МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра механики, мехатроники и робототехники

МИКРОПРОЦЕССОРНАЯ ТЕХНИКА В МЕХАТРОНИКЕ И РОБОТОТЕХНИКЕ

Методические указания к выполнению лабораторных и самостоятельных работ по дисциплине «Микропроцессорная техника в мехатронике и робототехнике», для студентов направления подготовки 15.03.06 «Мехатроника и робототехника»

Курск 2020

УДК 621.(076.1)

Составители: Мальчиков А.В., Яцун А.С.

Рецензент
Кандидат технических наук, доцент *Е.Н. Политов*

Микропроцессорная техника в мехатронике и робототехнике: методические указания по выполнению лабораторных и самостоятельных работ / Юго-Зап. гос. ун-т; сост.: А.В. Мальчиков; А.С. Яцун, Курск, 2020. 71 с., 45 ил.

Содержат сведения по вопросам программирования и настройке микроконтроллеров семейства AVR. Приводится пример выполнения лабораторной работы, краткие теоретические положения и контрольные вопросы для самостоятельной подготовки.

Методические указания соответствуют требованиям программы, утверждённой учебно-методическим советом по направлениям Мехатроника и робототехника.

Предназначены для студентов направлений направления подготовки 15.03.06 «Мехатроника и робототехника» всех форм обучения.

Текст печатается в авторской редакции
Подписано в печать . Формат 60x84 1/16
Усл.печ.л. Уч.-изд.л. Тираж 20 экз. Заказ .Бесплатно.
Юго-Западный государственный университет.
305040 Курск, ул. 50 лет Октября, 94

Содержание

Цель и задачи лабораторных и самостоятельных работ 4

Лабораторная работа №1. 5

Лабораторная работа №2 18

Лабораторная работа №3 34

Лабораторная работа №4 42

Лабораторная работа №5 55

Библиографический список 71

Цель и задачи лабораторных и самостоятельных работ

Целью работы является освоение студентами принципов построения и функционирования микропроцессорных устройств, основ программирования и работы с периферией микроконтроллеров, навыков использования специализированных средств разработки ПО для микропроцессорных устройств.

Предмет «Микропроцессорная техника в мехатронике и робототехнике» представляет дисциплину базовой части профессионального цикла учебного плана для направления подготовки 15.03.06Мехатроника и робототехника.

Выполнение работы ориентировано на формирование у студентов следующих элементов профессиональных компетенций:

ПК-2 - способностью разрабатывать программное обеспечение, необходимое для обработки информации и управления в мехатронных и робототехнических системах, а также для их проектирования

ПК-11 - способностью производить расчеты и проектирование отдельных устройств и подсистем мехатронных и робототехнических систем с использованием стандартных исполнительных и управляющих устройств, средств автоматики, измерительной и вычислительной техники в соответствии с техническим заданием

Лабораторная работа №1.

СРЕДА РАЗРАБОТКИ ПРОГРАММ ДЛЯ МИКРОКОНТРОЛЛЕРОВ CODE VISION AVR.

Цель работы:

1) Ознакомление с устройством и принципом работы микроконтроллера Atmega16.

2) Познакомиться с компилятором Си для AVR **CodeVisionAVR** и получить основные навыки разработки ПО для МК.

3) Закрепить знания по языку **Си для МК**.

Объект исследования: микроконтроллер Atmega16.

Аппаратные средства: виртуальная лаборатория на ЭВМ IBM PC, программный пакет CodeVisionAVR – компилятор Си для AVR микроконтроллеров.

Краткие теоретические сведения

Микроконтроллер (MCU) — микросхема, предназначенная для управления электронными устройствами. Типичный микроконтроллер сочетает в себе функции процессора и периферийных устройств, может содержать ОЗУ и ПЗУ. По сути, это однокристальный компьютер, способный выполнять простые задачи. Использование одной микросхемы, вместо целого набора, как в случае обычных процессоров, применяемых в персональных компьютерах, значительно снижает размеры, энергопотребление и стоимость устройств, построенных на базе микроконтроллеров.

Микроконтроллеры являются основой для построения встраиваемых систем, их можно встретить во многих современных приборах, таких, как телефоны, стиральные машины и т. п. Большая часть выпускаемых в мире процессоров — микроконтроллеры.

На сегодняшний день существует более 200 модификаций микроконтроллеров, совместимых с i8051, выпускаемых двумя десятками компаний, и большое количество микроконтроллеров других типов. Популярностью у разработчиков пользуются 8-битные микроконтроллеры PIC фирмы Microchip Technology и AVR фирмы Atmel, шестнадцатибитные MSP430 фирмы TI, а также

ARM, архитектуру которых разрабатывает фирма ARM и продаёт лицензии другим фирмам для их производства.

AVR это семейство МК от компании ATMEL, разработанных с учетом особенностей и удобства написания программ на языке Си.

Почему для курса выбраны именно микроконтроллеры AVR?

Это недорогие, широко доступные, надежные, простые, довольно быстро считающие, МК. Кроме того, большинство инструкций выполняется за 1 такт - т.е. при частоте 10 МГц выполняется до 10 млн. инструкций в секунду.

AVR имеют развитую периферию, т.е. набор аппаратуры окружающей процессор-вычислитель в одном корпусе МК или набор встроенных в МК электронных устройств, блоков, модулей.

Основные параметры AVR:

- тактовая частота до 20 МГц ;
- встроенный программируемый RC-генератор, частота 1, 2, 4, 8 МГц;
- Flash-ПЗУ программ, программируемое в системе, 10 000 циклов перезаписи;
- EEPROM данных (100 000 циклов) – блок, в котором сохраняется программа даже при отключении питания;
- внутреннее ОЗУ со временем доступа 1 такт;
- 6 аппаратных команд умножения (для семейства mega);
- развитая система адресации, оптимизированная для работы с С-компиляторами;
- 32 регистра общего назначения;
- синхронный (USART) или асинхронный (UART) (в mega64 и mega128 их по 2) интерфейс;
- синхронный последовательный порт (SPI);
- двухпроводной интерфейс TWI, совместимый с интерфейсом I2C;
- многоканальный PWM 8-, 9-, 10-, 16-битный ШИМ-модулятор;
- 10-битный АЦП с дифференциальными входами;
- программируемый коэффициент усиления перед АЦП 1, 10 и 200;
- встроенный источник опорного напряжения 2,56 В;

- аналоговый компаратор;
- сторожевой таймер - перезагружает МК при "зависании";
- настраиваемая схема задержки запуска после подачи питания;

Существуют AVR со встроенными интерфейсами USB, CAN и со встроенными радио приемо-передатчиками. Есть специализированные МК AVR для управления электроприводом, электродвигателями - серия AT90PWMxxxx.

Они позволяют выполнить любую задачу любительского уровня (и многие задачи профессионального уровня - это же серийный МК), они поддерживаются симуляторами электронных устройств на AVR - VMLAB или PTOTEUS (он также позволяет симулировать и другие МК семейств PIC, 8051, ARM7, Motorolla).

Также они имеют достаточно оперативной памяти и памяти для программ, имеют много выводов (ножек), что очень удобно. Они имеют встроенный RC генератор (тактирующее устройство, с частотой работы которого работает МК) и могут работать без внешнего кварцевого генератора - достаточно подать питание на новый МК и он заработает.

В рамках лабораторных работ используется один МК – Atmega16.

Выполнение работы

В рамках данной лабораторной работы напишем программу на Си для МК, целью которой является общее ознакомление с методикой работы в среде **Code Vision AVR**.

Инсталляционный пакет свободно распространяемой версии программы, рассчитанной на создание программ, результирующий код которых не превышает 2 Кбайт, можно скачать в Интернете по адресу http://www.hpinfotech.ro/html/download.htm.

Как по назначению, так и по структуре и устройству, программа Code Vision AVR очень напоминает AVR Studio. Главное отличие — отсутствие собственных средств отладки. Для отладки программ Code Vision AVR пользуется отладочными средствами системы AVR Studio.

Далее работаем по алгоритму:

- Устанавливаем компилятор Code Vision AVR в директорию по умолчанию *C:\CVAVR*

- В папке *C:\CVAVR* (где у вас должен находиться компилятор **CodeVisionAVR**) создайте папку **z1** для файлов этого проекта.

В компиляторах и симуляторах вы работаете с проектами. Проект - это некоторая совокупность файлов, относящихся к вашей задаче и располагающихся в специально созданной папке. При этом файл описания проекта обычно имеет расширение .prj

Советы:

1) Давайте папкам проектов осмысленные названия! Например: z1 - задача 1.

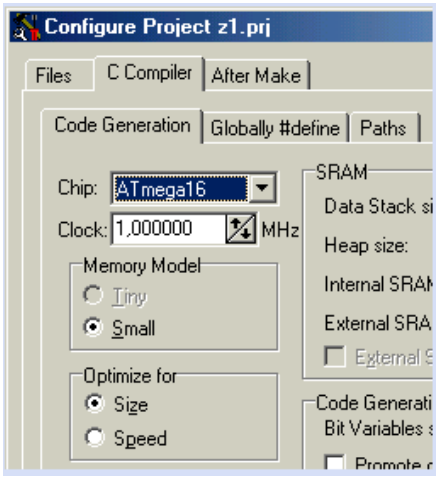
2) Удобно располагать файлы и компилятора и симулятора в одной общей папке проекта - тогда компилятор выдает выходные файлы .hex .cof __.c а симулятор имеет к ним доступ для использования в симуляции.

При этом вы одновременно запускаете на ПК и компилятор и симулятор и подправив код в компиляторе, заново компиллируете проект и, переключившись в симулятор, перестраиваете проект и запускаете симуляцию уже нового варианта прошивки.

3) Создайте в папке проекта еще одну папку, например, Files и помещайте в нее все документы, например даташиты(описания) и т.д., относящиеся к вашему проекту, но не являющиеся файлами компилятора или симулятора.

- Запустите компилятор. Для создания файла проекта нажимайте: **File -> new -> project -> ОК -> No** .
- Перейдите в созданную для проекта папку **z1** и введите в поле **"file name": z1**
- нажмите **"save"** - откроется окно конфигурации проекта.
- перейдите на закладку **"C compiler"**
- выберите МК **(Chip) ATmega16**
- установите частоту тактирования МК **(Clock) 1,0 МГц**
- убедитесь что в окне **"File Output Format(s)"** есть COF и HEX

Теперь все должно выглядеть так:



- Нажмите **ОК**.

.COF *файл содержит привязку к тексту программы на Си (в компиляторе* ***CodeVisionAVR*** *к тексту файла с расширением* ***__.c*** *- т.е. к копии а не к самому исходному файлу с текстом программы на Си - файлу* ***.c*** *) к содержимому файла прошивки МК -* ***.hex****. Эта информация позволяет при симуляции в VMLAB (или другом симуляторе) наблюдать движение программы прямо по коду на языке Си.*

Перед вами появится открытый текстовый файл **Project Notes - z1.prj** в котором вы можете записывать свои замечания и мысли по проекту.

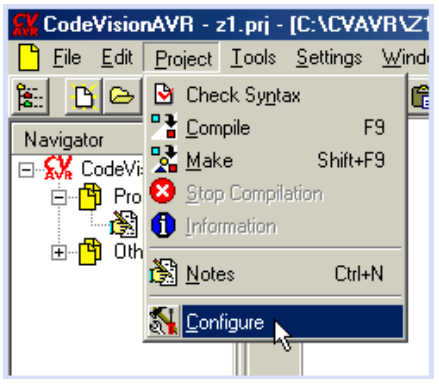
Теперь нужно создать главный для нас текстовый файл для набора исходного текста на Си - его расширение .c

- нажимайте: **File -> New -> Source -> ОК**. Появился файл **untitled.c**

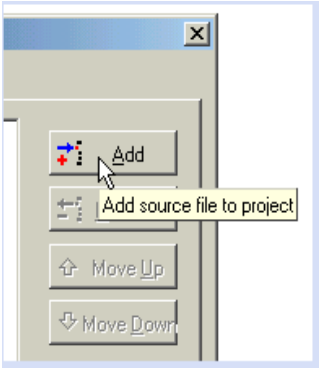
- нажимайте: **File – Save as** - введите в поле **"file name"**: **z1.c** и нажмите **Save.**

Нужно добавить созданный файл **z1.c** в список файлов проекта.

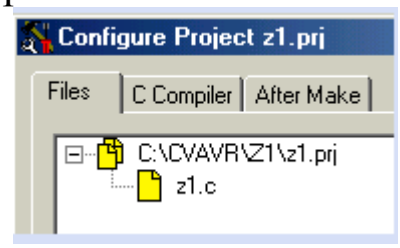
- откройте меню конфигурирования проекта: **Project -> Configure**:



В открывшимся диалоге, нужно выбрать ярлык **"Files"** и нажать кнопку **"Add"**



В новом диалоге, выберите файл "**z1.c**" и нажмите "**Open**". Теперь файл включен в проект:



- нажимайте: **ОК**

- максимизируйте (разверните) окно файла - **z1.c**

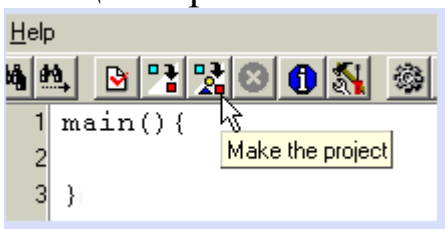
Теперь все готово к собственно программированию - т.е. к созданию текста программы на языке Си.

Написание программы на Си:

Минимальная программа на языке Си выглядит так:

```
main()  {    }
```

Вся программа состоит из одной строчки и содержит обязательную для программы на Си функцию: **main** (с англ. главная). Эта программа бесполезна, она ничего не делает, но ее можно скомпилировать, нажав кнопку - "**make the project**" выполнить полную компиляцию проекта:



Появится информационное окно и сообщит, что компиляция прошла успешно: Ни **ошибок(errors)**, ни **предупреждений(warnings)** нет.

Вы наверно догадались, что набираемый текст появляется в файле **z1.c** *и он же компилируется по кнопке* ***"make the project"***

Но что удивительно, программа уже имеет размер 92 слова или 184 байта - это составляет 1,1% объема памяти программ выбранного нами МК ATmega16.

Загляните в папку нашего проекта - z1 - в результате компиляции, там появилось много новых файлов. В процессе расширенной трансляции программы формируется не только HEX-файл, но и файл той же программы, переведенный на язык Ассемблер, а также специальный файл в COF-формате, предназначенный для передачи программы в AVR Studio для отладки. После создания и расширенной трансляции проекта его директория будет содержать файлы со следующими расширениями:

z1.hex - файл-прошивка для "загрузки" в МК

z1__.c - копия файла z1.c для симуляторов

z1.cof - информация связывающая содержимое файлов **z1__.c** и **z1.hex**.

Эти три файла удобно использовать в симуляторе **VMLAB** или **PROTEUS** и в других.

Необходимым для реального МК является лишь файл прошивки - **z1.hex**

prj — файл проекта Code Vision AVR;

txt — файл комментариев. Это простой текстовый файл, который вы заполняете по своему усмотрению;

c — текст программы на языке СИ;

asm — текст программы на Ассемблере (сформирован Code Vision);

cof — формат для передачи программы в другие системы для отладки;

eep — содержимое EEPROM (формируется одновременно с HEX-файлом);

hex — результат трансляции программы;

inc — файл-дополнение к программе на Ассемблере с описанием всех зарезервированных ячеек и определением констант;

1st — листинг трансляции программы на Ассемблере;

map — распределение памяти микроконтроллера для всех переменных программы на СИ;

obj — объектный файл (промежуточный файл, используемый при трансляции);

rom — описание содержимого программной памяти (та же информация, что и в HEX-файле, но в виде таблицы);

vec — еще одно дополнение к программе на Ассемблере содержащее команды переопределения векторов прерываний;

cwp — файл построителя проекта. Содержит все параметры, которые вы ввели в построитель

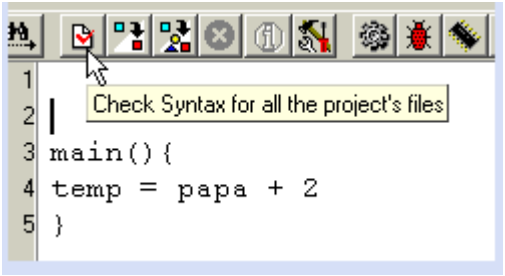
> ***В процессе написания в тексте программы неизбежно и обычно возникают ошибки.*** *Компилятор выполняет компиляцию лишь при отсутствии ошибок, предупреждения не мешают процессу компиляции.*

Давайте сделаем ошибку в тексте программы и посмотрим, как ее найти.

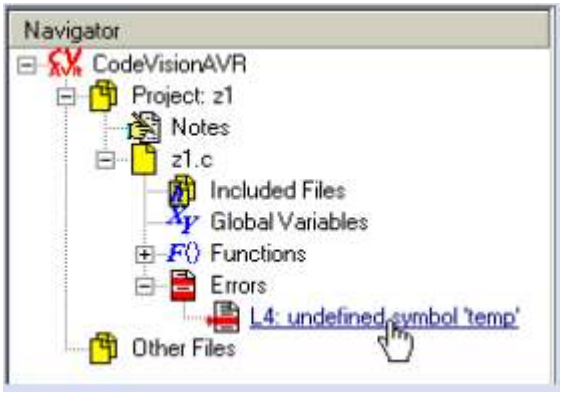
Добавим в **тело функции** (это пространство между фигурными скобками **{ }** т.е. что написано от скобки **{** до скобки **}**) **main** операцию с двумя не объявленными переменными и не поставим в конце точку с запятой:

```
main(){
temp = papa + 2
}
```

Нажмите кнопку  , чтобы компилятор проверил текст программы.



Внизу экрана в окне сообщений - **Messages** - появится сообщение только об одной ошибке. Сообщение содержит название файла и номер строки, в которой обнаружена ошибка, а так же краткое описание ошибки. Аналогичное сообщение появилось и в окне навигатора по проекту:



Если щелкнуть по ссылке, будет подсвечена строка в тексте программы, содержащая ошибку. Компилятор предупреждает об использовании в тексте программы неопределенного символа: **temp**

Разберем эту ситуацию:

Раз мы написали temp слева от знака =, значит, мы хотели присвоить temp какое-то значение, и значит temp - это переменная.

Очень важно: *В Си знак* ***= это не знак равно! Это оператор присваивания*** *значения справа от знака* **=** *тому, что стоит слева от знака. Поэтому запись:*

```
x = x + 3;
```

не лишена смысла, как это было бы в математике, а означает буквально: взять значение переменной `x`*, добавить к нему десятичное число* `3` *и записать результат обратно в переменную* `x`*.*

в Си каждая переменная должна быть объявлена перед использованием! Т.е. мы должны сообщить компилятору, что может содержать эта переменная и сколько памяти нужно зарезервировать под нее.

Подробно о переменных и типах данных вы можете прочитать в help'е компилятора - раздел ***Variables*** *и* ***Data Types****. Наиболее часто используемый в МК тип:* ***unsigned char*** *- без знаковый, символьный, - может хранить числа от 0 до 255 - это один байт или 8 бит или один регистр 8-ми битного МК AVR.*

Если переменная x была, например, объявлена так ***unsigned char*** *x; и x имела значение 254 то после строки*

```
x = x + 3;
```

ее значение станет 1 (десятичная единица), так как если прибавлять 3 раза по единице, то значение x будет меняться так:

вначале 255, затем 0, и наконец 1

Обязательно поймите это! ***Так меняются значения в регистре 8-ми битного таймера МК, когда он считает.***

Правило: ***при "переполнении" диапазона допустимых для переменной значений происходит переход от максимального значения к минимальному и затем значение увеличивается далее. т.е. 255 затем 0 затем 1 затем 2 и так далее, по кругу. В обратную сторону - аналогично!***

```
x = x + 3;
```

не лишена смысла, как это было бы в математике, а означает буквально: взять значение переменной x, добавить к нему десятичное число 3 и записать результат обратно в переменную x.

Если в свойствах проекта вы не убирали галочку:

то можете писать просто: **char** Таким образом, получаем следующий текст программы:

```
char temp;
main()
{
temp = papa + 2
}
```

Мы объявили переменную вне тела какой-либо функции. Такие переменные являются ***глобальными*** *- т.е. доступны и могут быть использованы и изменены в любом месте программы, в любой функции.*

```
x = x + 3;
```

Нажмите кнопку , чтобы компилятор снова проверил текст программы. Теперь компилятор информирует о не объявленной переменной – papa. Объявим переменную papa. Но не новой строкой, а в той же, где объявили temp. Кроме того, **инициализируем** ее, т.е. **присвоим ей значение при объявлении** - число 0xFE - это 16-ричное представление десятичного числа 254

```
char papa = 0xFE, temp;
```

В одну строку можно записать объявление нескольких переменных одного типа через запятую. После последней не забудьте поставить точку с запятой. Если при объявлении глобальной переменной не присвоить ей значение, то она будет содержать 0.

Совет: Всегда присваивайте значения переменным до их использования - чтобы точно знать, что они содержат, так как в других компиляторах правила "по умолчанию" могут быть иными!

Теперь обе наши переменные без знаковые, символьные, при этом

temp содержит число 0

papa содержит (хранит) значение 254.

Нажмите кнопку Компилятор сообщает нам о пропущенном символе ;

Обратите внимание: *Компилятор указывает на строку ниже, чем та, где мы забыли поставить точку с запятой.*

Знак ; нужно поставить после числа 2 - вот так:

```
char papa = 0xFE, temp;
main(){
temp = papa + 2;
}
```

Теперь проверка программы не выдает сообщений об ошибках и можно выполнить полную компиляцию программы, нажав кнопку:

Наконец в информационном окне мы видим, что ошибок нет и размер программы стал 99 слов или 1.2% памяти программ

ATmega16. Программа полностью готова для «прошивки» в реальный МК.

Требования к отчету

Отчет должен содержать следующие разделы:

1. Титульный лист
2. Цель работы.
3. Краткое описание микроконтроллера Atmega 16.
4. Краткое описание среды CodeVision AVR.
5. Выполнение работы (с текстом программы).
6. Выводы.

Контрольные вопросы для самостоятельной работы

1. Что такое микроконтроллер?
2. Назовите основные составляющие архитектуры микроконтроллера?
3. Назовите основные преимущества микроконтроллеров AVR?
4. Для какой цели используется кварцевый генератор. На каких частотах он работает?
5. Сколько регистров общего назначения есть в Atmega16?
6. Что такое EEPROM. Какой его размер?
7. Сколько таймеров есть у Atmega16. Для каких целей они предназначены?
8. Какие интерфейсы связи с внешними устройствами существуют для Atmega16?
9. Какие возможности есть для работы с аналоговым сигналом?
10. Для чего используется функция Reset?
11. Что такое компилятор?
12. Какие среды для разработки программ для микроконтроллеров существуют?
13. Какие файлы создаются при компиляции проекта в среде Code Vision AVR?
14. Что содержат файлы с расширениями .cof, .c, .hex, .asm?
15. Что такое тело функции?

16. Опишите возможности интерфейса среды Code Vision AVR?

17. Какие существуют типы переменных для программирования МК?

18. Как происходит инициализация переменных?

19. Чем отличаются глобальные переменные от локальных?

20. Какие ограничения существуют при написании программ для МК?

Лабораторная работа №2
ОРГАНИЗАЦИЯ СВЕТОДИОДНОЙ ИНДИКАЦИИ С ИСПОЛЬЗОВАНИЕМ МИКРОКОНТРОЛЛЕРА ATMEGA

Цель работы:

1) Ознакомление с принципом работы портов микроконтроллера.
2) Написать программу, организующую работу двух светодиодов от МК Atmega16.
3) Познакомиться со средой симуляции программ **Proteus.** Осуществить симуляцию программы

Объект исследования: микроконтроллер Atmega16.

Аппаратные средства: виртуальная лаборатория на ЭВМ IBM PC, программный пакет CodeVisionAVR – компилятор Си для AVR микроконтроллеров, программный пакет Proteus – среда симуляции работы микроконтроллеров и других электронных устройств.

Краткие теоретические сведения

В данной лабораторной работе организуется работа двух светодиодов от МК Atmega16. При этом, один светодиод горит при наличии питания на МК, второй – работает по алгоритму, задаваемому пользователем.

Таким образом, в этой работе рассматриваются алгоритмы вывода сигнала на порты МК. Порт в МК – это 8 ножек или линий ввода-вывода (выводов МК или IO или I-O или I/O), имеющие индивидуальные номера от 0 до 7 и общую букву A, B, C, D, …, отличающую этот порт от других.

Каждый порт в МК имеет 3 сопоставленных ему регистра. Например, порт B:

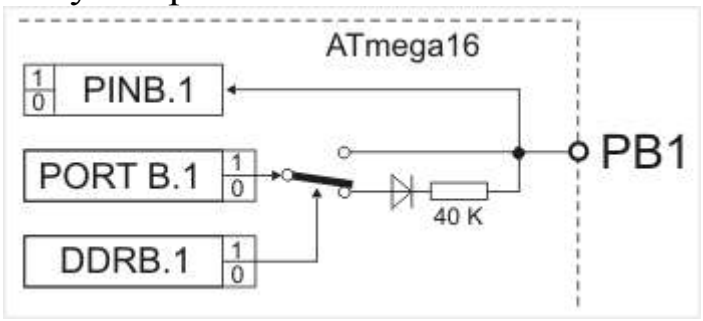
DDRB – значение битов в этом регистре определяет чем будет ножка этого порта с номером этого бита – начальное (при включении МК или после сброса) значение «0» – **ножка вход,** если сделать бит = «1» (говорят: **установить бит** англ. **Set bit**), то эта **ножка станет выходом**. Сделать бит = «0» – говорят: **сбросить или очистить бит** англ. **Clear bit**

PINB - биты этого регистра показывают чем («1» или «0») считает МК напряжение на ножке порта с номером этого бита.

(**этот регистр в Atmega16 нужно только читать**, записывать в него что-либо бесполезно).

PORTB – бит этого регистра нужно сделать «1» или «0», чтобы на ножке порта с номером этого бита появилась «1» или «0». При этом такой же бит регистра **DDRB** должен быть «1» – т.е. ножка должна быть выходом. Если она сконфигурирована как вход (т.е. её бит в регистре **DDRB** очищен или равен нулю) - то если очищен и соответствующий бит в регистре **PORTB** ножка будет высокоимпедансным входом (Z-состояние, вход с очень высоким входным сопротивлением более 10 Мом), а если бит в регистре **PORTB** установлен, т.е. равен «1» то включается «подтяжка» (**pull-up**) высокоимпедансного входа к плюсу питания МК через встроенный резистор примерно 40 КОм – ножку как-бы соединяют таким резистором с питанием МК.

На рисунке представлена упрощенная (без учета дополнительных функций этого вывода) схема вывода PB1, поясняющая логику его работы:



Напряжение на выводе **PB1** с задержкой в 1.5 такта преобразуется в логические уровни «1» или «0», которые можно прочитать в регистре **PINB** это бит_1 или **PINB.1** в **CVAVR**

Бит_1 в регистре **DDRB** управляет переключателем – на рисунке переключатель показан в положении бит_1 равен «0». Диод на схеме идеальный – значит если бит_1 в **PORTB** будет тоже равен «0» то вывод **PB1** будет высокоомным входом.

А если бит_1 в **PORTB** сделать «1» то вывод **PB1** через диод и резистор 40 Ком подключится к питанию МК – т.е. станет входом с подтяжкой.

Если бит_1 в регистре **DDRB** сделать «1» переключатель изменит состояние и значение бит_1 в **PORTB** будет выводится прямо на **PB1** – теперь это будет просто выход.

Таблица состояния ножки МК

Значение бита_x			Состояние вывода МК
Программа может только читать этот бит ! (Отличия для некоторых МК описаны выше)	Программа управляет этими битами		PBx
PINB.x	DDRB.x	PORTB.x	
1	1	1	Высокий лог. уровень (вывод **как бы** подсоединен к питанию МК резистором около 20 Ом)
0		0	Низкий лог. уровень (вывод **как бы** заземлен резистором около 20 Ом)
определяется только реальным напряжением на ножке МК ! Напряжение преобразуется в "1" или "0" по приведенным выше правилам.	0	1	Подтяжка - pull-up. Вывод **как бы** подсоединен к питанию МК через резистор 40 КОм
		0	Z-состояние. ВысокоОмный вход

Таким образом, 32 ножки IO микроконтроллера Atmega16 могут быть программно и индивидуально сконфигурированы (и переконфигурированы по мере необходимости) как:

1) **входы** с высоким (более 10 МОм) входным сопротивлением (для напряжений от 0 до напряжения питания МК) или **Z-вход**

2) **входы** по п.1), но с подключенным внутренним подтягивающим резистором на + питания МК (номинал резистора примерно 40 кОм)

3) как **выходы** способные обеспечить ток до 20 мА (но общий ток на порт только до 80 мА, а ток всех портов до 200 мА в DIP корпусе)

Важно: *эти 32 ножки МК имеют и дополнительное функциональное назначение описанное в ДШ и книгах - они являются входами-выходами (пишут: IO или I/O) и для устройств периферии МК. И при активации какого-либо периферийного устройства МК программой, соответствующие ножки МК автоматически конфигурируются так, как требуется для правильной работы этого устройства независимо от того, как они были сконфигурированы ранее. А после отключения устройства их конфигурация станет такой, что была задана последней по тексту программы.*

Выполнение работы

Рассмотрим, как программно реализовать конфигурацию портов МК, а также как осуществить вывод сигнала на ножки МК.

Разработку любого электронного устройства нужно **начинать с постановки задачи**. Это делают в виде ТЗ - технического задания. Сформулируем техническое задание для нашего устройства:

Разработать устройство на микроконтроллере ATmega16, которое будет осуществлять переключение светодиода с интервалом в 1с. Устройство питается постоянным стабилизированным напряжением от 3,3 до 5,5 вольт. Тактирование МК осуществляется от кварцевого резонатора с частотой 4 МГц. Для индикации наличия питания на МК используется светодиод. Диоды подключаются от ножек порта **A** через токо-ограничительные резисторы к земле МК.

Выполнение работы начинаем с создания схемы, которая физически может выполнить то, что написано в ТЗ.

1) Кварц на 4 МГц подключаем к ножкам МК XTAL1 и XTAL2 и эти ножки заземляем конденсаторами по 22 пФ на ножку GND номер 11.
2) Делаем цепь сброса МК - RESET. Для этого ножку 9 заземляем конденсатором 0.1 мкФ на ножку GND номер 11.
3) Ножки 30 и 32 соединяем с ножкой 10.
4) Ножку 31 с ножкой 11.
5) Питание МК подается так: 0 на ножку 11 и +5 вольт на ножку 10.
6) Поставьте блокировочный конденсатор 0,1 мкФ между ножками 10 и 11 для фильтрации возможных помех.
7) Светодиоды подключаются черточками к ножкам МК 39 и 40. К другим выводам светодиодов подсоединяются резисторы от 470 до

750 Ом - их тоже будет 2 - вторые выводы резисторов подсоединяются к земле МК - ножка 11. В создаваемых схемах я рекомендую всегда использовать один светодиод для индикации наличия питания на МК, обычно его подключают к 40 ножке МК (или к PortA.0).

Следующим этапом создания устройств на МК является **создание алгоритма программы**, который заставит работать созданную схему, выполняя ТЗ.

Алгоритм - это описание последовательности и логики шагов (действий) приводящих к нужному результату. Он не зависит от языка программирования и может быть записан :

- графически, с помощью специальной программы, либо от руки.

- псевдокодом, т.е. обычным текстом.

Мы напишем алгоритм в виде псевдокода.

а) старт программы

б) сделать ножки МК, к которым подключены светодиоды выходами.

в) послать на выход МК 40 единицу (организуем индикацию питания МК).

г) послать на выход МК 39 единицу (в этом пункте и далее организуем работу управляемого диода, диод начинает гореть)

д) ждать 1с

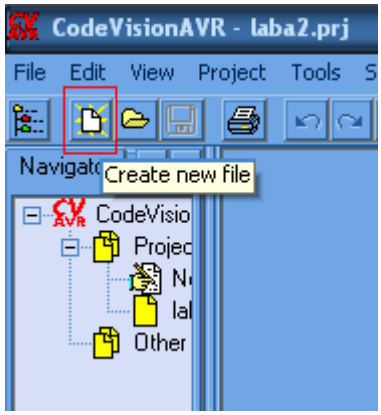
е) послать на выход МК 39 ноль (диод перестает гореть)

ж) ждать 1с

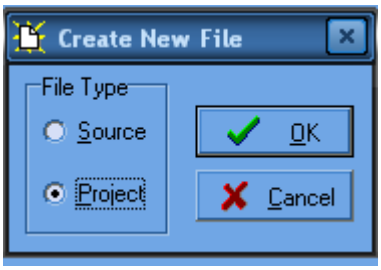
з) перейти к пункту г)

Далее нужно **записать алгоритм на выбранном языке программирования**. Мы будем использовать **Си для МК**.

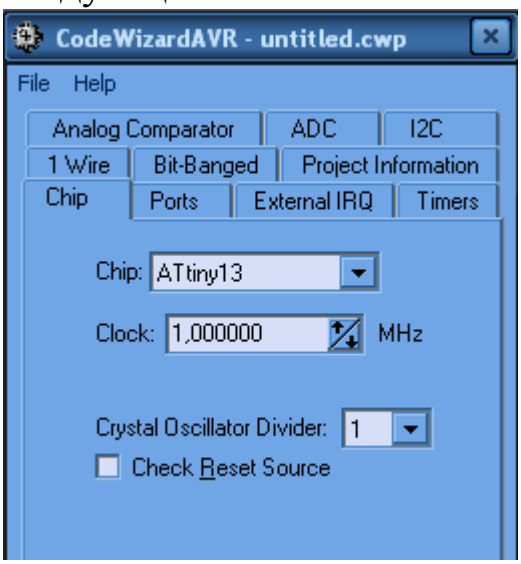
Создадим новую папку для нашего проекта C:\cvavreval\Laba2. Далее открываем CodeVision. Создаем новый проект, нажимаем



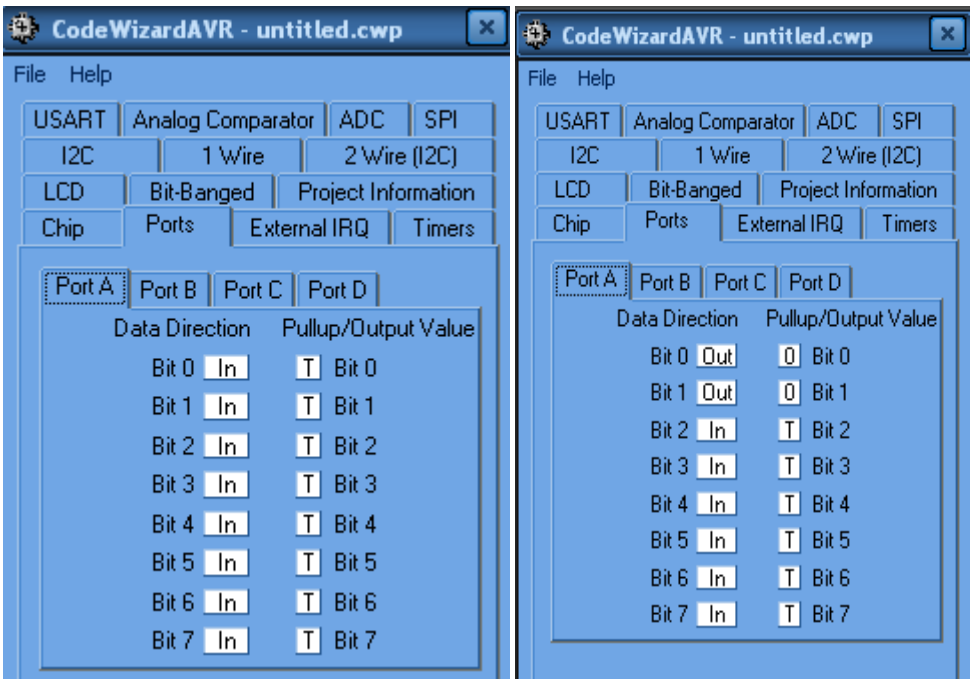
В следующем диалоговом окне выбираем **Project** и нажимаем **Ok.**



Далее вам предлагается использовать генератор начального кода **CodeWizardAVR**, нажимаем **YES.** При написании программ для МК очень удобно использовать генератор начального кода. Он позволяет очень просто сконфигурировать МК под задачи, реализующиеся в программе. Итак, если все сделано правильно, то перед вами появилось следующее окно.



Изменяем тип МК (**chip)** на Atmega16, и выбираем частоту квантования (**clock**) 4МГц. Далее переходим на вкладку порты (**ports**). Как мы видим, все порты по умолчанию назначены входами. В нашей задаче нам нужны 2 выхода – 40ая и 39ая ножки МК или биты нулевой и первый порта А (**PortA.0, PortA.1**), поэтому назначаем соответствующие биты порта А выходами. Для этого кликаем на надписях **In** напротив нужных битов, видим, что при этом **In** меняется на **Out.**



Таким образом, мы определили, что в нашей программе будет использован МК Atmega16, программа в нем будет выполняться с частотой 4МГц, все ножки МК, кроме 39 и 40 являются высокоомными входами, ножки же 39 и 40 – выходы.

Далее нажимаем **File - > Generate, Save and Exit.** В появившемся окне выбираем папку для сохранения программы и называем нашу программу. При этом сохраняются и генерируются 3 типа файлов, соот-но и сохранять каждый раз придется 3 раза.

Максимизируйте окно фала с расширением .c. Вы видите что часть программы уже сгенерирована. Вначале фиолетовым цветом написаны комментарии. Здесь описывается генератор начального кода, название проекта, тип используемого устройства, частота квантования и т.д. Все готово для написания нашей программы. Будем двигаться по созданному нами алгоритму.

а) старт программы В МК программа стартует автоматически после подачи питания и наличии "1" на ножке RESET. Т.е. на Си ничего специального писать не надо. Но мы напишем то, с чего обычно начинается программа на Си. **А программа на Си начинается** с директив препроцессора (это тот кто готовит текст программы к компиляции). Например, в нашем случае:

```
24  #include <mega16.h>
25
26  // Declare your global variables here
27
```

#include<> - означает, что вместо <....> препроцессор должен подставить текст файла - **mega16.h** - в нем описаны регистры МК и

именно ATmega16, чтобы компилятор знал их физические адреса в МК.

После знака // в Си идут комментарии. В компиляторе при генерации создаются строки с такими рекомендациями, например, данная строка рекомендует определить глобальные переменные вашей программы ниже нее. Пользоваться такими комментариями очень удобно.

Помимо этого, я рекомендую вам оставлять свои комментарии по мере создания программ, например, прописывать какая переменная за что отвечает. При создании сложных программ затем будет легко вспомнить, что выполнялось в том или ином участке программы.

Если вы, вдруг, решили поменять тип МК или частоту квантования, то это можно сделать, щелкнув ***Project - > Configure***

Следующий шаг алгоритма:

б) сделать ножки МК, к которым подключены светодиоды, выходами.

Рассмотрим основную процедуру (**main**). Начинается она с инициализации портов. Нас интересует порт А, видим, что для него определено следующее состояние регистров:

```
PORTA=0x00;
DDRA=0x03;
```

Обратимся к таблице истинности регистров порта (см. в краткой теории), видим, что в нашем случае регистр PortA содержит 0, соответственно все биты этого регистра (ножки МК с 33 по 40) могут быть либо высокоомными входами, либо выходами с выходом низкого логического уровня. Регистру DDRA присвоено значение 0x03. Таким образом в два бита данного регистра записана 1, т.к. 3 в двоичной системе это 11, если бы мы хотели сделать, например, еще бит2 выходом, то нужно было бы записать в этот регистр число 7, т.к. 7 в двоичной системе это 111 и т.д.

У всех остальных портов в регистры PortX и DDRX записаны нули. Таким образом, эти порты определены как высокоомные входы.

Далее идет инициализация таймеров и различных аппаратных функций МК. О них будет идти речь в последующих лабораторных работах.

Затем идет интересующая нас процедура – процедура **while (1) { }** Код в этой процедуре выполняется пока есть питание МК.

Пункты в) и г) алгоритма сходны по написанию в Си, поэтому объединим их. Вначале подаем напряжение на выход 40, напомню, что это вывод индикации питания.

```
PORTA.0=1;
```

Т.е. в этой строке нулевому биту порта А или 40й ножке, если смотреть по распиновке присваивается единица. Аналогично для 39ой ножки

```
PORTA.1=1;
```

д) ждать 1с В Code Vision AVR предусмотрена стандартная функция для организации временной задержки с задаваемым интервалом задержки – функция delay. Она записывается так:

```
delay_ms(1000);
```

Delay – это собственно сама функция, через нижнее подчеркивание _ указываются единицы измерения, в нашем случае это миллисекунды, но также в этой функции могут использоваться микросекунды(us). В скобках указывается временной промежуток задержки (1000). 1000мс=1с, что и нужно нам по алгоритму. Также для использования этой функции к проекту нужно прикрепить файл с описанием данной функции. Делается это сразу после строчки, в которой в проект включался файл с описанием нашего МК следующим образом:

```
#include <mega16.h>
        #include <delay.h>
// Declare your global variables here
```

е) послать на выход МК 39 ноль(диод перестает гореть). Как выполнить данный пункт, я думаю, уже понятно:

```
PORTA.1=0;
```

Т.е. мы посылаем 0 в бит порта А, соответствующий 39 ножке МК или биту1 порта А.

ж) ждать 1с. Выполнение этого пункта аналогично пункту д) строчку можно просто копировать, как в офисных приложениях.

з) перейти к пункту г) Для выполнения данного пункта ничего писать в программе не нужно, после выполнения последней строки программы в теле функции while(1), программа

автоматически вернется к первой строке тела этой функции. Таким образом получили следующий листинг программы:

```
#include <mega16.h>
        #include <delay.h>
void main(void)
{
PORTA=0x00;
DDRA=0x03;
PORTB=0x00;
DDRB=0x00;
PORTC=0x00;
DDRC=0x00;
PORTD=0x00;
DDRD=0x00;
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
MCUCR=0x00;
MCUCSR=0x00;
TIMSK=0x00;
ACSR=0x80;
SFIOR=0x00;

while (1)
      {
PORTA.0=1;
PORTA.1=1;
delay_ms(1000);
PORTA.1=0;
delay_ms(1000);
      };
}
```

Здесь специально нет комментариев, чтобы показать только нужные строки программного кода. При этом наша программа занимает лишь около 10 % от участка программы, сгенерированного генератором начального кода и занимает всего 6 строчек.

Моделирование

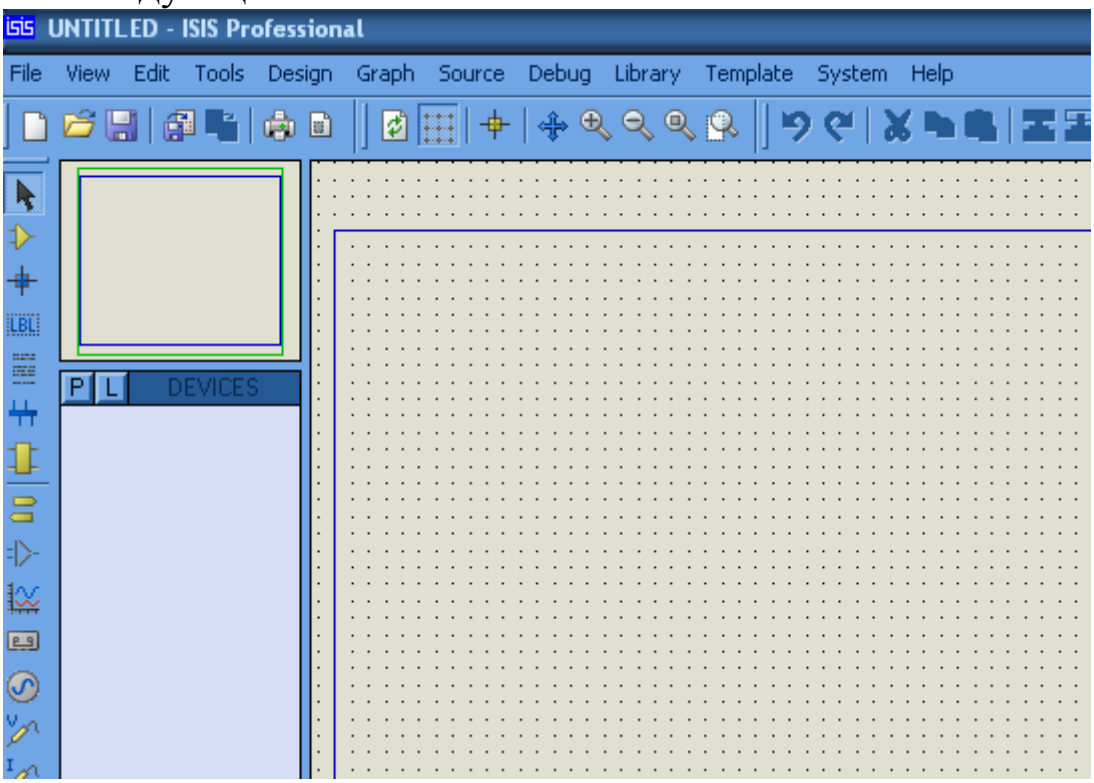
В данном разделе выполнения лабораторной работы рассмотрим вопросы, связанные с моделированием программ для МК.

При создании различных электронных, да и других устройств, перед воплощением устройства на практике всегда полезно смоделировать его работу на компьютере. При создании механических устройств для этого может использоваться CosmosExpress или CosmosMotion, при создании САУ – это пакет Matlab и т.д. В случае электронных устройств и устройств с

использованием МК будем использовать пакет Proteus – очень мощная программа для симуляции различного рода электроники.

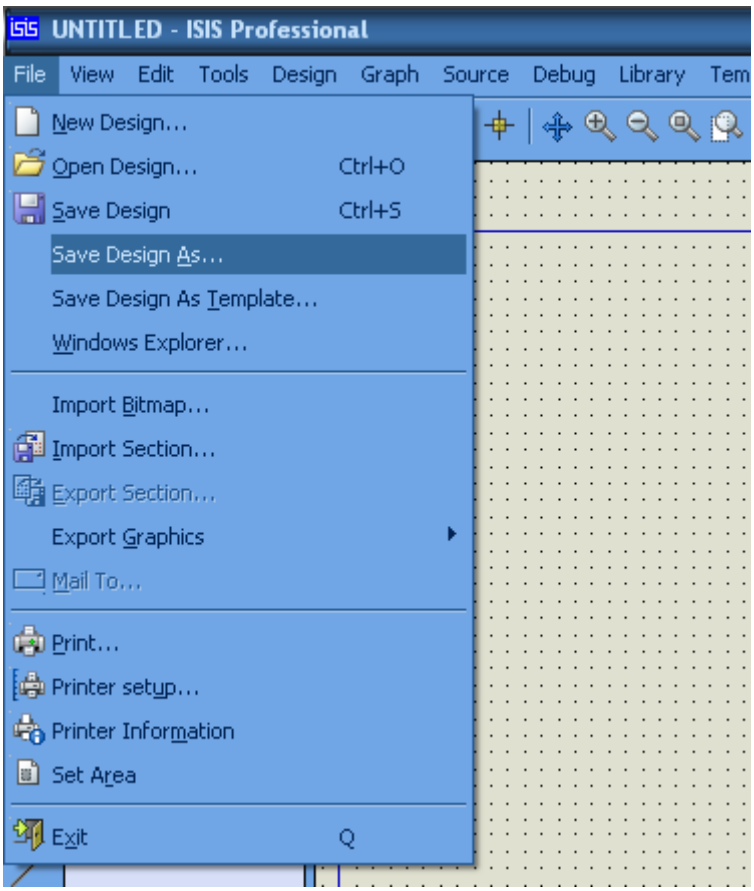
Произведем моделирование работы только что созданной программы.

Откроем **Proteus ISIS Professional** (*C:\Program Files\Labcenter Electronics\Proteus 7 Professional\BIN\ISIS.EXE*)**.** Перед вами появится следующее окно:

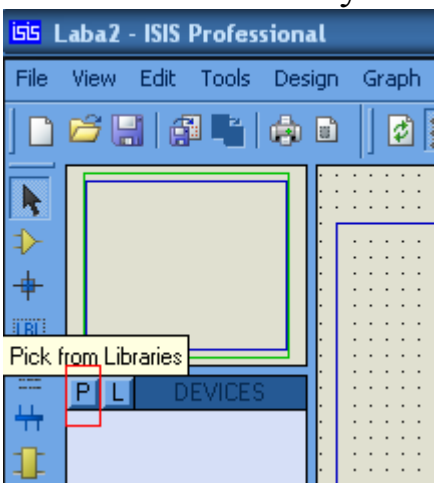


Перед работой над новым проектом рекомендуется сразу сохранить проект, причем сохранять лучше в папке с вашей программой, тогда не возникнет проблем при переносе файлов на различных носителях

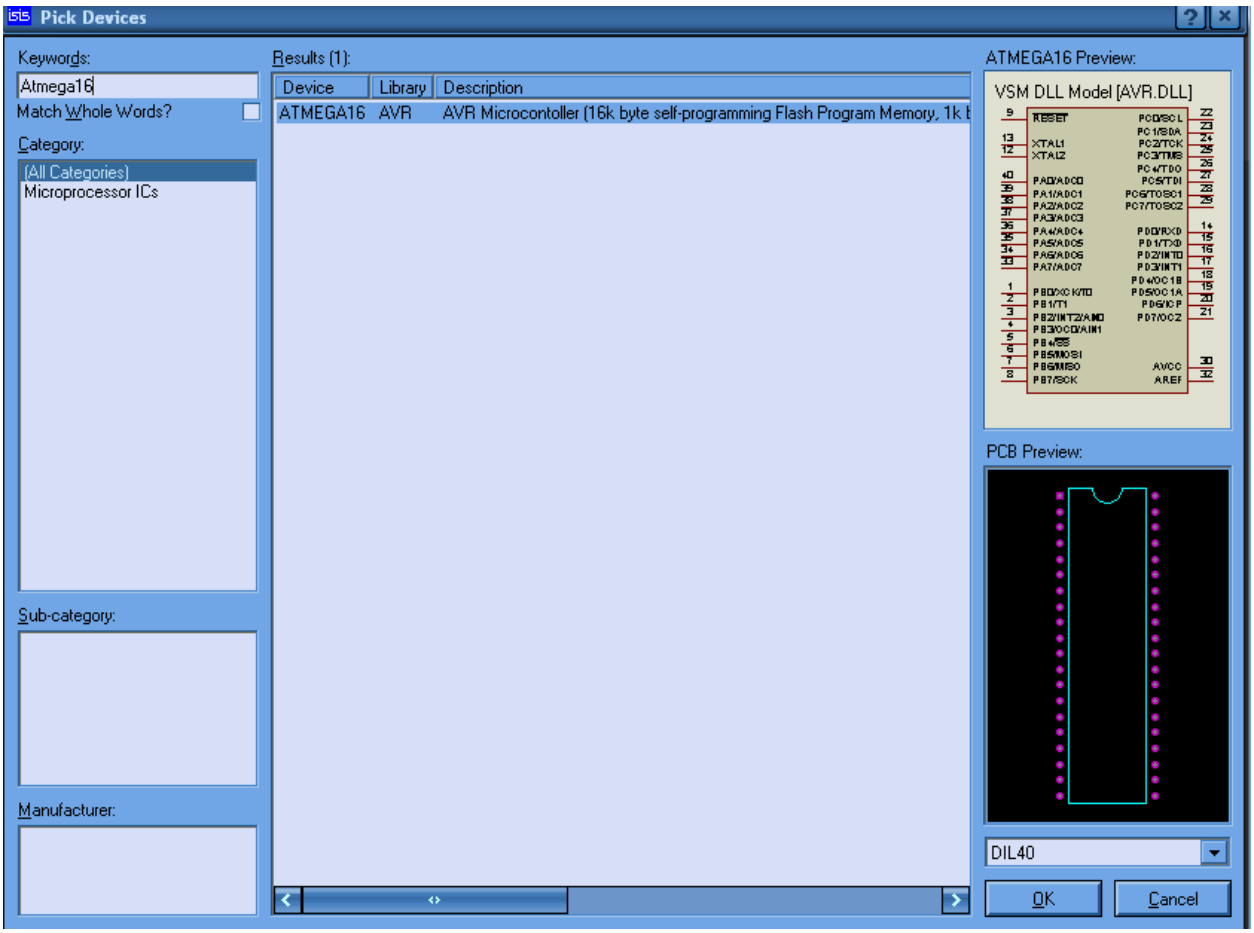
Нажимаем **File - > Save design as** и сохраняем проект в папке, в которой хранится созданная нами программа.



Теперь в пустую форму проекта нужно добавить нужные нам компоненты. Для этого нажимаем на кнопку **Pick from libraries**

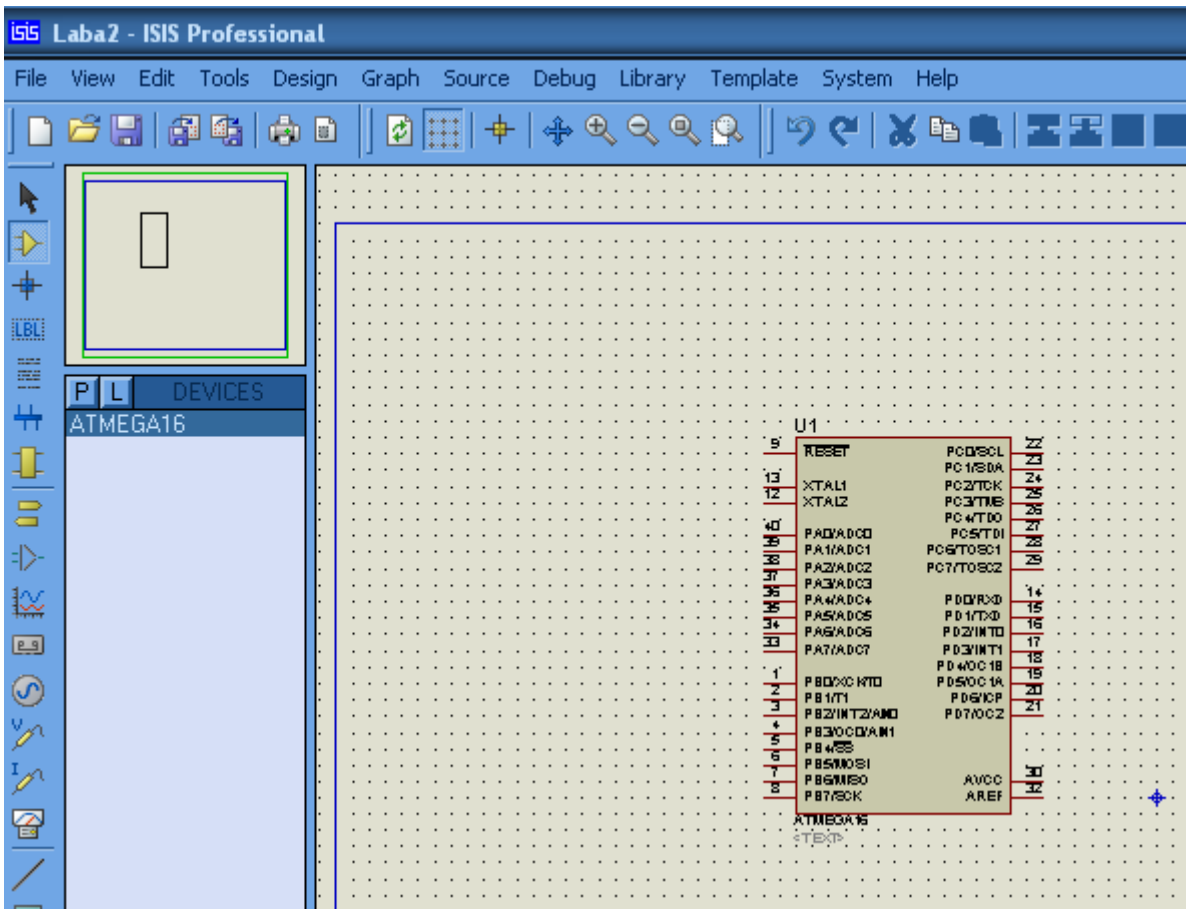


В появившемся окне под названием **Pick Devices** выбираем нужные нам компоненты. Выбор можно осуществлять либо по названию схемы или компонента, либо пользуясь готовыми разделами библиотеки. Наберем в строке **keywords** название нашего МК **Atmega16.**



В результате видим справа вверху представление этого МК как модели Proteusa и ниже тип корпуса и эскиз реальной схемы. Нажимаем Ок.

Теперь нажав на любое место проекта один раз, видим расположение компонента, нажав еще раз, располагаем компонент в удобном для нас месте, например, вот так:



Определимся с теми компонентами, которые нам еще нужны для схемы. Это 2 наших диода, токоограничительные резисторы

Задания на выполнение лабораторной работы

Написать программу, реализующую следующее задание и произвести ее моделирование:

1 группа Организовать индикацию питания на 6 ножке порта A. Организовать включение управляемого светодиода на 1с и выключение на 0,5мс в цикле на 0 ножке порта D.	**2 группа** Организовать индикацию питания на 4 ножке порта C. Организовать включение управляемого светодиода на 500 мс и выключение каждые 3с в цикле на 1 ножке порта B.
3 группа Организовать индикацию питания на 2 ножке порта A. Организовать включение управляемого светодиода на 300мс и выключение каждые 300мс в цикле на 1 ножке порта A.	**4 группа** Организовать индикацию питания на 7 ножке порта D. Организовать включение управляемого светодиода на 2с и выключение каждую 1с на 5 ножке порта C.
5 группа Организовать индикацию питания на 4 ножке порта B. Организовать включение управляемого светодиода на 3с и выключение каждые 1,5с в цикле на 3 ножке порта D.	**6 группа** Организовать индикацию питания на 5 ножке порта D. Организовать включение управляемого светодиода на 5с и выключение каждые 0,7с в цикле на 2 ножке порта A.

Требования к отчету

Отчет должен содержать следующие разделы:

7. Титульный лист
8. Цель работы.
9. Описание принципов работы с портами микроконтроллера Atmega 16.
10. Краткое описание среды Proteus ISIS.
11. Выполнение работы (с текстом программы и результатами моделирования).
12. Выводы.

Контрольные вопросы для самостоятельной работы

1. Что такое порт микроконтроллера? Сколько портов ввода-вывода у МК Atmega16?
2. Сколько регистров сопоставлено с каждым портом? Для чего они предназначены?
3. Объясните логику работы выводов МК.
4. Какие состояния выводов существуют?
5. Как осуществляется подтягивание сигнала? Для чего оно предназначено?
6. Опишите интерфейс и принцип работы с генератором начального кода.
7. Опишите структуру программ для МК.
8. Для чего предназначена функция *include*?
9. Как происходит инициализация портов МК?
10. Как реализуются временные задержки в МК?
11. Как управлять выводом сигнала с МК?
12. Какие среды моделирования электронных схем существуют?
13. Опишите интерфейс среды Proteus Isis.
14. Как осуществляется симуляция работы микропроцессорных устройств?

Лабораторная работа №3
ОРГАНИЗАЦИЯ ПРЕРЫВАНИЙ ПРИ РАБОТЕ МИКРОКОНТРОЛЛЕРА

Цель работы:

1. Ознакомление со способами организации и обработки прерываний при работе МК, закрепление навыков программирования МК.
2. Ознакомление с навыками генерации начального кода и отладки программ в пакетах «CodeVisionAVR» и «Proteus», создание программ, реализующих и обрабатывающих программные и аппаратные прерывания.
3. Осуществление симуляции работы микропроцессорной системы программ в среде «Proteus».
4. Осуществить запись (прошивку) программ в реальный МК, также провести демонстрацию работы программы на реальной схеме.

Объект исследования: микроконтроллер ATMega 16.

***Аппаратные средства*:** виртуальная лаборатория на ЭВМ IBM PC: программные пакеты «Proteus», «CodeVision AVR». МК ATMega 16, кнопки SWT, светодиоды blueLED.

Краткие теоретические сведения

Внешние электрические сигналы: это напряжения и токи поступающие к МК от подключенных к нему проводниками других компонентов электронного устройства.

Важнейший из них - это *напряжение питания МК*.

МК AVR серии ATMega могут работать, т.е. исполнять заложенную в них программу уже при подаче одного напряжения питания, а узнать о том, что он работает, можно по изменению тока потребляемого МК по проводу питания.

Диапазон допустимых напряжений питания указан на первой странице DataSheeta и составляет обычно 3,3 – 5,5 В постоянного напряжения - плюс которого подключается к выводам VCC МК.

Отрицательный вывод источника питания подключается к выводам МК GND и его потенциал принимается за ноль вольт и

относительно него измеряются все другие напряжения на ножках МК.

Проводник, соединенный с выводами GND МК, называют общим или нулевым или "земля" и на схеме обозначают специальным символом - например, жирной горизонтальной черточкой или несколькими горизонтальными полосками друг под другом убывающей длины.

Электрические сигналы это токи и вызываемые их протеканием напряжения. Но говоря о сигналах, поступающих в МК, мы рассматриваем их как, некоторые напряжения, измеряемые относительно ножек GND МК.

Важно: *В цифровой технике приняты некоторые правила по которым можно представить аналоговый сигнал, допустимый для подачи на ножку МК (он должен быть выше -0.5 В и ниже чем напряжение питания МК увеличенное на 0.3 - 0.5 В) как 1-битный цифровой сигнал или как одно из двух значений:*

"1" - высокий логический уровень (ВЛУ) - логическая единица

или

"0" - низкий логический уровень (НЛУ) - логический ноль.

Важный вывод - любое изменение напряжения на ножке МК лежащее между двумя пороговыми напряжениями не ведет к изменению того каким логическим уровнем считает МК напряжение на этой ножке в данный

Другие важные для работы МК внешние сигналы:

1) *сигнал сброса RESET* - при "0" на этой ножке МК останавливает выполнение программы, содержимое регистров МК становится начальным, а все выводы становятся высокоомными входами (говорят: Z - состояние).

После появления на этой ножке "1" и наличии питания МК - выполнение программы начнется с начала, как после включения питания МК.

2) *питание аналоговой части* МК, АЦП (входы ADC_x) ножка AVCC - ее нужно соединить с выводом VCC питания МК даже если вы не предполагаете использовать АЦП.

3) *опорное напряжение для* АЦП (входы ADC_x) ножка AREF - напряжение на ней должно быть от 2 В до напряжения питания МК. 4) ножки для подключения кварца или керамического резонатора XTAL1 XTAL2.

Прерывания в AVR.

Interrupts - прерывания, очень важная и мощная функция МК AVR ATmega и ATtiny.

Иногда требуется максимально быстрая реакция программы на какие-то события. Например, приход данных на USART или завершение АЦП или изменение уровня на ножке МК подключенной к контактному датчику или переполнение таймера.

Быструю реакцию обеспечивает механизм прерываний. Группа команд, выполняемых в ответ на запрос прерывания, называется подпрограммой обработки прерывания. Прерывание (если оно разрешено) вызывается в нормальном потоке программы сразу по окончании исполнения текущей команды. Логика прерывания заносит содержимое всех регистров в стек и, таким образом, их состояния могут быть восстановлены по завершении прерывания. Прерывание завершается командой возврата из прерывания, которая извлекает из стека и переписывает ранее сохраненное содержимое регистров и, таким образом, состояние процессора вновь становится таким, каким оно было до начала процедуры обслуживания прерывания.

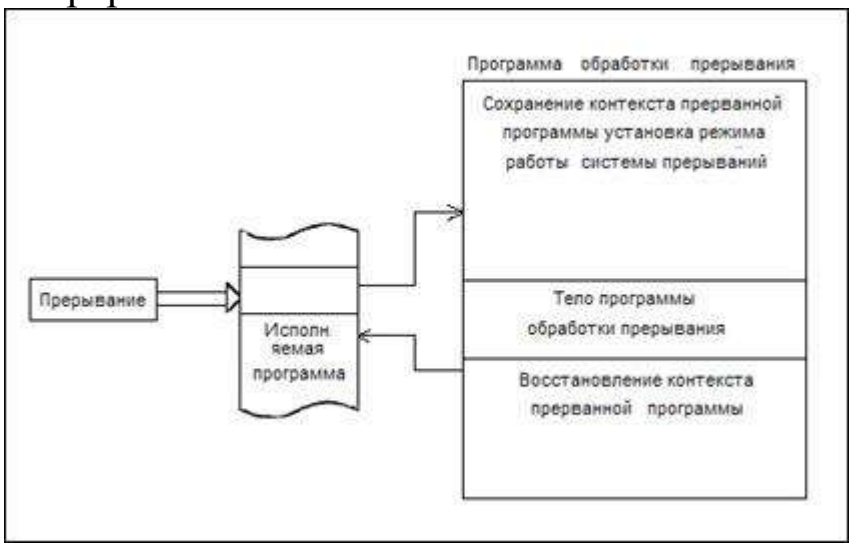


Схема принципа обработки прерываний.

Запомните: *При возникновении события которое может вызвать разрешенное в данный момент времени прерывание (список таких событий в даташите в разделе Interrupts таблица "Reset and Interrupt Vectors") и при ГЛОБАЛЬНОМ разрешении прерываний (бит 7 в регистре SREG "установлен"), выполнение программы МК останавливается, сохраняются (запоминаются) место остановки и некоторые нужные данные, бит7 в регистре SREG обнуляется, очищается флаг сработавшего прерывания и затем происходит вызов и выполнение функции обработчика данного прерывания.*

От момента наступления события до начала выполнения функции обработчика прерывания проходит не менее 4 тактов процессора. Таким образом, скорость реакции на прерывание напрямую зависит от частоты, на которой работает МК!

Если программа находится в функции обработчике прерывания и в этой функции не был установлен бит SREG.7 то другие события вызывающие прерывания не могут уже вызвать прерывание программы и

В конце функции обработчика прерывания компилятор ставит инструкцию RETI после выполнения которой бит 7 в регистре SREG становится "1" - т.е. прерывания опять ГЛОБАЛЬНО разрешаются и программа продолжает выполняться с того места где она была в момент возникновения прерывания.

НО! Если при глобальном разрешении прерываний обнаружится установленный флаг разрешенного прерывания, то будет вызвана функция обработчик этого прерывания.

Такая ситуация может возникнуть если во время выполнения обработчика прерывания возникло другое прерывание - т.е. установился его флаг. Если возникнет несколько разрешенных прерываний одновременно, то первым будет выполняться, то которое выше в списке векторов прерываний МК в ДШ. Соответственно по мере отработки накопившихся и разрешенных прерываний их флаги будут очищаться.

А вот флаги неразрешенных прерываний не очистятся, пока программа этого не сделает записью в них числа 1.

Вы можете запрещать и разрешать как все прерывания сразу, так и каждое по отдельности. Все сразу - изменяя бит7 в регистре SREG вот такими строчками в компиляторе CodeVisionAVR:

#asm("sei") /* бит_I сделать "1" теперь разрешенные прерывания будут обрабатываться, если есть установленный флаг прерывания, то произойдет вызов его функции обработчика */

#asm("cli") /* бит_I сделать "0" запретить все прерывания ГЛОБАЛЬНО. */

Прерывания легко настроить интерактивно с помощью мастеров начального кода компиляторов CVAV.

В МК AVR и других, прерывания могут возникать по многим событиям:

- изменение уровня на некоторых ножках МК,
- "0" на некоторых ножках МК,
- переполнение таймеров,
- "насчитывание" таймером определенного значения,
- завершение АЦП преобразования,
- изменение уровня на выходе компаратора,
- события в USART,
- другие события (Для ATmega16 прерывания перечислены в таблице 18 "Reset and Interrupt Vectors")

Порядок и методика выполнения работы

1) <u>Обработка внутренних прерываний</u>

Входы внешних прерываний INT0 INT1 INT2 микроконтроллера (МК) ATmega16 подключить через резисторы по 4,7 КОм к источнику питания микросхемы (+5В) для создания внешней подтяжки.

К этим же выводам подключить кнопки (button) коммутирующие землю.

Для индикации режима работы микроконтроллера подсоединим светодиод к нулевой ножке порта А.

Получили схему, показанную на рис.

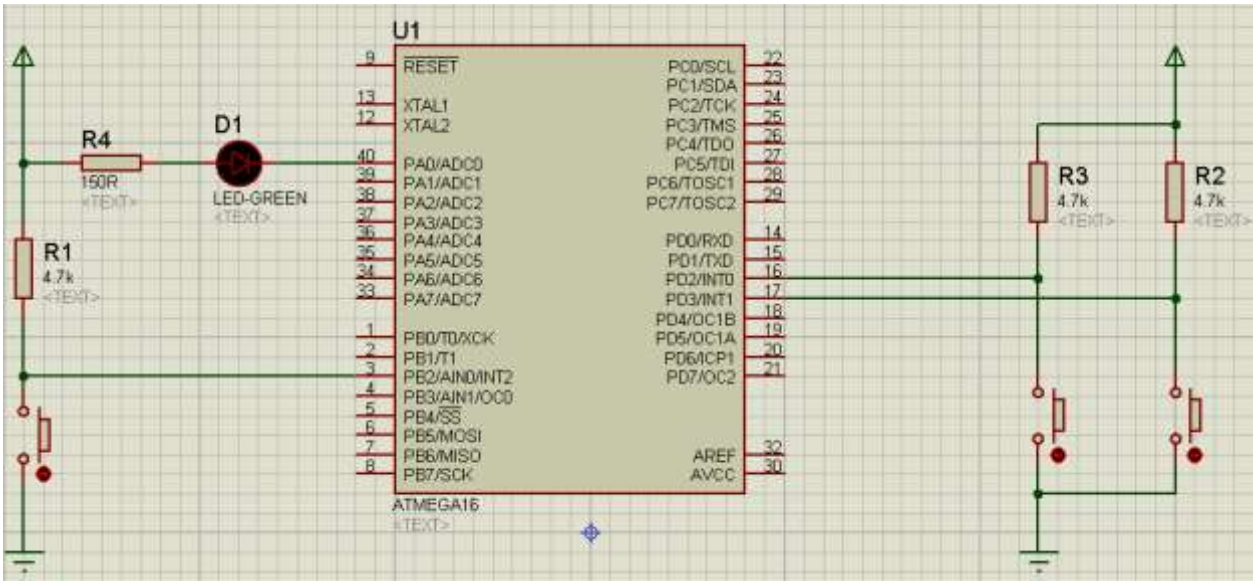


Схема подключения микроконтроллера в окне программы Proteus

Нажимая кнопку мышкой во время моделирования, можно замыкать соответствующую ножку на "землю" создавая на ней "0" на время пока кнопка нажата. Когда кнопки не нажаты на ножках "1".

Сконфигурировать прерывания с помощью мастера генератора начального кода CVAVR.

Прерывания INT0 INT1 разрешены и сконфигурированы "по любому изменению уровня" - т.е. прерывание может возникать и по фронту ("0" -> "1") и по спаду ("1"->"0") сигнала на ножке PD2 и PD3 соответственно.

Прерывание INT2 не сконфигурировано - оно оставлено "по-умолчанию" т.е. отключено.

После сохранения начального кода, напишите программу обработки прерываний, согласно варианту (табл).

Варианты задания для обработчиков прерываний

1 группа Организовать индикацию питания на 4 ножке порта C. Организовать включение управляемого светодиода на 0 ножке порта D: Однократно 1 сек - по прерыванию INT0 Трехкратно на 0.5 сек - по прерыванию INT1	**2 группа** Организовать индикацию питания на 4 ножке порта C. Организовать включение управляемого светодиода на на 1 ножке порта B. Однократно 3 сек - по прерыванию INT0 Двукратно на 0.5 сек - по прерыванию INT1
3 группа Организовать индикацию питания на 2 ножке порта A. Организовать включение управляемого светодиода на 1 ножке порта A: Двукратно на 0,3 сек - по прерыванию INT0 Трехкратно на 0.5 сек - по прерыванию INT1	**4 группа** Организовать индикацию питания на 7 ножке порта D. Организовать включение управляемого светодиода на 5 ножке порта C. Двукратно на 1 сек - по прерыванию INT0 Однократно на 2 сек - по прерыванию INT1
5 группа Организовать индикацию питания на 4 ножке порта B. Организовать включение управляемого светодиода на 3 ножке порта D: Двукратно на 3 сек - по прерыванию INT0 Однократно на 1 сек - по прерыванию INT1	**6 группа** Организовать индикацию питания на 5 ножке порта D. Организовать включение управляемого светодиода на 2 ножке порта A: Трехкратно на 1 сек - по прерыванию INT0 Однократно на 2 сек - по прерыванию INT1

Загрузив программу в микроконтроллер в программе Proteus, выполните симуляцию.

Проверьте правильность выполнения подпрограмм прерываний.

Если при выполнении обработчика одного прерывания вызвать второе, то они будут выполнятся последовательно.

Согласно теории прерываний по завершении текущей обработки прерывания INT0 должны произойти еще по 1 вызову обработчиков прерываний INT0 и INT1 - причем сейчас МК не "знает" какое из них случилось первым и значит, будет обрабатывать их по порядку перечисления в таблице 18 и datasheet.

Накопление необработанных прерываний крайне нежелательно, так как МК "не помнит" последовательность возникновения соответствующих событий!

Отчет должен содержать следующие разделы:

13. Титульный лист
14. Цель работы.
15. Краткое описание работы с прерываниями в Atmega 16.
16. Выполнение работы (с текстом программы).
17. Выводы.

Контрольные вопросы для самостоятельной работы

1. Что такое прерывание при работе микроконтроллера?
2. Какие бывают типы прерываний?
3. Как вызвать внешнее прерывание микроконтроллера?
4. Для чего используется таймеры в микроконтроллерах?
5. Почему нажатие и отпускание кнопки K0 вызвало две обработки прерывания INT0, а от кнопки K1 только одно?
6. Назовите основные случаи применения прерываний.
7. Какой алгоритм процедуры обработки прерывания?
8. В каком приоритете обрабатываются прерывания?

Лабораторная работа №4

АППАРАТНАЯ РЕАЛИЗАЦИЯ ШИРОТНО-ИМПУЛЬСНОЙ МОДУЛЯЦИИ НА ОСНОВЕ МИКРОКОНТРОЛЛЕРА ATMEGA16

Цель работы:

1) Ознакомление с принципами ШИМ управления.

2) Написание программы, организующей аппаратную реализацию ШИМ на основе МК Atmega16.

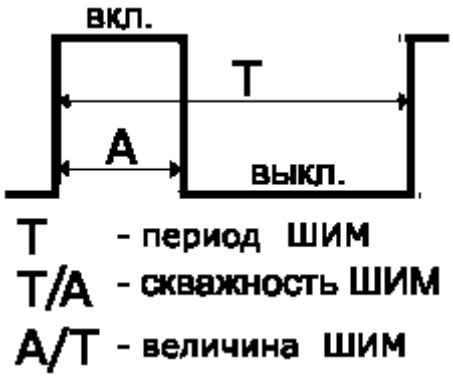
Объект исследования: микроконтроллер Atmega16.

Аппаратные средства: виртуальная лаборатория на ЭВМ IBM PC, программный пакет CodeVisionAVR – компилятор Си для AVR микроконтроллеров, программный пакет Proteus – среда симуляции работы микроконтроллеров и других электронных устройств.

Краткие теоретические сведения

ШИМ - это широтно-импульсная модуляция. т.е. модуляция (управление) напряжением или током путем изменения ширины импульсов при неизменной их величине.

На экране осциллографа **ШИМ** сигнал выглядит следующим образом:



Это практически цифровой сигнал. Он имеет 2 состояния - либо включено (на ножке МК это лог. "1"), либо выключено (на ножке МК это лог. "0"). Для создания **ШИМ** сигнала используются различные ключи – например, встроенные в МК или внешние транзисторы или реле.

ШИМ (англ. **PWM**) сигнал имеет следующие основные параметры:

- **период ШИМ** - это время между фронтами (или

спадами) соседних импульсов - обозначается T - обычно он постоянен по времени. С периодом связана обратная величина - частота ШИМ равная 1 / T в Гц.

- **величина ШИМ** - это отношение A / T умноженное на 100 - получаем проценты (англ. X % duty cycle).

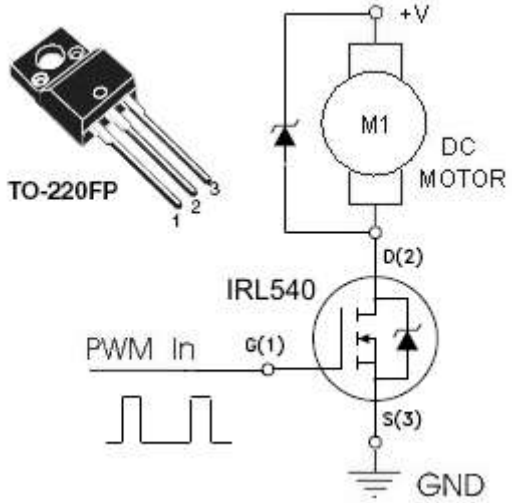
- **размах ШИМ** - это разность между значениями вкл. и выкл. Значение выкл. может быть и ненулевым.

Применение **ШИМ** позволяет:

1) регулировать мощность в нагрузке.

Регулирование мощности осуществляется изменением среднего времени подачи питания в нагрузку. При этом коммутирующий (включающий - выключающий) нагрузку транзисторный ключ работает в ключевом режиме и поэтому на нем выделяется минимум тепла.

Вот пример регулирования мощности в электродвигателе:



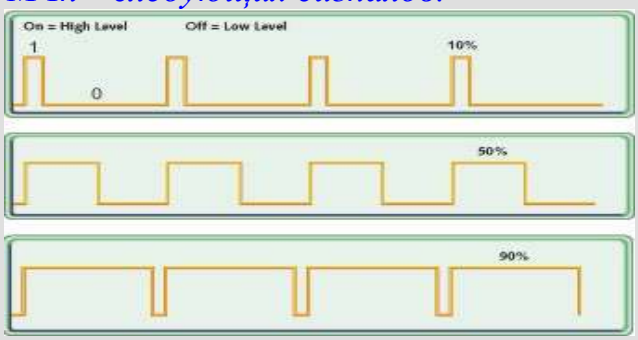
Мотор подключается к питанию +V когда напряжение "PWM In"
будет достаточным для открывания транзистора IRL540. Мощные полевые и IGBT транзисторы правильно переключать с помощью специальных драйверов – например, IRS2110.

Если частота переключений невысокая (до 2-3 КГц), то можно управлять полевыми транзисторами MOSFET серии IRL ножкой МК через резистор 100 Ом, но лучше использовать транзистор IRLZ44 .

Управляя полевым транзистором одной ножкой МК, можно переключать ток 50 -100А. Вместо электродвигателя может

быть другой тип нагрузки, например, лампа, нагревательный элемент или смешанная нагрузка - т.е. комбинация R, C и L.

Пример: Если взять электромотор с номинальным напряжением 6 вольт и подключить его по схеме, описанной выше, к питанию 12 вольт, то при подаче на "PWM In" следующих сигналов:



мотор будет развивать такую мощность:

- для верхней диаграммы ШИМ 10% - мощность будет 20% (от номинальной, при 6 вольтах).

- для средней диаграммы ШИМ 50% - мощность будет 100%

- для нижнего графика ШИМ 90% - значит мощность мотора превысит в 1.8 раз номинальную (в таком режиме мощность двигателя превышает номинальную почти в 2 раза и при таком режиме велика вероятность скорого выхода двигателя из строя).

2) **ШИМ** позволяет выполнить цифро-аналоговое преобразование, т.е. с помощью **ШИМ** можно выводить аналоговый сигнал. Нужно лишь добавить **ФНЧ** - фильтр низких частот.

ФНЧ может быть простейшим - к выводу МК, на который выводится **ШИМ** подключается резистор, а другой вывод резистора заземляется конденсатором - на этом конденсаторе будет результат ЦА преобразования PWM сигнала. Но лучше использовать более надежный фильтр низких частот – **ФНЧ** на основе операционных усилителей.

AVR ATmega16 имеет аппаратную возможность формировать 4 **ШИМ** сигнала на ножках **OC0 OC1A OC1B OC2.**

AVR ATmega48 -88 -168 имеют 6 аппаратных **ШИМ** сигналов с частотой до 78 КГц.

AVR ATmega128 имеет 6 аппаратных PWM с разрешением до 16 бит.

В данной работе, при организации аппаратного **ШИМ** сигнала, используются таймеры. Разберем, что это такое и для чего

они служат.

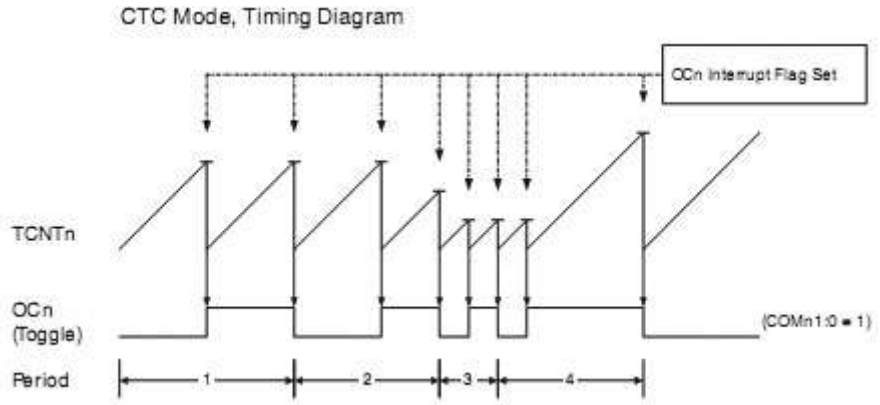
Таймер (от англ. Timer) — средство обеспечения задержек и измерения времени средствами микроконтроллера.

Существуют два вида таймеров:

- Аппаратные таймеры функционируют независимо от центрального процессора и в момент срабатывания посылают прерывание.
- Программные таймеры реализуются за счёт выполнения в цикле заданного количества одинаковых «пустых» операций. При фиксированной частоте работы процессора это позволяет точно определять прошедшее время. Главными минусами такого метода являются: зависимость количества итераций цикла от типа и частоты процессора, невозможность выполнения других операций во время задержки.

В МК **ATMega16** есть три таймера/счетчика – два 8-битных (**Timer/Counter0, Timer/Counter2**) и один 16-битный (**Timer/Counter1**). Каждый из них содержит специальные регистры, одним из которых является счетный регистр **TCNTn** (n – это число 0, 1 или 2). Каждый раз, когда процессор выполняет одну команду, содержимое этого регистра увеличивается на единицу (либо каждые 8, 64, 256 или 1024 тактов). Потому он и называется счетным. Помимо него, есть еще и регистр сравнения - **OCRn (Output Compare Register)**, в который можно записать какое-либо число. У 8-битного счетчика эти регистры 8-битные. По мере выполнения программы, содержимое **TCNTn** растет и в какой-то момент оно совпадет с содержимым **OCRn**. Тогда (если заданы специальные параметры) в регистре флагов прерываний **TIFR** (**Timer/Counter Interrupt Flag Register**) один из битов становится равен единице и процессор, видя запрос на прерывание, сразу же отрывается от выполнения бесконечного цикла и идет обслуживать прерывание таймера. После этого процесс повторяется.

Ниже представлена временная диаграмма режима **CTC (Clear Timer on Compare).** В этом режиме счетный регистр очищается в момент совпадения содержимого **TCNTn** и **OCRn**, соответственно меняется и период вызова прерывания.



Это далеко не единственный режим работы таймера/счетчика. Можно не очищать счетный регистр в момент совпадения, тогда это будет режим генерации широтно-импульсной модуляции. Можно менять направление счета, т. е. содержимое счетного регистра будет уменьшаться по мере выполнения программы. Также возможно производить счет не по количеству выполненных процессором команд, а по количеству изменений уровня напряжения на «ножке» **T0** или **T1** (режим счетчика), можно автоматически, без участия процессора, менять состояние ножек **OCn** в зависимости от состояния таймера. Таймер/Счетчик1 умеет производить сравнение сразу по двум каналам – A или B.

Как настроить таймеры на тот или иной режим работы будет рассмотрено ниже в разделе выполнение работы.

Выполнение работы

Вначале сформулируем задачу, которую мы хотим решить и напишем **ТЗ**. Разработать программу для микроконтроллера **ATmega16**, которая реализует **ШИМ** управление в следующем режиме: 2с **ШИМ** сигнал с величиной **ШИМ** 1%, 2с **ШИМ** сигнал величиной 25%, 2с – с величиной 50%, 2с – с величиной 75%, 2с – 99%. Разрядность **ШИМ** - 1024**.** Снимаемый **ШИМ** сигнал можно просматривать, подсоединяя соответствующий вывод МК к осциллографу

Следующим этапом создания устройств на МК является **создание алгоритма программы**, который заставит работать созданную схему, выполняя ТЗ.

Мы напишем алгоритм в виде псевдокода.

а) старт программы

б) сделать ножку МК, к которой подключен светодиод выходом.

в) сделать ножки МК, с которых снимаем ШИМ сигнал выходами.

г) сконфигурировать таймер1 МК в режим ШИМ сигнала, разрядностью 1024.

д) послать на выход МК 40 единицу(организуем индикацию питания МК).

е) организовать ШИМ сигнал с величиной ШИМ 1%

ж) ждать 2с

з) организовать ШИМ сигнал с величиной ШИМ 25%

и) ждать 2с

к) организовать ШИМ сигнал с величиной ШИМ 50%

л) ждать 2с

м) организовать ШИМ сигнал с величиной ШИМ 75%

н) ждать 2с

о) организовать ШИМ сигнал с величиной ШИМ 99%

п) ждать 2с

р) перейти к пункту д)

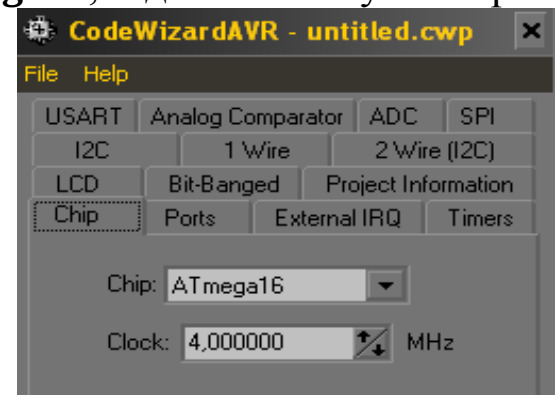
Далее нужно записать алгоритм на выбранном языке программирования. Мы будем использовать **Си для МК**.

Создадим новую папку для нашего проекта *C:\cvavreval\Laba4*. Далее открываем **CodeVision**. Создадим новый проект, при этом воспользуемся генератором начального кода.

В появившемся окне генератора начального кода настроим наш МК в соответствии с требованиями программы.

Далее действуем по следующему алгоритму:

1) На вкладке **Chip** выбираем микроконтроллер **Atmega16**, задаем частоту тактирования **4МГц**.



2) На вкладке **Ports** задаем **бит 0** порта **A** выходом (этот выход используется для индикации питания **МК**), также выходом назначаем **бит 5** порта **D** (этот выход используется для вывода ШИМ сигнала).

3) На вкладке **Timers** выбираем **Timer1** задаем **clock value 4000кГц**, выбираем режим (**mode**) **Fast PWM top=03FF** (в этой же вкладке можно задавать и другие режимы работы таймера, например, менять режим ШИМ сигнала, организовывать работу с таймером определенной частоты и т.д.)**.** Задаем выход A (**Out A**) как неинвертированный (**Non-Inv**) выход ШИМ.

4) Далее генерируем и сохраняем файл (**File -> Generate, Save&Exit**)

Далее будем следовать созданному нами алгоритму.

Пункт (а) – старт программы. Программа в **МК** начинает выполняться при наличии питания **МК**, единицы на ножке **Reset.** Таким образом, написания специального программного кода для старта программы для **МК** не требуется.

Объединим пункты (б) и (в) в один. Для их выполнения также не требуется написания программного кода. Эти этапы были реализованы при конфигурировании портов в генераторе начального кода. Чтобы задать те или иные биты портов МК выходами, нужно записать соответствующие значения в регистры **PORTx** и **DDRx.** В нашем случае, в регистр **DDRA** записали число **0x01,** в регистр **DDRD 0x20** (запись 0x01 означает число 1 в шестнадцатиричной форме или 00000001 в двоичной, соответственно для числа 0x20 представление в двоичной форме имеет следующий вид: 00100000).

Пункт (г) – сконфигурировать Таймер1, … Для реализации этого этапа алгоритма также не требуется написания программного кода. Он также был выполнен при генерации начального кода.

```
66 // Timer/Counter 1 initialization
67 // Clock source: System Clock
68 // Clock value: 4000,000 kHz
69 // Mode: Fast PWM top=03FFh
70 // OC1A output: Non-Inv.
71 // OC1B output: Discon.
72 // Noise Canceler: Off
73 // Input Capture on Falling Edge
74 // Timer 1 Overflow Interrupt: Off
75 // Input Capture Interrupt: Off
76 // Compare A Match Interrupt: Off
77 // Compare B Match Interrupt: Off
78 TCCR1A=0x83;
79 TCCR1B=0x09;
80 TCNT1H=0x00;
81 TCNT1L=0x00;
82 ICR1H=0x00;
83 ICR1L=0x00;
84 OCR1AH=0x00;
85 OCR1AL=0x00;
86 OCR1BH=0x00;
87 OCR1BL=0x00;
```

Строки 66-77 – комментарии, указывающие какой таймер и в каком режиме используется, какие выходы используются. Далее идет конфигурация таймера под нашу задачу. Изменять состояние таймера можно меняя значения в соответствующих регистрах (подробнее смотрите в даташите МК).

Пункт (д) – послать на ножку 40 **МК** единицу. С этого пункта начинается собственно написание самой программы для **МК**. Как организовать вывод сигнала на ножки МК, подробно было рассмотрено в предыдущих лабораторных работах. Для нашего случая это будет единственная строка

```
PortA.0=1;
```

Пункт (е) организовать ШИМ сигнал с величиной ШИМ 1%. В рамках данного пункта рассмотрим, как же организуется ШИМ управление. Чтобы выполнить данный пункт нужно записать следующий программный код:

```
OCR1AH=0x00;

OCR1AL=0x0A;
```

Таким образом, чтобы изменить величину **ШИМ** нужно изменить значение в регистре **OCR1A**. В нашем случае частота тактирования **МК 4МГц**, режим **ШИМ 03FF,** что означает, что разрядность **ШИМ 1024**. Т.е. частота импульсов в нашем случае 4000000/1024≈3,9кГц. Изменять частоту импульсов ШИМ можно, меняя частоту работы МК, либо меняя разрядность ШИМ сигнала. Также менять эту величину можно, меняя значение регистра TCNT1 (более подробное описание смотри в описании МК).

Рассмотрим, почему нужно записать именно число **A** в регистр **OCR1A**.**A** шестнадцатеричное представление числа 10, а **10** – это примерно **1%** от **1024**, таким образом мы обеспечиваем требуемую величину **ШИМ.**

Пункты (ж), (и), (л), (н), (п) объединим в один пункт. Как организовать задержку по времени было рассмотрено в предыдущих лабораторных работах. Программно это выглядит следующим образом:

```
delay_ms(2000);
```

Также необходимо добавить строку в директивы предпроцессора, включающую описание этой функции:

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

Пункты (з), (м), (н), (о) объединим в один пункт. В этих пунктах необходимо изменять величину ШИМ сигнала. Для этого необходимо менять содержимое регистра **OCR1A** подобно тому, как это делалось в пункте (е). Соответственно для 25% необходимо послать 1024*0.25=256, для 50% - 512, для 75% - 768, для 99% - 1014.

Для того, чтобы организовать переход в начало программы также не требуется записи какого-либо программного кода, после выполнения процедуры **while(1)** программа возвращается к началу этой процедуры.

Общий листинг программы, реализующей данный алгоритм выглядит следующим образом:

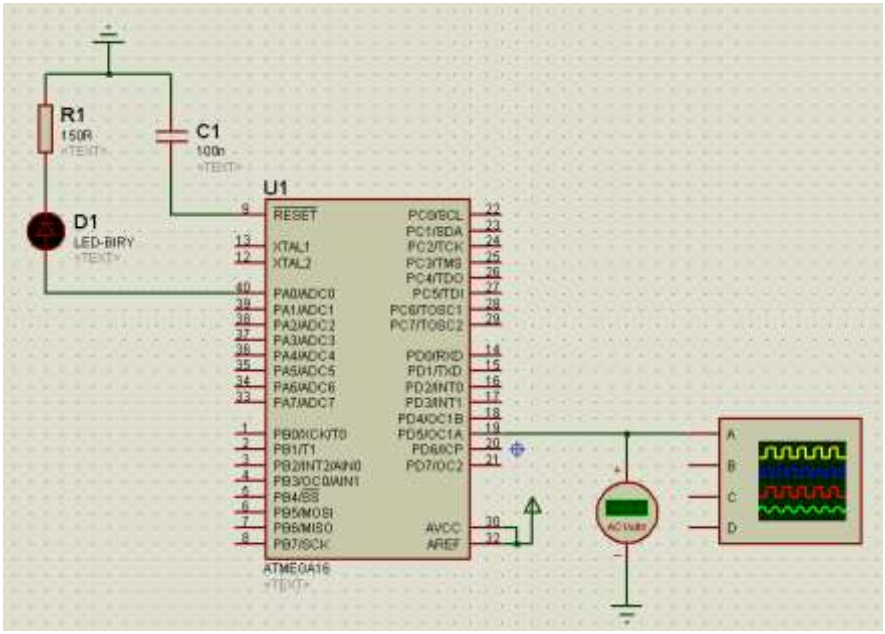
```
#include <mega16.h>
  #include <delay.h>
// Declare your global variables here
void main(void)
```

```
     {
     // Declare your local variables here
     // Input/Output Ports initialization
     // Port A initialization
     //  Func7=In  Func6=In  Func5=In  Func4=In  Func3=In  Func2=In
Func1=In Func0=Out
     //  State7=T  State6=T  State5=T  State4=T  State3=T  State2=T
State1=T State0=0
     PORTA=0x00;
     DDRA=0x01;
     // 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 C 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
     PORTC=0x00;
     DDRC=0x00;
     // Port D initialization
     //  Func7=In  Func6=In  Func5=Out  Func4=Out  Func3=In  Func2=In
Func1=In Func0=In
     //  State7=T  State6=T  State5=0  State4=0  State3=T  State2=T
State1=T State0=T
     PORTD=0x00;
     DDRD=0x30;
     // Timer/Counter 0 initialization
     // Clock source: System Clock
     // Clock value: Timer 0 Stopped
     // Mode: Normal top=FFh
     // OC0 output: Disconnected
     TCCR0=0x00;
     TCNT0=0x00;
     OCR0=0x00;
     // Timer/Counter 1 initialization
     // Clock source: System Clock
     // Clock value: 4000,000 kHz
     // Mode: Fast PWM top=03FFh
     // OC1A output: Non-Inv.
     // OC1B output: Non-Inv.
     // Noise Canceler: Off
     // Input Capture on Falling Edge
     // Timer 1 Overflow Interrupt: Off
     // Input Capture Interrupt: Off
     // Compare A Match Interrupt: Off
     // Compare B Match Interrupt: Off
     TCCR1A=0xA3;
```

```
TCCR1B=0x09;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

while (1)
      {
  PORTA.0=1;    // Place your code here
OCR1AH=0x00;
OCR1AL=0x0A;
   delay_ms(2000);
OCR1AH=0x01;
OCR1AL=0x00;
   delay_ms(2000);
OCR1AH=0x02;
OCR1AL=0x00;
   delay_ms(2000);
OCR1AH=0x03;
OCR1AL=0x00;
   delay_ms(2000);
OCR1AH=0x03;
OCR1AL=0xF6;
   delay_ms(2000);
        };
}
```

Следующим этапом является симуляция работы данной программы в среде Proteus. Для этого необходимо создать схему, представленную на следующем рисунке:



Далее необходимо осуществить симуляцию и получить осциллограммы исследуемого ШИМ сигнала.

Требования к отчету

Отчет должен содержать следующие разделы:

18. Титульный лист
19. Цель работы.
20. Описание принципов ШИМ управления.
21. Ход работы (с текстом программы и схемами в пакете Proteus, а также осциллограммами полученных сигналов).
22. Выводы.

Контрольные вопросы для самостоятельной работы

15. Что такое ШИМ?
16. Для чего используется ШИМ управление?
17. Основные параметры ШИМ сигнала.
18. Примеры расчета параметров работы двигателя при ШИМ управлении.
19. Принципы организации программной реализации ШИМ сигнала?

20. Принципы организации аппаратной реализации ШИМ сигнала?
21. Опишите существующие в МК таймеры.
22. Варианты применения таймеров в МК.
23. Отчего зависит время импульсов ШИМ сигнала при аппаратной реализации ШИМ управления?
24. Как можно добиться промежуточных значений времени импульса ШИМ сигнала?

Лабораторная работа №5
ИССЛЕДОВАНИЕ АНАЛОГО-ЦИФРОВОГО ПРЕОБРАЗОВАНИЯ (АЦП)

Цель работы:

1) Ознакомление с принципом работы и внутренним устройством, настройкой аналого-цифрового преобразователя (АЦП).
2) Написать программу, организующую работу АЦП МК Atmega16.

Объект исследования: микроконтроллер Atmega16.

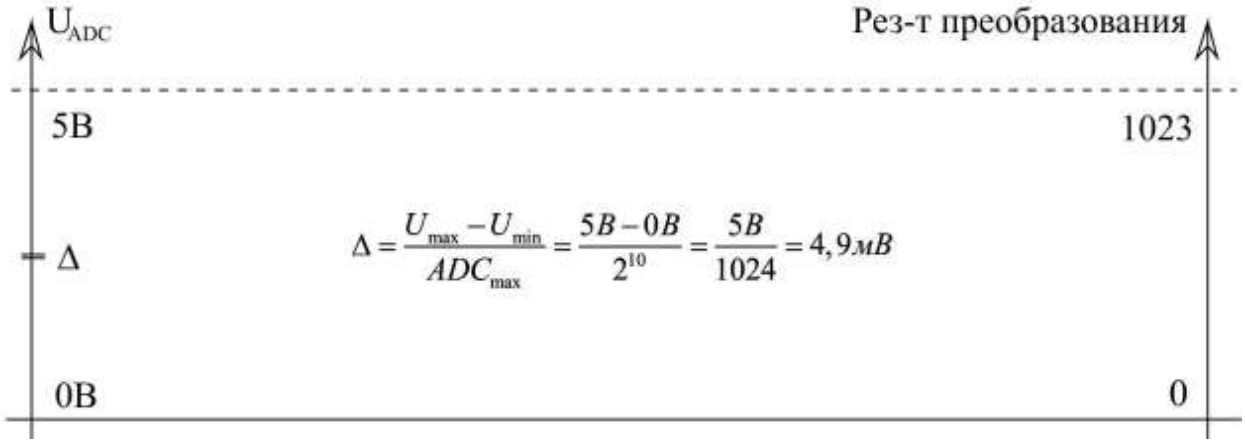
Аппаратные средства: виртуальная лаборатория на ЭВМ IBM PC, программный пакет CodeVisionAVR – компилятор Си для AVR микроконтроллеров, программный пакет Proteus – среда симуляции работы микроконтроллеров и других электронных устройств.

Краткая теория

Аналого-цифровой преобразователь (АЦП, англ. Analog-to-digital converter, ADC) - устройство, преобразующее входной аналоговый сигнал в дискретный код (цифровой сигнал). Обратное преобразование осуществляется при помощи ЦАП (цифро-аналогового преобразователя, DAC). Как правило, АЦП - электронное устройство, преобразующее напряжение в двоичный цифровой код. Тем не менее, некоторые неэлектронные устройства с цифровым выходом, следует также относить к АЦП, например, некоторые типы преобразователей угол-код.

Простейшим одноразрядным двоичным АЦП является компаратор. Разрядность АЦП Разрядность АЦП характеризует количество дискретных значений, которые преобразователь может выдать на выходе. В двоичных АЦП разрядность измеряется в битах. Разрядностью АЦП определяется и его разрешение – минимальное изменение величины входного аналогового сигнала, которое может быть зафиксировано данным АЦП. АЦП преобразовывает сигнал (напряжение) находящийся в диапазоне измеряемых сигналов. Нижняя и верхняя граница этого диапазона определяются напряжениями, поданными на соответствующие выводы. 1 Для МК со встроенным АЦП, нижняя граница – это уровень GND (0 В), а верхняя – подается на отдельный вывод (AREF) или используются внутренние источники опорных напряжений. При диапазоне входных напряжений от 0 В до 5 В и

использовании 10- битного АЦП мы имеем следующее разрешение АЦП Δ .



Разрешение 10-битного АЦП

Т.е. АЦП в состоянии различить сигналы, которые отличаются на 4,9 мВ. При увеличении сигнала на 4,9 мВ – результат преобразования увеличится на 1. Если для такого же диапазона входных сигналов использовать АЦП с большей разрядностью, то мы сможем зафиксировать меньшие значения, т.е. получить более точное значение сигнала (на рис представлены значения при использовании 24-битного АЦП).

На практике разрешение АЦП ограничено отношением сигнал/шум входного сигнала. При большой интенсивности шумов на входе АЦП различение соседних уровней входного сигнала становится невозможным, то есть 2 ухудшается разрешение. При этом реально достижимое разрешение описывается эффективной разрядностью (Effective Number Of Bits - ENOB), которая меньше, чем реальная разрядность АЦП. При преобразовании сильно зашумлённого сигнала младшие разряды выходного кода практически бесполезны, так как содержат шум.

Частота дискретизации АЦП Аналоговый сигнал является непрерывной функцией времени, в АЦП он преобразуется в последовательность цифровых значений. Следовательно, необходимо определить частоту выборки цифровых значений из аналогового сигнала. Частота, с которой производятся цифровые значения, получила название частота дискретизации АЦП.

Поскольку реальные АЦП не могут произвести аналого-цифровое преобразование мгновенно, входное аналоговое значение должно удерживаться постоянным, по крайней мере от начала до

конца процесса преобразования (этот интервал времени называют время преобразования). Эта задача решается путём использования специальной схемы на входе АЦП - устройства выборки-хранения. Устройство, хранит входное напряжение в конденсаторе, который соединён со входом через аналоговый ключ: при замыкании ключа происходит выборка входного сигнала (конденсатор заряжается до входного напряжения), при размыкании - хранение.

В микроконтроллерах частота дискретизации АЦП может быть настроена программно. Но, чем выше частота (более частая выборка) – тем больше ошибка преобразования (меньше точность).

Типы АЦП Существуют следующие типы АЦП:

- АЦП прямого преобразования (параллельный АЦП);
- последовательно-параллельные АЦП;
- АЦП последовательного приближения (с поразрядным уравновешиванием);
- АЦП дифференциального кодирования;
- АЦП сравнения с пилообразным сигналом;
- АЦП с уравновешиванием заряда;
- конвеьерные АЦП;
- АЦП с промежуточным преобразованием в частоту следования импульсов;
- сигма-дельта АЦП.

Для большинства АЦП разрядность составляет от 6 до 24 бит, частота дискретизации до 1 МГц. Мега- и гигагерцовые АЦП также доступны. Один из факторов увеличивающих стоимость микросхем - это количество выводов, поскольку они вынуждают делать корпус микросхемы больше, и каждый вывод должен быть присоединён к кристаллу.

Для уменьшения количества выводов часто АЦП, работающие на низких частотах дискретизации, имеют последовательный интерфейс.

Применение АЦП с последовательным интерфейсом зачастую позволяет увеличить плотность монтажа и создать плату с меньшей площадью. Часто микросхемы АЦП имеют несколько аналоговых входов, подключённых внутри микросхемы к единственному АЦП через аналоговый мультиплексор.

Различные модели АЦП могут включать в себя устройства выборки-хранения, инструментальные усилители или

высоковольтный дифференциальный вход и другие подобные цепи. Микроконтроллер MEGA16 содержит в своем составе 10-разрядный (10- битный), 8-иканальный АЦП последовательного приближения с встроенным устройством выборки-хранения.

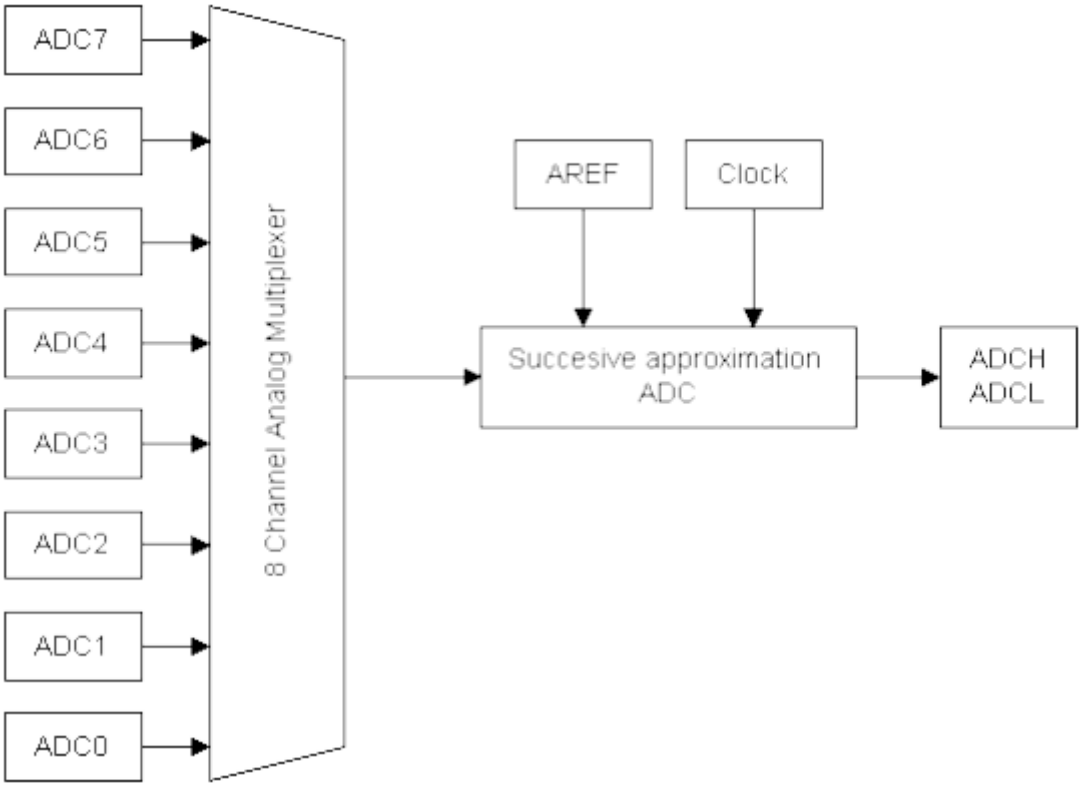


Схема АЦП В Atmega16

8 входов АЦП подключены к одному АЦП при помощи аналогового мультиплексора, т.е. АЦП в состоянии преобразовывать только один сигнал в 4 текущий момент времени.

Если необходимо преобразовывать несколько сигналов одновременно – то они преобразовываются последовательно.

На вывод МК под именем AREF подается опорное напряжение (верхняя граница для преобразователя), нижняя граница – всегда 0 В (GND).

В качестве верхней границы может быть использовано напряжение питания VCC или встроенный источник опорного напряжения 2,56 В.

АЦП в микроконтроллере, так же, как и все остальные устройства, работает от тактовой частоты МК. Частоту, подаваемую на АЦП можно понизить при помощи делителя частоты АЦП. Преобразование при этом будет выполняться более медленно, но и более качественно.

Применение АЦП Аналого-цифровое преобразование используется везде, где требуется обрабатывать, хранить или передавать сигнал в цифровой форме:

• АЦП являются составной частью систем сбора данных;

• быстрые видео АЦП используются, например, в ТВ-тюнерах (это параллельные и конвеерные АЦП);

• медленные встроенные 8, 10, 12 или 16-битные АЦП часто входят в состав микроконтроллеров (как правило они строются по принципу поразрядного уравновешивания, точность их невысока);

• очень быстрые АЦП необходимы в цифровых осциллографах (параллельные и конвеерные);

• современные весы используют АЦП с разрядностью до 24 бит, преобразующие сигнал непосредственно от тензометрического датчика (сигма-дельта АЦП);

• АЦП входят в состав радиомодемов и других устройств радиопередачи данных, где используются совместно с процессором цифровой обработки сигналов в качестве демодулятора;

• сверхбыстрые АЦП используются в антенных системах базовых станций (в так называемых SMART-антеннах) и в антенных решетках радиолокационных станций.

Описание регистров для программирования ADC

ADMUX (ADC Multiplexer Selection Register) – регистр мультиплексора АЦП

7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Bit 7:6 – REFS1:0: Reference Selection Bits

Эти биты определяют величину верхней границы опорного напряжения для АЦП (см. Табл.). Если данные биты изменить во время преобразования – изменение напряжения произойдет только при следующем преобразовании.

Таблица 1. Выбор опорного напряжения для АЦП

REFS1	REFS0	Выбор опорного напряжения
0	0	AREF (вывод МК), внутр. Vref выключено
0	1	AVCC с внешним конденсатором на выводе AREF
1	0	Зарезервированно
1	1	Внутр. 2.56 V напряжение с конденсатором на AREF

Bit 5 – ADLAR: ADC Left Adjust Result

ADLAR бит отвечает за представление результата АЦП в регистре данных АЦП. Установка бита в «1» приведет к выравниванию результата влево, «0» - вправо. Изменение бита приводит к немедленному изменению выравнивания результата АЦП (для текущего и последующих преобразований). Детальнее про выравнивание результата описано в регистре данных АЦП (регистры **ADCL** и **ADCH**).

Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits

Значения данных битов определяют комбинацию подключенных аналоговых сигналов, а также коэффициент усиления для дифференциальных режимов включения. Если биты изменить во премя преобразования, это отразится на следующем преобразовании.

ADCSRA (ADC Control and Status Register A) – регистр А настройки и статуса АЦП

7	6	5	4	3	2	1	0
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

Bit 7 – ADEN: ADC Enable

Данный бит включает/выключает АЦП. Сброс данного бита в «0» прерывает текущее преобразование.

Bit 6 – ADSC: ADC Start Conversion

В режиме одиночного преобразования, необходимо устанавливать данный бит в «1» для запуска каждого АЦП. ADSC

будет оставаться в «1» до тех пор, пока идет преобразование. Когда преобразование будет закончено, он автоматически сброситься в «0».

Bit 5 – ADATE: ADC Auto Trigger Enable

Этот бит отвечает за режим авто-запуска преобразования по сигналу. В данной работе данный резим не используется и не рассматривается. Для отключения данного режима – бит сбросить в «0».

Bit 4 – ADIF: ADC Interrupt Flag

Этот бит автоматически устанавливается в «1» по окончании преобразования. Обработчик прерывания по окончанию АЦП будет вызван, если прерывания для него разрешены в флаге ADIE и I-бит в регистре SREG установлен в «1». После выполнения обработчика, данный бит автоматически сбрасывается в «0».

Bit 3 – ADIE: ADC Interrupt Enable

Когда данный бит установлен в «1» и I-бит в регистре SREG установлен в «1», прерывания по окончанию преобразования разрешены.

Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits

Данные биты определяют делитель рабочей частоты МК для подачи ее на АЦП. Т.е. они определяют скрость работы АЦП (частоту дискретизации). Значения данных битов приведены в следующей таблице.

Таблица 3. Выбор входного канала АЦП

ADPS2	ADPS1	ADPS0	Делитель
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADCL и **ADCH** (The ADC Data Register) – регистр данных АЦП.

	7	6	5	4	3	2	1	0
ADCH	-	-	-	-	-	-	ADC9	ADC8
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0

Значения регистров при ADLAR = 0!!!

	7	6	5	4	3	2	1	0
ADCH	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
ADCL	ADC1	ADC0	-	-	-	-	-	-

Значения регистров при ADLAR = 1!!!

В результате преобразования 10-битное цифровое значение соответствующее напряжению на выводе МК размещается в двух 8-битных регистрах (ADCL и ADCH).

Выравнивание результата влево позволяет использовать более значимые биты (9..2). Т.е. младшие биты результата преобразования игнорируются, а используются только старшие – это приводит к потере точности преобразования, но работать с таким результатом (байт) гораздо удобнее, чем с 10-битным.

При чтении регистров сначала должен быть прочитан ADCL, затем ADCH. Если значения младших битов игнорируется – читают сразу ADCH.

Выполнение работы

Шаг 1:

Запускаем CoveVisionAVR.

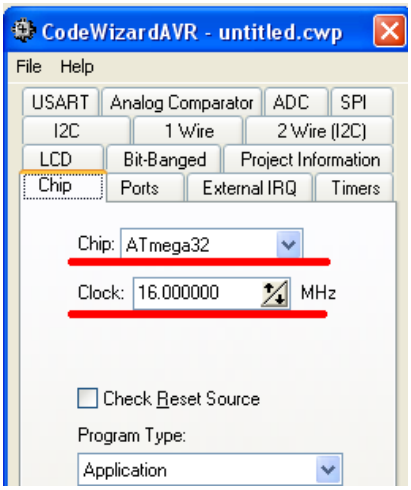
Шаг 2:

Создаем новый проект и запускаем **CODE Wizard AVR**.

Шаг 3:

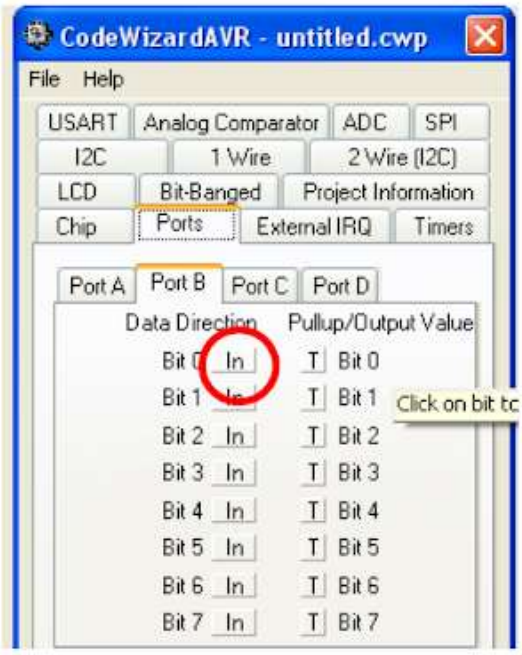
Настраиваем параметры проекта:

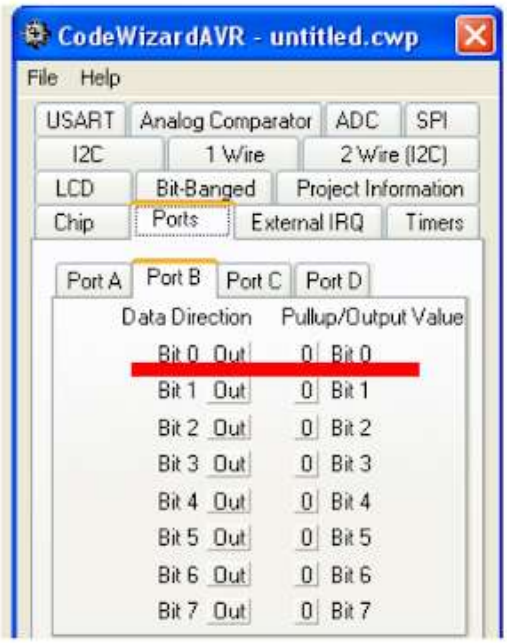
Контроллер выбираем Atmega16 Устанавливаем частоту 4.000000 MHz



Шаг 4:

Порты настраиваем следующим образом: все пины порта B назначаем выходами.

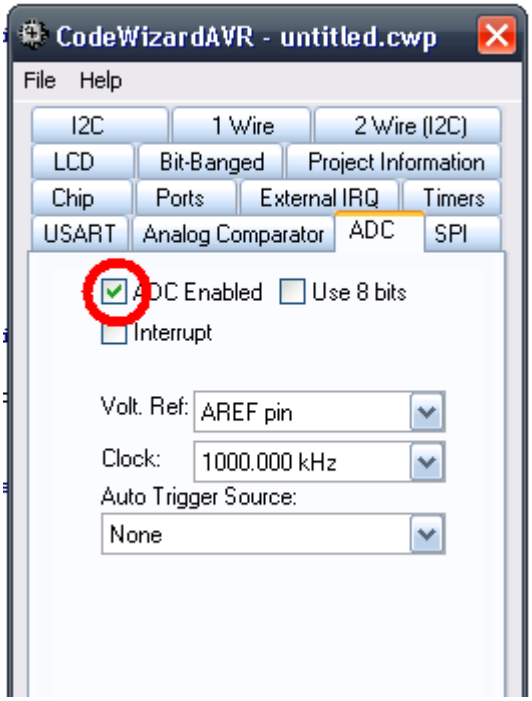




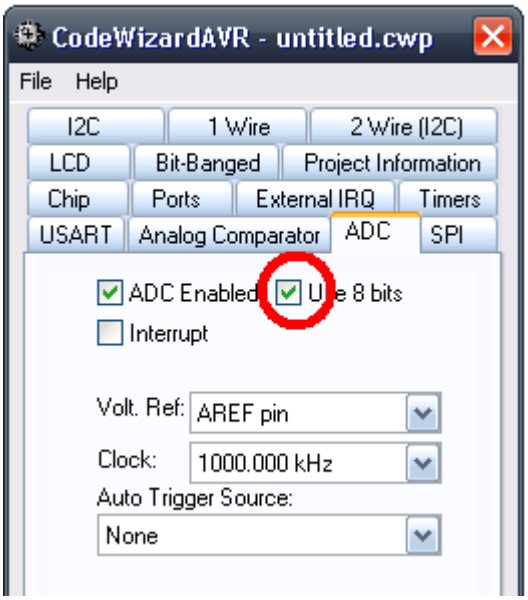
Шаг 5:

Настраиваем параметры АЦП (**ADC).**

Ставим галочку в чек бокс **ADC Enable** - тем самым активировав функцию АЦП.

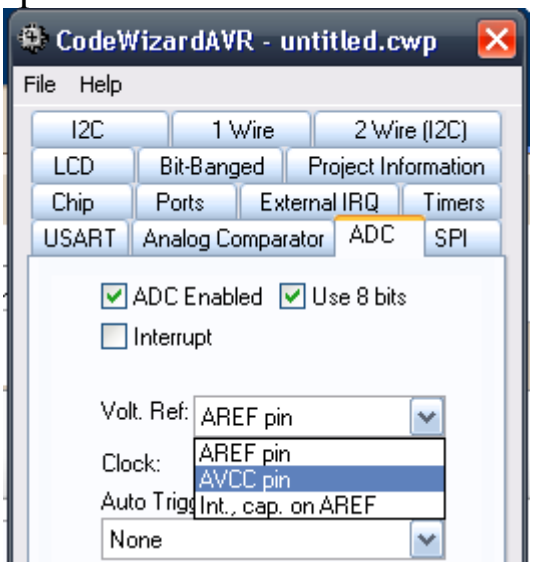


Так как для индикации работы будем использовать пины порта B, то в настройках указываем 8-битный режим работы АЦП. (это позволит нам использовать только младший регистр данных АЦП).



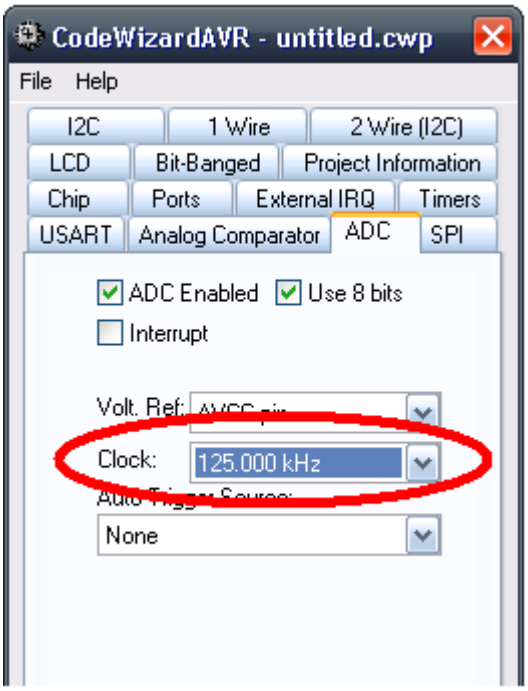
Шаг 6:

Далее настраиваем опорное напряжение (**Voltage Reference)** для ЦАП. В данном проекте будем использовать пин **AVCC** как источник опорного напряжения.



Шаг 7:

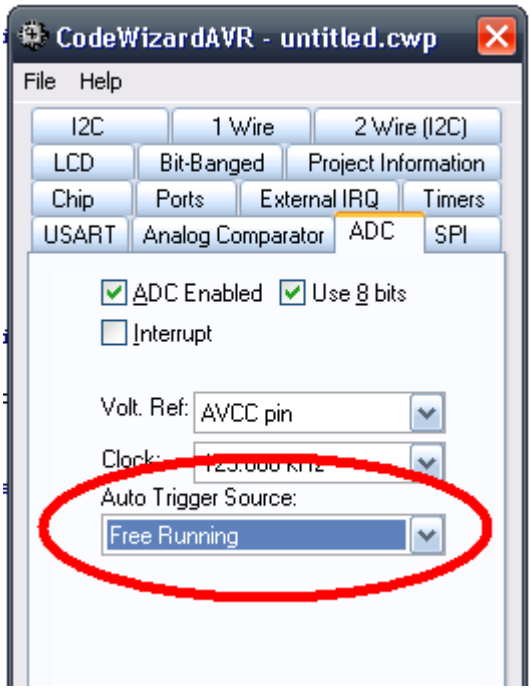
Далее настраиваем частоту работы АЦП **125.000 kHz**



Повышение частоты работы АЦП может привести к снижению точности показаний. более подробно эти вопросы изложены в документации на микроконтроллер.

Шаг 8:

Настраиваем условия запуска преобразования АЦП. В нашем случае это **Free Running** - то есть постоянная работа, не привязанная ни к прерываниям, ни к внутренним счетчикам.



Шаг 9:

На этом настройка завершена, можно применить опцию **Generate, Save and Exit**.

Шаг 10:

----------Рассмотрим сгенерированный код. Появилась функция **read_adc**.

```
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 16.000000 MHz
Memory model        : Small
External RAM size   : 0
Data Stack size     : 256
*****************************************************/

#include <mega16.h>

#include <delay.h>

#define ADC_VREF_TYPE 0x60

// Read the 8 most significant bits
// of the AD conversion result
unsigned char read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCH;
}

// Declare your global variables here
```

Шаг 11:

Далее переходим к написанию кода - функции **while**.

```
while (1)
      {
      // Place your code here

      };
}
```

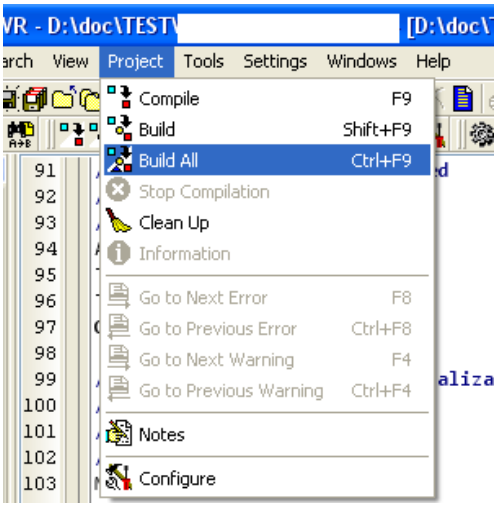
Шаг 12:

Для проверки работы, напишем код иллюстрирующий работы АЦП с помощью порта B.

```
while (1)
      {
      // Place your code here
      delay_ms(500);
      // Let us connect our analog input
      // at PIN0 (PIN zero) of PORTA.
      // Hence, we have to select channel zero
      // for A-D conversion.
      PORTB = read_adc(0);
      };
}
```

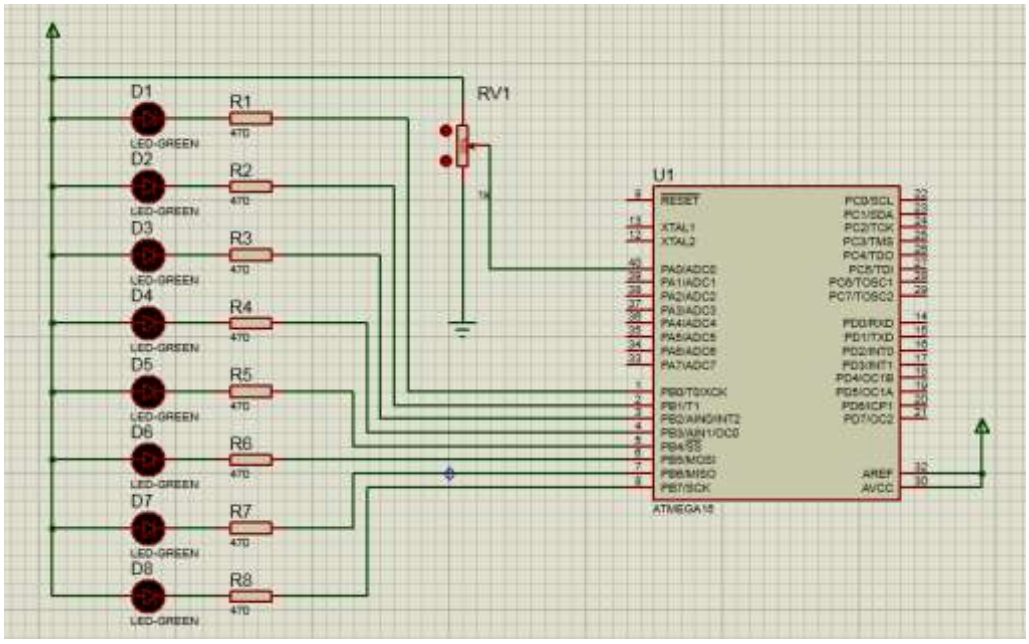
Шаг 13:

Компилируем проект и генерируем файл прошивки микроконтроллера. Следим, чтобы при компиляции компилятор не показал наличие ошибок.



Шаг 14:

Переходим к моделированию проекта в Proteus. Для это соберем схему состоящую из, микроконтроллера Atmega16, модели резистивного потенциометра POT-HG, 8 светодиодов LED и резисторов Resistor 470 Ом

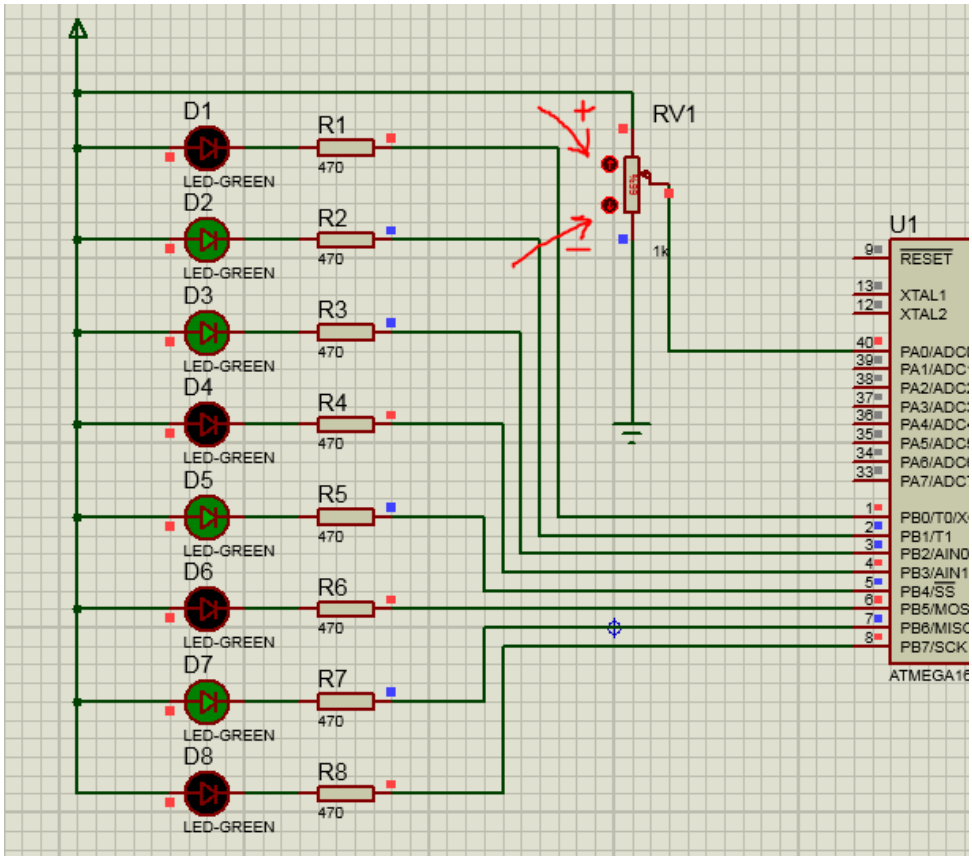


Шаг 15:

Указываем в настройках контроллера файл прошивки и моделируем работу устройства.

В процессе моделирования можно изменять положение ползунка переменного резистора, нажимая на кнопки "+" "-". При этом цифровой код значения АЦП, будет выводится на светодиодную панель.

Обратим внимание, что светодиоды подключены инвертировано - т.е. значению бита "1" соответствует не горящий светодиод.



Контрольные вопросы для самостоятельной работы

1. Что такое АЦП.
2. На что влияет разрядность АЦП.
3. Что характеризует частота дискретизации АЦП.
4. Какие типы АЦП существуют.
5. Области применения АЦП.
6. Характеристика АЦП для МК MEGA16.
7. Что измеряет АЦП – ток, напряжение или сопротивление.

Библиографический список

1. Гусев В. Г., Электроника и микропроцессорная техника : Учебник / Ю. М. Гусев. - 3-е изд., перераб. и доп. - М. : Высшая школа, 2004. - 790 с
2. Крекрафт Д., Аналоговая электроника. Схемы, системы, обработка сигнала [Текст] . - М. : Техносфера, 2005. - 360 с.
3. Наундорф У., Аналоговая электроника. Основы, расчет, моделирование [Комплект]: [учебное пособие] / пер. с нем. М. М. Ташлицкого. - М. :Техносфера, 2008. - 472 с
4. Розанов Ю. К., Силовая электроника [Текст] : учебник. - 2-е изд., стер. - М. : МЭИ, 2009. - 632 с.
5. Ревич Ю., Занимательная микроэлектроника [Текст] . - СПб. : БХВ-Петербург, 2007. - 592 с.
6. Семенов Б.Ю. Силовая электроника для любителей и профессионалов. // М.:СОЛОН-Р., 2001.
7. Хоровиц, Хилл. Искусство схемотехники. 7 перераб. изд. 2005 г. 700