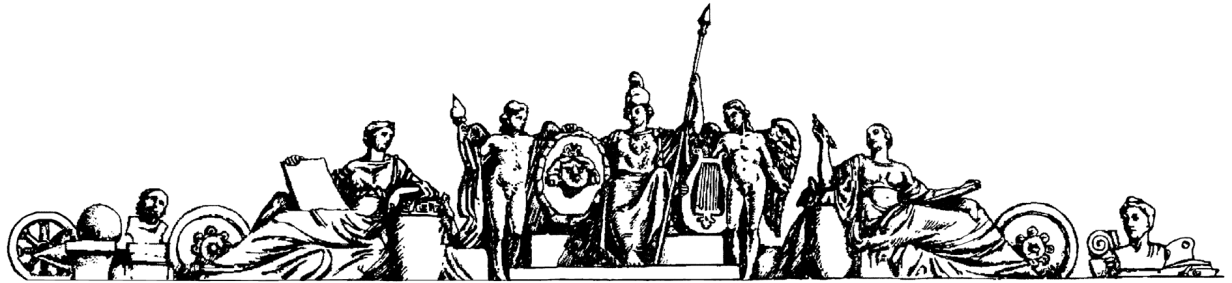
Федеральное агентство по образованию

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

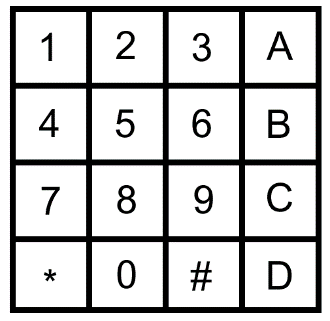
1. Светодиодный индикатор

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

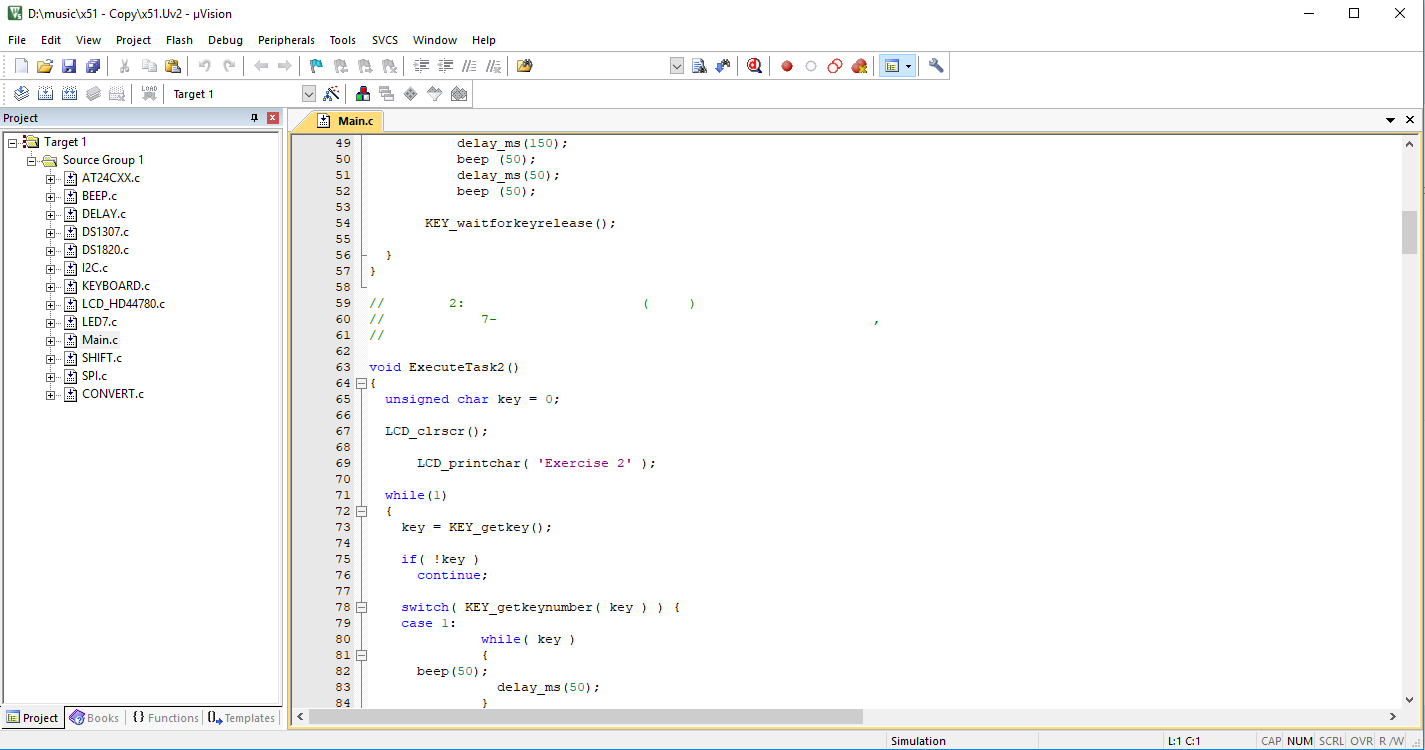
В режиме 0 частота передачи зависит только от резонансной частоты кварцевого резонатора fрез: F = fРЕЗ / 12

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.

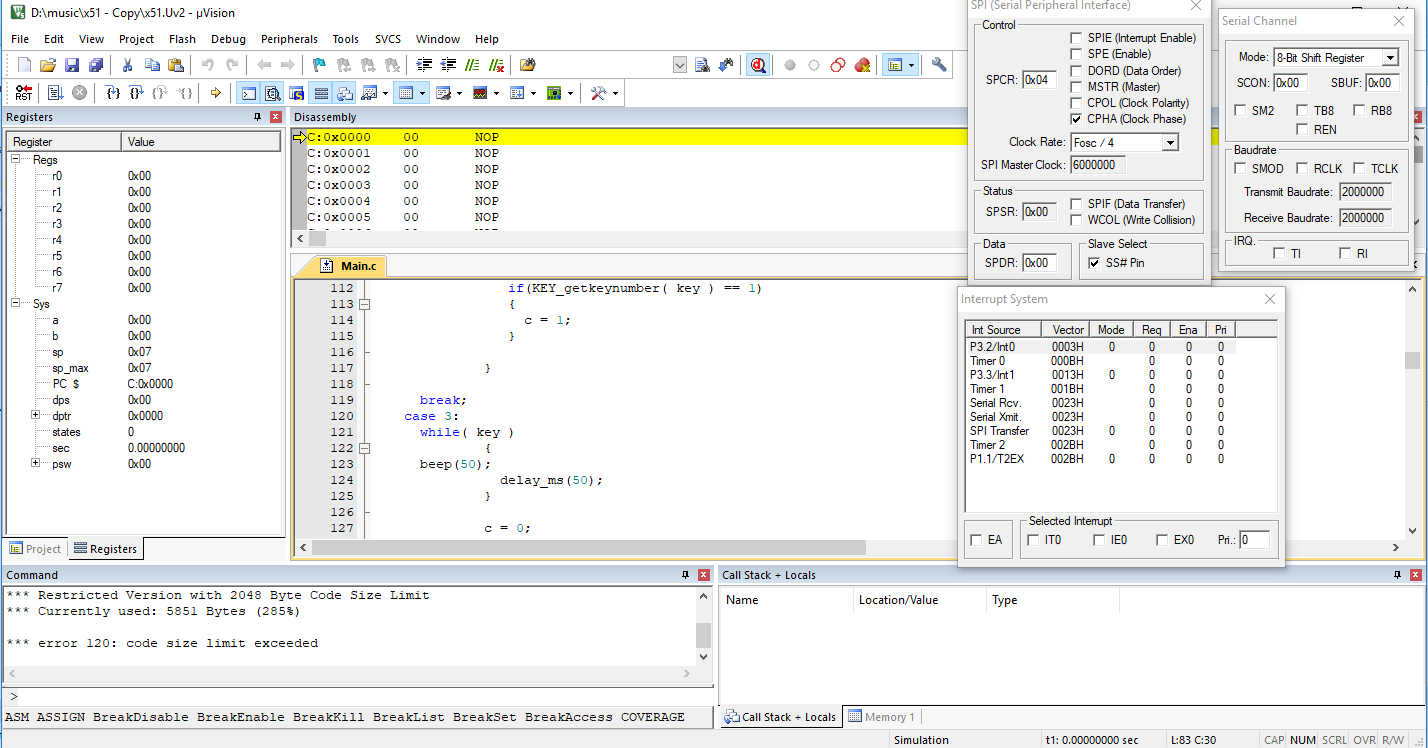
**Часть 1. Ознакомление с программно-аппаратными средствами составления и отладки программ на базе CISC микроконтроллеров семейства х51 в системе реального времени**

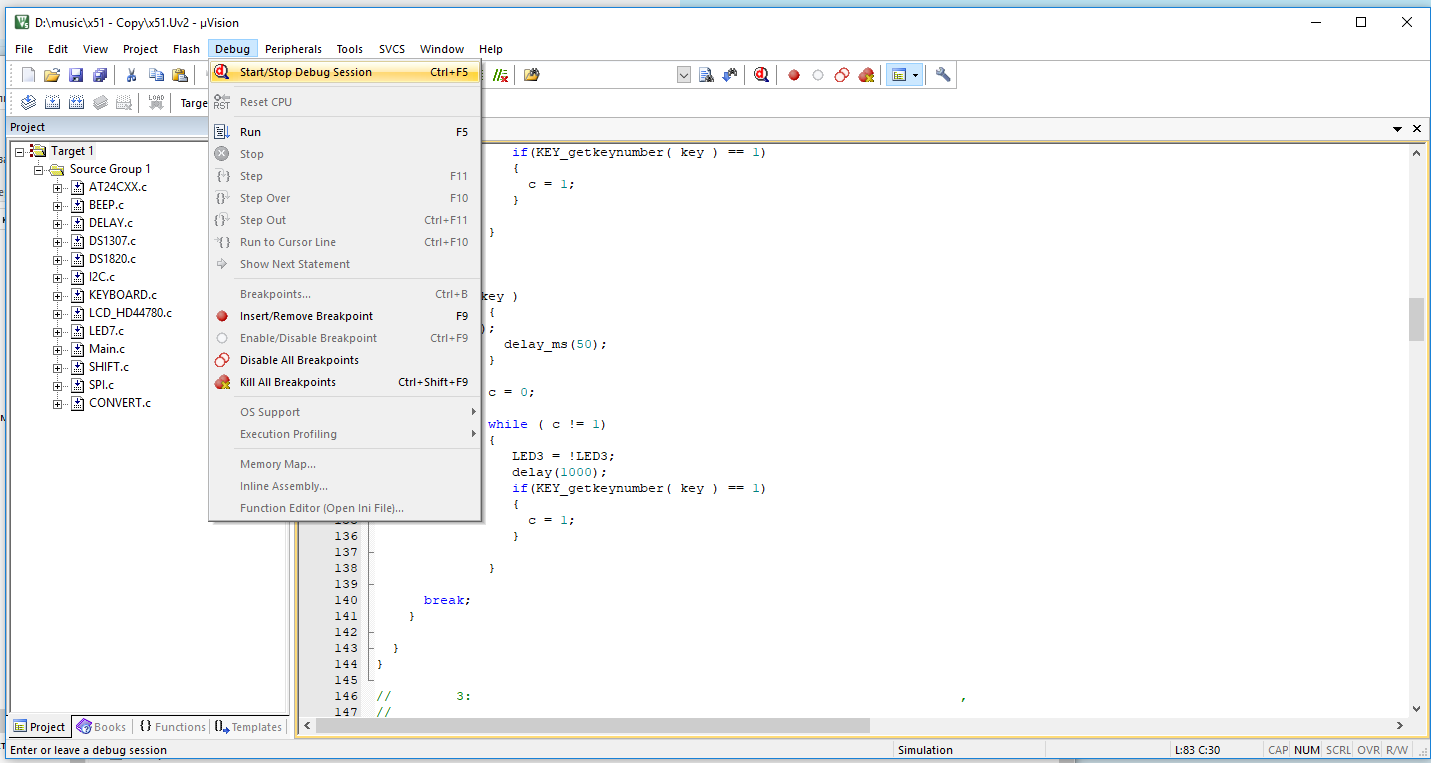
1) Описание процесса подготовки, прошивки контроллера и отладки программ

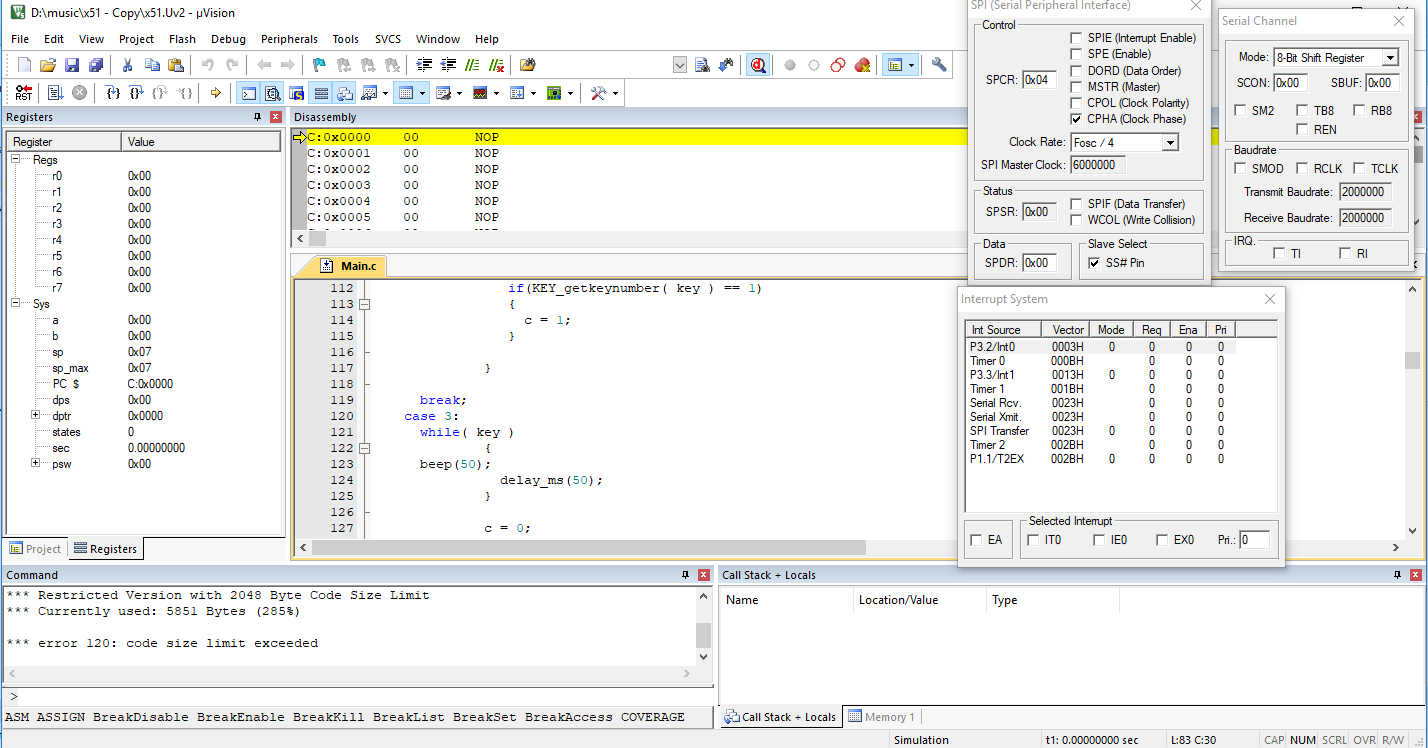
Программа разрабатывается в среде µVision2



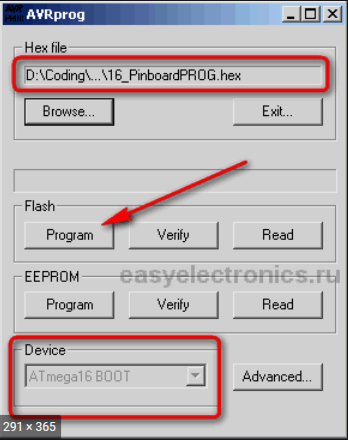
Для компиляции проекта нужно собрать проект, для этого нужно выполнить перестройку проекта.



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

**3.1) 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.

**3.2) Описание основных подпрограмм**

Функция **button\_handler()** служит для вызова других функций при наступлении определённых событий: нажата \* или нажата цифра от 0 до 8.

1. void button\_handler() {
2. // Обработчик нажатия кнопок клавиатуры
3. key = KEY\_getkey();                         // подлучаем адрес нажатой клавиши
4. curr\_key\_number = KEY\_getkeynumber(key);    // получаем цифру по адресу клавиши
5. if (key == KEY\_ASTERISK)                    //При нажатии \* светодиоды выключаются.
6. stop\_revealing();                       //Динамик при этом перестаёт издавать звуки.
8. if (curr\_key\_number >= 0 && curr\_key\_number < 8) {
9. is\_reveal\_process\_started = TRUE;
10. dur\_counter = ms\_counter; //обновляем счетчик для генерации сигналов зумера
11. }
12. }

Функция **reveal\_curr\_number()** реализует проигрывание зуммером и вывод битов двоичного числа светодиодами. Для того что бы не терять текущий проигрываемый зуммером бит введен глобальный счетчик текущего проигрываемого бита **singed\_bits\_counter** . Если бит полностью сыгран зуммером счетчик – увеличиваем счетчик на 1, при переполнении (более 3 битов) – сбрасываем до нуля и завершаем процесс приомгрывания двоичного числа is\_reveal\_process\_started = FALSE.

1. void reveal\_curr\_number()
2. {
3. // Вывести цифры от 0 до 7 десятичной системы счисления в двоичном представолении,
4. // через светодиоды стенда.
6. char is\_bit\_singed = FALSE; // проигран ли один из битов двоичного числа?
7. // если нажата цифра в диапоне от 0 до 7,
8. // то зажигаем светодиоды в соответсвии с битами двоичного числа
9. if (curr\_key\_number >= 0 && curr\_key\_number < 8 && is\_reveal\_process\_started == TRUE)
10. {
11. set\_led\_pins\_signals(curr\_key\_number);          // включить светодиоды в соответсвтии
12. // с двоичным числом
14. // вопспроизвести динамиком бит двочиного числа
15. is\_bit\_singed = sing\_binary\_bit\_by\_zoomer(curr\_key\_number, singed\_bits\_counter);
16. if (is\_bit\_singed)
17. singed\_bits\_counter++;
18. }
19. if (singed\_bits\_counter == BITS\_COUNT\_MINUS\_1) {    // если сыграли зуммером все биты сбрасываем счетчик битов,
20. singed\_bits\_counter = 0;                        // сбрасываем флаг is\_reveal\_process\_started на 0
21. is\_reveal\_process\_started = FALSE;
22. }
23. }

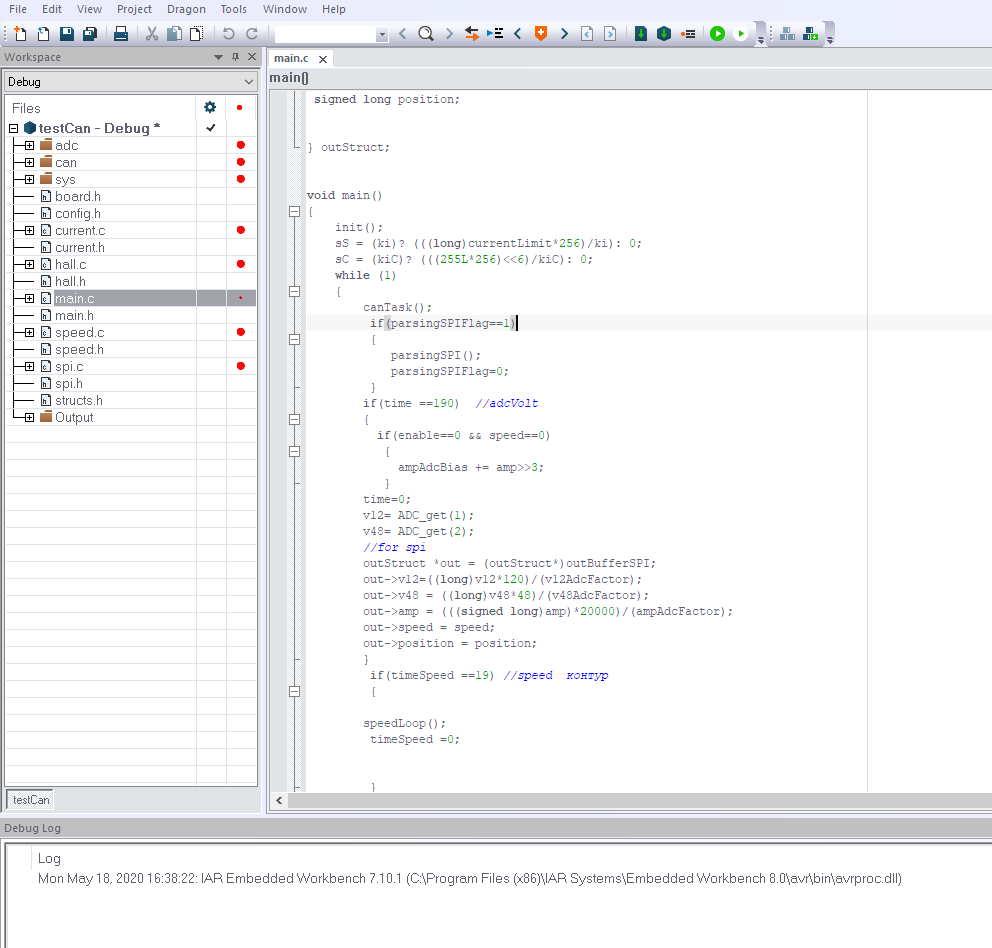
Особую сложность представляет функция контроля зуммера **sing\_binary\_bit\_by\_zoomer** так как на каждом тике системы реального времени мы не должны потерять текущую фазу выполнения длительного процесса проигрывания. Для контроля фаз работы зуммера используется флаг **need\_duration,** а для задания задержки между сигналами – счетчик задержки **dur\_counter,** который обновляет свое значение только при определенных условиях – в теле функции **sing\_binary\_bit\_by\_zoomer** (см. if конструкции) и при нажатии очередной цифры.

1. char sing\_binary\_bit\_by\_zoomer(unsigned int curr\_number, unsigned int curr\_bit)
2. {
3. // Вопспроизвести динамиком (зумером) двочиное число.
4. // Динный сигнал соответствует биту 1, короткий - 0.
5. // Число воспроизводится от младшего бита к старшему.
6. char is\_bit\_singed = FALSE; // возвращаемое значение
8. if (led\_pins\_signals[curr\_number][curr\_bit] == HIGH && need\_duration == FALSE) {
9. BEEP\_BIT = HIGH;            // если текущий бит двочиного числа равен 1,
10. // то включаем зумер на время BEEP\_HIGH.
11. if (ms\_counter - dur\_counter > BEEP\_HIGH) {
12. dur\_counter = ms\_counter;
13. BEEP\_BIT = LOW;
14. need\_duration = TRUE;
15. }
16. }
17. else
18. BEEP\_BIT = HIGH;            // иначе включаем зумер на время BEEP\_LOW
19. if (ms\_counter - dur\_counter > BEEP\_LOW && need\_duration == FALSE) {
20. dur\_counter = ms\_counter;
21. BEEP\_BIT = LOW;
22. need\_duration = TRUE;
23. }
24. // задержка между сыгранными битами величиной в BEEP\_DURATION
25. if (ms\_counter - dur\_counter > BEEP\_DURATION && need\_duration == TRUE) {
26. need\_duration = FALSE;
27. is\_bit\_singed = TRUE;
28. }
29. return is\_bit\_singed;
30. }

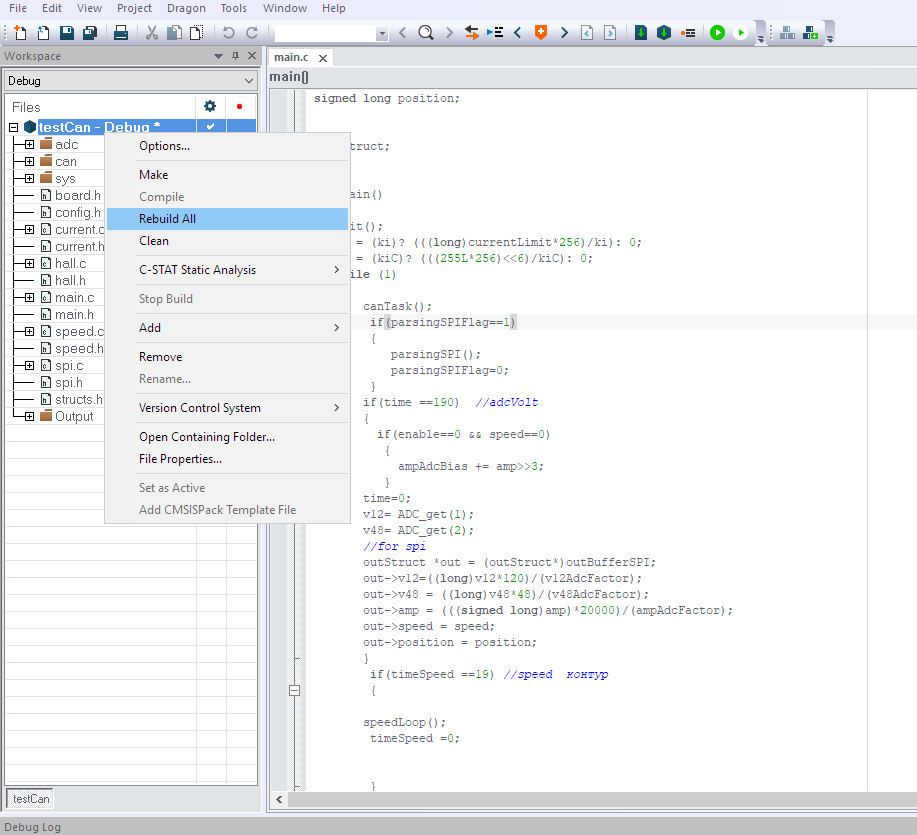
**Часть 2. Ознакомление с программно-аппаратными средствами составления и отладки программ на базе RISC микроконтроллеров семейства AVR в системе реального времени**

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

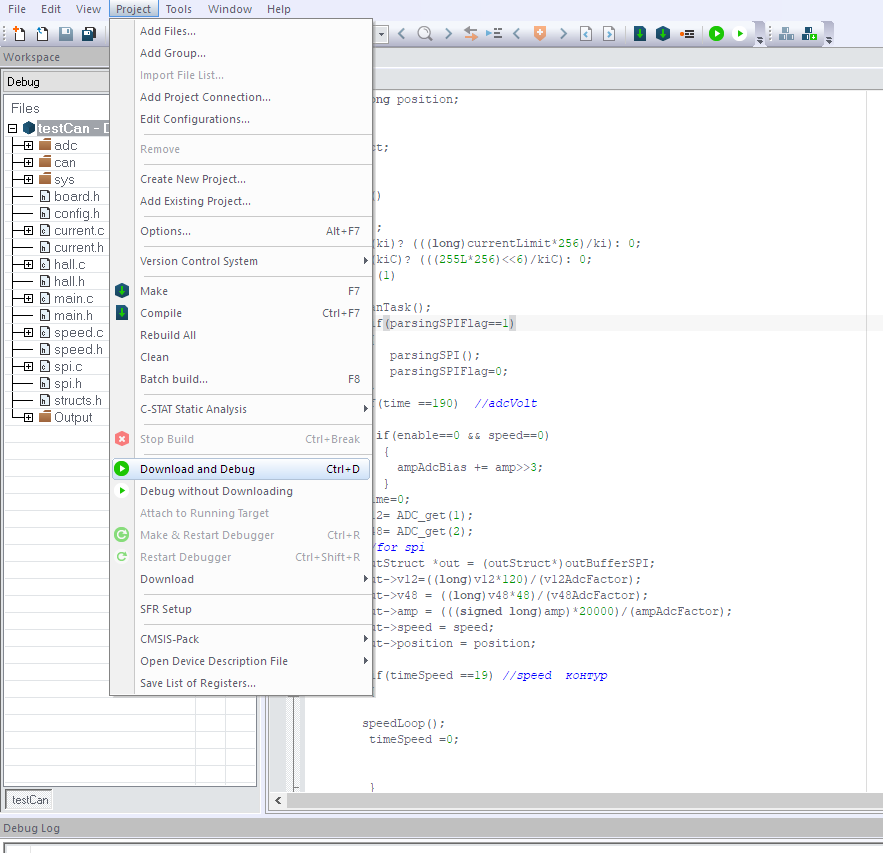
Программа разрабатывается в среде IAR Embedded Workbench IDE.

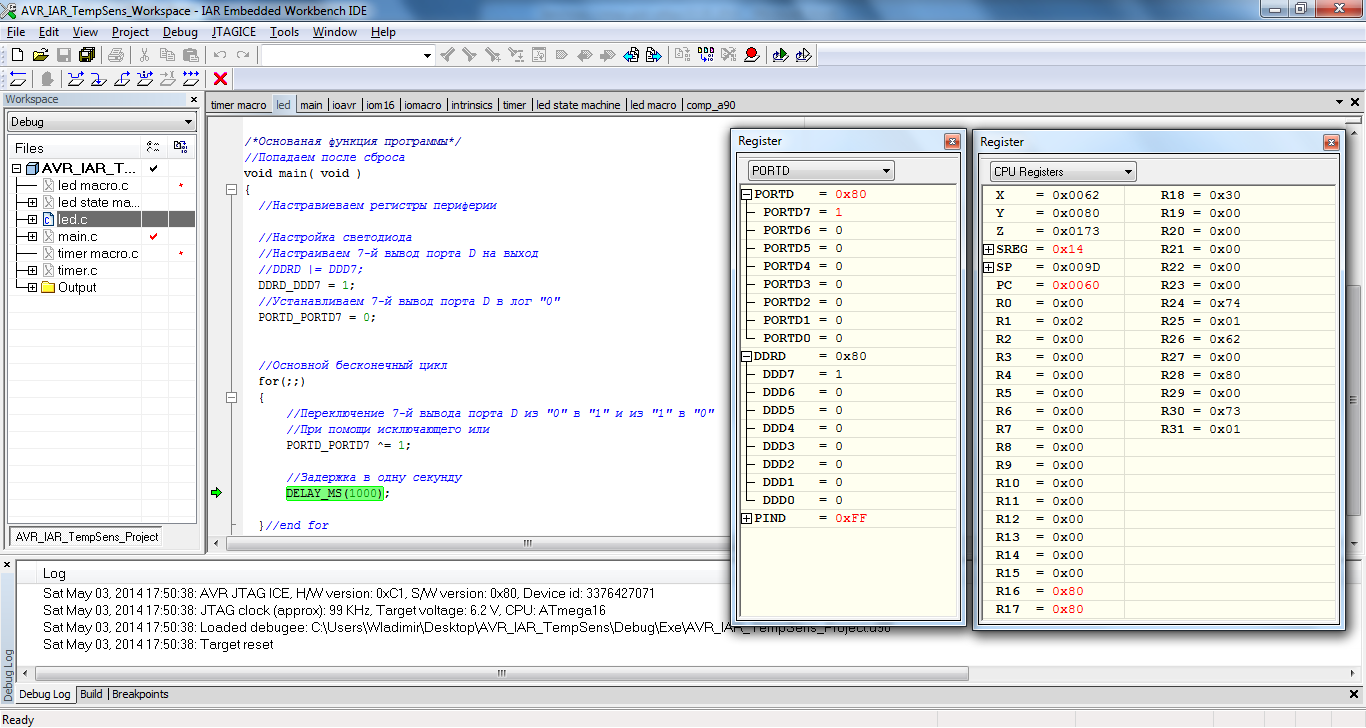


Для компиляции проекта нужно собрать проект, для этого нужно выполнить перестройку проекта (Rebuild All).

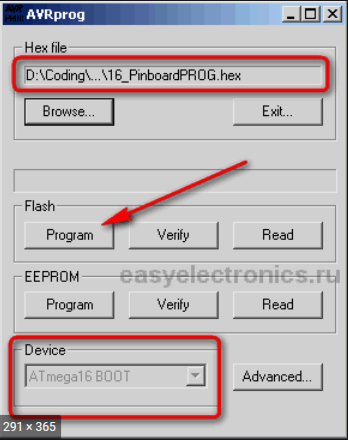


Так же для отладки кода и его дизассемблирования можно использовать режим отладки





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 # клавиатуры стенда включается режим изменения времени - пользователю нужно последовательно ввести час, минуты, секунды через клавиатуру ПК;

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.

Прерывание по компаратору А будет использовано для обработки нажатий кнопок клавиатуры стенда, по компаратору B – для обновления времени на ЖК дисплее.

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

В обработчике прерывания по компаратору А осуществляется приращение глобальной переменной-счётчика миллисекунд.

В обработчике прерывания по компаратору В происходит обновление данных времени/даты на ЖК дисплее с частотой 40 Гц.

|  |
| --- |
| 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. } |

**3.1) Описание основных подпрограмм**

В основном цикле программы происходит обработка нажатий клавиш клавиатуры стенда, проверка нажатия # и запуск подпрограммы установки текущего времени/даты.

1. while(1)
2. {
3. button\_handler();   // читаем данные с клавиатуры
4. change\_mode();      // устанавливаем режим изменения времени, если была нажата #
5. if (is\_time\_change\_mode == TRUE) {
6. set\_new\_time();
7. is\_time\_change\_mode == FALSE;
8. }
9. }

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

1. #include <stdio.h>
2. #include <intrins.h>
3. #include <string.h>
4. #include "PINDEF.H"
5. #include "KEYBOARD.H"
6. #include "SPI.H"
7. #include "LCD\_HD44780.H"
8. #include "DELAY.H"
9. #include "LED7.H"
10. #include "DS1307.H"
11. #include "DS1820.H"
12. #include "AT24CXX.H"
13. #include "BEEP.H"
14. #include "I2C.H"
15. #include "SHIFT.H"
16. /\*
17. Студент:    Балкунов К.С.
18. Группа:     СМ7-74Б
19. Формулировка задачи:
20. 1) Вывести цифры от 0 до 7 десятичной системы счисления, используя клавиатуру стенда,
21. в двоичном представолении, используя светодиоды стенда.
22. 2) Параллеьно с включением светодиодов, воспроизвести динамиком двоичное число
23. по следущем правилам:
24. \* Динный сигнал соответствует биту 1, короткий - 0.
25. \* Число воспроизводится от младшего бита к старшему.
26. 3) При нажатии \* светодиоды выключаются. Динамик при этом перестаёт издавать звуки.
27. \*/
28. #define CLOCK 22110000
29. #define TIMER\_FREQ\_HZ 1000
30. #define RELOAD\_VALUE (65536 - (CLOCK / 12 / TIMER\_FREQ\_HZ))
31. #define RELOAD\_VALUE\_H ((unsigned char)(RELOAD\_VALUE >> 8))
32. #define RELOAD\_VALUE\_L ((unsigned char)(RELOAD\_VALUE))
33. /\*Что бы не запутаться и учесть то, что МК осуществляет управление выводом отрицательной
34. полярности добавим макросы ВКЛЮЧЕНИЯ и ВЫКЛЮЧЕНИЯ светодиодов\*/
35. #define HIGH 0       // сигнал для подачи высокого уровня на выход для включения светодиода
36. #define LOW 1        // сигнал для подачи низкого уровня на выход для выключения светодиода
37. #define BITS\_COUNT 3            // кол-во битов двоичного числа
38. #define LEDS\_COUNT BITS\_COUNT   // количество светодиодов подключенных к пинам контроллера на стенде
39. #define BITS\_COUNT\_MINUS\_1 BITS\_COUNT - 1
40. #define BEEP\_DURATION 250       // зажкржка между сигналами
41. #define BEEP\_HIGH 500           // длительность сигнала соответствующему биту 1
42. #define BEEP\_LOW 250            // длительность сигнала соответствующему биту 0
43. #define ISR\_DURATION 25
44. #define ISR\_DURATION\_2 ISR\_DURATION\*2
45. #define FALSE 0
46. #define TRUE 1
47. char led\_pins\_signals[][LEDS\_COUNT] = {
48. {LOW, LOW, LOW},    // 000
49. {HIGH, LOW, LOW},   // 001
50. {LOW, HIGH, LOW},   // 010
51. {HIGH, HIGH, LOW},  // 011
52. {LOW, LOW, HIGH},   // 100
53. {HIGH, LOW, HIGH},  // 101
54. {LOW, HIGH, HIGH},  // 110
55. {HIGH, HIGH, HIGH}, // 111
56. };
57. unsigned int ms\_counter = 0;    // счетчик миллисекнуд
58. int singed\_bits\_counter = 0;    // счетчик сыгнранных зуммером битов двоичного числа
59. unsigned int dur\_counter = 0;   // миллисекндный счетчик для задержки между сигналами
60. unsigned char key = 0;   // Адрес клавиши
61. int curr\_key\_number = 0; // цифра по адресу клавиши
62. char is\_reveal\_process\_started = FALSE; // флажок процесса проигрывания числа и отображения
63. char need\_duration = FALSE;     // флажок задержки между проигрыванием битов
64. void init\_timer0(void)
65. {// инициализация таймера 0
66. TMOD = 0x01; // 16-битовый режим
67. TL0 = RELOAD\_VALUE\_L;
68. TH0 = RELOAD\_VALUE\_H;
69. ET0 = 1; // Бит разрешения прерывания от таймера 0
70. TR0 = 1; // Бит управления таймера 0 для пуска/останова таймера/счетчика
71. }
72. void stop\_revealing()
73. {//остановить проигрывание и отображение битов двоичного числа
74. LED1 = LOW;
75. LED2 = LOW;
76. LED3 = LOW;
77. singed\_bits\_counter = 0;
78. BEEP\_BIT = LOW;
79. need\_duration = FALSE;
80. is\_reveal\_process\_started = FALSE;
81. }
82. void button\_handler() {
83. // Обработчик нажатия кнопок клавиатуры
84. key = KEY\_getkey();                         // подлучаем адрес нажатой клавиши
85. curr\_key\_number = KEY\_getkeynumber(key);    // получаем цифру по адресу клавиши
86. if (key == KEY\_ASTERISK)                    //При нажатии \* светодиоды выключаются.
87. stop\_revealing();                       //Динамик при этом перестаёт издавать звуки.
89. if (curr\_key\_number >= 0 && curr\_key\_number < 8) {
90. is\_reveal\_process\_started = TRUE;
91. dur\_counter = ms\_counter; //обновляем счетчик для генерации сигналов зумера
92. }
93. }
94. void reveal\_curr\_number()
95. {
96. // Вывести цифры от 0 до 7 десятичной системы счисления в двоичном представолении,
97. // через светодиоды стенда.
98. char is\_bit\_singed = FALSE;             // проигран ли один из битов двоичного числа?
99. // если нажата цифра в диапоне от 0 до 7,
100. // то зажигаем светодиоды в соответсвии с битами двоичного числа
101. if (curr\_key\_number >= 0 && curr\_key\_number < 8 && is\_reveal\_process\_started == TRUE)
102. {
103. set\_led\_pins\_signals(curr\_key\_number);          // включить светодиоды в соответсвтии
104. // с двоичным числом
106. // вопспроизвести динамиком бит двочиного числа
107. is\_bit\_singed = sing\_binary\_bit\_by\_zoomer(curr\_key\_number, singed\_bits\_counter);
108. if (is\_bit\_singed)
109. singed\_bits\_counter++;
110. }
111. if (singed\_bits\_counter == BITS\_COUNT\_MINUS\_1) {    // если сыграли зуммером все биты сбрасываем счетчик битов,
112. singed\_bits\_counter = 0;                        // сбрасываем флаг is\_reveal\_process\_started на 0
113. is\_reveal\_process\_started = FALSE;
114. }
115. }
116. void set\_led\_pins\_signals(unsigned int curr\_number)
117. {
118. // Установить уровни сигналов пинов светодиодов в соответсвтии с двоичным числом
119. LED1 = led\_pins\_signals[curr\_number][0];
120. LED2 = led\_pins\_signals[curr\_number][1];
121. LED3 = led\_pins\_signals[curr\_number][2];
122. }
123. char sing\_binary\_bit\_by\_zoomer(unsigned int curr\_number, unsigned int curr\_bit)
124. {
125. // Вопспроизвести динамиком (зумером) двочиное число.
126. // Динный сигнал соответствует биту 1, короткий - 0.
127. // Число воспроизводится от младшего бита к старшему.
128. char is\_bit\_singed = FALSE; // возвращаемое значение
130. if (led\_pins\_signals[curr\_number][curr\_bit] == HIGH && need\_duration == FALSE) {
131. BEEP\_BIT = HIGH;            // если текущий бит двочиного числа равен 1,
132. // то включаем зумер на время BEEP\_HIGH.
133. if (ms\_counter - dur\_counter > BEEP\_HIGH) {
134. dur\_counter = ms\_counter;
135. BEEP\_BIT = LOW;
136. need\_duration = TRUE;
137. }
138. }
139. else
140. BEEP\_BIT = HIGH;            // иначе включаем зумер на время BEEP\_LOW
141. if (ms\_counter - dur\_counter > BEEP\_LOW && need\_duration == FALSE) {
142. dur\_counter = ms\_counter;
143. BEEP\_BIT = LOW;
144. need\_duration = TRUE;
145. }
146. // задержка между сыгранными битами величиной в BEEP\_DURATION
147. if (ms\_counter - dur\_counter > BEEP\_DURATION && need\_duration == TRUE) {
148. need\_duration = FALSE;
149. is\_bit\_singed = TRUE;
150. }
151. return is\_bit\_singed;
152. }
153. void timer0\_ISR(void) interrupt 1 using 0
154. {   // обработчик прерывания по таймеру
155. button\_handler();
157. if (ms\_counter % ISR\_DURATION\_2 == 0)
158. reveal\_curr\_number(ms\_counter);
159. ms\_counter+=ISR\_DURATION;
160. }
161. // главная функция
162. void main()
163. {
164. // Настраиваем порты на вывод
165. P0 = 0xFF;
166. P1 = 0xFF;
167. P2 = 0xFF;
168. P3 = 0xFF;
169. EA = 1;  // Разрешаем прерывания
170. while (1)
171. {
172. /\* бсконечный цикл \*/
173. }
174. }

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

1. #include "SPI.H"
2. #include "LCD\_HD44780.H"
3. #include "I2C.H"
4. #include "KEYBOARD.H"
5. #include "LED7.H"
6. #include "UART.H"
7. #include "DS1307.H"
8. #include "DS1820.H"
9. #include "AT24CXX.H"
10. #include "CONVERT.H"
11. #include "PINDEF.h"
12. /\*
13. Студент:    Балкунов К.С.
14. Группа:     СМ7-74Б
15. Формулировка задачи:
16. 1) Вывести название режима - ClockMode на LCD дисплей;
17. 2) Выводить текущее время на LCD дисплей, полученное из микросхемы DS1307
18. (информация на дисплее обновляетя каждые 250 мс - 4 Гц);
19. 3) При нажатии # клавиатуры стенда включается режим изменения времени - пользователю нужно последовательно
20. через клавиатуру ПК ввести час, минуты, секунды;
21. 4) После изменения текущего времени снова включается режим отображения.
22. \*/
23. #define DISPLAY\_DELAY 500 // [мс], время отображения одного кадра на LCD дисплее
24. #define FALSE 0
25. #define TRUE 1
26. flash char UartMessageClockMode[] = "ClockMode\r\n";
27. flash char timeStr[] = "Time: ";
28. flash char enterNewTimeStr[] = "Enter new time: ";
29. flash char enterHourStr[] = "Enter hour: ";
30. flash char enterMinuteStr[] = "Enter minute: ";
31. flash char enterSecondStr[] = "Enter second: ";
32. char is\_time\_change\_mode = FALSE;   // флаг режима изменения времени
33. unsigned char key = 0;              // ключ символа нажатой кнопки клавиатуры
34. unsigned int ms\_ctr = 0;            // счетчик миллисекунд от начала работы программы
35. // установить новое время в микросхему DS1307 через UART-интерфейс
36. void set\_new\_time()
37. {
38. unsigned char hour, minute, second;
39. UART\_sendstring\_flash( enterHourStr );
40. hour = UART\_receivevalue();
41. UART\_sendstring\_flash( enterMinuteStr );
42. minute = UART\_receivevalue();
43. UART\_sendstring\_flash( enterSecondStr );
44. second = UART\_receivevalue();
45. DS1307\_settime( hour, minute, second );
46. is\_time\_change\_mode = FALSE;
47. }
48. // вывести текущее время на LCD дисплей
49. void display\_time()
50. {
51. LCD\_clrscr();
52. char time[9];
53. DS1307\_gettime(time); // получить текущее вермя в массив time
54. UART\_sendstring\_flash( timeStr );
55. UART\_sendstring( time ); // вывести время в консоль
56. LCD\_gotoxy( 0, 0 );
57. LCD\_print(time);        // вывести время на ЖК дисплей
58. }
59. void init\_timer1() {                                     // Настройка таймера 1 (частота atmega16 - 8МГц)
60. // Настройка компоратора B для оргнаизации миллисекундного таймера 1000 Гц (1 мс)
61. // и обновления дисплея с частотой 40 Гц (25 мс)
62. TCCR1B = 0;                                          // Очистить регистр управления компаратором 1В
63. TCCR1B |= (1 << TCCR1B\_WGM12);                       // устанавливаем режим СТС (сброс по совпадению)
64. TCCR1B |= (1 << TCCR1B\_CS11);                        // Предделитель 8
65. OCR1A = 1000;                                        // Значение для достижения 1000 Гц (1 мс) компаратор А
66. OCR1B = 25000;                                       // Значение для достижения 40 Гц (25 мс) копаратор В
67. TIMSK\_OCIE1A |= 1;                                   // Запуск компаратора таймера А
68. TIMSK\_OCIE1B |= 1;                                   // Запуск компаратора таймера В
69. }
70. #pragma vector = TIMER1\_COMPA\_vect
71. \_\_interrupt void Timer1\_COMPA(void)
72. {
73. ms\_ctr++;
74. }
75. #pragma vector = TIMER1\_COMPB\_vect
76. \_\_interrupt void Timer1\_COMPB(void)
77. {
78. if ( is\_time\_change\_mode == FALSE ) // выводим текущее время
79. display\_time();                 // если не находимя в режиме изменения времени
80. }
81. void button\_handler() {     // обработчик нажатия кнопок клавиатуры стенда
82. if (ms\_ctr % 10 == 0)   // опрашиваем клавиатуру раз в 10 мс
83. key = KEY\_getkey();
84. }
85. void change\_mode() {
86. if (key == KEY\_BAR) //если нажата # активируем флажок изменения времени
87. is\_time\_change\_mode = TRUE;
88. }
89. int main()
90. {
91. I2C\_init();
92. SPI\_init();
93. LCD\_init();
94. LCD\_clrscr();
95. UART\_init( CALC\_UBRR( 57600 ) );
96. \_\_enable\_interrupt();
98. UART\_sendstring\_flash( UartMessageClockMode );
99. OutputStartLCDMessage();
100. while(1)
101. {
102. button\_handler();   // читаем данные с клавиатуры
103. change\_mode();      // устанавливаем режим изменения времени, если была нажата #
104. if (is\_time\_change\_mode == TRUE) {
105. set\_new\_time();
106. is\_time\_change\_mode == FALSE;
107. }
108. }
109. }