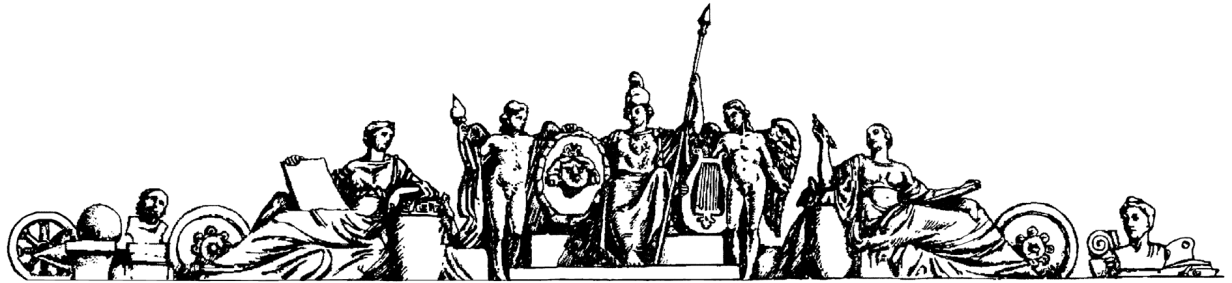
Федеральное агентство по образованию

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

**Лабораторные работы выполняются на стендах**

Используются микроконтроллеры:

• AT89S8252 с кварцевым резонатором 22.11 МГц,

• ATmega16 с кварцевым резонатором 14.7456 МГц.

**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.

**Ход выполнения лабораторной работы:**

1. Ознакомление с программно-аппаратными средствами составления и отладки программ на базе RISC микроконтроллеров семейства AVR в системе реального времени.
2. Подготовка кода программы для микроконтроллера AT89S8252 семейства x51 с кварцевым резонатором 21.11 МГц для отладки на стенде.
3. Подготовка кода программы для микроконтроллера семейства AVR ATmega16 с кварцевым резонатором 14.7456 МГц для отладки на стенде.
4. Отладка программ на экспериментальном стенде.
5. Составление отчета о лабораторной работе.

1) Описание периферии стенда

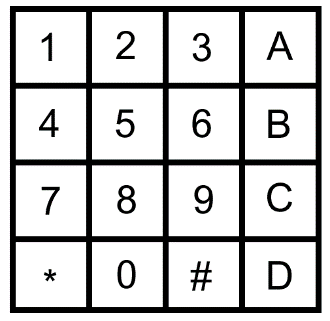
1. Светодиодный индикатор

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**

**Для микроконтроллера семейства x51:**

**Последовательный периферийный интерфейс SPI (Serial Peripheral Interface),** реализованный в микроконтроллерах семейства x51, 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.

Данный интерфейс используется для коммуникации со сдвиговыми регистрами на плате, выполняющими ввод-вывод данных в **семисегментный индикатор** и **ЖК-дисплей**.

**Для микроконтроллера семейства AVR:**

Последовательный периферийный интерфейс (Serial Peripheral Interface), реализованный в микроконтроллерах семейства, имеет два назначения. Прежде всего, через него может быть осуществлено программирование микроконтроллера (так называемый режим последовательного программирования), так же организация высокоскоростного обмена данными между микроконтроллером и различными периферийными устройствами, такими как цифровые потенциометры ЦАП/АЦП, Flash-ПЗУ и др. 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**

**Для микроконтроллера семейства x51:**

**Последовательный интерфейс UART (RS-232)**

Через универсальный асинхронный приемопередатчик UART (Universal Asynchronous Receiver-Transmitter) осуществляются прием и передача информации, представленной последовательным кодом (младшими битами вперед), в полном дуплексном режиме обмена.

Последовательный порт 8051 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**;

**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 **f**рез:

**F** = **f**РЕЗ / 12

В режиме 2 частота передачи определяется выражением:

**F** =2SMOD**f**РЕЗ / 64

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 **f**OVLT и определяется следующим образом:

**F** =2SMOD **f**OVLT / 32

**Для микроконтроллера семейства AVR:**

**UART -** Микроконтроллеры Atmega 16x и Atmega 162x 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:

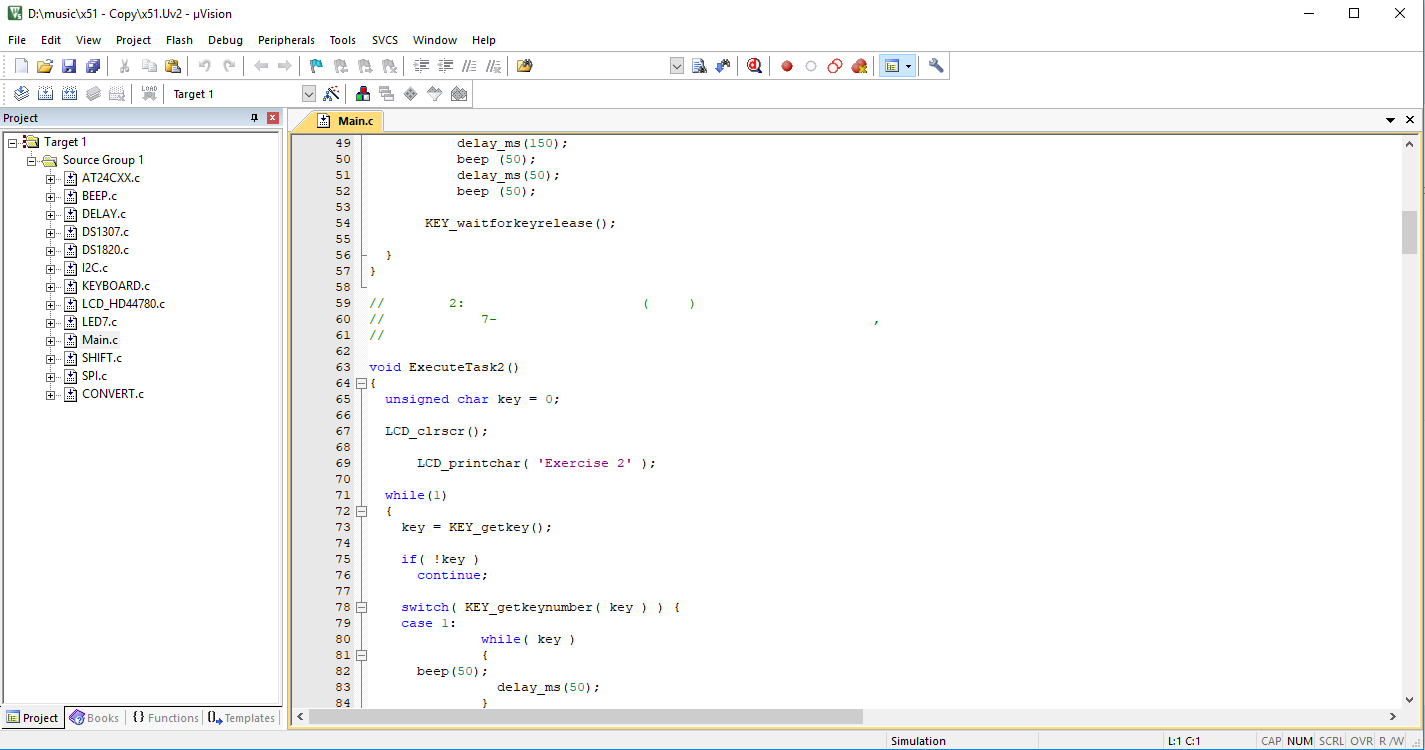
* переполнение;
* ошибка кадрирования;
* неверный старт-бит.

Для 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.

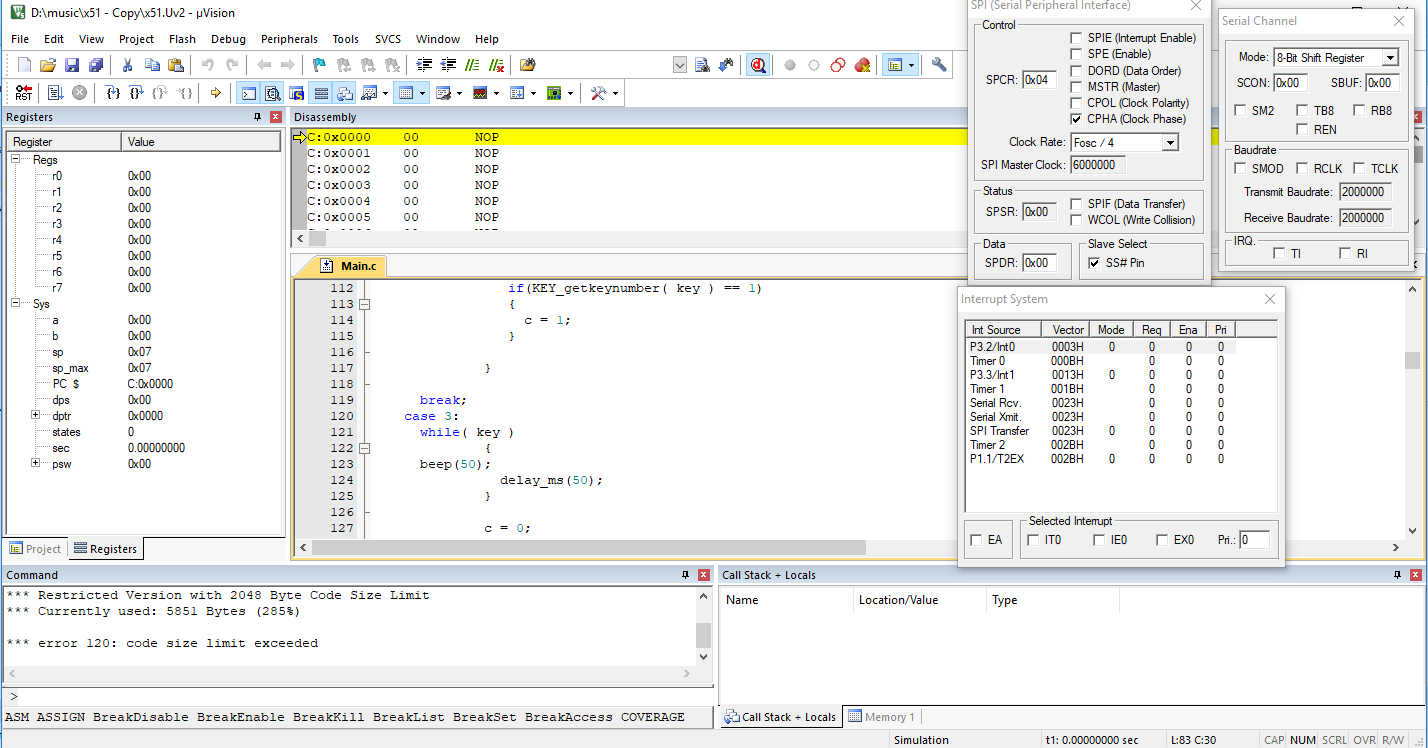
**Часть 1. Ознакомление с программно-аппаратными средствами составления и отладки программ на базе CISC микроконтроллеров семейства х51 в системе реального времени**

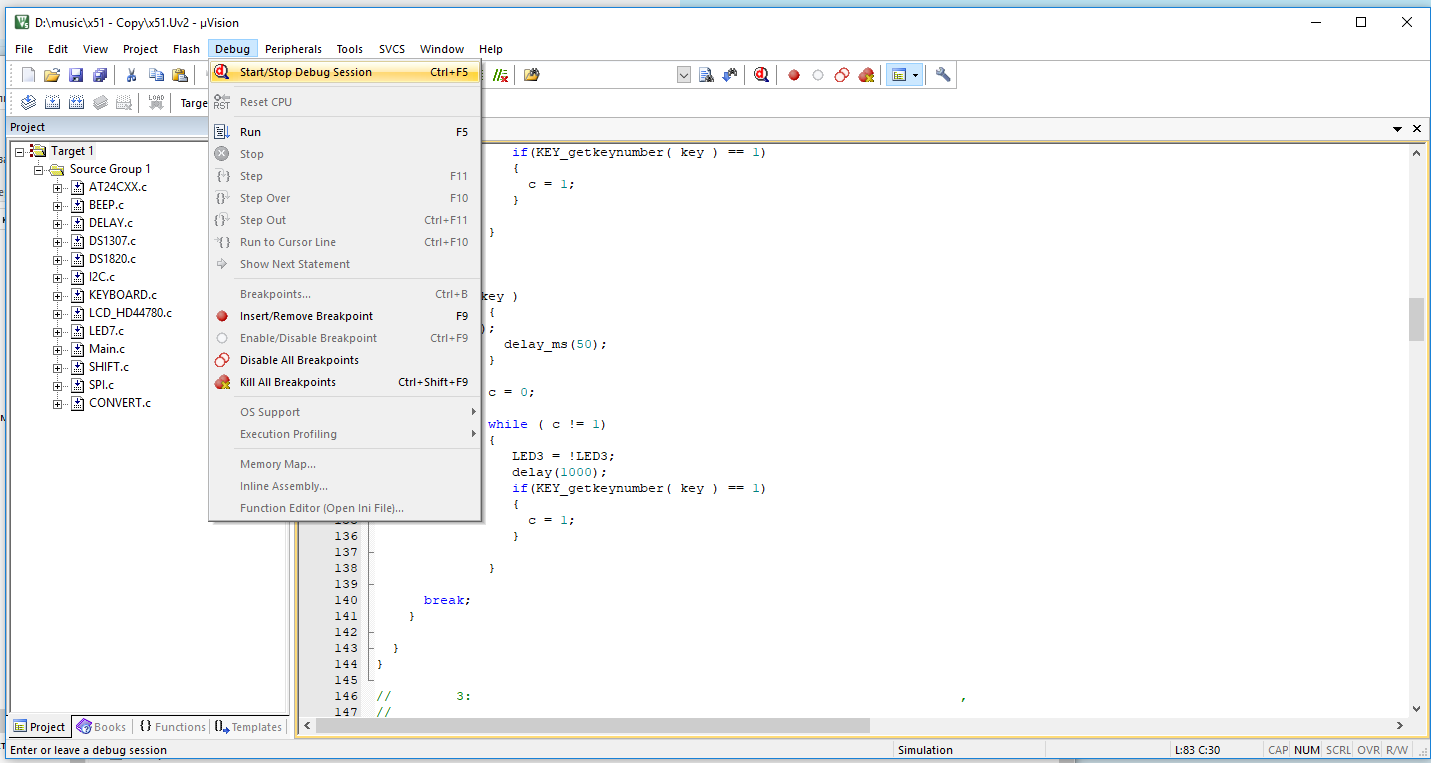
1) Описание процесса подготовки, прошивки контроллера и отладки программ

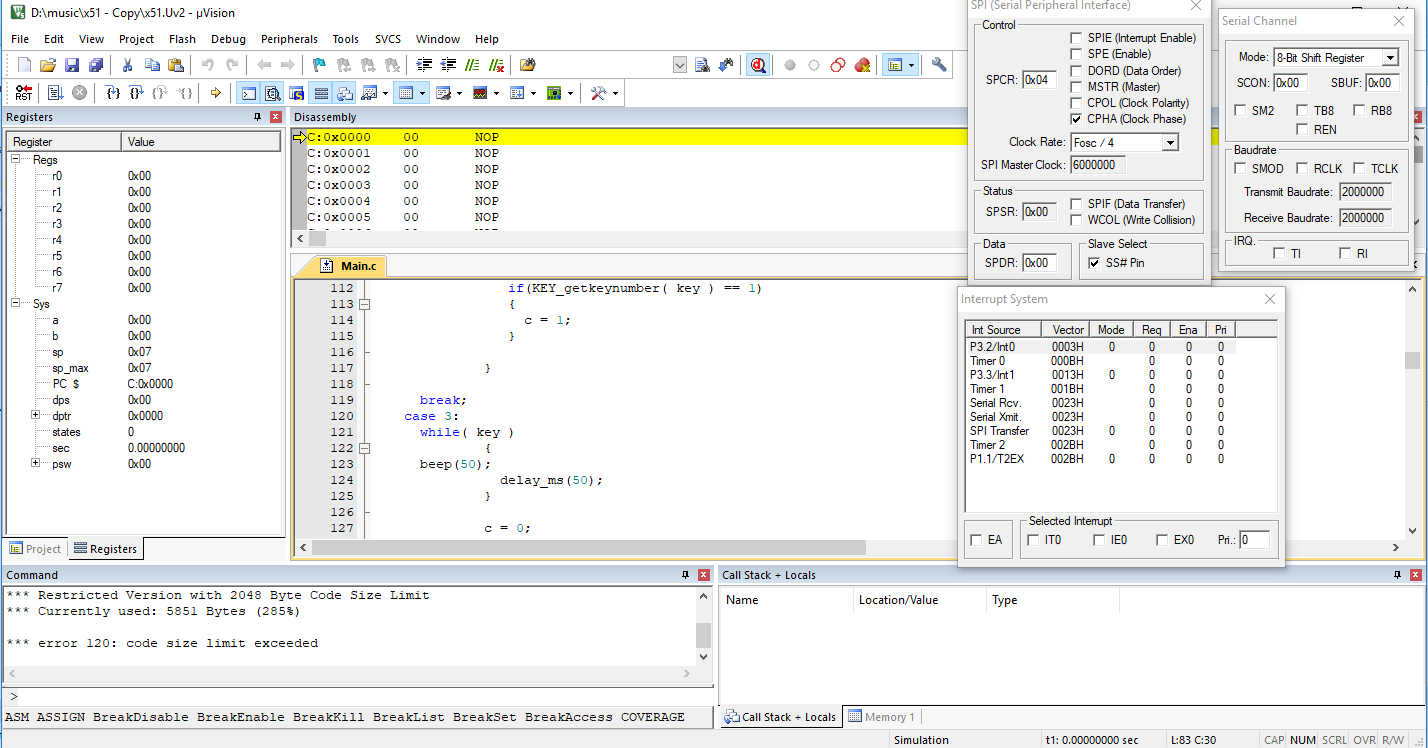
Программа разрабатывается в среде µVision2



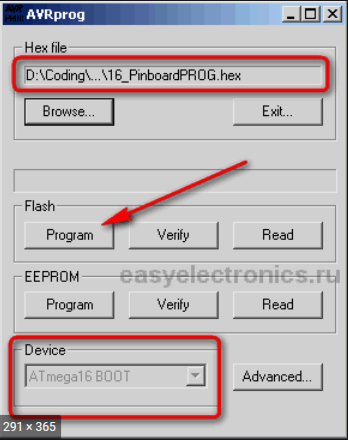
Для компиляции проекта нужно собрать проект, для этого нужно выполнить перестройку проекта.



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.

**Расчет частоты UART**

Последовательный порт 8051 будет работать в 3 режиме, поэтому частота приёма/передачи задается частотой таймера 1. Выберем частоту UART 9600 бод и SMOD = 0. Расчет частоты для таймера 1:

где fРЕЗ=22118400 Гц

Для кварца 22118400 Гц нужно поместить константу 0xFA (250) в TH1 и в TL1 для получения частоты UART 9600 бод.

Инициализация UART

1. void init\_UART()
2. {// инициализация UART
3. SCON = 0x50; // 8-битовый UART
4. TMOD = 0x20; // Таймер 1: 8-битовый режим, авто-перезагружаемый
5. TH1 = 250;   // Задаем начальное значение таймера для скорости 9600 бод
6. TL1 = 250;
7. TR1 = 1; // Запускаем таймер
8. TI = 1;
9. ES = 1; // Разрешаем прерывание от UART
10. }

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.Реализуем частоту прерываний 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

, где fРЕЗ=22118400 Гц, f = 1000 Гц.

TH0 = RELOAD\_VALUE\_H = 112 - старшие 8 бит таймера

TL0 = RELOAD\_VALUE\_L = 248 - младшие 8 бит таймера

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 3. if (ms\_counter % 10 == 0) 4. button\_handler(); 5. if (ms\_counter % 50 == 0) 6. reveal\_curr\_number(); 7. ms\_counter++; 8. } |

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(). Функция button\_handler() запускается каждую 10 мс, а функция reveal\_curr\_number() запускается каждые 50 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. {
3. // Обработчик нажатия кнопок клавиатуры
4. key = KEY\_getkey(); // подлучаем адрес нажатой клавиши
5. if (key != prev\_key)
6. {
7. if (button\_read\_flag == FALSE)
8. {
9. button\_read\_flag = TRUE;
10. button\_ms\_counter = ms\_counter;
11. return;
12. }
13. if (button\_read\_flag == TRUE && ms\_counter - button\_counter > 20)
14. button\_read\_flag = FALSE;
15. else
16. return;
17. curr\_key\_number = KEY\_getkeynumber(key); // получаем цифру по адресу клавиши
18. if (key == KEY\_ASTERISK) // При нажатии \* светодиоды выключаются.
19. stop\_revealing();    // Динамик при этом перестаёт издавать звуки.
20. if (curr\_key\_number >= 0 && curr\_key\_number < 8)
21. {
22. is\_reveal\_process\_started = TRUE;
23. dur\_counter = ms\_counter; // обновляем счетчик для генерации сигналов зумера
24. }
25. prev\_key = key;
26. }
27. }

**Логика работы учёта дребезга контактов кнопок**

Сперва происходит проверка изменения состояния кнопки. Если текущий адрес клавиши (key) отличается от предыдущего (prev\_key), то если button\_read\_flag = FALSE происходит запись текущего времени работы программы в переменную button\_ms\_counter измение флага button\_read\_flag на значнеие TRUE и выход из функции. Это нужно, для реализации задержки в 20 мс, которая обеспечит исключение дребезга контактов. При последующих вхождениях в функцию осуществляется проверка ms\_counter - button\_counter > 20. После выполнения этого условия флаг button\_read\_flag выставляется в FALSE, новое значение кнопки можно считать истинным, реализуется обработка нового значения.

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. }

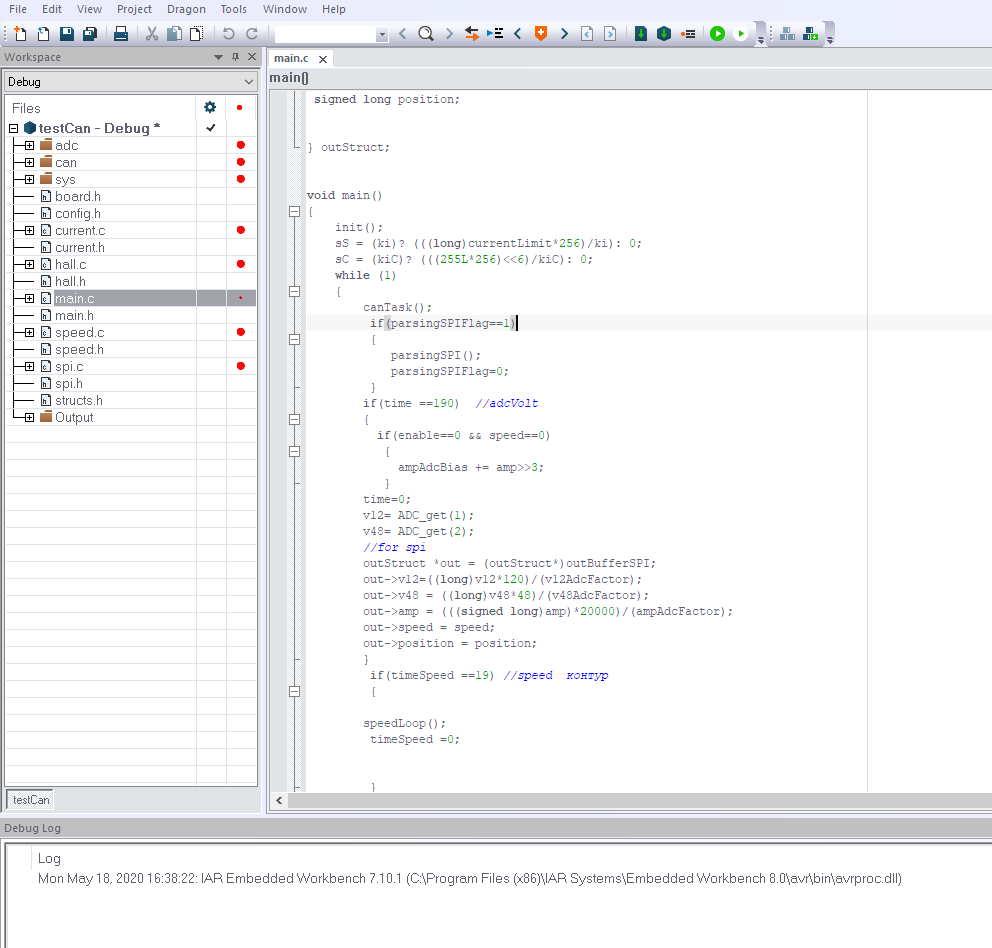
Особую сложность представляет функция контроля зуммера **sing\_binary\_bit\_by\_zoomer** так как на каждом тике системы реального времени мы не должны потерять текущую фазу выполнения длительного процесса проигрывания. Для контроля фаз работы зуммера используется флаг **need\_duration,** а для задания 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. }

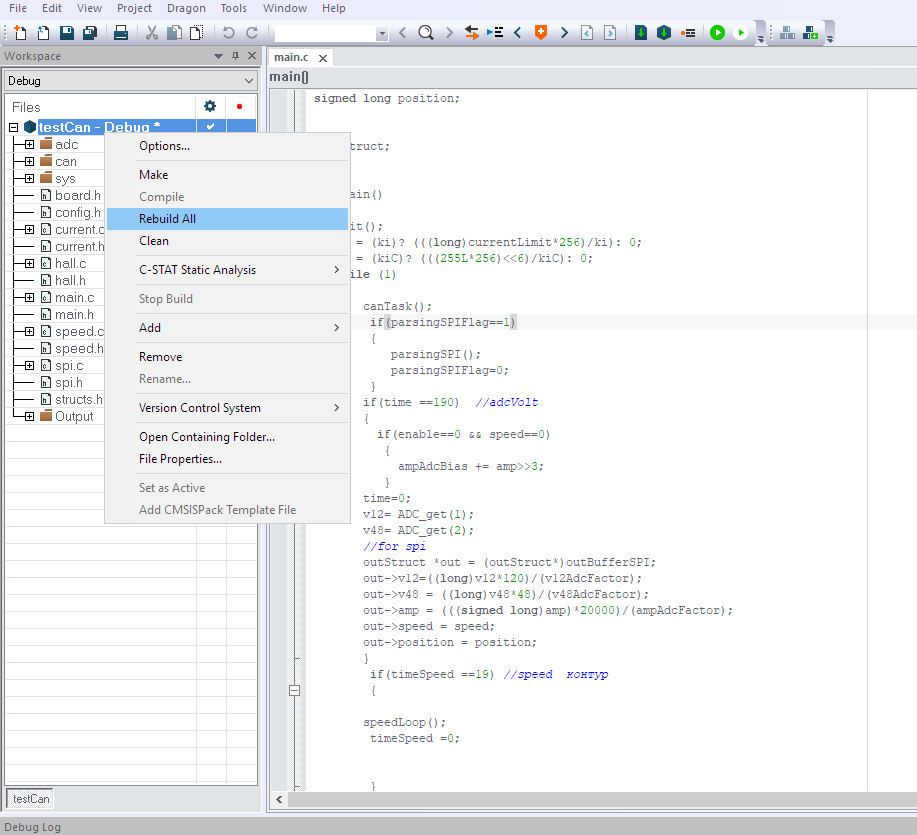
**Часть 2. Ознакомление с программно-аппаратными средствами составления и отладки программ на базе RISC микроконтроллеров семейства AVR в системе реального времени**

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

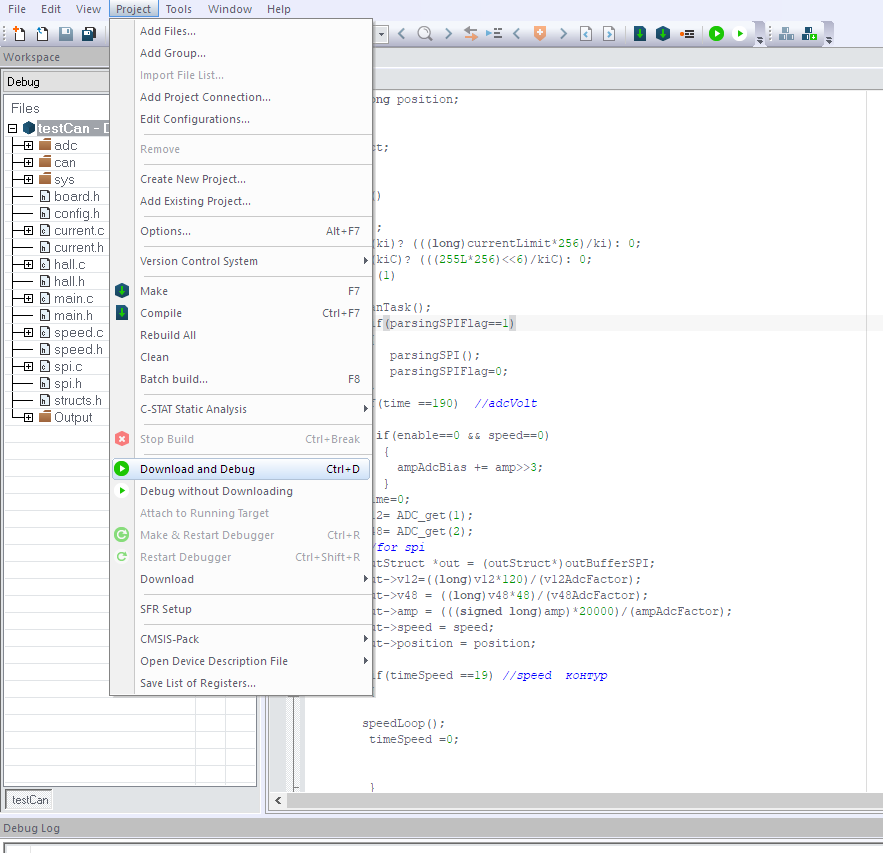
Программа разрабатывается в среде IAR Embedded Workbench IDE.

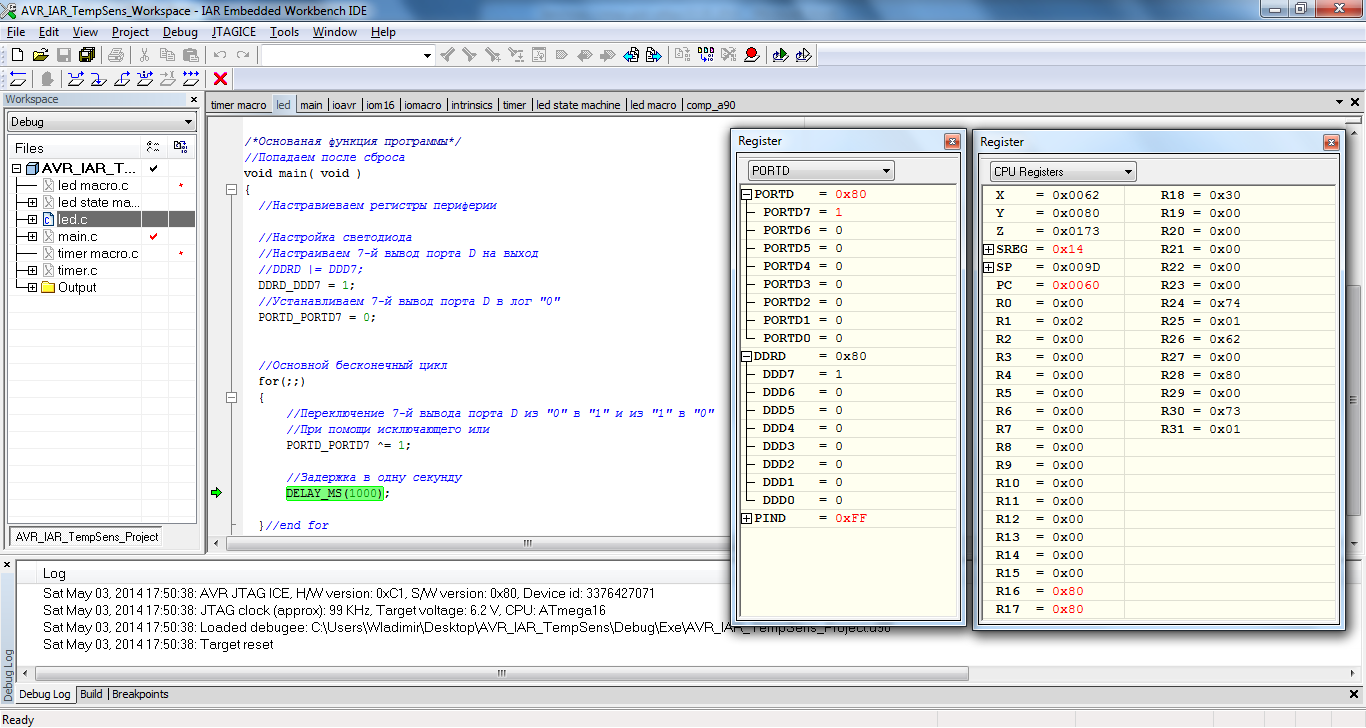


Для компиляции проекта нужно собрать проект, для этого нужно выполнить перестройку проекта (Rebuild All).

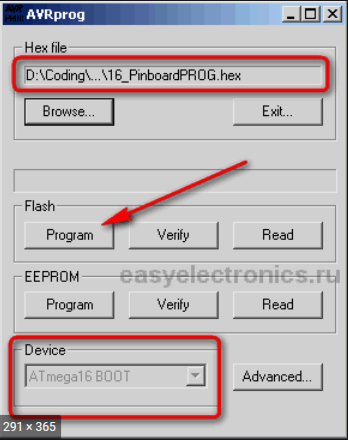


Так же для отладки кода и его дизассемблирования можно использовать режим отладки





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 UART**

Инициализация UART

|  |
| --- |
| 1. UART\_init( CALC\_UBRR( 9600 ) ); |

Функции UART\_init() и CALC\_UBRR() даны были даны в методических материалах.

|  |
| --- |
| 1. void UART\_init( unsigned short UBRR\_value ) {  UART\_UBRRL = ( unsigned char ) UBRR\_value;  UART\_UBRRH = ( unsigned char ) ( UBRR\_value >> 8 );    UART\_UCSRB\_RXEN = 1;  UART\_UCSRB\_TXEN = 1;  UART\_UCSRB\_RXCIE = 1; } |

1. Расчет частоты

Установим нормальный асинхронный двунаправленный режим работы со скоростью 9600 бод.

где f=14756000 Гц, band=9600. Тогда UBRRH = 0, UBRRL = 51.

**3.2) Прерывания по таймеру 1**

1) Расчет частоты таймера

В учебных целях воспользуемся таймером 1 и настроим его на генерацию двух прерываний с разными частотами. Используем режим CTC – сброс по совпадению, задаем два значения компараторов A и B для реализации частоты в 1000 Гц и 40 Гц.

Выберем предделитель , тогда

Таким образом получим необходимые значения на регистрах сравнения: OCR1A = 1844, OCR1B = 46080.

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() 2. {// Настройка таймера 1 3. // Настройка компоратора B для оргнаизации миллисекундного таймера 1000 Гц (1 мс) 4. // и обновления дисплея с частотой 40 Гц (25 мс) 5. TCCR1B = 0;                                          // Очистить регистр управления компаратором 1В 6. TCCR1B |= (1 << TCCR1B\_WGM12);                       // устанавливаем режим СТС (сброс по совпадению) 7. TCCR1B |= (1 << TCCR1B\_CS11);                        // Предделитель 8 8. OCR1A = 1844;                       // Значение для достижения 1000 Гц (1 мс) компаратор А 9. OCR1B = 46080;                     // Значение для достижения 40 Гц (25 мс) копаратор В 10. TIMSK\_OCIE1A |= 1;                                   // Запуск компаратора таймера А 11. TIMSK\_OCIE1B |= 1;                                   // Запуск компаратора таймера В 12. } |

3) Обработчики прерываний по таймеру

В обработчике прерывания по компаратору А осуществляется приращение глобальной переменной-счётчика миллисекунд и запуск основынх функций программы.

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. if (ms\_ctr % 10 == 0) 5. button\_handler();   // читаем данные с клавиатуры раз в 10 мс 6. if (is\_time\_change\_mode == TRUE) { 7. set\_new\_time(); 8. is\_time\_change\_mode == FALSE; 9. } 10. ms\_ctr++; 11. } 12. #pragma vector = TIMER1\_COMPB\_vect 13. \_\_interrupt void Timer1\_COMPB(void) 14. { 15. if ( is\_time\_change\_mode == FALSE ) // выводим текущее время 16. display\_time();                 // если не находимя в режиме изменения времени 17. } |

**3.1) Описание основных подпрограмм**

Функция **display\_time()** осуществляет вывод текущего времени на ЖК дисплей. Эти данные так же дублируются в консоль через UART.

1. // вывести текущее время на LCD дисплей
2. void display\_time()
3. {
4. LCD\_clrscr();
5. char time[9];
6. DS1307\_gettime(time); // получить текущее вермя в массив time
7. UART\_sendstring\_flash( timeStr );
8. UART\_sendstring( time ); // вывести время в консоль
9. LCD\_gotoxy( 0, 0 );
10. LCD\_print(time);        // вывести время на ЖК дисплей
11. }

Функция **set\_new\_time()** осуществляет запись нового времени в DS1307 через UART.

1. // установить новое время в микросхему DS1307 через UART-интерфейс
2. void set\_new\_time()
3. {
4. unsigned char hour, minute, second;
5. UART\_sendstring\_flash( enterHourStr );
6. hour = UART\_receivevalue();
7. UART\_sendstring\_flash( enterMinuteStr );
8. minute = UART\_receivevalue();
9. UART\_sendstring\_flash( enterSecondStr );
10. second = UART\_receivevalue();
11. DS1307\_settime( hour, minute, second );
12. is\_time\_change\_mode = FALSE;
13. }

В обработчике нажатий клавиш клавиатуры стенда **button\_handler()**, происходит проверка нажатия #, при выполнении этого взводится флаг режима изменения времени is\_time\_change\_mode = TRUE.

1. void button\_handler()
2. {
3. // Обработчик нажатия кнопок клавиатуры
4. key = KEY\_getkey(); // подлучаем адрес нажатой клавиши
5. if (key != prev\_key)
6. {
7. if (button\_read\_flag == FALSE)
8. {
9. button\_read\_flag = TRUE;
10. button\_ms\_counter = ms\_counter;
11. return;
12. }
13. if (button\_read\_flag == TRUE && ms\_counter - button\_ms\_counter > 20)
14. button\_read\_flag = FALSE;
15. else
16. return;
17. if (key == KEY\_BAR) //если нажата # активируем флажок изменения времени
18. is\_time\_change\_mode = TRUE;
19. prev\_key = key;
20. }
21. }

**Логика работы учёта дребезга контактов кнопок**

Сперва происходит проверка изменения состояния кнопки. Если текущий адрес клавиши (key) отличается от предыдущего (prev\_key), то если button\_read\_flag = FALSE происходит запись текущего времени работы программы в переменную button\_ms\_counter измение флага button\_read\_flag на значнеие TRUE и выход из функции. Это нужно, для реализации задержки в 20 мс, которая обеспечит исключение дребезга контактов. При последующих вхождениях в функцию осуществляется проверка ms\_counter - button\_counter > 20. После выполнения этого условия флаг button\_read\_flag выставляется в FALSE, новое значение кнопки можно считать истинным, реализуется обработка нового значения.

**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 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 int button\_ms\_counter = 0; // счетчик миллисекнуд для исключения дребезга контактов

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

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

unsigned char key = 0;      // Адрес клавиши

unsigned char prev\_key = 0; // Адрес клавиши на прошлом тике опроса клавиатуры

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

char button\_read\_flag = FALSE;  //флажок чтения адресов кнопок клавиатуры стенда

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 init\_UART()

{// инициализация UART

    SCON = 0x50; // 8-битовый UART

    TMOD = 0x20; // Таймер 1: 8-битовый режим, авто-перезагружаемый

    TH1 = 250;   // Задаем начальное значение таймера для скорости 9600 бод

    TL1 = 250;

    TR1 = 1; // Запускаем таймер

    TI = 1;

    ES = 1; // Разрешаем прерывание от UART

}

void stop\_revealing()

{ // остановить проигрывание и отображение битов двоичного числа

    LED1 = LOW;

    LED2 = LOW;

    LED3 = LOW;

    singed\_bits\_counter = 0;

    BEEP\_BIT = LOW;

    need\_duration = FALSE;

    is\_reveal\_process\_started = FALSE;

}

void button\_handler()

{

    // Обработчик нажатия кнопок клавиатуры

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

    if (key != prev\_key)

    {

        if (button\_read\_flag == FALSE)

        {

            button\_read\_flag = TRUE;

            button\_ms\_counter = ms\_counter;

            return;

        }

        if (button\_read\_flag == TRUE && ms\_counter - button\_ms\_counter > 20)

            button\_read\_flag = FALSE;

        else

            return;

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

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

            stop\_revealing();    // Динамик при этом перестаёт издавать звуки.

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

        {

            is\_reveal\_process\_started = TRUE;

            dur\_counter = ms\_counter; // обновляем счетчик для генерации сигналов зумера

        }

        prev\_key = key;

    }

}

void reveal\_curr\_number()

{

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

    // через светодиоды стенда.

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

    // 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);

        if (is\_bit\_singed)

            singed\_bits\_counter++;

    }

    if (singed\_bits\_counter == BITS\_COUNT\_MINUS\_1)

    {                            // если сыграли зуммером все биты сбрасываем счетчик битов,

        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)

{

    // Вопспроизвести динамиком (зумером) двочиное число.

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

    // Число воспроизводится от младшего бита к старшему.

    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 - dur\_counter > BEEP\_HIGH)

        {

            dur\_counter = ms\_counter;

            BEEP\_BIT = LOW;

            need\_duration = TRUE;

        }

    }

    else

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

    if (ms\_counter - dur\_counter > BEEP\_LOW && need\_duration == FALSE)

    {

        dur\_counter = ms\_counter;

        BEEP\_BIT = LOW;

        need\_duration = TRUE;

    }

    // задержка между сыгранными битами величиной в BEEP\_DURATION

    if (ms\_counter - dur\_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

    if (ms\_counter % 10 == 0)

        button\_handler();

    if (ms\_counter % 50 == 0)

        reveal\_curr\_number();

    ms\_counter++;

}

// главная функция

void main()

{

    // Настраиваем порты на вывод

    P0 = 0xFF;

    P1 = 0xFF;

    P2 = 0xFF;

    P3 = 0xFF;

    EA = 1; // Разрешаем прерывания

    init\_UART();

    while (1)

    {

        /\* бсконечный цикл \*/

    }

}

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

#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

   через клавиатуру ПК 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

#define FALSE 0

#define TRUE 1

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: ";

char is\_time\_change\_mode = FALSE;   // флаг режима изменения времени

char button\_read\_flag = FALSE;      // флажок чтения адресов кнопок клавиатуры стенда

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

unsigned char prev\_key = 0; // Адрес клавиши на прошлом тике опроса клавиатуры

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

unsigned int button\_ms\_counter = 0; // счетчик миллисекнуд для исключения дребезга контактов

// 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

    // 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 = 1844;                                        // Значение для достижения 1000 Гц (1 мс) компаратор А

    OCR1B = 46080;                                       // 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)

{

    if (ms\_ctr % 10 == 0)

        button\_handler();   // читаем данные с клавиатуры раз в 10 мс

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

        set\_new\_time();

        is\_time\_change\_mode == FALSE;

    }

    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 нажатия кнопок клавиатуры

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

    if (key != prev\_key)

    {

        if (button\_read\_flag == FALSE)

        {

            button\_read\_flag = TRUE;

            button\_ms\_counter = ms\_counter;

            return;

        }

        if (button\_read\_flag == TRUE && ms\_counter - button\_ms\_counter > 20)

            button\_read\_flag = FALSE;

        else

            return;

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

            is\_time\_change\_mode = TRUE;

        prev\_key = key;

    }

}

int main()

{

    I2C\_init();

    SPI\_init();

    LCD\_init();

    LCD\_clrscr();

    UART\_init( CALC\_UBRR( 9600 ) );

    \_\_enable\_interrupt();

    UART\_sendstring\_flash( UartMessageClockMode );

    OutputStartLCDMessage();

    while(1)

    {

    }

}