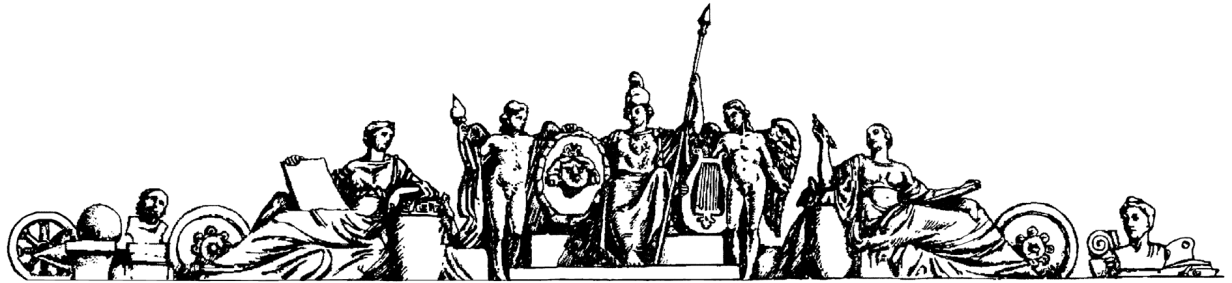
Федеральное агентство по образованию

Государственное образовательное учреждение высшего профессионального образования

Московский государственный технический университет им. Н.Э. Баумана



Кафедра «Робототехнические системы и мехатроника»

**Отчёт по лабораторной работе №1**

По курсу

**" Микроконтроллерные устройства управления "**

Тема

**Ознакомление с программно-аппаратными средствами составления и отладки программ на базе микроконтроллеров с CISC и RISC архитектурой семейства х51 и семейства AVR в системе реального времени**

*Преподаватель:*

***к.т.н. доц. Солнцев В.И.***

*Студент группы СМ7-84Б:*

***Балкунов К.С.***

2024

**Цель лабораторной работы:** ознакомление с программно-аппаратными средствами составления и отладки программ на базе микроконтроллеров с CISC и RISC архитектурой семейства х51 и семейства AVR в системе реального времени.

1) Описание периферии стенда

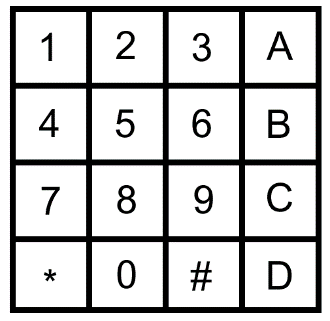
1. Светодиодный индикатор

Управление светодиодными индикаторами осуществляется непосредственно от портов ввода-вывода микроконтроллера. Всего имеется 3 светодиода (LED3, LED2, LED1). Зажигание светодиода происходит, когда на соответствующий вывод порта подаётся 0. Выключение зажженного светодиода происходит, когда на соответствующий вывод порта подаётся 1.

1. Зуммер

Зуммер подключен к выводу P1.4. Для получения звука надо подавать на зуммер меандр, то есть полпериода подавать на P1.4 единицу, полпериода – ноль.

1. Клавиатура

Клавиатура состоит из 16 кнопок и подключается к порту 2 (выводы P2.0 – P2.7)

1. Семисегментный индикатор

Семисегментный индикатор подключён к SPI (последовательный периферийный интерфейс) микроконтроллера через 8-битный сдвиговый регистр 74HC595, обеспечивающий последовательный ввод и параллельный вывод данных.

1. Жидкокристаллический индикатор HD44780

Линии данных DB0…DB7 жидкокристаллического индикатора подключёны к SPI-интерфейсу микроконтроллера (как и семисегментный индикатор) через 8-битный сдвиговый регистр 74HC595, обеспечивающий последовательный ввод и параллельный вывод данных.

1. Часы реального времени DS1307

Для часов реального времени DS1307 адресом устройства является последовательность 1101000. Работа с устройством производится после инициализации модуля I2C микроконтроллера.

1. Устройство ЭСППЗУ EEPROM 24C01A

Для EEPROM 24C01A адресом устройства является последовательность 1010000. Работа с устройством производится после инициализации модуля I2C микроконтроллера.

1. Цифровой термодатчик DS1820

Данные с датчика считываются через 1-проводную последовательную шину (1-WIRE). Эта шина подключена к P0.2.

1. Шина I2C

EEPROM 24C01A подключена к I2C-интерфейсу микроконтроллера. I2C шина является одной из модификаций последовательных протоколов обмена данных. Помимо EEPROM к I2C шине подключаются часы реального времени DS1307. Для осуществления процесса обмена информацией по I2C шине, используется всего два сигнала линия данных SDA линия синхронизации SCL.

Каждое устройство, подключённое к этой шине, распознается по уникальному адресу и может работать как передатчик или приёмник, в зависимости от назначения устройства. Кроме того, устройства могут быть классифицированы как ведущие и ведомые при передаче данных. В нашем случае ведущим является микроконтроллер, а ведомыми - EEPROM 24C01A и часы реального времени DS1307.

2) Описание стандартных интерфейсов контроллера

1. **SPI**

Последовательный периферийный интерфейс SPI (Serial Peripheral Interface), реализованный в микроконтроллерах семейства, имеет два назначения. Прежде всего, через него может быть осуществлено программирование микроконтроллера (так называемый режим последовательного программирования).

Вторым назначением интерфейса является организация высокоскоростного обмена данными между микроконтроллером и различными периферийными устройствами, такими как цифровые потенциометры ЦАП/АЦП, Flash-ПЗУ и др. Посредством этого интерфейса также может производиться обмен данными между несколькими микроконтроллерами AVR.

При обмене данными по интерфейсу SPI микроконтроллер AVR может работать как в режиме Master, так и в режиме Slave. При этом пользователь может задать следующие параметры:

* скорость передачи (четыре программируемых значения);
* формат передачи (от младшего разряда к старшему или наоборот).

Дополнительной возможностью подсистемы SPI является «пробуждение» микроконтроллера из режима Idle при поступлении данных.

1. **UART**

Микроконтроллеры Atmega 16x и i51 имеют в своем составе модуль полнодуплексного универсального асинхронного приемопередатчика (UART).

Через него осуществляется прием и передача информации, представленной последовательным кодом, поэтому модуль UART часто называют также последовательным портом. С помощью этого модуля микроконтроллер может обмениваться данными с различными внешними устройствами.

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

Известно, что при передаче данных могут происходить различные сбои. Модуль UART может обнаруживать и сигнализировать о следующих внештатных ситуациях:

* переполнение;
* ошибка кадрирования;
* неверный старт-бит.

Для уменьшения вероятности сбоев в модуле реализована такая полезная функция, как фильтрация помех.

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».

Выводы микроконтроллера, используемые модулем UART, являются линиями портов ввода/вывода общего назначения.

Последовательный порт может работать в четырех различных режимах:

Режим 0. Информация и передается, и принимается через вывод входа приемника (RXi, TXi).

Режим 1. В этом режиме передаются через вывод TXD или принимаются через RXD 10 бит информации: старт-бит (0), 8 бит данных и стоп-бит (1) при приеме информации в бит RB8 регистра управления/статуса приемопередатчика SCON заносятся стоп-бит Скорость приема/передачи — величина переменная и задается таймером;

Режим 2. В этом режиме через вывод TXD передаются или через RXD принимаются 11 бит информации: старт-бит, 8 бит данных, программируемый девятый бит и стоп-бит. Частота приема/передачи выбирается программой и может быть равна либо 1/32, либо 1/64 частоты резонатора в зависимости от управляющего бита SMOD; В режиме 2 частота передачи определяется выражением:

F =2SMODfРЕЗ / 64

Режим 3. Совпадает с режимом 2 во всех деталях, за исключением частоты приема/передачи, которая является величиной переменной и задается таймером

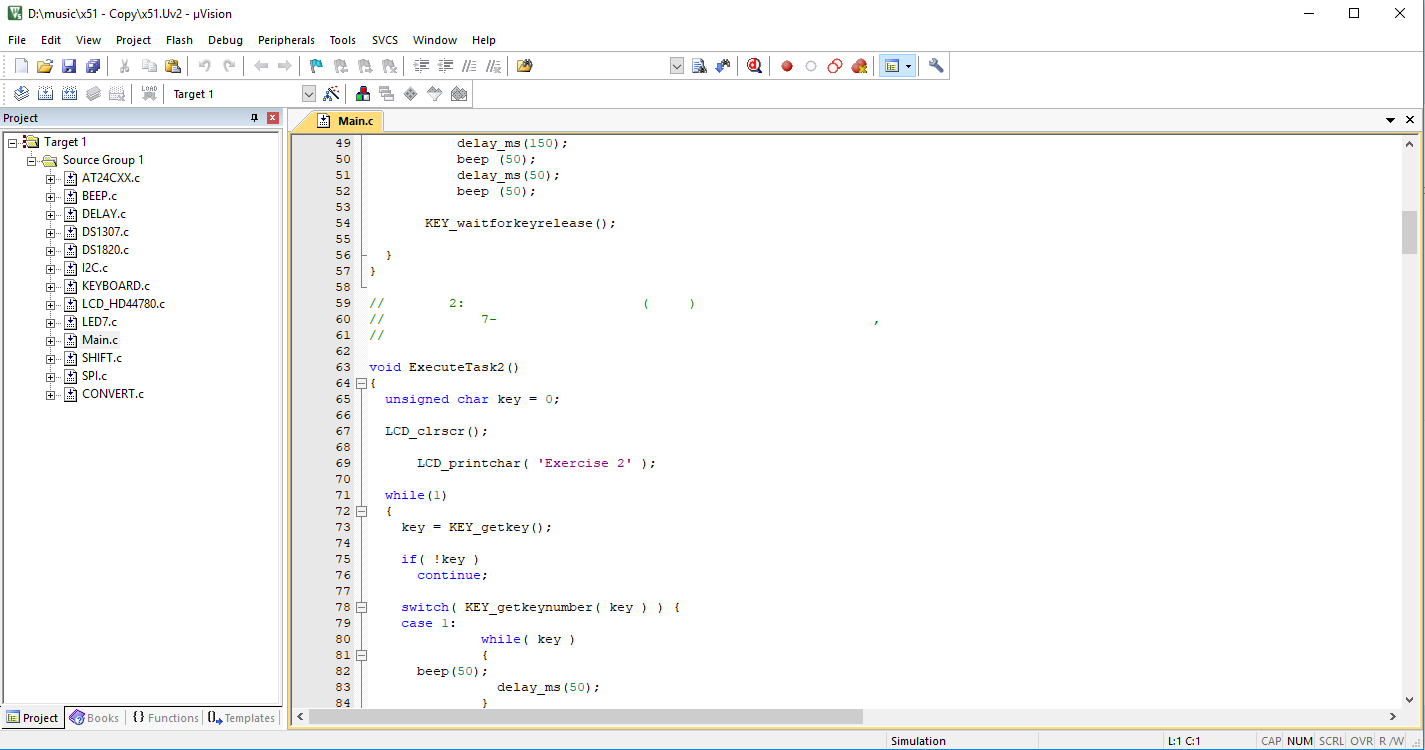
В режиме 0 частота передачи зависит только от резонансной частоты кварцевого резонатора fрез: F = fРЕЗ / 12

В режимах 1 и 3 в формировании частоты передачи, кроме управляющего бита SMOD, принимает участие таймер 1. При этом частота передачи f зависит от частоты переполнения fOVLT и определяется следующим образом: F =2SMOD fOVLT / 32.

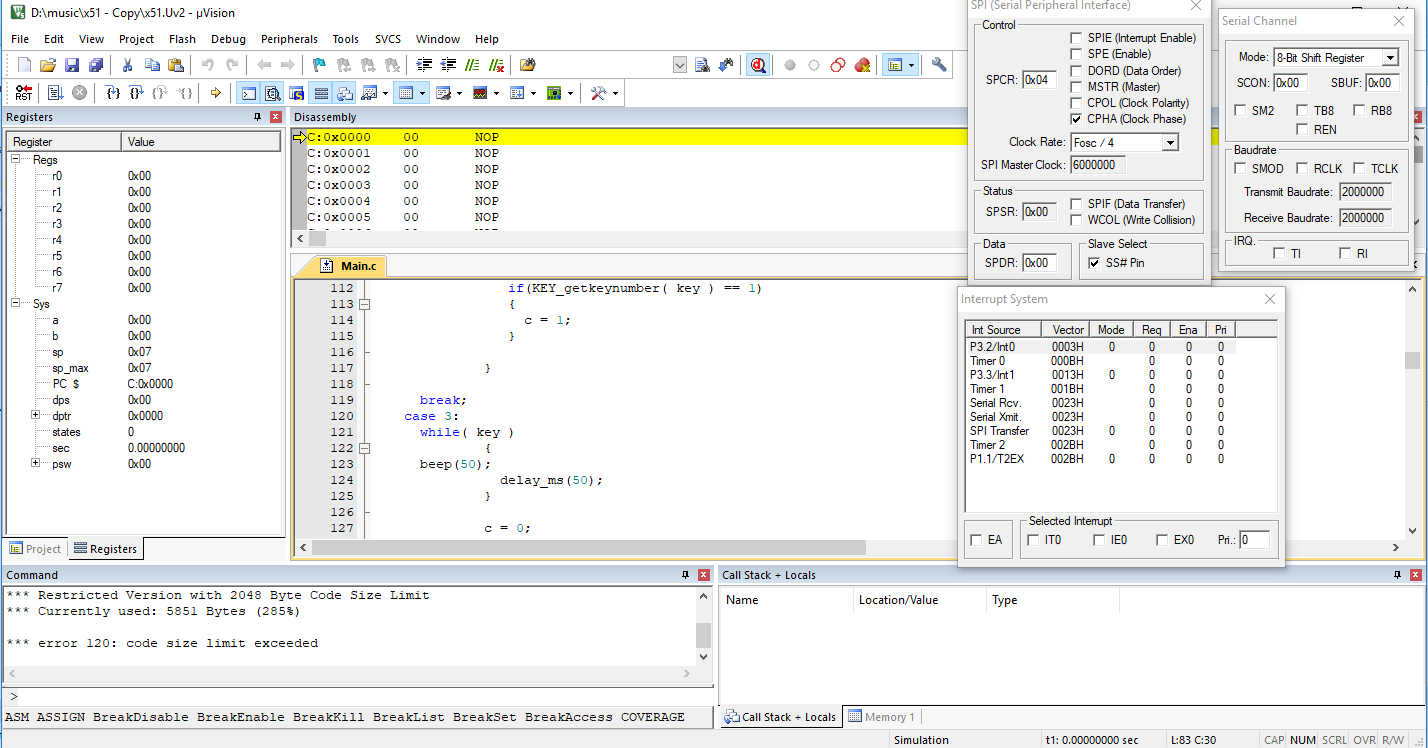
**Часть 1. Ознакомление с программно-аппаратными средствами составления и отладки программ на базе CISC микроконтроллеров семейства х51 в системе реального времени**

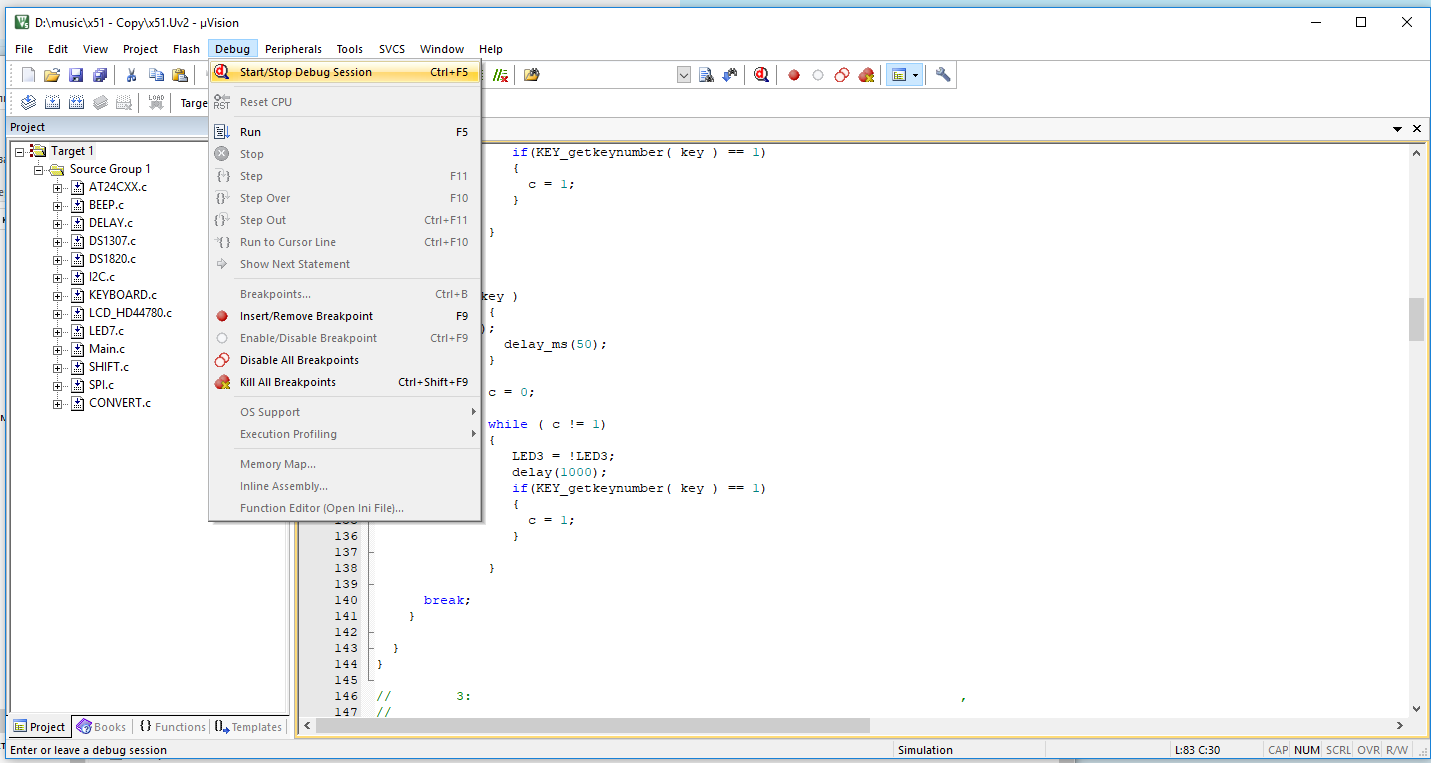
1) Описание процесса подготовки, прошивки контроллера и отладки программ

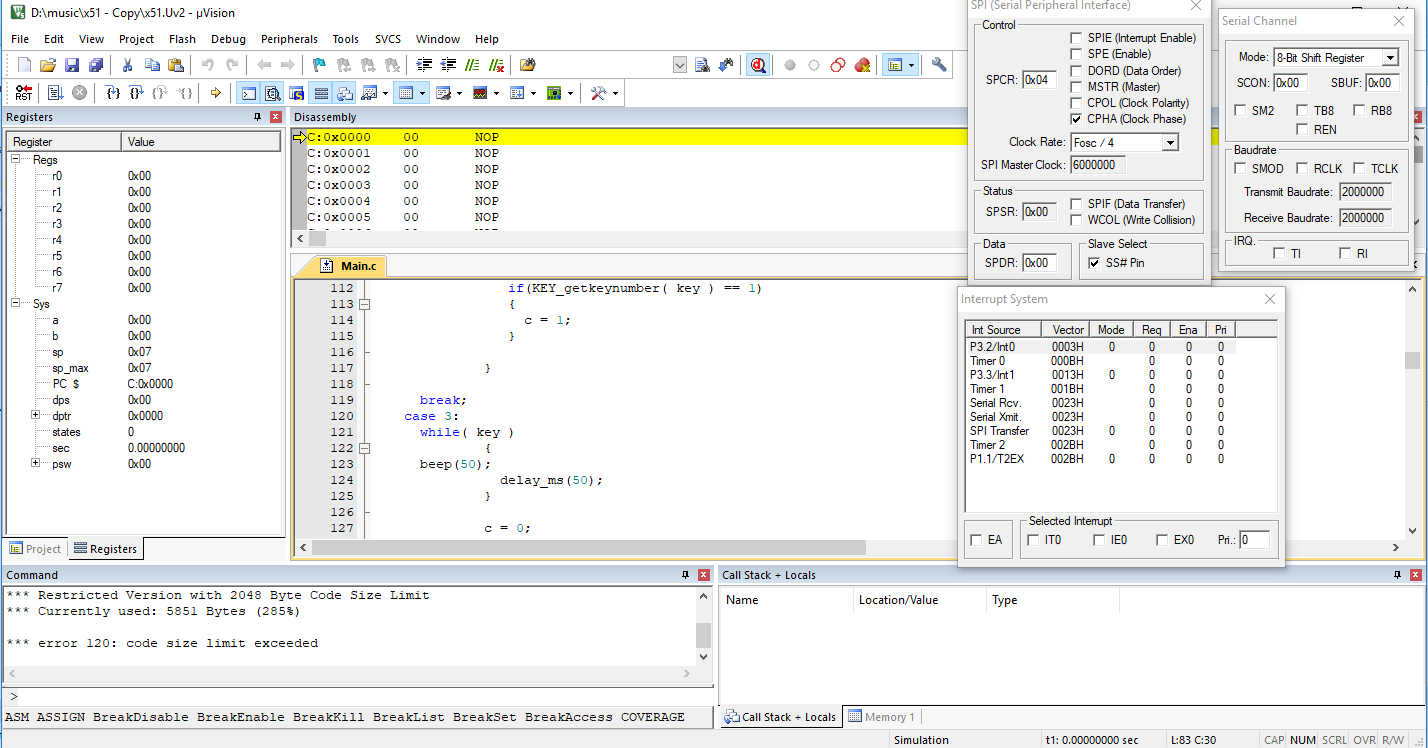
Программа разрабатывается в среде µVision2



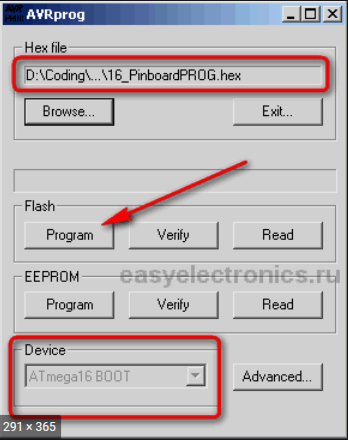
Для компиляции проекта нужно собрать проект, для этого нужно выполнить перестройку проекта.



Так же для отладки кода и его дизассемблирования можно использовать режим отладки



После успешного написания текста программы, созданный файл машинных кодов нужно записать в контролер, программу AVRprog



Порядок действий

а) Выбрать устройство

б) выбрать путь к файлу

в) нажать кнопку Program

2) Программа на базе микроконтроллера семейства х51

**Формулировка задачи**

1) Вывести цифры от 0 до 7 десятичной системы счисления (ввод осуществляется через клавиатуру стенда), в двоичном представлении, используя светодиоды стенда.

2) Параллельно с включением светодиодов, воспроизвести динамиком двоичное число по следующим правилам:

* Длинный сигнал соответствует биту 1, короткий - 0.
* Число воспроизводится от младшего бита к старшему.

3) При нажатии «\*» светодиоды выключаются. Динамик при этом перестаёт издавать звуки.

Текст программы приведен в Приложении 1.

Все процессы обрабатываются практически одновременно, так как система является системой реального времени, работа осуществляется в прерываниях.

Для обработки событий, происходящих асинхронно по отношению к выполнению программы, лучше всего подходит механизм прерываний. Прерывание можно рассматривать как некоторое особое событие в системе, требующее моментальной реакции.

3) Реализация прерываний в программе

**Прерывания по таймеру**

Для реализации прерываний по таймеру используем таймер 0 в 16 - битном режиме для этого меняем регистр TMOD.Для реализации отсчета в 1 миллисекунду будет использовать частоту 1000 Гц. Частоту тика 16-битного таймера разбиваем на младшие и старшие 8 бит.

1. #define CLOCK 22118400 // частота микроконтроллера  
   #define TIMER\_FREQ\_HZ 1000 // необходимая для решения задания частота  
   #define RELOAD\_VALUE (65536-(CLOCK/12/TIMER\_FREQ\_HZ)) // задаем частоту тика таймера  
   #define RELOAD\_VALUE\_H ((unsigned char)(RELOAD\_VALUE>>8)) // старшие 8 бит  
   #define RELOAD\_VALUE\_L ((unsigned char)(RELOAD\_VALUE&256)) // младшие 8 бит

2) Инициализация таймера 0.

1. void init\_timer0(void)
2. {// инициализация таймера 0
3. TMOD = 0x01; // 16-битовый режим
4. TL0 = RELOAD\_VALUE\_L;
5. TH0 = RELOAD\_VALUE\_H;
6. ET0 = 1; // Бит разрешения прерывания от таймера 0
7. TR0 = 1; // Бит управления таймера 0 для пуска/останова таймера/счетчика
8. }

3) Обработчик прерываний от таймера 0.

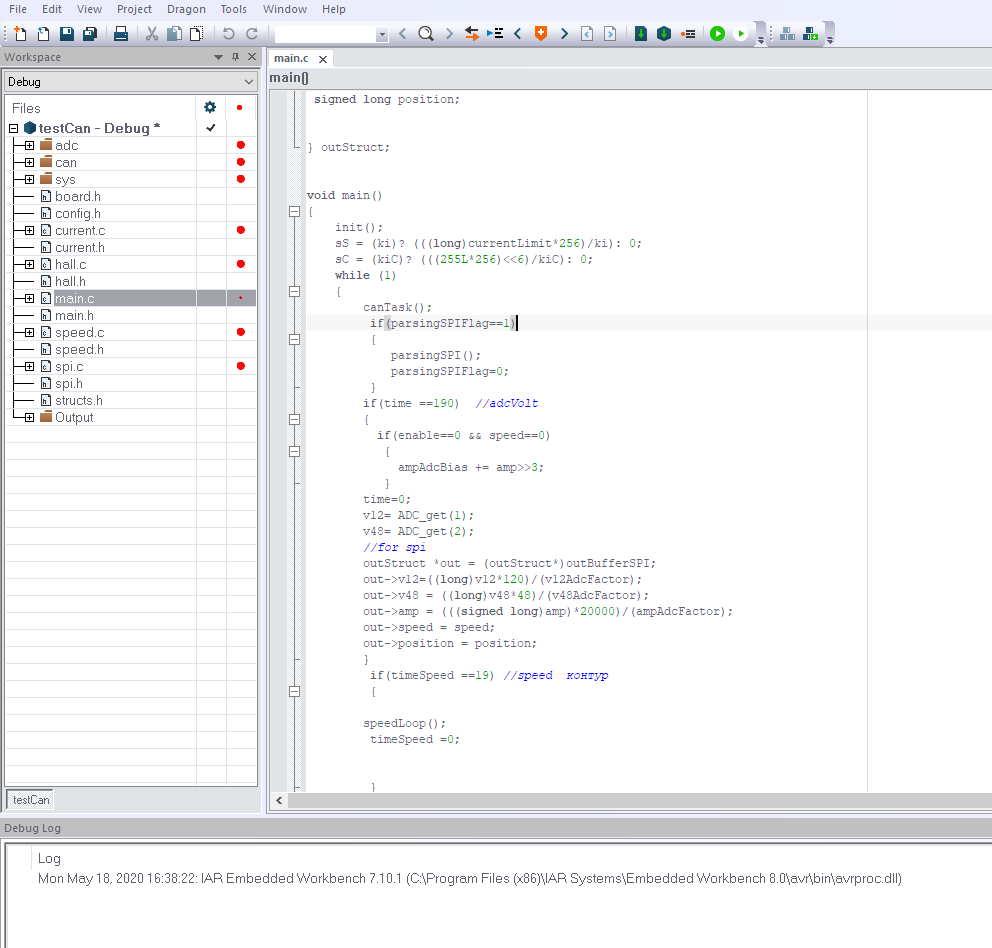
|  |
| --- |
| 1. void timer0\_ISR(void) interrupt 1 using 0 2. {   // обработчик прерывания по таймеру 4. button\_handler(); 6. if (ms\_counter % ISR\_DURATION\_2 == 0) 7. reveal\_curr\_number(ms\_counter); 8. ms\_counter+=ISR\_DURATION; 9. } |

В обработчике прерываний находится функция обработчики нажатия кнопок клавиатуры стенда button\_handler(), а так же основная функция программы для отображения двоичных чисел в визуальном и звуковом представлении reveal\_curr\_number(ms\_counter). Функция reveal\_curr\_number запускается каждые ISR\_DURATION = 25 мс.

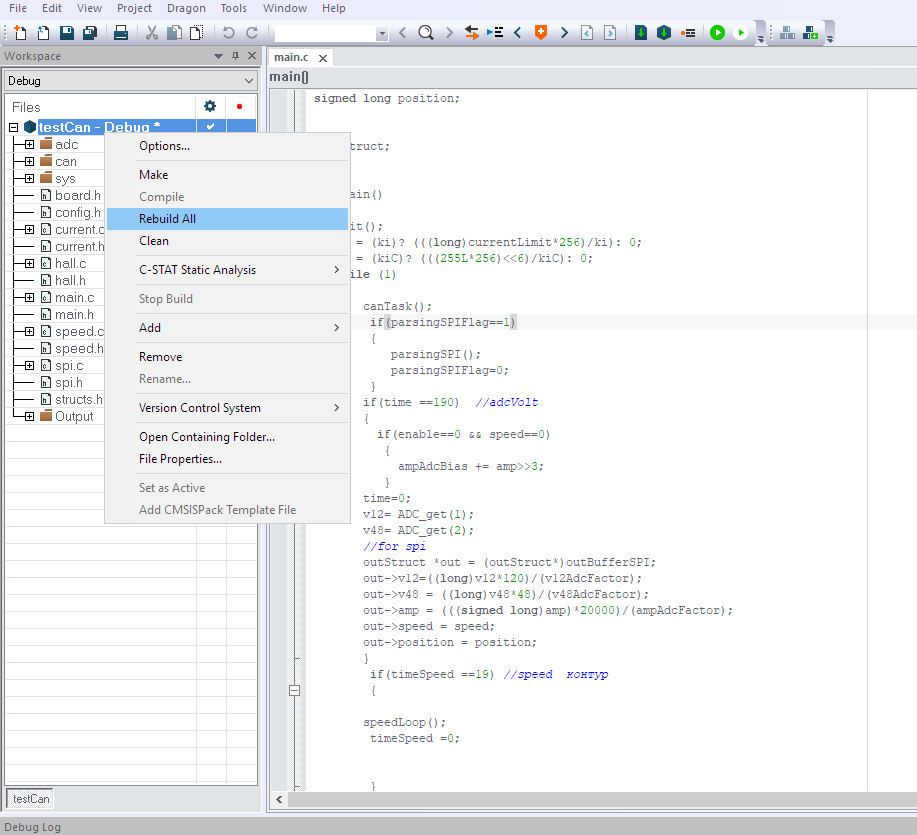
**Часть 2. Ознакомление с программно-аппаратными средствами составления и отладки программ на базе RISC микроконтроллеров семейства AVR в системе реального времени**

1) Описание процесса подготовки, прошивки контроллера и отладки программ.

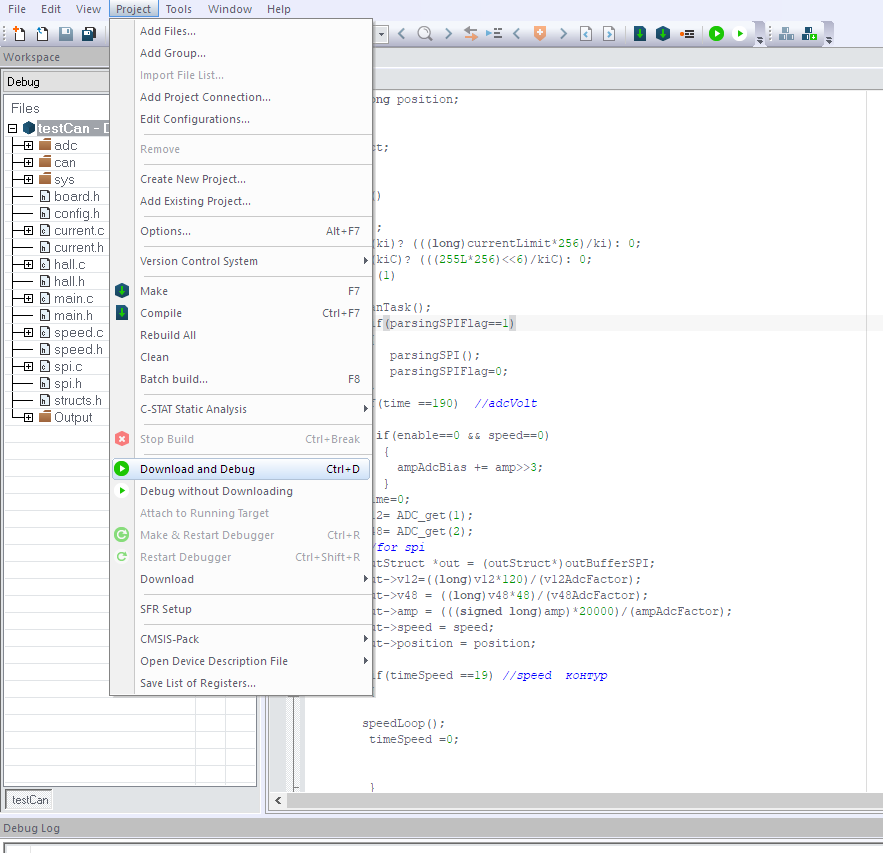
Программа разрабатывается в среде IAR Embedded Workbench IDE.

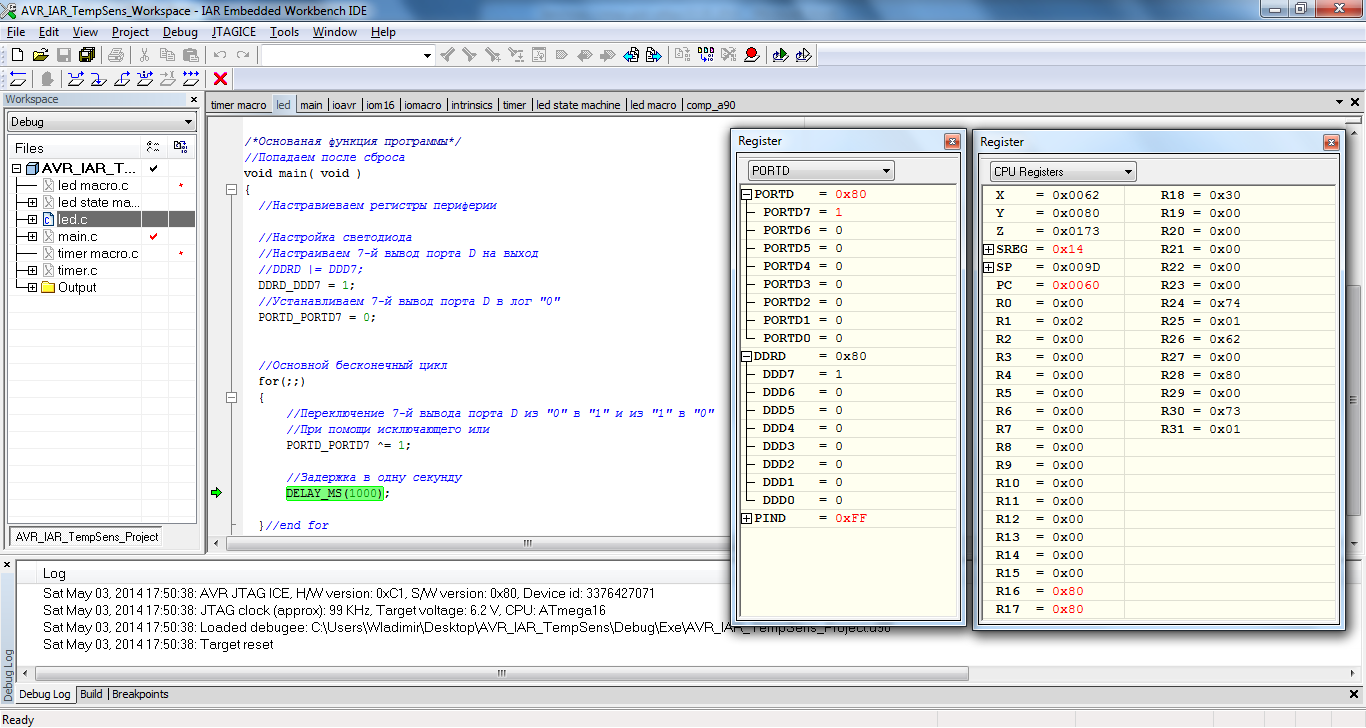


Для компиляции проекта нужно собрать проект, для этого нужно выполнить перестройку проекта (Rebuild All).

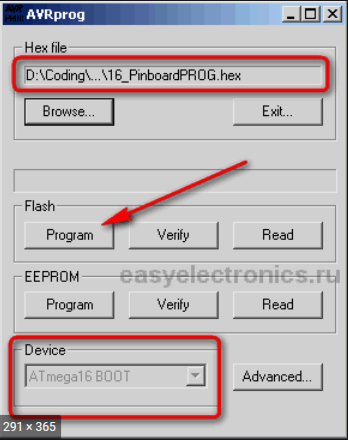


Так же для отладки кода и его дизассемблирования можно использовать режим отладки





После успешного написания текста программы, созданный файл машинных кодов нужно записать в контроллер, для этого используем программу AVRprog



Порядок действий

а) Выбрать устройство

б) выбрать путь к файлу

в) нажать кнопку Program

2) Программа на базе микроконтроллера семейства AVR

**Формулировка задачи**

1) Вывести название режима - ClockMode на LCD дисплей;

2) Выводить текущее время на LCD дисплей, полученное из микросхемы DS1307

   (информация на дисплее обновляетя каждые 250 мс - 4 Гц);

3) При нажатии # включается режим изменения времени - пользователю нужно последовательно

   ввести час, минуты, секунды;

4) После изменения текущего времени снова включается режим отображения.

Текст программы приведен в Приложении 2.

Все процессы обрабатываются практически одновременно, так как система является системой реального времени, работа осуществляется в прерываниях.

Для обработки событий, происходящих асинхронно по отношению к выполнению программы, лучше всего подходит механизм прерываний. Прерывание можно рассматривать как некоторое особое событие в системе, требующее моментальной реакции.

3) Реализация прерываний в программе

**3.1) Прерывания по таймеру**

1) Расчет частоты таймера

В учебных целях воспользуемся таймером 1 и настроим его на 2 прерывания с разными частотами.

Для асинхронного режима используем режим CTC – сброс по совпадению, задаем два значения компараторов A и B для реализации частоты в 1000 Гц и 40 Гц.

Частота atmega16 равна 8МГц. Выбрав переделить 8, получим частоту 1МГц, далее для получения частоты 1000 Гц на регистре компаратора A выставляем значение OCR1A = 1000, а для частоты 40 Гц на регистре компаратора B выставляем значение OCR1B = 25000.

2) Инициализация таймера

|  |
| --- |
| 1. void init\_timer1() {      // Настройка таймера 1 (частота atmega16 - 8МГц) 2. // Настройка компоратора B для оргнаизации миллисекундного таймера 1000 Гц (1 мс) 3. // и обновления дисплея с частотой 40 Гц (25 мс) 4. TCCR1B = 0;     // Очистить регистр управления компаратором 1В 5. TCCR1B |= (1 << TCCR1B\_WGM12); // устанавливаем режим СТС (сброс по совпадению) 6. TCCR1B |= (1 << TCCR1B\_CS11); // Предделитель 8 7. OCR1A = 1000;         // Значение для достижения 1000 Гц (1 мс) компаратор А 8. // Значение для достижения 40 Гц (25 мс) копаратор В 9. TIMSK\_OCIE1A |= 1;     // Запуск компаратора таймера А 10. TIMSK\_OCIE1B |= 1;     // Запуск компаратора таймера В 11. } |

3) Обработчики прерываний по таймеру

|  |
| --- |
| 1. #pragma vector = TIMER1\_COMPA\_vect 2. \_\_interrupt void Timer1\_COMPA(void) 3. { 4. ms\_ctr++; 5. } 6. #pragma vector = TIMER1\_COMPB\_vect 7. \_\_interrupt void Timer1\_COMPB(void) 8. { 9. if ( is\_time\_change\_mode == false ) // выводим текущее время 10. display\_time();                 // если не находимя в режиме изменения времени 11. } |

**Приложение 1 для микроконтроллера с архитектурой семейства х51**

#include <stdio.h>

#include <intrins.h>

#include <string.h>

#include "PINDEF.H"

#include "KEYBOARD.H"

#include "SPI.H"

#include "LCD\_HD44780.H"

#include "DELAY.H"

#include "LED7.H"

#include "DS1307.H"

#include "DS1820.H"

#include "AT24CXX.H"

#include "BEEP.H"

#include "I2C.H"

#include "SHIFT.H"

/\*

Студент:    Балкунов К.С.

Группа:     СМ7-74Б

Формулировка задачи:

    1) Вывести цифры от 0 до 7 десятичной системы счисления, используя клавиатуру стенда,

        в двоичном представолении, используя светодиоды стенда.

    2) Параллеьно с включением светодиодов, воспроизвести динамиком двоичное число

        по следущем правилам:

            \* Динный сигнал соответствует биту 1, короткий - 0.

            \* Число воспроизводится от младшего бита к старшему.

    3) При нажатии \* светодиоды выключаются. Динамик при этом перестаёт издавать звуки.

\*/

#define CLOCK 22110000

#define TIMER\_FREQ\_HZ 1000

#define RELOAD\_VALUE (65536 - (CLOCK / 12 / TIMER\_FREQ\_HZ))

#define RELOAD\_VALUE\_H ((unsigned char)(RELOAD\_VALUE >> 8))

#define RELOAD\_VALUE\_L ((unsigned char)(RELOAD\_VALUE))

/\*Что бы не запутаться и учесть то, что МК осуществляет управление выводом отрицательной

полярности добавим макросы ВКЛЮЧЕНИЯ и ВЫКЛЮЧЕНИЯ светодиодов\*/

#define HIGH 0       // сигнал для подачи высокого уровня на выход для включения светодиода

#define LOW 1        // сигнал для подачи низкого уровня на выход для выключения светодиода

#define BITS\_COUNT 3            // кол-во битов двоичного числа

#define LEDS\_COUNT BITS\_COUNT   // количество светодиодов подключенных к пинам контроллера на стенде

#define BITS\_COUNT\_MINUS\_1 BITS\_COUNT - 1

#define BEEP\_DURATION 250       // зажкржка между сигналами

#define BEEP\_HIGH 500           // длительность сигнала соответствующему биту 1

#define BEEP\_LOW 250            // длительность сигнала соответствующему биту 0

#define ISR\_DURATION 25

#define ISR\_DURATION\_2 ISR\_DURATION\*2

#define FALSE 0

#define TRUE 1

char led\_pins\_signals[][LEDS\_COUNT] = {

    {LOW, LOW, LOW},    // 000

    {HIGH, LOW, LOW},   // 001

    {LOW, HIGH, LOW},   // 010

    {HIGH, HIGH, LOW},  // 011

    {LOW, LOW, HIGH},   // 100

    {HIGH, LOW, HIGH},  // 101

    {LOW, HIGH, HIGH},  // 110

    {HIGH, HIGH, HIGH}, // 111

};

unsigned int ms\_counter = 0;    // счетчик миллисекнуд

unsigned char key = 0;   // Адрес клавиши

int curr\_key\_number = 0; // цифра по адресу клавиши

char is\_reveal\_process\_started = FALSE; // флажок процесса проигрывания числа и отображения

char need\_duration = FALSE;     // флажок задержки между проигрыванием битов

void init\_timer0(void)

{// инициализация таймера 0

    TMOD = 0x01; // 16-битовый режим

    TL0 = RELOAD\_VALUE\_L;

    TH0 = RELOAD\_VALUE\_H;

    ET0 = 1; // Бит разрешения прерывания от таймера 0

    TR0 = 1; // Бит управления таймера 0 для пуска/останова таймера/счетчика

}

void button\_handler() {

    // Обработчик нажатия кнопок клавиатуры

    key = KEY\_getkey();                         // подлучаем адрес нажатой клавиши

    curr\_key\_number = KEY\_getkeynumber(key);    // получаем цифру по адресу клавиши

    if (key == KEY\_ASTERISK) {                  //При нажатии \* светодиоды выключаются.

        is\_reveal\_process\_started = false;      //Динамик при этом перестаёт издавать звуки.

        return;

    }

    if (curr\_key\_number >= 0 && curr\_key\_number < 8)

        is\_reveal\_process\_started = true;

}

void reveal\_curr\_number(unsigned int ms\_counter)

{

    // Вывести цифры от 0 до 7 десятичной системы счисления в двоичном представолении,

    // через светодиоды стенда.

    static int singed\_bits\_counter = 0;     // счетчик сыгнранных зуммером битов двоичного числа

    static unsigned int local\_counter = 0;  // миллисекндный счетчик для задержки между сигналами

    char is\_bit\_singed = FALSE;             // проигран ли один из битов двоичного числа?

    // если нажата \* (звездочка) выключаем все светодиоды и зуммер

    if (key == KEY\_ASTERISK)

    {

        LED1 = LOW;

        LED2 = LOW;

        LED3 = LOW;

        singed\_bits\_counter = 0;

        BEEP\_BIT = LOW;

        need\_duration = false;

    }

    // если нажата цифра в диапоне от 0 до 7,

    // то зажигаем светодиоды в соответсвии с битами двоичного числа

    if (curr\_key\_number >= 0 && curr\_key\_number < 8 && is\_reveal\_process\_started == true)

    {

        set\_led\_pins\_signals(curr\_key\_number);          // включить светодиоды в соответсвтии

                                                        // с двоичным числом

        // вопспроизвести динамиком бит двочиного числа

        is\_bit\_singed = sing\_binary\_bit\_by\_zoomer(curr\_key\_number, singed\_bits\_counter, ms\_counter);

        if (is\_bit\_singed)

            singed\_bits\_counter++;

    }

    if (singed\_bits\_counter == BITS\_COUNT\_MINUS\_1) {    // если сыграли зуммером все биты сбрасываем счетчик,

        singed\_bits\_counter = 0;                        // сбрасываем флаг is\_reveal\_process\_started на 0

        is\_reveal\_process\_started = false;

    }

}

void set\_led\_pins\_signals(unsigned int curr\_number)

{

    // Установить уровни сигналов пинов светодиодов в соответсвтии с двоичным числом

    LED1 = led\_pins\_signals[curr\_number][0];

    LED2 = led\_pins\_signals[curr\_number][1];

    LED3 = led\_pins\_signals[curr\_number][2];

}

char sing\_binary\_bit\_by\_zoomer(unsigned int curr\_number, unsigned int curr\_bit, unsigned int ms\_counter)

{

    // Вопспроизвести динамиком (зумером) двочиное число.

    // Динный сигнал соответствует биту 1, короткий - 0.

    // Число воспроизводится от младшего бита к старшему.

    static unsigned int local\_counter = 0;

    char is\_bit\_singed = FALSE; // возвращаемое значение

    if (led\_pins\_signals[curr\_number][curr\_bit] == HIGH && need\_duration == false) {

        BEEP\_BIT = HIGH;            // если текущий бит двочиного числа равен 1,

                                // то включаем зумер на время BEEP\_HIGH.

        if (ms\_counter - local\_counter > BEEP\_HIGH) {

            local\_counter = ms\_counter;

            BEEP\_BIT = LOW;

            need\_duration = true;

        }

    }

    else

        BEEP\_BIT = HIGH;            // иначе включаем зумер на время BEEP\_LOW

        if (ms\_counter - local\_counter > BEEP\_LOW && need\_duration == false) {

            local\_counter = ms\_counter;

            BEEP\_BIT = LOW;

            need\_duration = true;

        }

    // задержка между сыгранными битами величиной в BEEP\_DURATION

    if (ms\_counter - local\_counter > BEEP\_DURATION && need\_duration == true) {

        need\_duration = false;

        is\_bit\_singed = true;

    }

    return is\_bit\_singed;

}

void timer0\_ISR(void) interrupt 1 using 0

{   // обработчик прерывания по таймеру

    button\_handler();

    if (ms\_counter % ISR\_DURATION\_2 == 0)

        reveal\_curr\_number(ms\_counter);

    ms\_counter+=ISR\_DURATION;

}

// главная функция

void main()

{

    // Настраиваем порты на вывод

    P0 = 0xFF;

    P1 = 0xFF;

    P2 = 0xFF;

    P3 = 0xFF;

    EA = 1;  // Разрешаем прерывания

    while (1)

    {

        /\* бсконечный цикл \*/

    }

}

**Приложение 2 для микроконтроллера с архитектурой семейства AVR**

#include <stdbool.h>

#include "SPI.H"

#include "LCD\_HD44780.H"

#include "I2C.H"

#include "KEYBOARD.H"

#include "LED7.H"

#include "UART.H"

#include "DS1307.H"

#include "DS1820.H"

#include "AT24CXX.H"

#include "CONVERT.H"

#include "PINDEF.h"

/\*

Студент:    Балкунов К.С.

Группа:     СМ7-74Б

Формулировка задачи:

1) Вывести название режима - ClockMode на LCD дисплей;

2) Выводить текущее время на LCD дисплей, полученное из микросхемы DS1307

   (информация на дисплее обновляетя каждые 250 мс - 4 Гц);

3) При нажатии # клавиатуры стенда включается режим изменения времени - пользователю нужно последовательно

   ввести час, минуты, секунды;

4) После изменения текущего времени снова включается режим отображения.

\*/

#define DISPLAY\_DELAY 500 // [мс], время отображения одного кадра на LCD дисплее

flash char UartMessageClockMode[] = "ClockMode\r\n";

flash char timeStr[] = "Time: ";

flash char enterNewTimeStr[] = "Enter new time: ";

flash char enterHourStr[] = "Enter hour: ";

flash char enterMinuteStr[] = "Enter minute: ";

flash char enterSecondStr[] = "Enter second: ";

bool is\_time\_change\_mode = false;   // флаг режима изменения времени

unsigned char key = 0;              // ключ символа нажатой кнопки клавиатуры

unsigned int ms\_ctr = 0;            // счетчик миллисекунд от начала работы программы

// установить новое время в микросхему DS1307 через UART-интерфейс

void set\_new\_time()

{

    unsigned char hour, minute, second;

    UART\_sendstring\_flash( enterHourStr );

    hour = UART\_receivevalue();

    UART\_sendstring\_flash( enterMinuteStr );

    minute = UART\_receivevalue();

    UART\_sendstring\_flash( enterSecondStr );

    second = UART\_receivevalue();

    DS1307\_settime( hour, minute, second );

    is\_time\_change\_mode = false;

}

// вывести текущее время на LCD дисплей

void display\_time()

{

    LCD\_clrscr();

    char time[9];

    DS1307\_gettime(time); // получить текущее вермя в массив time

    UART\_sendstring\_flash( timeStr );

    UART\_sendstring( time ); // вывести время в консоль

    LCD\_gotoxy( 0, 0 );

    LCD\_print(time);        // вывести время на ЖК дисплей

}

void init\_timer1() {                                     // Настройка таймера 1 (частота atmega16 - 8МГц)

    // Настройка компоратора B для оргнаизации миллисекундного таймера 1000 Гц (1 мс)

    // и обновления дисплея с частотой 40 Гц (25 мс)

    TCCR1B = 0;                                          // Очистить регистр управления компаратором 1В

    TCCR1B |= (1 << TCCR1B\_WGM12);                       // устанавливаем режим СТС (сброс по совпадению)

    TCCR1B |= (1 << TCCR1B\_CS11);                        // Предделитель 8

    OCR1A = 1000;                                        // Значение для достижения 1000 Гц (1 мс) компаратор А

    OCR1B = 25000;                                       // Значение для достижения 40 Гц (25 мс) копаратор В

    TIMSK\_OCIE1A |= 1;                                   // Запуск компаратора таймера А

    TIMSK\_OCIE1B |= 1;                                   // Запуск компаратора таймера В

}

#pragma vector = TIMER1\_COMPA\_vect

\_\_interrupt void Timer1\_COMPA(void)

{

    ms\_ctr++;

}

#pragma vector = TIMER1\_COMPB\_vect

\_\_interrupt void Timer1\_COMPB(void)

{

    if ( is\_time\_change\_mode == false ) // выводим текущее время

        display\_time();                 // если не находимя в режиме изменения времени

}

void button\_handler() {     // обработчик кнопки

    if (ms\_ctr % 10 == 0)   // опрашиваем клавиатуру раз в 10 мс

        key = KEY\_getkey();

}

void change\_mode() {

    if (key == KEY\_BAR) //если нажата # активируем флажок изменения времени

        is\_time\_change\_mode = true;

}

int main()

{

    I2C\_init();

    SPI\_init();

    LCD\_init();

    LCD\_clrscr();

    UART\_init( CALC\_UBRR( 57600 ) );

    \_\_enable\_interrupt();

    UART\_sendstring\_flash( UartMessageClockMode );

    OutputStartLCDMessage();

    while(1)

    {

        button\_handler();   // читаем данные с клавиатуры

        change\_mode();      // устанавливаем режим изменения времени, если была нажата #

        if (is\_time\_change\_mode == true) {

            set\_new\_time();

            is\_time\_change\_mode == false;

        }

    }

}