Ministerul Educației şi Tineretului al Republicii Moldova

Universitatea Tehnică a Moldovei

Departament „Informatica aplicată”

RAPORT

Lucrarea de laborator nr. 6

LA DISCIPLINA”Programarea aplicaţiilor incorporate şi independente de platformă ”

A efectuat:

St. gr. FAF 141 A. Dariev

A verificat:

Lect . supp A.Bragrarenco

Chişinău 2017

Task: Create a simple program based on the knowledges from previous labs

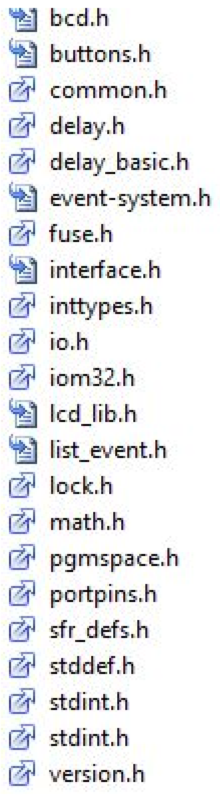
My Task: To create a simple watch on **AVR** controller

As an example of an event-triggered system on the table, I chose such a popular device as the clock on the microcontroller. And it's functional is to display and set the time.
From hardware I've used a microcontroller AVR - atmega8535, character lcd and buttons - to navigate through the clock menu and set the time. Three pieces - Enter, Up, Down would be enough. Cancel button we will neglect.
The buttons will be asked to interrupt timer T0 and T1 with a timer will be formed second intervals.

Current state	Event	Next state	Event handler function
Display Time	Enter is pressed Second timer stops	Go to setting time state State doesn't change	Set the cursor Increment counter, change display state
Setting Hours	Enter is pressed Up button is pressed Down is pressed	Setting minutes State doesn't change State doesn't change	Set the cursor Increment counter, change display state Decrement counter. change display state

Setting Minutes	Enter is pressed Up button is pressed Down is pressed	Go to display time State doesn't change State doesn't change	Hide the cursor Increment minutes change display state Decrement minutes

Project structure:



```
bcd.h
buttons.h
common.h
delay.h
delay_basic.h
event-system.h
fuse.h
interface.h
inttypes.h
io.h
iom32.h
lcd_lib.h
list_event.h
lock.h
math.h
pgmspace.h
portpins.h
sfr_defs.h
stddef.h
stdint.h
stdint.h
version.h
```

Codes of states and events I've set in the header file list_event.h, it is used in some software modules.

```
//Event code
#define EVENT_NULL          0
#define EVENT_KEY_UP        1
#define EVENT_KEY_DOWN      2
#define EVENT_KEY_ENTER     3
#define EVENT_SYS_TIMER     4
 //State code
#define STATE_NO_CHANGE     0
#define STATE_NORMAL        1
#define STATE_SET_HOUR      2
#define STATE_SET_MINUTE    3
```

Prototypes of functions(interface.h):

```
void GUI_General(void);
void GUI_SelectHour(void);
void GUI_SelectMinute(void);
void GUI_IncHour(void);
void GUI_DecHour(void);
void GUI_IncMinute(void);
void GUI_DecMinute(void);
void GUI_SetTime(void);
void GUI_ChangeTime(void);
```

Transition table is in the blank event system - event-system.c

```
 __flash struct ROW_TABLE table[] = {
             //STATE                              EVENT                              NEXT  STATE
STATE_FUNC
        {STATE_NORMAL,               EVENT_KEY_ENTER,                STATE_SET_HOUR,
GUI_SelectHour},
        {STATE_NORMAL,              EVENT_SYS_TIMER,              STATE_NO_CHANGE,
GUI_ChangeTime},

         {STATE_SET_HOUR,           EVENT_KEY_ENTER,                STATE_SET_MINUTE,
GUI_SelectMinute},
        {STATE_SET_HOUR,          EVENT_KEY_UP,                         STATE_NO_CHANGE,
GUI_IncHour},
         {STATE_SET_HOUR,           EVENT_KEY_DOWN,                  STATE_NO_CHANGE,
GUI_DecHour},

           {STATE_SET_MINUTE,         EVENT_KEY_ENTER,                    STATE_NORMAL,
GUI_General},
        {STATE_SET_MINUTE,       EVENT_KEY_UP,                         STATE_NO_CHANGE,
GUI_IncMinute},
         {STATE_SET_MINUTE,       EVENT_KEY_DOWN,                  STATE_NO_CHANGE,
GUI_DecMinute},

     {           0,                                       0,                                        0,
EmptyFunc}
};
```

Next.
I did module timer.c, timer.h. There are three functions - function timers T0 and T1 initialization and interrupt function. T0 is the interrupt every 10ms, T1 - every second.

```
void TIM_Init(void)
{
   …..
}

//button request
#pragma vector = TIMER0_COMP_vect
__interrupt void Timer0Comp(void)
{
  BUT_Debrief();  //function for button request from button.c
}

//Second timer
#pragma vector = TIMER1_COMPA_vect
```

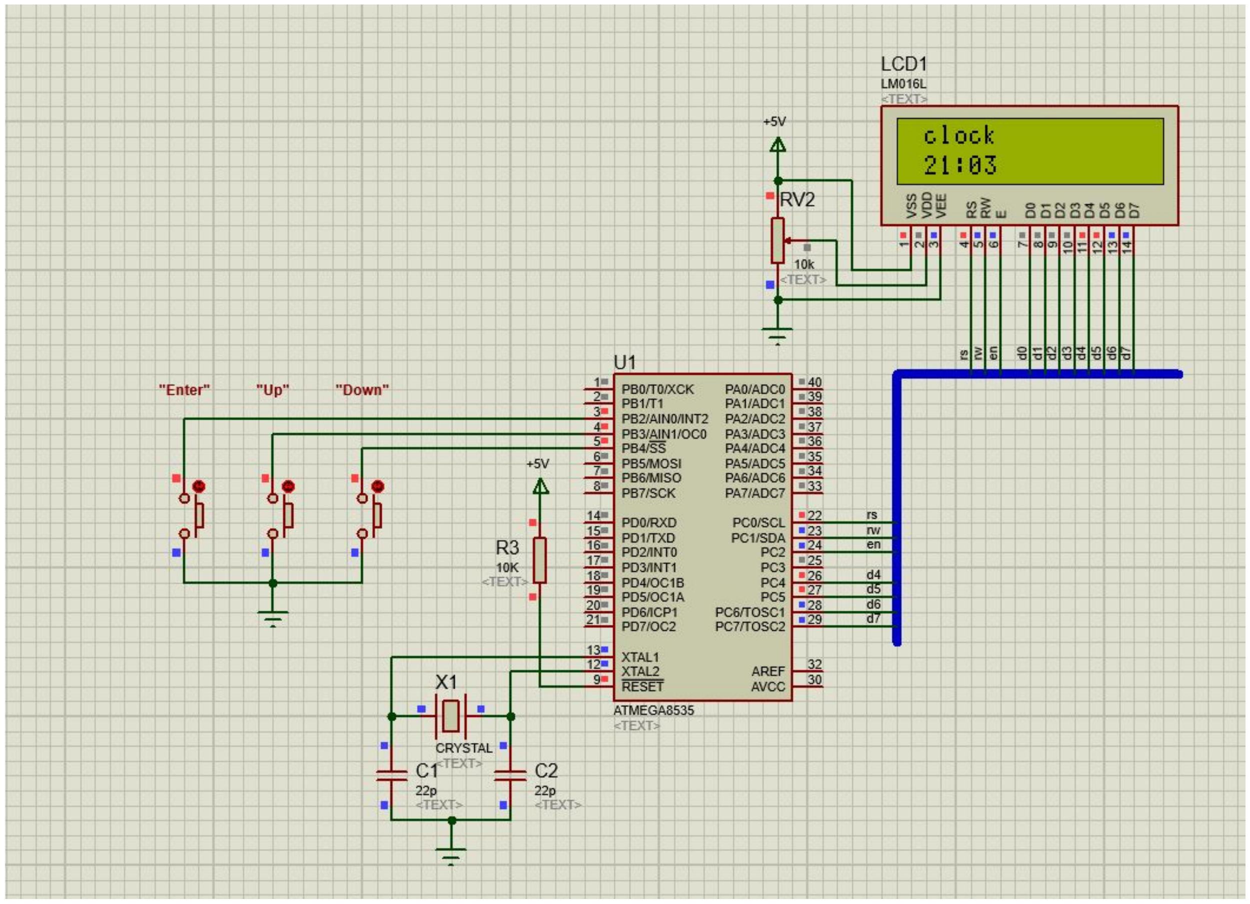
```
__interrupt void Timer1CompA(void)
{
  ES_PlaceEvent(EVENT_SYS_TIMER);
}
```

in button.h - pins, which are connected to buttons and port
in lcd_lib.h - pins, which is connected to the lcd, port, quartz frequency, the survey of employment and the flag type of controller.
in bcd.h - redefined the output function on the lcd

Further included in the main header files required modules and entered the initialization functions - timer, display, event system and buttons.

```
//main module
#include <ioavr.h>
#include <intrinsics.h>
#include "lcd_lib.h"
#include "buttons.h"
#include "event-system.h"
#include "interface.h"
#include "timer.h"

int main( void )
{
  unsigned char event = 0;

  LCD_Init();
  BUT_Init();
  TIM_Init();
  ES_Init();
  GUI_General();

  __enable_interrupt();
  while(1){
    event = ES_GetEvent();
    if (event){
      ES_Dispatch(event);
    }
  }

  return 0;
}
```

Scheme:



Conclusion:

In this laboratory work I’ve implemented almost all knowledges about AVR, buttons, lcd, interrupts and etc. The most annoying thing was the separation our code to the right modules. In the end we’ve got a simple clock where we can customize our time.

APPENDIX:
main.c:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "lcd_lib.h"
#include "buttons.h"
#include "event-system.h"
#include "interface.h"
#include "timer.h"
```

```
int main( void )
{
  unsigned char event = 0;

  LCD_Init();
  BUT_Init();
  TIM_Init();
  ES_Init();
  GUI_General();

  sei();
  while(1){
          event = ES_GetEvent();
    if (event){
       ES_Dispatch(event);
    }
  }

  return 0;
}
```

lcd.c:

```
#include "lcd_lib.h"

//макросы для работы с битами
#define ClearBit(reg, bit)       reg &= (~(1<<(bit)))
#define SetBit(reg, bit)         reg |= (1<<(bit))

#define FLAG_BF 7

inline unsigned char __swap_nibbles(unsigned char data)
{
  asm volatile("swap %0" : "=r" (data) : "0" (data));
  return data;
}

void LCD_WriteComInit(unsigned char data)
{
  _delay_us(40);
  ClearBit(PORT_SIG, RS);
#ifdef BUS_4BIT
  unsigned char tmp;
  tmp  = PORT_DATA & 0x0f;
  tmp |= (data & 0xf0);
  PORT_DATA = tmp;
#else
  PORT_DATA = data;
#endif
  SetBit(PORT_SIG, EN);
  _delay_us(2);
  ClearBit(PORT_SIG, EN);
}


inline static void LCD_CommonFunc(unsigned char data)
{
#ifdef BUS_4BIT
  unsigned char tmp;
  tmp  = PORT_DATA & 0x0f;
  tmp |= (data & 0xf0);

  PORT_DATA = tmp;                  //вывод старшей тетрады
  SetBit(PORT_SIG, EN);
  _delay_us(2);
  ClearBit(PORT_SIG, EN);

  data = __swap_nibbles(data);
  tmp  = PORT_DATA & 0x0f;
  tmp |= (data & 0xf0);

  PORT_DATA = tmp;                  //вывод младшей тетрады
  SetBit(PORT_SIG, EN);
  _delay_us(2);
  ClearBit(PORT_SIG, EN);
#else
  PORT_DATA = data;                     //вывод данных на шину индикатора
  SetBit(PORT_SIG, EN);       //установка E в 1
  _delay_us(2);
  ClearBit(PORT_SIG, EN);          //установка E в 0 - записывающий фронт
#endif
}

inline static void LCD_Wait(void)
{
#ifdef CHECK_FLAG_BF
  #ifdef BUS_4BIT

  unsigned char data;
  DDRX_DATA &= 0x0f;            //конфигурируем порт на вход
  PORT_DATA |= 0xf0;              //включаем pull-up резисторы
  SetBit(PORT_SIG, RW);         //RW в 1 чтение из lcd
  ClearBit(PORT_SIG, RS);          //RS в 0 команды
  do{
    SetBit(PORT_SIG, EN);
    _delay_us(2);
    data = PIN_DATA & 0xf0;      //чтение данных с порта
    ClearBit(PORT_SIG, EN);
    data = __swap_nibbles(data);
    SetBit(PORT_SIG, EN);
    _delay_us(2);
    data |= PIN_DATA & 0xf0;      //чтение данных с порта
    ClearBit(PORT_SIG, EN);
    data = __swap_nibbles(data);
  }while((data & (1<<FLAG_BF))!= 0 );
  ClearBit(PORT_SIG, RW);
  DDRX_DATA |= 0xf0;

  #else
  unsigned char data;
  DDRX_DATA = 0;                //конфигурируем порт на вход
  PORT_DATA = 0xff;                //включаем pull-up резисторы
  SetBit(PORT_SIG, RW);         //RW в 1 чтение из lcd
  ClearBit(PORT_SIG, RS);          //RS в 0 команды
  do{
    SetBit(PORT_SIG, EN);
    _delay_us(2);
    data = PIN_DATA;           //чтение данных с порта
    ClearBit(PORT_SIG, EN);
  }while((data & (1<<FLAG_BF))!= 0 );
  ClearBit(PORT_SIG, RW);
  DDRX_DATA = 0xff;
  #endif
#else
  _delay_us(40);
#endif
}

//функция записи команды
void LCD_WriteCom(unsigned char data)
{
  LCD_Wait();
  ClearBit(PORT_SIG, RS);          //установка RS в 0 - команды
  LCD_CommonFunc(data);
}
```

```
//функция записи данных
void LCD_WriteData(unsigned char data)
{
  LCD_Wait();
  SetBit(PORT_SIG, RS);       //установка RS в 1 - данные
  LCD_CommonFunc(data);
}

//ooieoey eieoeaeecaoee
void LCD_Init(void)
{
#ifdef BUS_4BIT
  DDRX_DATA |= 0xf0;
  PORT_DATA |= 0xf0;
#else
  DDRX_DATA |= 0xff;
  PORT_DATA |= 0xff;
#endif

  DDRX_SIG |= (1<<RW)|(1<<RS)|(1<<EN);
  PORT_SIG |= (1<<RW)|(1<<RS)|(1<<EN);
  ClearBit(PORT_SIG, RW);
  _delay_ms(40);

#ifdef HD44780
  LCD_WriteComInit(0x30);
  _delay_ms(10);
  LCD_WriteComInit(0x30);
  _delay_ms(1);
  LCD_WriteComInit(0x30);
#endif

#ifdef BUS_4BIT
  LCD_WriteComInit(0x20); //4 разрядная шина
  LCD_WriteCom(0x28); //4-разрядная шина, 2 - строки
#else
  LCD_WriteCom(0x38); //8-разрядная шина, 2 - строки
#endif
  LCD_WriteCom(0x08);
  LCD_WriteCom(0x0c);  //дисплей вкл, курсор и мерцание выключены
  LCD_WriteCom(0x01);  //0b00000001 - очистка дисплея
  _delay_ms(2);
  LCD_WriteCom(0x06);  //0b00000110 - курсор движется вправо, сдвига нет
}
```

```
//функция вывода строки из флэш памяти
void LCD_SendStringFlash(const char *progstr)
{
  unsigned char data = pgm_read_byte(progstr);
  while (data)
  {
    LCD_Wait();
    SetBit(PORT_SIG, RS);
    LCD_CommonFunc(data);
    progstr++;
    data = pgm_read_byte(progstr);
  }
}

//фунция вывода строки из RAM
void LCD_SendString(char *str)
{
  unsigned char data;
  SetBit(PORT_SIG, RS);
  while (*str)
  {
    data = *str++;
    LCD_Wait();
    SetBit(PORT_SIG, RS);
    LCD_CommonFunc(data);
  }
}


void LCD_Clear(void)
{
  LCD_WriteCom(0x01);
  _delay_ms(2);
}
```

event-system.c

```
#include "event-system.h"

//кольцевой буфер
static volatile unsigned char cycleBuf[SIZE_BUF];
static volatile unsigned char tailBuf = 0;
static volatile unsigned char headBuf = 0;
static volatile unsigned char countBuf = 0;
```

```
}

//взять событие
unsigned char ES_GetEvent(void)
{
  unsigned char event;
  if (countBuf > 0){                    //если приемный буфер не пустой
    event = cycleBuf[headBuf];          //считать из него событие
    countBuf--;                         //уменьшить счетчик
    headBuf++;                          //инкрементировать индекс головы буфера
    if (headBuf == SIZE_BUF) headBuf = 0;
    return event;                       //вернуть событие
  }
  return 0;
}

//положить событие
void ES_PlaceEvent(unsigned char event)
{
  if (countBuf < SIZE_BUF){                   //если в буфере еще есть место
      cycleBuf[tailBuf] = event;              //кинуть событие в буфер
      tailBuf++;                              //увеличить индекс хвоста буфера
      if (tailBuf == SIZE_BUF) tailBuf = 0;
      countBuf++;                             //увеличить счетчик
  }
}

//**************************************************************************

volatile unsigned char currentState = 0;

typedef struct PROGMEM
{
    unsigned char state;            //состояние
    unsigned char event;            //событие
    unsigned char nextState;        //следующее состояние
    void (*pStateFunc)(void);
}ROW_TABLE;

void ES_Init(void)
{
  tailBuf = 0;
  headBuf = 0;
  countBuf = 0;
  currentState = STATE_NORMAL;
}
```

```
ROW_TABLE const table[] PROGMEM  = {
//  STATE                    EVENT                  NEXT STATE               STATE_FUNC
   {STATE_NORMAL,       EVENT_KEY_ENTER,       STATE_SET_HOUR,
GUI_SelectHour},
   {STATE_NORMAL,       EVENT_SYS_TIMER,       STATE_NO_CHANGE,
GUI_ChangeTime},

   {STATE_SET_HOUR,     EVENT_KEY_ENTER,       STATE_SET_MINUTE,
GUI_SelectMinute},
   {STATE_SET_HOUR,     EVENT_KEY_UP,          STATE_NO_CHANGE,
GUI_IncHour},
   {STATE_SET_HOUR,     EVENT_KEY_DOWN,        STATE_NO_CHANGE,
GUI_DecHour},

   {STATE_SET_MINUTE,   EVENT_KEY_ENTER,       STATE_NORMAL,
GUI_General},
   {STATE_SET_MINUTE,   EVENT_KEY_UP,          STATE_NO_CHANGE,
GUI_IncMinute},
   {STATE_SET_MINUTE,   EVENT_KEY_DOWN,        STATE_NO_CHANGE,
GUI_DecMinute},
   {       0,               0,                   0,                       NULL}
};

#define prb(data) pgm_read_byte(&(data))
#define prw(data) pgm_read_word(&(data))

void ES_Dispatch(unsigned char currentEvent)
{
   void (*pStateFunc)(void);
   unsigned char i;

   pStateFunc = NULL;

   for (i=0; prb(table[i].state); i++)
   {
      if (prb(table[i].state) == currentState && prb(table[i].event) == currentEvent)
      {
         if (prb(table[i].nextState) != STATE_NO_CHANGE)
           currentState = prb(table[i].nextState);
         pStateFunc = (void *)prw(table[i].pStateFunc);
         break;
      }
   }
```

```
    if (pStateFunc) pStateFunc();
}
```