

Ministerul Educatiei al Republicii Moldova
Universitatea Tehnica a Moldovei
Filiera Anglofona

Report

Embedded Systems

Laboratory Work #2

Performed by:

Vlad Croitoru

Verified by:

Andrei Bragarenco

Chisinau 2017

Topic:

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

Objectives:

- Getting the basic concepts of GPIO
- Connecting and interfacing LCDs, LEDs, Buttons.

Task:

A simple program that connects a LED to a button, which when pressed lights the LED up and vice versa when released.

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.

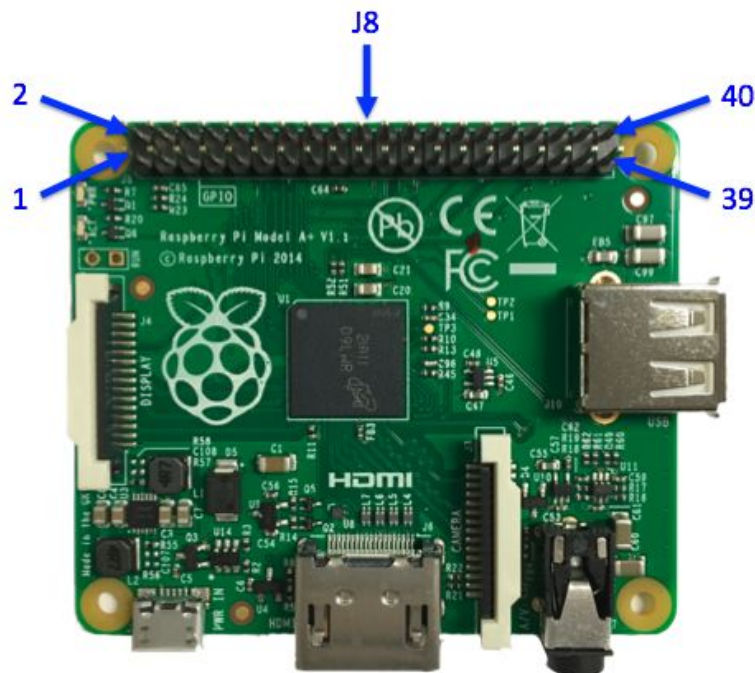


Figure 1: GPIO pins

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.

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

1. 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.

V_{cc}, V_{ss}, and V_{ee}

While V_{cc} and V_{ss} provide +5V and ground, respectively, V_{ee} 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, D0-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 | VSS | - | Ground |
| 2 | VCC | - | +5V power supply |
| 3 | VEE | - | Power supply to control contrast |
| 4 | RS | 1 | RS=0 to select command register, RS=1 to select data register. |
| 5 | R/W | 1 | 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 characters, 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.

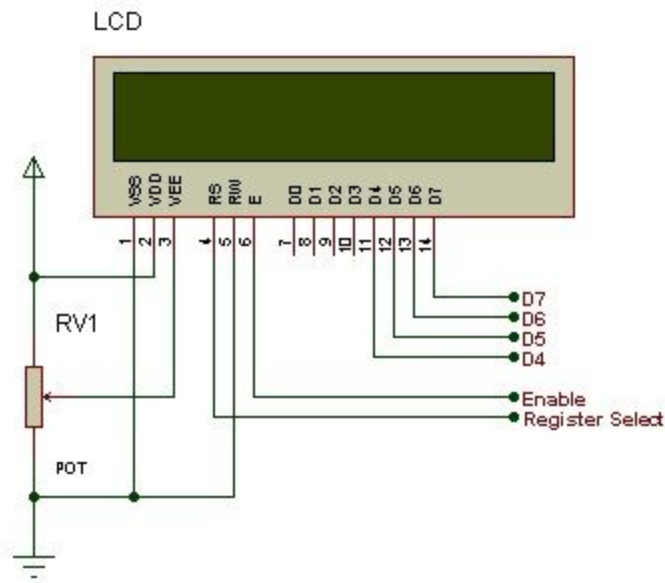


Figure 3: LM016L

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 mm²) 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.

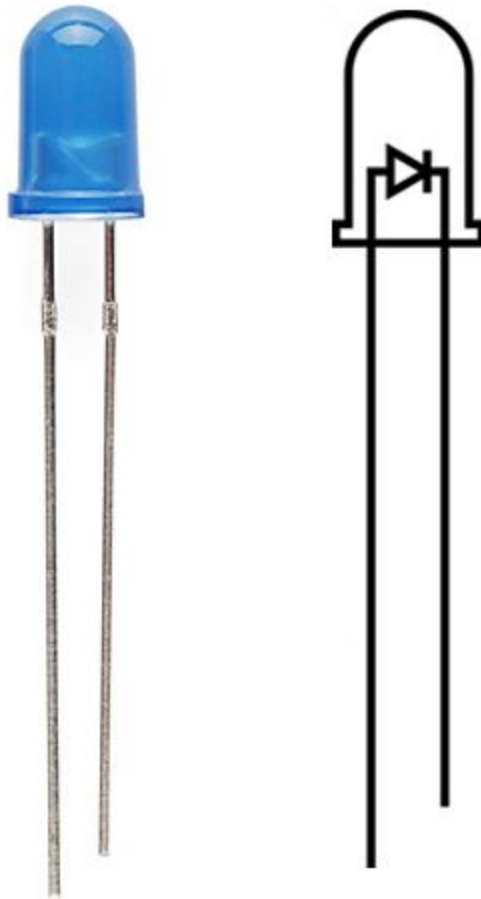


Figure 4: a LED and its notation

Used Tools:

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:

So this time we had to deal with an LCD screen, an led and a button that will be used to interact with the other components.

Implementations

Led

```
void initLed();
```

```
void ledOn();
```

```
void ledOff();
```

Those are used to initiate an LED instance and then turn it on/off.

Button

```
int isPressed();
```

```
void init();
```

As well as with LED, first of all initiate and instance then in an infinite loop check if the button is pressed.

main

The main function is again an infinite loop as we are required to write an event driven software.

So first of all we define configurations for the button and for the LED, then we initialize them (again button and LED), start the infinite loop which checks if the button is pressed, and if it is lights up the LED, with a 100 ms delay.

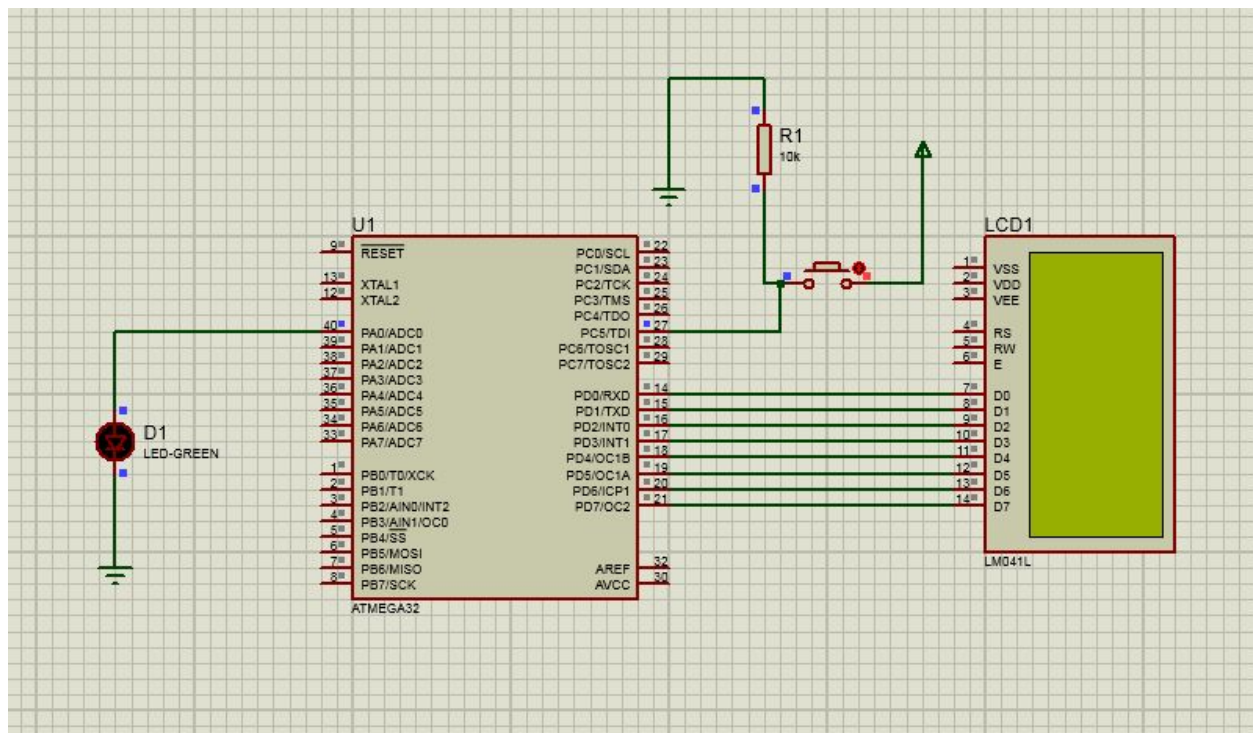


Figure 5: The initial scheme

The result:

When the button is pressed we have the following result.

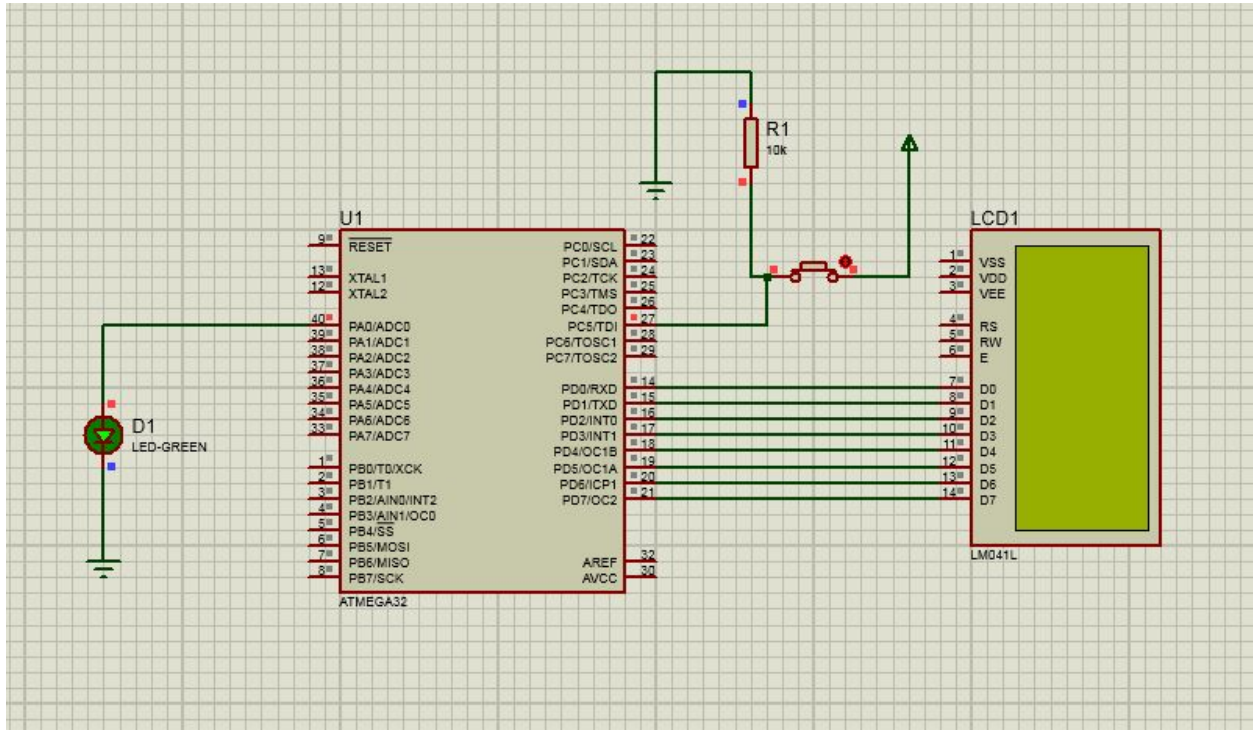


Figure 6: LED is on

Conclusion:

Gained basic knowledge of GPIO, different components that can be used with the ATMEGA32 microcontroller such as LEDs, LCD displays and buttons. The button can be in two states and works like a switch, while the LED is a semiconductor that we integrated in our scheme and making them interact which was our main task.

Appendix:

main

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

int main() {

    init();
    initLed();

    while(1) {
        _delay_ms(100);
        if(isPressed()) {
            ledOn();
        } else {
            ledOff();
        }
    }

    return 0;
}
```

led.h

```
#ifndef LED_H_
#define LED_H_
#include <avr/io.h>

void initLed();
void ledOn();
void ledOff();

#endif /* LED_H_ */
```

led.c

```
#include "led.h"

void initLed() {
    DDRA |= (1 << PORTA0);
}

void ledOn() {
    PORTA |= (1 << PORTA0);
}

void ledOff() {
    PORTA &= ~(1 << PORTA0);
}
```

button.h

```
#ifndef BUTTON_H_
#define BUTTON_H_
#include <avr/io.h>

int isPressed();
void init();

#endif /* BUTTON_H_ */
```

button.c

```
#include "button.h"

void init() {
    DDRC &= ~(1 << PORTC5) ;
}

int isPressed() {
    return PINC & (1<<PORTC5);
}
```