Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тульский государственный университет»

Институт высокоточных систем им. В.П. Грязева
Кафедра «Системы автоматического управления»

Утверждено на заседании кафедры
«Системы автоматического управления»
«13» января 2021 г., протокол № 8

Заведующий кафедрой
_____О.В.Горячев

СБОРНИК МЕТОДИЧЕСКИХ УКАЗАНИЙ К ПРАКТИЧЕСКИМ ЗАНЯТИЯМ

по дисциплине

«Программирование встраиваемых систем»

основной профессиональной образовательной программы высшего образования – программы специалитета

по специальности
24.05.06 Системы управления летательными аппаратами

со специализацией
Системы управления беспилотными летательными аппаратами

Форма обучения: очная

Идентификационный номер образовательной программы: 240506-01-21

Тула 2021 год

ЛИСТ СОГЛАСОВАНИЯ

Разработчик:

Ефромеев Андрей Геннадьевич, доц. каф. САУ, к.т.н. _____

(ФИО, должность, ученая степень, ученое звание) *(подпись)*

Практическое занятие № 1

Знакомство со средой разработки и отладочным комплектом для микроконтроллера

ЦЕЛЬ И ЗАДАЧИ

Изучить интегрированную среду программирования и отладки AVR Studio. Получить навыки отладки программ в симуляторе AVR.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

AVR Studio – интегрированная отладочная среда разработки приложений (IDE) для микроконтроллеров семейства AVR фирмы Atmel.

AVR Studio содержит:

- транслятор языка ассемблера (AVR Assembler)
- отладчик (Debugger)
- программное обеспечение верхнего уровня для поддержки программирования (AVR Programmer)
- программный симулятор (AVR Simulator)

Отладчик, входящий в состав AVR Studio, поддерживает все типы микроконтроллеров AVR и имеет два режима работы: режим программной симуляции и режим управления различными типами внутрисхемных эмуляторов (например, JTAG ICE).

AVR Studio поддерживает разработку программ, как на языке ассемблера, так и на языке Си. В последнем случае для компиляции программы используется свободно распространяемый компилятор AVR GCC (GNU C Compiler).

ПРАКТИЧЕСКАЯ ЧАСТЬ

Создание проекта.

Запустите AVR Studio. Появившееся окно предоставляет возможность открыть существующий или создать новый проект. Создадим новый проект, нажав кнопку **New Project**. Появится окно создания проекта (рис.1):

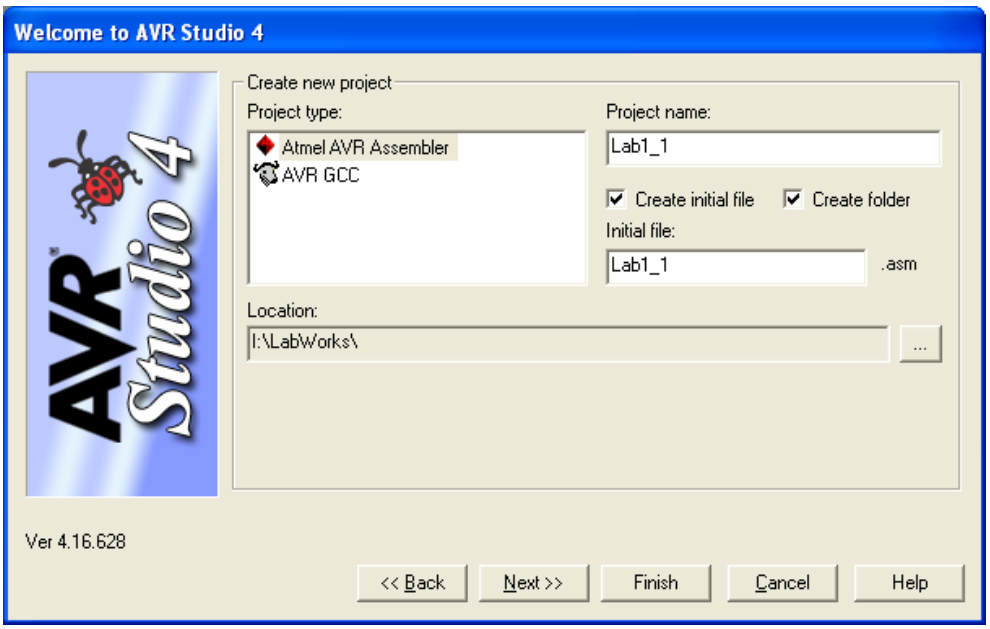

Рис.1. Создание нового проекта.

Для создание проекта на Ассемблере в списке **Project type** выбираем Atmel AVR Assembler. В поле **Project name** задаём имя проекта **L1_1**. Это же имя автоматически будет присвоено главному файлу проекта (**L1_1.asm**). При необходимости его можно изменить в поле **Initial file**. Для определения папки, в которой находится проект, щёлкните по кнопке, находящейся справа от поля **Location**. Выбираем папку C:\LabWorks.

Теперь нажимаем кнопку **Next**. Появляется окно выбора платформы отладки и целевого микроконтроллера (Рис.2). В списке платформ **Debug platform** выбираем AVR Simulator 2, а в списке целевых микроконтроллеров выбираем ATmega128.

Теперь нажимаем кнопку **Finish** и наш проект создан.

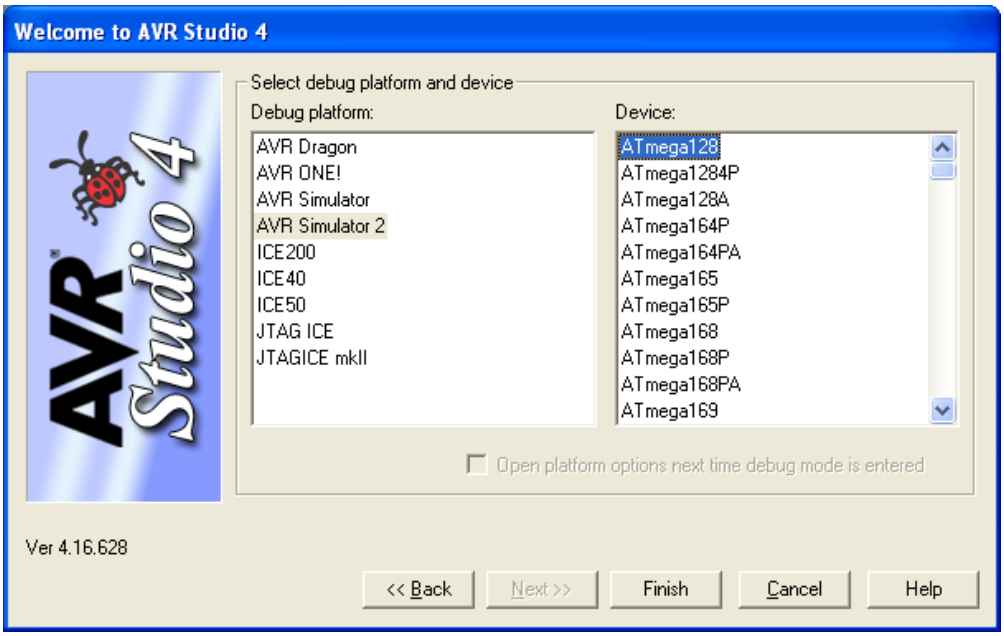

Рис.2. Определение типа микроконтроллера.

Ввод текста программы.

После создания проекта на экране появляется основное окно программы AVR Studio (Рис.3). В левой части окна можно видеть древовидную структуру проекта. Внизу окно служебных сообщений. Основную часть окна занимает область ввода кода.

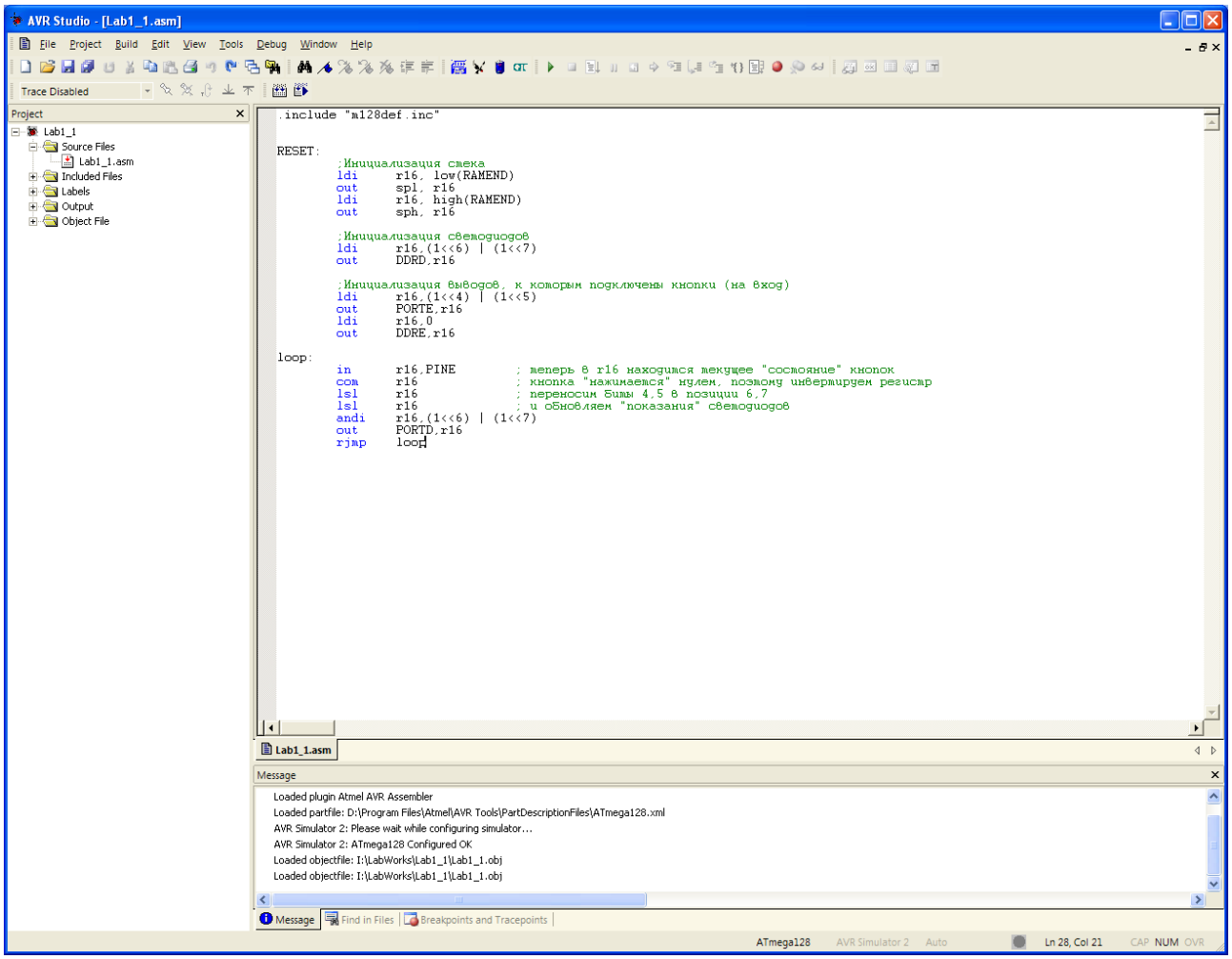

Рис.3. Основное окно программы AVR Studio.

Наберите представленную ниже программу (Листинг 1.1). Для дальнейшего удобства редактирования программы рекомендуется использовать клавишу табуляции для отделения меток, команд с операндами и комментариев.

Программа (Листинг 1.1) позволяет зажигать светодиоды VD2 или VD3 на отладочной плате AS-megaM при нажатии на клавиши SW1 или SW2 соответственно.

Листинг 1.1

```
.include "m128def.inc"
.def    temp = r16

RESET:
        ;Инициализация стека
        ldi     temp, low(RAMEND)
        out     spl, temp
        ldi     temp, high(RAMEND)
        out     sph, temp

        ;Инициализация светодиодов
        ldi     temp, (1<<6) | (1<<7)
        out     DDRD, temp

        ;Инициализация выводов, к которым подключены кнопки (на вход)
        ldi     temp, (1<<4) | (1<<5)
        out     PORTE, temp
        ldi     temp, 0
        out     DDRE, temp

loop:
        in      temp, PINE   ;теперь в r16 находится текущее "состояние" кнопок
        com     temp         ;кнопка "нажимается" нулем, поэтому инвертируем
        lsl     temp         ;переносим биты 4,5 в позиции 6,7
        lsl     temp         ;и обновляем "показания" светодиодов
        andi    temp, (1<<6) | (1<<7)
        out     PORTD, temp
        rjmp    loop
```

Ассемблирование программ.

Для ассемблирования программы можно воспользоваться меню **Project/Build**, кнопкой F7 или кнопкой **Build** панели инструментов. В окно **Build** (Рис.4) выводится информация о результатах ассемблирования:

```
Build
AVRASM: AVR macro assembler 2.1.30 (build 592 Nov  7 2008 12:38:17)
Copyright (C) 1995-2008 ATMEL Corporation

I:\LabWorks\Labl_l\Labl_l.asm(1): Including file 'D:\Program Files\Atmel\AVR Tools\AvrAssembler2\Appnotes\ml28def.inc'
I:\LabWorks\Labl_l\Labl_l.asm(32): No EEPROM data, deleting I:\LabWorks\Labl_l\Labl_l.eep

ATmega128 memory use summary [bytes]:
Segment   Begin    End      Code   Data   Used    Size   Use%
---------------------------------------------------------------
[.cseg] 0x000000 0x0000ae     36      0     36  131072   0.0%
[.dseg] 0x000100 0x000100      0      0      0    4096   0.0%
[.eseg] 0x000000 0x000000      0      0      0    4096   0.0%

Assembly complete, 0 errors. 0 warnings

Build | Message | Find in Files | Breakpoints and Tracepoints
```

Рис. 4. Результаты ассемблирования.

В этом окне можно увидеть найдены ли ошибки в коде, и если ассемблирование прошло успешно, то можно увидеть информацию о количестве использованной памяти микроконтроллера. В данном примере мы использовали 36 байт памяти команд из 131072 доступных. Память данных и энергонезависимая память (EEPROM) не использовались. Также здесь представлена информация о файлах, подключённых директивой *.include.*

Результат ассемблирования может выводится в различных форматах. По умолчанию результат сохраняется в файл *.hex в папке проекта.

Если в программе обнаружены ошибки, содержимое окна Build поможет локализовать их.

Введём ошибку в программу. В строке *.def temp = r16* заменим регистр *r16* на *r15*. В результате ассемблирования получим следующее сообщение (Рис.5):

```
Build
AVRASM: AVR macro assembler 2.1.30 (build 592 Nov  7 2008 12:38:17)
Copyright (C) 1995-2008 ATMEL Corporation

I:\LabWorks\Labl_l\Labl_l.asm(1): Including file 'D:\Program Files\Atmel\AVR Tools\AvrAssembler2\Appnotes\ml28def.inc'
I:\LabWorks\Labl_l\Labl_l.asm(9): error: Invalid register
I:\LabWorks\Labl_l\Labl_l.asm(11): error: Invalid register
I:\LabWorks\Labl_l\Labl_l.asm(15): error: Invalid register
I:\LabWorks\Labl_l\Labl_l.asm(19): error: Invalid register
I:\LabWorks\Labl_l\Labl_l.asm(21): error: Invalid register
I:\LabWorks\Labl_l\Labl_l.asm(29): error: Invalid register
I:\LabWorks\Labl_l\Labl_l.asm(32): No EEPROM data, deleting I:\LabWorks\Labl_l\Labl_l.eep

Assembly failed, 6 errors, 0 warnings
```

Рис. 5. Сообщение об ошибках.

Каждой ошибке соответствует сообщение, представленное отдельной строкой в окне **Build**. При этом указывается файл и номер строки в нём, в которой содержится ошибка, а также тип ошибки.

В данном случае обнаружена ошибка Invalid register (Неверный регистр). Перейдём к одной из строк, где обнаружена ошибка:

```
ldi     temp, 0
```

Установим курсор на инструкцию *ldi* и нажмём F1 для вызова справки по инструкции:

```
Syntax:            Operands:                        Program Counter:

(i) LDI Rd,K       16 ≤ d ≤ 31, 0 ≤ K ≤ 255         PC <- PC + 1
```

Как видно из справки, инструкция *ldi* работает только с регистрами *r16...r31*, а мы присвоили символическое имя *temp* регистру *r15*. Исправляем эту ошибку.

Отладка программы с помощью симулятора.

Вновь ассемблируем программу, не содержащую ошибок. Для выполнения пошаговой отладки открываем меню **Debug/Start Debugging** или кнопку **Start Debugging** на панели инструментов. В окне программы слева от первой команды появится жёлтая стрелка, указывающая на текущую инструкцию. Опции симулятора можно изменить с помощью меню **Debug/AVR Simulator 2 Options**.

Выполним первый шаг отладки, нажав клавишу F11 (меню **Debug/Trace into**). Жёлтая стрелка указателя отладчика установлена против первой команды *ldi temp, low(RAMEND)*.

Откройте (**View/Toolbars/Processor**) окно просмотра состояния процессора (Рис.6). В окне указанна выбранная частота микроконтроллера, состояние счётчика тактов микроконтроллера (Cycle Counter), указатель стека (Stack Pointer), счётчик команд (Program Counter), время выполнения (Stop Watch) и т.д.

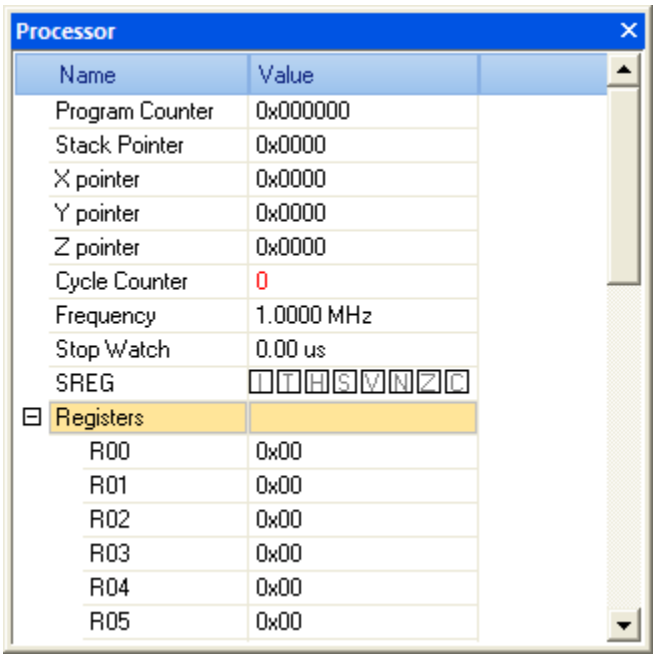

Рис.6. Окно **Processor**.

На рис.7 показано окно состояния регистров ввода-вывода **I/O View**, в котором эти регистры организованы в тематическую структуру. В данном окне можно также изменять состояния регистром в процессе отладки программы.

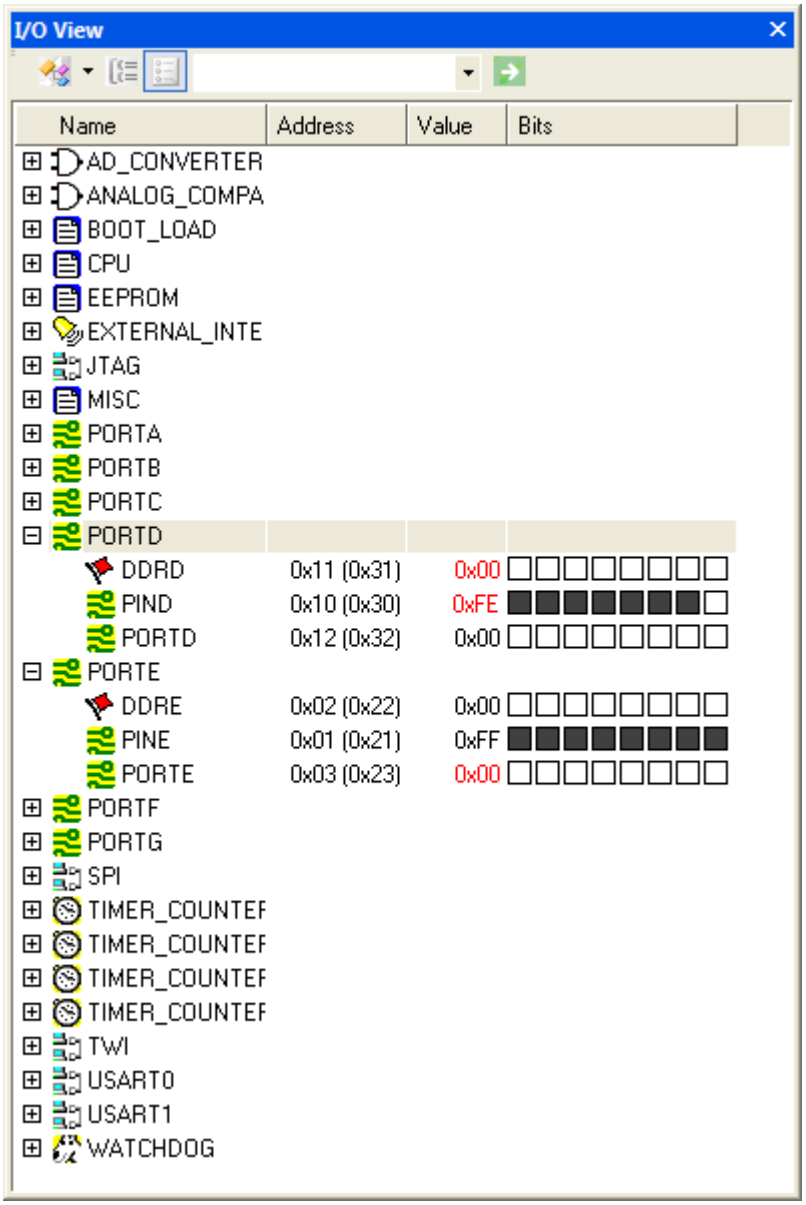

Рис. 7. Окно состояния регистров ввода вывода.

Теперь познакомимся с точками останова (Breakpoints). Они предназначены для приостановки выполнения программы в определённых местах алгоритма. Для того чтобы установить точки останова необходимо установить курсор на то место в программе, где бы вы хотели приостановить выполнение алгоритма и нажать кнопку **Toggle Breakpoint** (F9). Установим точки останова на строках *in temp*, *PINE* и *rjmp loop*. Управлять точками останова удобно с помощью окна **Breakpoint and Tracepoint** (рис. 8). В этом окне можно сортировать точки

останова, отключать или удалять их, быстро переходить к соответствующим местам в коде и т.д.

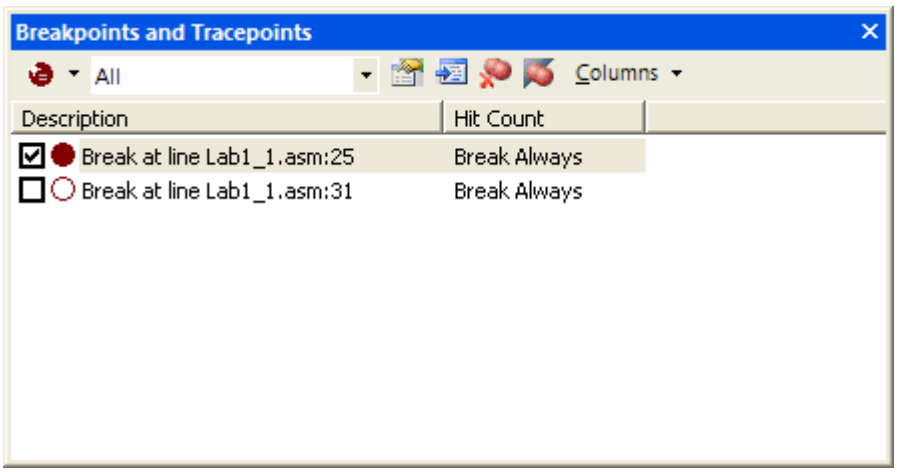

Рис. 8. Окно управления точками останова.

В окне **Breakpoint and Tracepoint**, отключите вторую точку останова (ту, которая указывает на строку *rjmp loop).* Теперь можно продолжить выполнение программы, нажав кнопку **Run** (F5). По достижении команды *in temp, PINE* отладка автоматически приостановится. Несколько раз нажмите F5, чтобы прогнать основной цикл несколько раз. Теперь в окне I/O View найдите регистр PINE и сбросьте бит 4. Теперь найдите регистр PORTD и продолжите выполнение программы, нажав кнопку F5. Таким образом, мы сымитировали нажатие кнопки, подключённой к линии PE5 порта E. При следующей остановки программы, мы увидим, что бит 7 регистра PORTD установлен, т.е. на выводе PD7 порта D появился высокий уровень напряжения. Следовательно, программа работает правильно.

Описание набора STK500.

STK500 – завершенный стартовый набор и система проектирования для AVR микроконтроллеров корпорации Atmel. Он разработан для проектировщиков, желающих быстро приступить к разработке программного кода и выполнить тестирование новой разработки. Помимо функций отладочной платы, STK500 выполняет функции параллельного программатора (для всех типов микроконтроллеров AVR), и последовательного внутрисхемного программатора (кроме Tiny11 и Tiny28, не поддерживающих функцию последовательного программирования). При программировании микроконтроллера можно производить конфигурирование микроконтроллера установкой FUSE-битов (FUnction SEt).

Функции внутрисхемного программирования и управления STK500 реализованы на двух микроконтроллерах: ATtiny2313 и ATmega8535. Кроме них на плате STK500 смонтированы следующие элементы:

- Панели для установки микроконтроллеров в корпусах DIP8, DIP20, DIP28 и DIP40;
- Стабилизированный источник питания с программно управляемым выходным напряжением;
- Преобразователи уровней сигналов (для случая, когда напряжение питания целевого микроконтроллера отличается от напряжения питания управляющих микроконтроллеров);
- Программно управляемый источник опорного напряжения для внутреннего АЦП микроконтроллера;
- Посадочное место для микросхемы DataFlash AT45D021 (внешняя последовательная Flash-память);
- Двухканальный формирователь уровней сигналов интерфейса RS232 (один канал используется для связи STK500 с управляющим персональным компьютером, другой может быть использован в разрабатываемом приложении);
- Восемь кнопок и восемь светодиодов для организации ввода и отображения выходной информации;
- Разъемы расширения для подключения внешних устройств.

Внешний вид платы STK500 представлен на рис.9. Укрупненная блок схема аппаратного обеспечения стартового набора STK500 приведена на рис.10.

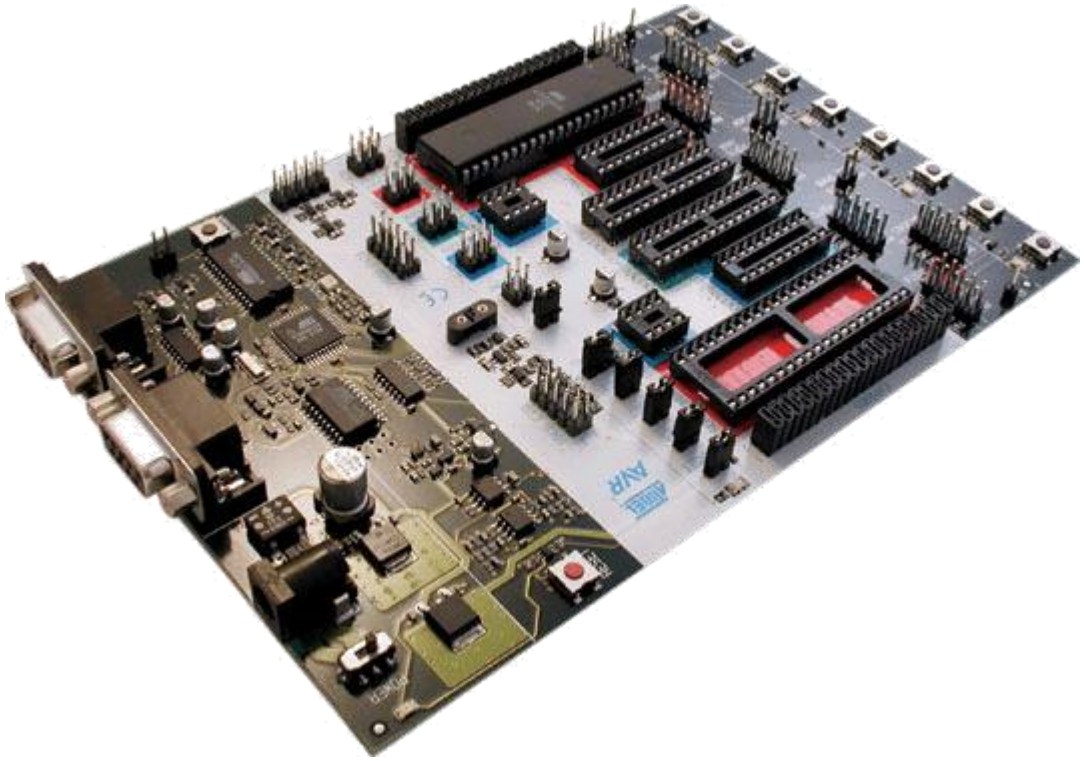

Рис.9. Внешний вид STK500.

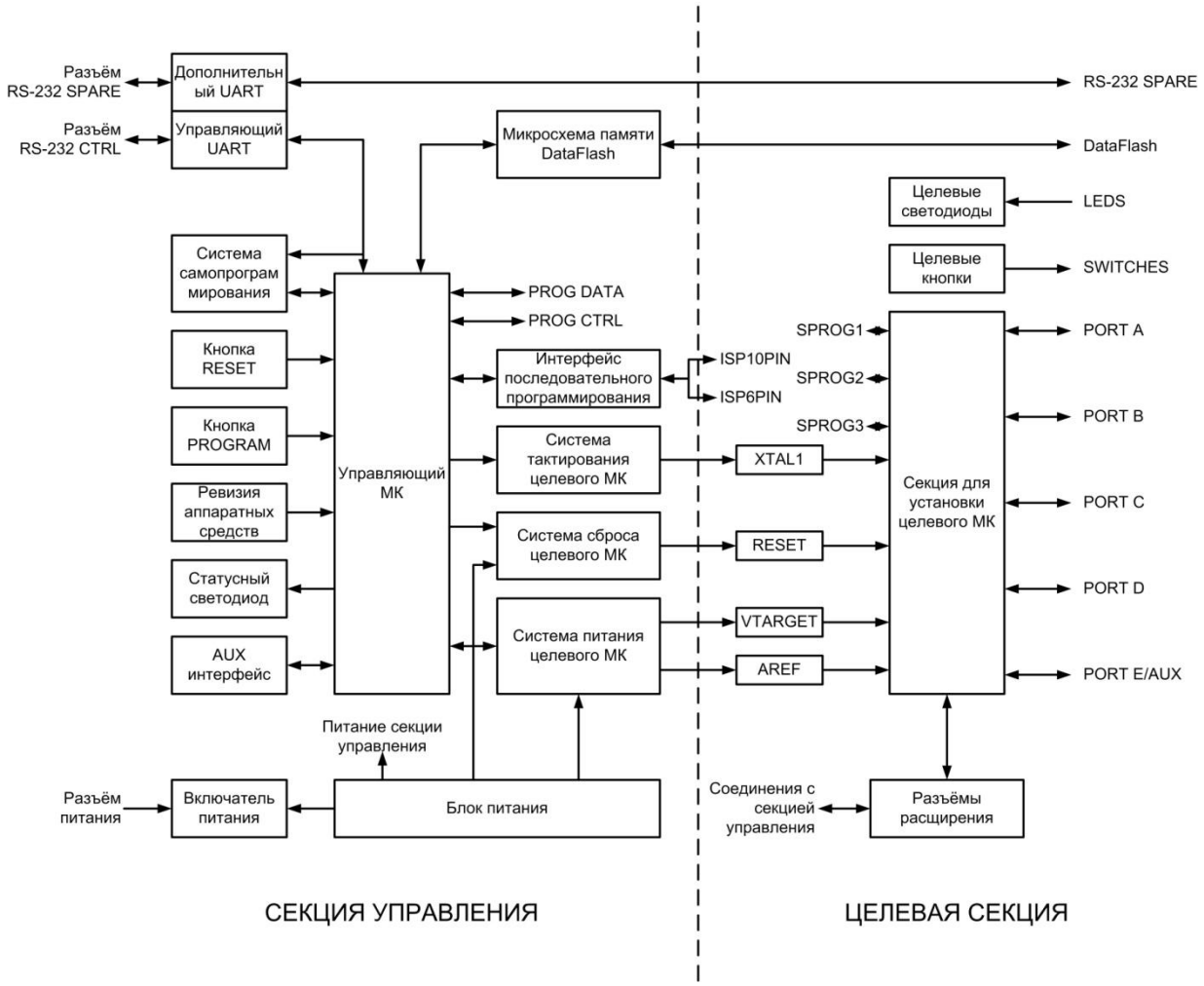

Рис.10. Блок-схема STK500.

Для тактирования отлаживаемых микроконтроллеров на плате STK500 предусмотрены два источника тактовых сигналов. Один из них представляет собой генератор, построенный на инвертирующих вентилях с кварцевой стабилизацией частоты. Пользователь имеет возможность задавать частоту этого генератора, устанавливая кварцевый резонатор с необходимой частотой в специальную панель (помечена CRYSTAL).

Второй источник тактовых сигналов представляет собой выход установленного на плате STK500 управляющего микроконтроллера ATmega8535. Частота этого тактового сигнала может быть задана программно. Доступны значения 3.69, 1.84, 1.23 МГц, а также 32.7 кГц и полная остановка.

Переключатель OSCSEL служит для выбора одного из источников тактового сигнала для микроконтроллера.

Описание основных элементов платы STK500.

Схематическое расположение основных элементов платы STK500 представлено на рис.11.

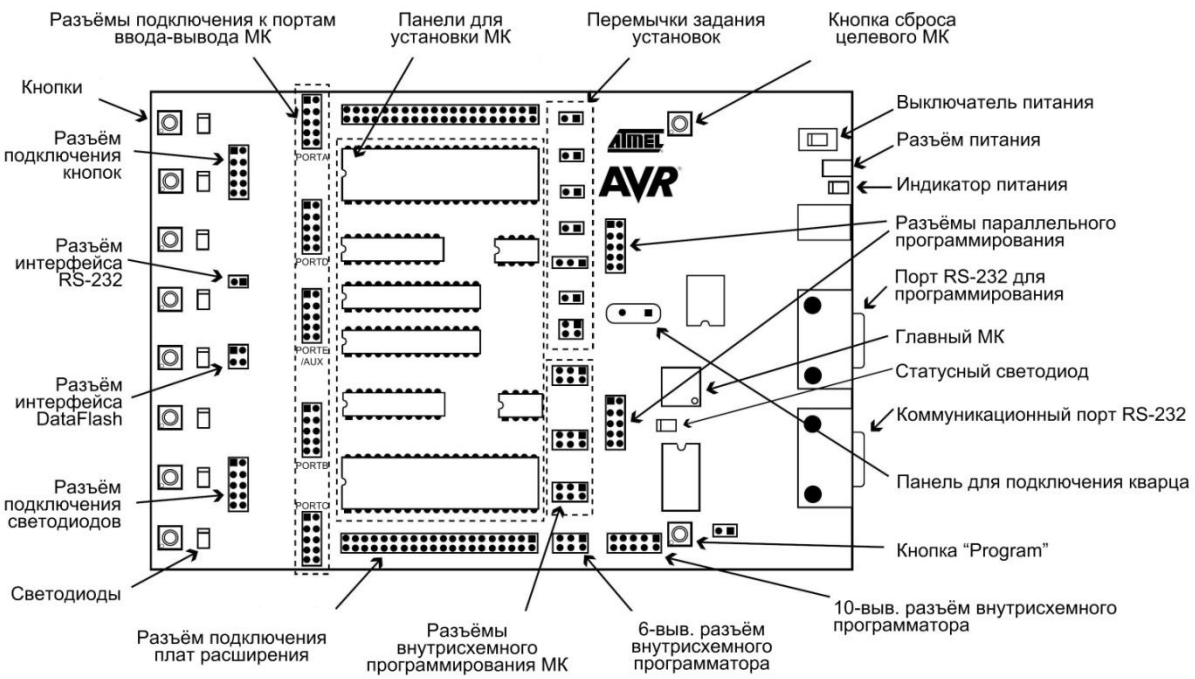

Рис.11. Расположение компонентов на плате STK500.

Светодиоды общего назначения.

В набор STK500 входят 8 желтых светодиодов и 8 кнопок без фиксации. Светодиоды и кнопки электрически отделены от остальной части платы за счет подключения к собственным разъемам. Таким образом, они могут быть подключены к портам ввода-вывода AVR-микроконтроллеров при помощи 10-проводного шлейфа (рис.12).

Шлейфы должны быть подключены непосредственно между разъемами портов и разъемами светодиодов или кнопок. Шлейф не должен скручиваться. Красный проводник шлейфа указывает на вывод 1. Непосредственно перед включением убедитесь, что шлейф подключен к первому выводу каждого разъема. На рисунке 4 показано как реализовано управление светодиодом. Данное решение позволяет получить одинаковую интенсивность свечения светодиода при нахождении напряжения питания микроконтроллера в диапазоне 1,8В…6,0В.

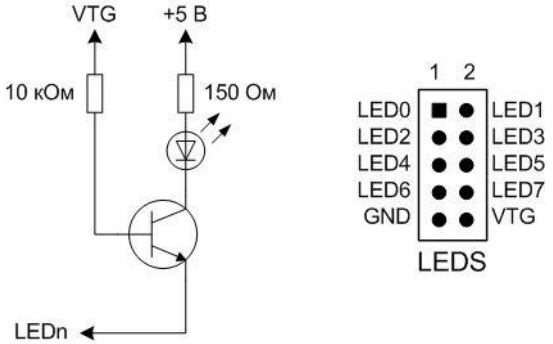

Рис.12. Схема включения светодиода и подключения светодиодов к разъему

Порты AVR-микроконтроллеров могут управлять непосредственно светодиодной нагрузкой, как втекающим током, так и вытекающим. Однако в STK500 используются транзистор и два резистора для поддержания постоянной яркости свечения светодиодов при любом значении напряжения питания микроконтроллера (VTG), а также для выключении светодиодов, когда VTG отсутствует.

Кнопки общего назначения.

Кнопки подключены к разъему по схеме изображенной на рисунке 13. При нажатии на кнопку на выводе SWn будет низкий уровень напряжения, а при отпускании – высокий (VTG). Рабочий диапазон напряжения VTG =1,8…6,0В.

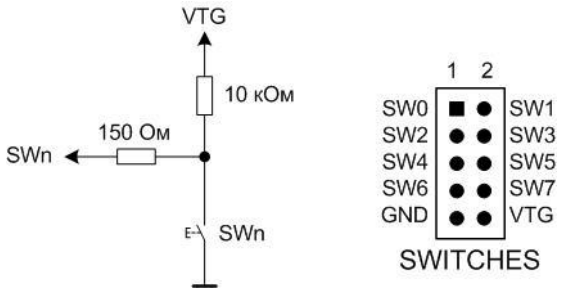

Рис.13. Схема включения кнопок и подключения к разъему.

На линиях портов ввода-вывода AVR-микроконтроллеров имеется возможность активизации встроенных подтягивающих резисторов к плюсу питания. Это свойство можно использовать в целях исключения внешнего подтягивающего резистора. В STK500 добавлены внешние подтягивающие резисторы 10 кОм для формирования лог. «1» на выводах SWn при отжатом состоянии кнопок. Резистор 150 Ом выполняет функцию защитного токоограничения, например, в случае ошибочной настройки линий ввода-вывода, связанных с кнопками, на вывод.

Любой порт ввода-вывода AVR-микроконтроллера может быть подключен к светодиодам и кнопкам, используя 10-проводной шнур, входящий в состав набора. На разъемы в дополнение к линиям портов выводятся напряжение питания целевого микроконтроллера VTG (VCC) и общий провод GND.

Разъёмы, для подключения к портам целевого микроконтроллера.

На плате STK500 имеются несколько 10-контактных разъёмов для подключения к соответствующим портам ввода-вывода целевого микроконтроллера. Вид разъёмов показан на рисунке 14:

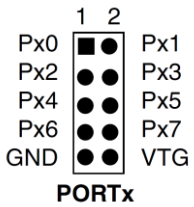

Рис.14. Разъём подключения к порту ввода-вывода.

Также на плате есть специальный разъём PORTE/AUX, который имеет ряд специальных выводов в дополнение к выводам порта E.

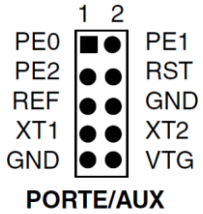

Рис.15. Разъём PORTE/AUX.

Специальные выводы разъёма PORTE/AUX:

- RST – линия сброса (RESET) целевого микроконтроллера
- REF – линия опорного источника напряжения АЦП целевого МК
- XT1 – линия главного тактирующего сигнала XTAL1
- XT2 – линия XTAL2, используемая совместно с XTAL1, для подключения внешнего кварцевого резонатора

Пользовательский RS-232 интерфейс.

STK500 включает в себя два порта RS-232. Один из них используется для связи STK500 с AVRStudio, а другой может быть использован для связи между целевым МК и последовательным портом компьютера. Для этого нужно, чтобы выводы TxD и RxD UART (УАПП) микроконтроллера были подключены к двухконтактному разъёму “RS232 SPARE” на плате STK500 (рис. 16, 17).

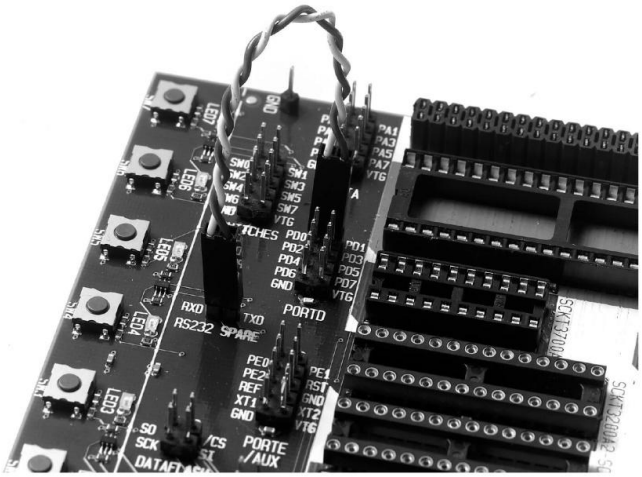

Рис. 16. Подключение RS232 к выводам UART микроконтроллера.

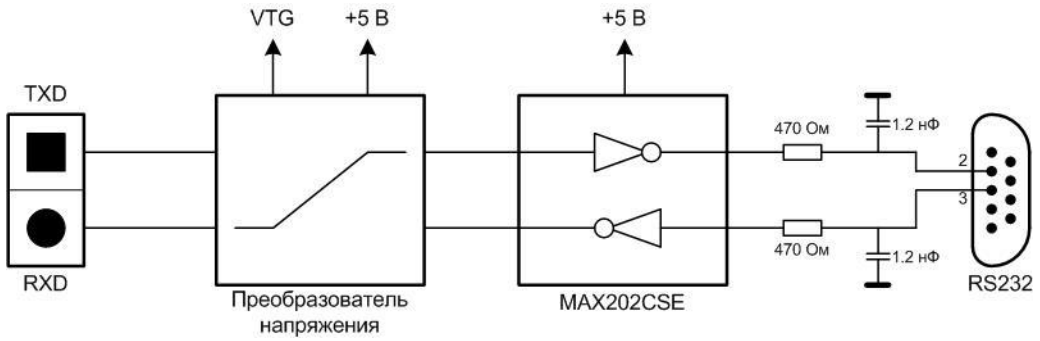

Рис. 17. Схема подключения RS232 к выводам UART МК.

Настроечные перемычки.

На плате STK500 расположен ряд перемычек (рис.18), позволяющих подключать к целевому МК различные устройства, а также управлять их работой.

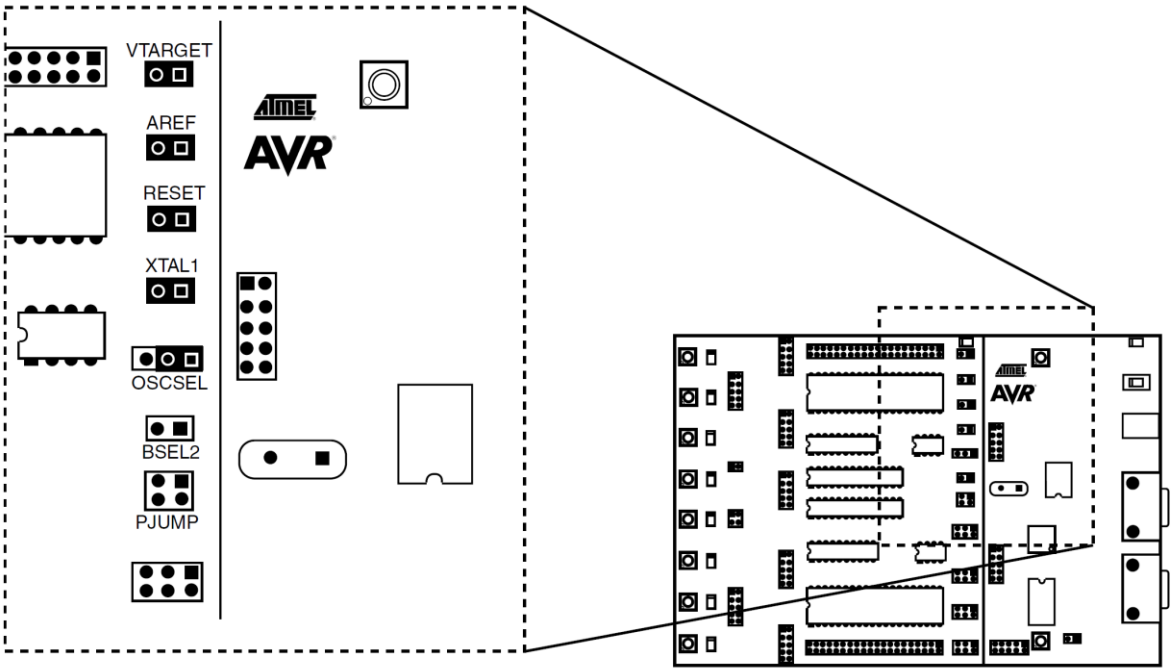

Рис. 18. Расположение управляющих перемычек и их значения по умолчанию.

Назначение каждой перемычки кратко описано в таблице 1.

Таблица 1.

VTARGET	Подключает к целевому МК источник питания
AREF	Подключает к целевому МК источник опорного напряжения АЦП
RESET	Подключает к целевому МК систему сброса
XTAL1	Подключает к целевому МК систему тактирования
OSCSEL	Управляет выбором источника тактового сигнала
BSEL2	Замыкается при высоковольтном программировании МК ATmega8, ATmega16, ATmega161, ATmega163, ATmega128 и ATmega323

PJUMP	Замыкается при высоковольтном программировании МК AT90S2333, AT90S4433 и ATmega8

ПРАКТИЧЕСКАЯ ЧАСТЬ

Начало работы с STK500.

В начале работы с STK500 необходимо подключить к персональному компьютеру и источнику постоянного напряжения 10-15В, как показано на рисунке 19.

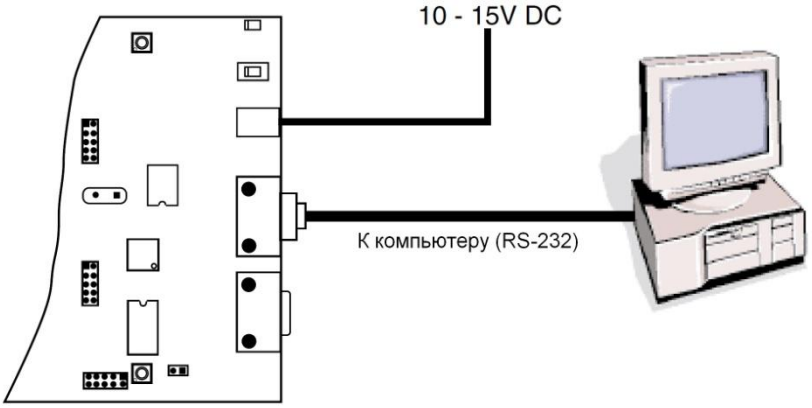

Рис.1191. Подключение STK500 к ПК и источнику питания.

Управление STK500 производится через COM-порт (RS-232) персонального компьютера. Управляющая программа является составной частью AVR Studio и запускается из меню **Tools→Program AVR→Connect…**. Кроме STK500 эта управляющая программа поддерживает такие аппаратные средства отладки, как внутрисхемный программатор ATAVRISP и внутрисхемный эмулятор ATJTAGICE. Подробнее возможности программы будут рассмотрены ниже. Здесь мы ограничим описание только функциями, уникальными для STK500. На рисунке 20 представлено окно программы, управляющей настройками STK500.

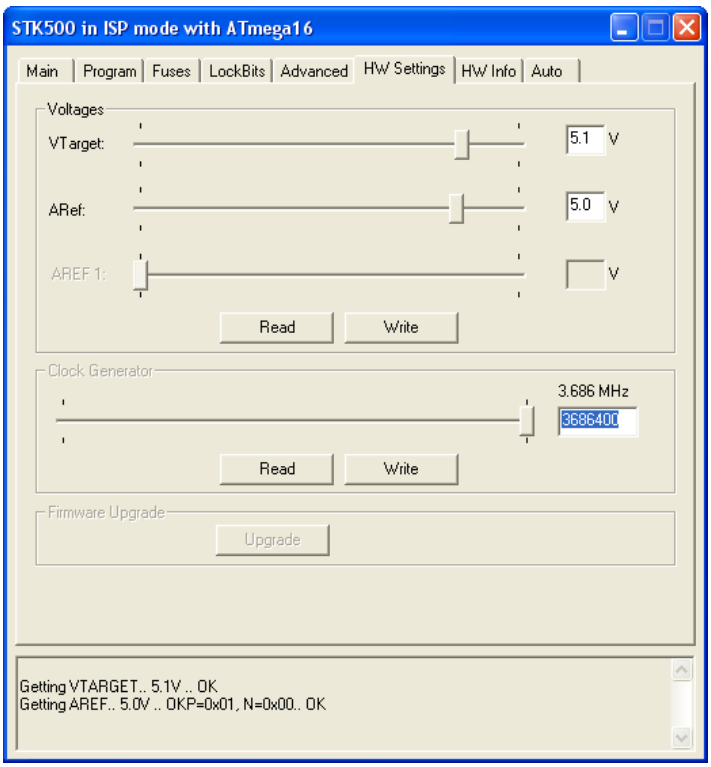

Рис.20. Окно управления настройками STK500.

Закладка **HW Setting** служит для установки параметров STK500:

- Напряжения питания целевого микроконтроллера (**VTarget**);
- Опорного напряжения АЦП микроконтроллера (**ARef**);
- Частоты управляемого источника тактового сигнала (**Clock Generator**).

STK500 поддерживает различные режимы программирования целевых микроконтроллеров: режим последовательного внутрисхемного программирования, и два режима программирования с использованием повышенного программирующего напряжения – параллельный и последовательный. Выбор режима программирования осуществляется в окне управления параметрами STK500 в закладке **Main**.

Для использования STK500 в качестве внутрисхемного программатора микроконтроллеров, установленных в целевом устройстве, необходимо соединить целевое устройство с разъемом ISP10PIN или ISP6PIN платы STK500, используя плоский десятижильный или шестижильный шлейф соответственно.

Программная поддержка новых микроконтроллеров AVR в STK500 производится обновлением прошивки (firmware) памяти программ управляющих микроконтроллеров (Revision Upgrade). Обновленные версии firmware включаются в состав AVR Studio. Управляющая программа в момент запуска проверяет связь COM-порта персонального компьютера с STK500, и в случае его присутствия запрашивает версию firmware. Если в AVR Studio содержится более

новая версия прошивки, управляющая программа предлагает обновить firmware путем перепрограммирования управляющих микроконтроллеров. Перед входом в режим перепрограммирования STK500 необходимо извлечь целевые микроконтроллеры из DIP-панелей, если они туда установлены. Для входа в режим перепрограммирования необходимо включить питание STK500 с нажатой кнопкой Program.

В комплекте с STK500 поставляются микроконтроллеры AT90S8515 и ATmega16, а также диск AVR Software and Technical Library, включающий в состав AVR Studio. Также в состав набора входит комплект кабелей.

Внутрисхемное последовательное программирование.

Внутрисхемное программирование (In-System Programming, ISP), так же называемое низковольтным последовательным программированием, позволяет запрограммировать Flash- и EEPROM-память микроконтроллера, а также задать значения установочных битов (FUSE-биты) и битов защиты (Lock-биты), не извлекая микроконтроллер из целевого устройства и не используя дополнительного высоковольтного напряжения. При таком программировании микроконтроллер находиться в рабочем состоянии, значения напряжения питания и тактовой частоты, лежат в пределах, установленных для МК данного типа.

Для внутрисхемного программирования контроллеров семейства AVR используется последовательный интерфейс SPI. Для того чтобы МК вошёл в режим программирования необходимо подключить к земле вывод RESET. Последовательный интерфейс состоит из трёх выводов: SCK, MOSI (вход) и MISO (выход). Также на вывод XTAL1 МК должен подаваться тактирующий сигнал (если не используется внутренний источник тактового сигнала) и подключено питающее напряжение (Рис.21).

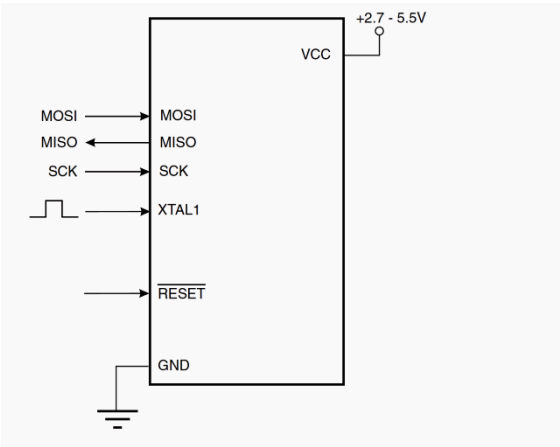

Рис.21. Подключение к МК линий последовательного внутрисхемного программатора.

Для того чтобы запрограммировать память целевого микроконтроллера с помощью встроенного в STK500 внутрисхемного программатора, необходимо подключить его к соответствующим выводам МК. Для этого нужно соединить разъём ISP6PIN программатора с одним из разъёмов SPROG1, SPROG2 или SPROG3. Для МК ATmega16 это будет разъём SPROG3 красного цвета (рис.22).

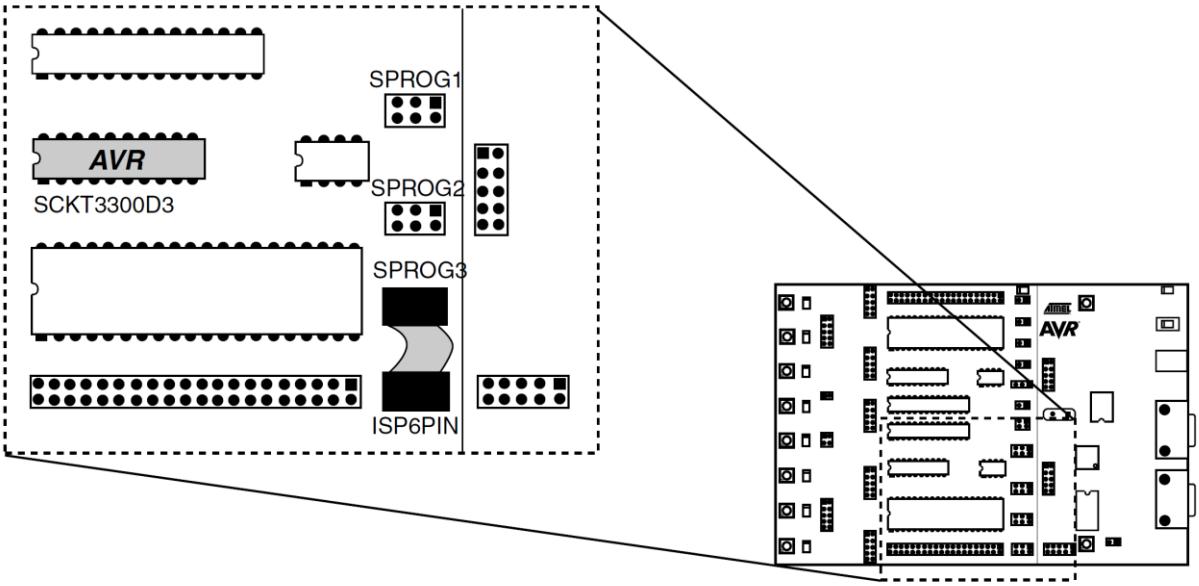

Рис.22. Подключение разъема ISP6PIN для внутрисхемного программирования целевого микроконтроллера.

Теперь нужно подать питание на STK500 (если оно было отключено) и в AVRStudio открыть программатор **Tools→Program AVR→Auto Connect**. После этого на экране должно появится окно, показанное на рисунке 23.

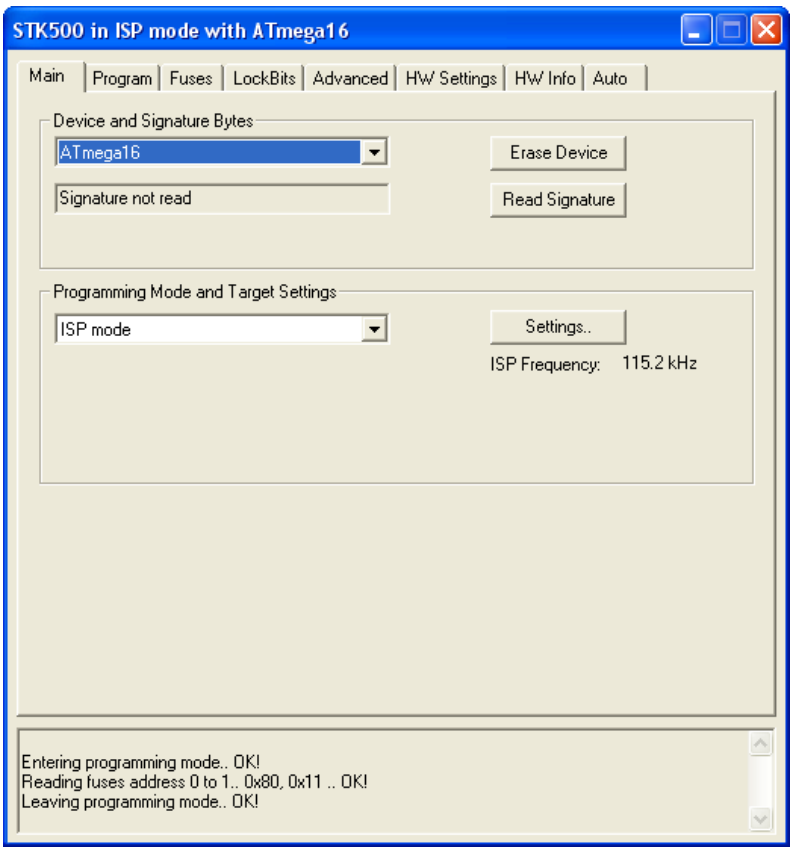

Рис.23. Вкладка **Main** утилиты AVR Programmer.

На вкладке Main можно выбрать целевой микроконтроллер (по умолчанию загружается тот, что указан в проекте) и режим программирования. Для внутрисхемного программирования выбираем **ISP Mode** в поле **Programming Mode and Target Settings**. Также на вкладке **Main** находиться кнопка **Read Signature**, которая позволяет считать сигнатурный код микроконтроллера. Сигнатурный код – это три однобайтовых числа, в которых зашифрована фирма производитель и модель микроконтроллера. Так, для ATmega16, сигнатурный код – 1E 94 03. Обратите внимание на то, что если сигнатурный код не соответствует выбранному устройству, то программа предупредит об этом.

Управление непосредственно программирование МК осуществляется на вкладке **Program** (Рис.24).

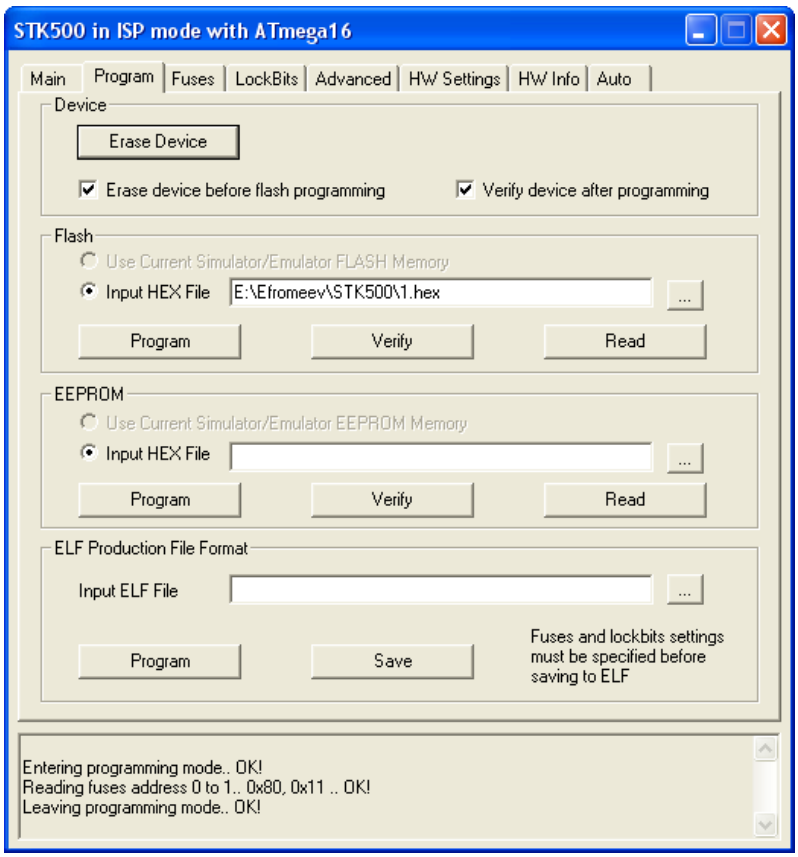

Рис.24. Вкладка **Program** утилиты AVR Programmer.

Кнопка **Erase Device** позволяет стереть память МК, это необходимо всегда делать при программировании. Что бы стирание производилось автоматически при программировании нужно поставить флажок **Erase device before flash programming**. В группах **Flash** и **EEPROM** находятся органы управления программированием Flash- и EEPROM-памяти соответственно. В поле **Input HEX File** помещается адрес HEX-файла, который будет загружен в соответствующую память МК. Кнопка **Program** позволяет загрузить новые данные в память МК из этого файла. При нажатии на кнопку **Verify** содержимое памяти МК сравнивается с содержимым HEX-файла. Кнопка **Read** позволяет сохранить содержимое памяти МК в HEX-файл. Результаты всех операций отображаются в нижней части окна. При удачном программировании там должны появиться следующие строки:

Reading FLASH input file.. OK (Чтение входящего файла)
Entering programming mode.. OK! (Вход в режим программирования)
Erasing device.. OK! (Стирание устройства)
Programming FLASH .. OK! (Программирование FLASH-памяти)
Reading FLASH .. OK! (Чтение FLASH-памяти)
FLASH contents is equal to file.. OK (Данные во FLASH эквивалентны файлу)

Leaving programming mode.. OK! (Выход из режима программирования)

Установка настроечных (FUSE) битов производится на вкладке **Fuses** окна утилиты AVRProgrammer (Рис.25).

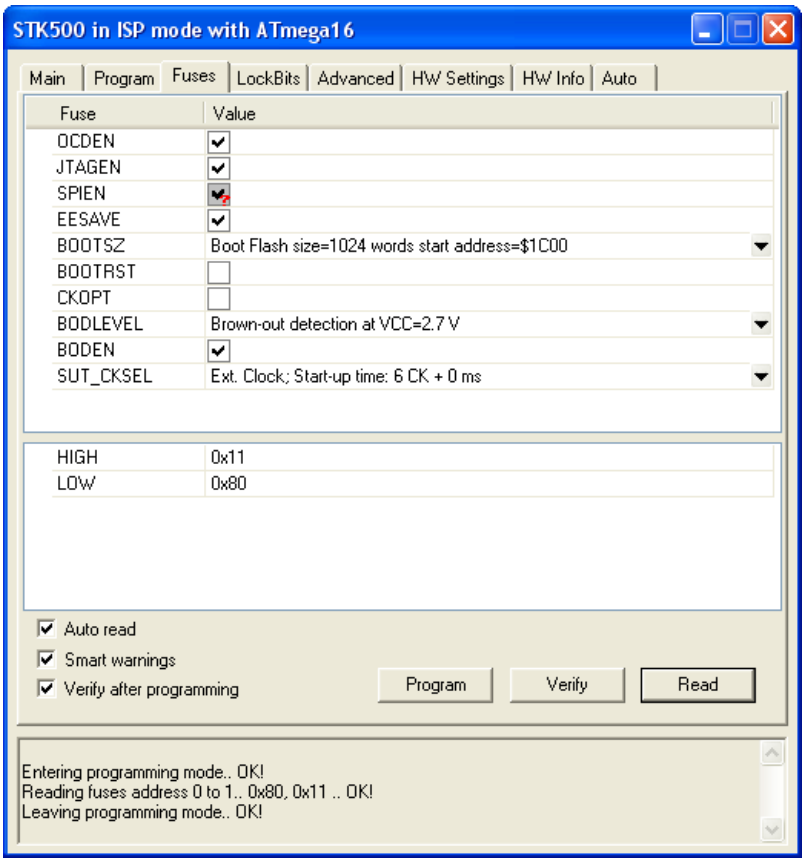

Рис.25. Вкладка **Fuses** утилиты AVR Programmer.

Правильная настройка FUSE-битов играет крайне важное значение для работы МК, поэтому рассмотрим каждый из них:

- OCDEN – включение встроенного блока отладки, который необходим для внутрисхемной отладки МК с помощью специального прибора – отладчика;
- JTAGEN – включение JTAG-интерфейса, который может быть использован для подключения отладчика и для внутрисхемного программирования МК;
- SPIEN – разрешение последовательной загрузки программы и данных, которое необходимо для внутрисхемного программирования по SPI (обратите внимание, что этот бит помечен красным знаком вопроса, т.к. программа AVRProgrammer подключена к МК как раз через внутрисхемный программатор, и поэтому, если бит SPIEN будет сброшен, то после выхода из режима программирования, подключится к МК с помощью внутрисхемного программатора будет невозможно);

- EESAVE – запрет стирания EEPROM-памяти командой стирания кристалла;
- BOOTSZ – это группа FUSE-битов (BOOTSZ0 и BOOTSZ1), которая задаёт размер загрузочного сектора;
- BOOTRST – выбор вектора сброса. Если конфигурационный бит BOOTRST установлен, то при сбросе микроконтроллер выполняет переход на адрес сброса в загрузочном секторе, а не на адрес $0000, как обычно;
- CKOPT – настройка генератора. Если CKOPT запрограммирован, то амплитуда колебаний выходного сигнала генератора будет ограничена уровнями питания. Данный режим рекомендуется использовать при высоком уровне окружающих шумов или при использовании выхода XTAL2 в качестве источника синхронизации внешней схемы. Если CKOPT сброшен, то амплитуда выходных колебаний генератора снижается. Использование данного режима позволяет существенно снизить потребляемую мощность, но при этом ограничен частотный диапазон и нельзя XTAL2 использовать для внешней синхронизации;
- BODLEVEL – порог срабатывания супервизора питания (2,7 или 4 В);
- BODEN – разрешение супервизора питания (Brown-out detection). Супервизор питания выключает микроконтроллер, если питающее напряжение упало ниже установленного уровня;
- SUT_CKSEL – группа настроечных бит (CKSEL0, CKSEL1, CKSEL2 и CKSEL3), которые задают источник тактового сигнала и его параметры.

Запись новых значений FUSE-битов в микроконтроллер осуществляется после нажатия кнопки **Program**.

На вкладке LockBits окна утилиты AVRProgrammer (Рис.26) производится установка битов защиты памяти, возможные варианты их конфигурации приведены в таблицах 2-4.

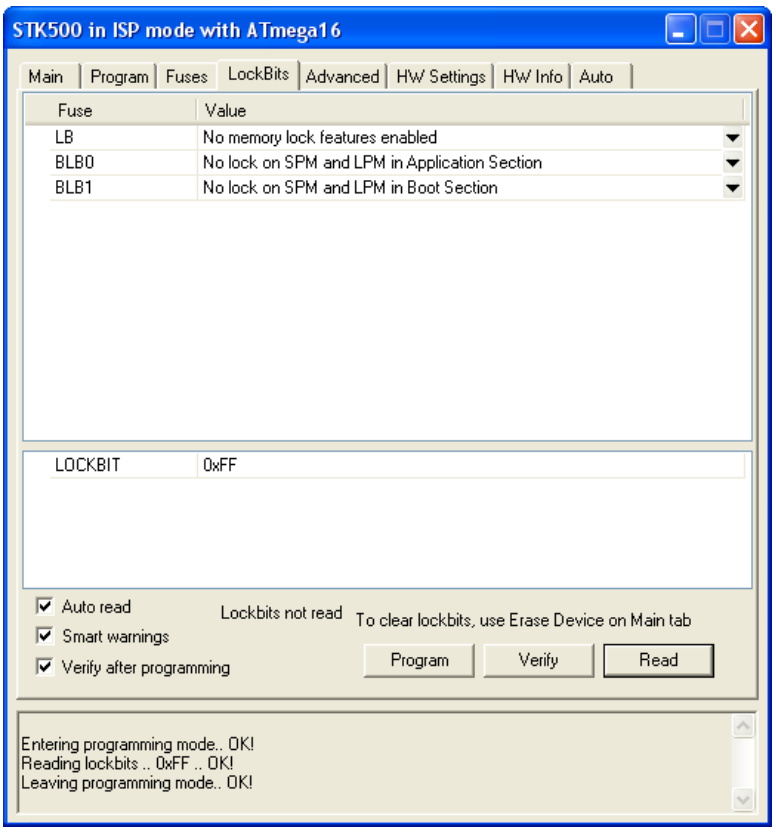

Рис.26. Вкладка **LockBits** утилиты AVR Programmer.

Таблица 2

Режим LB	Описание защиты
No memory lock features enabled	Нет защиты памяти
Further programming disabled	Дальнейшее программирование флэш-памяти и ЭСППЗУ отключено при параллельном и последовательном (SPI/JTAG) программировании. Конфигурационные биты защищены при любом способе программирования
Further programming and verification disabled	Дальнейшее программирование и проверка флэш-памяти и ЭССПЗУ отключена как при параллельном, так и при последовательном программировании через SPI/JTAG. Конфигурационные биты защищены при любом способе программирования

Таблица 3

Режим BLB0	Описание защиты
No lock on SPM and LPM in Application Section	Нет ограничений действия инструкций SPM или (E)LPM при адресации сектора прикладной программы

SPM prohibited in Application Section	SPM не записывает данные в сектор прикладной программы
LPM and SPM prohibited in Application Section	SPM не записывает данные в сектор прикладной программы, а выполнение инструкции (E)LPM в загрузочном секторе не позволяет считать данные из сектора прикладной программы. Если векторы прерываний размещены в загрузочном секторе, то при выполнении команд в секторе прикладной программы прерывания отключаются
LPM prohibited in Application Section	Выполнение (E)LPM в загрузочном секторе не позволяет считать данные из сектора прикладной программы. Если векторы прерываний размещены в загрузочном секторе, то при выполнении команд в секторе прикладной программы прерывания отключаются

Таблица 4

Режим BLB1	Описание защиты
No lock on SPM and LPM in Boot Section	Нет ограничений действия инструкций SPM или (E)LPM при адресации загрузочного сектора
SPM prohibited in Boot Section	SPM не записывает данные в загрузочный сектор
LPM and SPM prohibited in Boot Section	SPM не записывает данные в загрузочный сектор, а выполнение инструкции (E)LPM в секторе прикладной программы не позволяет считать данные из загрузочного сектора. Если векторы прерываний размещены в секторе прикладной программы, то при выполнении команд в загрузочном секторе прерывания отключаются
LPM prohibited in Boot Section	Выполнение (E)LPM в секторе прикладной программы не позволяет считать данные из загрузочного сектора. Если векторы прерываний размещены в секторе прикладной программы, то при выполнении команд в загрузочном секторе прерывания отключаются

Высоковольтное параллельное программирование.

В тех случаях, когда внутрисхемное программирование невозможно (например, если не установлен FUSE-бит SPIEN в целевом МК или по схеме не предусмотрено подключение внутрисхемного программатора), для программирования МК применяют высоковольтное программирование. При этом необходимо извлечь МК из целевой схемы и установить его в программатор, например STK500.

Существует два вида высоковольтного программирование: последовательное и параллельно. Последовательное программирование применяется только на микроконтроллерах, имеющих только 8 выводов, т.к. параллельное программирование этих МК просто невозможно.

Выводы МК, которые задействованы при параллельном программировании показаны на рисунке 27.

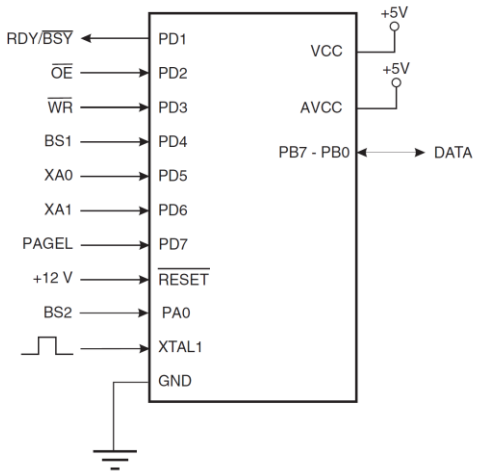

Рис.27. Подключение к МК выводов параллельного программатора.

Для того чтобы запрограммировать целевой МК методом параллельного высоковольтного программирования с помощью стартового набора STK500, необходимо установить этот МК в соответствующий разъём на плате STK500 и подключить порт B к разъёму PROG DATA, а порт D – к разъёму PROG CTRL (Рис.28).

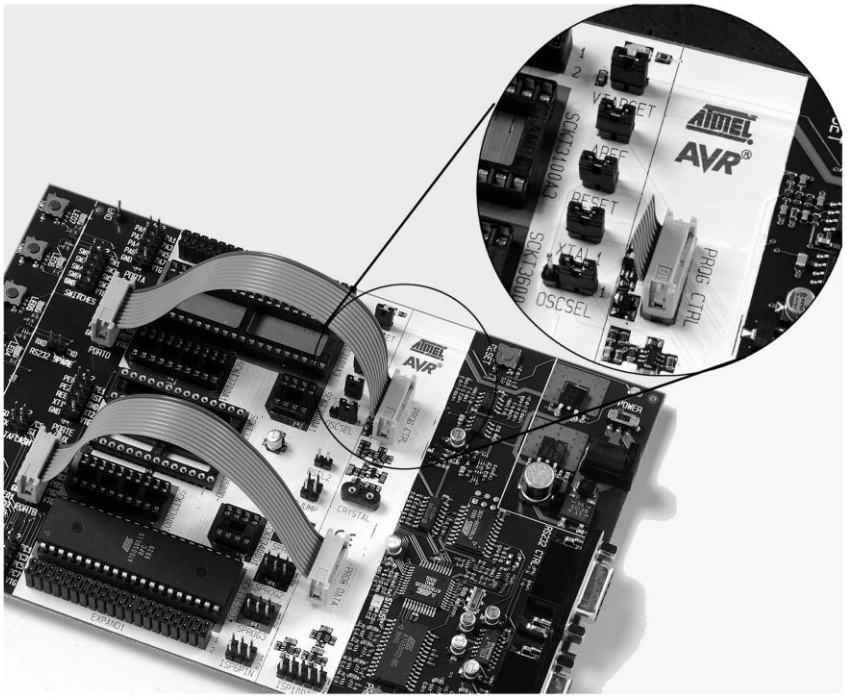

Рис.28. Подключение при параллельном высоковольтном программировании.

Помимо этого надо установить перемычки на плате STK500 так, как это показано на рисунке 29. При программировании микроконтроллера ATmega16 перемычка BSEL2 должна быть установлена, а PJUMP – нет.

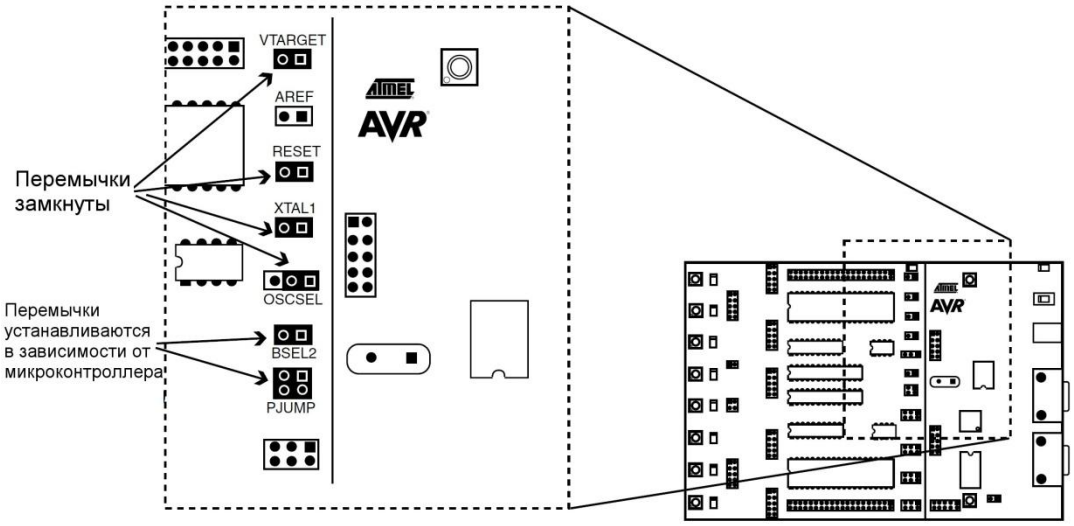

Рис.29. Настройки перемычек при высоковольтном параллельном программировании.

После того как перемычки установлены можно подать питание на STK500 и открыть в AVR Studio утилиту AVRProgrammer. В поле **Programming Mode and Target Settings** необходимо выбрать **PP/HVSP mode** (Parallel Programming/High-Voltage Serial Programming mode). Дальнейшие действия аналогичны действиям при внутрисхемном программировании рассмотренным ранее.

ЗАДАНИЕ

Изучить устройство стартового набора STK500.

Научиться производить начальную настройку STK500 и целевого микроконтроллера.

Научиться загружать программу в микроконтроллер различными способами.

ПОРЯДОК ВЫПОЛНЕНИЯ

1. Начало работы и загрузка программы в МК.

1) Запустите AVR Studio и откройте проект **L2_1**.
2) Подключите STK500 к персональному компьютеру.
3) Подключите шлейф для последовательного внутрисхемного программирования к соответствующим разъёмам на плате STK500.
4) Подключите к порту B (PORTB) целевого МК светодиоды (LEDS).
5) Включите питание STK500.
6) Скомпилируйте открытый проект.
7) Откройте программу AVRProgrammer и настройте на последовательное внутрисхемное программирование.
8) Загрузите в целевой микроконтроллер ранее скомпилированную программу.
9) Если всё сделано правильно, вы должны увидеть «бегающий» справа налево светодиод.

2. Настройка источника тактового сигнала.

1) В программе AVRProgrammer откройте вкладку **Fuses**.
2) В поле SUT_CKSEL выберите в качестве источника тактового сигнала внутренний RC-генератор, и настройте его на частоту 8 МГц (Int. RC Osc. 8 MHz). Нажмите кнопку **Program**, чтобы загрузить новое значение FUSE-битов в МК.
3) Теперь настройте внутренний RC-генератор на частоту 1 МГц (Int. RC Osc. 1 MHz).
4) Обратите внимание на то, что светодиоды стали переключаться медленнее, т.к. тактовая частота микроконтроллера уменьшилась.
5) Далее установите в качестве источника тактового сигнала внешний тактовый генератор, установленный на плате STK500. Для этого в поле SUT_CKSEL выберите строку: Ext. Clock.
6) Изменяя настройки тактового генератора, встроенного в STK500, на вкладке **HW Settings** программы AVRProgrammer, наблюдайте за тем, как меняется скорость мигания светодиодов.

3. Настройка супервизора питания.

1) Включите супервизор питания (Brown-out detector).
2) Установите порог срабатывания супервизора питания равным 4 В (Brown-out detection at VCC=4.0 V).
3) Теперь на вкладке **HW Settings** установите напряжение питания целевого МК (**VTarget**) равным 3,5 В.
4) После того, как новые значения вступят в силу, МК перестанет загружаться, т.к. напряжение питания ниже установленного уровня.
5) Установите порог срабатывания супервизора питания равным 2,7 В (Brown-out detection at VCC=2.7 V). Как видите, микроконтроллер снова заработал.
6) Верните напряжение питания к исходному значению 5 В.

4. Высоковольтное программирование.

1) Сбросьте FUSE-бит SPIEN, чтобы запретить последовательное внутрисхемное программирование. Закройте окно программатора. Попытайтесь снова войти в режим программирования и убедитесь, что это невозможно.
2) Выключите питание платы STK500.
3) Отсоедините шлейф для последовательного внутрисхемного программирования, соединяющий разъёмы ISP6PIN и SPROG3 на плате STK500.
4) Соедините разъем PORTB с разъёмом PROG DATA, а разъем PORTD – с разъёмом PROG CTRL.
5) Установите необходимые перемычки.
6) Включите питание на STK500.
7) Откройте программу AVRProgrammer и выберите режим параллельного высоковольтного программирования (HV/SPHV mode).
8) Считайте сигнатурный код целевого устройства, чтобы убедиться, что связь с МК установлена.
9) Теперь снова установите FUSE-бит SPIEN.
10) Закройте программу AVRProgrammer и выключите питание платя STK500.
11) Верните STK500 в исходное состояние, которое было при последовательном программировании.
12) Подайте питание на STK500, откройте окно программатора и выберите режим последовательного внутрисхемного программирования (ISP mode).

13) Убедитесь, что связь с микроконтроллером вновь установлена и внутрисхемное программирование разрешено.

ОФОРМЛЕНИЕ ОТЧЁТА

Отчёт должен содержать:

- краткие теоретические сведения
- описание проделанных действий и их результаты
- вывод по проделанной работе

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назначение и основные возможности STK500.
2. Различные способы программирования памяти микроконтроллера и их особенности.
3. FUSE-биты и их назначение.

Практическое занятие №2.

Отладка программного обеспечения с помощью внутрисхемного отладчика

ЦЕЛЬ ЗАНЯТИЯ

Изучение внутрисхемного отладчика JTAG ICE. Получение навыков отладки программы с помощью внутрисхемного отладчика.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Отладка программного обеспечения на основе разработанных аппаратных средств представляет собой практически важную задачу, которая не может быть решена отдельно с использованием только программного симулятора AVRSimulator. Для решения таких задач необходимы специальный инструментальные средства, переносящие функции тестирования симуляторов на реальную аппаратуру – фоновые внутрисхемные отладчики. Для МК семейства AVR таким инструментальным средством является отладчик JTAG ICE.

Интерфейс JTAG был разработан группой ведущих специалистов по проблемам тестирования электронных компонентов (Joint Test Action Group). В дальнейшем он был зарегистрирован в качестве промышленного стандарта IEEE Std 1149.1-1990 (IEEE Standart Test Access Port and Boundary-Scan Architecture).

Встроенный в ряд микроконтроллеров AVR интерфейс JTAG может быть использован как для внутрисхемной фоновой отладки, так и для программирования Flash и EEPROM памяти контроллера.

Доступ к модулю JTAG осуществляется через четыре вывода микроконтроллеров, составляющих так называемый «порт тестового доступа» (Test Access Port, TAP): TMS, TCK, TDI и TDO.

JTAG ICE(рис.1) представляет собой инструмент для отладки устройств, выполненных на основе микроконтроллеров семейства AVR фирмы Atmel.

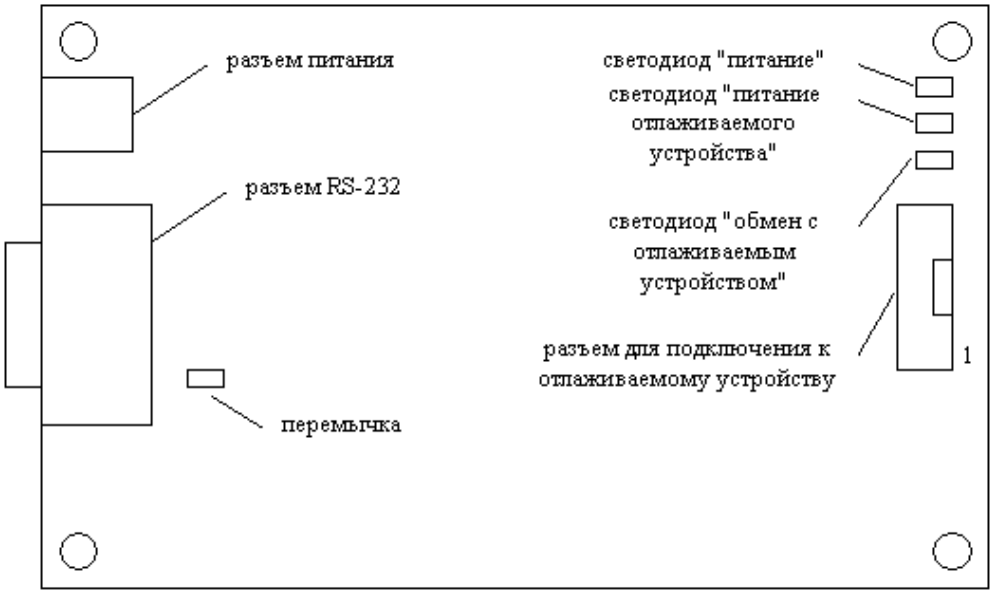

Рис.1. Расположение элементов на плате AVR JTAG.

Рассматриваемый в работе JTAG ICE поддерживает следующие микроконтроллеры семейства, имеющие интерфейс JTAG: ATmega16, ATmega162, ATmega169, ATmega32, ATmega323, ATmega64, ATmega128.

Чтобы программа AVR Studio могла автоматически обнаружить JTAG ICE, необходимо подсоединить JTAG ICE к компьютеру (порт COM), отлаживаемому устройству и подать питание на JTAG ICE и отлаживаемое устройство до запуска AVR Studio.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Подключение к компьютеру.

JTAG ICE подключается к стандартному COM порту компьютеру. Для подключения используется 9-ти контактный "прямой" кабель, который входит в комплект поставки.

При запуске AVR Studio автоматически ищет поддерживаемые им устройства, обращаясь по порядку ко всем COM портам компьютера. Поиск прекратится, как только будет найдено поддерживаемое устройство. Например, если к COM1 подключен программатор AVR ISP, а к COM2 отладчик – JTAG ICE, то AVR Studio обнаружит только AVR ISP. Поэтому при запуске AVR Studio убедитесь, что остальные поддерживаемые им устройства отключены или после запуска AVR Studio вручную задайте порт, к которому подключен JTAG ICE.

Если COM порт, к которому подключен JTAG ICE, используется другими программами (например, терминалом), то AVR Studio не сможет обнаружить JTAG ICE. Завершите эти программы перед запуском AVR Studio.

Подключение к отлаживаемому устройству.

Для подключения к отлаживаемому устройству используются 6 линий: TCK, TDO, TDI, TMS, VTref и GND (Рис.2), подключение этих линий необходимо для правильной работы JTAG ICE с отлаживаемым устройством.

Дополнительно к отлаживаемому устройству может быть подключена линия nSRST (используется для управления и наблюдения за линией сброса микроконтроллера). Использование этого сигнала не является необходимым для отладки, однако, если программа микроконтроллера установит бит JTD в регистре MCUCSR, то интерфейс JTAG будет отключен и для его включения потребуется, чтобы JTAG ICE мог управлять линией сброса микроконтроллера.

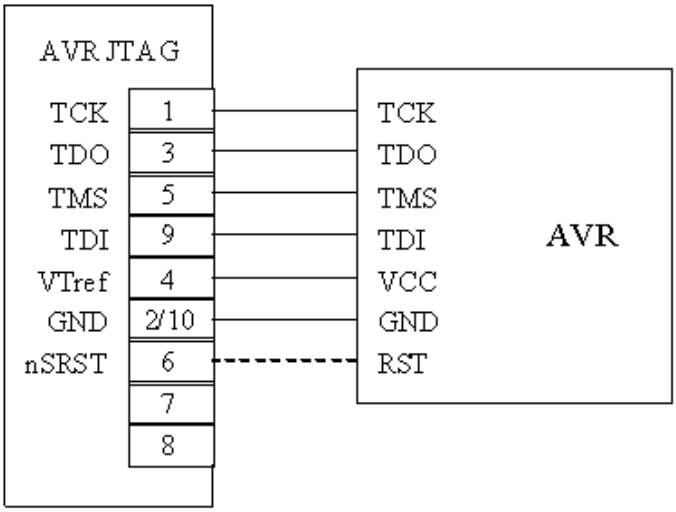

Рис.2. Подключение AVR JTAG к отлаживаемому устройству.

Для подключения отладчика JTAG ICE к целевому контроллеру, установленному на STK500, удобно использовать специальный переходник, устанавливаемый в гнездо расширения EXPAND0 платы STK500. Но можно и непосредственно подсоединить линии порта JTAG отладчика к соответствующим выводам микроконтроллера.

Подключение питания.

JTAG ICE получает питание от внешнего источника питания (входит в комплект поставки) постоянного или переменного тока. Для подключения источника к JTAG ICE используется гнездо под штекер 2,5 мм. Полярность источника постоянного тока не имеет значения.

Для того чтобы AVR Studio обнаружило JTAG ICE необходимо, чтобы питание JTAG ICE и отлаживаемого устройства было включено до начала отладки (кнопкой Start Debugging AVR Studio).

Рекомендуется следующий порядок включения:

1. включите питание отлаживаемого устройства;
2. включите питание JTAG ICE;
3. запустите AVR Studio.

Для работы отладчика JTAG ICE необходимо, чтобы в целевом микроконтроллере были установлены FUSE-бит JTAGEN, разрешающий работу порта JTAG, и FUSE-бит OCDEN, разрешающий фоновую внутрисхемную отладку, а все Lock-биты должны быть сброшены.

JTAG ICE, помимо отладки, может быть использован для внутрисхемного программирования микроконтроллера по интерфейсу JTAG. Это очень полезная функция, т.к. по умолчанию на всех микросхемах, поступающих от производителя, FUSE-бит JTAGEN установлен, а FUSE-бит OCDEN сброшен, т.е. JTAG интерфейс включен, но фоновая отладка запрещена. Для того чтобы использовать JTAG ICE в качестве внутрисхемного программатора, необходимо в AVRStudio выбрать меню **Tools→Program AVR→Connect…** В появившемся

окне **Select AVR Programmer** (Рис.3) выбираем JTAG ICE и порт, к которому он подключен.

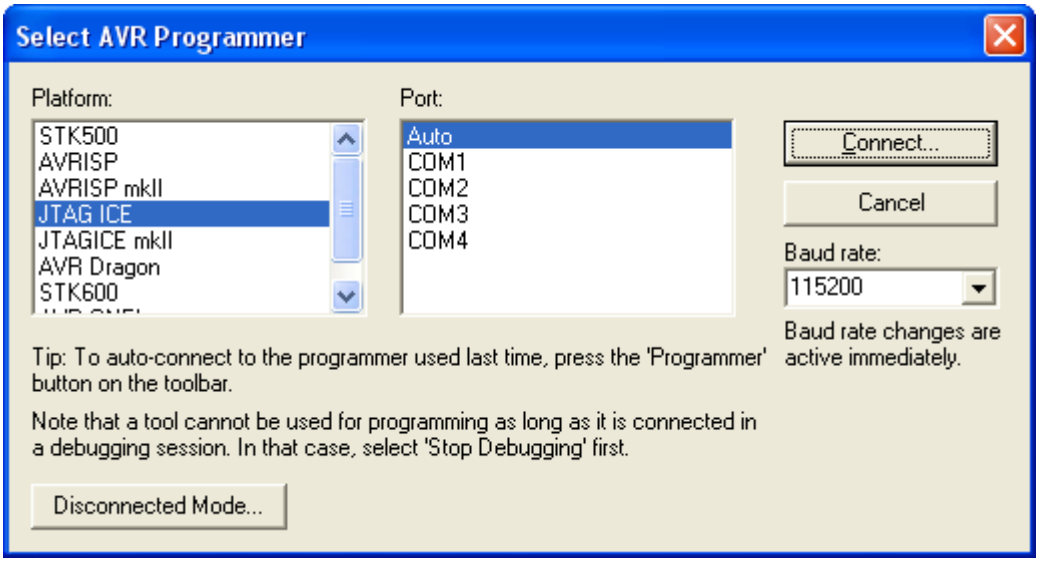

Рис.3. Окно выбора программатора.

После чего нажимаем на кнопку **Connect…** и, если всё подключено правильно, появится окно программатора AVR (Рис.4), знакомое вам по работе с STK500. Принципы работы с JTAG ICE в режиме внутрисхемного программирования через интерфейс JTAG , аналогичны принципам работы с STK500 в режиме внутрисхемного программирования через интерфейс SPI, рассмотренным ранее. Поэтому обратим внимание лишь на то, чтобы FUSE-бит OCDEN был установлен (Рис.5).

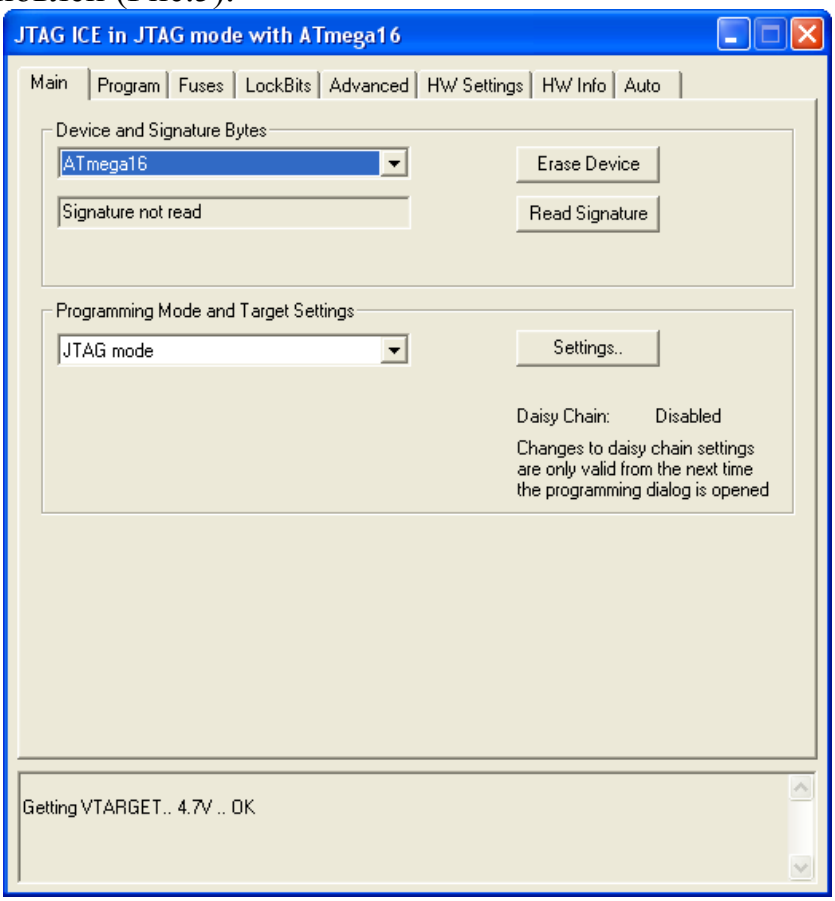

Рис.4. Окно AVR программатора.

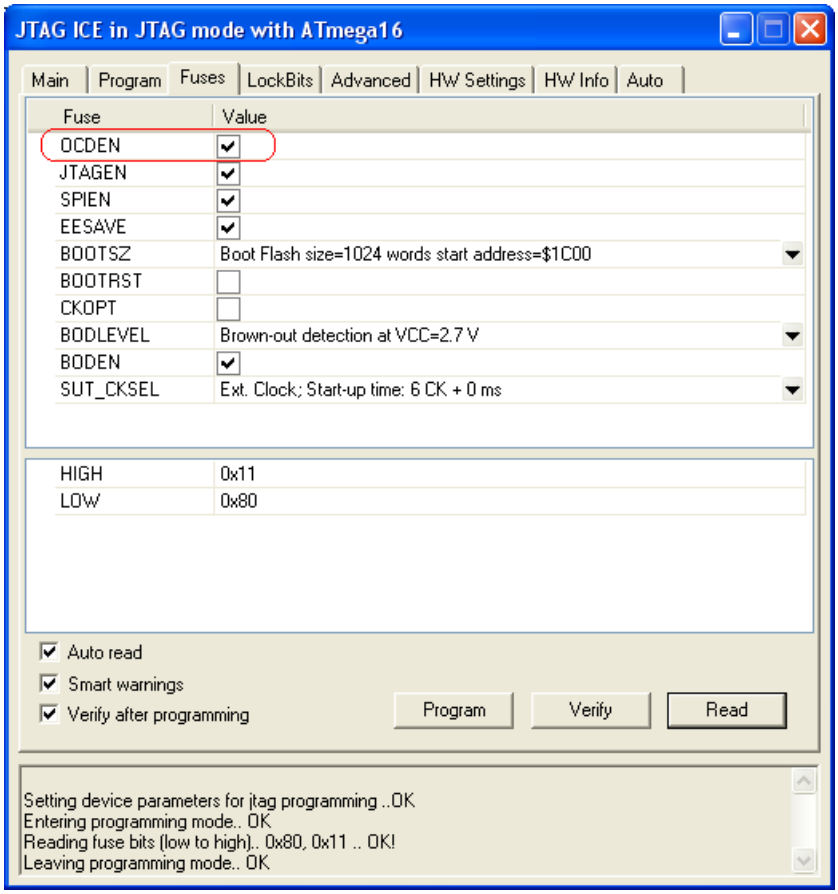

Рис.5. Программирование FUSE-бит.

Отладка учебной программы с помощью JTAG ICE.

В качестве примера работы с фоновым отладчиком JTAG ICE будем использовать учебную программу **L3_1.ASM** (Листинг 1). Эта программа написана для микроконтроллера ATmega16, установленного в стартовый набор STK500 таким образом, что светодиоды подключены к порту B микроконтроллера, а кнопки к порту D. Данная программа использует линию внешнего прерывания INT0 микроконтроллера ATmega16, к которой подключена кнопка SW2. При нажатии на эту кнопку происходит прерывание, обработчик которого (SW2INT) изменяет состояние светодиода LED2 и увеличивает счётчик нажатий (ClickCounter) на единицу.

Листинг 1.

```
.include "m16def.inc"
.include "STK500cfg.inc"

.def  temp1 = r16
.def  temp2 = r17

.DSEG
      ClickCounter:     .BYTE 1
```

```
.CSEG
        rjmp    RESET
.org    INT0addr
        rjmp    SW2INT
.org    INT_VECTORS_SIZE

RESET:
        ;Инициализация стека
        ldi     temp1, low(RAMEND)
        out     SPL, temp1
        ldi     temp1, high(RAMEND)
        out     SPH, temp1

        ;Очищаем счётчик нажатий
        clr     temp1
        sts     ClickCounter, temp1

        ;Инициализация светодиодов
        ldi     temp1, 0xFF
        out     DDR_LED, temp1
        out     PORT_LED, temp1

        ;Инициализация выводов, к которым подключены кнопки (на вход)
        ldi     temp1, 0
        out     DDR_SW, temp1
        out     PORT_SW, temp1

        ;Инициализация внешнего прерываня INT0
        ldi     temp1, (1<<ISC01)|(0<<ISC00)
        out     MCUCR, temp1
        ldi     temp1, (1<<INT0)
        out     GICR, temp1

        sei
loop:
        nop
        rjmp    loop

SW2INT:
        ;Изменяем состояние светодиода LED2
        in      temp1, PORT_LED
        mov     temp2, temp1
        sbrs    temp2, bLED2
        sbr     temp1, (1<<bLED2)
        sbrc    temp2, bLED2
        cbr     temp1, (1<<bLED2)
        out     PORT_LED, temp1
        ;Инкрементируем счётчик нажатий
        lds     temp1, ClickCounter
        inc     temp1
        sts     ClickCounter, temp1
        reti
```

Итак, запустите AVRStudio и откройте проект **L3_1**. Теперь необходимо выберите JTAG ICE в качестве отладочной платформы, если это не было сделано ранее (информацию о текущей платформе и целевом устройстве можно увидеть в строке состояния главного окна AVRStudio). Для этого воспользуйтесь меню **Debug→Select Platform and Device…**, после чего в открывшемся окне (Рис.6) в списке **Debug platform** выберите JTAG ICE, а в списке **Device** – целевой контроллер, в нашем случае это ATmega16.

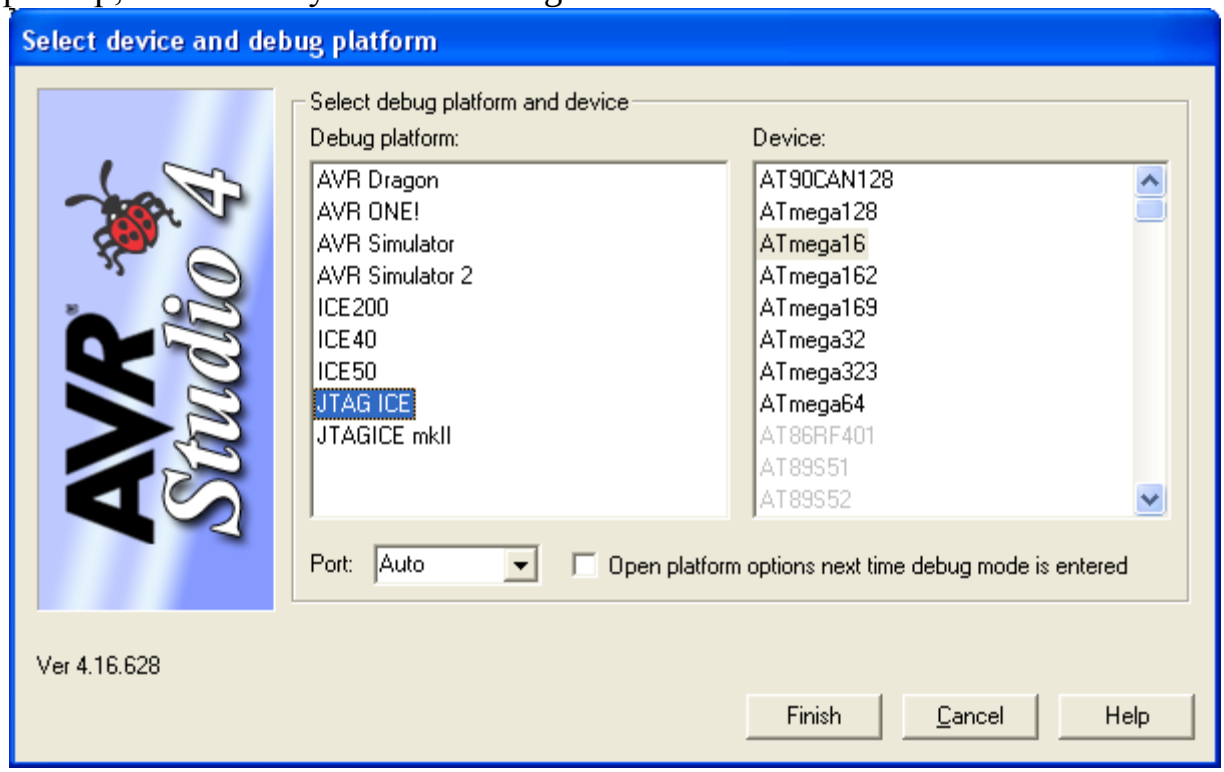

Рис.6. Окно выбора платформы отладки.

Теперь можно скомпилировать программу и запустить отладку, также как и при работе с симулятором. При этом новый код автоматически загрузиться в микроконтроллер.

Запустите программу, используя кнопку **Run** панели инструментов Debug либо клавишу F5. Нажмите на кнопку SW2 на плате STK500 несколько раз, светодиод LED2 должен изменять своё состояние (загораться или гаснуть) при каждом нажатии на кнопку SW2. Теперь сделайте так, чтобы светодиод LED2 находился во включённом состоянии и приостановите выполнение программы, нажав на кнопку **Break** панели инструментов Debug. В целом управление отладкой программы на JTAG ICE, аналогично управлению отладкой на симуляторе AVRSimulator.

Итак, выполнение программы приостановлено, и мы можем посмотреть текущее состояние регистров микроконтроллера в окне **I/O View**. Откройте в нём регистры порта B (Рис.7), к которому подключены светодиоды:

Рис.7. Состояние регистров порта B.

Как видите бит 2 регистра PORTB сброшен, что соответствует включённому состоянию светодиода LED2, подключенного к линии PB2 порта B. Изменяйте состояния этого и других битов регистра PORTB, при этом соответствующие светодиоды на плате STK500 будут загораться или гаснуть. Как видите, в отличие от симулятора, при работе с фоновым отладчиком, программист имеет доступ непосредственно к регистрам (а также всей памяти) целевого микроконтроллера. Для того, что бы посмотреть количество нажатий на кнопку, в окне **Memory** выберите область памяти Data и введите адрес переменной ClickCounter (его можно узнать, наведя курсор мыши на имя ClickCounter в тексте программы). Теперь можно посмотреть количество нажатий на кнопку. Но для работы с короткими переменными, окном **Memory** пользоваться неудобно, поэтому воспользуемся окном **Watch**. Кликните правой кнопкой мыши на имя ClickCounter в тексте программы и в открывшемся меню выберите пункт **Add Watch: "ClickCounter"**. Теперь в окне **Watch** появилась соответствующая запись (Рис.8):

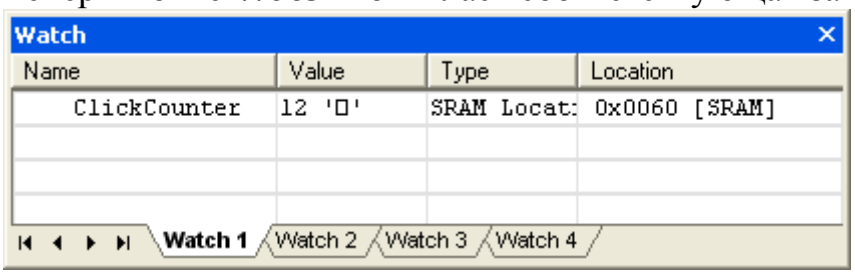

Рис.8. Окно **Watch**.

Продолжите выполнение программы и нажмите на кнопку SW2 несколько раз. После этого приостановите выполнение программы и в окне **Watch** посмотрите, как изменилось значение переменной ClickCounter

Теперь перезапустите программу нажав на кнопку Reset панели инструментов Debug. Поставьте точку останова на начало обработчика прерывания INT0, для чего установите курсор на строчку **in** temp1, PORT_LED нажмите кнопку Toggle Breakpoint или клавишу F9. Запустите выполнение программы и нажмите на кнопку SW2 на плате STK500. Выполнение программы прервётся на строчке, помеченной точкой останова. Аналогично переменой ClickCounter добавьте в окно **Watch** регистры с именами temp1 и temp2 и продолжите выполнение программы в пошаговом режиме, нажимая кнопку Step Over панели Debug или клавишу F10. Обратите внимание на то, как меняются значения регистров temp1 и temp2, переменной ClickCounter, а также изменяется состояние светодиода LED2, в процессе выполнения программы. После возврата из процедуры обработки прерывания завершите отладку программы.

ЗАДАНИЕ

Изучить возможности внутрисхемного отладчика JTAG ICE на примере отладки учебной программы **L3_1.asm**. Изменить учебную программу таким образом, чтобы при нажатии на кнопку SW2 включались светодиодными то чётными, то с нечётными номерами.

ПОРЯДОК ВЫПОЛНЕНИЯ

1. Подключите JTAG ICE к ПК и к целевому устройству.
2. Запустите AVRStudio и откройте учебный проект **L3_1**.
3. Подайте питание на отладчик и целевое устройство.
4. Откройте окно утилиты AVRProgrammer и убедитесь, что связь установлена.

ОФОРМЛЕНИЕ ОТЧЁТА

Отчёт должен содержать:

- краткие теоретические сведения
- описание проделанных действий и их результаты
- вывод по проделанной работе

Практическое занятие №3-4

Работа с портами ввода-вывода и прерываниями

ЦЕЛЬ И ЗАДАЧИ

Изучить устройство портов ввода-вывода микроконтроллеров семейства AVR и научиться работать с ними.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Порты ввода-вывода.

Микроконтроллеры серии AVR всегда имеют в своём составе от одного до семи портов ввода/вывода. Каждый разряд такого порта подсоединен к одному из выводов (контактов) микросхемы. Порты ввода/вывода служат для обмена информацией с внешними устройствами. Порты могут быть полными и неполными. Полный порт содержит 8 разрядов. В неполных портах задействовано менее восьми разрядов. Каждый порт имеет свое имя. Они именуются латинским буквами от A до G.

Все порты ввода-вывода (ПВВ) AVR-микроконтроллеров работают по принципу чтение-модификация-запись при использовании их в качестве портов универсального ввода-вывода. Это означает, что изменение направления ввода-вывода одной линии порта командами SBI и CBI будет происходит без ложных изменений направления ввода-вывода других линий порта. Данное распространяется также и на изменение логического уровня (если линия порта настроена на вывод) или на включение/отключение подтягивающих резисторов (если линия настроена на ввод). Каждый выходной буфер имеет симметричную характеристику управления с высоким втекающим и вытекающим выходными токами. Выходной драйвер обладает нагрузочной способностью, которая позволяет непосредственно управлять светодиодными индикаторами. Ко всем линиям портов может быть подключен индивидуальный выборочный подтягивающий к плюсу питания резистор, сопротивление которого не зависит от напряжения питания. На всех линиях ПВВ установлены защитные диоды, которые подключены к VCC и Общему (GND), как показано на рисунке 1.

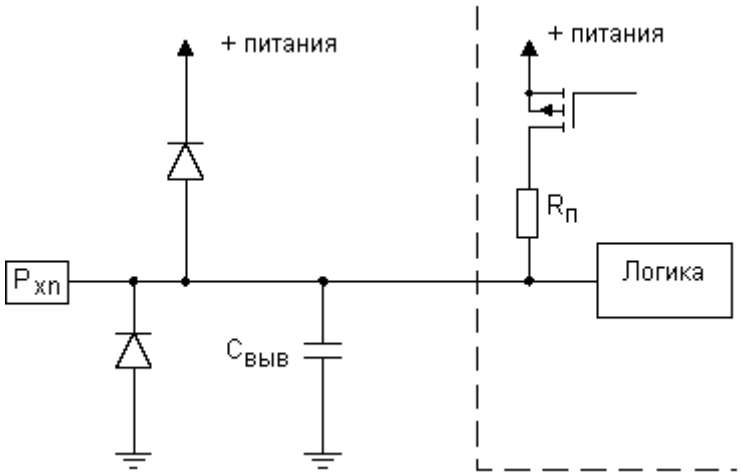

Рис.1. Эквивалентная схема ПВВ.

Для управления каждым портом ввода/вывода используют три специальных регистров ввода/вывода (РВВ). Это регистры: PORTx, DDRx и PINx. Под «x» здесь подразумевается конкретная буква – имя порта. Например, для порта А имена регистров управления будут такими: PORTA, DDRA и PINA. Рассмотрим теперь назначение каждого из этих регистров:

PORTx – регистр данных (используется для **вывода** информации);

DDRx – регистр направления передачи информации;

PINx – регистр **ввода** информации.

Отдельные разряды приведенных выше регистров также имеют свои имена.

Разряды регистра PORTx обычно именуются как Pxi, где «i» - номер соответствующего разряда. К примеру разряды порта PORTA будут именоваться следующим образом: PA0, PA1, PA2 – PA7.

Разряды регистра DDRx именуются как DDXi, для порта А: DDA0, DDA1 – DDA7.

Разряды регистра PINx именуются как PINxi, для порта А: PINA0, PINA1 – PINA7.

Для других портов буква А заменяется соответственно на B, C, D, E, F, G.

Функциональная схема одного разряда порта ввода/вывода приведена на рисунке 2.

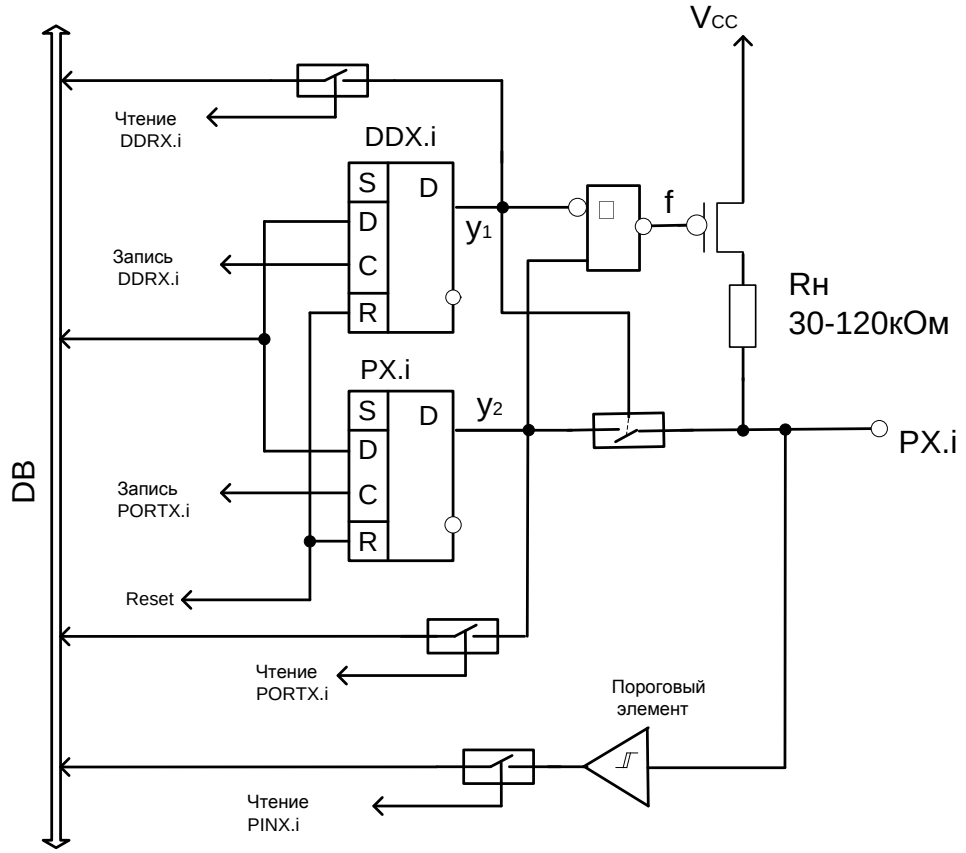

Рис.2. Функциональная схема одного разряда порта ввода/вывода.

Для того, чтобы выдать нужную информацию на внешний вывод микросхемы МК, нужно в соответствующий разряд регистра DDRx записать логическую единицу, а затем записать байт данных в регистр PORTx. При этом содержимое соответствующего бита этого байта тут же появится на внешнем выводе микросхемы и будет присутствовать там постоянно, пока не будет заменено другим, либо пока данная линия порта не переключится на ввод.

Для того чтобы прочитать информацию с внешнего вывода МК, необходимо сначала перевести нужный разряд порта в режим ввода. То есть записать в соответствующий разряд регистра DDRx ноль. Только после этого на данный вывод МК можно подавать цифровой сигнал от внешнего устройства. Далее МК просто читает байт из регистра PINx. Содержимое соответствующего бита прочитанного байта будет соответствовать сигналу на внешнем выводе порта.

Порты ввода/вывода МК серии AVR имеют еще одну полезную функцию. В режиме ввода информации они могут при необходимости подключать к каждому выводу порта внутренний нагрузочный резистор. Внутренний резистор позволяет значительно расширить возможности порта. Такой резистор создает вытекающий ток для внешних устройств, подключенных между выводом порта и общим проводом.

Благодаря этому резистору упрощается подключение внешних контактов и кнопок. Обычно контакты требуют внешнего резистора. Теперь внешнего резистора можно обойтись. Включением и отключением внутренних резисторов управляет регистр PORTx, если порт находится в режиме ввода. Указанные режимы сведены в таблицу 1 (конфигурирование порта ввода/вывода).

Таблица 1

DDxn	Pxn	Режим	Резистор	Примечание
0	0	Вход	Отключен	Вывод отключен от схемы
0	1	Вход	Подключен	Вывод является источником тока
1	0	Выход	Отключен	На выходе «0»
1	1	Выход	Отключен	На выходе «1»

Задание значения выходного сигнала на отдельных выводах порта выполняется с использованием команд с мнемокодами:

```
CBI   PortX, I     ; задание низкого уровня
SBI   PortX, I     ; задание высокого уровня
```

При совместном использовании всех разрядов порта для приема и выдачи байтов используются команды с мнемокодами

```
IN    Rd, PinX         ; ввод байта
OUT   PortX, Rd        ; вывод байта.
```

Обращение к параллельному порту для приема и выдачи байтов может быть выполнено с использованием команд с обращением по адресам в адресном пространстве SRAM:

```
LD    Rd, Y      ; ввод байта
ST    Y, Rd      ; вывод байта.
```

Внешние прерывания.

Внутри МК или в связанной с ним внешней среде могут возникать разнообразные события, которые требуют немедленной реакции на свое появление, т.е. временной приостановки процесса обработки основной программы («прерывания») и выполнения другой программы (процедуры), предназначенной для обслуживания возникшего события. Соответственно такие события называются прерывающими событиями. Реакция контроллера на внешнее событие должна завершаться возвратом к основной программе.

Особенностью системы прерываний МК является то, что в МК, все прерывания аппаратные, которые в свою очередь делятся на внутренние и внешние.

Внутреннее прерывание это событие, которое возникает в МК и прерывает выполнение основной программы. Внутренних прерываний в МК семейства AVR довольно много и в совокупности они служат для организации взаимодействия устройств с ядром системы. Внутренние прерывания могут возникать от таймеров, аналогового компаратора, последовательного порта и т. п.

Внешние прерывания инициируются сигналами, которые формируются на соответствующих выводах МК, устройствами, являющимися внешними по отношению к МК. Эти выводы обозначаются INTx (где x – номер соответствующей линии внешнего прерывания). Функции ввода сигналов прерывания являются альтернативными функциями портов МК.

Выделяют три вида событий, вызывающих внешнее прерывание: это может быть низкий уровень напряжения, а также положительный или отрицательный фронт на соответствующем выводе. Таким образом, система прерываний может работать в одном из четырех возможных режимов:

- режим прерывания по уровню (статический, асинхронный режим);
- режим прерывания по фронту (нарастанию);
- режим прерывания по срезу сигнала (спаду);
- режим прерывания по фронту и срезу сигнала.

Последние три режима относятся к динамическим режимам и в свою очередь могут быть асинхронными и синхронными.

Для управления прерываниями используются специальные регистры ввода-вывода. В микроконтроллере ATmega16 это главный регистр управления прерываниями GICR (таблица 1) и частично регистр управления микроконтроллером MCUCR (таблица 2).

Таблица 1.

7	6	5	4	3	2	1	0	
INT1	INT0	INT2	-	-	-	IVSEL	IVSE	**GICR**

Если разряд **INT1** установлен в лог. 1, то внешнее прерывание 1 будет разрешено до тех пор, пока будет установлен разряд общего разрешения прерываний в регистре состояния.

Если разряд **INT0** установлен в лог. 1, то будет разрешено внешнее прерывание 0 до тех пор, пока в регистре состояния будет установлен разряд общего разрешения прерываний.

Если разряд **INT2** установлен в лог. 1, то будет разрешено внешнее прерывание 2 до тех пор, пока в регистре состояния будет установлен разряд общего разрешения прерываний.

Таблица 2.

7	6	5	4	3	2	1	0	
SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	**MCUCR**

Выбор режима вызова внешнего прерывания INTx определяется установкой соответствующих бит ISCx1:0 регистра управления микроконтроллером MCUCR (Таблица 3).

Таблица 3.

ISCx1	ISCx0	Описание
0	0	Низкий уровень на входе INTx генерирует запрос на прерывание
0	1	Любое изменение логического уровня на входе INTx генерирует запрос на прерывание
1	0	Спад входного сигнала INTx вызывает запрос на прерывание
1	1	Фронт входного сигнала INTx вызывает запрос на прерывание

ПРАКТИЧЕСКАЯ ЧАСТЬ

Для демонстрации работы с портами ввода-вывода воспользуемся программой L4_1.asm (Листинг 1).

Листинг 1

```
.include "m16def.inc"

.def temp = r16
.def delay1 = r17
.def delay2 = r18

.CSEG
RESET:
        ;Инициализация светодиодов
        ldi     temp, 0xFF
        out     DDRB, temp
        out     PORTB,temp

        ;Инициализация выводов, к которым подключены кнопки (на вход)
        ldi     temp, 0
```

```
        out     DDRD, temp
        out     PORTD, temp

        ldi     temp, 0b11111110  ;Значение соответствует включению св. LED0
        sec                       ;Устанавливаем Carry-флаг
LOOP:
        out     PORTB,temp        ;Загружаем содержимое регистра temp в порт B
        sbis    PIND, 2           ;Нажата кнопка SW0
        ror     temp              ;Сдвигаем регистр temp вправо
        brcs    M1
        ldi     temp, 0b01111111 ;Значение соответствует включению св. LED7
        sec                       ;Устанавливаем Carry-флаг
M1:
        sbis    PIND, 3           ;Нажата кнопка SW1
        rol     temp              ;Сдвигаем регистр temp влево
        brcs    DELAY
        ldi     temp, 0b11111110  ;Значение соответствует включению св. LED0
        sec                       ;Устанавливаем Carry-флаг

DELAY:
        ;Задержка
        inc     delay1
        brne    DELAY
        inc     delay2
        brne    DELAY
        rjmp    LOOP
```

Программа L4_1.asm выполняет следующие действия: при нажатии на кнопку SW0 (SW1) включает светодиод, находящийся справа (слева) от того, который светится в данный момент, а его самого выключает. При достижении крайнего светодиода, следующим включается другой крайний.

Как видно из листинга 1 в основной цикл программы введена задержка. Это сделано для того, чтобы событие нажатия возникало не постоянно при нажатой кнопке, а через значительные по меркам МК интервалы времени. Если эту задержку убрать, то при нажатии на кнопку мы увидим, как загорятся сразу все светодиоды, т.к. наш глаз просто не успевает отследить, как происходят переключения.

Более широкие возможности для работы с кнопками и другими внешними источниками событий предоставляют внешние прерывания. В листинге 2 представлена программа из проекта **L4_2**, выполняющая те же действия, что и предыдущая, но с использованием внешних прерываний.

Листинг 2.

```
.include "m16def.inc"

.def temp = r16

.CSEG
        rjmp    RESET
```

```
.org    INT0addr      ;Адрес обработчика прерывания 0 в таблице прерываний
        rjmp  INT0Routing ;Переход на обработчик прерывания 0
.org    INT1addr      ;Адрес обработчика прерывания 1 в таблице прерываний
        rjmp  INT1Routing ;Переход на обработчик прерывания 1
.org    INT_VECTORS_SIZE

RESET:
        ;Инициализация стека
        ldi     temp, low(RAMEND)
        out     SPL, temp
        ldi     temp, high(RAMEND)
        out     SPH, temp

        ;Инициализация светодиодов
        ldi     temp, 0xFF
        out     DDRB, temp
        ldi     temp, 0xFE
        out     PORTB, temp

        ;Инициализация выводов, к которым подключены кнопки (на вход)
        ldi     temp, 0
        out     DDRD, temp
        out     PORTD, temp

        ;Инициализация внешних прерываний INT0 и INT1
        ldi     temp, (1<<ISC01)|(1<<ISC00)|(1<<ISC11)|(1<<ISC10)
        out     MCUCR, temp
        ldi     temp, (1<<INT0)|(1<<INT1)
        out     GICR, temp

        sec     ;Устанавливаем Carry-флаг
        sei     ;Разрешаем прерывания
LOOP:
        nop
        rjmp  LOOP


INT0Routing:
        ;Сохраняем контекст
        push  temp
        in      temp, SREG
        push  temp

        in      temp, PORTB         ;Загружаем в регистр temp содержимое порта B
        ror     temp                ;Сдвигаем регистр temp вправо
        brcs  INT0Routing_m
        ldi     temp, 0b01111111
        sec                         ;Устанавливаем Carry-флаг
INT0Routing_m:
        out     PORTB, temp         ;Загружаем содержимое регистра temp в порт B

        ;Востанавливаем контекст
```

```
        pop     temp
        out     SREG, temp
        pop     temp
        reti


INT1Routing:
        ;Сохраняем контекст
        push    temp
        in      temp, SREG
        push    temp

        in      temp, PORTB         ;Загружаем в регистр temp содержимое порта B
        rol     temp                ;Сдвигаем регистр temp влево
        brcs    INT1Routing_m
        ldi     temp, 0b11111110
        sec                         ;Устанавливаем Carry-флаг
INT1Routing_m:
        out     PORTB, temp         ;Загружаем содержимое регистра temp в порт B

        ;Востанавливаем контекст
        pop     temp
        out     SREG, temp
        pop     temp
        reti
```

Как видно из листинга 2 прерывания INT0 и INT1 настроены на возникновение по фронту соответствующего входного сигнала. Обратите внимание на то, что в каждом прерывании сохраняется и восстанавливается т.н. контекст, в данном случае – это регистр статуса (SREG) и регистр общего назначения temp (r16). Т.е. это те регистры, которые используются в прерывании, а так как прерывание может возникнуть в любом участке программы, который также может использовать эти регистры (это и называется контекст текущей задачи), необходимо сохранить их значения и восстановить перед выходом из прерывания. Конечно, в данной программе необходимости в сохранении контекста нет, т.к. в отсутствии прерываний выполняется бесконечный пустой цикл, но для самодисциплины лучше никогда не пренебрегать сохранением контекста.

ЗАДАНИЕ

Изучить на практике работу портов ввода-вывода микроконтроллера. Научиться использовать внешние прерывания. Сравнить работу с кнопками без использования и с использованием прерываний.

ПОРЯДОК ВЫПОЛНЕНИЯ

1. Подключите светодиоды (разъём LEDS на плате STK500) к порту B целевого микроконтроллера с помощью десятижильного плоского кабеля.

2. Подключите кнопки SW0 и SW1 к выводам PD2 и PD3 микроконтроллера соответственно с помощью двухжильного провода.
3. Запустите AVR Studio и откройте проект **L4_1**.
4. Включите питание на STK500 и загрузите в целевой МК программу проекта **L4_1**. Изучите исходный код программы.
5. Нажимая кнопки SW0 и SW1, понаблюдайте за переключением светодиодов.
6. Измените программу таким образом, чтобы в том случае, когда светится один из крайних светодиодов при нажатии соответствующей кнопки не загорался светодиод с другого края, а продолжал светиться текущий светодиод.
7. Теперь закройте текущий проект и откройте проект **L4_2**. Изучите программу, представленную в этом проекте.
8. Откомпилируйте и загрузите в целевой МК программу из проекта **L4_2**. Убедитесь в её работоспособности.
9. Измените настройку внешних прерываний INT0 и INT1 таким образом, чтобы событие по прерыванию 0 возникало при отпускании кнопки SW0, а событие по прерыванию 1 возникало как при нажатии, так и при отпускании кнопки SW1.

Практическое занятие №5

Работа с таймерами/счетчиками и формирование ШИМ

ЦЕЛЬ И ЗАДАЧИ

Получить навыки работы с таймерами-счетчиками общего назначения микроконтроллеров семейства AVR. Научится формировать ШИМ-сигнал с помощью таймера-счетчика. Изучить работу сторожевого таймера.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Любой микроконтроллер серии AVR содержит несколько встроенных таймеров. Причем по своему назначению их можно разделить на две категории. К первой категории относятся таймеры общего назначения. Вторую категорию составляет сторожевой таймер. Сторожевой таймер предназначен для автоматического перезапуска микроконтроллера в случае «зависания» его программы.

Таймеры общего назначения используются для формирования различных интервалов времени и прямоугольных импульсов заданной частоты. Кроме того, они могут работать в режиме счетчика и подсчитывать тактовые импульсы заданной частоты, измеряя, таким образом, длительность внешних сигналов, а также при необходимости подсчитывать количество любых внешних импульсов. По этой причине данные таймеры называют «таймеры/счетчики». В микросхемах AVR применяются как восьмиразрядные, так и шестнадцатиразрядные таймеры/счетчики. Их количество для разных микроконтроллеров различно.

Каждый восьмиразрядный таймер представляет собой один восьмиразрядный регистр, который для микроконтроллера является регистром ввода-вывода. Этот регистр хранит текущее значение таймера и называется счетным регистром. Шестнадцатиразрядные таймеры имеют шестнадцатиразрядный счетный регистр. Каждый счетный регистр имеет свое имя.

Счетный регистр восьмиразрядного таймера именуется TCNTx, где «x» – это номер таймера. Для таймера T/C0 регистр называется TCNT0. Для таймера T/C2 – TCNT2. Шестнадцатиразрядные регистры именуются похожим образом. Отличие в том, что каждый шестнадцатиразрядный счетный регистр для микроконтроллера представляет собой два регистра ввода-вывода. Один предназначен для хранения старших битов числа, а второй – для хранения младших битов. К имени регистра старших разрядов добавляется буква H, а для регистра младших разрядов добавляется буква L. Таким образом, счетный регистр таймера T/C1 – это два регистра ввода-вывода: TCNT1H и TCNT1L.

Микроконтроллер может записать в любой счетный регистр любое число в любой момент времени, а также в любой момент прочитать содержимое любого счетного регистра. Когда таймер включается в режим счета, то на его вход начинают поступать счетные импульсы. После прихода каждого такого импульса содержимое счетного регистра увеличивается на единицу. Счетными импульсами могут служить как специальные тактовые импульсы, вырабатываемые внутри

самого микроконтроллера, так и внешние импульсы, поступающие на специальные входы микросхемы. При переполнении счетного регистра его содержимое обнуляется, и счет начинается сначала.

Любой таймер жестко завязан с системой прерываний. Вызвать прерывание может целый ряд событий, связанных с таймером. Например, существует прерывание по переполнению таймера, по срабатыванию специальной схемы совпадения. Отдельные прерывания может вызывать сторожевой таймер.

Управление работой таймера осуществляется с помощью специальных регистров ввода-вывода. Набор этих регистров несколько отличается в зависимости от типа таймера и микроконтроллера. Для 16-разрядного таймера/счётчика 1 микроконтроллера ATmega16 это регистры TCCR1A, TCCR1B, TIMSK.

Регистр управления таймером TCCR1A

Бит	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Начальное значение	0	0	0	0	0	0	0	0	

- Биты 7:6 – COM1A1:0 : Режим выхода по совпадению канала A.
- Биты 5:4 – COM1B1:0 : Режим выхода по совпадению канала B.

Таблица 1. Функции битов COMnx1:0 при работе таймера/счетчика в *Нормальном режиме* и *Режиме очистки по совпадению (CTC)*.

COMnA1,COMnB1	COMnA0,COMnB0	Описание
0	0	Выход OCnx отключен от порта.
0	1	Переключение выхода OCnx при совпадении.
1	0	Очистка выхода OCnx при совпадении (установка на линии низкого логического сигнала).
1	1	Установка выхода OCnx при совпадении (установка на линии высокого логического сигнала).

Таблица 2. Функции битов COMnx1:0 при работе таймера/счетчика в *Режиме быстродействующая ШИМ*.

COMnA1,COMnB1	COMnA0,COMnB0	Описание
0	0	Выход OCnx отключен от порта.
0	1	1)Для режима 15 (WGM13:0=15) переключение OCnA при совпадении, выходы OCnB/C отключен от порта; 2)Для всех остальных режимов работы выход OCnx отключен от порта.
1	0	Очистка OCnx при совпадении, установка при достижении базовым счетчиком максимального значения.
1	1	Установка OCnx при совпадении,

		очистка при достижении базовым счетчиком максимального значения.

Таблица 3. Функции битов COMnx1:0 при работе таймера/счетчика в *Режиме ШИМ с коррекцией фазы* и в *Режиме ШИМ с коррекцией фазы и частоты*.

COMnA1,COMnB1	COMnA0,COMnB0	Описание
0	0	Выход OCnx отключен от порта.
0	1	1)Для режимов 9 и 14 (WGM13:0=9 или 14) переключение OCnA при совпадении, выходы OCnB/C отключен от порта; 2)Для всех остальных режимов работы выход OCnx отключен от порта.
1	0	Очистка OCnx при совпадении, когда счетчик увеличивается. Установка OCnx при совпадении, когда счетчик уменьшается.
1	1	Установка OCnx при совпадении, когда счетчик увеличивается. Очистка OCnx при совпадении, когда счетчик уменьшается.

- Бит 3 – FOC1A : Стробирование совпадения по каналу A.
- Бит 2 – FOC1B : Стробирование совпадения по каналу B.
- Биты 1:0 – WGM11:0 : Режим работы Таймера/Счетчика1.

Режим работы таймера/счетчика определяется установкой битов режима генерации сигнала (Waveform Generation mode) WGMn3:0. Они расположены в регистрах управления таймера/счетчика A и B (TCCRnA и TCCRnB). В регистре TCCRnB расположены биты **WGM13:2**.

Регистр управления таймером TCCR1B

Бит	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Начальное значение	0	0	0	0	0	0	0	0	

- Бит 7 – ICNC1: Фильтр шума модуля захвата.
- Бит 6 – ICES1: Датчик фронта модуля захвата.

ICESn (Input Capture Edge Select). Если ICESn=0, то датчик фронта срабатывает при спадающем фронте импульса на входе. А если ICESn=1, то датчик фронта срабатывает при нарастающем фронте импульса.

- Бит 5 – Зарезервированный бит.
- Биты 4:3 – WGM13:2 : Режим работы Таймера/Счетчика1.
- Биты 2:0 – CS2:0 :Выбор источника тактирования.

Таблица 4. Функции битов WGMn3:0.

Режим	WGMn3	WGMn2	WGMn1	WGMn0	Режим работы	Макс. величина	Момент обновления регистра OCRnx	Момент установки флага переполнения TOVn
0	0	0	0	0	Нормальный	0xFFFF	Немедл.	0xFFFF
1	0	0	0	1	ШИМ с коррекцией фазы, 8 бит	0x00FF	Максим. величина	0x0000
2	0	0	1	0	ШИМ с коррекцией фазы, 9 бит	0x01FF	Максим. величина	0x0000
3	0	0	1	1	ШИМ с коррекцией фазы, 10 бит	0x03FF	Максим. величина	0x0000
4	0	1	0	0	CTC	OCRnA	Немедл.	0xFFFF
5	0	1	0	1	Быстродействующая ШИМ, 8 бит	0x00FF	Максим. величина	Максим. величина
6	0	1	1	0	Быстродействующая ШИМ, 9 бит	0x01FF	Максим. величина	Максим. величина
7	0	1	1	1	Быстродействующая ШИМ, 10 бит	0x03FF	Максим. величина	Максим. величина
8	1	0	0	0	ШИМ с коррекцией фазы и частоты	ICRn	0x0000	0x0000
9	1	0	0	1	ШИМ с коррекцией фазы и частоты	OCRnA	0x0000	0x0000
10	1	0	1	0	ШИМ с коррекцией фазы	ICRn	Максим. величина	0x0000
11	1	0	1	1	ШИМ с коррекцией фазы	OCRnA	Максим. величина	0x0000
12	1	1	0	0	CTC	ICRn	Немедл.	0xFFFF
13	1	1	0	1	(Зарезервировано)	–	–	–
14	1	1	1	0	Быстродействующая ШИМ	ICRn	Максим. величина	Максим. величина
15	1	1	1	1	Быстродействующая ШИМ	OCRnA	Максим. величина	Максим. величина

Таблица 5. Функции битов CS2:0.

CSn2	CSn1	CSn0	Описание
0	0	0	Источник тактирования не выбран. (Счетчик остановлен).
0	0	1	$clk_{I/O}/1$ (Не предварительного деления).
0	1	0	$clk_{I/O}/8$ (Тактовый сигнал с пред. делителя).
0	1	1	$clk_{I/O}/64$ (Тактовый сигнал с пред. делителя).
1	0	0	$clk_{I/O}/256$ (Тактовый сигнал с пред. делителя).
1	0	1	$clk_{I/O}/1024$ (Тактовый сигнал с пред. делителя).
1	1	0	Внешнее тактирование с вывода Tn. Такт по спадающему фронту сигнала.
1	1	1	Внешнее тактирование с вывода Tn. Такт по нарастающему фронту сигнала.

Регистр маски прерываний таймеров TIMSK.

Бит	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Начальное значение	0	0	0	0	0	0	0	0	

- Бит 5 – TICIE1: Разрешение прерывания по захвату.
- Биты 4 – OCIE1A: Разрешение прерывания по совпадению канала A.
- Биты 3 – OCIE1B: Разрешение прерывания по совпадению канала B.
- Бит 2 – TOIE1: Разрешение прерывания по переполнению.

Сторожевой таймер.

Сторожевой таймер (Watchdog Timer, WDT) – одно из самых полезных устройств в составе микроконтроллеров, потому, что он применяется на практике почти во всех разрабатываемых приложениях. Причем его полезность не всегда очевидна: в нормальном режиме работы, когда все настроено идеально, он, в общем-то, и вовсе не нужен.

Однако особенностью функционирования МК-систем управления является то, что они практически всегда работают в автономном режиме (без участия или контроля со стороны человека-оператора). Причем МК-системы зачастую формируют коды управляющих воздействий для исполнительных механизмов и ошибки, а тем более неконтролируемые действия могут стоить очень дорого, поэтому они должны быть исключены независимо от вариантов развития событий.

Например, вполне реальными представляются следующие ситуации: МК настроен на обработку некоторой совокупности данных, при этом возможно возникновение их сочетания, которое не предусмотрено программистом, что может привести к зацикливанию программы. Или на практике достаточно распространенным является случай сбоя программного кода при перепадах напряжения, что также может привести к непредвиденным ситуациям. Наиболее разумным выходом в данных и им подобных ситуациях является перезапуск программного кода (перезапуск контроллера), который должен быть выполнен без участия человека.

Перезапуск контроллера – выполнение полного сброса системы (переинициализации путем сброса и запуска с нуля).

Идентификация аварийного состояния (аварийного хода выполнения программы) может быть проведена путем оценки времени выполнения совокупности стандартных операций (основного цикла) МК-системы.

В принципе эту работу может выполнить один из стандартных таймеров счетчиков МК, однако здесь есть свои особенности. Во-первых стандартные Т/С необходимы для решения задач управления, а кроме того, это устройство должно тактироваться отдельным генератором, для того, чтобы иметь возможность управлять выходом из различных режимов энергосбережения.

Перечисленные выше задачи и решает WDT, который сбросит МК по истечении некоторого срока, если его вовремя не остановить.

Для предотвращения перехода микроконтроллера в режим бесконечного цикла, когда на него невозможно повлиять извне (такие ситуации возникают при ошибках в программах пользователя) компания Atmel оснастила все микроконтроллеры базовой серии семейства AVR так называемыми сторожевыми

таймерами – WDT.
Управление работой сторожевого таймера осуществляется с помощью регистра WDTCR и инструкции **WDR**.

Регистр управления сторожевым таймером WDTCR.

Бит	7	6	5	4	3	2	1	0	
	-	-	-	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Начальное значение	0	0	0	0	0	0	0	0	

- Бит 4 – WDTOE (Watchdog Turn Off Enable): Разрешение изменения режимов сторожевого таймера. Этот бит используется при изменении состояния битов предварительного делителя и бита WDE. Перед тем, как сбросить бит WDE и/или изменить состояние битов предварительного делителя, должен быть установлен разряд WDCE. После записи в него единицы бит WDTOE будет автоматически сброшен аппаратным путем по истечении четырех машинных циклов.
- Бит 3 – WDE (Watchdog Enable): Разрешение режима сброса. Флаг WDE дублирует флаг WDRF в регистре MCUSR. Это означает, что WDE всегда установлен, если установлен WDRF. Для того, чтобы очистить WDE, нужно сначала. очистить WDRF. Эта особенность гарантирует многократный сброс микроконтроллера до тех пор, пока существует условие, вызывающее зависание программы и безопасный запуск программы сразу после окончания этого условия.
- Биты 2..0 – WDP2:0: Выбор режима работы предварительного делителя охранного таймера. Биты WDP2:0 определяют коэффициент деления предварительного делителя сторожевого таймера. Все возможные коэффициенты деления и соответствующие им защитные периоды времени приведены в табл. 6.

Таблица 6. Настройка времени до наступления сторожевой задержки при рабочем напряжении Vcc=+5В.

WDP2	*WDP1*	*WDP0*	Время до наступления сторожевой задержки
0	0	0	около 16 мс
0	0	1	около 32 мс
0	1	0	около 64 мс
0	1	1	около 128 мс
1	0	0	около 256 мс
1	0	1	около 512 мс
1	1	0	около 1 с
1	1	1	около 2,1 с

Команда **WDR** сбрасывает сторожевой таймер. Если эта инструкция не будет выполнена в течении времени, установленного битами WDP2:0, то сторожевой таймер автоматически перезапустит микроконтроллер.

ПРАКТИЧЕСКАЯ ЧАСТЬ

В предыдущей работе, в которой изучались внешние прерывания, мы переключали светодиоды по очереди, нажимая на кнопку. Изменим эту программу таким образом, чтобы переключение происходило не при нажатии на кнопку, а по событию от таймера. Новая программа находится в проекте **L5_1.** Для удобства расчёта временных интервалов, МК должен быть настроен на тактирование от внутреннего источника частотой 1МГц. T/C1 настроен на прерывания по совпадению (CTC), таким образом, что прерывание происходит с частотой 1Гц. Частота, с которой будет происходить прерывание, вычисляется по формуле:

$$f_{OC1A} = \frac{f_{clk}}{2 \cdot N \cdot (1 + OCR1A)}$$, где

f_{clk} – рабочая тактовая частота МК

N – предделитель таймера

В следующей программе (**L5_2)** таймер 1 настроен на режим быстрой ШИМ. К выводам OC1A и OC1B подключены светодиоды LED1 и LED0. При каждом нажатии на кнопку SW0 светодиод LED1 постепенно гаснет, а LED0 – постепенно загорается. При нажатии на кнопку SW1 происходит обратный процесс.

Для изучения работы сторожевого таймера изменим программу **L5_1** таким образом, чтобы при нажатии на кнопку SW0 происходило прерывание, в котором выполняется бесконечный цикл, т.е. программа зависает. Причём, т.к. зависание происходит при обработке прерывания, то никакие другие прерывания больше не происходят, и это сразу видно по тому, что светодиоды перестают переключаться. Из этого состояния МК можно вывести, только перезагрузив его. Новая программа находится в проекте **L5_3**.

ЗАДАНИЕ

Изучить работу таймера/счетчика в различных режимах.

ПОРЯДОК ВЫПОЛНЕНИЯ

1. Подключите светодиоды LEDS к порту D целевого микроконтроллера.
2. Загрузите проект **L5_1** и изучите его работу.
3. Измените программу **L5_1** так, чтобы переключение светодиодов происходило в 2 раза быстрее.
4. Теперь подключите светодиоды LED0 и LED1 к контактам PB5 и PB6 порта B соответственно. А кнопки SW0 и SW1 – к контактам PB3 и PB4 порта B.

5. Загрузите проект **L5_2** и изучите его.
6. Отключите светодиоды и подключите к контактам PB5 и PB6 порта B осциллограф. Изменяя значения в регистрах OCR1A и OCR1B, посмотрите на осциллографе как меняется скважность сигнала.
7. Настройте таймер 1 на режим «ШИМ с коррекцией фазы» и посмотрите, как изменятся показания осциллографа.
8. Загрузите проект **L5_3** и изучите его. Загрузите программу в МК и убедитесь, что при нажатии на кнопку программа зависает, и светодиоды перестают переключаться.
9. Добавьте в программу инструкции, производящие настройку сторожевого таймера на интервал 1с и запускающие его. Установите инструкцию сброса сторожевого таймера WDR таким образом, чтобы программа нормально работала, а при нажатии на кнопку SW0 и последующем зависании происходила перезагрузка МК.