Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare,Informatică și Microelectronică

Laboratory work nr.2

# REPORT

## At Embedded systems

"General Purpose Input/Output registers on AVR."

st. gr. FAF-141: Cristea Victor

Verified by: Andrei Bragarenco

Chişinău 2016

# Goal of the work:

- Understanding GPIO
- Connecting LED
- Connecting Button
- Integrate LCD Display deviceTask

# Condition:

Write a C program and schematics for **Micro Controller Unit** (MCU) using led which will be turned on by pushing on button and turned off when button is released.

Additionally use LCD Display which will display current state of led.

# Theory

### GPIO

Every micro-controller has GPIO ports. GPIO stands for general purpose input output. These GPIO ports are used to take input on a micro-controller pin or output a value on micro-controller pin.

AVR is 8 bit microcontroller. All its ports are 8 bit wide. Every port has 3 registers associated with it each one with 8 bits. Every bit in those registers configure pins of particular port. Bit0 of these registers is associated with Pin0 of the port, Bit1 of these registers is associated with Pin1 of the port, …. and like wise for other bits.

These three registers are as follows : (x can be replaced by A,B,C,D as per the AVR you are using)

- DDRx register
- PORTx register
- PINx register

### LCD operation

In recent years the LCD is finding widespread use replacing LEDs (seven-segment LEDs or other multisegment LEDs). This is due to the following reasons:

I. The declining prices of LCDs.

2. The ability to display numbers, characters, and graphics. This is in contrast to LEDs, which are limited to numbers and a few characters.

3. Incorporation of a refreshing controller into the LCD, thereby relieving the CPU of the task of refreshing the LCD. In contrast, the LED must be refreshed by the CPU (or in some other way) to keep displaying the data.

4. Ease of programming for characters and graphics.

**LCD pin descriptions**

The LCD discussed in this section has 14 pins. The function of each pin is given in Table 12-1. Figure 12-1 shows the pin positions for various LCDs.

**Vcc, Vss, and Vee**

While V cc and V ss provide +5V and ground, respectively, VEE is used for controlling LCD contrast.

**RS, register select**

There are two very important registers inside the LCD. The RS pin is used for their selection as follows. If RS = 0, the instruction command code register is selected, allowing the user to send commands such as clear display, cursor at home, and so on. If RS = 1 the data register is selected, allowing the user to send data to be displayed on the LCD.

**RJw, read/write**

R/W input allows the user to write information to the LCD or read information from it. R/W = 1 when reading; R/W = 0 when writing.

## DDRx register

DDRx (Data Direction Register) configures data direction of port pins. Means its setting determines whether port pins will be used for input or output. Writing 0 to a bit in DDRx makes corresponding port pin as input, while writing 1 to a bit in DDRx makes corresponding port pin as output.

**Example:**

to make all pins of port A as input pins :
DDRA = 0b00000000;

to make all pins of port A as output pins :
DDRA = 0b11111111;

to make lower nibble of port B as output and higher nibble as input :
DDRB = 0b00001111;

## PINx register

PINx (Port IN) used to read data from port pins. In order to read the data from port pin, first you have to change port's data direction to input. This is done by setting bits in DDRx to zero. If port is made output, then reading PINx register will give you data that has been output on port pins.

Now there are two input modes. Either you can use port pins as tri stated inputs or you can activate internal pull up. It will be explained shortly.

**To read data from port A**.

DDRA = 0x00 //set port for input

X = PINA // input

## PORTx register

PORTx is used for two purposes.

1) To output data : when port is configured as output
2) To activate/deactivate pull up resistors – when port is configures as input

### *To output data*

When you set bits in DDRx to 1, corresponding pins becomes output pins. Now you can write data into respective bits in PORTx register. This will immediately change state of output pins according to data you have written.

In other words to output data on to port pins, you have to write it into PORTx register. However do not forget to set data direction as output.

### To activate/deactivate pull up resistors

When you set bits in DDRx to 0, i.e. make port pins as inputs, then corresponding bits in PORTx register are used to activate/deactivate pull-up registers associated with that pin. In order to activate pull-up resister, set bit in PORTx to 1, and to deactivate (i.e to make port pin tri stated) set it to 0.

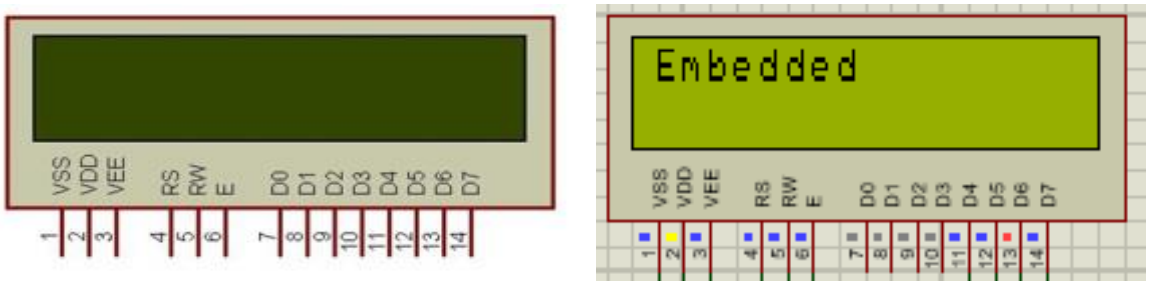
In input mode, when pull1-up is enabled, default state of pin becomes '1'. So even if you don't connect anything to pin and if you try to read it, it will read as 1. Now, when you externally drive that pin to zero(i.e. connect to ground / or pull-down), only then it will be read as 0.

However, if you configure pin as tri state. Then pin goes into state of high impedance. We can say, it is now simply connected to input of some OpAmp inside the uC and no other circuit is driving it from uC. Thus pin has very high impedance. In this case, if pin is left floating (i.e. kept unconnected) then even small static charge present on surrounding objects can change logic state of pin. If you try to read corresponding bit in pin register, its state cannot be predicted. This may cause your program to go haywire, if it depends on input from that particular pin.

Thus while, taking inputs from pins / using micro-switches to take input, always enable pull-up resistors on input pins.

During using on chip ADC, ADC port pins must be configured as tri stated input.

## LCD Display LM016L




**Dimensions**: 16 charracter x 2 lines

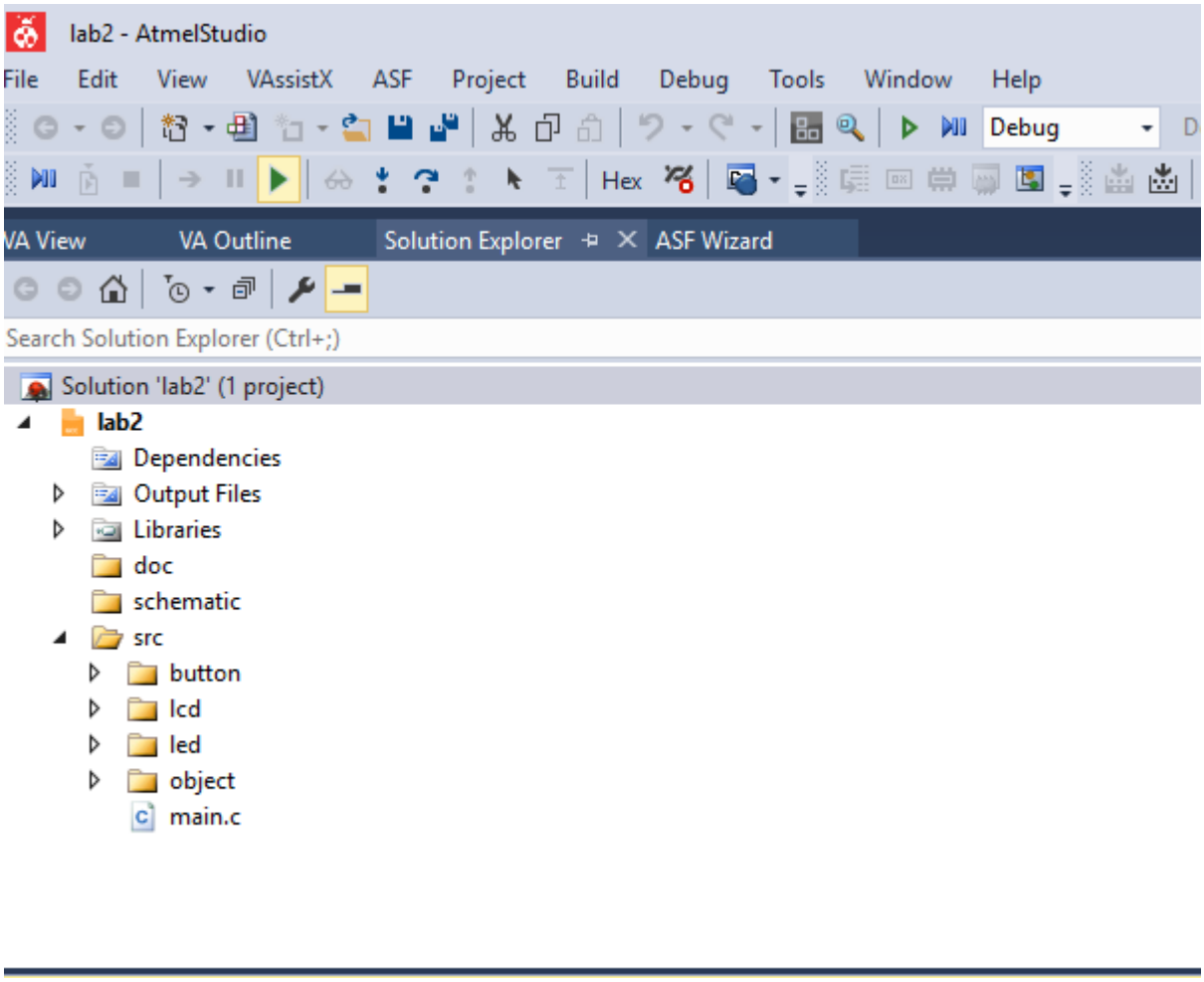
**Power:** +5V

| Pin No. | Name | Description |
|---|---|---|
| Pin no. 1 | **VSS** | Power supply (GND) |
| Pin no. 2 | **VCC** | Power supply (+5V) |
| Pin no. 3 | **VEE** | Contrast adjust |
| Pin no. 4 | **RS** | 0 = Instruction input<br>1 = Data input |
| Pin no. 5 | **R/W** | 0 = Write to LCD Module<br>1 = Read from LCD module |
| Pin no. 6 | **EN** | Enable signal |
| Pin no. 7 | **D0** | Data bus line 0 (LSB) |
| Pin no. 8 | **D1** | Data bus line 1 |
| Pin no. 9 | **D2** | Data bus line 2 |
| Pin no. 10 | **D3** | Data bus line 3 |
| Pin no. 11 | **D4** | Data bus line 4 |
| Pin no. 12 | **D5** | Data bus line 5 |
| Pin no. 13 | **D6** | Data bus line 6 |
| Pin no. 14 | **D7** | Data bus line 7 (MSB) |

# Solving

First of all, to make use LED,Button we should define drivers for each of them. So we will have 3 new drivers: Button,Led,LCD Display.

Generally, **Project Structure** looks in this way




In order to make the program efficient and elegant, I have chosen to represent each connected device to a port of the MCU with a general struct:

```
struct IO_Object {
    uint8_t pinNr;
        volatile uint8_t *ddr;
        volatile uint8_t *ioReg;
};
```

**pinNr -** is the index of the pin at some specific port(A, B, C or D)

**ddr -** configuration on input or output of the whole port

**ioReg-** pin or port - in dependence of the configuration (input or output)

**Led**

In this case, led is nothing more than a Object type device, connected to the MCU. In this laboratory work, the led is connected to PC6 pin. It's initialization in code, looks like this:

```
ObjectInit(&led, PINC6, &DDRC, &PORTC);
```

Two files were created: led.c and led.h in order to organize the code. See the code in the Appendix.

**Button**

In this case, led is nothing more than a Object type device,connected to the MCU. In this laboratory work, the led is connected to PC6 pin. It's initialization in code, looks like this:

```
ObjectInit(&btn, PINC5, &DDRC, &PINC);
```

Two files were created: button.c and led.h in order to organize the code. See the code in the Appendix.

**LCD**

For LCD interfacing I used a library found on internet - written by eXtreme Electronics India**.** For more info, check the link Extreme Elecrtonics.

## Main Program

Main function is the entry point of the program. It works in the following way:

1) Initializes the led and button objects:

```
initObjects();
```

2) Initializes the LCD:

```
LCDInit(LS_BLINK);
```

3) Enters the infinite while loop:

With a frequency of 1000 ms (_delay_ms(1000);)

(a) Clears the display and moves the cursor to home:

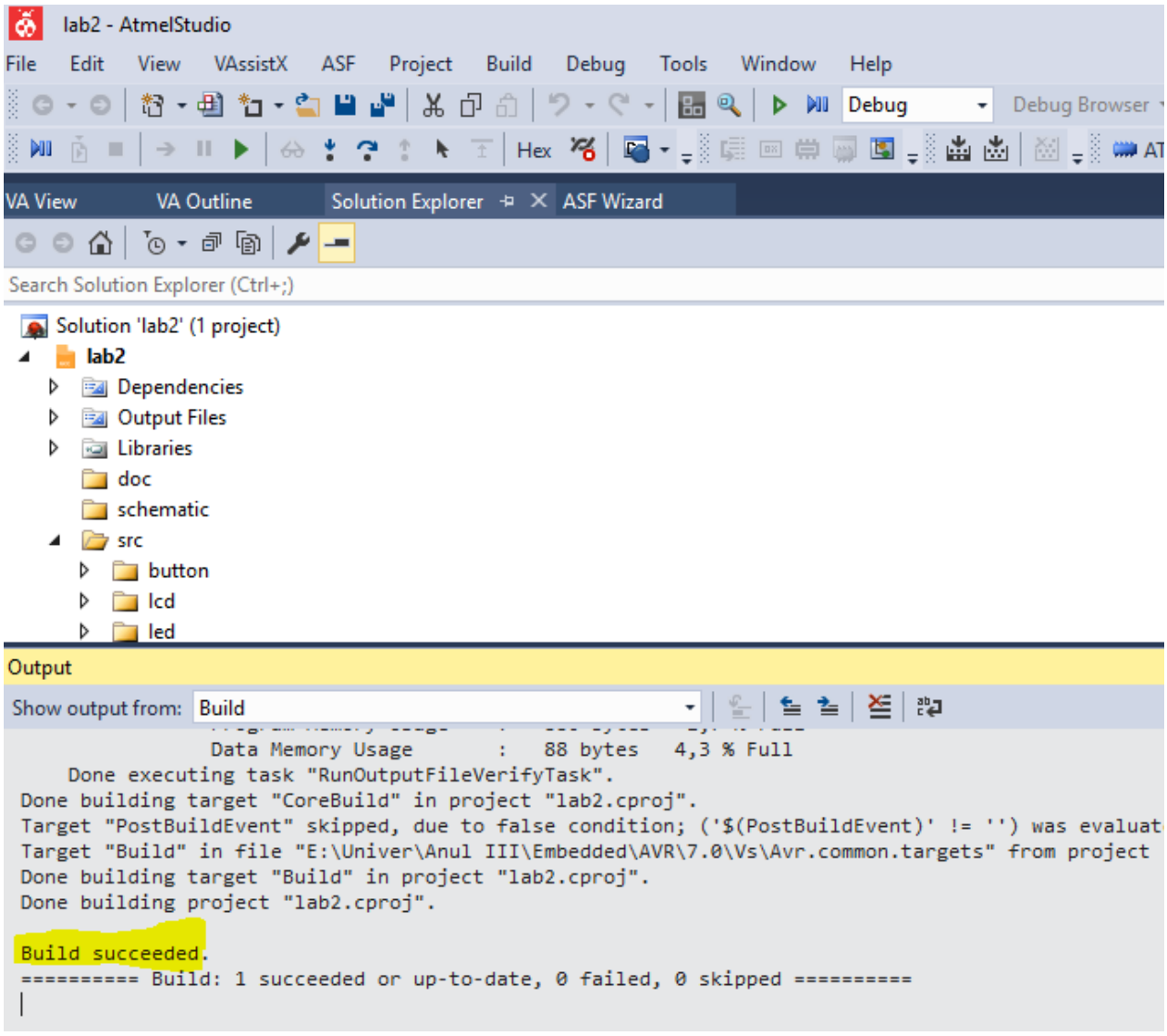
```
LCDClear();

LCDHome()
```

(b) Enables the objects:

```
enableObjects();
```

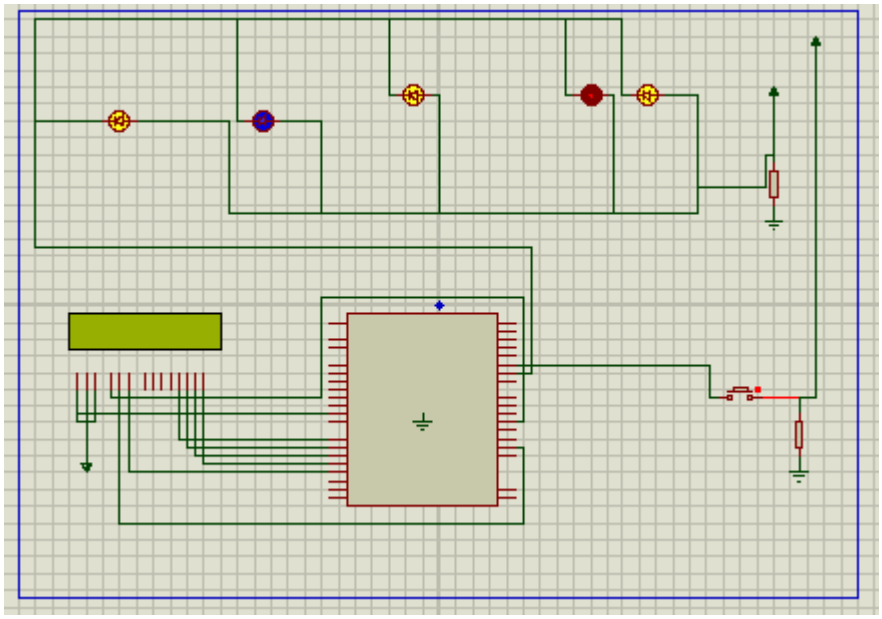
After code implementation, we should now **Build Hex** which will be written to MCU ROM.



## Schematics

For our laboratory work we need

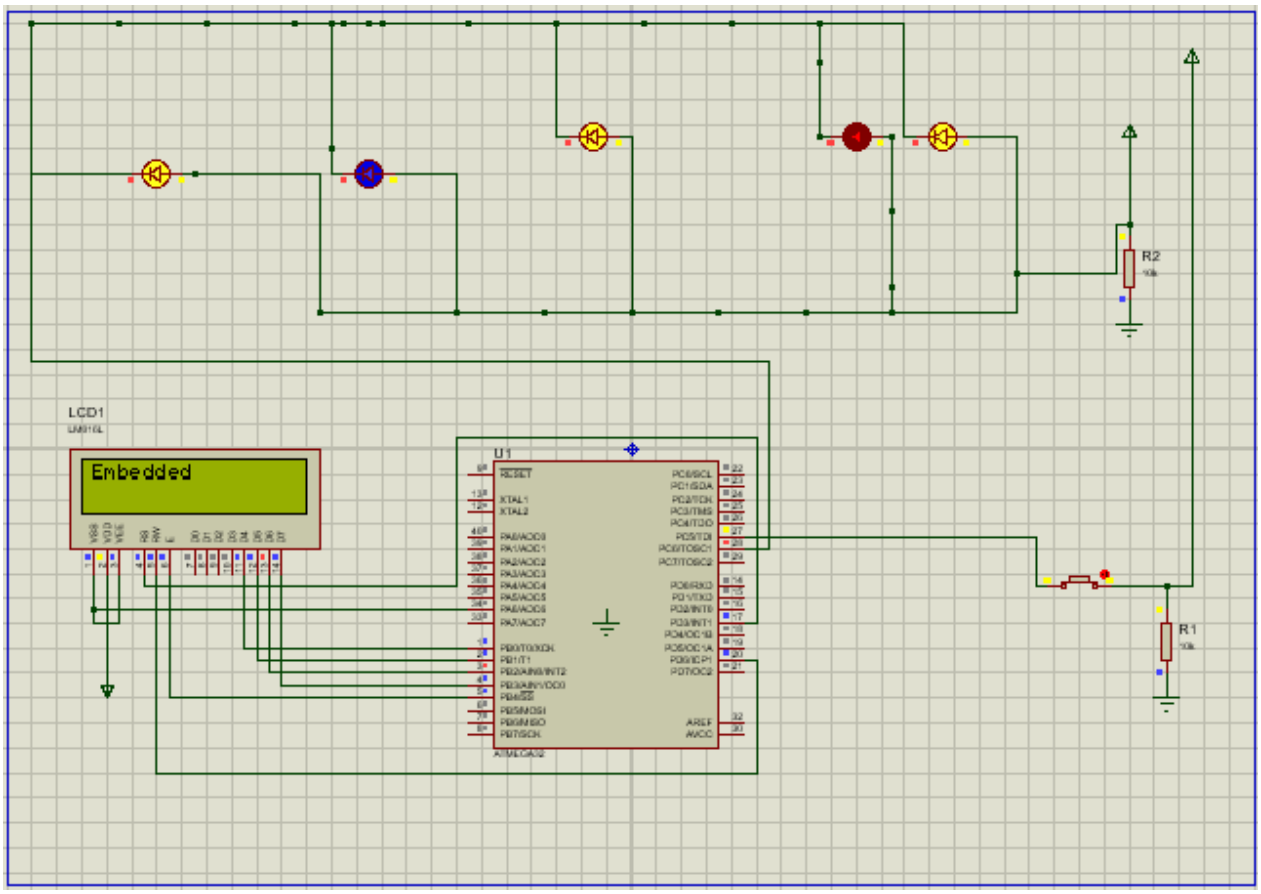
1. ATMega32 MCU
2. LED
3. Push button
4. LCD LM016L (16x2)



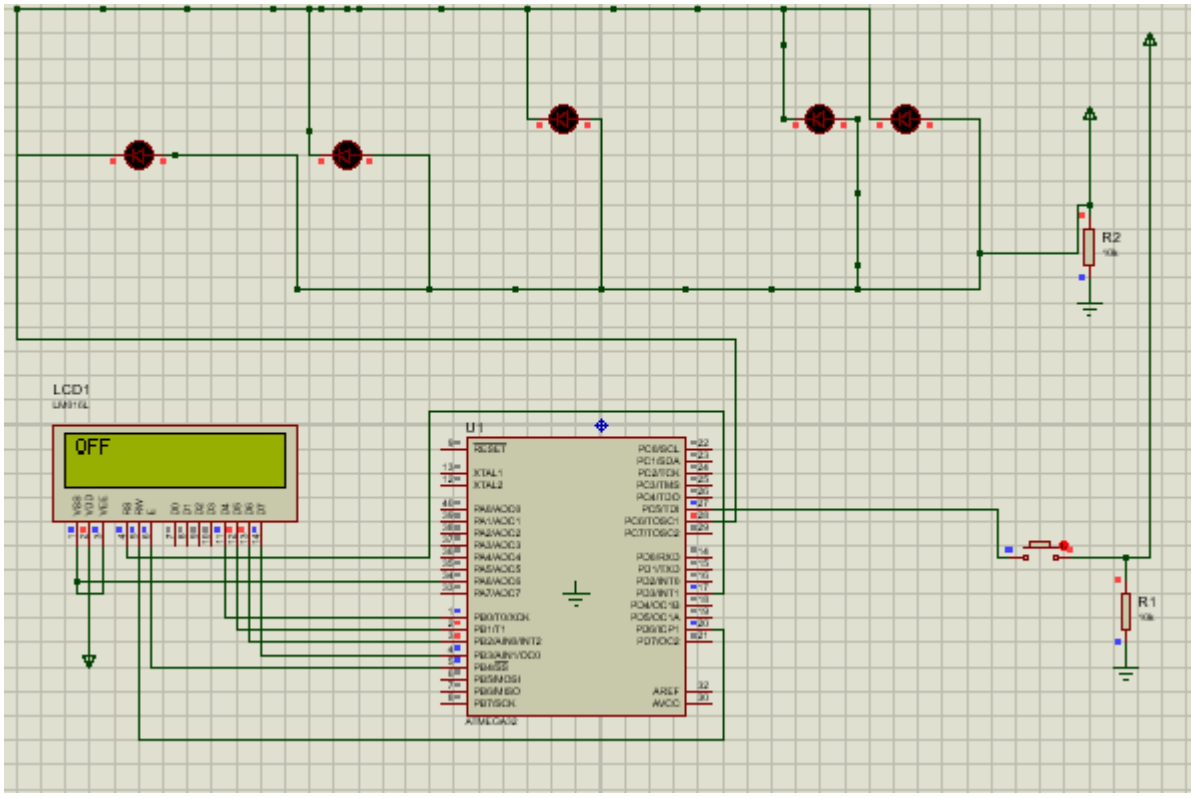
MCU is transmitter and peripheral is receiver. No vice versa connection because we don’t need it in our laboratory work.

## Simulation Result

### Button Pressed




### Button Released



# Conclusion

This laboratory work allowed us to gain the basic knowledge on GPIO and how to work with it. By interfacing the display I was able to display a message, and to prove basics on event based programming. . We gained knowledge on how to connect a peripheral to MC using 4 Atmega Ports, how to configure device for Input or output,how to receive/send data through PIN/PORT registers. Implementing button/led with Configuration Structure, has given us possibility to easy export these drivers and use where you want, with configuration you want. Also there was some key point in this laboratory work like, connecting resistance to LED. As our power source is 5v , but we need 2.5V only for our RED Led.

# Appendix

## main.c

```
#include <avr/delay.h>
#include "led/led.h"
#include "button/button.h"
#include "lcd/lcd_hd44780_avr.h"


   struct IO_Object btn;
   struct IO_Object led;


   void initObjects(void);
   void enableObjects(void);


   int main(void) {


       initObjects();
       LCDInit(LS_BLINK);


       while (1) {
           LCDClear();
           LCDHome();

           enableObjects();

           _delay_ms(100);
       }
   }


   void initObjects(void) {
       ObjectInit(&btn, PINC5, &DDRC, &PINC);
       setObjectDDR(&btn);

       ObjectInit(&led, PINC6, &DDRC, &PORTC);
       setObjectDDR(&led);
   }



   void enableObjects(void) {
       if(isButtonPressed(&btn)) {
           LedOn(&led);
           LCDWriteString("I %4 UTM");
       } else {
           LedOff(&led);
           LCDWriteString("Lights OFF");
       }
   }
```

*In uart folder:*

## button.h

```
#ifndef BUTTON_H_
#define BUTTON_H_

#include "../object/object.h"
   char  isButtonPressed(struct IO_Object *obj);
#endif
```

## button.c

```
#include "button.h"


char  isButtonPressed(struct IO_Object *obj) {
       if((*(obj->ioReg))&(1<<obj->pinNr))
       return 1;
       return 0;
}
```

## led.h

```
#ifndef LED_H_
#define LED_H_


#include "../object/object.h"


void   LedOn(struct IO_Object *obj);
void   LedOff(struct IO_Object *obj);


#endif
```

## Led.c

```
#include "led.h"

void LedOn(struct IO_Object *obj) {
       *(obj->ioReg) &= ~(1<< obj->pinNr);
}

void LedOff(struct IO_Object *obj) {
       *(obj->ioReg) |= (1<< obj->pinNr);
}
```

## object.h

```
#ifndef OBJECT_H_
#define OBJECT_H_

#include <stdint.h>
#include <avr/io.h>

struct IO_Object {
   uint8_t pinNr;
   volatile uint8_t *ddr;
   volatile uint8_t *ioReg;
};

void ObjectInit(struct IO_Object *obj,
   uint8_t _pinNr,
   volatile uint8_t *_ddr,
   volatile uint8_t *_ioReg
);

void setObjectDDRHigh(struct IO_Object *obj);
void setObjectDDRLow(struct IO_Object *obj);

#endif
```

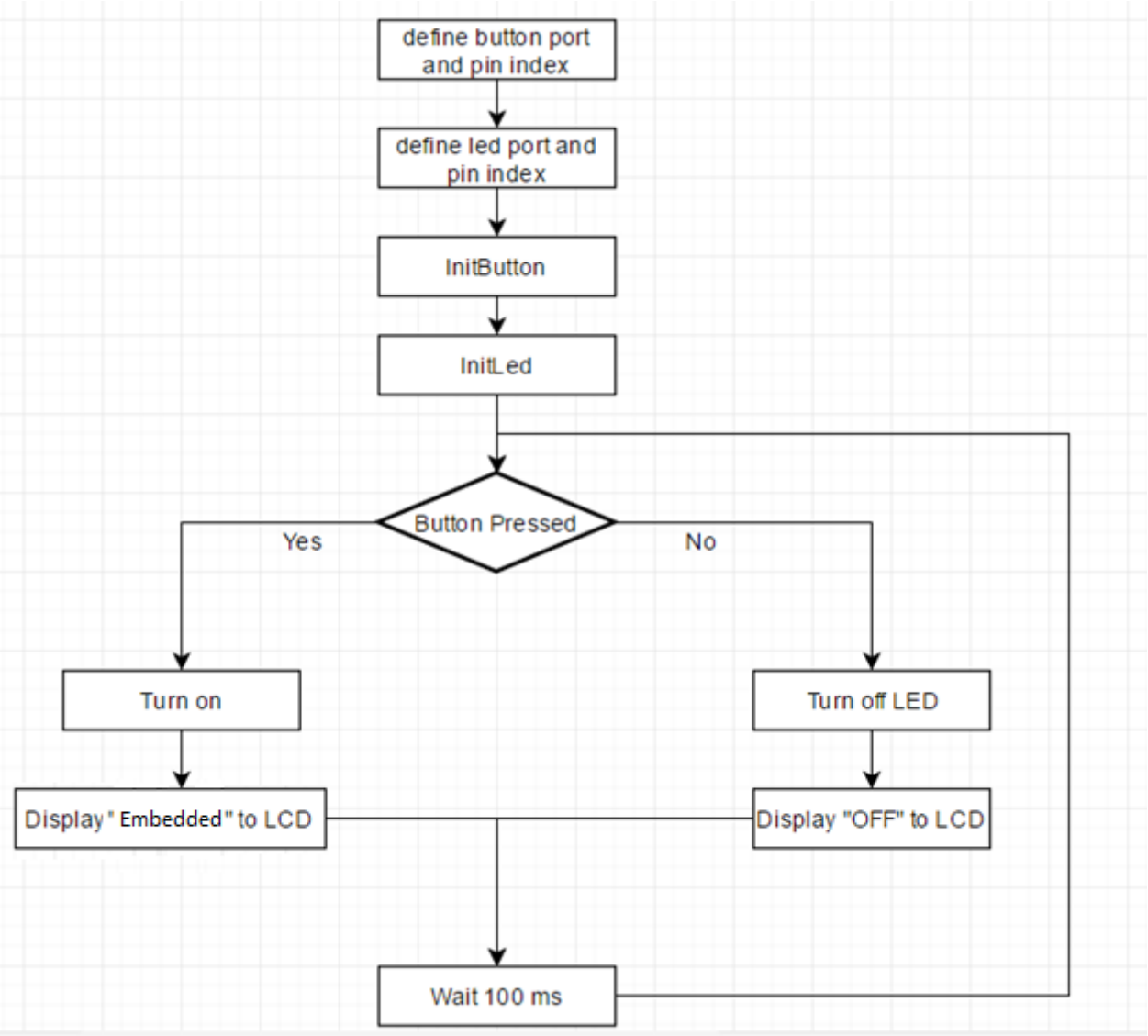
## object.c

```
#include "object.h"

void ObjectInit(struct IO_Object *obj,
   uint8_t _pinNr,
   volatile uint8_t *_ddr,
   volatile uint8_t *_ioReg ) {
          obj->pinNr = _pinNr;
          obj->ddr = _ddr;
          obj->ioReg = _ioReg;
       }

void setObjectDDR(struct IO_Object *obj) {
   *(obj->ddr) |= 1<<obj->pinNr;
}
```

# FlowChart


define button port and pin index
define led port and pin index
InitButton
InitLed
Button Pressed
Yes
No
Turn on
Turn off LED
Display " Embedded " to LCD
Display "OFF" to LCD
Wait 100 ms