

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
1. ЛАБОРАТОРНАЯ РАБОТА №1. ПАКЕТ AVR STUDIO. МИКРОКОНТРОЛЛЕР ATMEGA16.....	5
1.1. Создание проекта и программы	5
1.2. Отладка программы.....	10
1.3. Программирование МК	12
1.4. Задания.....	15
2. ЛАБОРАТОРНАЯ РАБОТА №2. ПОРТЫ ВВОДА-ВЫВОДА.....	18
2.1. Теория	18
2.2. Примеры программ	19
2.3. Отладка программы.....	20
2.4. Задания.....	23
3. ЛАБОРАТОРНАЯ РАБОТА №3. ПОСЛЕДОВАТЕЛЬНЫЙ ПОРТ USART	26
3.1. Теория	26
3.2. Примеры программ	29
3.3. Отладка программы.....	40
3.4. Задания.....	41
4. Лабораторная работа №4. 8-битный таймер/счетчик 0.....	43
4.1. Теория	43
4.1.2. Режимы работы таймера	43
4.1.2.1. Нормальный режим (Normal Mode)	43
4.1.2.2. Режим обнуления по совпадению (Clear Timer on Compare Match (CTC) Mode)	44
4.1.2.3. Режим быстрого ШИМ (Fast PWM).....	44
4.2. Примеры программ	46
4.2.1. Нормальный режим (Normal Mode)	46
4.2.2. Режим обнуления по совпадению (CTC Mode)	49
4.2.3. Режим быстрого ШИМ (Fast PWM).....	53
4.3. Задания.....	55
5. ЛАБОРАТОРНАЯ РАБОТА №5. ПАМЯТЬ МК	57
5.1. Теория	57
Библиографический список.....	60
Ресурсы	61

ВВЕДЕНИЕ

Настоящее методическое руководство содержит лабораторный практикум, необходимый студентам для освоения программирования микроконтроллеров AVR. С относительно недавнего времени микроконтроллеры (не только AVR), благодаря своей невысокой стоимости, развитой периферии, относительно высокой вычислительной мощности, доступных средствах разработки программного обеспечения и т.п. стали весьма популярны. Их можно встретить в стиральной машинке, в микроволновой печи, в приводе станка или робота, в мобильном телефоне и даже на космической станции. Отличительной особенностью микроконтроллеров от микропроцессоров является функциональная законченность. В одном корпусе микросхемы содержится микропроцессор, оперативная память, перепрограммируемая память программ и данных, порты ввода-вывода, разнообразные таймеры, последовательные порты, компараторы, аналого-цифровые преобразователи. При своих малых габаритах микроконтроллеры весьма разнообразны. Это могут быть простые 8-разрядные МК с 2 Кб памяти программ, 512 байтами ОЗУ и с небольшим набором периферии, так и 32-разрядные – с 512 Кб памяти программ, 98 Кб оперативной памяти и широчайшим набором периферийных устройств. Для изучения были выбраны 8-разрядные МК AVR фирмы ATMEL. В лабораторных работах используется микроконтроллер ATmega16. Этот выбор является своего рода компромиссным вариантом по параметрам: цена – простота – возможности. Для удобства макетирования схем фирма ATMEL выпускает отладочную плату STK500, на которой смонтированы панельки для различных корпусов микроконтроллеров, 8 кнопок и светодиодов, а также выведены все порты ввода-вывода.

Для разработки программного обеспечения для микроконтроллера (далее МК) используется бесплатный пакет AVR Studio фирмы ATMEL. Этот пакет представляет собой интегрированную среду разработки (IDE) и позволяет осуществить полный цикл программирования МК, включающий непосредственно само написание программы, ее отладку и загрузку исполняемого кода в память программ МК. Программа может быть написана на языке ассемблера или на языке Си/Си++. В последнем случае необходимо установить свободно распространяемый компилятор WinAVR (см. разд. «Ресурсы»). Отладку программы можно выполнять на симуляторе или аппаратно, используя ядро и периферию МК непосредственно. Симулятор использует математическую модель МК и позволяет визуально видеть значения всех регистров, содержимое всех видов памяти, поведение периферии и ход выполнения программы. Симулятор не моделирует МК в реальном времени. В случае аппаратной отладки необходим специальный отладчик, который подключает отлаживаемый МК к ПК. Для этого на МК предусмотрен отладочный интерфейс JTAG. В этом случае отладка ведется в режиме реального времени, программист может в любой момент прервать выполнение программы, посмотреть и если необходимо, изменить содержимое оперативной памяти и регистров МК, запустить программу в пошаговой отладке, задать точки останова, т.е. провести отладку «на кристалле». Микроконтроллер

может быть запрограммирован прямо в собранной схеме (ISP) режим, либо используя высоковольтный параллельный (последовательный) режимы, а также с помощью отладчика через интерфейс JTAG. Мы будем использовать режим ISP.

В конце методического пособия есть три раздела. Раздел «Термины и сокращения» расшифровывает все аббревиатуры, которые Вы встретите в тексте. Раздел «Ресурсы» содержит некоторые полезные ссылки в Интернете. И раздел «Библиографический список» поможет подобрать литературу, касающуюся тематики МК AVR.

1. ЛАБОРАТОРНАЯ РАБОТА №1. ПАКЕТ AVR STUDIO. МИКРОКОНТРОЛЛЕР АТМЕГА16.

Цель работы: познакомиться с созданием рабочего проекта в среде AVR Studio, опциями и настройками, предоставляемыми на этом этапе, отладкой простейшей программы и программированием МК.

1.1. Создание проекта и программы

1.1.1. Запустите пакет AVR Studio: обычно через главное меню Windows: «Пуск-> Все программы -> Atmel AVR Tools -> AVR Studio 4». Появится главное окно программы (рис. 1.1).

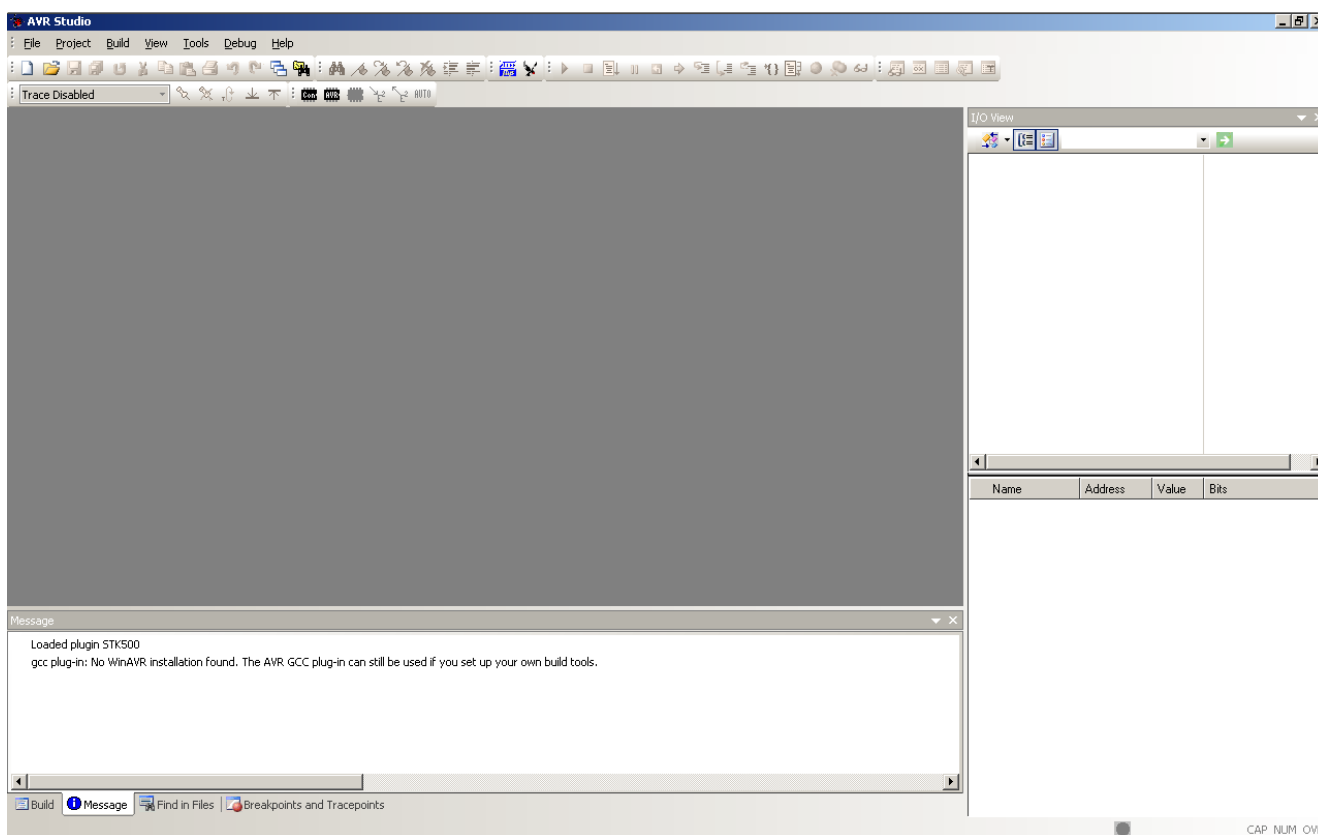


Рис. 1.1. Главное окно пакета AVR Studio после запуска

1.1.2. Для создания нового проекта выберите меню «Project -> New Project». Появится окно создания проекта (рис. 1.2). Окно содержит следующие поля и опции (в квадратных скобках указаны опции, которые следует выбрать):

1.1.2.1. «**Project type**». Здесь можно выбрать язык программирования, на котором будет написана программа: ассемблер или Си/Си++. [**Atmel AVR Assembler**].

1.1.2.2. «**Location**». В этом поле указывается путь к проекту. Пусть не должен содержать имен с русскими буквами и пробелами. Разместите свой проект на Вашем сетевом диске или, в крайнем случае, на локальном диске машины, но обязательно в c:\temp.

1.1.2.3. **«Project Name»**. В этом поле указывается имя проекта. К имени проекта предъявляются те же требования, что и к пути проекта (см. пункт 1.1.2.2). [**«blink»**].

1.1.2.4. **«Create initial file»**. Если опция выбрана, то создается пустой файл программы с именем из поля «Initial file» и расширением asm. Если листинг программы существует – «галочку» можно снять. [**«не снимать»**].

1.1.2.5. **«Create folder»**. Если опция выбрана, то в месте «Location» создается подкаталог с именем проекта в котором размещается весь проект. [**«не снимать»**].

После того как все поля заполнены и опции выбраны нажмите кнопку «Next».

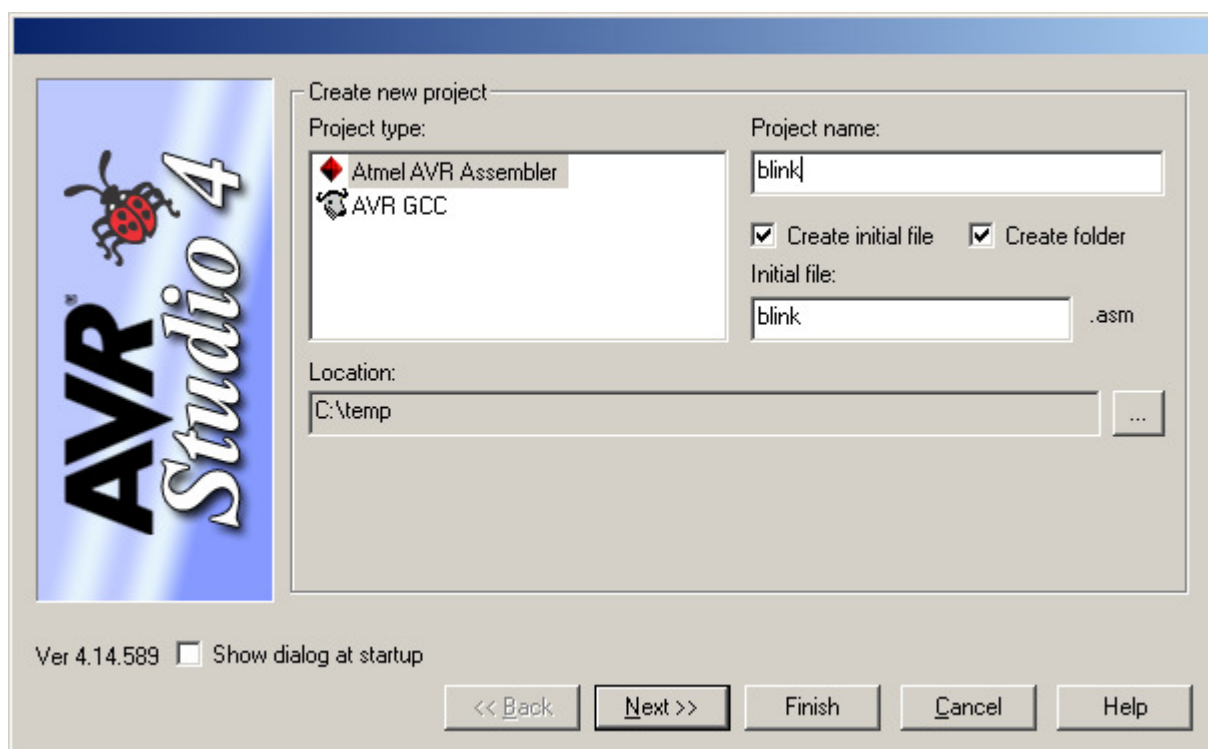


Рис. 1.2. Окно создания проекта

1.1.3. Следующее окно позволяет выбрать способ отладки программы и тип МК (рис. 1.3). Окно содержит следующие опции:

1.1.3.1. **«Debug platform»**. Здесь указывается тип аппаратного отладчика для отладки через интерфейс JTAG, либо симулятор AVR Simulator или AVR Simulator 2. [**«AVR Simulator»**].

1.1.3.2. **«Device»**. Здесь выбирается тип МК. [**«ATmega16»**].

После того как все опции выбраны, нажмите кнопку «Finish».

Вид главного окна после создания проекта показан на рис. 1.4. Основную часть экрана занимает белое поле с курсором. Эту область в дальнейшем мы будем называть «окном листинга» или просто «листингом». В этом окне набирается программа. Левее центральной части экрана расположено окно «Project», которое содержит древовидную структуру файлов и меток проекта. Самое нижнее окно «Build» - в него выводится отчет о трансляции программы с языка ассемблера в машинный код. Справа расположено окно «I/O View», которое

представляет обзор всей периферии МК. Следует заметить, что расположение окон не является статичным и может быть любым другим. Но Вы всегда можете разместить окна так, как удобно Вам. Наша первая программа будет чрезвычайно проста. Но она позволит увидеть весь цикл работы с МК, начиная от ввода программы и заканчивая результатом – программированием МК.

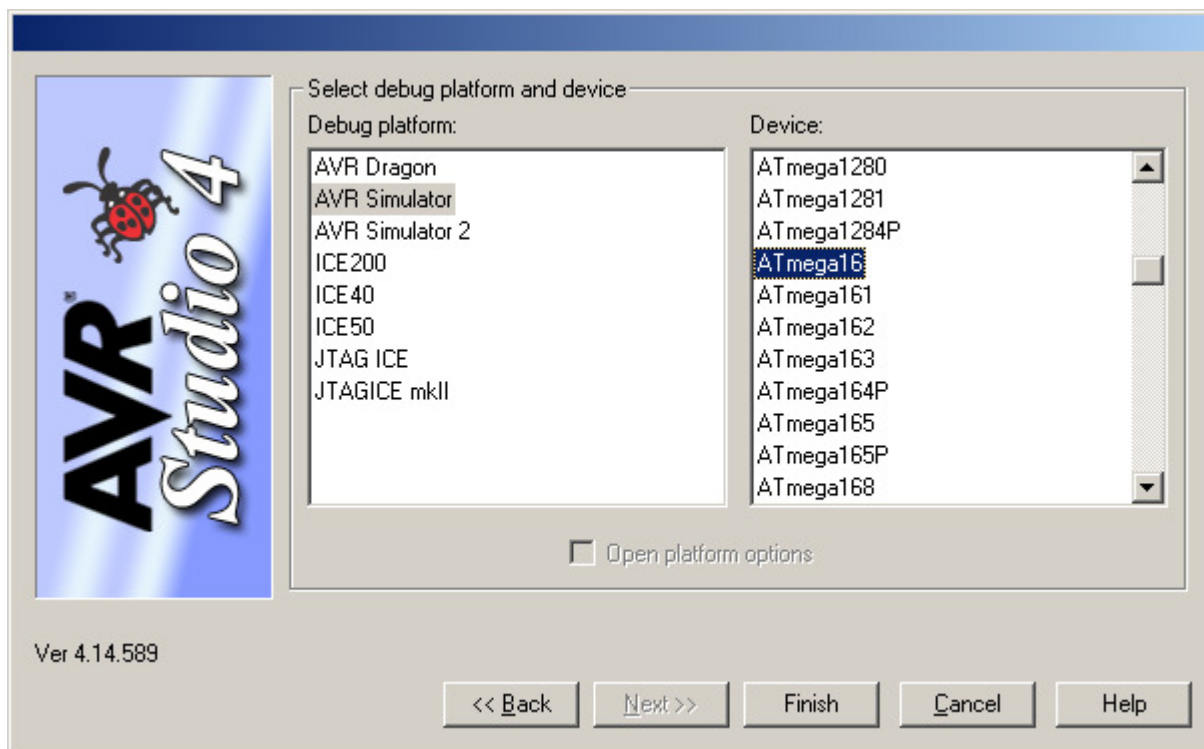


Рис. 1.3. Окно выбора способа отладки программы и типа МК

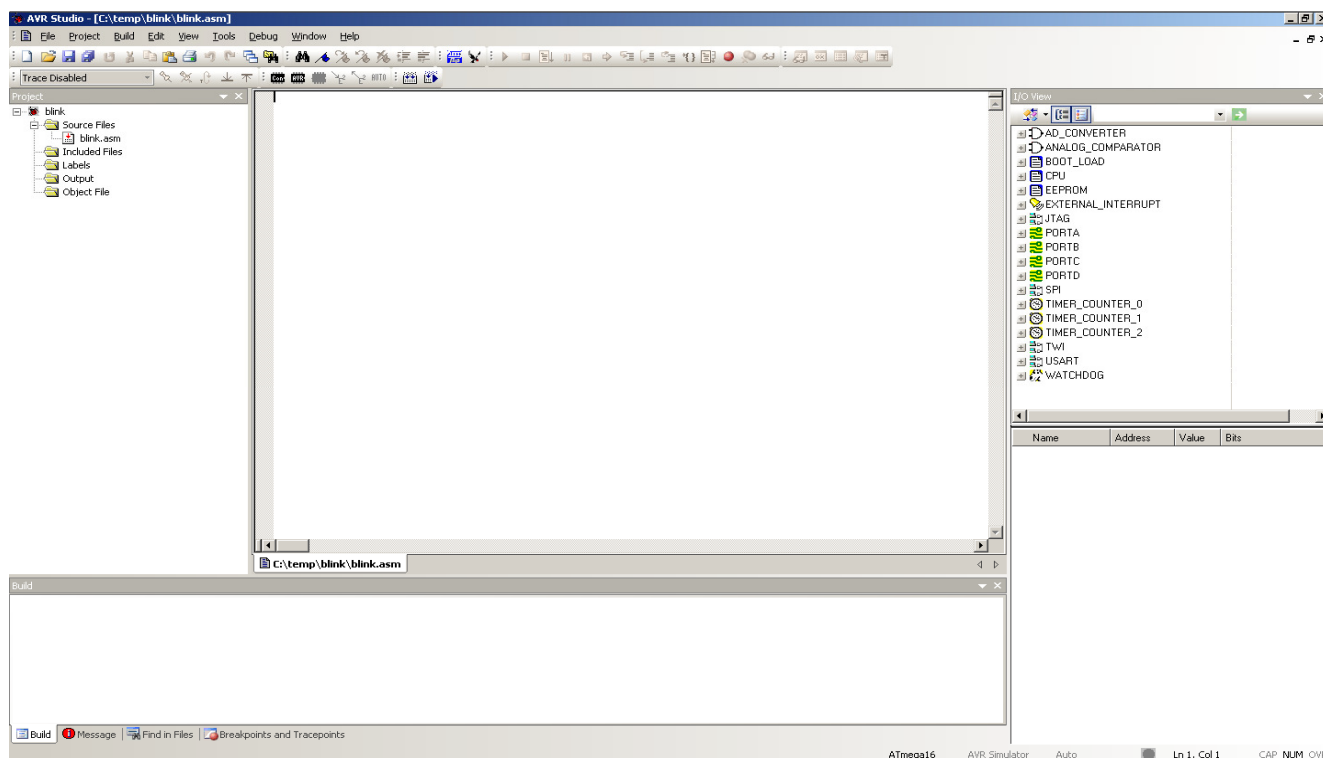


Рис. 1.4. Главное окно после создания проекта

Перейдите в окно листинга и наберите текст программы, приведенной ниже. Не копируйте текст через буфер обмена, а наберите вручную. Это способствует обучению. Комментарии к программе можно не набирать, но внимательно прочитать – следует. Также можно не набирать «шапку» программы, с описанием ее назначения, датой создания и авторства. Все комментарии начинаются с символа «точка с запятой».

```
; Первая ознакомительная программа.
; Демонстрация эффекта "мигающий светодиод".
; Рассчитано на частоту 8 МГц.
; Светодиоды подключены к PORTB
;
; 20080829 - дата создания
;
; ИрГТУ, кафедра ОАМ, автор Якимов Ю.А. (АМ-06-2)

#include "m16def.inc"      ; Подключаем заголовочный файл,
                           ; который содержит
                           ; определения ресурсов
                           ; микроконтроллера

; Установка указателя стека SP
; Т.к. все регистры 8-битные, а адрес ОЗУ 16-битное
; число, то указатель стека инициализируется в два
; этапа.

ldi r16, low(RAMEND) ; Загружаем младший байт
                     ; последнего адреса ОЗУ в регистр
                     ; r16 (константа RAMEND
                     ; определена в m16def.inc)
out  SPL, r16        ; Пересылаем значение из r16 в
                     ; SPL
ldi r16, high(RAMEND) ; Загружаем старший байт
                     ; последнего адреса ОЗУ в
                     ; регистр r16
out  SPH, r16        ; Пересылаем значение из r16
                     ; в SPH
; Теперь указатель стека инициализирован числом 0x45F
; (для ATmega16)

; Настраиваем все линии порта B на выход
ldi r16, 0xff
```

```
out DDRB, r16      ; В регистре DDRB все разряды
                   ;установлены в 1
; Устанавливаем лог. 1 на всех разрядах порта В
; (светодиоды не горят)
ldi r16, 0xff
out PORTB, r16

; На этом этапе конфигурация
; периферии микроконтроллера завершена

; Реализация эффекта
_main_cycle:      ; главный цикл
cbi PORTB, 0      ; устанавливаем в лог. 0
                  ; линию 0 PORTB
rcall _delay_500ms ; вызываем задержку на 500 мс
sbi PORTB, 0      ; возвращаем лог. 1
rcall _delay_500ms ; задержка
rjmp _main_cycle  ; переходим на начало цикла

; Подпрограмма задержки на 500 мс
_delay_500ms:
ldi r20, 0x15
_loop2:
ldi r19, 0xff
_loop1:
ldi r18, 0xf8
_loop0:
dec r18
brne _loop0
dec r19
brne _loop1
dec r20
brne _loop2
ret
```

Убедитесь, что текст набран без ошибок. Оттранслируйте программу, выбрав в главном меню пункт «Build -> Build and Run» или нажав сочетание клавиш «CTRL+F7». Окно «Build» не должно содержать ошибок: в его последней строке должно быть написано «Assembly complete, 0 errors. 0 warnings» (рис. 5).

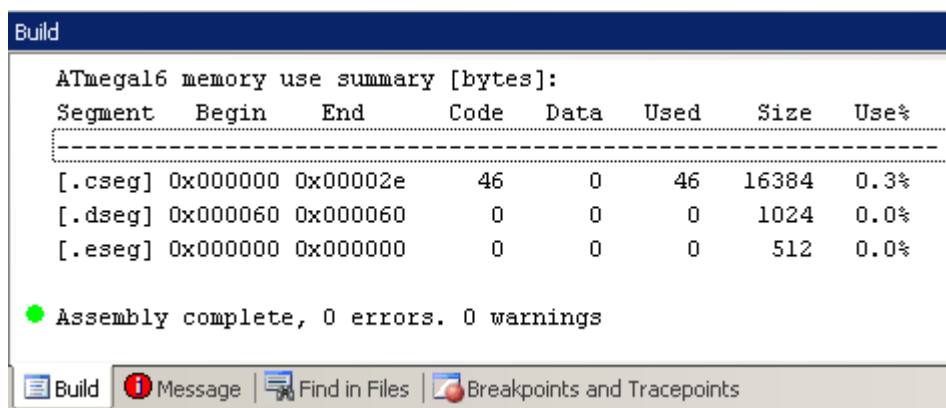


Рис. 1.5. Отчет о трансляции программы

1.2. Отладка программы

Симулятор, входящий в состав пакета AVR Studio позволяет промоделировать работу запрограммированного МК с целью отладки программы. Сам МК при этом **не используется**. Если трансляция программы прошла успешно, то слева от листинга появится желтая стрелочка. Это признак того, что симулятор готов к работе. Прежде чем исполнять программу, настроим симулятор. Выберите пункт меню «Debug -> AVR Simulator Options» или нажмите сочетание клавиш «CTRL+O». Появится окно, показанное на рис. 1.6. Настройте симулятор согласно рисунку. На вкладке «Stimuli and logging» ничего изменять не нужно.

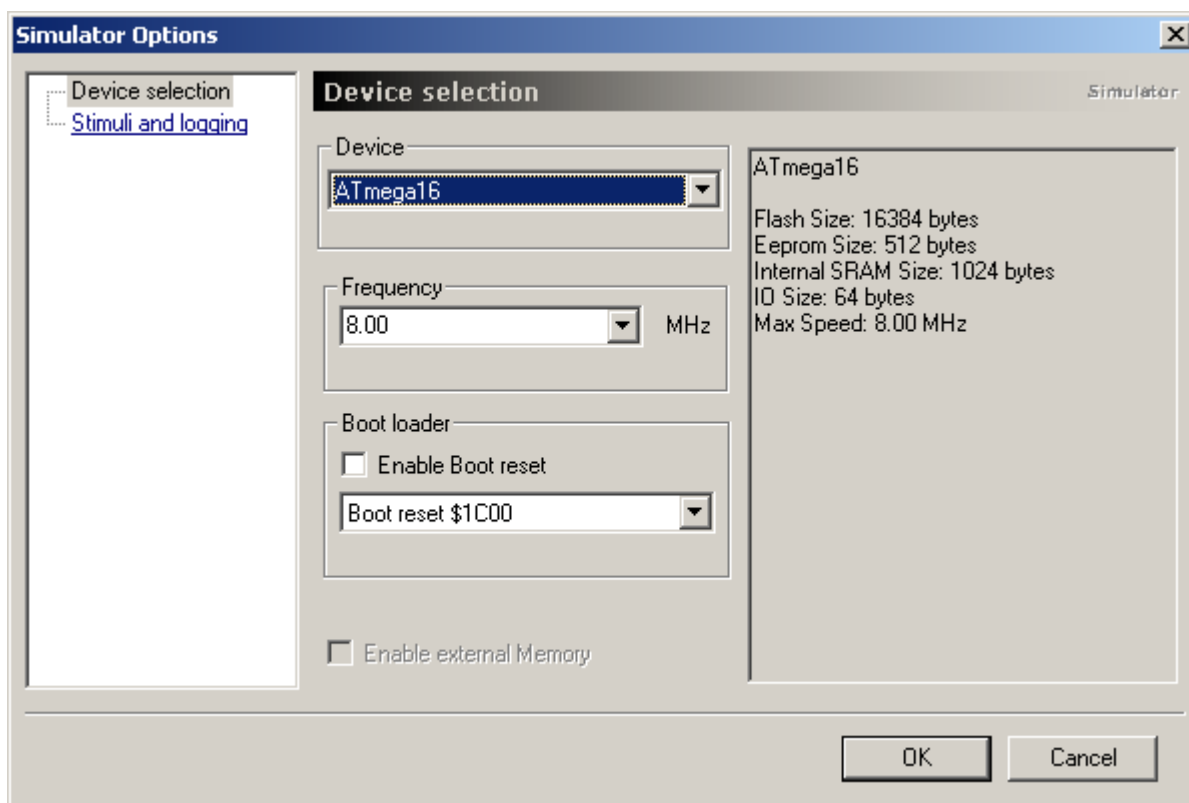


Рис. 1.6. Настройки симулятора

Окончив настройки, нажмите кнопку «ОК». Теперь симулятор будет моделировать МК ATmega16 на частоте 8 МГц.

Обратите внимание, что в листинге программы первые четыре строчки инициализируют указатель стека. Для того, чтобы выполнить одну команду МК можно выбрать пункт меню «Debug -> Step Into». Выделите пункт «CPU» в окне «I/O View». Чуть ниже будут отображены все регистры, относящиеся у этому пункту. Нас интересует регистр SP – указатель стека. Сейчас его значение равно нулю. Выполните первые четыре команды МК, внимательно следя за изменениями регистра SP. Вы заметили, что регистр был инициализирован числом 0x045F в два этапа (рис. 1.7).

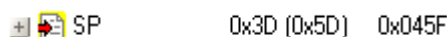


Рис. 1.7. Регистр SP

Сначала в него было записано число 0x5F – младший байт, а затем 0x04 – старший байт. Это связано с тем, что указатель стека – шестнадцатиразрядный регистр, а регистры общего назначения, с помощью которых происходит большинство операций в МК – восьмиразрядные. Следующие четыре команды инициализируют все разряды порта PORTB на выход и записывают в него число 0xFF, т.е. все разряды порта установлены в лог. 1. Выполните эти команды. Далее следует метка «_main_cycle». С нее начинается главный цикл программы – цикл, из которого МК не выходит никогда, образован конструкцией:

```
_main_cycle:
; код
rjmp _main_cycle
```

Команда `rjmp` обозначает безусловный переход на метку. Таким образом, МК постоянно выполняет код между меткой «_main_cycle» и последней командой безусловного перехода.

Выделите пункт «PORTB» в окне «I/O View». В нижней половине окна, появятся обозначения трех регистров, относящихся к управлению и контролю порта PORTB:

1.2.1. **DDRB** – этот регистр контролирует направление каждой линии порта. Если разряд установлен в лог. 1, соответствующая линия порта будет настроена на вывод. Если разряд сброшен в лог. 0 - наоборот. В нашем случае все линии порта настроены на вывод, т.е. каждый разряд этого регистра установлен в лог. 1.

1.2.2. **PINB** – этот регистр позволяют считать реальный логический уровень на каждой линии порта, независимо от того, настроена эта линия на ввод или на вывод.

1.2.3. **PORTB** – этот регистр позволяет установить нужный логический уровень на соответствующей линии порта, если она настроена на вывод или подключить внутренний подтягивающий резистор, если она настроена на ввод.

Выполните одну команду программы. Младший разряд регистра PORTB изменил свое состояние на лог. 0. Следующая команда `rcall` – вызов подпрограммы. В данном случае это подпрограмма, которая выполняет вложенные циклы на общее время 500 мс. Т.е. подпрограмма задержки. Таким образом, светодиод мигает с частотой 1 Гц. В окне «Processor» щелкните правой клавишей мыши и в контекстном меню выберите пункт «Reset Stopwatch». Выполните пункт меню «Debug -> Step Over». В нижней строке состояния программы AVR Studio появится зеленый кружок со значком «плюс» внутри (рис. 1.8).

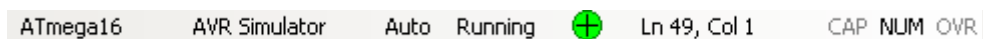


Рис. 1.8. Строка состояния

Этот значок говорит о том, что программа выполняется. Как Вы заметили, вход в подпрограмму задержки не произошел. Также не было пошаговой отладки. Дело в том, что различают два режима пошаговой отладки:

1.2.1. «Step Into» - пошаговая отладка, со входом в подпрограммы. Выполняется из меню «Debug->Step Into».

1.2.2. «Step Over» - пошаговая отладка без входа в подпрограммы. Выполняется из меню «Debug->Step Over».

Второй режим предпочтительнее, когда подпрограмма является отлаженной, долго выполняется и т.п. После истечения времени задержки, симулятор остановится на следующей команде, а параметр «Stop Watch» в окне «Processor» покажет время исполнения подпрограммы задержки – 500301.88 мкс. Можно переключить отображение времени из микросекунд в секунды с помощью контекстного меню в окне «Processor», пункт «Show stopwatch as milliseconds». Счетчик «Stop Watch» суммирует время всех выполненных команд. Если Вам необходимо измерить время выполнения какого-либо фрагмента кода, сбросьте счетчик, как было показано выше, перед выполнением этого фрагмента.

Вы можете продолжить симулировать программу и убедиться в том, что она работает. Назначение команд `cbi`, `sbi` должно быть понятно из контекста.

1.3. Программирование МК

Строго соблюдая технику безопасности при работе с отладочными комплектами STK500, извлеките наборе из футляра и аккуратно разместите на рабочем столе, не допуская натяжения кабелей. **Внимательно следя за правильностью подключения (см. ТБ)**, соедините плоским кабелем разъемы «PORTB» и «LEDS» на плате STK500. Проверьте, что кабель последовательного порта подключен к разъему «RS232 CTRL» на плате STK500. **Убедившись в правильности подключений**, подайте питание на плату. Светодиод «POWER» должен гореть красным цветом. Светодиод «STATUS» должен загореться красным, затем желтым + красным и зеленым цветом, погаснуть и снова загореться зеленым цветом (рис. 1.9). **Если этого не произошло, немедленно**

выключите питание платы и сообщите об этом преподавателю или технику лаборатории.

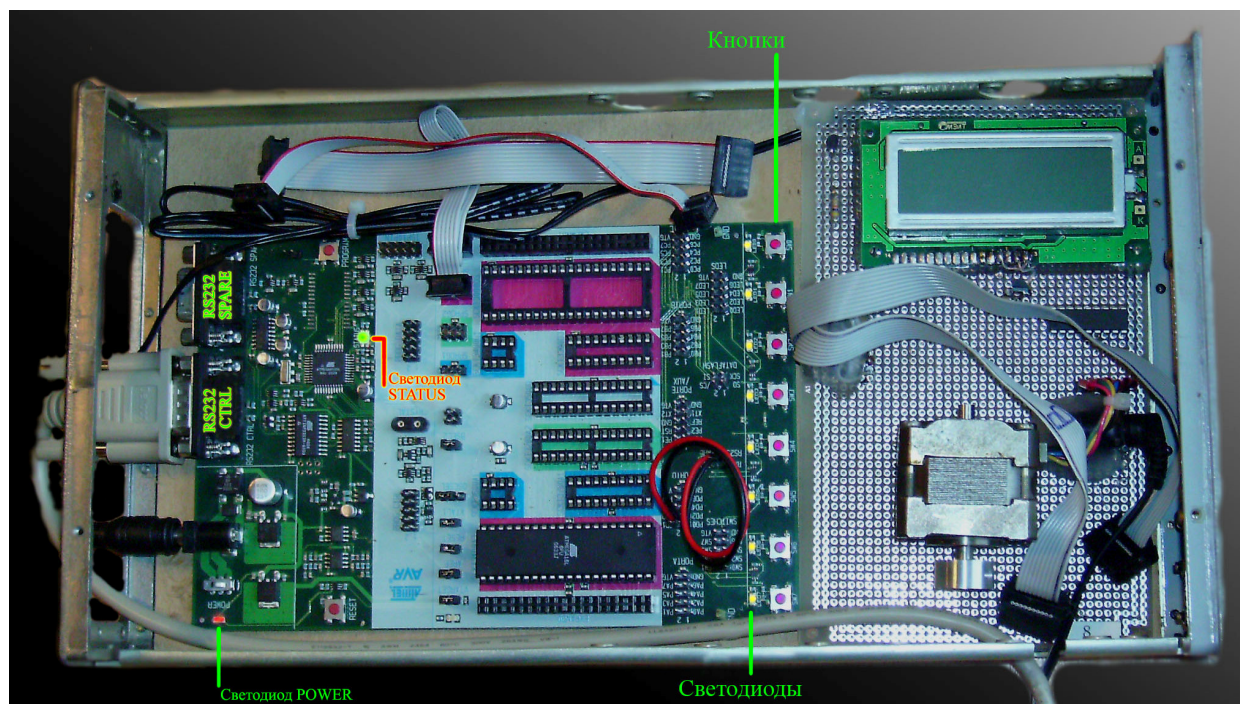


Рис. 1.9. Отладочный набор STK500

Выберите пункт меню «Tools->Program AVR->Auto Connect». Спустя некоторое время появится окно «STK in ISP mode with ...». Где вместо троеточия будет модель МК. Если она отличается от ATmega16, перейдите на вкладку «Main» и в выпадающем списке выберите нужный МК (рис. 1.10).

Затем перейдите на вкладку «Program» и выберите файл blink.hex для загрузки в память программ МК, а также убедитесь, что все параметры соответствуют приведенным на рис. 1.11. Установленный флажок «Erase device before flash programming» говорит о том, что память программ МК будет стерта перед записью новой программы. Это условие обязательно! Флажок «Verify device after programming» говорит о том, что память программ МК будет проверена на верность записанных данных после программирования. Эта опция желательна.

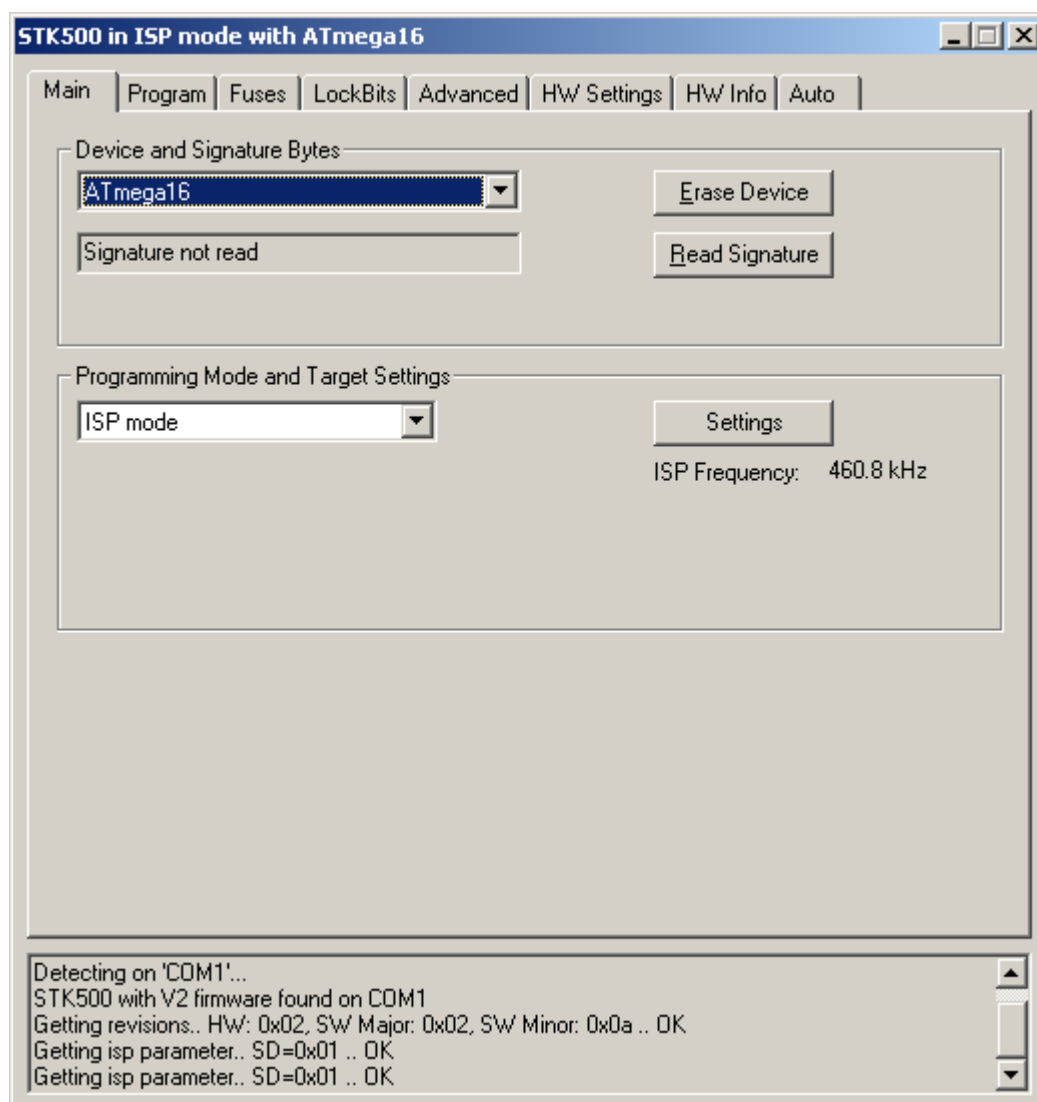


Рис. 1.10. Окно программатора и выбора МК

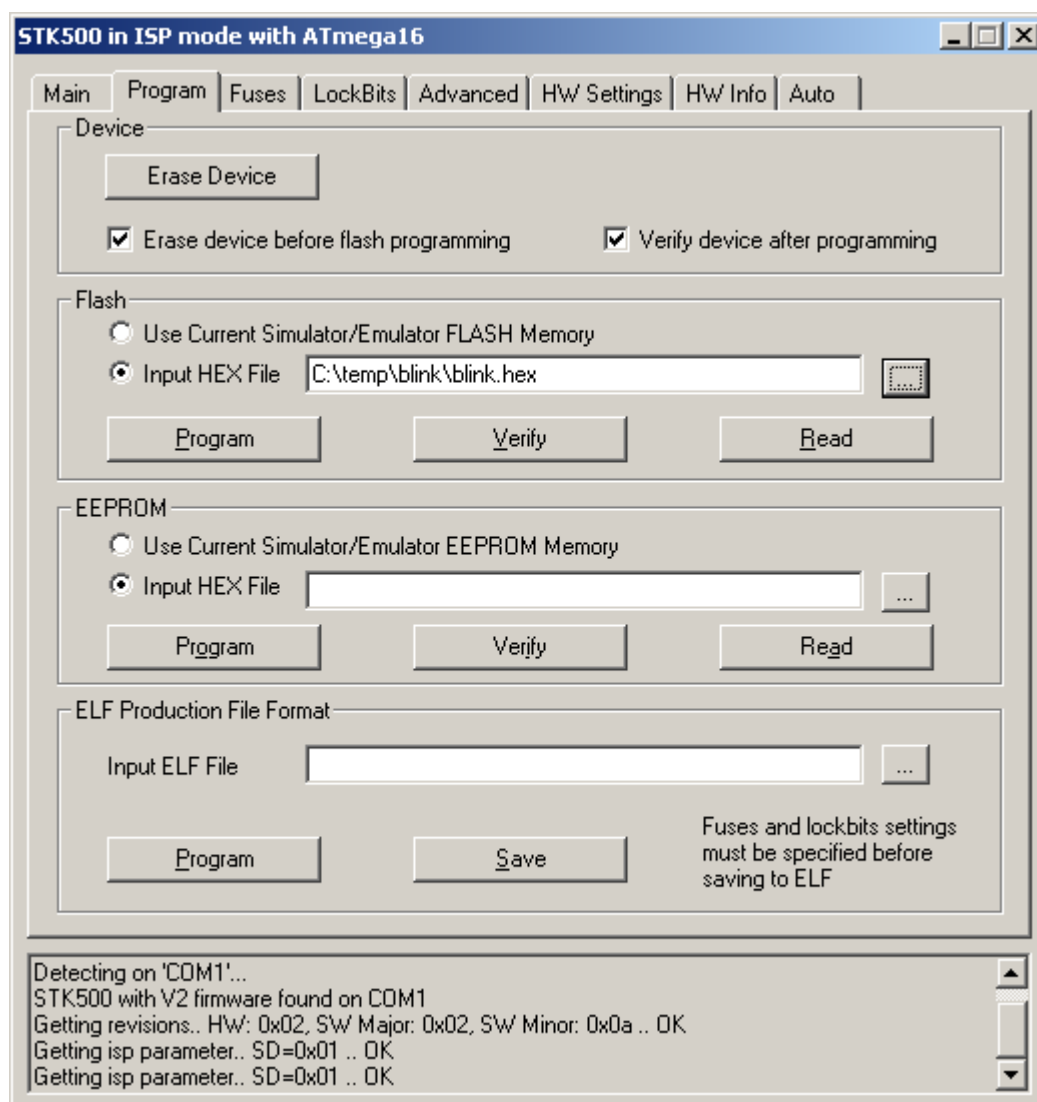


Рис. 1.11. Окно программатора, выбора файла программы и настройки программирования

После того как все настройки выполнены, нажмите кнопку «Program» в секции «Flash». В нижнюю часть окна будет выведен отчет о программировании МК, и каждая его строка будет оканчиваться словом «OK». А крайний правый светодиод (младший разряд) на плате STK500 будет мигать с частотой 1 Гц. Если хотя бы один из пунктов был нарушен, внимательно проверьте все вышеуказанные действия, а именно: правильность подключения плоского кабеля, тип программируемого МК, флажки управления программатором. Если Вы не можете найти неисправность самостоятельно, обратитесь к преподавателю или технику лаборатории.

1.4. Задания

№ варианта	Задание
1	Измените частоту мигания светодиода на 0.4 Гц, PORTB, номер разряда светодиода 1.
2	Измените частоту мигания светодиода на 0.6 Гц, PORTB, номер разряда

№ варианта	Задание
	светодиода 2.
3	Измените частоту мигания светодиода на 0.8 Гц, PORTB, номер разряда светодиода 3.
4	Измените частоту мигания светодиода на 1.2 Гц, PORTB, номер разряда светодиода 4.
5	Измените частоту мигания светодиода на 1.4 Гц, PORTB, номер разряда светодиода 5.
6	Измените частоту мигания светодиода на 1.6 Гц, PORTB, номер разряда светодиода 6.
7	Измените частоту мигания светодиода на 1.8 Гц, PORTB, номер разряда светодиода 7.
8	Измените частоту мигания светодиода на 2.0 Гц, PORTA, номер разряда светодиода 0.
9	Измените частоту мигания светодиода на 2.2 Гц, PORTA, номер разряда светодиода 1.
10	Измените частоту мигания светодиода на 2.4 Гц, PORTA, номер разряда светодиода 2.
11	Измените частоту мигания светодиода на 2.6 Гц, PORTA, номер разряда светодиода 3.
12	Измените частоту мигания светодиода на 2.8 Гц, PORTA, номер разряда светодиода 4.
13	Измените частоту мигания светодиода на 3.0 Гц, PORTA, номер разряда светодиода 5.
14	Измените частоту мигания светодиода на 3.2 Гц, PORTA, номер разряда светодиода 6.
15	Измените частоту мигания светодиода на 3.4 Гц, PORTA, номер разряда светодиода 7.
16	Измените частоту мигания светодиода на 3.6 Гц, PORTA, номер разряда светодиода 0.
17	Измените частоту мигания светодиода на 3.8 Гц, PORTA, номер разряда светодиода 1.
18	Измените частоту мигания светодиода на 4.0 Гц, PORTA, номер разряда светодиода 2.
19	Измените частоту мигания светодиода на 4.2 Гц, PORTA, номер разряда светодиода 3.
20	Измените частоту мигания светодиода на 4.4 Гц, PORTA, номер разряда светодиода 4.
21	Измените частоту мигания светодиода на 4.6 Гц, PORTA, номер разряда светодиода 5.
22	Измените частоту мигания светодиода на 4.8 Гц, PORTA, номер разряда светодиода 6.
23	Измените частоту мигания светодиода на 5.0 Гц, PORTA, номер разряда светодиода 7.
24	Измените частоту мигания светодиода на 5.2 Гц, PORTD, номер разряда светодиода 0.
25	Измените частоту мигания светодиода на 5.4 Гц, PORTD, номер разряда светодиода 1.
26	Измените частоту мигания светодиода на 5.6 Гц, PORTD, номер разряда светодиода 2.

№ варианта	Задание
27	Измените частоту мигания светодиода на 5.8 Гц, PORTD, номер разряда светодиода 3.
28	Измените частоту мигания светодиода на 6.0 Гц, PORTD, номер разряда светодиода 4.
29	Измените частоту мигания светодиода на 6.2 Гц, PORTD, номер разряда светодиода 5.
30	Измените частоту мигания светодиода на 6.4 Гц, PORTD, номер разряда светодиода 6.

2. ЛАБОРАТОРНАЯ РАБОТА №2. ПОРТЫ ВВОДА-ВЫВОДА

Цель работы: изучить принципы работы с портами ввода-вывода на примере опроса кнопок и управления светодиодами.

2.1. Теория

В л.р. №1 мы познакомились с простейшей организацией вывода информации, когда написали программу, заставляющую мигать светодиод. Т.к. порты МК восьмиразрядные (т.е. имеют 8 линий), мы можем подключить восемь кнопок и светодиодов, которые размещены на отладочной плате. Для этого необходимо двумя плоскими шлейфами соединить попарно разъемы «PORTA» и «SWITCHES», «PORTB» и «LEDS» соответственно. Поставим задачу: опрашивать все кнопки. Если какая-либо кнопка нажата, то включаем светодиод в соответствующем разряде, если отпущена – выключаем. Например, если нажата кнопка, подключенная к разряду 7 PORTA, то включаем светодиод, также подключенный к разряду 7 PORTB.

Рассмотрим схему подключения одного светодиода на отладочной плате STK500 (рис. 2.1). Остальные 7 светодиодов имеют аналогичное подключение.

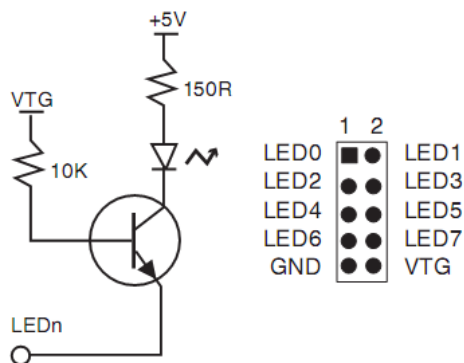


Рис. 2.1. Схема подключения одного светодиода

Из схемы видно, что включение светодиода происходит в том случае, если на клемму LEDn будет подан лог. 0.

Рассмотрим схему подключения одной кнопки на плате STK500 (рис. 2.2). Остальные 7 кнопок подключены аналогично. Из схемы видно, что при отпущенной кнопке на клемме SWn присутствует лог. 1. При нажатой кнопке на этой клемме появится лог. 0. Таким образом, для решения поставленной ранее задачи мы должны считывать значение порта, к которому подключены кнопки, и выдавать его в порт, к которому подключены светодиоды, т.к. логические уровни нажатой кнопки и уровни включения светодиодов совпадают. Такое совпадение с точки зрения логики управления в общем случае случайность, хотя схемотехнически такое выполнение цепей кнопок и светодиодов вполне обоснованно и не случайно.

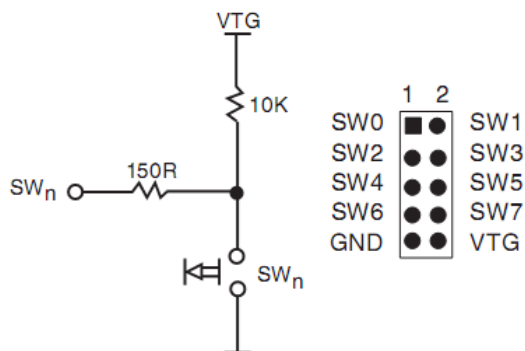


Рис. 2.2. Схема подключения одной кнопки

2.2. Примеры программ

Создайте проект (см. л.р. №1) с любым названием и введите текст программы, комментарии можно не вводить.

```
; Программа обработки кнопок.
; Реализовано считывание информации с кнопок и управление
; светодиодами на ее основе.
; Рассчитано на частоту 8 МГц.
; Светодиоды подключены к PORTB.
; Кнопки подключены к PORTA.
;
; 20090110 – дата создания
;
; ИрГТУ, кафедра ОАМ, автор Якимов Ю.А. (АМ-06-2)

#include "m16def.inc"      ; Подключаем заголовочный файл,
                           ; который содержит
                           ; определения ресурсов
                           ; микроконтроллера

; Установка указателя стека SP
; Т.к. все регистры 8-битные, а адрес ОЗУ 16-битное
; число, то указатель стека инициализируется в два
; этапа.

ldi r16, low(RAMEND); Загружаем младший байт
                           ; последнего адреса ОЗУ в регистр
                           ; r16 (константа RAMEND
                           ; определена в m16def.inc)
out  SPL, r16             ; Пересылаем значение из r16 в
                           ; SPL
ldi r16, high(RAMEND)     ; Загружаем старший байт
                           ; последнего адреса ОЗУ в
```

```
                                ; регистр r16
out SPH, r16                    ; Пересылаем значение из r16
                                ; в SPH
; Теперь указатель стека инициализирован числом 0x45F
; (для ATmega16)

; Настраиваем все линии PORTB на выход
ldi r16, 0xff
out DDRB, r16                  ; В регистре DDRB все разряды
                                ; установлены в 1
; Устанавливаем лог. 1 во всех разрядах PORTB
; (светодиоды не горят)
ldi r16, 0xff
out PORTB, r16

; Настраиваем все линии PORTA на вход
ldi r16, 0
out DDRA, r16                  ; В регистре DDRA все разряды
                                ; установлены в 0
; Устанавливаем лог. 1 во всех разрядах PORTA
; (подтягивающие резисторы подключены)
ldi r16, 0xff
out PORTA, r16

; На этом этапе конфигурация
; периферии микроконтроллера завершена

; Реализация программы
_main_cycle:                    ; главный цикл
in r16, PINA                   ; считываем состояние кнопок
out PORTB, r16                  ; выводим полученную информацию
                                ; в порт управления светодиодами
rjmp      _main_cycle          ; главный цикл
```

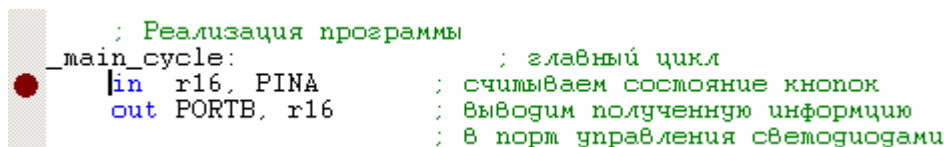
2.3. Отладка программы

Настройте симулятор и запустите отладку программы (л.р. №1, п. 1.2). После трансляции программы убедиться, что нет сообщений об ошибках или предупреждений.

Познакомимся с понятием «точка останова». Установим курсор на строке

```
in r16, PINA                   ; считываем состояние кнопок
```

и выберем пункт «Toggle Breakpoint». В строке слева появится темнокрасный кружок (рис. 2.3), обозначающий точку останова.



```

; Реализация программы
main_cycle: ; главный цикл
    in r16, PINA ; считываем состояние кнопок
    out PORTB, r16 ; выводим полученную информацию
    ; в порт управления светодиодами

```

Рис. 2.3. Точка останова

Выберете пункт меню «Debug->Reset» для сброса микроконтроллера. Теперь исполнение программы гарантированно начнется сначала. Выберите пункт меню «Debug->Run». Симулятор исполнит всю программу от начала до строки с точкой останова и остановится на ней. Как Вы уже поняли, назначение «точки останова» состоит в том, чтобы предоставить программисту возможность выполнить какой-либо участок кода без его участия (если код длинный или выполняется продолжительное время). Как только симулятор встретит точку останова, выполнение программы прекращается и симулятор ждет дальнейших действий программиста. В окне «I/O View» выберете пункт «PORTA». В окне ниже будут отображены все регистры, относящиеся к этому порту. Согласно п. 1.2.2 (л.р. №1), регистр PINA содержит реальные значения логических уровней на линиях порта. Щелчком левой клавиши мыши по всем разрядам регистра PINA установим лог. 1. Таким образом, согласно рис. 2.2, имитируется ситуация, когда все кнопки разомкнуты (подтягивающий резистор создает лог. 1 на линии порта). Выполним один шаг по программе. В регистр r16 будет скопировано содержимое регистра PINA. В окне «I/O View» выберем пункт PORTB. Выполним еще один шаг по программе. Во все разряды регистра PORTB будут также записаны лог. 1. Далее в программе встречается команда `rjmp`, которая замыкает главный цикл. Снова установим желтую стрелочку на строку

```
in r16, PINA ; считываем состояние кнопок
```

т.е. микроконтроллер готов к считыванию состояний кнопок. В регистре PINA установим любую комбинацию логических уровней, например, как показано на рис. 2.4.



Рис. 2.4. Имитация нажатых кнопок (лог. 0)

Выполним два шага по программе. Откроем содержимое регистра PORTB (рис. 2.5).

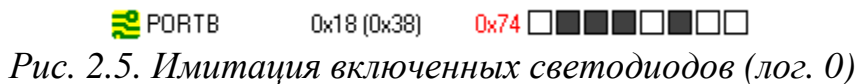


Рис. 2.5. Имитация включенных светодиодов (лог. 0)

Как видим, программа будет работать бесконечно долго, считывая данные с кнопок и выводя их на светодиоды. Каждая нажатая кнопка будет зажигать светодиод в своем разряде.

Программа демонстрирует простейшую реализацию ввода-вывода. Нужно заметить, что для реализации более сложных вариантов клавиатуры, где важно

именно количественно нажатие кнопок, нужно учитывать так называемый дребезг – многократное быстрое замыкание-размыкание контактов кнопки. Явление дребезга можно наблюдать, например, в микрокалькуляторах, когда нажимаемая цифра выводится на дисплей несколько раз. Вместо кнопок могут быть подключены практически любые контактные датчики, источники данных с выходным логическим уровнем. Вместо светодиодов могут быть подключены транзисторные ключи, которые могут управлять мощной нагрузкой.

Приведенный выше пример программы показывает, как считать информацию с порта, не анализируя отдельных его разрядов. В большинстве случаев такой анализ необходим, т.к. нужно знать состояние каждой кнопки (датчика и другого источника сигнала), а не их совместное значение.

Проверить значение любого разряда порта ввода-вывода можно с помощью двух команд:

2.3.1. `sbis` (Skip if Bit in I/O Register is Set) – пропустить следующую команду, если бит в регистре ввода-вывода установлен (лог. 1).

2.3.2. `sbic` (Skip if Bit in I/O Register is Cleared) – пропустить следующую команду, если бит в регистре ввода-вывода сброшен (лог. 0).

Пример использования этих двух команд проиллюстрируем программой, которая будет опрашивать две кнопки «Пуск» и «Стоп», подключенные к разрядам 0 и 1 PORTA соответственно. По нажатию кнопки «Пуск» светодиод, подключенный к разряду 0 PORTB должен загореться, а по нажатию кнопки «Стоп» - погаснуть. Остальные 6 кнопок в программе не опрашиваются, и их нажатие не играет роли. Текст программы с метки `_main_cycle` приведен ниже, для проверки работы можно заменить им фрагмент предыдущей программы.

```
_main_cycle:                ; главный цикл
    sbi    PORTB, 0          ; гасим светодиод

    sbic    PINA, 0          ; опрос кнопки "Пуск", если лог.
                             ; 1, то
    rjmp    PC-1             ; возвращаемся на одну команду
                             ; назад

    cbi     PORTB, 0          ; иначе – зажигаем светодиод

    sbis    PINA, 1          ; и опрашиваем кнопку "Стоп",
                             ; если лог. 0, то
    rjmp    _main_cycle      ; возвращаемся в главный цикл
    rjmp    PC - 2           ; иначе – возвращаемся на две
                             ; команды назад

Запустите симулятор. Поставьте точку останова на строке
    sbi     PORTB, 0          ; гасим светодиод
```

и запустите исполнение программы. При помощи симулятора убедитесь в правильной работе программы. Не забывайте, что при отпущенных кнопках на линиях PINA присутствует лог. 1.

Запрограммируйте (см. л.р. №1, п. 1.3) МК обоими вариантами программы. Программа будет работать на реальном МК точно также, как и в симуляторе.

2.4. Задания

№ варианта	Задание
1	Напишите программу, которая опрашивает все кнопки и по нажатию любой из них, включает светодиод в соответствующем разряде. При повторном нажатии кнопки соответствующий светодиод должен гаснуть. Для устранения дребезга используйте задержку (примерно 20 мс) после опроса кнопки.
2	Напишите программу, которая опрашивает две кнопки «Быстрее» и «Медленнее». Обе кнопки должны управлять скоростью мигания светодиода. При многократном нажатии или удерживании кнопки «Быстрее» частота мигания светодиода должна возрастать. При достижении максимальной частоты, она должна сбрасываться до минимального значения. Границы минимальных и максимальных частот, а также шага регулирования выбираются студентом. Для устранения дребезга используйте задержку (примерно 20 мс) после опроса кнопки.
3	Напишите программу, которая включает светодиод только в том случае, если все 8 кнопок нажаты в определенной (кодовой последовательности). Если хотя бы одна кнопка нажата ошибочно, или повторно, то загорается светодиод «Ошибка», а вся процедура ввода должна начинаться сначала (все верно нажатые кнопки «забываются»). Для устранения дребезга используйте задержку (примерно 20 мс) после опроса кнопки.
4	Напишите программу, которая опрашивает две кнопки «Больше» и «Меньше». При нажатии кнопки «Больше» линейка светодиодов должна «заполняться» горящими светодиодами, по одному на каждое нажатие. При нажатии кнопки «Меньше» в линейке должен погасать каждый правый горящий светодиод. Для устранения дребезга используйте задержку (примерно 20 мс) после опроса кнопки.
5	Напишите программу, которая опрашивает одну кнопку и по ее нажатию поочередно зажигает светодиоды, т.е. реализует эффект управляемого бегущего огня. Для устранения дребезга используйте задержку (примерно 20 мс) после опроса кнопки.
6	Напишите программу «секундомер». По нажатию кнопки «Старт» начинается посекундный отсчет с выводом количества прошедших секунд в двоичном виде на светодиоды. Кнопка «Пауза» должна останавливать отсчет, продолжение отсчета должно выполняться по нажатию кнопки «Пауза» или кнопки «Старт». Кнопка «Сброс» должна обнулять показания секундомера. Для устранения дребезга используйте задержку (примерно 20 мс) после опроса кнопки.
7	Напишите программу, которая реализует небольшую коллекцию световых эффектов (3 – 5 будет достаточно). Две кнопки («Вперед» и «Назад») должны перебирать эффекты по порядку. Две других кнопки («Быстрее» и «Медленнее») должны управлять скоростью переключения светодиодов. Для устранения дребезга используйте задержку (примерно 20 мс) после опроса кнопки.
8	Реализуйте игру «Кто быстрее?». По нажатию кнопки «Начать игру» программа должна спустя некоторое время зажигать светодиод. Двое игроков должны по возможности быстрее нажать каждый свою кнопку. Побеждает тот, кто сделает

№ варианта	Задание
	это быстрее своего соперника. Напротив кнопки выигравшего должен замигать светодиод с частотой 10 Гц. Напротив кнопки проигравшего – с частотой 2 Гц. Начать новую игру можно нажатием кнопки «Начать игру». Для устранения дребезга используйте задержку (примерно 20 мс) после опроса кнопки.
9	Реализуйте реверсивный двоичный счетчик. Программа должна опрашивать четыре кнопки «+», «-», «RESET» и «SAVE». По нажатию кнопки «+» счетчик должен инкрементироваться, по нажатию кнопки «-» декрементироваться, его значение – выводиться на светодиоды. По нажатию кнопки «RESET» счетчик должен сбрасываться. По нажатию кнопки «SAVE» текущее значение счетчика запоминается, а счетчик сбрасывается. При дальнейшем счете счетчик должен всегда сбрасываться, когда его значение будет достигать ранее сохраненного. Для устранения дребезга используйте задержку (примерно 20 мс) после опроса кнопки.
10	Напишите программу, которая зажигает любой светодиод при одновременном нажатии и удерживании только определенных кнопок и гасит – при отпускании. Если в этот момент нажаты и другие кнопки, то светодиод зажигаться не должен.
11	То же самое, что и п. 10, только при нажатии кнопок, не предусмотренных комбинацией, должен мигать «аварийный» светодиод с частотой 1 Гц.
12	Напишите программу, которая случайно (по заранее задуманной комбинации) зажигает один из восьми светодиодов, и ожидает нажатия кнопки под включенным светодиодом в течение 1 с. Если соответствующая кнопка была нажата в течение 1 с, то также случайно зажигается следующий светодиод и ожидается нажатие кнопки под ним в течение 1 с. Цикл повторяется бесконечно. Количество случайных комбинаций не следует брать более 20. Если нажимается кнопка не под включенным светодиодом, либо интервал в 1с истекает до нажатия, то все светодиоды мигают с частотой 1 Гц до перезапуска МК.
13	Напишите программу, которая отображает эффект «бегущей змейки» из двух светодиодов. Змейка начинает бежать в любую сторону от середины к одному из краев. Необходимо успеть нажать кнопку в тот момент, когда «голова» змейки (т.е. крайний светодиод) окажется над кнопкой. После нажатия кнопки змейка меняет направление и бежит в другую сторону, где необходимо также успеть нажать кнопку. С каждым новым направлением скорость змейки увеличивается. Если кнопка была нажата раньше или позже, чем «голова» змейки окажется над ней, то игра начинается сначала. На какой-либо максимальной скорости «змейки» игра заканчивается победой и все светодиоды мигают с частотой 5 Гц. Эта скорость определяется экспериментально так, чтобы глаз успевал различать «змейку». Кнопки используются крайняя левая и крайняя правая (т.е. разряд 0 и 7).
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	

№ варианта	Задание
25	
26	
27	
28	
29	
30	

3. ЛАБОРАТОРНАЯ РАБОТА №3. ПОСЛЕДОВАТЕЛЬНЫЙ ПОРТ USART

Цель работы: изучить принцип работы с последовательным портом на примере организации связи с ПК.

3.1. Теория

Последовательный порт (далее просто порт) представляет собой аппаратный модуль, расположенный на кристалле МК. Его основные возможности:

- 3.1.1. Полнодуплексный режим работы.
- 3.1.2. Гибкая настройка скорости передачи данных.
- 3.1.3. Количество бит данных 5, 6, 7, 8 и 1, 2 стоп-бита.
- 3.1.4. Генерация и проверка бита паритета.
- 3.1.5. Генерация прерываний по событиям.

Перечислены лишь основные возможности, которые будут использоваться в рамках лабораторного практикума. За полной информацией о работе порта следует обратиться к документации на соответствующий МК.

Для работы с портом его необходимо инициализировать, т.е. установить скорость передачи данных, количество бит данных и стоповых бит. Настройка осуществляется с помощью трех регистров контроля и статуса: UCSRA, UCSRB, UCSRC и двух регистров управления скоростью передачи UBRRH, UBRL. Назначение битов регистров управления подробно можно изучить в документации на соответствующий МК, мы же рассмотрим только те, которые будут использоваться нами.

Первым делом мы включим приемник и передатчик. После этого линии RXD и TXD МК будут работать только как линии последовательного порта, их использование в качестве линий порта ввода-вывода будет невозможным до отключения приемника и передатчика.

```
ldi r16, (1 << TXEN) | (1 << RXEN)
out UCSRB, r16          ; Включаем передатчик и приемник
```

Возможно, несколько необычная запись названия битов в скобках Вам встречается впервые. Поясим, что она обозначает. TXEN и RXEN – это константы, численное значение которых определено в файле m16def.inc. В этом легко убедиться, открыв его содержимое (для этого необходимо переключиться в окно «Project», открыть список «Included Files» и два раза кликнуть левой клавишей мыши по имени файла, рис. 3.1).

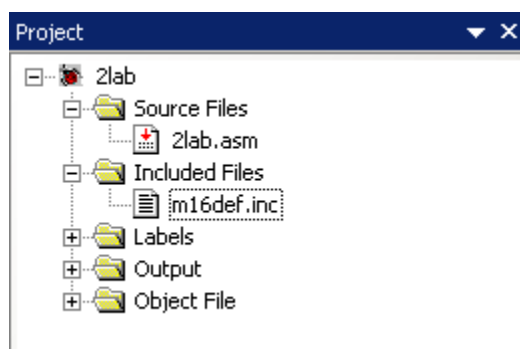


Рис. 3.1. Открытие подключаемого файла *m16def.inc*

Найдите с помощью поиска (меню «Edit -> Find...») строку «TXEN». Вы увидите подобную запись:

```
.equ TXEN = 3 ; Transmitter Enable
```

Она обозначает, что TXEN – это численная константа, равная трем. Аналогично определены и другие имена битов. Запись $(1 \ll TXEN)$ обозначает, что лог. 1 будет сдвинута влево на количество разрядов, которому равно значение константы TXEN. Таким образом, лог. 1 будет сдвинута влево на 3 разряда, что даст число 8. Константа RXEN равна 4. Лог. 1, сдвинутая влево на 4 разряда, даст число 16. Оператор «|» – это побитовая операция «ИЛИ». Следовательно, запись вида $(1 \ll TXEN) | (1 \ll RXEN)$ может быть сведена к записи $8 | 16$ или к записи $00001000 + 00010000$ в двоичном виде. В результате будет получено число 00011000. Как видно, в нем в лог. 1 установлены биты в разряде 3 и 4, что при записи в регистр UCSRA приведет к включению приемника и передатчика порта (см. описание регистра UCSRB в документации на МК). Такой вид записи значительно повышает наглядность программы: заранее определив понятными именами названия битов, можно очень быстро модифицировать программу, не запоминая какие номера битов за что отвечают. С первого взгляда можно понять, что и зачем записывается в регистр. Немного позже мы увидим, что подобным образом написана оставшаяся часть программы. Такого стиля оформления программы рекомендуется придерживаться всегда.

Установим длину данных 8 бит, установив в лог. 1 биты UCSZ1 и UCSZ0 в регистре UCSRC. Для доступа к регистру UCSRC необходимо также установить в лог. 1 бит URSEL.

```
ldi r16, (1 << UCSZ1) | (1 << UCSZ0) | (1 << URSEL)
out UCSRC, r16 ; 8 бит данных
```

Скорость работы порта рассчитывается по формуле:

$$BAUD = \frac{f_{osc}}{16(UBRR+1)}, \quad (3.1)$$

где: BAUD – значение скорости, бит/с;

f_{osc} – системная частота МК, Гц;

UBRR – совместное значение регистровой пары UBRRH:UBRRL (0 – 4095).

Соответственно значение UBRR для необходимой скорости можно посчитать по формуле:

$$UBRR = \frac{f_{osc}}{16 \cdot BAUD} - 1. \quad (3.2)$$

Рассчитаем значение UBRR для скорости 19200 бит/с по формуле (3.2):

$$UBRR = \frac{8000000}{16 \cdot 19200} - 1 = 25,$$

с округлением до целого.

Рассчитаем погрешность установленной скорости. Вычислим реальную скорость по (3.1):

$$BAUD = \frac{8000000}{16(25+1)} = 19231,$$

найдем абсолютную погрешность

$$\frac{19231-19200}{19200} \cdot 100\% = 0,2\%,$$

с округлением до десятых. Такая погрешность допустима.

Запишем число 25 в регистровую пару UBRRH:UBRRL :

```
ldi r16, high(25)      ; Скорость 19200 бод
out UBRRH, r16
ldi r16, low(25)       ; для системной частоты
                        ; микроконтроллера 8 МГц
out UBRRL, r16
```

Макросы high() и low() предназначены для извлечения старшего и младшего байта от 16-разрядного числа. Естественно, что старший байт от числа 25 будет равен нулю. Тем не менее, не следует опускать извлечение старшего байта даже в этих случаях. На этом инициализация последовательного порта полностью окончена и он готов к работе.

Для приема и передачи байта служит регистр UDR. Запись в него позволяет передать байт, а чтение из него – принять. Узнать, можно ли записывать данные или считывать их можно с помощью флагов, которые находятся в регистре UCSRA. Когда регистр UDR пуст и готов для записи новых отправляемых данных, флаг UDRE установлен в лог. 1. Это служит верным признаком разрешения записи байта, который нужно передать. Обычно последовательность команд, позволяющих передать байт, выглядит следующим образом:

```
sbis UCSRA, UDRE ; Если бит UDRE = 0, то регистр UDR
                  ; не пуст
rjmp PC - 1      ; возвращаемся на шаг назад (на
                  ; проверку бита)
out UDR, r16     ; иначе, выводим байт
```

Когда байт принят и находится в регистре UDR, флаг RXC устанавливается в лог. 1. Это служит верным признаком, что можно считать принятый байт. Ниже приведен код, позволяющий удостовериться в приходе байта, а затем считать его:

```
sbis UCSRA, RXC ; Если бит RXC = 0, то байт не принят
```

```
rjmp PC - 1
in r16, UDR          ; Иначе принимаем байт
```

На этом знакомство с последовательным портом можно завершить и перейти к рассмотрению примеров.

3.2. Примеры программ

Создайте проект и скопируйте текст программы (оригинальный файл также находится в каталоге `uart`):

```
; Демонстрация работы с последовательным портом (модуль
; USART) .
; Порт настроен на скорость 19200 бит/с, 8 бит данных,
; 1 стоповый бит.
; Программа принимает один байт, инкрементирует его
; и отправляет назад.
; Для проверки программы запустите на компьютере любой
; эмулятор терминала, например
; Hyper Terminal для Windows.
; Работа программы рассчитана на системную частоту 8 МГц.
; Дата 04-02-2008.
;
; Автор: Якимов Юрий (АМ-06-2) .
```

```
.include "m16def.inc"
.org 0x0000          ; Начало кода по адресу 0

; Инициализируем стек
ldi r16, low(RAMEND)
out SPL, r16
ldi r16, high(RAMEND)
out SPH, r16

ldi r16, (1 << TXEN) | (1 << RXEN)
out UCSRB, r16      ; Включаем передатчик и приемник
ldi r16, (1 << UCSZ1) | (1 << UCSZ0) | (1 << URSEL)
out UCSRC, r16      ; 8 бит данных

ldi r16, high(25)    ; Скорость 19200 бод
out UBRRH, r16
ldi r16, low(25)     ; для системной частоты
                    ; микроконтроллера 8 МГц
out UBRL, r16

; Выводим в порт строку "RS-232 Demo"
```

```
ldi r16, 'R'
rcall rs232_putc
ldi r16, 'S'
rcall rs232_putc
ldi r16, '-'
rcall rs232_putc
ldi r16, '2'
rcall rs232_putc
ldi r16, '3'
rcall rs232_putc
ldi r16, '2'
rcall rs232_putc
ldi r16, ' '
rcall rs232_putc
ldi r16, 'D'
rcall rs232_putc
ldi r16, 'e'
rcall rs232_putc
ldi r16, 'm'
rcall rs232_putc
ldi r16, 'o'
rcall rs232_putc
```

```
; Возвращаем курсор в начало строки
ldi r16, '\r'
rcall rs232_putc
; Переводим курсор на следующую строку
ldi r16, '\n'
rcall rs232_putc
; Выводим символ @
ldi r16, '@'
rcall rs232_putc
```

```
; Главный цикл программы
_main_cycle:
rcall rs232_getc      ; Ожидаем любой байт
inc r16               ; Увеличиваем его на 1
rcall rs232_putc      ; И выводим назад в
                      ; последовательный порт
rjmp _main_cycle
```

```
; Подпрограмма вывода байта в последовательный порт
rs232_putc:
sbis UCSRA, UDRE      ; Если бит UDRE = 0, то UART не
```

```
                                ; готов к отправке байта
rjmp PC - 1                    ; возвращаемся на шаг назад (на
                                ; проверку бита)
out UDR, r16                   ; иначе, выводим байт
ret

; Подпрограмма приема байта из последовательного порта
rs232_getc:
sbis UCSRA, RXC               ; Если бит RXC = 0, то байт не
                                ; принят
rjmp PC - 1
in r16, UDR                   ; Иначе принимаем байт
ret
```

Оттранслируйте программу, убедитесь, что нет ошибок и предупреждений. В дальнейшем, транслируя программы, обязательно проверяйте отчет транслятора. Он не должен содержать ошибок и предупреждений (рис. 1.5). Запрограммируйте МК. Выключите питание отладочной платы STK500 и подключите кабель последовательного порта к разъему «RS-232 SPARE» (рис. 1.9). Запустите программу Hyper Terminal (меню «Пуск -> Все программы -> Стандартные -> Связь -> Hyper Terminal»). Появится окно «Сведения о местонахождении», нажмите кнопку «Отмена». В окне «Предупреждение отмены» нажмите кнопку «Да». На предупреждение о том, что необходимо ввести сведения о местонахождении и т.п. нажмите кнопку «Ок». В окне «Описание подключения» Вы можете ввести любое название подключения и выбрать иконку подключения, затем нажмите кнопку «Ок». В окне «Сведения о местонахождении» нажмите кнопку «Отмена». Согласитесь с подтверждением отмены. В окне «Подключение» выберите порт «COM1» (рис. 3.2) и нажмите кнопку «Ок».

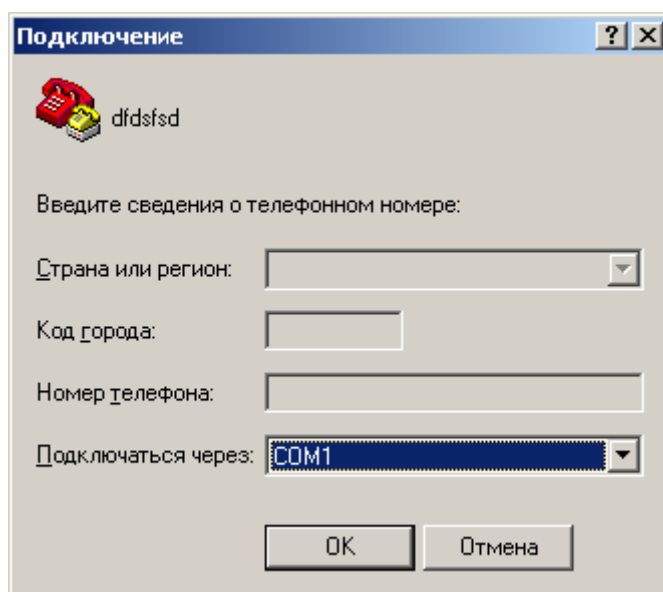


Рис. 3.2. Окно «Подключение» программы Hyper Terminal

В окне «Свойства: COM1» установите параметры, как показано на рис. 3.3 и нажмите кнопку «Ок». Убедитесь, что кнопка «Вызов» на панели инструментов нажата (рис. 3.4). Включите питание платы STK500. В главном окне терминала должны появиться строки приветствия (рис. 3.5). Если этого не произошло, внимательно проверьте правильность подключения всех кабелей или обратитесь к преподавателю или технику.

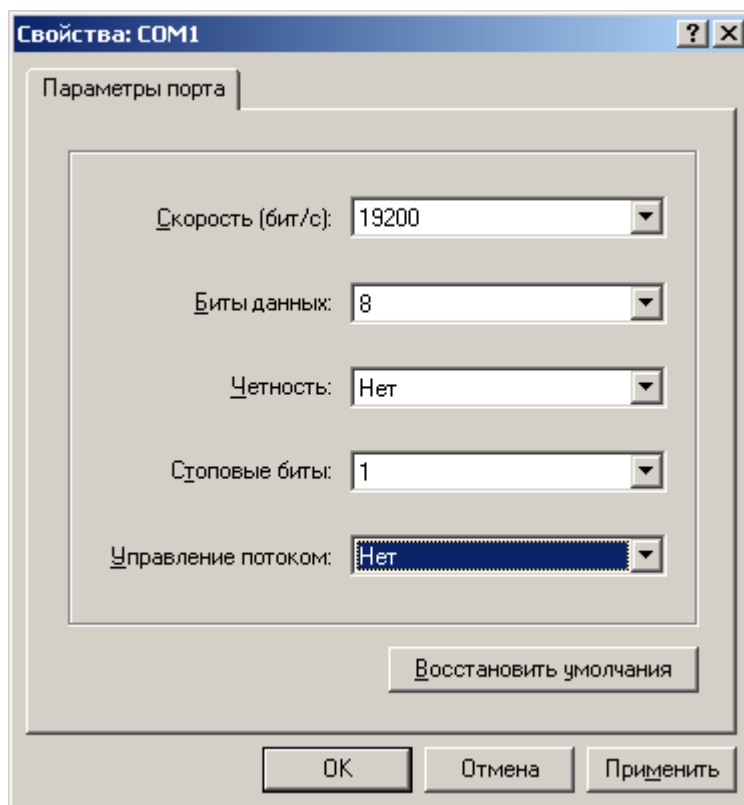


Рис. 3.3. Окно «Свойства: COM1» программы Hyper Terminal

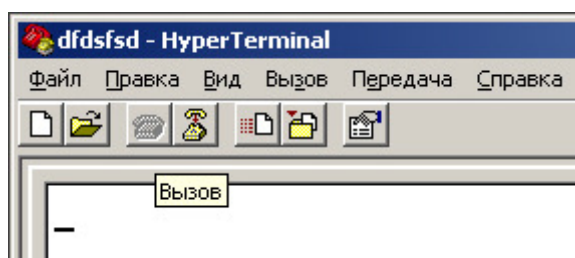


Рис. 3.4. Кнопка «Вызов» нажата

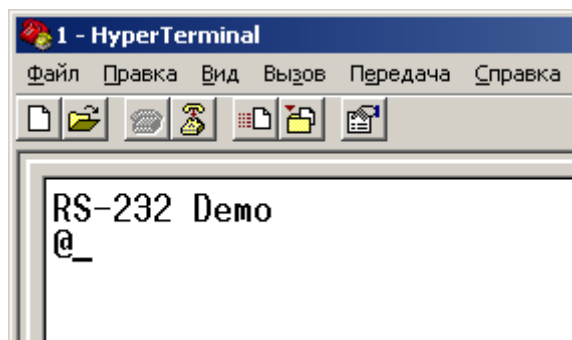


Рис. 3.5. Приветствие, выводимое программой микроконтроллера

Как уже выше было отмечено, программа ожидает поступление любого байта, затем инкрементирует его и посылает назад компьютеру. Нажмите на клавиатуре компьютера цифру «1». Вы увидите, что в терминале появилась цифра «2». Нажмите букву «d» - появится буква «e» (рис. 3.6). Мы видим, что наша программа работает.

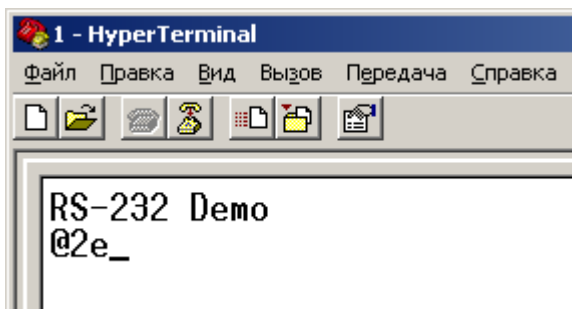


Рис. 3.6. Демонстрация работы программы

Слегка усложним задачу. Напишем программу, которая будет включать светодиод на 500 мс, номер которого определится нажатой цифрой на клавиатуре компьютера. А в терминале будем выводить номер нажатой кнопки на плате STK500. Под номером подразумеваются разряды байта от 0 до 7. Таким образом, на клавиатуре компьютера можно нажимать цифры 0 – 7. Если нажата любая другая клавиша, то зажигаются все светодиоды на 500 мс, а затем наснут. На плате STK500 крайняя правая кнопка соответствует разряду «0», крайняя левая – разряду «7». Наша новая программа в основном похожа на предыдущую, хотя и содержит новые элементы кода. Основная сложность заключается в том, что необходимо выполнять две задачи:

- 3.1. Ожидать байт с компьютера для его дальнейшего анализа и включения нужного светодиода, если была принята цифра.
- 3.2. Ожидать нажатия кнопок с целью отправить номер нажатой компьютеру.

Как видим нам необходимо *одновременно* выполнять две задачи. В этом случае использовать подпрограмму ожидания байта с последовательного порта в том виде, в каком она есть нельзя, потому что в этот момент микроконтроллер находится в цикле ожидания и не может опрашивать кнопки. Следовательно, нам

необходимо модифицировать программу. Это можно сделать следующим образом:

```
    ; Главный цикл программы
_main_cycle:
    sbis UCSRA, RXC      ; Если бит RXC = 0, то байт не
                        ; принят
    rjmp _button_scan    ; переходим на опрос кнопок
    in r16, UDR           ; Иначе принимаем байт
    ; Здесь мы анализируем принятый байт
    ; и выполняем необходимые действия
    ;
    ; тут необходим переход на начало программы
    rjmp _main_cycle

    ; Опрос кнопок
_button_scan:
    ; здесь выполняется опрос кнопок
    ; любым способом и необходимая информация
    ; передается компьютеру
    ;
    ; здесь необходим переход на начало программы
    rjmp _main_cycle
```

Основа программы создана и одновременно выполняет две задачи. Программа умышленно не наполнена кодом, чтобы нагляднее показать механизм многозадачности. Главный цикл программы начинается с того, что проверяется, принят ли байт по последовательному порту. Если байт принят, то происходит его анализ и в результате выполняется какое-либо действие, например, включается светодиод. После этого исполнение главного цикла начинается сначала. Если байт не принят, то микроконтроллер не ждет установки флага RXC, а просто переходит на метку `_button_scan`. С этой метки выполняется код, который проверяет состояние кнопок и если хотя бы одна нажата, то компьютеру отсылается ее номер. И главный цикл начинается сначала.

Допишем программу. Скопируйте ее в листинг проекта. Оттранслируйте программу и запрограммируйте МК. Для этого при отключенном питании платы STK500 не забудьте подключить кабель последовательного порта к разъему «RS232 CTRL»

```
    ; Демонстрация работы с последовательным портом
    ; (модуль USART) .
    ; Порт настроен на скорость 19200 бит/с, 8 бит данных,
    ; 1 стоповый бит.
    ; Программа демонстрирует простейшие принципы управления:
    ;   управления светодиодами с клавиатуры компьютера,
    ;   передачи информации о нажатых кнопках
    ;   на плате STK500 компьютеру.
```

```
; Программа зажигает на 500 мс светодиод, номер которого
; определяется нажатой цифрой на клавиатуре компьютера.
; Также в окно терминала выводится номер нажатой кнопки
; на плате STK500.
; Для проверки программы запустите на компьютере любой
; эмулятор терминала,
; например, Hyper Terminal для Windows.
; Работа программы рассчитана на системную частоту 8 МГц.
; Дата ??-02-2008.
;
; Автор: Якимов Юрий (АМ-06-2).
```

```
.include "m16def.inc"
.org 0x0000          ; Начало кода по адресу 0

; Инициализируем стек
ldi r16, low(RAMEND)
out SPL, r16
ldi r16, high(RAMEND)
out SPH, r16

ldi r16, (1 << TXEN) | (1 << RXEN)
out UCSRB, r16      ; Включаем передатчик и приемник
ldi r16, (1 << UCSZ1) | (1 << UCSZ0) | (1 << URSEL)
out UCSRC, r16      ; 8 бит данных

ldi r16, high(25)    ; Скорость 19200 бод
out UBRRH, r16
ldi r16, low(25)     ; для системной частоты
                    ; микроконтроллера 8 МГц
out UBRRL, r16

; Настраиваем все линии порта В на выход
ldi r16, 0xff
out DDRB, r16        ; В регистре DDRA все разряды
                    ; установлены в 1
; Устанавливаем лог. 1 на всех разрядах порта В
; (светодиоды не горят)
ldi r16, 0xff
out PORTB, r16

; Настраиваем все линии порта А на вход
ldi r16, 0
out DDRA, r16        ; А регистре DDRB все разряды
```

```
                                ; установлены в 0
ldi r16, 0xff
; Включаем внутренние подтягивающие резисторы на все
; линии PORTA
out PORTA, r16
; Инициализация периферии МК завершена

; Выводим в порт строку "RS-232 Control"
ldi r16, 'R'
rcall rs232_putc
ldi r16, 'S'
rcall rs232_putc
ldi r16, '-'
rcall rs232_putc
ldi r16, '2'
rcall rs232_putc
ldi r16, '3'
rcall rs232_putc
ldi r16, '2'
rcall rs232_putc
ldi r16, ' '
rcall rs232_putc
ldi r16, 'C'
rcall rs232_putc
ldi r16, 'o'
rcall rs232_putc
ldi r16, 'n'
rcall rs232_putc
ldi r16, 't'
rcall rs232_putc
ldi r16, 'r'
rcall rs232_putc
ldi r16, 'o'
rcall rs232_putc
ldi r16, 'l'
rcall rs232_putc

; Возвращаем курсор в начало строки
ldi r16, '\r'
rcall rs232_putc
; Переводим курсор на следующую строку
ldi r16, '\n'
rcall rs232_putc
; Выводим символ @
ldi r16, '@'
```

```
rcall rs232_putc
```

```
; Главный цикл программы
```

```
_main_cycle:
```

```
sbis UCSRA, RXC      ; Если бит RXC = 0, то байт не
                     ; принят
```

```
rjmp _button_scan    ; переходим на опрос кнопок
```

```
in r16, UDR           ; Иначе принимаем байт
```

```
cpi r16, '0'          ; это ASCII код цифры 0?
```

```
breq _led0_on         ; да, переход
```

```
cpi r16, '1'          ; это ASCII код цифры 1?
```

```
breq _led1_on         ; да, переход
```

```
cpi r16, '2'          ; далее аналогичные проверки
```

```
breq _led2_on
```

```
cpi r16, '3'
```

```
breq _led3_on
```

```
cpi r16, '4'
```

```
breq _led4_on
```

```
cpi r16, '5'
```

```
breq _led5_on
```

```
cpi r16, '6'
```

```
breq _led6_on
```

```
cpi r16, '7'
```

```
breq _led7_on
```

```
ldi r16, 0            ; среди принятых цифр нет
```

```
out PORTB, r16        ; включаем все светодиоды
```

```
rcall _delay_500ms    ; задержка 500 мс
```

```
com r16               ; инвертируем r16 (теперь в нем
                     ; лог. 1 во всех разрядах)
```

```
out PORTB, r16        ; выключаем все светодиоды
```

```
rjmp _main_cycle      ; возврат на начало
```

```
_led7_on:             ; код включения светодиода
```

```
cbi PORTB, 7          ; в 7 разряде
```

```
rcall _delay_500ms    ; задержка 500 мс
```

```
sbi PORTB, 7          ; выключаем светодиод
```

```
rjmp _main_cycle      ; возврат на начало
```

```
_led6_on:             ; далее аналогичные фрагменты
                     ; кода
```

```
cbi PORTB, 6
```

```
rcall _delay_500ms
```

```
sbi PORTB, 6
```

```
    rjmp _main_cycle
_led5_on:
    cbi PORTB, 5
    rcall _delay_500ms
    sbi PORTB, 5
    rjmp _main_cycle
_led4_on:
    cbi PORTB, 4
    rcall _delay_500ms
    sbi PORTB, 4
    rjmp _main_cycle
_led3_on:
    cbi PORTB, 3
    rcall _delay_500ms
    sbi PORTB, 3
    rjmp _main_cycle
_led2_on:
    cbi PORTB, 2
    rcall _delay_500ms
    sbi PORTB, 2
    rjmp _main_cycle
_led1_on:
    cbi PORTB, 1
    rcall _delay_500ms
    sbi PORTB, 1
    rjmp _main_cycle
_led0_on:
    cbi PORTB, 0
    rcall _delay_500ms
    sbi PORTB, 0
    rjmp _main_cycle

; Опрос кнопок
_button_scan:
    sbis PINA, 0          ; если в разряде лог. 0
    rjmp _btn0_pressed    ; то переходим, иначе пропуск
                           ; перехода
    sbis PINA, 1          ; и проверка следующего разряда
    rjmp _btn1_pressed    ; далее аналогичные проверки
    sbis PINA, 2
    rjmp _btn2_pressed
    sbis PINA, 3
    rjmp _btn3_pressed
    sbis PINA, 4
    rjmp _btn4_pressed
```

```
    sbis PINA, 5
    rjmp _btn5_pressed
    sbis PINA, 6
    rjmp _btn6_pressed
    sbic PINA, 7          ; если в разряде лог. 1
    rjmp _main_cycle     ; то переходим на начало, иначе

_btn7_pressed:          ; переходим сюда:
                        ; кнопка в 7разряде нажата
    ldi r16, '7'         ; заносим ASCII код цифры 7 в r16
    rcall rs232_putc     ; и передаем через
                        ; последовательный порт
    rjmp _main_cycle     ; возврат на начало
_btn6_pressed:
    ldi r16, '6'
    rcall rs232_putc
    rjmp _main_cycle
_btn5_pressed:
    ldi r16, '5'
    rcall rs232_putc
    rjmp _main_cycle
_btn4_pressed:
    ldi r16, '4'
    rcall rs232_putc
    rjmp _main_cycle
_btn3_pressed:
    ldi r16, '3'
    rcall rs232_putc
    rjmp _main_cycle
_btn2_pressed:
    ldi r16, '2'
    rcall rs232_putc
    rjmp _main_cycle
_btn1_pressed:
    ldi r16, '1'
    rcall rs232_putc
    rjmp _main_cycle
_btn0_pressed:
    ldi r16, '0'
    rcall rs232_putc
    rjmp _main_cycle
```

```
    ; Подпрограмма вывода байта в последовательный порт
rs232_putc:
```

```
sbis UCSRA, UDRE      ; Если бит UDRE = 0, то UART не
                        ; готов к отправке байта
rjmp PC - 1           ; возвращаемся на шаг назад (на
                        ; проверку бита)
out UDR, r16           ; иначе, выводим байт
ret

; Подпрограмма задержки на 500 мс
_delay_500ms:
    ldi r20, 0x15
_loop2:
    ldi r19, 0xff
_loop1:
    ldi r18, 0xf8
_loop0:
    dec r18
    brne _loop0
    dec r19
    brne _loop1
    dec r20
    brne _loop2
    ret
```

3.3. Отладка программы

Настройте программу HyperTerminal аналогично, как это было сделано выше. Включите питание платы STK500. В окне терминала должно появиться сообщение (рис. 3.7).

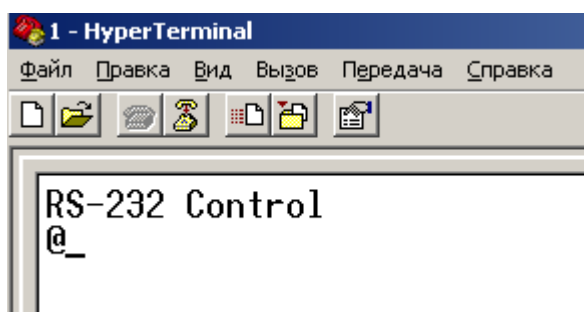


Рис. 3.7. Сообщение, выводимое программой

Нажимая цифры на клавиатуре компьютера, убедитесь, что светодиоды в соответствующих разрядах включаются примерно на 500 мс, затем гаснут. Проверьте реакцию светодиодов на остальные клавиши. Затем нажмите любую кнопку на отладочной плате STK500. В окно терминала будет выводиться цифра, соответствующая разряду кнопки (рис. 3.8).

Как Вы заметили, номер разряда будет выводиться до тех пор, пока нажата кнопка. С точки зрения удобства интерфейса это не красиво. Предлагаем Вам самостоятельно подумать над устранением этого недостатка программы.

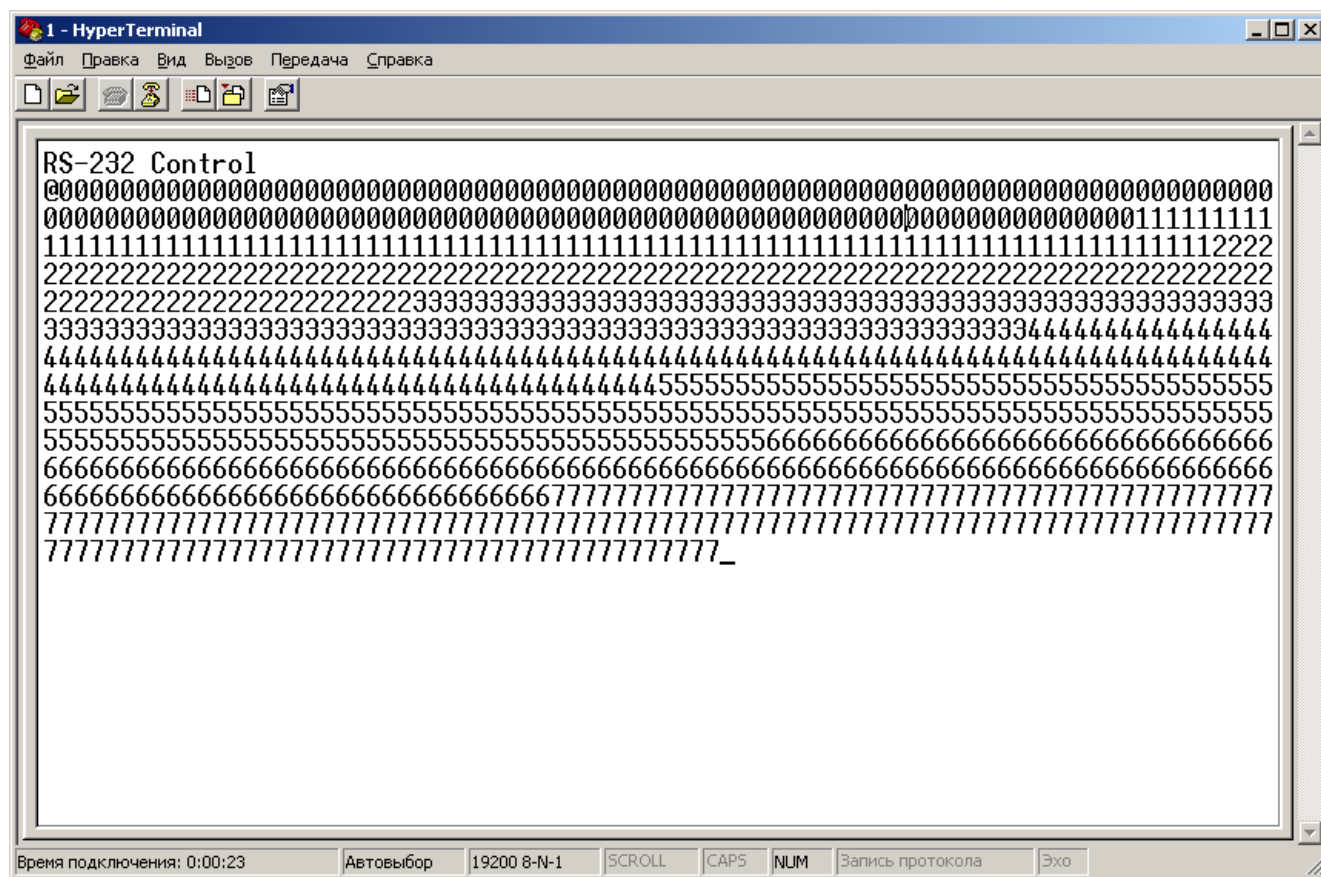


Рис. 3.8. Номера разрядов нажимаемых кнопок, выводимые в окно терминала

3.4. Задания

(Во всех заданиях предлагается настроить последовательный порт на определенную скорость передачи данных. При расчете делителя UBRR следует исходить из заданной скорости и системной частоты МК 8 МГц, также результата относительной погрешности установленной скорости. Если она превышает $\pm 0,2\%$, следует выбрать другую скорость, ближайшую к заданной. Процесс повторять до тех пор, пока ошибка не будет превышать $\pm 0,2\%$. Ряд скоростей: 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 бит/с)

№ варианта	Задание
1	Напишите программу, которая ожидает нажатия на клавиатуре компьютера любой цифры и выводит ее двоичный эквивалент на светодиоды. Если нажата любая другая клавиша (не цифра), то необходимо мигнуть всеми светодиодами 2 раза, с периодом 500 мс. Порт настройте на скорость 110 бит/с.
2	Напишите программу, которая имитирует эффект «бегущий огонь» на светодиодах. Для управления скоростью и направлением переключения светодиодов назначьте любые клавиши компьютера. Порт настройте на скорость 300 бит/с.
3	Напишите программу управляемых световых эффектов. С клавиатуры компьютера задаются включенные светодиоды. Т.е. для включения/выключения каждого светодиода можно назначить цифру, соответствующую номеру разряда

№ варианта	Задание
	светодиода. Таким образом, задается последовательность включенных светодиодов. По нажатию клавиши «г» указанная последовательность должна циклически сдвигаться влево «по кругу», при этом крайний левый разряд должен появляться в младшем правом разряде. Порт настройте на скорость 1200 бит/с.
4	Напишите программу, которая ожидает любой ASCII символ с компьютера. Если этот символ относится к буквам латинского или русского алфавита, то необходимо включить светодиод в разряде 0. Если этот символ относится к цифрам, то необходимо зажечь светодиод в разряде 1. В любом другом случае ошибку можно отметить, как в задании 1. Программа должна быть написана максимально коротко (проверки символов). Порт настройте на скорость 2400 бит/с.
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	

4. Лабораторная работа №4. 8-битный таймер/счетчик 0

Цель работы: изучить принцип работы 8-битного таймера, научиться его программировать и использовать в типовых задачах.

4.1. Теория

МК ATmega16 имеет три встроенных таймера/счетчика общего назначения:

4.1.1. 8-битный таймер 0 (8-bit Timer/Counter0 with PWM).

4.1.2. 16-битный таймер 1 (16-bit Timer/Counter1).

4.1.3. 8-битный таймер 2 (8-bit Timer/Counter2 with PWM and Asynchronous Operation).

Мы рассмотрим работу с таймерами на примере 8-битного таймера/счетчика 0. Более подробную информацию о режимах работы таймеров можно получить из документации [1].

Основные возможности таймера 0, которые будут использованы в лабораторной работе:

4.1.1. **Нормальный режим (Normal Mode).** В этом режиме счетчик всегда инкрементируется от 0 до максимального значения (0xFF или 255), затем сбрасывается и счет повторяется. В момент переполнения может быть вызвано прерывание.

4.1.2. **Режим обнуления по совпадению (Clear Timer on Compare Match (CTC) Mode).** В этом режиме максимальное значение, до которого инкрементируется счетчик, контролируется программно.

4.1.3. **Режим генерации ШИМ (PWM) сигнала.** В этом режиме при помощи таймера можно получить управляемый ШИМ сигнал на линии **OC0**.

Более подробно эти режимы работы с соответствующими примерами будут рассмотрены ниже.

В качестве источника тактовой частоты **таймера/счетчика 0** может быть использована системная частота МК, поданная через управляемый предделитель, либо внешняя частота, поданная на вход **T0**. Если не один источник тактовой частоты не выбран, то **таймер/счетчик 0** остановлен.

Источник тактовой частоты и коэффициент предделителя выбираются битами **CS02:0** в регистре **TCCR0** (см. табл. 4.3).

4.1.2. Режимы работы таймера

В различных режимах работы поведение счетчика и вывода **OC0** определяется состоянием битов **WGM01:0** и **COM01:0** в регистре **TCCR0** соответственно.

4.1.2.1. Нормальный режим (Normal Mode)

В этом режиме (**WGM01:0** = 0) счетчик всегда инкрементируется от 0 до своего максимального значения 0xFF (255 десятичное). После этого счетчик сбрасывается в 0 и счет продолжается заново. В момент сброса устанавливается

флаг **TOV0** в регистре **TIFR** и если разрешено прерывание по переполнению (установлен бит **OCIE0** в регистре **TIMSK**, разрешены глобальные прерывания), то оно будет вызвано. Флаг **TOV0** будет сброшен автоматически при выходе из обработчика прерывания (по команде **reti**). Также флаг можно сбросить программно, путем записи в него лог. 1. В этом режиме флаг **TOV0** выступает в роли 9 бита таймера, за исключением того, что его сброс нужно проводить принудительно. Однако, это не мешает программно увеличить разрядность счетчика для отсчета больших интервалов времени. Хотя более рационально использовать для этой цели 16-разрядный таймер.

4.1.2.2. Режим обнуления по совпадению (Clear Timer on Compare Match (CTC) Mode)

В этом режиме (**WGM01:0 = 2**) регистр **OCR0** используется для установки максимального значения, до которого счетчик инкрементируется. Счетчик сбрасывается в ноль, когда его значение становится равным значению, записанному в регистре **OCR0**. В этот момент также устанавливается флаг **OCF0** и может быть вызвано прерывание по совпадению. Более подробно см. п. 4.1.2.1, где описывается обработка прерывания.

Каждый раз при совпадении значения счетчика и регистра **OCR0** возможно управлять аппаратно линией **OC0** в соответствии с состоянием битов **COM01:0** в регистре **TCCR0** (табл. 4.1).

Таблица 4.1.

COM01	COM00	Закон изменения OC0
0	0	Не управляется таймером
0	1	Переключается в инверсное состояние по отношению к текущему
1	0	Очищается
1	1	Устанавливается

Для аппаратного управления линией **OC0** таймером, необходимо настроить ее на выход (см. л.р. 1).

Частоту управления линией **OC0** определяет следующее выражение:

$$f_{OC0} = \frac{f_{osc}}{2 \cdot N \cdot (1 + OCR0)}, \quad (4.1.2.2)$$

где: f_{OC0} - частота на выходе **OC0**, Гц;

f_{osc} - системная частота МК, Гц;

N - значение делителя (табл. 4.3).

$OCR0$ - значение регистра **OCR0**.

4.1.2.3. Режим быстрого ШИМ (Fast PWM)

В этом режиме (**WGM01:0 = 3**) счетчик считает от нуля до своего максимального значения 0xFF (255 десятичное). Затем он сбрасывается в 0 и

продолжает счет заново. Управление линией **OC0** в этом режиме происходит в соответствии с состоянием битов **COM01:0** в регистре **TCCR0** (табл. 4.2).

Таблица 4.2.

COM01	COM00	Закон изменения OC0
0	0	Не управляется таймером
0	1	Резервная комбинация битов
1	0	Очищается при совпадении значения счетчика и регистра OCR0, устанавливается при сбросе счетчика
1	1	Устанавливается при совпадении значения счетчика и регистра OCR0, очищается при сбросе счетчика

Для аппаратного управления линией **OC0** таймером, необходимо настроить ее на выход (см. л.р. 1).

Флаг **TOV0** устанавливается каждый раз, когда счетчик достигает своего максимального значения. Если это прерывание разрешено (см. п. 4.1.2.1), то оно будет вызвано и может использоваться для обновления значения регистра **OCR0**.

Частота ШИМ сигнала рассчитывается по формуле:

$$f_{OC0PWM} = \frac{f_{osc}}{N \cdot 256}, \quad (\text{ф. 4.1.2.3})$$

где: f_{OC0PWM} - частота ШИМ сигнала на линии **OC0**, Гц;

f_{osc} - системная частота МК, Гц;

N - значение предделителя (табл. 4.3).

Таблица 4.3. Описание битов выбора тактовой частоты таймера/счетчика 0

CS02	CS01	CS00	Описание
0	0	0	Нет источника частоты, таймер остановлен
0	0	1	$f_{osc} / 1$ (нет деления частоты)
0	1	0	$f_{osc} / 8$ (частота делится предделителем)
0	1	1	$f_{osc} / 64$ (частота делится предделителем)
1	0	0	$f_{osc} / 256$ (частота делится предделителем)
1	0	1	$f_{osc} / 1024$ (частота делится предделителем)
1	1	0	Частота подается на вход T0. Счет по срезу импульса.
1	1	1	Частота подается на вход T0. Счет по фронту импульса.

Обратите внимание, что если значение регистра **OCR0** равно нулю, то на выходе **OC0** будут появляться узкие импульсы (пики) каждый раз при переполнении таймера. Если значение регистра **OCR0** установлено в максимальное **0xFF**, то на выходе **OC0** будет постоянный высокий или низкий логический уровень (это зависит от состояния битов **COM01:0** (табл. 4.2.)).

4.2. Примеры программ

4.2.1. Нормальный режим (Normal Mode)

Рассмотрение примеров начнем с нормального режима работы таймера. Частота, с которой будет переполняться 8-битный таймер в нормальном режиме

$$f_{ovf} = \frac{f_{osc}}{N \cdot 2^m} \quad (4.2.1)$$

где: f_{ovf} - частота переполнения 8-битного таймера, Гц;

f_{osc} - системная частота МК, Гц;

N - значение предделителя;

m - количество разрядов таймера.

Минимальная частота переполнения таймера

$$f_{ovf} = \frac{8000000}{1024 \cdot 2^8} = 30,517578125 \text{ Гц}.$$

Поставим задачу: необходимо мигать светодиодом, подключенным к РВЗ с частотой 5 Гц. Период сигнала будет равен

$$T = \frac{1}{F} = \frac{1}{5} = 0,2 \text{ с}.$$

За один период логический уровень на выходе РВЗ должен два раза инвертироваться, следовательно, частота переполнения таймера должна быть в два раза выше, т.е. 10 Гц, а период – 0,1 с соответственно. Как показано выше, минимальная частота переполнения таймера 30,5... Гц. Однако, в п. 4.1.2.1 было замечено, что для отсчета больших интервалов времени возможно делить частоту переполнения таймера программно, путем введения дополнительной переменной. Именно такой способ решения проблемы и будет рассмотрен.

Учитывая, что высокая точность выдержки времени не требуется, округлим частоту переполнения таймера до 30,5 Гц. Она превышает расчетную в $30,5 / 10 = 3,5$ раза. Таким образом, необходимо отслеживать флаг переполнения **TOV0**, а каждые 3,5 раза его установки инвертировать выход РВЗ. Это невозможно. Поэтому округлим 3,5 в большую и меньшую стороны, и посчитаем реальные интервалы:

$$\frac{30,5}{4} = 7,63 \text{ Гц}; \text{ относительная погрешность } \frac{7,63-10}{10} \cdot 100\% = -23,7\%.$$

$$\frac{30,5}{3} = 10,17 \text{ Гц}; \text{ относительная погрешность } \frac{10,17-10}{10} \cdot 100\% = 1,7\%.$$

Как видим, при округлении числа переполнений таймера в меньшую сторону, относительная погрешность минимальна. Из этого следует, что мы должны делить частоту таймера на 3.

Создайте проект и скопируйте текст программы, который приведен ниже.

```
; Демонстрация работы с 8-битным таймером.  
; Таймер работает в нормальном режиме (Normal  
; Mode) с переполнением. Для увеличения разрядности
```

```
; и для понижения частоты использована дополнительная
; переменная. Таким образом, минимальная частота
; переполнения таймера делиться на 3 и
; снижается с 30,5 Гц до 10,17 Гц. Следовательно,
; светодиод, который используется для демонстрации,
; мигает с частотой 5,1 Гц.
; Работа программы рассчитана на системную частоту 8 МГц.
; Дата 06-04-2009.
;
; Автор: Якимов Юрий (АМ-06-2).
```

```
; Определяем константу с номером разряда
; к которому подключен светодиод
.equ LedBit = 3
```

```
; Подключаем заголовочный файл
.include "m16def.inc"
```

```
; Начало сегмента кода
.org 0x0000
```

```
    ; Инициализируем стек
    ldi r16, low(RAMEND)
    out SPL, r16
    ldi r16, high(RAMEND)
    out SPH, r16
```

```
    ; Предделитель N = 1024
    ldi r16, (1 << CS02) | (1 << CS00)
    out TCCR0, r16
```

```
    ; 3 разряд PORTB настраиваем на выход
    ldi r16, (1 << LedBit)
    out DDRB, r16
```

```
_main_cycle:
```

```
    ; дополнительно делим частоту
    ; переполнения таймера на 3
    ldi r17, 3
```

```
_loop0:
```

```
    ; Ожидаем переполнения таймера (флаг TOV0 = 1)
    in r16, TIFR
    sbrs r16, TOV0
    rjmp _loop0
```

```

; программно сбрасываем флаг прерываний,
; записывая в него лог. 1
out TIFR, r16

dec r17
brne _loop0

; инвертируем разряд светодиода
sbis PORTB, LedBit
rjmp _log1
cbi PORTB, LedBit
rjmp _main_cycle
_log1:
sbi PORTB, LedBit
rjmp _main_cycle

```

Оттранслируйте программу и загрузите ее в память программ МК. Убедитесь, что светодиод в 3 разряде PORTB мигает с частотой примерно 5 Гц. Если этого не произошло, проверьте, соединены ли плоским кабелем разъемы «PORTB» и «LEDS».

Проверьте работу программы в симуляторе. Для этого поставьте точку останова (л.р. №2) на строку `dec r17` (рис. 4.1).

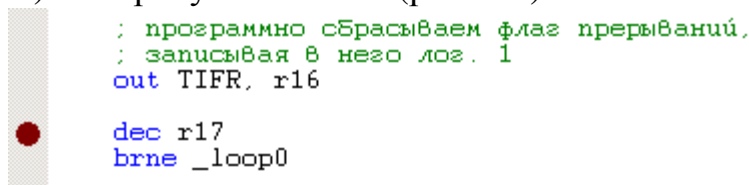


Рис. 4.1. Точка останова

Сбросьте микроконтроллер и счетчик «Stop Watch». Запустите программу, когда желтая стрелка остановится напротив точки останова, проверьте, что время выполнения фрагмента кода равно 32,77 мс (рис. 4.2). Повторите действие еще два раза. После этого значение регистра R17 станет равным единице. В чем легко убедиться, открыв окно «View -> Register». Далее программу выполняйте пошагово, потому что значение регистра r17 станет равным нулю и переход по метке не будет выполнен. Посмотрите, как инвертируется значение третьего разряда PORTB. После этого, можно снова запустить МК в режиме непрерывного исполнения программы три раза (до R17 равным единице) и, снова выполнив пошагово программу, убедиться, что произошла инверсия третьего бита в PORTB.

Stop Watch | 32.77 ms

Рис. 4.2. Счетчик Stop Watch

Таким образом, мы убедились, в работоспособности программы, как в симуляторе, так и на реальном МК.

4.2.2. Режим обнуления по совпадению (CTC Mode)

Несколько усложним нашу задачу: потребуем регулировки частоты мигания светодиода. Эту функцию можно реализовать в нормальном режиме работы таймера программно, но мы воспользуемся для этого режимом обнуления по сравнению. Ведь именно в этом режиме аппаратно реализована возможность дискретно изменять разрешающую способность таймера с шагом $\frac{1}{256}$.

Таким образом, при известном значении регистра **OCR0**, частота совпадений в Гц, а следовательно, и установки флага **OCF0**, определится выражением:

$$f_{COMP} = \frac{f_{osc}}{N(OCR0+1)}. \quad (\text{ф. 4.2.2})$$

Тогда диапазон частот совпадений (установки флага **OCF0**), которые можно получить при изменении значения регистра OCR0 от 255 до 0 соответственно, равен (выражение ф. 4.2.2):

$$f_{COMP\min} = \frac{8000000}{1024(255+1)} = 30,52 \text{ Гц},$$

$$f_{COMP\max} = \frac{8000000}{1024(0+1)} = 7812,5 \text{ Гц}.$$

Учитывая рассуждения, приводимые в п. 4.2, необходимо эти частоты поделить на 2, получим 15,26 Гц и 3906,25 Гц соответственно. Как видно, минимальная частота достаточно велика и практически не будет восприниматься человеческим глазом. Введя дополнительную переменную и деля частоты на 30, получим диапазон частот от 0,5 до 130 Гц. Дробные части отбрасываются, т.к. деление целочисленное. Ограничивая значение максимальной частоты, допустим до 10 Гц, получим плавную регулировку частоты мигания светодиода от 1 до 10 Гц.

Следуя всем рассуждениям, изложенным выше, конкретизируем задачу. Управлять частотой будем с помощью двух кнопок, подключенных к PA0 (кнопка «Уменьшить») и PA1 (кнопка «Увеличить»). При подаче питания на МК начальная частота мигания светодиода равна нулю. При нажатии на кнопку «Увеличить» частота будет плавно увеличиваться. При нажатии на кнопку «Уменьшить» - уменьшаться. Для защиты от дребезга воспользуемся задержкой на 20 мс. Ограничение максимальной частоты в 10 Гц будем выполнять отслеживанием значения в регистре **OCR0** (ф. 4.2.2):

$$2 \cdot 10 \cdot 30 = \frac{8000000}{1024(OCR0+1)},$$

откуда значение **OCR0** равно 12. Таким образом, мы должны изменять значение регистра OCR0 от 255 (минимальная частота) до 12 (максимальная частота).

Программа, выполняющая требуемые действия, показана ниже и основана на предыдущем примере.

- ; Демонстрация работы с 8-битным таймером.
- ; Таймер работает в режиме обнуления по совпадению (CTC mode).


```
; Реализован режим плавного изменения частоты мигания
; светодиода, подключенного к PB3.
; Кнопка "Уменьшить" (PA0) служит для плавного
; понижения частоты, кнопка "Увеличить" (PA1) –
; для плавного увеличения частоты.
; Работа программы рассчитана на системную частоту 8 МГц.
; Дата 24-06-2009.
;
; Автор: Якимов Юрий (АМ-06-2).
```

```
; Определяем константу с номером разряда
; к которому подключен светодиод
.equ LedBit = 3
```

```
; Аналогично определяем константы
; с номерами разрядов кнопок
.equ KeyDec = 0 ; кнопка "Уменьшить"
.equ KeyInc = 1 ; кнопка "Увеличить"
```

```
; Подключаем заголовочный файл
.include "m16def.inc"
```

```
; Начало сегмента кода
.org 0x0000
```

```
    ; Инициализируем стек
    ldi r16, low(RAMEND)
    out SPL, r16
    ldi r16, high(RAMEND)
    out SPH, r16
```

```
    ; Предделитель N = 1024, режим Normal
    ldi r16, (1 << CS02) | (1 << CS00) | (1 << WGM01)
    out TCCR0, r16
```

```
    ; 3 разряд PORTB настраиваем на выход
    ldi r16, (1 << LedBit)
    out DDRB, r16
```

```
    ; Настраиваем на ввод разряды с кнопками
    ; (обнуляя нужные разряды и не затрагивая остальные
    ; при помощи маски)
    in r16, PORTA
    andi r16, ~((1 << KeyDec) | (1 << KeyInc))
```

```
    out PORTA, r16

    ; При инициализации задаем минимальную частоту
    ; мигания светодиода
    ldi r16, 0xff
    out OCR0, r16

_main_cycle:
    ; дополнительно делим частоту
    ; переполнения таймера на 30
    ldi r17, 30
_working:
    ; Опрашиваем кнопки
    ; и ожидаем совпадения (флаг OCF0 = 1)
    sbis PINA, KeyDec
    rjmp _KeyDecProc
    sbis PINA, KeyInc
    rjmp _KeyIncProc

_AfterKey:
    in r16, TIFR
    sbrs r16, OCF0
    rjmp _working

    ; программно сбрасываем флаг прерываний,
    ; записывая в него лог. 1
    out TIFR, r16

    dec r17
    brne _working

    ; инвертируем разряд светодиода
    sbis PORTB, LedBit
    rjmp _log1
    cbi PORTB, LedBit
    rjmp _main_cycle
_log1:
    sbi PORTB, LedBit
    rjmp _main_cycle

; Обработка кнопки "Уменьшить"
_KeyDecProc:
    in r16, OCR0
    ; Если в регистре OCR0 находится число 255,
    ; то не уменьшаем частоту
```

```
    cpi r16, 0xff
    breq _AfterKey
    inc r16
    out OCR0, r16
    ; Подавлениедребезга контактов
    rcall delay_20ms
    rjmp _AfterKey

; Обработка кнопки "Увеличить"
_KeyIncProc:
    in r16, OCR0
    ; Если в регистре OCR0 находится число 12,
    ; то не увеличиваем частоту
    cpi r16, 0x0c
    breq _AfterKey
    dec r16
    out OCR0, r16
    ; Подавлениедребезга контактов
    rcall delay_20ms
    rjmp _AfterKey

; Подпрограмма задержки на 20 мс
delay_20ms:
    ldi r18, 0xff
_loop1:
    ldi r19, 0xff
_loop0:
    dec r19
    brne _loop0
    dec r18
    brne _loop1
    ret
```

Оттранслируйте текст программы и запрограммируйте МК. Светодиод в разряде PB3 должен мигать с частотой 4 Гц. С помощью кнопок PA0 и PA1 возможно изменять частоту мигания. Для того, чтобы эффект плавного изменения частоты был замечен глазу, нажатия на кнопки должны быть кратковременными. В начале диапазона увеличение частоты будет малозаметным. В конце диапазона – почти скачкообразным. Это происходит из-за того, что зависимость частоты от значения регистра OCR0 обратнопропорциональна.

В силу относительной простоты и знакомых элементов программы, мы не будем рассматривать ее работу на симуляторе. Вы можете это сделать самостоятельно.

4.2.3. Режим быстрого ШИМ (Fast PWM)

Последний режим работы таймера, который мы рассмотрим – это режим быстрого ШИМ (Fast PWM).

Кратко – задача ШИМ состоит в том, чтобы варьировать количеством мощности, отдаваемой в нагрузку. При этом используется прямоугольный сигнал с изменяемым коэффициентом заполнения. Чем он больше, тем больше мощности будет отдано нагрузке.

При помощи ШИМ можно управлять скоростью вращения коллекторного двигателя постоянного тока; яркостью свечения ламп; количеством тепла, отдаваемым обогревателем и т.д. Также можно генерировать различные по форме аналоговые сигналы (цифровым методом), например, воспроизводить речь. Мы рассмотрим пример изменения яркости свечения светодиода средствами ШИМ.

Поставим задачу плавного изменения яркости светодиода от минимума до максимума и обратно в бесконечном цикле.

В режиме быстрого ШИМ необходимо аппаратно управлять выводом **OC0**. Согласно ф. 4.1.2.3. выберем частоту ШИМ 488,3 Гц:

$$f_{OC0PWM} = \frac{8000000}{64 \cdot 256} = 488,3 \text{ Гц.}$$

Следует заметить, что частота ШИМ остается постоянной. Изменяется только коэффициент заполнения. В каталоге с программой «fast_pwm» находится видеоролик, снятый с экрана осциллографа. На видео видно плавное изменение коэффициента заполнения ШИМ сигнала.

Программа, реализующая эффект, показана ниже.

```
; Демонстрация работы с 8-битным таймером.
; Таймер работает в режиме быстрого ШИМ.
; Реализован эффект медленного увеличения,
; затем медленного уменьшения яркости светодиода
; в бесконечном цикле.
; Работа программы рассчитана на системную частоту 8 МГц.
; Дата 30-04-2009.
;
; Автор: Якимов Юрий (АМ-06-2).

; Определяем константу с номером разряда
; к которому подключен светодиод
.equ LedBit = 3

; Подключаем заголовочный файл
.include "m16def.inc"

; Начало сегмента кода
.org 0x0000
```

```
; Инициализируем стек
ldi r16, low(RAMEND)
out SPL, r16
ldi r16, high(RAMEND)
out SPH, r16

; Предделитель N = 8, режим Fast PWM
ldi r16, (1 << CS01) | (1 << WGM01) | (1 << WGM00) | (1
<< COM01)
out TCCR0, r16

; 3 разряд PORTB настраиваем на выход
; также это линия OC0
ldi r16, (1 << LedBit)
out DDRB, r16

_main_cycle:
; Изначально коэффициент заполнения 0,
; светодиод погашен.
ldi r16, 0
; Записываем 0 в регистр r27, это режим увеличения
; яркости. Если в регистре 1, то – режим уменьшения
; яркости.
ldi r27, 0
_working:
; Обновляем значение в регистре OCR0
out OCR0, r16
; Вызываем задержку с целью замедлить процесс
; изменения яркости
rcall delay_5ms

; Определяем режим работы (увеличение или
; уменьшение яркости светодиода)
cpi r27, 0
brne _bright_down

; Увеличение яркости
_bright_up:
; Проверяем достигло ли максимального
; значения содержимое регистра OCR0
cpi r16, 0xff
; Если достигло, то переходим на смену режима
; (уменьшение яркости)
breq _up_end
```

```

; Иначе – увеличиваем яркость
inc r16
rjmp _working
_up_end:
; Меняем режим
ldi r27, 1
rjmp _working

; Уменьшение яркости
_bright_down:
cpi r16, 0
breq _down_end
dec r16
rjmp _working
_down_end:
ldi r27, 0
rjmp _working

; Подпрограмма задержки на 5 мс
delay_5ms:
ldi r18, 0x3f
_loop1:
ldi r19, 0xff
_loop0:
dec r19
brne _loop0
dec r18
brne _loop1
ret

```

Как и предыдущая программа, текущая не нуждается в отладке на симуляторе, т.к. не содержит новых элементов. В случае необходимости Вы можете провести отладку самостоятельно.

4.3. Задания

№ варианта	Задание
1	Напишите программу, которая раз в 3 секунды будет посылать в последовательный порт состояние всех кнопок в виде x7 x6 x5 x4 x3 x2 x1 x0. Таймер используйте в режиме Normal Mode. Последовательный порт – в любом режиме.
2	Напишите программу, которая позволит плавно изменять яркость светодиода от нуля до максимума с помощью двух кнопок: «+» и «-» (каждую выберите самостоятельно). Таймер используйте в режиме Fast PWM.
3	Напишите программу, позволяющую изменять яркость светодиода, путем прямой записи значения в регистр OCR0 через последовательный порт, используя любой

№ варианта	Задание
	эмулятор терминала, например HyperTerminal. Число должно задаваться в десятичном виде от 0 до 255 и подтверждаться нажатием на клавишу ENTER. Таймер используйте в режиме Fast PWM. Последовательный порт – в любом режиме.
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	

5. ЛАБОРАТОРНАЯ РАБОТА №5. ПАМЯТЬ МК

Цель работы: освоить использование памяти SRAM, FLASH и EEPROM.

5.1. Теория

6. Лабораторная работа №6. Многозадачность.
7. Лабораторная работа №7.

Термины и сокращения

В этом разделе описаны термины и сокращения, встречающиеся в тексте.

1. МК – микроконтроллер. Обычно представляет собой законченное электронное устройство в виде СБИС, функционирование и алгоритм работы которой определяется программным обеспечением.
2. ПК – персональный компьютер. Другое название IBM PC.
3. JTAG (Joint Test Action Group) – общее название стандарта IEEE 1149.1. Описывает методы тестирования печатных плат. Аббревиатура также часто используется для названия порта отладки интегральных микросхем.
4. ISP (In-System Programming) – технология программирования микросхем, например микроконтроллеров, установленных в системе без их изъятия.
5. EEPROM (Electrically Erasable Programmable Read-Only Memory) – тип энергонезависимой памяти, обычно небольшого объема, стираемой и записываемой электрически. Она предназначена для хранения данных при выключенном питании устройства, таких, как калибровочные данные, настройки прибора и т.п.
6. FLASH – тип энергонезависимой памяти, которая стирается и программируется электрически. В отличие от памяти EEPROM, которая доступна побайтно, память FLASH стирается и программируется блоками, что позволяет произвести запись гораздо быстрее. Также она значительно дешевле, из-за чего и получила широкое распространение в тех случаях, когда необходимо хранить большое число данных, например в USB-накопителях, карманных компьютерах, мобильных телефонах, плеерах и т.п..
7. РОН – регистр общего назначения. Сверхбыстрая оперативная память, расположенная на кристалле микроконтроллера. РОНЫ предназначены для хранения промежуточных результатов вычислений, для хранения данных, доступ к которым должен быть максимально быстрым и т.п.
8. л.р. – лабораторная работа.
9. ШИМ (англ. PWM – Pulse Width Modulation) – широтно-импульсная модуляция.

Библиографический список

1. Документация на МК ATmega16 (8-bit Microcontroller with 16K Bytes In-System Programmable Flash).

Ресурсы

В этом разделе размещены некоторые ресурсы Интернет, касающиеся вопросам МК AVR, электроники и т.п.

1. <http://www.atmel.com> (англ.) – сайт фирмы «ATMEL».
2. <http://www.atmel.ru> (рус.) – зеркало фирмы «ATMEL» на русском языке.
3. <http://electronix.ru> (рус.) – мощный русскоязычный портал по аналоговой и цифровой электронике, программированию и близлежащим областям.
4. <http://www.avrfreaks.net> (англ.) – англоязычный портал, посвященный микроконтроллерам семейства AVR. Имеется множество свободно доступных проектов, написанных на ассемблере, Си; на форумах обсуждаются самые различные проблемы касающиеся МК AVR.
5. http://sourceforge.net/project/showfiles.php?group_id=68108 (англ.) – бесплатный компилятор с языка Си/Си++ – WinAVR, который, не смотря на свободное распространение, имеет неплохие показатели по генерации оптимального кода, хотя и уступает мощному коммерческому компилятору фирмы IAR. Рекомендуемая к использованию версия – WinAVR-20071221.
6. <http://www.ecircuitcenter.com/index.htm> (англ.) -ресурс посвящен моделированию схем при помощи языка SPICE. Имеет большую коллекцию схем, поясняющих работу операционных усилителей, фильтров, других элементов. Каждая схема сопровождается описанием на языке SPICE.