Faculty of Computers, Informatics and Microelectronics
Technical University of Moldova

REPORT

Laboratory work #6

Embedded Systems

Performed by:
st. gr. FAF – 141, Postica Denis

Verified by:
Senior Lecturer, Andrei Bragarenco

Chișinău 2017

Topic:

Individual application.

Objectives:

Use of accumulated knowledge.

Task:

Develop an application which would contain methods used in previous laboratory works.

Overview:

General-purpose input/output

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. Some system integrators (Acer Inc. laptops) use the first GPIO (GPIO0) on the ALC260 to turn on the amplifier for the laptop's internal speakers and external headphone jack.

Usage:

Manufacturers use GPIOs in:

a. Devices with pin scarcity: integrated circuits such as system-on-a-chip, embedded and custom hardware, and programmable logic devices (for example, FPGAs)

b. Multifunction chips: power managers, audio codecs, and video cards

c. Embedded applications (Arduino, BeagleBone, PSoC kits, Raspberry Pi,[3] etc.) use GPIO for reading from various environmental sensors (IR, video, temperature, 3-axis orientation, and acceleration), and for writing output to DC motors (via PWM), audio, LC displays, or LEDs for status.

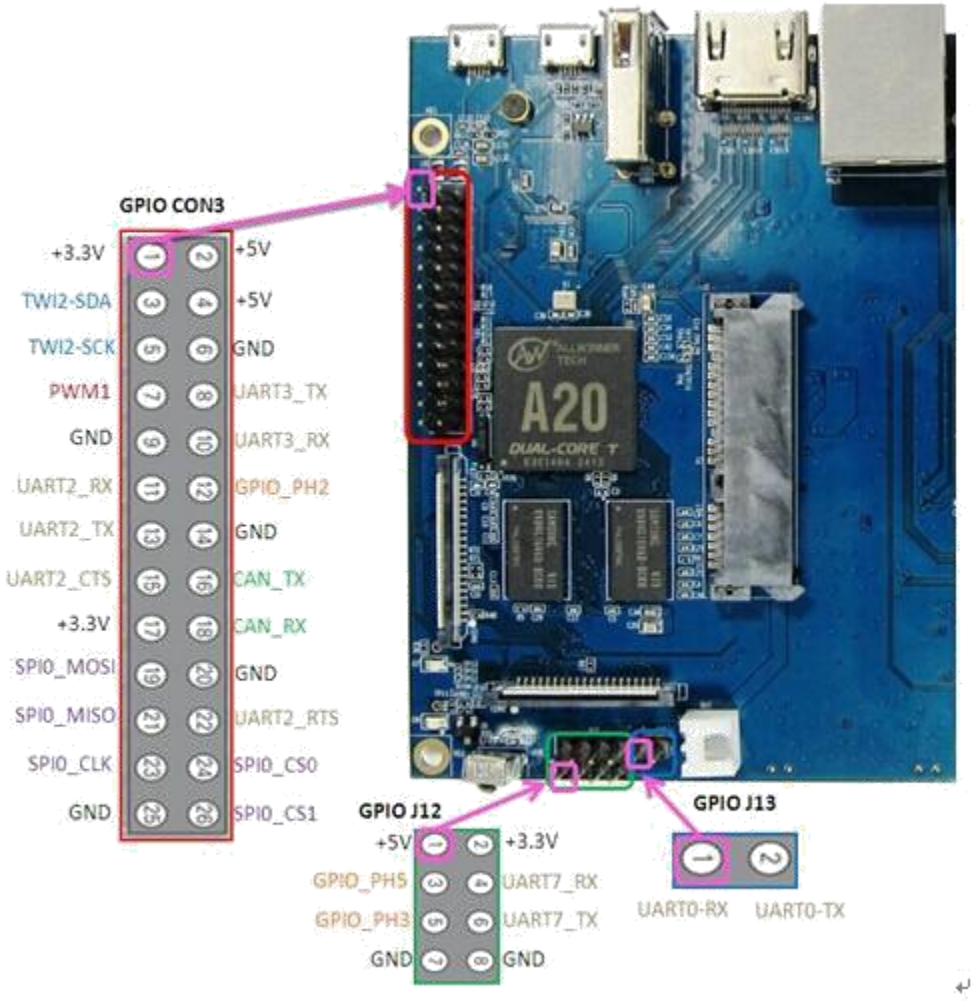

Figure 1. Male GPIO pins.

LED:

A light-emitting diode (LED) is a semiconductor device that emits visible light when an electric current passes through it. The light is not particularly bright, but in most LEDs it is monochromatic, occurring at a single wavelength. The output from an LED can range from red (at a wavelength of approximately 700 nanometers) to blue-violet (about 400 nanometers). Some LEDs emit infrared (IR) energy (830 nanometers or longer); such a device is known as an infrared-emitting diode (IRED).

An LED or IRED consists of two elements of processed material called P-type semiconductors and N-type semiconductors. These two elements are placed in direct contact, forming a region called the P-N junction. In this respect, the LED or IRED resembles most other diode types, but there are important differences. The LED or IRED has a transparent package, allowing visible or IR energy to pass

through. Also, the LED or IRED has a large PN-junction area whose shape is tailored to the application.

Benefits of LEDs and IREDs, compared with incandescent and fluorescent illuminating devices, include:

1. Low power requirement: Most types can be operated with battery power supplies.

2. High efficiency: Most of the power supplied to an LED or IRED is converted into radiation in the desired form, with minimal heat production.

3. Long life: When properly installed, an LED or IRED can function for decades.

Typical applications include:

1. Indicator lights: These can be two-state (i.e., on/off), bar-graph, or alphabetic-numeric readouts.

2. LCD panel backlighting: Specialized white LEDs are used in flat-panel computer displays.

3. Fiber optic data transmission: Ease of modulation allows wide communications bandwidth with minimal noise, resulting in high speed and accuracy.

4. Remote control: Most home-entertainment "remotes" use IREDs to transmit data to the main unit.

5. Optoisolator: Stages in an electronic system can be connected together without unwanted interaction.

Tools and technologies:

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

The Proteus Design Suite is a Windows application for schematic capture, simulation, and PCB layout design. It can be purchased in many configurations, depending on the size of designs being produced and the requirements for microcontroller simulation. All PCB Design products include an autorouter and basic mixed mode SPICE simulation capabilities.

Schematic capture in the Proteus Design Suite is used for both the simulation of designs and as the design phase of a PCB layout project. It is therefore a core component and is included with all product configurations.

The micro-controller simulation in Proteus works by applying either a hex file or a debug file to the microcontroller part on the schematic. It is then co-simulated along with any analog and digital electronics connected to it. This enables it's used in a broad spectrum of project prototyping in areas such as motor control, temperature control and user interface design. It also finds use in the general hobbyist community and, since no hardware is required, is convenient to use as training or teaching tool.

Application development:

AVR Studio

led.h

Header file where are declared led related function like:

```
void ledInit();
void ledGoLeft(char leftArrow[]);
void ledGoRight(char rightArrow[]);
```

led.c

The file where functions defined in led header file are implemented.
First one is used to initialize led, meaning sets direction of DDRB and DDRA to out (1).
The second one is used to animate left moving arrow.
The third one is used to animate right moving arrow.

button.h

Header file where are declared button related function like:

```
void buttonInit();
int buttonIsPressed()
```

button.c

The file where functions defined in button header file are implemented.
First one is used to initialize button, meaning sets direction of DDRC to in (0).
The second one is used to set return the value of PORTC0.

lab6.c

Here is checked if button is pressed and if is pressed the animated arrow on led goes right, otherwise left.

Proteus ISIS

After twice building the code there is created a file with .hex extension which will be used in Proteus application scheme.

The scheme of the application can be seen in figure 2.

To be able to run the application we have to specify for ATmega32 the path to the hex file.

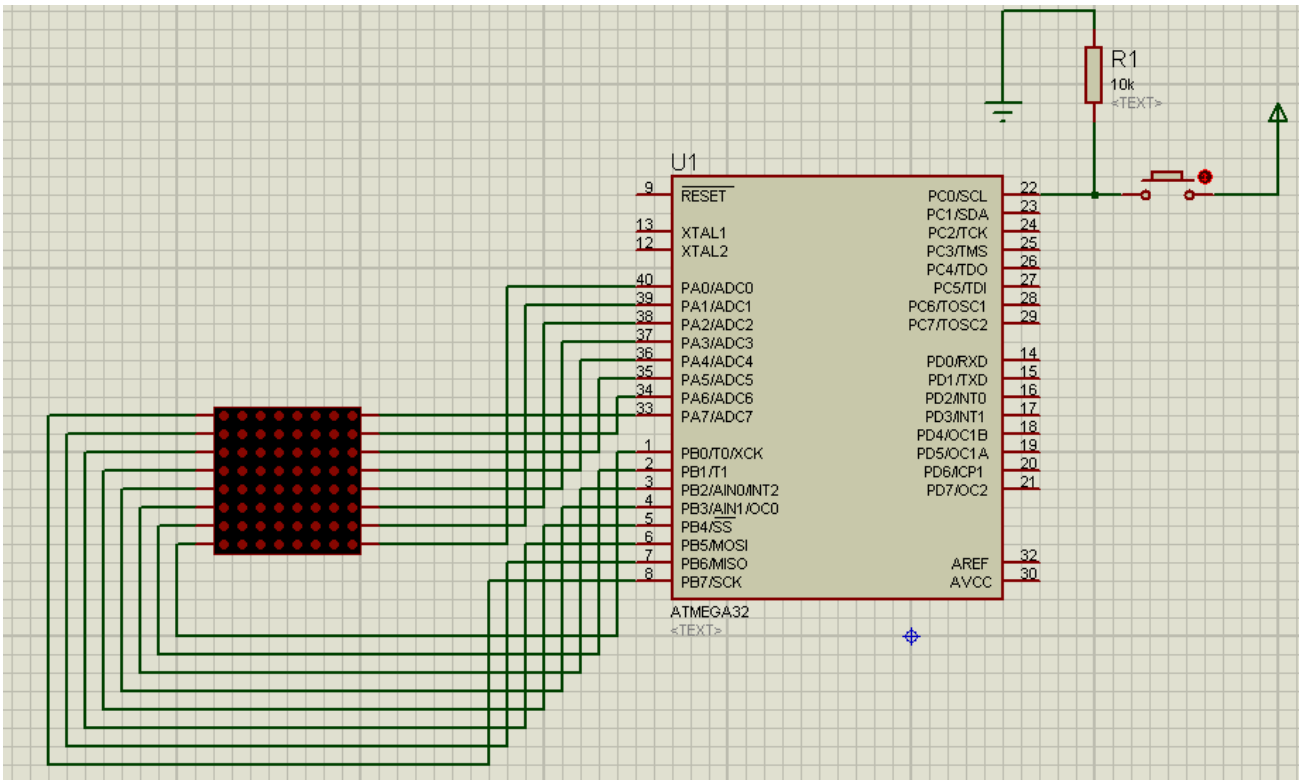

Figure 2. Scheme of the application in Proteus ISIS.

When running the application we can observe that when button is pressed the animated arrow on led points right, otherwise left (see fig. 3,4).

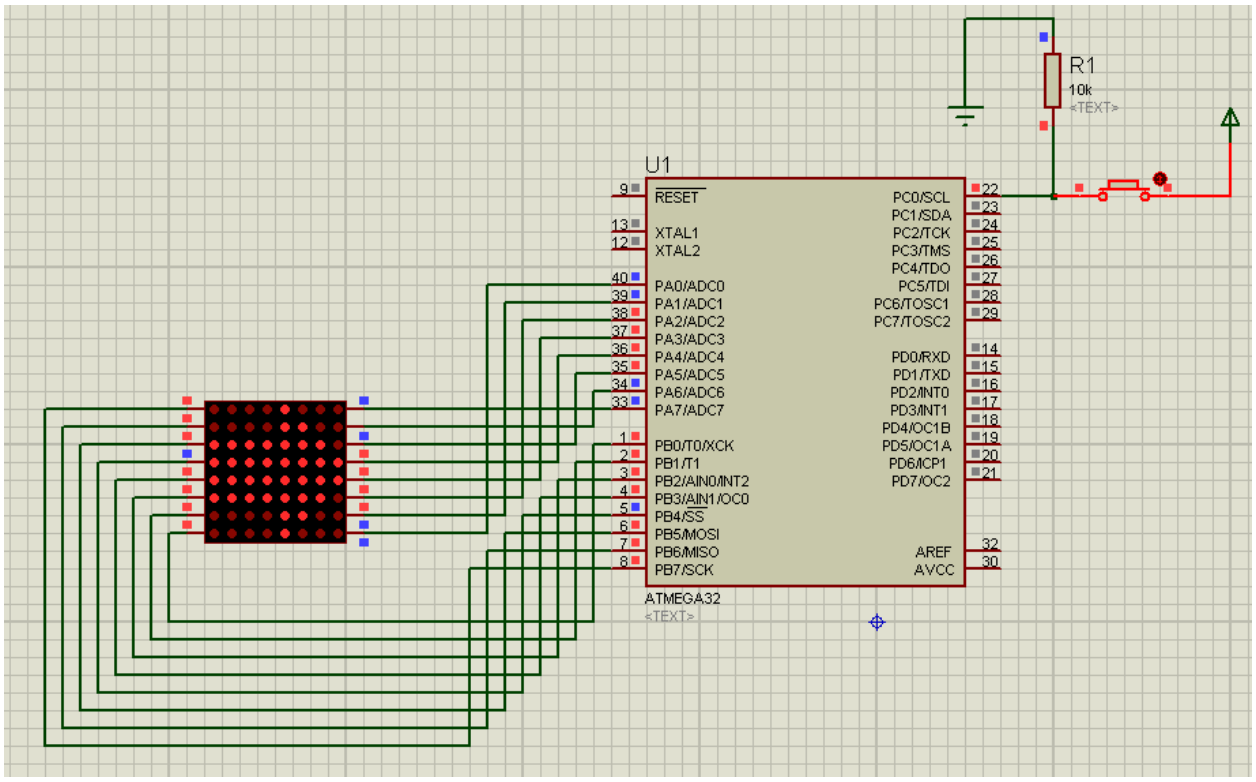

Figure 3. Button pressed.

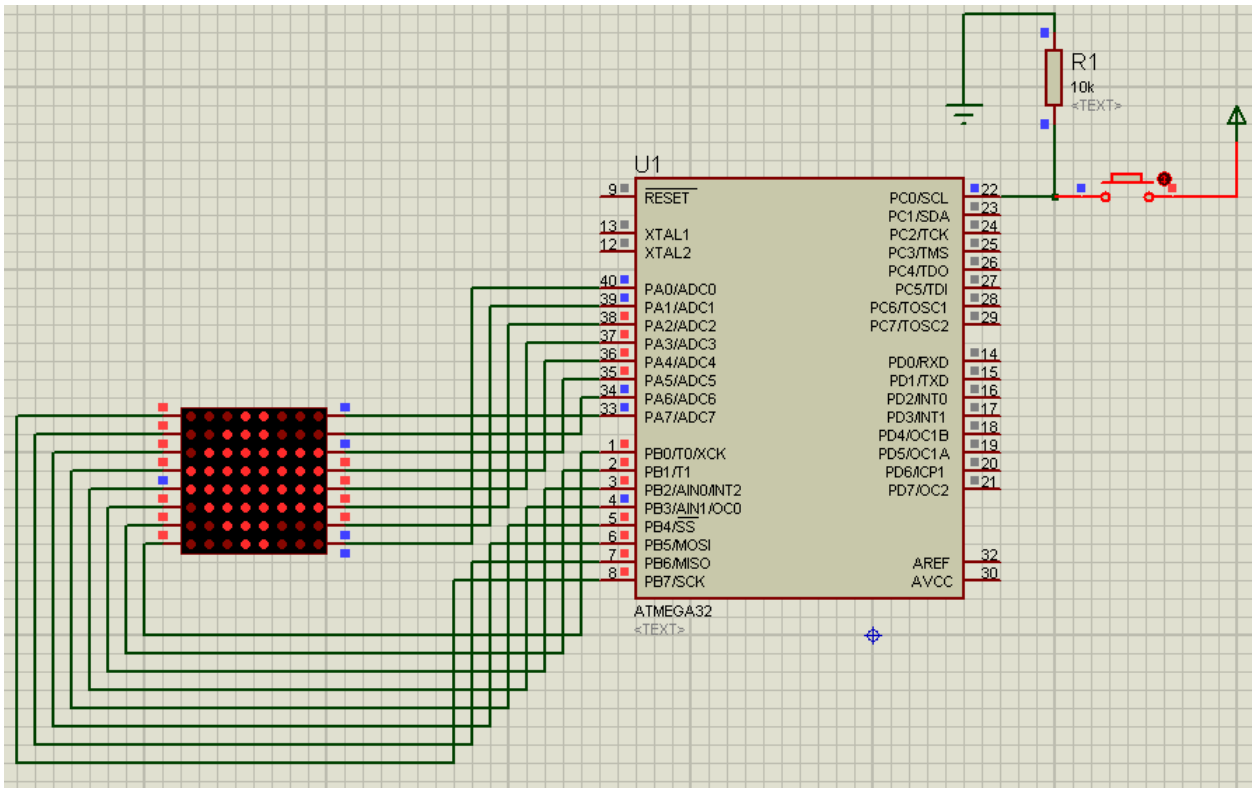

Figure 4. Button released

Appendix:

led.h

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

char PORT[8];

void ledInit();
void ledGoLeft(char leftArrow[]);
void ledGoRight(char rightArrow[]);

#endif /* LED_H_ */
```

led.c

```
#include <util/delay.h>  //delay header
#include "led.h"

char PORT[8] = {1,2,4,8,16,32,64,128};

void ledInit(void) {

    DDRB = 0xFF; //PORTB as output
    DDRA = 0xFF; //PORTA as output
}

void ledGoLeft(char leftArrow[]) {

    for(int x=17;x>0;x--) // shift values of leftArrow[] after each loop execution
        {
            for(int a=0;a<20;a++) //show each character 20 times before shifting a column
                {

                    for (int i=0;i<8;i++)
                    {
                        PORTB = ~PORT[i];    //ground the PORTB pin
                        PORTA = leftArrow[i+x];  //power the PORTA
                        _delay_ms(0.5);
                        PORTB = PORT[i];     //clear pin after 0.5 msecs
                    }

                }
        }

}

void ledGoRight(char rightArrow[]) {

    for(int x=0;x<18;x++) // shift values of rightArrow[] after each loop execution
        {
            for(int a=0;a<20;a++) //show each character 20 times before shifting a column
                {

                    for (int i=0;i<8;i++)
                    {
                        PORTB = ~PORT[i];    //ground the PORTB pin
                        PORTA = rightArrow[i+x];  //power the PORTA
                        _delay_ms(0.5);
                        PORTB = PORT[i];     //clear pin after 0.5 msecs
                    }

                }
        }

}
```

button.h

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

void buttonInit();
int buttonIsPressed();

#endif /* BUTTON_H_ */
```

button.c

```
#include "button.h"

void buttonInit() {
    DDRC &= ~(1 << PORTC0) ;    /* sets direction of DDRC to in (0).*/
}

int buttonIsPressed() {

    return PINC & (1<<PORTC0); /* returns the value of PORTC0*/
}
```

lab6.c

```
#include "button.h"
#include "led.h"

char rightArrow[]={
    0,0,0,0,0,0,0,0,
    0,0b00011000, 0b00111100, 0b01111110, 0b11111111, 0b00111100, 0b00111100, 0b00111100, 0b00111100,
    0,0,0,0,0,0,0,0

    };

char leftArrow[]={
    0,0,0,0,0,0,0,0,
    0,0b00111100, 0b00111100, 0b00111100, 0b00111100, 0b11111111, 0b01111110, 0b00111100, 0b00011000,
    0,0,0,0,0,0,0,0

    };

int main(void)
{
    ledInit();
    buttonInit();

    while(1)
    {
        if (buttonIsPressed()) {

            ledGoRight(rightArrow); }

        else {

            ledGoLeft(leftArrow);
        }
    }
}
```