Государственное образовательное учреждение

высшего профессионального образования

«Московский Государственный Технический Университет им. Н.Э.Баумана»

**Методические указания к выполнению лабораторных работ**

**«Рассмотрение принципов работы микроконтроллеров при помощи стенда EASYAVR 5»**

по дисциплине «Электронные устройства мехатронных и робототехнических систем»

2016г.

**Содержание**

|  |  |
| --- | --- |
| Введение | 3 |
| Работа со стендом EasyAVR5 | 5 |
| Работа со стендом | 6 |
| Работа с программным обеспечением AVRStudio | 9 |
| Лабораторная работа №1. «Изучение структуры стенда EasyAVR 5 и САПР AVRStudio» | 16 |
| Лабораторная работа №2. «Система прерываний. Работа с таймером/счетчиком 0» | 24 |
| Лабораторная работа №3 «Работа с универсальным синхронно-асинхронным приемопередатчиком» | 37 |
| Лабораторная работа №4 «Работа с модулем АЦП» | 56 |

**Введение**

Стенд Easyavr5 представляет собой лабораторный макет с микроконтроллером. К каждому выводу микроконтроллера подключена кнопка и светодиод. На макете установлено четыре семисегментных индикатора. Также имеется возможность подключения к макету внешних устройств через специальные устройства. Питание макета осуществляется либо от внешнего источника напряжения 8–16В, либо через USB-порт. Кроме этого на макете установлен внутрисхемный программатор.

В лабораторных работах будет использоваться микроконтроллер фирмы ATMEL – ATmega16.

Условные обозначения:

* Flash ROM - объем энергонезависимой памяти программ (в килобайтах);
* EEPROM - объем энергонезависимой памяти данных (в байтах);
* RAM - объем статической памяти данных (в байтах);
* External RAM - возможность подключения к микроконтроллеру дополнительной микросхемы внешней статической памяти данных (в килобайтах);
* ISP - возможность программирования микроконтроллера в системе (на целевой плате) при основном напряжении питания;
* SPM - функция самопрограммирования Flash ROM памяти микроконтроллера в системе без участия внешнего программатора;
* JTAG - встроенный JTAG - интерфейс;
* I/O (pins) - максимальное количество доступных линий ввода / вывода;
* Timer(s) 8/16 bit - количество и разрядность таймеров/счетчиков;
* USI - универсальный коммуникационный интерфейс;
* AC - аналоговый компаратор;
* ADC (channels) - количество каналов аналого-цифрового преобразования;
* Internal RC - наличие внутренней RC-цепочки для автономной работы микроконтроллера (без внешнего источника опорной частоты);
* WDT - сторожевой таймер;
* BDC - аппаратный программируемый блок защиты от сбоев при внезапном (в том числе и кратковременном) пропадании напряжения питания микроконтроллера;
* UART - асинхронный последовательный приемопередатчик;
* SPI - синхронный трехпроводной последовательный интерфейс;
* I2C - двухпроводной последовательный интерфейс;
* RTC - система реального времени;
* PWM (channels) - количество независимых каналов широтно - импульсной модуляции;
* Command Set - количество различных инструкций в системе команд микроконтроллера;
* Vcc - диапазон рабочих напряжений питания (в Вольтах);
* Clock - диапазон рабочих частот (в мегагерцах);

**Блок-схема ATmega16**

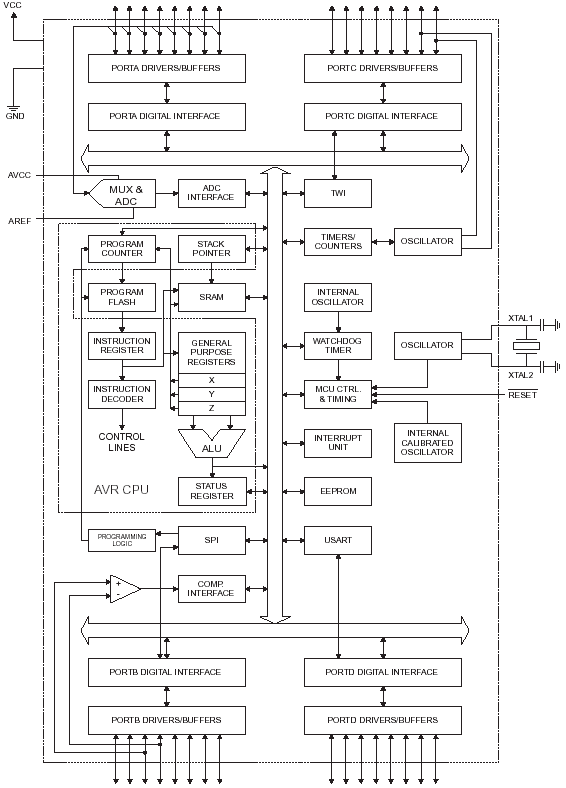
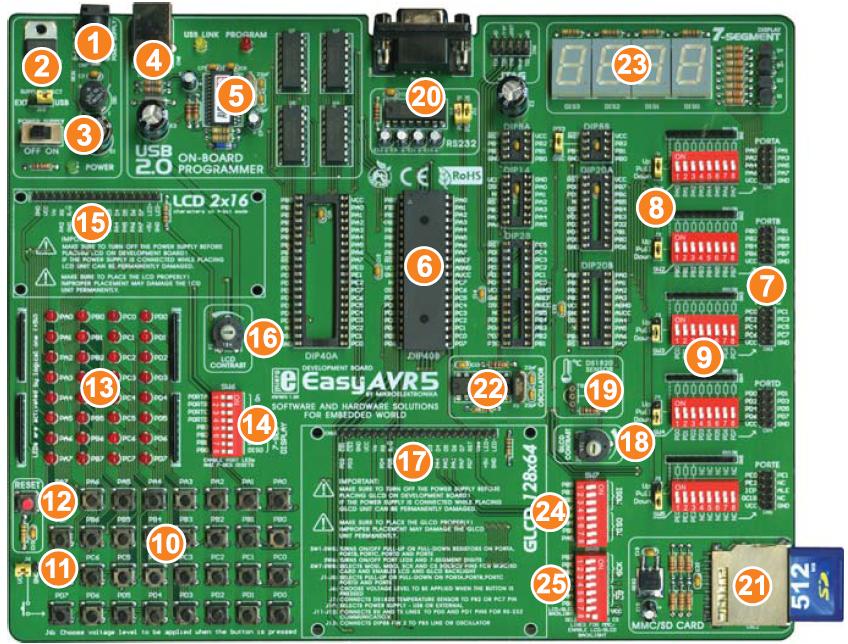


Рисунок 1 – Блок схема ATmega16

**Работа со стендом EasyAVR5**



1 – разъем для подключения внешнего питания 8–16В;

2 – выбор между внешним питанием и питанием от USB;

3 – выключатель;

4 – разъем USB;

5 – встроенный внутрисхемный программатор;

6 – панельки для установки различных типов МК;

7 – разъемы для доступа к портам МК;

8 – джамперы для выбора типа подтягивающих резисторов (к «1» или к «0»);

9 – группы переключателей для подключения подтягивающих резисторов к выводам МК;

10 – 32 кнопки, подключенные к выводам МК;

11 – джампер выбора высокого или низкого состояния при нажатии кнопки:

12 – кнопка сброса;

13 – светодиоды, каждый подключенный к выводу МК;

14 – группа переключателей, подключающая или отключающая светодиоды к МК;

15 – разъем для подключения символьной ЖК-матрицы;

16 – потенциометр для регулировки яркости символьной ЖК-матрицы;

17 – разъем для подключения графической ЖК-матрицы;

18 – потенциометр для регулировки яркости графической ЖК-матрицы;

19 – разъем для подключения датчика температуры DS1820;

20 – разъем порта RS232;

21 – слот для подключения MMC/SD карт памяти;

22 – встроенный генератор частоты;

23 – 4 семисегментных индикатора;

24 – группа переключателей для разрешения внутрисхемного программирования;

25 – группа переключателей для разрешения работы с MMC/SD картами памяти.

Рисунок 2 – Внешний вид стенда EasyAVR5

Внешний вид стенда представлен на рисунке 2. Данный стенд позволяет подключить к каждому выводу микроконтроллера светодиод, кнопку или подать на него любой внешний сигнал. На стенде предустановленны 4 семисегментных индикатора.

Схема подключения светодиодов к выводам МК представлена на рисунке 3.

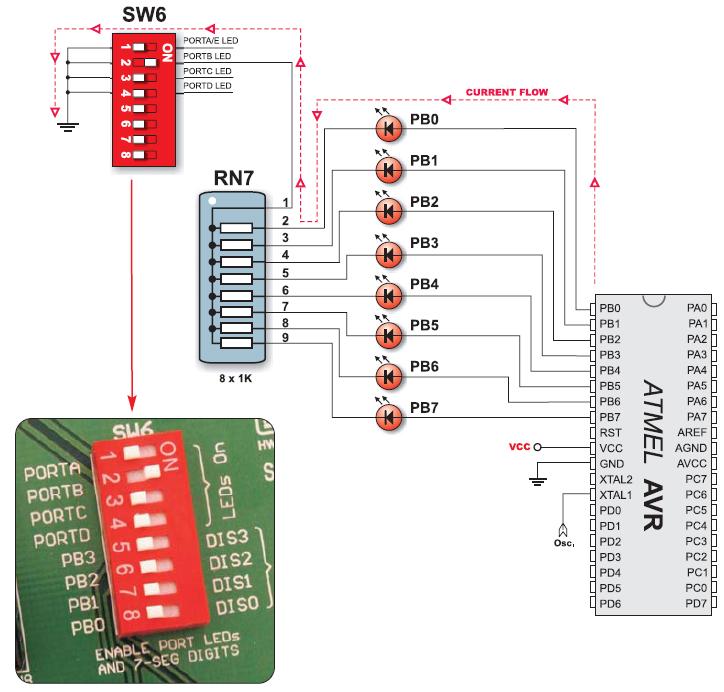


Рисунок 3 – Схема подключения светодиодов к выводам МК

Подключение светодиодов осуществляется группами одновременно к каждому из портов. Управление подключением осуществляется при помощи переключателя SW6.

Схема подключения кнопок к портам микроконтроллера представлена на рисунке 4.

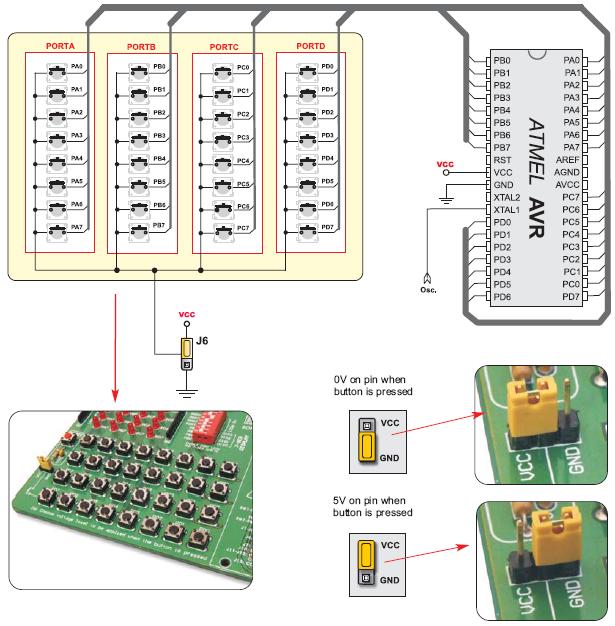


Рисунок 4 –Схема подключения кнопок к выводам МК

К каждому выводу микроконтроллера подключена кнопка. При нажатии кнопки на вывод микроконтроллера может быть сигнал либо нулевого, либо единичного уровня. Выбор уровня осуществляется при помощи джампера J6.

Схема подключения семисегментных индикаторов к портам микроконтроллера представлена рисунке 5.

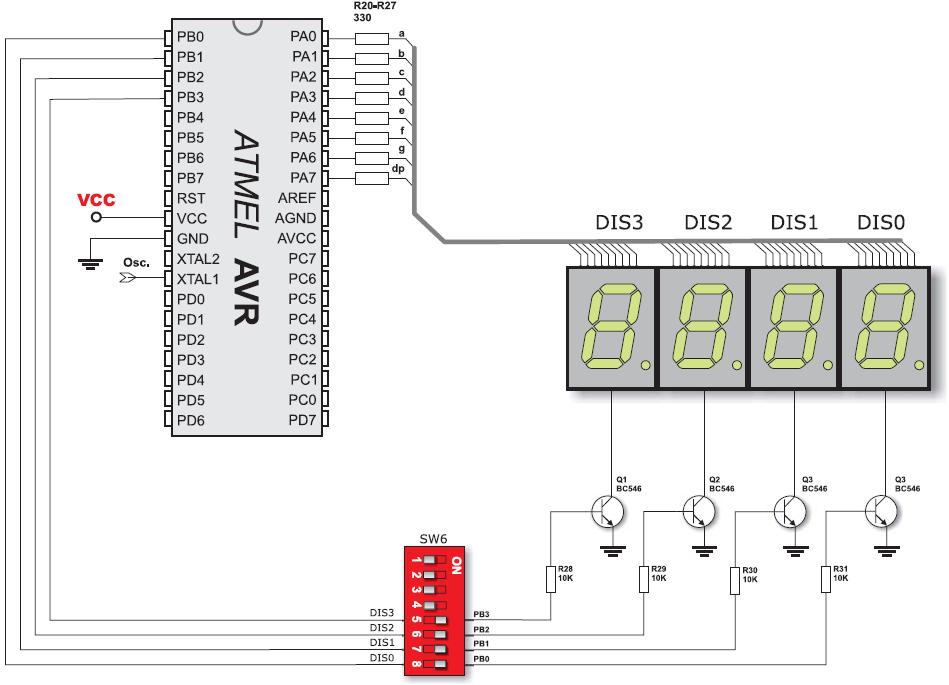


Рисунок 5 – Схема подключения семисегментных индикаторов к портам МК

В зависимости от типа макета семисегментные индикаторы подключаются либо к порту А, либо к порту С. Управление их работой происходит при помощи выводов порта В. Разрешение управления проводится группой переключателей SW6.

Загрузка исполняемой программы в микроконтроллер стенда проводится при помощи программы AVRFLASH Programmer. Перед программированием необходимо подключить стенд к USB-порту компьютера. Главное окно программы представлено на рисунке 6.

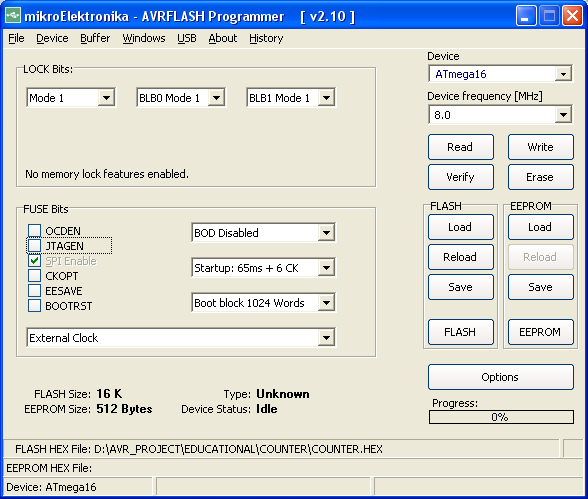


Рисунок 6 – Главное окно программы AVRFLASH Programmer

Перед программирование в правом внешнем углу окна необходимо выбрать тип микроконтроллера (ATMega16) и рабочую частоту устройства (8 МГц). Па панели FUSE Bits необходимо выставить параметры, которые указаны на рисунке. На панели FLASH при помощи кнопки Load выбирается hex-файл, который будет загружен в память программ. Далее производится запись, для чего необходимо нажать кнопку Write. Перед программированием обязательно провести стирание устройства при помощи кнопки Erase.

**Работа с программным обеспечением AVRStudio**

После запуска программы открывается мастер нового проекта, также создать новый проект можно через меню выбрать пункт Project, а в нем New project (рисунок 7).

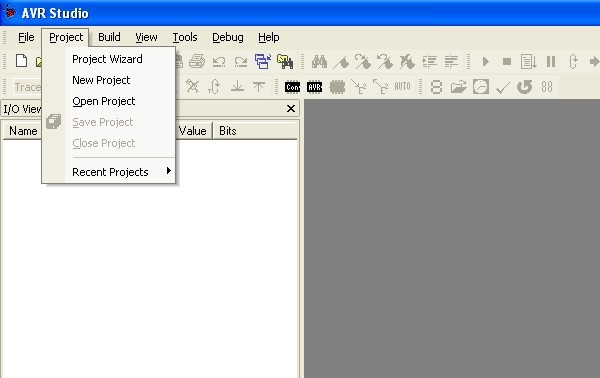


Рисунок 7 – Создание нового проекта в AVRStudio

В открывшемся окне следует выбрать тип проекта - Atmel AVR Assembler (рисунок 8).

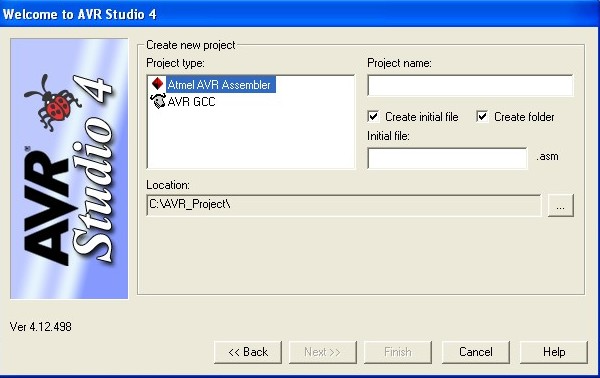


Рисунок 8 – Выбор типа проекта

Поставьте галочку «Create folder» - создать папку, для того, чтоб разместить файлы проекта в отдельной директории. Если поставить галочку «Create initial file» - будет создан начальный файл. Введите имя начального файла. Назнаьчте рабочую папку - место размещения проекта. Затем перейти далее >> на форму где необходимо определится с типом используемого отладочного средства (рисунок 9).

**Внимание: В пути файла проекта не должно быть русских символов.**

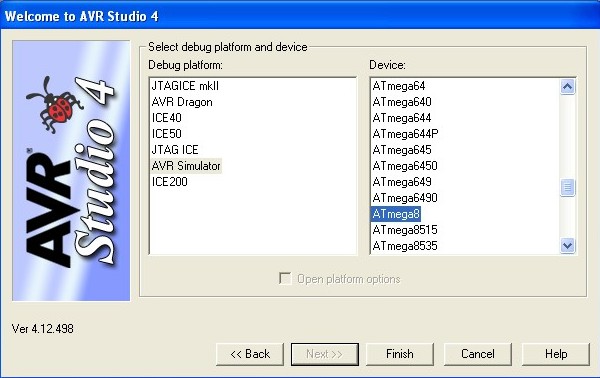


Рисунок 9 – Выбор отладочного средства и МК, для которого создается программа

Если предполагается воспользоваться встроенным симулятором следует выбрать пункт AVR Simulator и выбрать используемый тип микроконтроллера. После произведенных действий откроется окно программного кода (рисунок 10).

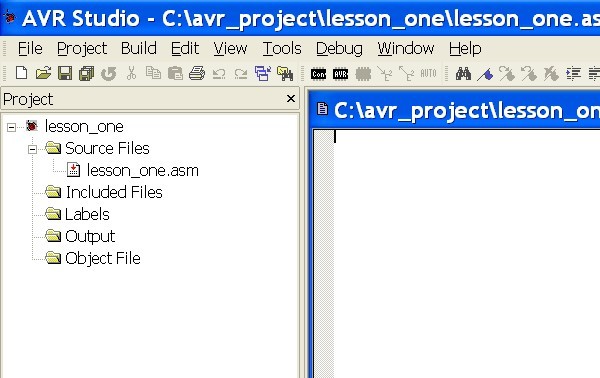


Рисунок 10 – Окно программного кода

В этом окне набирается и корректируется текст программы. Помимо окна с кодом программы, в AVR Studio существует еще несколько полезных окон: Project, I/O View и Info, служащих для отображения состава проекта, состояния регистров и портов микроконтроллера с возможностью управления состоянием регистров и портов и дополнительной справочной информации о микроконтроллере. Подробнее об этих окнах сказано ниже.

*Ассемблирование программы*

После того как создан текст программы, начинается отладка. Для того что бы запустить программу на выполнение, необходимо сначала - собрать проект, выберите пункт Build в открывшемся меню снова выберите Build (рисунок 11).

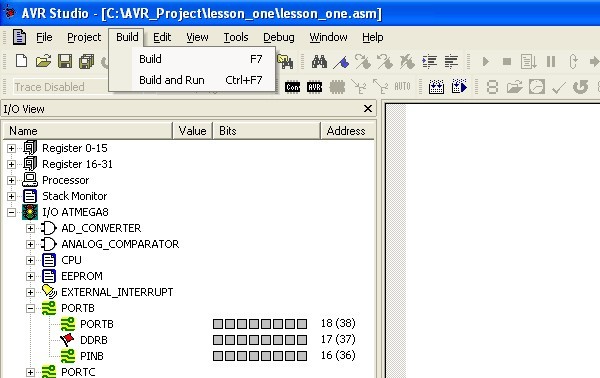


Рисунок 11 – Ассемблирование программы

В AVR Studio для вывода сообщений о прохождении ассемблирования существует окно Build (рисунок 12).

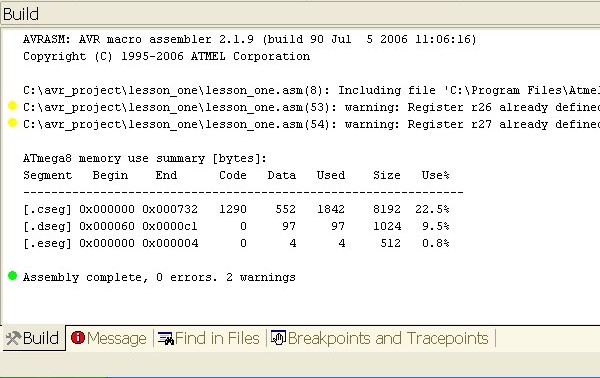


Рисунок 12 – Окно вывода результатов ассемблирования

После ассемблирования в окне Build появляется сообщение: Assembly complete, хх errors. хх warnings. Если ассемблирование закончилось с ошибками в окне Build появится список этих ошибок, перейти на строку содержащую ошибку можно кликнув на на строке с этим сообщением в окне.

*Отладка программы на ассемблере. Выполнение программ в пошаговом режиме. Просмотр регистров*

После успешного ассемблирования можно запустить программу на выполнение в симуляторе нажатием кнопки Start debugging (рисунок 13).

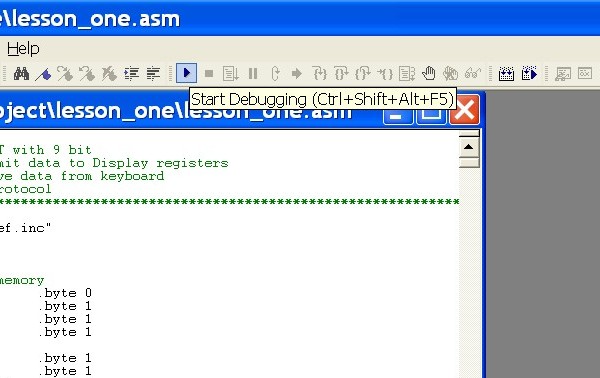


Рисунок 13 – Запуск симулятора

При этом в окне исходного кода программы появится маркер, который находится на строке, которая будет выполнена в следующем шаге (рисунок 14).

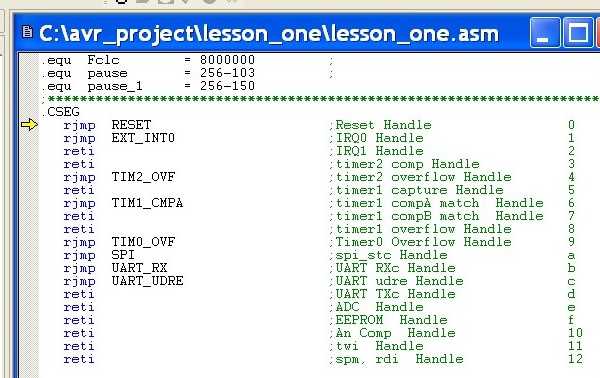


Рисунок 14 – Начало выполнения программы в симуляторе

С помощью средств управления ходом выполнения программы, можно выполнять программу в пошаговом режиме, или выполнить программу до места, где стоит курсор (рисунок 15):

- стрелка Play – запуск симуляции;

- квадратик Stop – остановка симуляции;

- листок со стрелочкой – запуск программы;

- пауза – остановка программы в текущем положении;

- синяя стрелка – виртуальный Reset контроллера;

- желтая стрелка – переход к строке, где программа находится в текущий момент;

- Step Into – сделать шаг по тексту программы;

- Step Over – сделать шаг по тексту, но не заходить в процедуры и функции (процедуры и функции при этом выполняются);

- Step-Out – Шаг “назад”. Используется если в програмее имеется цикл, при нажатии МК сделает одну итерацию цикла и встанет над инструкцией;

- фигурные скобки и стрелка – выполнить программу до текущего положения курсора.

- красная точка – BreakPoint поставить точку останова;

- крестик и две красных точки – удачение всех точек останова из проекта.



Рисунок 15 – Средства управления хода выполнения программы

Точка останова устанавливается либо двойным щелчком мыши по строчке, где необходимо ее поставить, либо нажатием красной точки (см. рисунок 15). При достижении точек останова выполнение программы приостанавливается. Точки останова сохраняются между сессиями работы.

Если во время выполнения программы в окне исходного кода поместить курсор на имя регистра то появится подсказка о значении этого регистра, его расположении в памяти (рисунок 16).

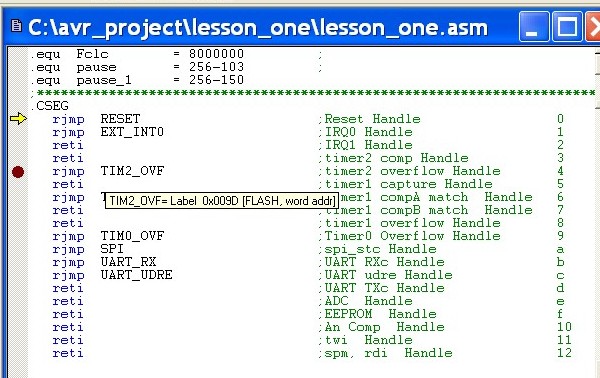


Рисунок 16 – Подсказка о значении регистра и его местоположении

Для облегчения отладки программного обеспечения можно использовать следующие окна.

*Watch window.* Окно показывает значения и адреса заданных переменных (рисунок 17).

Переменные можно добавлять вручную, либо при помощи меню правой кнопки мыши. Значения переменных можно добавлять и изменять во время остановки работы программы.

*Register window*. Окно показывает содержимое регистров (рисунок 18). Регистры можно изменять во время остановки программы.

Memory windows. Окна показывают содержимое памяти программ, данных, портов ввода/вывода и энергонезависимого ПЗУ (рисунок 19). Память можно просматривать в HEX, двоичном или десятичном форматах. Содержимое памяти можно изменять во время остановки программы.

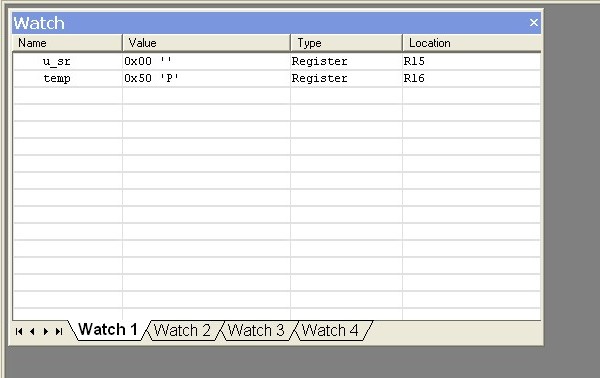


Рисунок 17 – Watch window

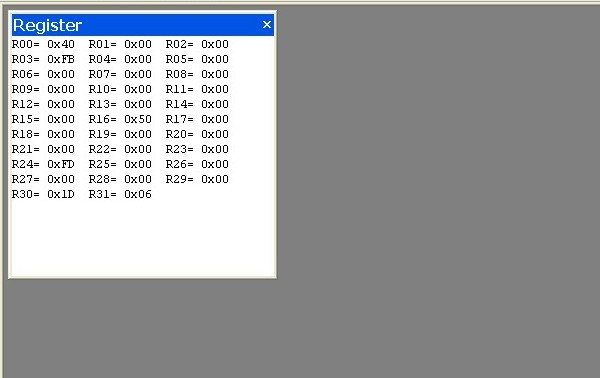


Рисунок 18 – Register window

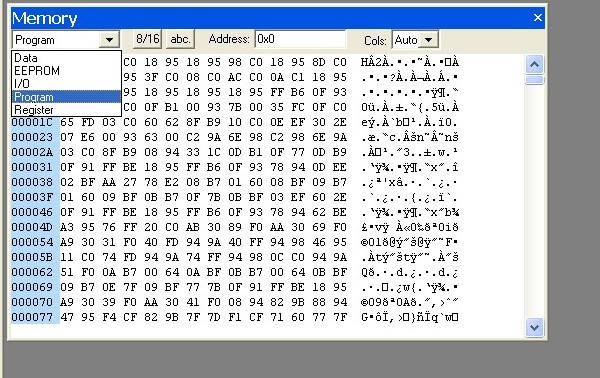


Рисунок 19 – Memory window

*I/O window.* Показывает содержимое различных регистров ввода/вывода: EEPROM, USART, таймеры и др (рисунок 20). Набор регистров ввода/вывода зависит от выбранного МК.

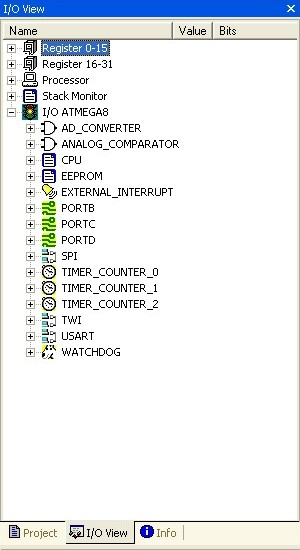


Рисунок 20 – I/O window

*Project window*. Окно показывает название файлов входящих в проект.

*Info window*. Полезная информация о выбранном микроконтроллере - о назначении выводов, источниках прерываний, адресах регистров и др.

*Message window*. Окно показывает сообщения от AVR Studio.

**Лабораторная работа №3**

**Работа с универсальным синхронно-асинхронным приемопередатчиком**

**1 Цель работы**

Изучить работу универсального синхронно-асинхронного приемопередатчика. Изучить принципы сопряжения микроконтроллерных систем с ПК.

**2 Краткие теоретические сведения**

2.1 Модуль универсального синхронно-асинхронного приемопередатчика

Через модуль USART микроконтроллер осуществляет прием и передачу информации, представленной последовательным кодом, поэтому его часто называют также последовательным портом. С помощью этого модуля микроконтроллер может обмениваться данными с различными внешними устройствами. Блок схема модуля USART представлена на рисунке 6.

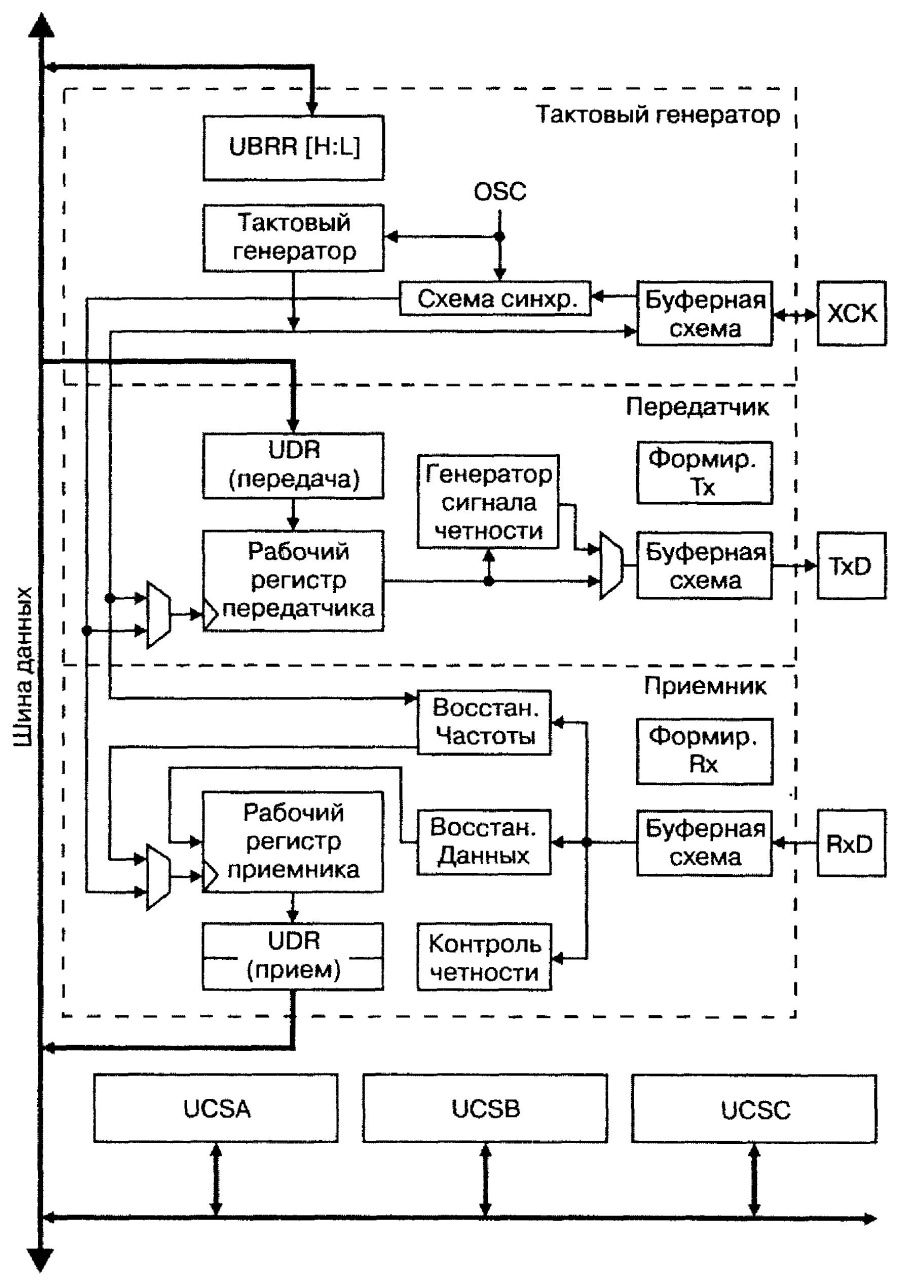


Рисунок 6 – USART

Скорость передачи данных может варьироваться в широких пределах, причем высокие скорости передачи могут быть достигнуты даже при относительно низкой тактовой частоте микроконтроллера.

Для взаимодействия с программой в модуле предусмотрены 3 раздельных прерывания, запрос на которые генерируется при наступлении следующих событий: «передача завершена», «регистр данных передатчика пуст» и «прием завершен».

2.2 Интерфейс RS232 (COM-порт)

Интерфейс RS-232C предназначен для подключения к компьютеру стандартных внешних устройств (принтера, сканера, модема, мыши и др.), а также для связи компьютеров между собой. Данные в RS-232C передаются в последовательном коде побайтно. Каждый байт обрамляется стартовым и стоповыми битами. Данные могут передаваться как в одну, так и в другую сторону (дуплексный режим).

Компьютер имеет 25-контактный (DB25P) или 9-контактный (DB9P) разъем для подключения RS-232C. Скорость передачи по RS-232C может выбираться из ряда: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 бит/с.

Все сигналы RS-232C передаются специально выбранными уровнями, обеспечивающими высокую помехоустойчивость связи (рисунок 7.). Отметим, что данные передаются в инверсном коде (логической единице соответствует низкий уровень, логическому нулю - высокий уровень).

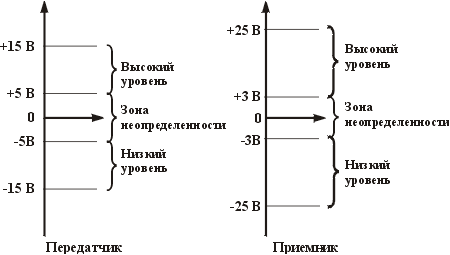


Рисунок 7 – Уровни сигналов RS-232C на передающем и принимающем концах линии связи

Для подключения произвольного УС к компьютеру через RS-232C обычно используют трех- или четырехпроводную линию связи, но можно задействовать и другие сигналы интерфейса. Для преобразования уровней сигналов с выхода модуля USART микроконтроллера к уровням интерфейса RS232 используются специальные микросхемы преобразования уровней (например, MAX232).

2.3 Протокол обмена информацией при последовательной передаче данных

При передаче данных необходимо, чтобы имелся протокол обмена, согласно которому участвующие в обмене устройства будут интерпретировать данные. Кроме этого необходимо оговорить количество информационных бит в кадре данных, количество стоп-бит и т.д. В предлагаемой лабораторной работе принят следующий протокол обмена. Передача/прием данных осуществляется со следующими параметрами:

а) скорость обмена 19200/38400 бит/сек;

б) формат посылки 10/11 бит информации:

- старт-бит;

8 бит данных;

- бит контроля четности (может отсутствовать);

- один стоп бит.

Порядок обмена информацией.

С персонального компьютера, выполняющего роль ведущего в обмене (master), осуществляет передача кадра запроса выполнения простейшей арифметической операции. Кадр состоит из следующей последовательности:

1) Байт кода операции:

- 0x01 – сложение 2-х однобайтных двоичных чисел без знака (0 до 255);

- 0х02 – вычитание 2-х однобайтных двоичных чисел без знака (0 до 255);

- 0х03 – умножение 2-х однобайтных двоичных чисел без знака (0 до 255);

- 0х04 – деление 2-х однобайтных двоичных чисел без знака (0 до 255).

2) Байт первого операнда (1-ое слагаемое, уменьшаемое, множимое, делимое).

3) Байт второго операнда (2-ое слагаемое, вычитаемое, множитель, делитель).

4) Байт контрольной суммы с инверсией.

Микроконтроллер, ведомый (slave) принимает запрос на выполнение операции, осуществляет выполнение арифметической операции и возвращает кадр ответа выполненной операции. Кадр ответа состоит из следующей последовательности:

1) Старшего байта результата выполнения операции/(частное);

2) Младший байт результата выполнения операции/(остаток от деления);

3) Байт контрольной суммы с инверсией.

Передача запроса на вычисление и выдача ответа осуществляются асинхронно по нажатию соответствующих кнопок на экране монитора персонального компьютера и макетной платы.

**3 Практическое задание**

Лабораторная работа выполняется в следующей последовательности.

3.1 Создайте новый проект, который сохраните в отдельной папке (в пути файла проекта не должно быть русских символов).

3.2 Вставьте в рабочую область AVRStudio текст программы из файла contr\_UART.asm.

Начальный блок программы имеет следующий вид.:

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*;\* Designer Mechtcherjakowa R.I.*

*;\* Version: 1.0*

*;\* Date 15.03.2010*

*;\* Title: contr\_UART.asm*

*;\* Device ATmega16*

*;\* Clock frequency:Частота кв.резонатора 9,216 mHz*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; учебная*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*;Программа осуществляет обмен по интерфейсу RS-232 между персональным компьютером*

*;и макетной платой. В макетной плате устанавливается микроконтроллер ATmega16*

*;с программируемым последовательным портом USART. Преобразование в интерфейс*

*;RS-232 выполняется микросхемой Max232, установленной в макетной плате.*

*;Передача/прием данных по USART осуществляется со следующими параметрами:*

*;а)скорость обмена 19200бит/сек;*

*;б)формат посылки 10 бит информации:*

*; - старт-бит;*

*; - 8 бит данных;*

*; - один стоп бит.*

*;Выводы USART(RXD) PD0 (вход),(TXD) PD1(выход)*

*;При приеме кадра запроса загорается светодиод СИД "Запрос", подключенный*

*;к PB7 ( выход)*

*;Кадры принятых и подготовленных к ответу данных можно просмотреть на*

*;семисегмент.индикаторах последовательно байт за байтом,нажимая кнопку*

*;"Просмотр",подключенную к PD4( байт1, байт2, байт3, байт4,байт5,байт6,байт7,байт1..).*

*;Крайний левый индикатор показываетномер номер просматриваемого байта*

*;Кнопка "Ответ", инициализирующая ответную посылку микроконтр.подключена к PD7,при этом*

*;СИД гаснет*

*;семисегментные индикаторы подключены к PC0-PC7-выходы,семигегментный индикатор к PA0-PA7*

*;* *PC0-a,PC1-b,PC2-c,PC3-d,PC4-e,PC5-f,PC6-g,PC7-h;сигналы выбора индикатора PB3-номер байта,PB2 - сотни,PB1-десятки,PB0 - единицы- выходы*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*.include "m16def.inc"*

*; присоединение файла описаний;*

*.list ;включение листинга*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; Register Variables*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*.def temp\_L =R16*

*.def temp\_H =R17*

*;*

*.def Number =R18*

*.def Hundreds =R19;*

*.def Tens =R20*

*.def Ones =R21*

*;*

*.def Disp\_Numb =R22*

*.def Disp\_Count =R23*

*;*

*.def res\_a\_op =R8;*

*.def Time =R25;счетчик переполн.Т0(1024\*255/9216000\*Xotc=1cek(Xotc=35)*

*;*

*.def Cou\_Rec =R4;счетчик принятых байт*

*.def Cou\_Tran =R5;счетчик переданных байт*

*.def c\_sumREC =R6;контр сумма прин. байт*

*.def c\_sumTRAN =R7;контр сумма переданных байт*

*.def n\_ar\_op =R24*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*.def Byte\_fl =R2; байт флагов*

*;-------------------*

*.equ Disab\_Key =0; бит запрещения опроса кнопки "Просмотр"*

*.equ F\_recive =1;флаг принятого запроса*

*.equ F\_trans =2;флаг завершения передачи*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; Constants*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*.equ Val\_dispCount =50;величина константы,опр. время вкл индикатора*

*.equ CID =2;PD2 подключение СИД*

*.equ ENV =7;Кнопка "Ответ", подключена к PD7*

*;========================================*

*;Количество байт обмена с ПЭВМ*

*;----------------------------------------*

*.equ VAL\_TR =3*

*.equ VAL\_REC =4*

*.equ VAL\_time =30;30 переполнений соответствуют 1 сек*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*;Variable*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*.DSEG*

*;*

*varBuf\_Rxd: .BYTE 8;буфер приема(4 байта)*

*;*

*varBuf\_Txd: .BYTE 8;буфер передачи (3 байта)*

*;*

*varBuf\_disp: .BYTE 8;буфер данных,выводимых на дисплей (7 байт)*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*.cseg*

*.org $0000*

*rjmp Init*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*.org INT0addr;=$002 ;External Interrupt0 Vector Address*

*reti;*

*.*

*.org OC1Aaddr;=$00C ;Output Compare1A Interrupt Vector Address*

*rjmp Time\_OUT*

*.*

*.org OVF0addr;=$012 ;Overflow0 Interrupt Vector Address*

*rjmp time\_d\_k ;*

*.*

*.org URXCaddr;=$016 ;UART Receive Complete Interrupt Vector Address*

*rjmp REC\_date*

*.org UDREaddr;=$018 ;UART Data Register Empty Interrupt Vector Address*

*rjmp B\_TRANS*

*.org UTXCaddr; =$01A ;UART Transmit Complete Interrupt Vector Address*

*rjmp TRANdate*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

В данной программе используется шестнадцать регистров РОН и им присвоены следующие имена, определяющие их функциональное назначение:

- *temp\_L, temp\_H* – два регистра для хранения временных данных, используются для промежуточного хранения данных при обращении к регистрам РВВ и при выполнении арифметических операций;

- *Number* – регистр, используемый для хранения адреса символа по таблице перекодировки, отображаемого на семисегментном индикаторе «номер байта»;

- *Handreds* – регистр, используемый для хранения адреса символа по таблице перекодировки, отображаемого на семисегментном индикаторе «сотни»;

- *Tens* – регистр, используемый для хранения адреса символа по таблице перекодировки, отображаемого на семисегментном индикаторе «десятки»;

- *Ones* – регистр, используемый для хранения адреса символа по таблице перекодировки, отображаемого на семисегментном индикаторе «единицы»;

- *Disp\_Numb* – регистр, используемый для хранения номера активного семисегментного индикатора; используется при динамической индикации;

- *Disp\_Count* – регистр, используемый в качестве счетчика времени активности каждого из семисегментных индикаторов при динамической индикации;

- *res\_a\_op* – буфер хранения промежуточных результатов при выполнении арифметических операций;

- *Time* – регистр, используемый в качестве счетчика числа переполнений таймера Т0 при формировании защитной паузы для подавления дребезга контактов;

- *Cou\_Rec* – регистр, используемый в качестве счетчика числа принятых байт модулем USART;

- *Cou\_Tran* – регистр, используемый в качестве счетчика переданных байт модулем USART;

- *c\_sumREC* – регистр, используемый для накопления контрольной суммы принятых модулем USART байт;

- *с\_sumTRAN* – регистр, используемый для накопления контрольной суммы передаваемых модулем USART байт;

- *n\_ar\_op* – регистр, используемый в качестве счетчика числа циклов суммирования при выполнении операции умножения;

- *Byte\_fl* – регистр, используемый для хранения различных флагов, сигнализирующих о различных событиях в программе;

Далее в программе определены следующие константы:

- *Disab\_Key = 0* – номер бита в регистре *Byte\_fl*, используемый как флаг, запрещающий опрос кнопки «Просмотр»;

- *F\_receive =1* – номер бита в регистре *Byte\_fl*, используемый как флаг, сигнализирующий о принятом кадре данных модулем USART;

- *F\_trans =2*– номер бита в регистре *Byte\_fl*, используемый как флаг, сигнализирующий о завершении передачи данных модулем USART;

- *Val\_dispCount = 50* – константа, определяющая время активности каждого из семисегментных индикаторов;

- *CID = 7* – константа, определяющая номер бита порта B, который управляет светодиодом СИД;

- *ENV = 7* – константа, определяющая номер бита порта D, на который поступает сигнал от кнопки «Ответ»;

- *VAL\_TR =3* – константа, определяющая количество бит к кадре, передаваемом модулем USART;

- *VAL\_REC =4* – константа, определяющая количество бит к кадре, принимаемом модулем USART;

- *VAL\_time =35* – константа, определяющая количество переполнений таймера Т0 для формирования защитной паузы при подавлении дребезга контактов;

Далее следует директива ассемблера DSEG, которая определяет начало сегмента данных. Исходный файл может состоять из нескольких сегментов данных, которые объединяются в один сегмент при компиляции. Сегмент данных обычно состоит только из директив BYTE и меток. Сегменты данных имеют свои собственные побайтные счётчики положения. Директива не имеет параметров. В ОЗУ выделяется место для трех массивов данных при помощи директивы BYTE. Если Вы хотите иметь возможность ссылаться на выделенную область памяти, то директива BYTE должна быть предварена меткой. Директива принимает один обязательный параметр, который указывает количество выделяемых байт. Эта директива может использоваться только в сегменте данных. Выделенные байты не инициализируются. Таким образом выделяет три массива данных по восемь бит каждый (хотя полностью из них не используется ни один):

- *varBuf\_Rxd* – буфер для хранения принятых модулем USART данных;

- *varBuf\_Txd* – буфер для хранения передаваемых модулем USART данных;

- *varBuf\_disp* – буфер для хранения данных, выводимых на семисегментные индиваторы.

В данной программе используется пять прерываний: прерывание по совпадению канала А таймера Т1 OC1A, прерывание по переполнению таймера Т0 OVF0, прерывание по окончанию приема модуля USART URXC, прерывание при опустошении буфера данных модуля USART UDRE и прерывание по окончанию передачи модуля USART. Поэтому в таблице векторов прерываний по этим адресам расположены команды безусловного перехода на подпрограммы обработки этих прерываний: *Time\_OUT*, *time\_d\_k*, *REC\_date*, *B\_TRANS*, *TRANdate*.

Следующая часть программы является подпрограммой инициализации МК:

*; Start Of Main Program*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*Init:*

*ldi temp\_L,LOW(RAMEND);выбор вершины стека*

*out SPL, temp\_L;Указатель стека*

*ldi temp\_L,HIGH(RAMEND)*

*out SPH,temp\_L*

*;*

*; ------Инициализация портов В/B*

*;*

*ldi temp\_L,0b11111111;(PB1-PB7)-выходы,*

*out DDRB,temp\_L*

*ldi temp\_L,0b00001000;выбраны PB3-номер байта (1-й индикатор)*

*out PORTB,temp\_L*

*;*

*ldi temp\_L,0b00000110;PD1,PD2-выходы(Txd,СИД),(PD0,PD3-PD7)-входы,*

*out DDRD,temp\_L*

*ldi temp\_L,0b10111011;Вкл подтяжка на кнопках PD3-PD5. PD2-CИД выключен*

*out PORTD,temp\_L;*

*;*

*ldi temp\_L,0b11111111;;(PB1-PB7)-выходы*

*out DDRC,temp\_L*

*;*

*;INIT USART*

*ldi temp\_L,29;(Частота кв. 9,216 мГц,скорость обмена 19200),U2X0=0,*

*ldi temp\_H,00*

*out UBRRL,temp\_L;*

*out UBRRH,temp\_H*

*ldi temp\_L,(1<<RXEN)|(1<<RXCIE);UCSZ2=0,UCSZ1=1,UCSZ0=1 - 8 bit*

*out UCSRB,temp\_L*

*ldi temp\_L,(1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1);- 1 stop bit,*

*out UCSRC,temp\_L;*

*; Инициализация таймера TCNT1*

*ldi temp\_L,00;compare A,(COM1A1,COM1A0=00) OC1A disconnect*

*out TCCR1A,temp\_L;*

*ldi temp\_L,(1<<WGM12);WGM13=0,WGM12=1,WGM11=0,WGM10=0,режим CTC*

*; No clock source,CS42,CS41,CS40=000*

*out TCCR1B,temp\_L;(No prescaling CS10=1*

*ldi temp\_H,0xD8 ; time\_out 6mcek - 0,006\*9216000=55296(D800)*

*ldi temp\_L,0x00 ;*

*out OCR1AH,temp\_H*

*out OCR1AL,temp\_L*

*;*

*ldi Disp\_Count,Val\_dispCount*

*clr Disp\_Numb*

*clr Byte\_fl*

*clr Cou\_Rec*

*clr Cou\_Tran*

*clr c\_sumREC*

*clr Time*

*ldi Number,1*

*;Обнуление SRAM*

*ldi YL,low(varBuf\_disp) ; Y register low буфер данных,выводимых на дисплей*

*ldi YH,high(varBuf\_disp) ; Y register high буфер данных,выводимых на дисплей*

*ldi temp\_L,0x00;начальная установка буфера данных,выводимых на дисплей*

*ldi temp\_H,* *VAL\_TR + VAL\_REC*

*LoadBdsp: st Y+,temp\_L*

*dec temp\_H*

*cpi temp\_H,0x00*

*brne LoadBdsp*

*;*

*ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема*

*ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема*

*ldi temp\_L,0x00*

*ldi temp\_H,VAL\_REC*

*ld\_b\_r: st Y+,temp\_L;начальная установка буфер приема*

*dec temp\_h*

*cpi temp\_H,0x00*

*brne ld\_b\_r*

*;*

*ldi YL,low(varBuf\_Txd); Load Y register low буфер передачи*

*ldi YH,high(varBuf\_Txd) ; Load Y register high буфер передачи*

*ldi temp\_L,0x00*

*ldi temp\_H,VAL\_TR*

*ld\_b\_t: st Y+,temp\_L;начальная установка буфер передачи*

*dec temp\_h*

*cpi temp\_H,0x00*

*brne ld\_b\_t*

*;*

*ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема*

*ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема*

*;*

*sei ;разрешаем прерывания*

*clr Hundreds*

*clr Tens*

*clr Ones*

*;==================================================*

В данной подпрограмме инициализируют необходимые регистры, включаются и конфигурируются используемые периферийные модули, осуществляется инициализация программного стека.

Порт C инициализируется как выходы для управления сегментами семисегментных индикаторов. Порт В конфигурируется как выходы для последовательно включения семисегментных индикаторов при динамической индикации. Часть выводов порта D конфигурируются как выходы для управления светодиодом «СИД», а часть – как входы для приема команд от кнопок «Ответ» и «Просмотр».

У модуля USART устанавливается скорость передачи равная 19200 бод, а также выставляются режимы работы: асинхронный, 8 бит данных и один стоп-бит.

У таймера Т1 разрешается прерывание по совпадению канала А.

Сбрасываются необходимы регистры, а также буферы в области данных.

В следующем сегменте кода приведена основная программа:

*;начало цикла программы*

*;==================================================*

*Start: rcall Display*

*sbrc Byte\_fl,F\_receive;проверка флага принятого запроса*

*rjmp Receive*

*sbrc Byte\_fl,Disab\_Key;*

*rjmp Start*

*sbic PinD,4;Проверка нажатия кнопки*

*rjmp Start*

*rcall ch\_pos\_D*

*rjmp Start*

*Receive:*

*sbi PORTD,CID*

*rcall Wr\_D\_rec*

*rcall arifm\_op*

*rcall pre\_date*

*rcall Wr\_D\_tr*

*Wait: sbis PIND,ENV*

*rjmp Trans*

*rcall Display*

*sbrc Byte\_fl,Disab\_Key;*

*rjmp Wait*

*sbic PinD,4;Проверка нажатия кнопки*

*rjmp Wait*

*rcall ch\_pos\_D*

*rjmp wait*

*Trans: ldi temp\_L,(1<<TXEN)|(1<<UDRIE);UCSZ2=0,UCSZ1=1,UCSZ0=1 - 8 bit*

*out UCSRB,temp\_L;переход к передаче через вызов прерывания UDRE*

*Wait\_tr: sbrs Byte\_fl,F\_trans*

*rjmp Wait\_tr*

*clt*

*bld Byte\_fl,F\_trans*

*rjmp Start*

*;==================================================*

*; конец цикла программы*

*;==================================================*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

В основном цикле программы вначале вызывается подпрограмма управления динамической индикацией *Display*. После этого проверяется состояние флага *F\_receive*, который показывает принят ли кадр данных модулем USART. Если данный флаг установлен, то МК переходит на строку с меткой *Receive*. Далее проверяется состояние флага *Disab\_Key*, который активен в момент формирования защитного интервала при подавлении дребезга контактов от кнопки «Просмотр». Если флаг установлен, то запрещается отрос кнопки «Просмотр», если сброшен – то эта кнопка опрашивается. Если кнопка нажата, то вызывается подпрограмма смены текущего отображаемого байта *ch\_pos\_D*. После этого программа переходит в начальное состояние.

После строки с меткой *Receive* расположены команды, которые выполняют следующие функции. Вначале включается светодиод СИД. Далее последовательно вызываются подпрограмма записи принятых данных в буфер для отображения на семисегментных индикаторах *Wr\_D\_rec*, подпрограмма выполнения арифметической операции *arifm\_op*, подпрограмма подготовки данных для передачи модулем USART *pre\_date*, подпрограмма записи передаваемых данных в буфер для отображения на семисегментных индикаторах *Wr\_D\_rec*.

Далее следует группа команд, которые проверяют состояние кнопок «Ответ» и «Просмотр». Если нажата кнопка «Просмотр», то вызывается подпрограмма *ch\_pos\_D*, которая осуществляет смену текущего отображаемого байта. Если нажата кнопка «Ответ», то МК начинает передачу данных при помощи модуля USART, переходя к строке с меткой *Trans*. При этом разрешается прерывание по опустошению буфера передачи, что приводит к вызову соответствующей подпрограммы обработки, т.к. флаг прерывания уже будет установлен. МК находится в этой части программы пока не будет установлен флаг *F\_trans*, информирующий о том, что все данные переданы. После этого МК переходит в начало программы.

Далее следует подпрограмма изменения текущего просматриваемого на семисегментных индикаторах байта:

*;Подпрограмма изменения позиции просмотра на дислее*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*ch\_pos\_D:*

*cpi Number,7;7 позиций просмотра*

*brne PC+2*

*ldi Number,0*

*ldi XL,low(varBuf\_disp) ; XL register low буфер данных,выводимых на дисплей*

*ldi XH,high(varBuf\_disp) ; XH register high буфер данных,выводимых на дисплей*

*ldi temp\_H,0x00*

*add XL,Number*

*adc XH,temp\_H*

*ld temp\_L,X*

*rcall digitConvert*

*;*

*inc Number*

*set*

*bld Byte\_fl,Disab\_Key; бит запрещ. опроса кнопки "Просмотр" от дребезга*

*ldi temp\_L,(1<<CS02)|(1<<CS00);частота TCNT0 Clk/1024,(CS02,CS01,CS00) - 101*

*out TCCR0,temp\_L*

*in temp\_L,TIMSK;UDRIE 5 10*

*set*

*bld temp\_L,TOIE0 ;TOIE0-Timer/Counter0 Overflow Interrupt Enable*

*out TIMSK,temp\_L; разрешить прерывание Tov0*

*ret*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

При вызове данной подпрограммы производится чтение из буфера *varBuf\_disp* значения по относительному адресу, находящемуся в регистре *Number*. При чтении используется косвенная адресация с помощью пары регистров *Х*. Для преобразования прочитанного значения двоично-десятичный код используется подпрограмма *digitConvert*. Затем выставляется флаг *Disab\_Key*, запрещающий опрос кнопки «Просмотр», и запускается таймер Т0 для формирования защитного промежутка, используемого при подавлении дребезга контактов. При этом разрешается прерывание по переполнению таймера Т0.

Далее следует подпрограмма управления динамической индикацией и таблица перекодировки символов:

*; Подпрограмма Display работы с дисплеем (динам. индикация)*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*Display: dec Disp\_Count*

*brne ex\_displ*

*;*

*ldi Disp\_Count,Val\_dispCount*

*;*

*inc Disp\_Numb*

*cpi Disp\_Numb,4*

*brne Out\_disp*

*clr Disp\_Numb*

*Out\_disp:*

*clr ZH*

*ldi ZL,18; ;указатель на Number*

*add ZL,Disp\_Numb*

*ld temp\_L,Z*

*; преобразуем в семисегментный код*

*ldi ZL,low(TABLE\*2);загружаем адрес начала*

*ldi ZH,high(TABLE\*2);таблицы в памяти программ (\*2 - для байтовой*

*add ZL,temp\_L ;адресации)*

*lpm temp\_L,Z ;читаем семисегментный код значения ;*

*out PortC,temp\_L; передаем на индикатор*

*;*

*in temp\_L,PINB*

*lsr temp\_L*

*brcc PC+2*

*ldi temp\_L,0b00001000; в начало (Number)*

*out PORTB,temp\_L*

*;*

*ex\_displ: ret*

*;*

*;==================================================*

*;------- Таблица перекодировки символов*

*TABLE: .db 0b00111111,0b00000110; коды "0","1"*

*.db 0b01011011,0b01001111; коды "2","3"*

*.db 0b01100110,0b01101101;;коды "4","5"*

*.db 0b01111101,0b00000111;;коды "6","7"*

*.db 0b01111111,0b01101111;;коды "8","9"*

*.db 0b10000000,0b00000000; коды " "," "*

*;==================================================*

Данная подпрограмма идентична, приведенной в лабораторной работе №2.

Следующим блоком программы является подпрограмма записи принятых данных в буфер для отображения на семисегментных индикаторах:

*Wr\_D\_rec: ;Подпрограмма записи принятых данных в буфер дисплея*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема*

*ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема*

*ldi XL,low(varBuf\_disp) ; XL register low буфер данных,выводимых на дисплей*

*ldi XH,high(varBuf\_disp)*

*ldi temp\_H,* *VAL\_REC*

*Loop\_wD: ld temp\_L,Y+*

*st X+,temp\_L*

*subi temp\_H,1*

*breq PC+2*

*rjmp Loop\_wD*

*ret*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

В работе данной подпрограммы используются регистры косвенного доступа к памяти *X* и *Y*. В регистре *X* хранится адрес буфера *var\_Buf\_Rxd*, а в регистре *Y* – *varBuf\_disp*. Затем проводится *VAL\_REC = 4* циклов чтения и буфера *rav\_Buf\_Rxd* и записи в буфер *varBuf\_disp*. В качестве промежуточного буфера используется регистр *temp\_L*.

Далее следует подпрограмма выполнения арифметических операций:

*arifm\_op:;Подпрограмма выполнения арифметических операций*

*;результат старший байт- temp\_H,младший байт - temp\_L*

*;==================================================*

*clr res\_a\_op;байт промежуточных результатов*

*ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема*

*ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема*

*ld temp\_L,Y*

*cpi temp\_L,0x01*

*breq add\_1\_b*

*cpi temp\_L,0x02*

*breq sub\_1\_b*

*cpi temp\_L,0x03*

*breq mul\_1\_b*

*cpi temp\_L,0x04*

*breq div\_1\_b*

*ex\_ar\_op: ret*

*;------------------------*

*;сложение двоичных чисел*

*add\_1\_b: ldd temp\_L,Y+1*

*ldd temp\_H,Y+2*

*add temp\_L,temp\_H*

*ldi temp\_H,0x00*

*mov res\_a\_op,temp\_H*

*adc temp\_H,res\_a\_op*

*rjmp ex\_ar\_op*

*;------------------------*

*;вычитание двоичных чисел*

*sub\_1\_b: ldd temp\_L,Y+1*

*ldd temp\_H,Y+2*

*sub temp\_L,temp\_H*

*ldi temp\_H,0x00*

*sbci temp\_H,0x00*

*rjmp ex\_ar\_op*

*;-------------------------*

*;умножение двоичных чисел*

*mul\_1\_b: ldd temp\_H,Y+1;множимое*

*ldd temp\_L,Y+2;множитель*

*ldi n\_ar\_op,8;восьмиразрядное число*

*clr res\_a\_op*

*cycle\_m: sbrc temp\_L,0*

*add res\_a\_op,temp\_H*

*lsr res\_a\_op*

*ror temp\_L*

*subi n\_ar\_op,1*

*brne cycle\_m*

*mov temp\_H,res\_a\_op*

*rjmp ex\_ar\_op*

*;-------------------------*

*;деление двоичных чисел*

*div\_1\_b: ldd temp\_H,Y+1;делимое*

*ldd temp\_L,Y+2;делитель*

*Divide: sub temp\_H,temp\_L*

*brcs DoneDividing*

*inc res\_a\_op*

*rjmp Divide*

*DoneDividing:*

*neg temp\_L*

*sub temp\_H,temp\_L*

*mov temp\_L,temp\_H*

*mov temp\_H,res\_a\_op*

*rjmp ex\_ar\_op*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

Подпрограмма осуществляет, используя косвенную адресацию, чтение первого принятого байта, в котором содержится код выполняемой арифметической операции. В соответствии с прочитанным кодом выполняется одна из следующих ниже подпрограмм.

Подпрограмма add\_1\_b осуществляет сложение двух чисел. Слагаемые считываются из буфера *var\_Buf\_Rxd* в регистры *temp\_L* и *temp\_H*. Затем осуществляется их сложение, младший байт результата записывается в регистр *temp\_L*, а старший – в регистр *temp\_H*.

Подпрограмма *sub\_1\_b* осуществляет вычитание одного числа из другого. Уменьшаемое и вычитаемое считываются из буфера *var\_Buf\_Rxd* в регистры *temp\_L* и *temp\_H*. Затем осуществляется их вычитание, младший байт результата записывается в регистр *temp\_L*, а старший – в регистр *temp\_H*.

Подпрограмма *mul\_1\_b* осуществляет умножение двух чисел. Множители считываются из буфера *var\_Buf\_Rxd* в регистры *temp\_L* и *temp\_H*. Затем осуществляется их умножение при помощи метода сложения ряда частичных произведений, младший байт результата записывается в регистр *temp\_L*, а старший – в регистр *temp\_H*.

Подпрограмма *div\_1\_b* осуществляет целочисленное деление одного числа на другое. Множители считываются из буфера *var\_Buf\_Rxd* в регистры *temp\_L* и *temp\_H*. Затем осуществляется их деление путем последовательного вычитания делителя из делимого, результат записывается в регистре *temp\_H*, а остаток от деления – в регистр *temp\_L*.

В следующем блоке содержится подпрограмма подготовки данных к передаче и подпрограмма записи данных передачи в буфер дисплея:

*pre\_date:;Подпрограмма подготовки данных к передаче*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*ldi YL,low(varBuf\_Txd); Load Y register low буфер передачи*

*ldi YH,high(varBuf\_Txd) ; Load Y register high буфер передачи*

*clr c\_sumTRAN*

*add c\_sumTRAN,temp\_H*

*st Y+,temp\_H*

*add c\_sumTRAN,temp\_L*

*st Y+,temp\_L*

*com c\_sumTRAN*

*st Y+,c\_sumTRAN*

*ret*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*Wr\_D\_tr:;Подпрограмма записи данных передачи в буфер дисплея*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*ldi YL,low(varBuf\_Txd); Load Y register low буфер передачи*

*ldi YH,high(varBuf\_Txd) ; Load Y register high буфер передачи*

*ldi temp\_L,VAL\_REC; смещение 4*

*ldi temp\_H,0x00*

*ldi XL,low(varBuf\_disp);XL register low буфер данных,выводимых на дисплей*

*ldi XH,high(varBuf\_disp)*

*add XL,temp\_L*

*adc XH,temp\_H*

*ldi temp\_H,VAL\_TR*

*rcall Loop\_wD*

*ret*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

Подпрограмма *pre\_date* осуществляет вычисление контрольной суммы передаваемых данных и запись передаваемых данных в буфер *rav\_Buf\_Txd*.

Подпрограмма Wr\_D\_tr осуществляет чтение из буфера *rav\_Buf\_Txd* и запись в буфер *varBuf\_disp*, начиная с четвертого адреса. В качестве промежуточного буфера используется регистр *temp\_L*.

Далее следует подпрограмма конвертации двоичного однобайтового числа в двоично-десятичный код, которая аналогична, описанной в лабораторной работе №2. Затем идет пять подпрограмм обработки прерываний. Первая из них – это подпрограмма обработки прерывания при переполнении таймера Т0:

*;Subroutine interrupt Overflow 0*

*;Подпрограмма обработки прерывания переполнения таймера Т0*

*;Т0 - 8 разр таймер. Counter0 оverflow соотвествуетвремени 256\*1024/8000000=0,033с*

*;за 1cек =(1024/9216000)\*256\*N\_отсчетов(переполнений)=35 переполнений*

*;(предотвращение срабатываний от дребезга контактов кнопки "Просмотр")*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*time\_d\_k: push temp\_L*

*in temp\_L,SREG*

*push temp\_L*

*;*

*inc Time*

*cpi Time,VAL\_time*

*brne ex\_timDK*

*clt*

*bld Byte\_fl,Disab\_Key; бит разр. опроса кнопки "Просмотр" от дребезга*

*ldi temp\_L,0x00;(1<<CS02)|(1<<CS00);No clock source,(CS02,CS01,CS00) - 000*

*out TCCR0,temp\_L*

*in temp\_L,TIMSK*

*clt*

*bld temp\_L,TOIE0 ;TOIE0-Timer/Counter0 Overflow Interrupt Enable*

*out TIMSK,temp\_L; запрещение прерывание Tov0*

*clr Time*

*;*

*ex\_timDK: pop temp\_L*

*out SREG,temp\_L*

*pop temp\_L*

*reti*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

В начале подпрограммы осуществляется сохранение регистра *temp\_L* и регистра состояния в стек. При каждом вызове подпрограммы осуществляется инкрементирование регистра *Time*. Когда регистр достигнет значения *VAL\_time = 35* осуществляется сброс флага *Disab\_Key*, чем разрешается отрос кнопки «Просмотр». Кроме этого таймер Т0 выключается и запрещается прерывание по его переполнению. В конце подпрограммы восстановляются регистр *temp\_L* и регистр состояния из стека.

Далее следует подпрограмма обработки прерывания по окончанию приема модуля USART:

*;Subroutine interrupt USART RX Complete*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*REC\_date: push temp\_L*

*push temp\_H*

*in temp\_L,SREG*

*push temp\_L*

*;*

*in temp\_H,UCSRA*

*rd\_UDR: in temp\_L,UDR*

*rjmp rt\_rec*

*;*

*pop\_rec:*

*pop temp\_L*

*out SREG,temp\_L*

*pop temp\_H*

*pop temp\_L*

*reti*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*rt\_rec:*

*andi temp\_H,(1<<FE)|(1<<DOR)|(1<<PE)*

*breq USART2NoError*

*rjmp pop\_rec*

*USART2NoError:*

*st Y+,temp\_L*

*inc Cou\_Rec*

*mov temp\_H,Cou\_Rec*

*cpi temp\_H,0x01*

*breq rec\_1\_b*

*cpi temp\_H,VAL\_REC*

*breq rec\_end*

*add c\_sumREC,temp\_L*

*rjmp pop\_rec*

*rec\_1\_b: add c\_sumREC,temp\_L*

*in temp\_L,TIMSK*

*set*

*bld temp\_L,OCIE1A;OCIE1A разрешить прерывание*

*out TIMSK,temp\_L*

*ldi temp\_L,(1<<WGM12)|(1<<CS10);WGM3=0,WGM2=1,WGM1=0,WGM0=0,режим CTC*

*; clkI/O/1 (No prescaling),CS2,CS1,CS0=001*

*out TCCR1B,temp\_L;(No prescaling CS0=1*

*rjmp pop\_rec*

*rec\_end:*

*com temp\_L;инверсия принятого байта*

*cp temp\_L,c\_sumREC; проверка контр суммы*

*brne ex\_rec*

*set*

*bld Byte\_fl,F\_receive;прием верных данных*

*;*

*ex\_rec: in temp\_L,TIMSK*

*clt*

*bld temp\_L,OCIE1A;OCIE1A запретить прерывание*

*out TIMSK,temp\_L*

*ldi temp\_L,(1<<WGM12);WGM3=0,WGM2=1,WGM1=0,WGM0=0,режим CTC*

*; clkI/O/1 (stop TCNT1),CS2,CS1,CS0=000*

*out TCCR1B,temp\_L*

*ldi temp\_L,0x00*

*ldi temp\_H,0x00*

*out TCNT1H,temp\_H*

*out TCNT1L,temp\_L*

*clr Cou\_Rec*

*clr c\_sumREC*

*ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема*

*ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема*

*rjmp pop\_rec*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

В начале подпрограммы осуществляется сохранение регистров *temp\_L*, *temp\_H* и регистра состояния в стек. Далее осуществляется чтение регистра состояния *UCSRA* в регистр *temp\_H*, а регистра данных *UDR* в регистр *temp\_L*, после чего МК переходит на строку с меткой *rt\_rec*.

Начиная со строки с меткой *rt\_rec*, осуществляется проверка флагов возможных ошибок приема. Если ошибок нет, по принятый байт сохраняется в буфер *rav\_Buf\_Rxd*.

Если принятый байт является первым, то МК переходит на строку с меткой rec\_1\_b, где запускается таймер Т1 для формирования времени таймаута принимаемых данных и разрешается прерывание по совпадению канала А таймера Т1.

Если принятый байт – последний (четвертый), то МК переходит на строку с меткой *rec\_end*, где данный байт сравнивается с инвертированным значением регистра *c\_sumREC*. Если котрольная сумма верна, то выставляется флаг *F\_receive*. Затем останавливается таймер Т1 и запрещаются прерывания от него, сбрасываются регистры *Cou\_REC* и *c\_sumREC*.

Если принятый байт не является ни первым, ни последним, то он добавляется к регистру *c\_sumREC* для вычисления контрольной суммы. Подпрограмма *pre\_date* осуществляет вычисление контрольной суммы передаваемых данных и запись передаваемых данных в буфер *rav\_Buf\_Txd*.

В конце подпрограммы осуществляется восстановление сохраненных регистров из стека.

Далее следует подпрограмма обработки прерывания при опустошении буфера передачи модуля USART:

*;Subroutine interrupt USART Data register Empty*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*B\_TRANS: push temp\_L*

*in temp\_L,SREG*

*push temp\_L*

*;*

*ldi temp\_L,(1<<TXEN)|(1<<TXCIE);разрешить прерывание TXC*

*out UCSRB,temp\_L*

*ldi YL,low(varBuf\_Txd); Load Y register low буфер передачи*

*ldi YH,high(varBuf\_Txd); Load Y register high буфер передачи*

*ld temp\_L,Y+*

*clt*

*bld Byte\_fl,F\_receive;*

*out UDR,temp\_L*

*;*

*pop temp\_L*

*out SREG,temp\_L*

*pop temp\_L*

*reti*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

В начале подпрограммы осуществляется сохранение регистра *temp\_L* и регистра состояния в стек. Далее осуществляется разрешение прерывания по окончанию передачи модуля USART, а прерывание по опустошению буфера передачи запрещается. Затем записывается первое значение из буфера *rav\_Buf\_Txd* в регистр передачи и сбрасывается флаг *F\_receive*.

В конце подпрограммы осуществляется восстановление сохраненных регистров из стека.

Далее следует подпрограмма обработки прерывания по окончании передачи модуля USART:

*;Subroutine interrupt USART, Tx Complete*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*TRANdate: push temp\_L*

*in temp\_L,SREG*

*push temp\_L*

*push temp\_H*

*rjmp r\_trans*

*;*

*pop\_tran:*

*pop temp\_H*

*pop temp\_L*

*out SREG,temp\_L*

*pop temp\_L*

*;*

*reti*

*;*

*r\_trans: inc Cou\_Tran*

*mov temp\_L,Cou\_Tran*

*cpi temp\_L,VAL\_TR*

*breq end\_tr*

*ld temp\_L,Y+*

*out UDR,temp\_L*

*rjmp pop\_tran*

*end\_tr: clr Cou\_Tran*

*ldi temp\_L,(1<<RXEN)|(1<<RXCIE);*

*out UCSRB,temp\_L*

*ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема*

*ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема*

*cbi PORTD,CID*

*set*

*bld Byte\_fl,F\_trans*

*rjmp pop\_tran*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

В начале подпрограммы осуществляется сохранение регистра *temp\_L* и регистра состояния в стек. Далее МК переходит на строку с меткой *r\_trans*. Осуществляется запись байта данных из буфера *rav\_Buf\_Txd* в регистр передачи. Если значение регистра *Cou\_Tran* стало равно *VAL\_TR*, по передача прекращается, т.к. все данные в буфере *rav\_Buf\_Txd* закончились.

После этого запрещается работа передатчика и прерывание по окончанию передачи и разрешается работа приемника и прерывание по окончанию, устанавливается флаг *F\_receive*.

В конце подпрограммы осуществляется восстановление сохраненных регистров из стека.

Далее следует подпрограмма обработки прерывания при совпадении канала А таймера Т1:

*;Subroutine interrupt OC1A*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*Time\_OUT: push temp\_L*

*push temp\_H*

*in temp\_L,SREG*

*push temp\_L*

*;*

*ldi temp\_L,(1<<WGM12);WGM13=0,WGM12=1,WGM11=0,WGM10=0,режим CTC*

*; No clock source,CS42,CS41,CS40=000*

*out TCCR1B,temp\_L;*

*in temp\_L,TIMSK*

*clt*

*bld temp\_L,OCIE1A;OCIE1A запретить прерывание*

*out TIMSK,temp\_L*

*ldi temp\_H,0x00*

*ldi temp\_L,0x00 ;*

*out TCNT1H,temp\_H;сброс счетчика*

*out TCNT1L,temp\_L*

*clr Cou\_Rec*

*clr c\_sumREC*

*ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема*

*ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема*

*;*

*pop temp\_L*

*out SREG,temp\_L*

*pop temp\_H*

*pop temp\_L*

*reti*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

В начале подпрограммы осуществляется сохранение регистров *temp\_L, temp\_H* и регистра состояния в стек. Далее останавливается работа таймера Т1 и запрещается прервынае при совпадении канала А таймера Т1. Затем сбрасываются регистры *Cou\_Rec* и *c\_sumREC*, что приводит к тому, что данные, принятые модулем USART будут записываться с начала буфера *rav\_Buf\_Rxd*.

В конце подпрограммы осуществляется восстановление сохраненных регистров из стека.

3.3 Проверьте правильность работы программы в симуляторе, для этого нажмите кнопку Assemble and run (ctrl+F7).

Добавьте в окно Watch все перечисленные ранее регистры, а также регистры XL, XH, YL, YX, ZH и ZL. Настройте столбец Value окна Watch так, чтобы он отображал значения регистров в шестнадцатиричном формате.

Откройте окно Memory (View – Memory) и закрепите его в рабочем пространстве. В качестве контролируемых данных выберите Data в выпадающем списке окна Memory. Нажимая кнопку abs в окне Memory, добейтесь, чтобы в одной строке отображалось восемь бит памяти данных.

Используя клавишу F11 (пошаговое выполнение программы) дойдите до метки *Start*. При этом происходит инициализация используемой периферии, сбрасываются буферы в памяти данных, присваивается начальное значение регистрам. Наблюдайте в окне I/O View изменение состояния регистров при инициализации. Выводы PD4, PD7 МК используются и конфигурируются как входы. Логическое состояние на них задается при помощи регистра *PIND*. Неактивное состояние кнопки соответствует логической «1», поэтому необходимо установить соответствующие биты регистра PIND в единицу.

Начиная с метки *Start* начинается основная программа. На первой ее строчке вызывается подпрограмма динамической индикации *Display*. Нажмите несколько раз кнопку F11, наблюдая за работой программы. Вы можете увидеть, как постепенно уменьшается значение регистра *Disp\_Count*, который отвечает за формирование времени активности семисегментного индикатора.

Симулируйте прием одного кадра данных приемников модуля USART. Для этого выставите флаг прерывания по завершению приема RXC в регистре *UCSRA* в окне *View* (седьмой бит). Нажмите один раз кнопку F11, Вы перейдете на соответствующий вектор прерывания. Нажмите несколько раз кнопку F11 и дойдите до строки с меткой *rd\_UDR*, на строке с данной меткой происходит чтение регистра принятых данных *UDR*. Чтобы прочитать из этого регистра данные, отличные от ноля, необходимо перед циклом чтения записать в регистр требуемые данные. В окне View введите значение 0х01 в регистр *UDR* и нажмите *Enter*. **Будьте внимательны, при симулировании данные в регистре UDR хранятся только один такт, на следующий такт они сбрасываются.** Нажмите кнопку F11 несколько раз, чтобы выйти из подпрограммы обработки прерывания. Наблюдайте при этом за изменением регистров в окне *View* и буферов в памяти данных в окне *Memory*.

Симулируйте прием еще трех кадров данных по приведенной выше методике. Последовательно записывайте данные: 0хАА, 0х55, 0хFF. Данная последовательность означает, что будет выполнять операция суммирования (0х01) чисел 170 (0хАА) и 85 (0х55).

После выполнение перечисленных выше действий МК готов к исполнению арифметических операций над принятыми данными. Нажмите кнопку F11 несколько раз до перехода на строчку с меткой *Receive*. С этой строки начинается исполнение ряда подпрограмм для обработки данных: *Wr\_D\_rec*, *arifm\_op*, *pre\_date*, *Wr\_D\_tr*. Нажимая кнопку F11 внимательно просимулируйте работу каждой из подпрограмм. После выполнения всех перечисленных подпрограмм в окне *Memory* должны обновиться данные во всех буферах.

Далее Вы попадете на строку с меткой *Wait*. Начиная с этой строки, находится группа команд, проверяющих условие нажатия кнопок «Просмотр» (меняет отображаемый на семисегментных индикаторах байт) и «Ответ» (инициирует передачу информации с выхода модуля USART). Симулируйте нажатие кнопки «Просмотр» переводом соответствующего бита регистра *PIND* в ноль. Нажмите несколько раз кнопку F11 до перехода в подпрограмму *ch\_pos\_D*. Полностью просимулируйте работу этой подпрограммы, следя за изменением регистров в окне *View*. По завершении работы подпрограммы значение регистров *Number,* *Hundreds*, *Tens* и *Ones* должно измениться. Симулируйте отпускание кнопки «Просмотр» переводом соответствующего бита регистра *PIND* в единицу.

Симулируйте нажатие кнопки «Ответ» переводом соответствующего бита регистра *PIND* в ноль. Нажмите клавишу F11 несколько раз до перехода к строке с меткой *Trans*. В данной строке разрешается прерывание по опустошению буфера передачи модуля USART. Флаг этого прерывания будет уже установленным, поэтому нажмите кнопку F11 несколько раз до перехода к подпрограмме обработки прерывания по опустошению буфера передачи. В данной подпрограмме осуществляется запись первого байта данных в регистр *UDR*, запрещается прерывание по опустошению буфера и разрешается прерывание по окончанию передачи. Нажмите кнопку F11 несколько раз, чтобы выйти из данной подпрограммы.

Программа перейдет на строку с меткой *wait\_tr* и будет находиться там до окончания передачи данных. Передача следующего кадра будет осуществлена после завершения передачи текущего кадра. Чтобы симулировать работу прерываний Tx Complete необходимо выставить точку останова на строке rjmp TRANdate в описании векторов прерываний. Нажмите кнопку F5, наблюдайте переход к подпрограмме обработки прерывания по завершению передачи. Нажимая кнопку F11, пройдите ее полностью, при этом обращайте внимание на изменение регистров в окне *Watch* и регистра UDR в окне *View*. После выхода из подпрограммы симулируйте прерывание еще два раза и пройдите подпрограмму снова (требуется передать три байта информации). В последнем цикле работы подпрограммы устанавливается флаг *F\_trans* и программа переходит в начальное состояние. Таким образом цикл работы программы завершен.

При симуляции не была проверена работа следующих веток программы:

- подавление дребезга контактов при помощи таймера Т0;

- превышение времени таймаута, задаваемого таймером Т1;

- выполнение арифметических операций вычитания, умножения и деления.

Выйдите из симулятора, нажав кнопку Stop Debugging (Ctrl+Shift+F5).

3.5 Запрограммируйте МК стенда и проверьте работу программы на нем. Для этого через, USB-кабель подключите ПК, к COM-порту стенда EasyAVR5. Запустите программу AVRStend. В данной программе можно задавать скорость передачи, а также наличие бит паритета. После каждой смены параметров передачи необходимо нажимать кнопку **Установить**. Далее идет поле ввода данных, после изменения данных необходимо нажать кнопку **Сформировать сообщение**. Новое сообщение появиться в соответствующей строке.

3.6 Выполните передачу команды. Для этого необходимо нажать кнопку **Передать сообщение.** Если стенд подключен правильно, то после передачи должен загореться светодиод СИД. Проверьте правильность принятых данных последовательно нажимая кнопку «Просмотр». Крайний левый индикатор показывает номер просматриваемого байта.

1) Байт кода операции:

- 0x01 – сложение 2-х однобайтных двоичных чисел без знака (0 до 255);

- 0х02 – вычитание 2-х однобайтных двоичных чисел без знака (0 до 255);

- 0х03 – умножение 2-х однобайтных двоичных чисел без знака (0 до 255);

- 0х04 – деление 2-х однобайтных двоичных чисел без знака (0 до 255).

2) Байт первого операнда (1-ое слагаемое, уменьшаемое, множимое, делимое).

3) Байт второго операнда (2-ое слагаемое, вычитаемое, множитель, делитель).

4) Байт контрольной суммы с инверсией

Кадр ответа состоит из следующей последовательности:

5) Старшего байта результата выполнения операции/(частное);

6) Младший байт результата выполнения операции/(остаток от деления);

7) Байт контрольной суммы с инверсией.

3.7 Для передачи ответа на стенде необходимо нажать кнопку «Ответ», при этом в программе AVRStend появится информация в строках **Прием данных** и **Отображение результата**.

3.8 При нажатии кнопки «Просмотр» наблюдается необходимость 1сек паузы между нажатиями. Т.е. время, выбранное для подавления дребезга контактов кнопки выбрано необоснованно большим (рекомендуемое 5мсек). Доработать программу сократив паузу между нажатиями.

3.9 Проведите эксперимент по изменению и передаче команд арифметических операций и правильности их приема стендом EasyAVR5:

- сложение двух чисел, задавая числа в десятичном, двоичном, шестнадцатиричном формате

- вычитание двух чисел, задавая числа в десятичном, двоичном, шестнадцатиричном формате

- умножение двух чисел, задавая числа в десятичном, двоичном, шестнадцатиричном формате

- деление двух чисел, задавая числа в десятичном, двоичном, шестнадцатиричном формате

3.10 Измените скорость передачи в программе МК и программе AVRStend, проверьте и добейтесь правильной работы изменной программы.

3.11 Добавьте бит паритета в программах МК и AVRStend и проверьте и добейтесь правильной работы изменной программы.

**4 Контрольные вопросы**

1. Структура модуля USART.

2. Режимы работы модуля USART.

3. Назначение бита паритета и контрольной суммы. Расчет значения бита четности.

4. Скорость передачи информации в модуле USART. Расчет скорости передачи. Регистры, определяющие скорость передачи.

5. Форматы кадра передачи

6. Флаги ошибки приемника

7. Флаг «Регистр данных пуст». Передача данных, управляемая прерыванием «Регистр данных пуст».

8. Флаг «Передача окончена». Передача данных, управляемая прерыванием «Передача окончена»

9. Приемник USART. Флаг готовности приемника и вызов прерывания

10. Назначение и характеристики последовательного интерфейса RS232.

11. Реализация алгоритма умножения методом сложения ряда частичных произведений.

12. Реализация алгоритма деления последовательным вычитанием.

13. Организация записи и чтения в память данных методом косвенной адресации.

14. Режим работы СТС таймера/счетчика Т1. Назнатение подпрограммы Time\_OUT (interrupt OC1A).