Ministerul Educatiei al Republicii Moldova
Universitatea Tehnica a Moldovei
Filiera Anglofona

Report

at Embedded Systems

Laboratory Work #2

Topic: Input/Output Registres. Work with LED, LCD and Button.

Performed by: Diana ARTIOM

Verified by: Andrei BRAGARENCO

Chisinau 2016

Topic:

Input/Output Registres. Work with LED, LCD and Button.

Objectives:

- Understand GPIO
- Interfacing LCD
- Connecting LED
- Connect Button

Task:

Write a program. Work with LED, LCD and use a button to simulate the process of event driven programming.

Overview:

GPIO

General-purpose input/output (GPIO) is a generic pin on an integrated circuit or computer board whose behavior—including whether it is an input or output pin—is controllable by the user at run time.

GPIO pins have no predefined purpose, and go unused by default. The idea is that sometimes a system integrator who is building a full system might need a handful of additional digital control lines—and having these available from a chip avoids having to arrange additional circuitry to provide them. For example, the Realtek ALC260 chips (audio codec) have 8 GPIO pins, which go unused by default.

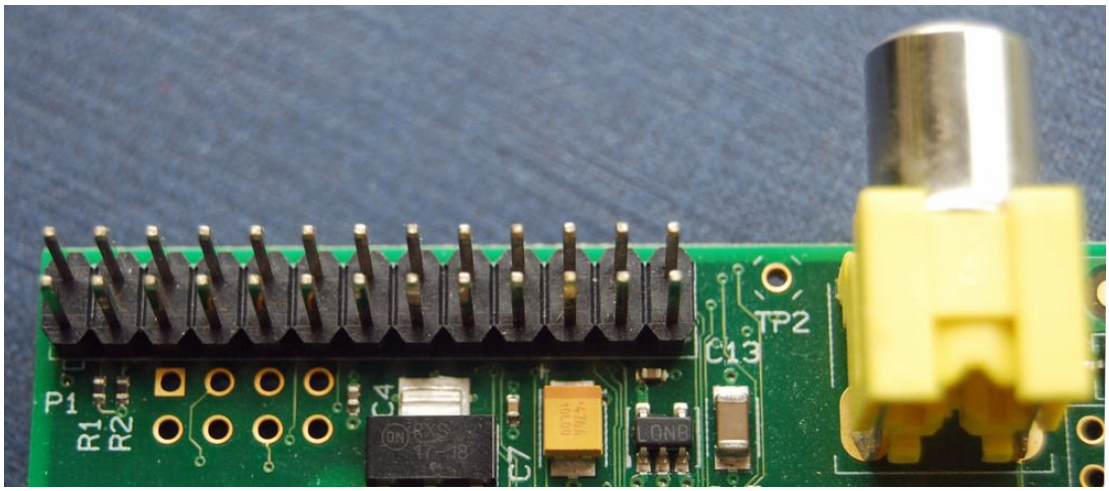

Figure 1: Real life representation GPIO pins

LCD

A **liquid-crystal display** (**LCD**) is a flat-panel display or other electronic visual display that uses the light-modulating properties of liquid crystals. Liquid crystals do not emit light directly. LCDs are available to display arbitrary images (as in a general-purpose computer display) or fixed images with low information content, which can be displayed or hidden, such as preset words, digits, and 7-segment displays, as in a digital clock. They use the same basic technology, except that arbitrary images are made up of a large number of small pixels, while other displays have larger elements.

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.

E, enable

The enable pin is used by the LCD to latch information presented to its data pins. When data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins. This pulse must be a minimum of 450 ns wide.

D0-D7

The 8-bit data pins, DO-D7, are used to send information to the LCD or read the contents of the LCD's internal registers.

Pin	Symbol	I/O	Description
1	V_{SS}	--	Ground
2	V_{CC}	--	+5 V power supply
3	V_{EE}	--	Power supply to control contrast
4	RS	I	RS = 0 to select command register, RS = 1 to select data register
5	R/W	I	R/W = 0 for write, R/W = 1 for read
6	E	I/O	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 8-bit data bus
12	DB5	I/O	The 8-bit data bus
13	DB6	I/O	The 8-bit data bus
14	DB7	I/O	The 8-bit data bus

Figure 2: LCD Pin Description

LCDs are used in a wide range of applications including computer monitors, televisions, instrument panels, aircraft cockpit displays, and indoor and outdoor signage. Small LCD screens are common in portable consumer devices such as digital cameras, watches, calculators, and mobile telephones, including smartphones. LCD screens are also used on consumer electronics products such as DVD players, video game devices and clocks. LCD screens have replaced heavy, bulky cathode ray tube (CRT) displays in nearly all applications. LCD screens are available in a wider range of screen sizes than CRT and plasma displays, with LCD screens available in sizes ranging from tiny digital watches to huge, big-screen television set.

The most commonly used LCDs found in the market today are 1 Line, 2 Line or 4 Line LCDs which have only 1 controller and support at most of 80 charachers, whereas LCDs supporting more than 80 characters make use of 2 HD44780 controllers.

Most LCDs with 1 controller has 14 Pins and LCDs with 2 controller has 16 Pins (two pins are extra in both for back-light LED connections). Pin description is shown in the table below.

In this laboratory work I used the LM016L lcd.

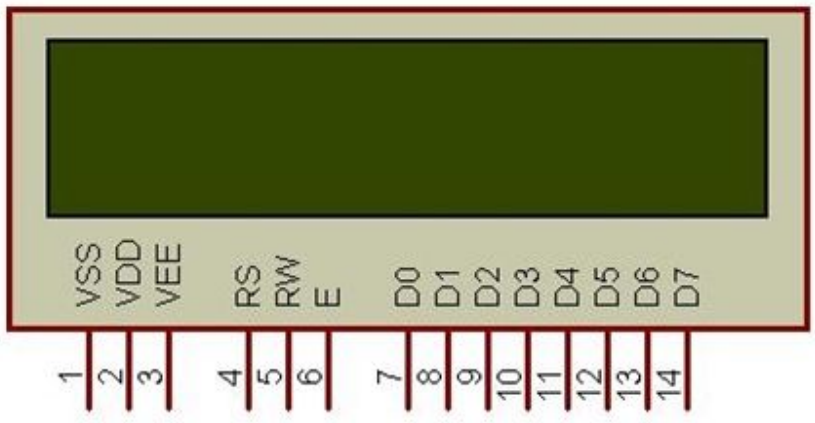

Figure 2: Representation of LM016L

LED

A **light-emitting diode** (**LED**) is a two-lead semiconductor light source. It is a p–n junction diode, which emits light when activated. When a suitable voltage is applied to the leds, electrons are able to recombine with electron holes within the device,

releasing energy in the form of photons. This effect is called electroluminescence, and the color of the light (corresponding to the energy of the photon) is determined by the energy band gap of the semiconductor.

An LED is often small in area (less than 1 mm^2) and integrated optical components may be used to shape its radiation pattern.

Appearing as practical electronic components in 1962, the earliest LEDs emitted low-intensity infrared light. Infrared LEDs are still frequently used as transmitting elements in remote-control circuits, such as those in remote controls for a wide variety of consumer electronics. The first visible-light LEDs were also of low intensity, and limited to red. Modern LEDs are available across the visible, ultraviolet, and infrared wavelengths, with very high brightness. Early LEDs were often used as indicator lamps for electronic devices, replacing small incandescent bulbs. They were soon packaged into numeric readouts in the form of seven-segment displays, and were commonly seen in digital clocks.

AVR microcontrollers such as the ATMega8515 only supply a current of about 20mA and so we can drive an LED directly from the microcontroller port eliminating the resistor. In fact if a resistor is added the intensity of the LED will be low.

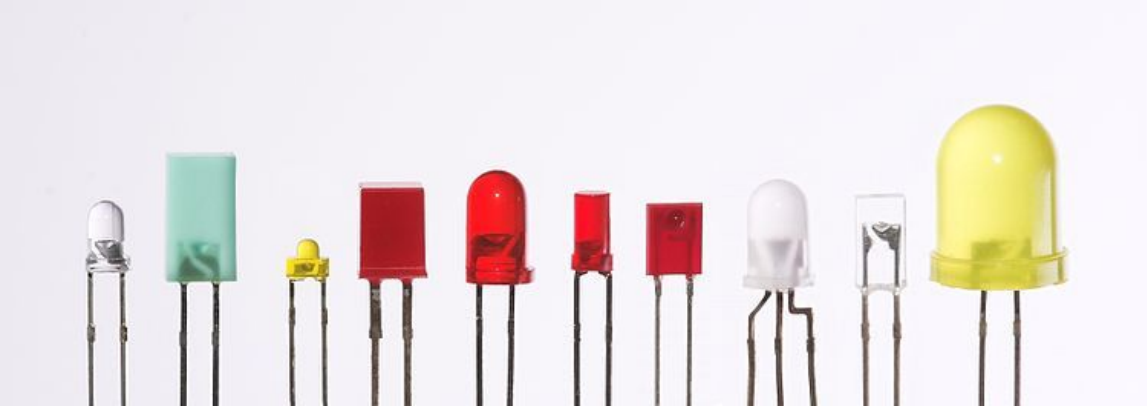

Figure 3: Representation of different LEDs

Tools and Technologies used:

Atmel Studio

Atmel Studio 7 is the integrated development platform (IDP) for developing and debugging Atmel® SMART ARM®-based and Atmel AVR®

microcontroller (MCU) applications. Studio 7 supports all AVR and Atmel SMART MCUs. The Atmel Studio 7 IDP gives you a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code. It also connects seamlessly to Atmel debuggers and development kits.

Additionally, Atmel Studio includes Atmel Gallery, an online apps store that allows you to extend your development environment with plug-ins developed by Atmel as well as by third-party tool and embedded software vendors. Atmel Studio 7 can also able seamlessly import your Arduino sketches as C++ projects, providing a simple transition path from Makerspace to Marketplace.

Proteus

Proteus is a Virtual System Modelling and circuit simulation application. The suite combines mixed mode SPICE circuit simulation, animated components and microprocessor models to facilitate co-simulation of complete microcontroller based designs. Proteus also has the ability to simulate the interaction between software running on a microcontroller and any analog or digital electronics connected to it. It simulates Input / Output ports, interrupts, timers, USARTs and all other peripherals present on each supported processor.

Solution:

In this laboratory work I had to deal with what a understanding of LCD interfacing, led and button usage in an Embedded System.

Before proceeding to explain which is what, I will first include here the project structure of the project:

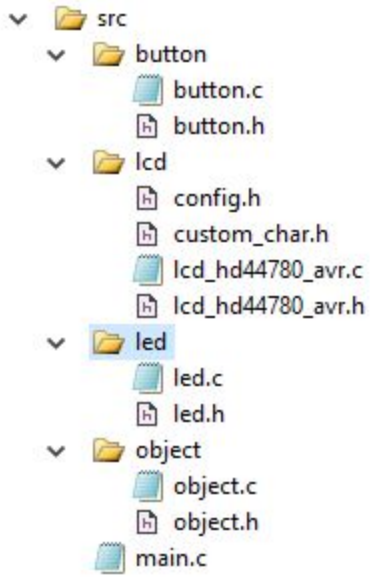

Figure 4: Project structure

Implementations

Object

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

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();
```

The full code can be found in Appendix.

Preparing for Proteus Simulation

After implementing the solution in terms of C code, I compiled it using the **Build Solution** option in Atmel Studio. The path for the program to run on MCU in Proteus, is the file with extension **.hex**, found in .../lab2/lab2/debug folder, with the name lab2.hex:

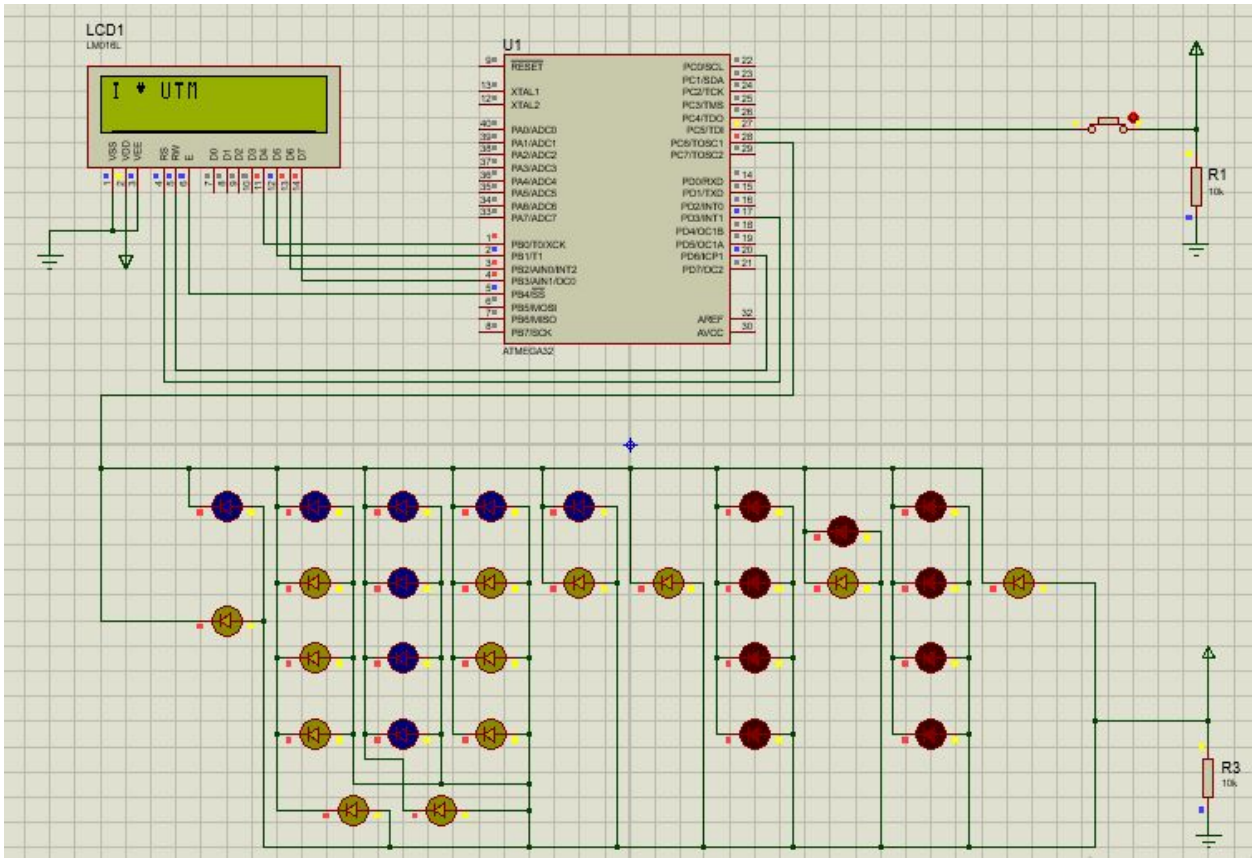

Figure 5: Construction of the scheme - Led, Button and LCD interfacing

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. On the same paradigm - the diodes fired up: when the button is clicked, the pin to which the diode is connected is configured, and the diode start to light. Notice that the I involved here the resistance, to be sure of circuit security.

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;
}
```