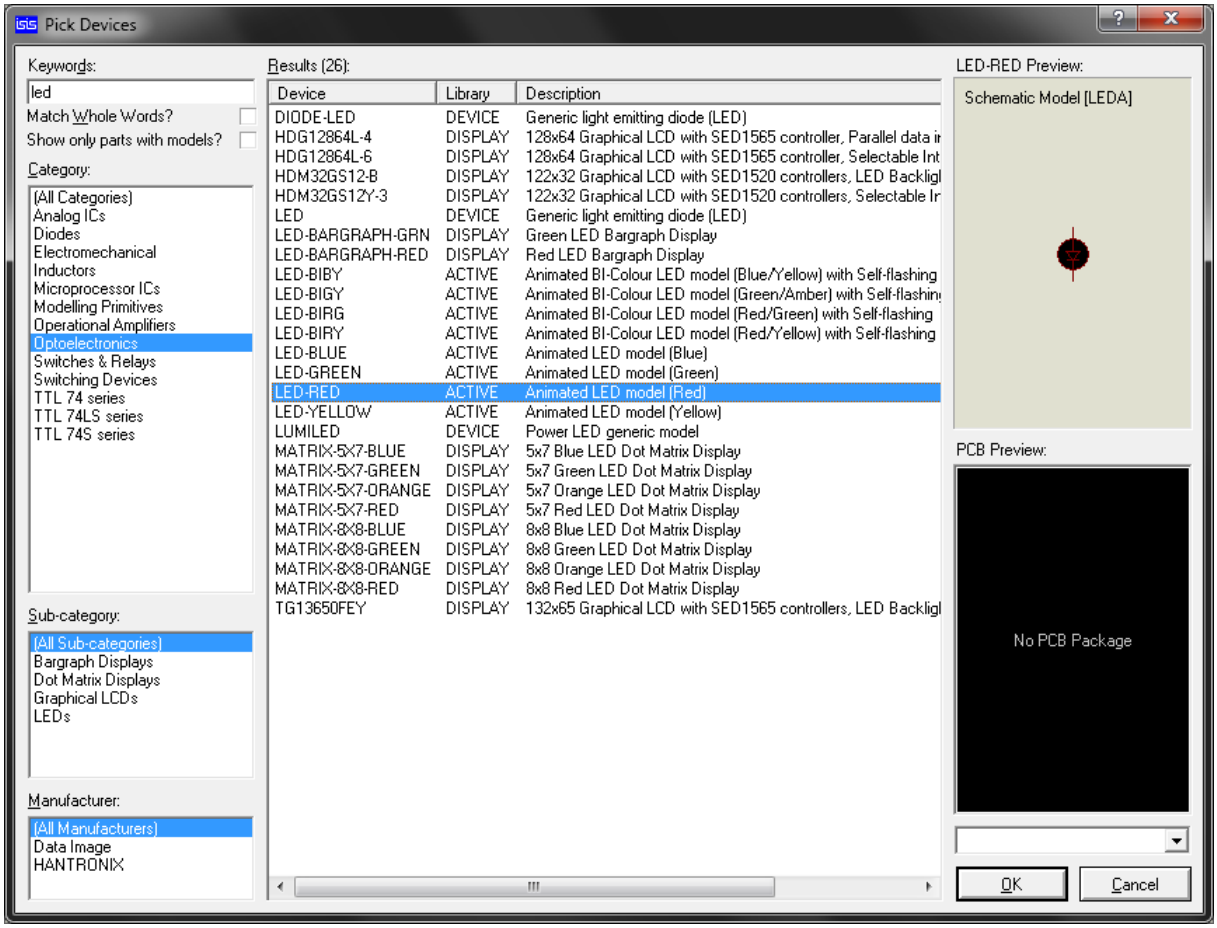

Рис. 1.11. Выбор модели светодиода

3. Таким же образом разместим на чертежном поле резистор (аналоговый примитив) по ключевому слову RESISTOR.

4. Для размещения клемм питания, необходимо в левой боковой панели выбрать режим Terminals mode и в появившемся

справа списке поочередно выбрать Power и Ground, размещая их щелчками мыши на чертежном поле.

5. После проделывания указанных операций, на чертежном поле должны быть элементы, показанные на рис. 1.12.

6. Так же нам понадобятся измерительные приборы. В данном случае необходимы один амперметр постоянного тока и два вольтметра постоянного тока. Их можно достать в левой боковой панели Virtual Instruments.

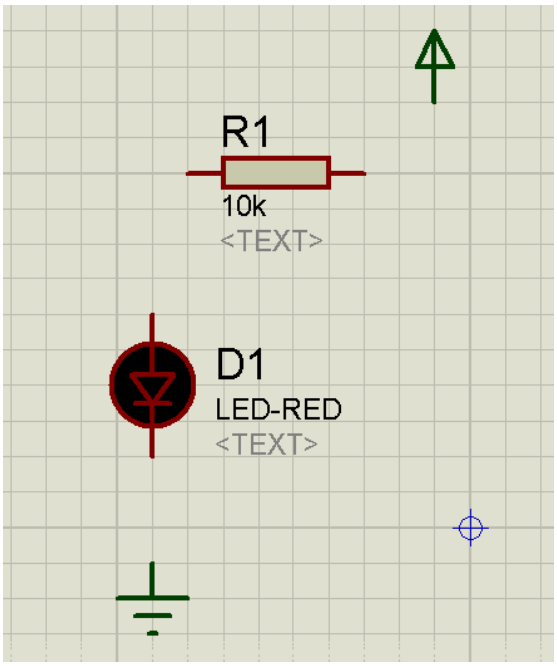

Рис. 1.12. Размещенные компоненты для тестирования светодиода

7. Выполните соединение элементов как на рис. . Соединения проводятся путем щелчка ЛКМ по одному из выводов элемента (вывод помечается пунктирным квадратом). После того, как провод «привяжется» к курсору, следует провести его до следующей точки соединения и выполнить щелчок ЛКМ. Напряжение питания схемы задается двойным щелчком ЛКМ по клемме питания. Введите туда строку «+5V» без пробелов между символами. Важно помнить, что желательно указывать знаки питания, напряжение и единицы его измерения.

Иначе в проекте с различными номиналами питания возможна путаница. Вообще, свойства объектов, размещенных на чертежном поле, изменяются по двойному щелчку ЛКМ на объекте. Поменяйте сопротивление резистора на 330 Ом, а отображение амперметра с ампер на миллиамперы (рис. 1.13) (к слову, единицы измерения в Proteus – в системе СИ. Если не ставить после значения сопротивления каких-либо символов – то это Омы. Если нужны килоОмы – ставим k. Мега – M (именно прописную, т.к. строчная m – это мили.)).

8. Запустим симуляцию нажатием F12 на клавиатуре, либо кнопкой Play в левом нижнем углу экрана (рис.1.14).

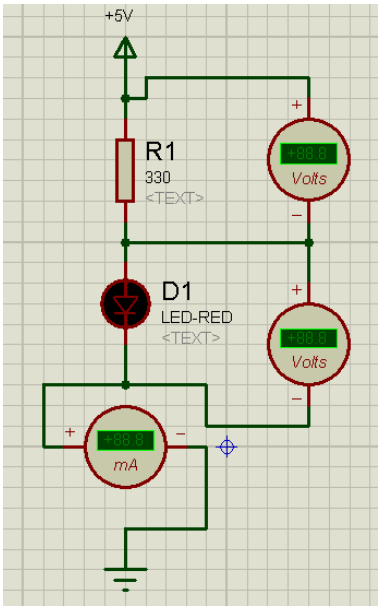

Рис.1.13. Итоговая схема включения светодиода

Вольтметры показывают падение напряжения на резисторе и светодиоде соответственно. Амперметр показывает общий ток в цепи. Слева, внизу, рядом с кнопками управления симуляцией появился значок i в зеленом кружке. По нажатию на этот значок появляется окно Simulation Log, в котором отображаются системные сообщения о прохождении симуляции. Как видим, никаких предупреждений нет. Остановите симуляцию двойным нажатием Esc, либо кнопкой Stop. А теперь, измените значение напряжения. Например, выставьте 100 В и запустите симуляцию

(рис. 1.15). Как видите, с точки зрения симулятора всё в порядке. В реальной схеме уже в первую секунду произошел бы пробой p-n перехода светодиода.

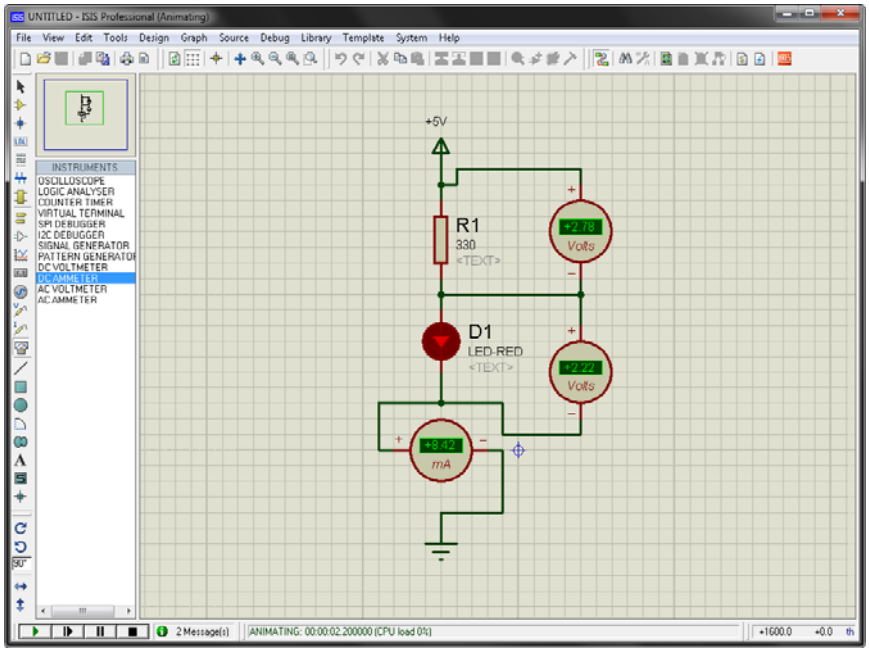

Рис. 1.14. Запущенная симуляция

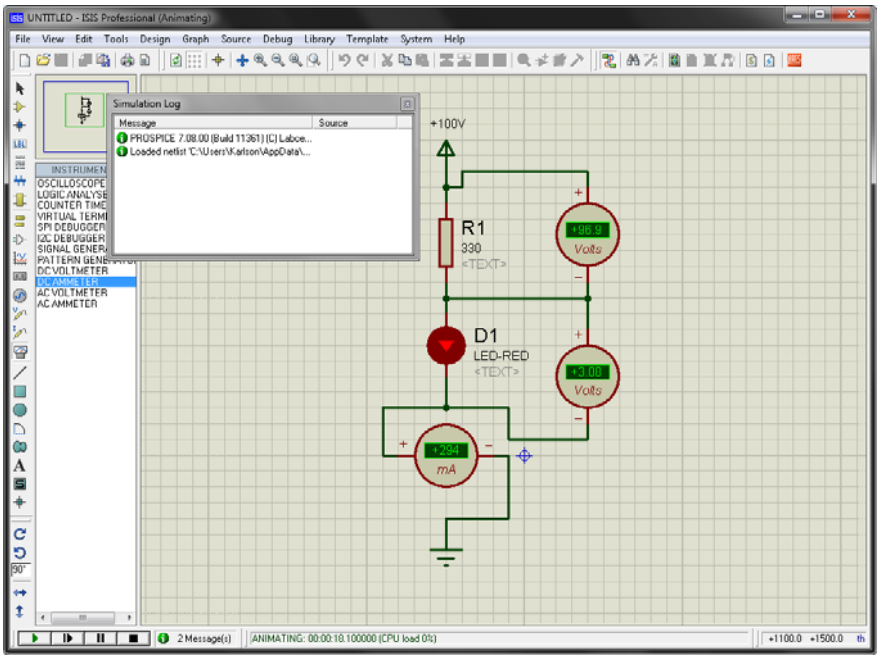

Рис. 1.15. Симуляция с «неправильным» напряжением

Поэкспериментируйте с другими моделями светодиодов. Большинство из них ведут себя столь же некорректно. Об этом необходимо помнить всегда. В некоторых случаях ограничения на симуляцию компонентов прописаны в справке.

Однако в таком упрощении есть и некоторые положительные стороны. Например, для симуляции микроконтроллера (МК) не требуется правильное подключение цепи сброса или кварцевого резонатора, а так же шин питания – всё это Proteus делает самостоятельно.

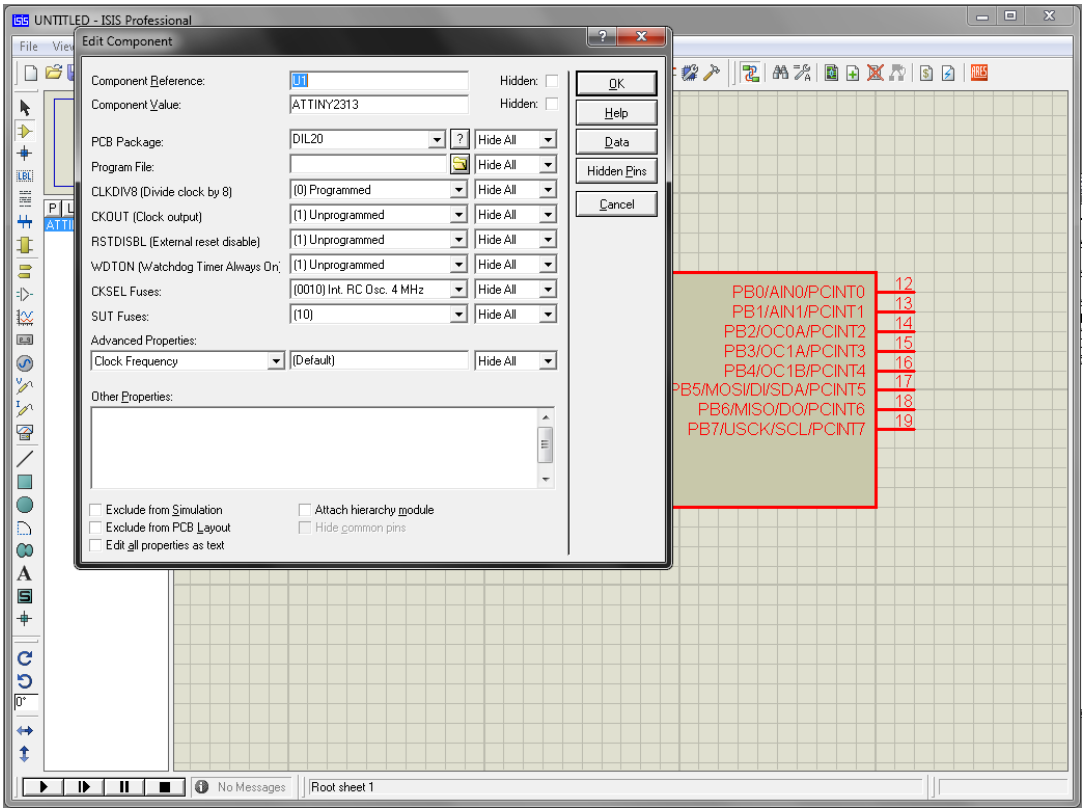

Рис. 1.16. Окно свойств МК ATTiny2313.

Для дальнейшей работы нам понадобится микроконтроллер типа ATTiny2313. Разместите его на чертежном поле и откройте свойства МК (рис. 1.16).

Значения всех пунктов можно оставить по умолчанию. Программа, исполняемая МК подгружается из поля «Program File» (необходимо нажать на папку и через обычное меню выбора

файла открыть .hex файл с созданной программой).

Теперь нам необходимо познакомиться с созданием исполняемых микроконтроллером программ.

Многие современные микроконтроллеры построены на вариациях гарвардской архитектуры. В частности, микроконтроллеры AVR имеют гарвардскую архитектуру (программа и данные находятся в разных адресных пространствах) и систему команд, близкую к идеологии RISC.

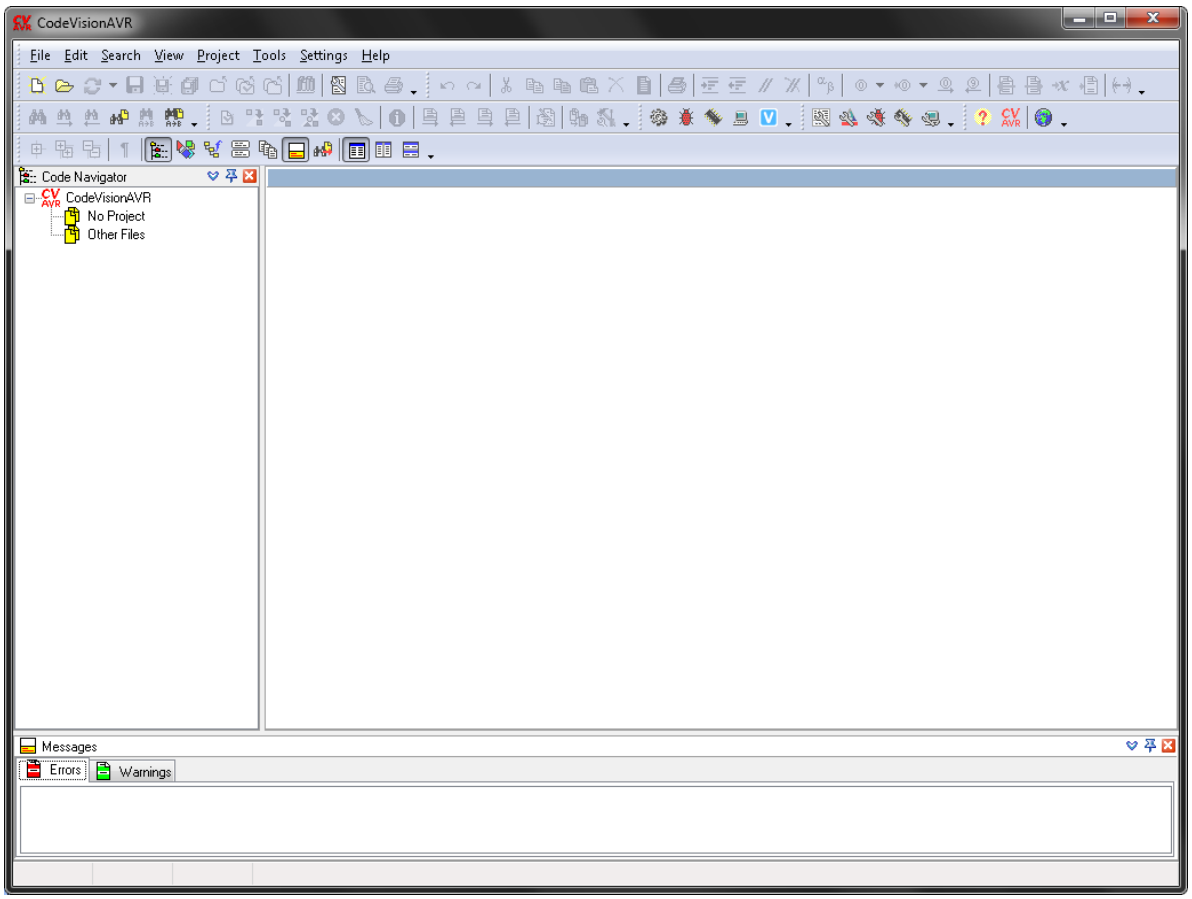

Рис.1.17. Внешний вид главного окна CVAVR.

Программы для МК можно писать на различных языках программирования, начиная с ассемблера и Си и заканчивая такими высокоуровневыми языками, как Java или C#. Соответственно, существует большое количество сред разработки ПО и компиляторов. Разумеется, есть проприетарные

версии компиляторов и свободно распространяемые. CodeVisionAVR относится к проприетарным средствам разработки, но налагаемые на бесплатную версию ограничения в случае выполнения лабораторных работ не имеют особого значения. CVAVR очень удобен тем, что не требует глубокого знания и понимания процесса компиляции программ (не нужно писать вручную make-файл) для МК, поэтому мы будем пользоваться этой средой (рис. 1.17).

Для создания нового проекта необходимо нажать иконку шестерни в одной из верхних панелей. В открывшемся диалоговом окне убедиться, что помечен радиобаттон AT90, ATtiny, ATmega и нажать OK (рис. 1.18).

Далее, во вкладке Chip, необходимо выбрать из выпадающего списка тип МК и задать его тактовую частоту (рис. 1.19).

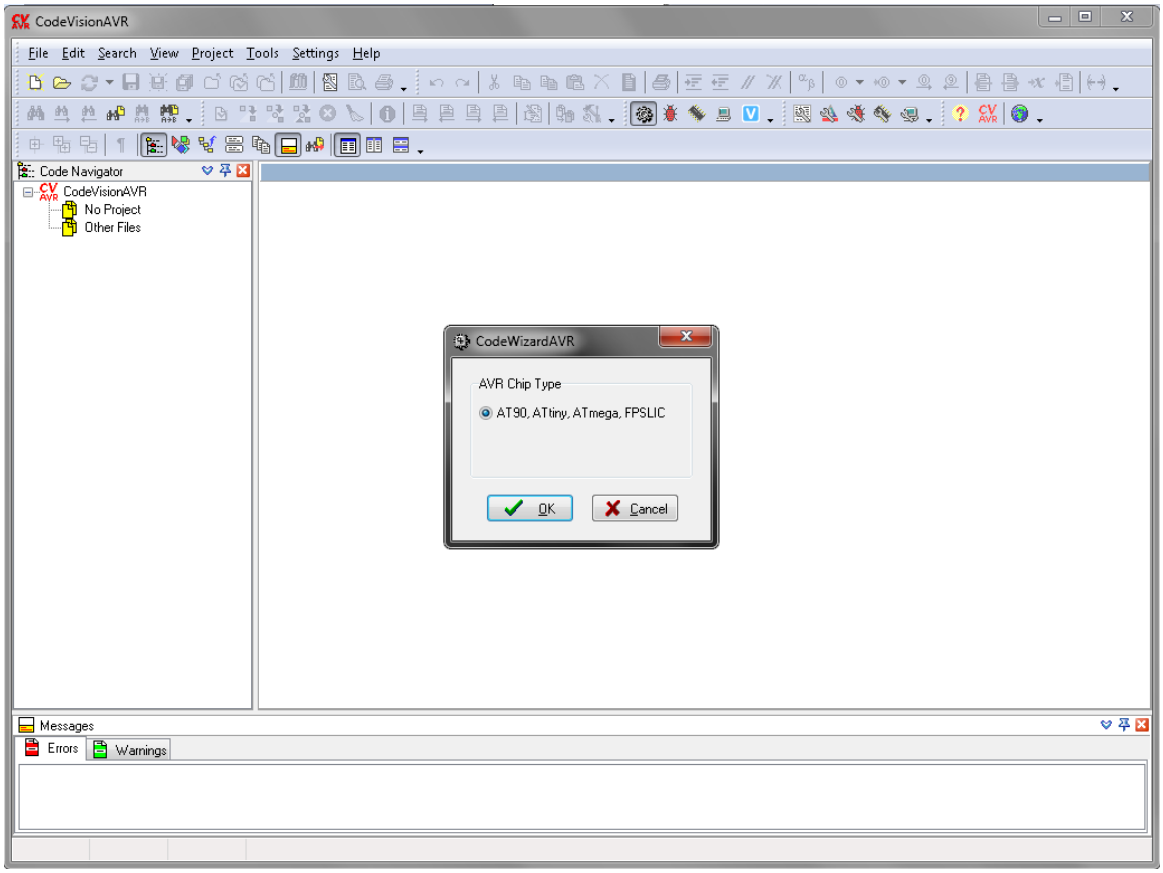

Рис. 1.18. Создание нового проекта.

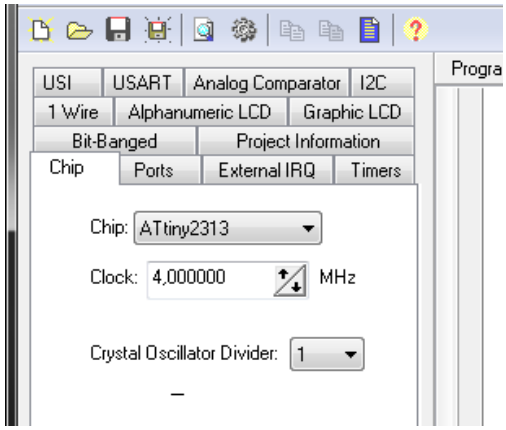

Рис. 1.19. Выбор МК и тактовой частоты.

Тактовая частота в 4МГц вполне достаточна для большинства учебных задач, к тому же МК, симулируемый в Proteus на такой частоте, не будет сильно тормозить (в частности на слабых компьютерах).

Для того чтобы среда сгенерировала файл проекта, необходимо нажать иконку шестерни (Generate program, save and exit) в верхней панели CodeWizardAVR. Лучше всего завести отдельную папку для каждой группы студентов и для каждой лабораторной, чтобы не смешивать файлы.

По умолчанию мастер создания кода инициализирует всю периферию МК. Но на первом этапе большая её часть нам совершенно не нужна. Необходимо оставить только инициализацию портов ввода-вывода и служебные инструкции для компилятора.

После удаления ненужных частей, начальный код программы может выглядеть следующим образом:

```
#include <tiny2313.h>

// Declare your global variables here

void  main(void)
{
// Declare your local variables here

// Crystal Oscillator division factor: 1
```

```
#pragma optsize-
CLKPR=0x80;
CLKPR=0x00;
#ifdef _OPTIMIZE_SIZE_
#pragma optsize+
#endif

// Input/Output Ports initialization
// Port A initialization
// Func2=In Func1=In Func0=In
// State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;

// Port D initialization
// Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;

while (1)
      {
      // Place your code here

      }
}
```

Для того чтобы помигать светодиодом, подключенным к МК, надо настроить какой-либо из пинов какого-либо порта ввода-вывода в режим Out (основываясь на лекции, сделайте это самостоятельно). Далее в цикле while нужно менять состояние отдельного пина.

Обращение к конкретному биту порта ввода-вывода

выглядит так:

```
PORTD.2 = 1;
```

Сначала пишем имя порта, потом через точку номер бита в нём, далее присваиваем значение.

Последовательность команд, реализующая мигание светодиодом будет такая:

```
while (1)
      {
        PORTD.2 = 1;
        PORTD.2 = 0;
      }
```

Однако если скомпилировать сейчас эту программу (Ctrl+F9) и запустить симуляцию в Proteus, можно увидеть беспорядочное мигание. Почему так происходит? Потому что мы меняем состояние порта ввода-вывода моментально, не даем светодиоду никакой паузы между «включено»-«выключено». Паузу можно добавить, подключив заголовочный файл с макросом, реализующим задержку:

```
#include <delay.h>
```

Сам макрос вызывается функциями:

```
delay_ms();
delay_us();
```

где в скобках указывается задержка в миллисекундах или микросекундах соответственно.

Стоит немного рассказать про реализацию этого макроса. Можно самостоятельно написать функцию, реализующую задержку. Очевидно, что зная тактовую частоту микроконтроллера, можно рассчитать, за какое время выполняется та или иная ассемблерная инструкция (информация о количестве машинных тактов, необходимых для выполнения той или иной ассемблерной инструкции находится в документации на МК). Поскольку при вызове задержки мы не

хотим ничего делать, то логично использовать NOP (это ассемблерная инструкция, которая говорит АЛУ МК не делать ничего). Теперь, если составить цикл со счетчиком, внутри которого будем вызывать NOP, то, казалось бы, всё будет хорошо. Однако на самом деле это не так. Программа при компиляции так или иначе оптимизируется. И при оптимизации пустые циклы выкинутся, поскольку они будут тормозить программу. Поэтому для того, чтобы реализовать задержку, необходимо делать какие-то осмысленные действия, которые при этом не мешали бы основной программе. К примеру, можно выполнять сложение каких-то двух чисел. И выполнять его столько раз, насколько необходима задержка. Макросы задержки именно так и делают – при их вызове учитывается тактовая частота микроконтроллера (именно поэтому она задается в настройках проекта) и создается определенное количество циклов, внутри которых выполняется сложение.

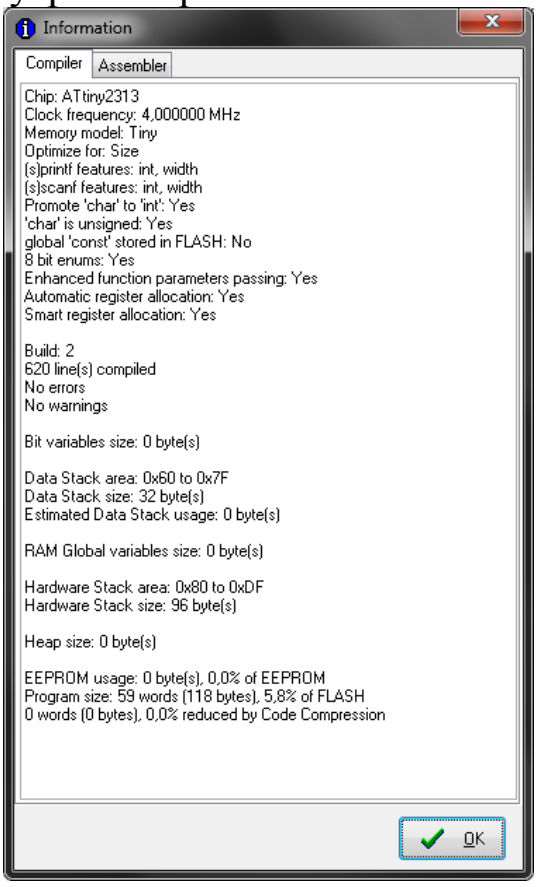

Рис.1.20. Информационное окно после успешной компиляции

Если программа, скомпилирована без ошибок, то пользователю показывается уведомляющее окно, в котором указаны параметры компилятора и параметры получившейся программы (рис. 1.20). Сама программа при этом, если открыть .hex файл выглядит набором нулей и единиц, расположенных определенным образом:

```
:0600000012C0FECFFDCF8F
:10000600FCCFFBCFFACFF9CFF8CFF7CFF6CFF5CFAE
:10001600F4CFF3CFF2CFF1CFF0CFEFCFEECFEDCFDE
:10002600F894EE27ECBBE5BFF8E1A4B7A77FA4BF21
:10003600F1BDE1BD8DE0A2E0ED938A95E9F780E898
:10004600A0E6ED938A95E9F7E0E0E3BBE4BBE5BB08
:10005600EFEDEDBFC0E800C0E0E8E6BDE0E0E6BDDC
:10006600EBBBEABBE8BBE7BBE2BBE1BBFFCFFFCF25
:00000001FF
```

В случае если при компиляции возникли какие-либо ошибки или предупреждения, компилятор уведомляет нас о несостоявшейся компиляции и выдает стандартные сообщения об ошибках в исходном коде с указанием конкретных мест и какие именно ошибки были допущены.

После выполнения лабораторной работы следует сделать выводы о проделанной работе.

ЛАБОРАТОРНАЯ РАБОТА №2

Программный способ борьбы с дребезгом контактов

Цель работы: Ознакомиться с проблемой дребезга контактов и вариантами её решения. Реализовать программный механизм борьбы с дребезгом.

Время выполнения работы – 4 учебных часа.

Задание на лабораторную работу:

1. Изучить проблему дребезга контактов.

2. Ознакомиться с вариантами решения проблемы.

3. Написать программу для микроконтроллера AVR, реализующую программный метод борьбы с дребезгом.

Методика выполнения работы:

Разместить на чертежном поле Proteus микроконтроллер ATtiny2313, выполнить его настройку, если это необходимо.

Поскольку в Proteus нет элементов (кнопок), симулирующих дребезг контактов, необходимо провести данную симуляцию самостоятельно.

Для этого будем пользоваться т.н. генератором сигналов (DPATTERN), который находится в левой панели Generator Mode. После размещения генератора на чертежном поле, зайдем в его свойства (рис. 2.1).

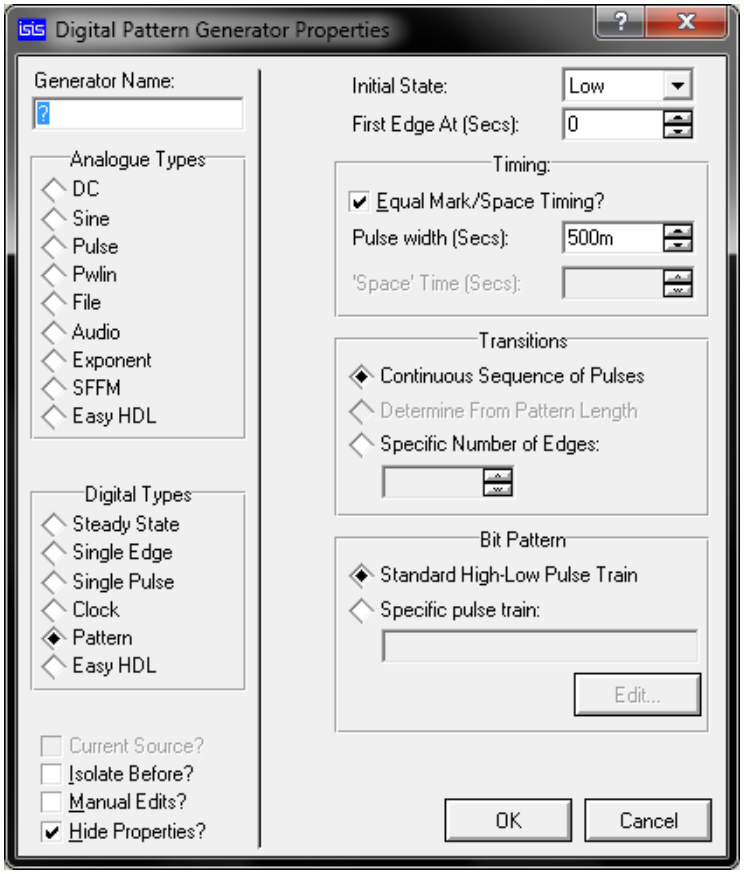

Рис.2.1. Свойства цифрового генератора сигналов

Генератор цифровой, поэтому имеет дискретные состояния

и дискретную частоту переключений (Pulse width). С учетом знаний о явлении дребезга, выставим ширину импульса. Далее следует создать последовательность дребезга, выбрав пункт Specific pulse train и нажав кнопку Edit. Перед нами откроется окно редактирования формы сигнала. Необходимо сформировать некоторую последовательность переключений (рис. 2.2). Выполним соединение генератора и МК.

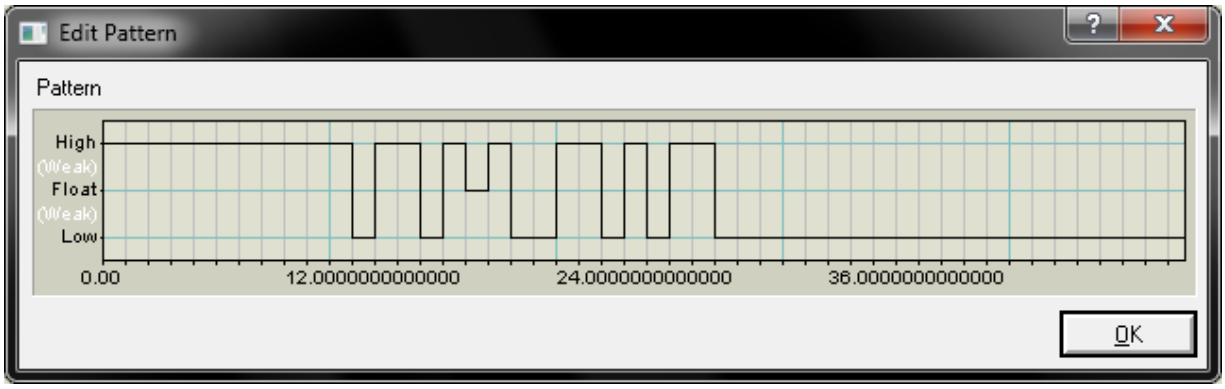

Рис. 2.2. Пример цифровой последовательности, имитирующей явление дребезга

В Proteus существует очень мощный инструмент проведения анализа схем – Graph Mode (размещен в левой панели). Для изучения реакции программы МК на входной дребезг будем использовать тип анализа DIGITAL. Для размещения анализатора необходимо выбрать указанный тип анализа, а на чертежном поле выполнить щелчок ЛКМ и растянуть окно анализатора до нужных размеров. Размещение окна на чертежном поле не принципиально, т.к. изучение сигналов всё равно проходит в отдельном окне.

Далее необходимо разместить пробники сигналов. Поскольку мы будем следить за цифровыми сигналами, т.е. фактически за напряжением, выберем в левой панели Voltage Probe Mode. Пробники размещаются так же, как и любые компоненты. После этого следует соединить выводы пробников с изучаемыми цепями. Однако размещение и соединение пробников не переместит их в окно анализа. Чтобы сигналы,

исследуемые пробниками, стали доступны в окне анализатора, необходимо выбрать пробник и перетащить его на окно анализатора (как в Windows функция Drag and Drop).

Сам анализ выполняется не в режиме интерактивной симуляции, т.е. просто запустив симуляцию (F12), мы не запустим анализ. Необходимо при остановленной симуляции выбрать окно анализатора и нажать пробел на клавиатуре. После выполнения анализа можно развернуть окно анализатора, выполнив щелчок ЛКМ по заголовку окна анализатора.

По умолчанию время анализа – одна секунда. Для изучения явления дребезга это слишком большое время. Параметры анализатора настраиваются по двойному щелчку ЛКМ внутри окна анализатора.

Следует самостоятельно составить блок–схему алгоритма подавления дребезга.

Руководствуясь составленной схемой, написать программу для микроконтроллера и, используя изученный графический анализатор, продемонстрировать правильность работы алгоритма борьбы с дребезгом (например, можно изменять состояние светодиода в зависимости от сигнала с генератора сигналов).

Необходимо помнить, что реальный уровень сигнала (лог. 0 или лог. 1) на ножке порта ввода-вывода МК отслеживается через регистр PIN (а не PORT). Доступ к битам данного регистра аналогичен доступу к битам регистра PORT.

Сделать выводы о проделанной работе.

ЛАБОРАТОРНАЯ РАБОТА №3
Вычисление параметров импульсного сигнала

***Цель*:** получить практические навыки работы с прерываниями по внешнему воздействию и получения временных параметров импульсного сигнала при помощи встроенного таймера.

***Время выполнения работы*:** 4 академических часа.

1. Организация работы с прерываниями в PIC-контроллерах

Микроконтроллеры серии PIC16FXX имеют 4 прерывания: прерывание по переполнению таймера TMR0, прерывание по изменению значения на выводах RB4-RB7, прерывание по окончанию записи в перепрограммируемое ПЗУ EEPROM, внешнее прерывание по сигналу на выводе INT(RB0).

Для всех прерываний в микроконтроллере зарезервирован 1 вектор прерывания, расположенный по адресу 0x04. Т.е. при срабатывании любого из прерываний программа автоматически перейдет на адрес 0x04 и очистит бит GIE в регистре INTCON, тем самым, запретив все прерывания. Для выхода из прерывания используется команда RETIE, которая возвращает программу на тот адрес, на котором она была в момент возникновения прерывания, и выставляет бит GIE, разрешая прерывания.

За работу прерываний отвечает регистр INTCON. Назначение битов регистра INTCON приведено в табл. 3.1. Все биты регистра INTCON доступны и для чтения, и для записи. По умолчанию все биты регистра INTCON, кроме RBIF, равны 0. Значение бита RBIF по умолчанию неопределенно. Для того, чтобы обеспечить переход программы на адрес 0x04 при срабатывании того или иного прерывания, следует записать 1 в бит GIE и биты, отвечающие за разрешение конкретных прерываний.

Для того чтобы узнать, какое из прерываний сработало, надо проверить значение флагов прерываний T0IF, INTF, RBIF.

После обработки прерываний перед вызовом инструкции RETFIE следует самостоятельно очистить флаги прерываний, тем самым, предотвратив повторное срабатывание прерываний.

2. Настройка параметров таймера TMR0 и прерывания по внешнему сигналу на выводе INT

За настройку параметров таймера TMR0 и прерывания по внешнему сигналу на выводе INT отвечает регистр OPTION. Назначение битов регистра OPTION приведено в табл.3.2.

Все биты регистра OPTION доступны как для записи, так и для чтения. По умолчанию все биты регистра OPTION равны 1.

Таблица 3.1

Назначение битов регистра INTCON

Номер бита	**Название**	**Назначение**
7	GIE	Глобальное разрешение или запрещение прерываний; 0 — все прерывания запрещены; 1 - все разрешенные битами EEIE, T0IE, INTE, RBIE прерывания разрешены.
6	EEIE	Разрешение/запрещение прерывания по окончанию записи в EEPROM; 0(1) - прерывание запрещено (разрешено).
5	T0IE	Разрешение/запрещение прерывания по переполнению таймера TMR0; 0(1) - прерывание запрещено (разрешено)
4	INTE	Разрешение/запрещение прерывания по сигналу на выводе INT(RB0); 0(1) - прерывание запрещено (разрешено).
3	RBIE	Разрешение/запрещение прерывания по изменению значений на выводах RB4-RB7; 0(1) - прерывание запрещено (разрешено).
2	T0IF	Флаг прерывания по переполнению таймера TMR0; 0 - переполнение таймера не произошло; 1 - таймер переполнен.
1	INTF	Флаг прерывания по сигналу на выводе INT(RB0); 0 - прерывание не произошло; 1 - прерывание произошло.
0	RBIF	Флаг прерывания по изменению значений на выводах RB4-RB7; 0(1) - состояние ни одного из выводов RB4-RB7 не изменилось (состояние RB4-RB7 изменилось).

В микроконтроллере PIC16F84A реализован 8-битный таймер/счетчик. Значение таймер/счетчика хранится в регистре TMR0.

Приведем возможный порядок инициализации таймера:

- установка разрешения прерывания таймера в регистре INTCON;

- установка начального значения таймера в регистр TMR0;
- установка источника тактовых импульсов и значения делителя частоты.

Таблица 3.2

Назначение битов регистра OPTION

№ бита	**Название**	**Назначение**
7	RBPU	Не используется в данной работе.
6	INTEDG	Выбор фронта срабатывания прерывания по внешнему сигналу на выводе INT(RB0); 0(1) - прерывание срабатывает по заднему (переднему) фронту сигнала.
5	T0CS	Выбор источника тактовых импульсов для таймера TMR0; 0(1) - источником тактовый импульсов служит внутренний тактовый генератор процессора (внешний сигнал на выводе T0CKI).
4	T0SE	Выбор фронта срабатывания таймера TMR0 при работе с внешним источником тактовых импульсов; 0(1) - срабатывание счетчика по переднему (заднему) фронту тактового импульса
3	PSA	Выбор таймера, которому будет присвоено значение делителя частоты; 0(1) - значение делителя частоты будет присвоено таймеру TMR0 (сторожевому таймеру WDT).
2-0	PS2-PS0	Выбор значения делителя частоты.

Выбор значения делителя частоты осуществляется при помощи установки соответствующего значения в младшую тетраду регистра OPTION (биты PSA-PS0). В таблице 3.3 приведены значения делителя частоты.

Для того, чтобы остановить таймер достаточно установить в 1 бит T0CS в регистре OPTIONS, переведя таймер в работу от внешнего источника тактовых импульсов. При записи значения в регистр TMR0 значение делителя частоты в регистре OPTION сбрасывается, однако сами установки делителя частоты остаются неизменными.

Таблица 3.3

Значения делителя частоты

Биты PSA-PS0	**Значение делителя частоты**
0000	1:2
0001	1:4
0010	1:8
0011	1:16
0100	1:32
0101	1:64
0110	1:128
0111	1:256
1000	1:1. В этом случае значение делителя частоты будет присвоено сторожевому WDT таймеру, а таймер TMR0 будет работать без делителя.

3. Работа с внешними портами ввода/вывода

Микроконтроллер PIC16F84A имеет 2 двунаправленных порта ввода/вывода: PORTA (выводы RA0-RA4) и PORTB (RB0-RB7). Некоторые из выводов портов могут быть настроены для выполнения специальных функций (например, вывод RB0 порта PORTB может быть настроен для генерации прерывания по внешнему сигналу), в этом случае вывод не будет работать как вывод порта ввода/вывода.

Для настройки режима работы каждого вывода портов PORTA и PORTB используются регистры TRISA (0x85) и TRISB (0x86) соответственно. Если в регистре TRISA (TRISB) бит **n** установлен в 1, то вывод RAn (RBn) будет работать в режиме ввода информации, если же в регистре TRISA (TRISB) бит **n** установлен в 0, то вывод RAn (RBn) будет работать в режиме вывода.

Для того, чтобы получить информацию с того или иного вывода, надо настроить его при помощи регистра TRISA (TRISB) в режим ввода информации и прочитать соответствующий бит в регистре PORTA (PORTB).

Пример 1. Для того, чтобы получить значение на выводе RA3,

следует:
1. Установить 3-й бит регистра TRISA в 1.
2. Прочитать значение 3-го бита регистра PORTA.
Для того, чтобы вывести информацию на тот или иной вывод, надо настроить его при помощи регистра TRISA (TRISB) в режим вывода информации.
Пример 2. Для того, чтобы получить значение на выводе RB5, следует:
1. Установить 5-й бит регистра TRISB в 0.
2. Записать необходимое значение в 5-й бит регистра PORTB.
По умолчанию оба порта PORTA и PORTB находятся в режиме ввода информации (т.е. все биты регистров TRISA, TRISB равны 1).

4. Задание

Необходимо написать программу формирования заданной выходной импульсной последовательности на выводе RB7 микроконтроллера. Ширина первого импульса равна ***a*** тактов, количество импульсов - ***N***, импульсы, следующие после первого, должны иметь ширину на ***b*** меньше предыдущего. Период импульсов остается постоянным и равным 256 тактам.

Таблица 3.4

Варианты задания

Вариант	N	a	b
1	4	200	30
2	5	100	10
3	3	120	30
4	4	110	25
5	5	90	5
6	3	70	20
7	4	220	30
8	5	150	20
9	4	140	25

5. Содержание отчёта

Отчёт должен содержать:
- тему и цель лабораторной работы;
- задание и номер варианта;
- блок-схему алгоритма;
- текст программы, соответствующей варианту задания, с комментариями;
- результат реализации программы - импульсная последовательность (количество и длительности импульсов в соответствии с вариантом задания).

6. Контрольные вопросы

1. Сколько и каких прерываний имеет микроконтроллер PIC16F84? Какая команда осуществляет выход из прерывания?
2. На какой адрес перейдёт программа при возникновении прерывания? Какова структура регистра INTCON?
3. Как можно узнать какое из прерываний сработало?

ЛАБОРАТОРНАЯ РАБОТА №4
Программная реализация сигнала ШИМ

***Цель*:** получение практических навыков работы с встроенным таймером/счетчиком и портом ввода/вывода.
***Время выполнения работы*:** 4 академических часа.

1. Задание

Создать программу для микроконтроллера PIC16F84 (тактовая частота - 5 МГц), реализующую ШИМ заданной частоты и скважности на выходе RB7 микроконтроллера (см. рис. 4.1).
Исходные данные приведены в табл. 4.1.

СОДЕРЖАНИЕ ОТЧЁТА
Отчёт должен содержать:
- тему и цель лабораторной работы;
- задание и номер варианта;
- блок-схему алгоритма;

- текст программы, соответствующей варианту задания, с комментариями.
- результат реализации программы – сигнал с широтно-импульсной модуляцией (параметры сигнала должны соответствовать варианту задания).

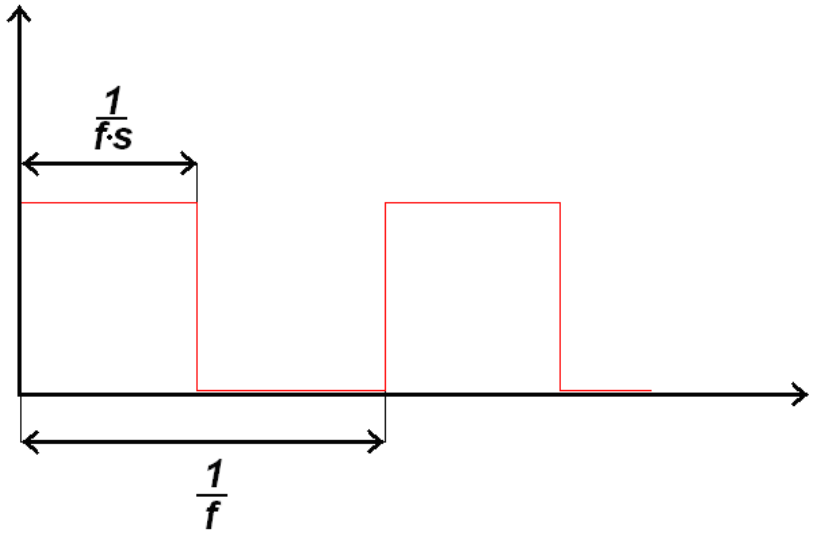

Рис. 4.1. Сигнал ШИМ

Таблица 4.1

Исходные данные

№ Варианта	Частота f	Скважность s	Вывод МК
1	610 Гц	256/10	PORTA.2
2	4,88 кГц	256/30	PORTA.1
3	19,5 кГц	256/50	PORTA.0
4	305 Гц	256/70	PORTB.2
5	2,44 кГц	256/90	PORTB.1
6	76,3 Гц	256/110	PORTB.0
7	9,7 кГц	256/130	PORTB.3
8	1,22 кГц	256/150	PORTB.4
9	152,6 Гц	256/170	PORTA.3

3. Контрольные вопросы

1. За что отвечает регистр OPTION и по какому адресу он

находится?
2. Каково назначение выбранного преподавателем бита регистра OPTION?
3. Как в программе можно инициализировать таймер?
4. Сколько портов ввода/вывода имеет микроконтроллер PIC16F84A? Какие имена и адреса они имеют?
5. Как прочитать информацию с произвольного вывода порта?
6. Как передать информацию по произвольному выводу порта?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. М.Б Лебедев. CodeVision AVR пособие для начинающих. М. Додэка, 2010.

2. Р. Токхейм. Основы цифровой электроники. М. Мир, 1988.

3. **А.В. Евстифеев.** Микроконтроллеры AVR семейств Tiny и Mega фирмы Atmel. М. Додэка, 2006.