Faculty of Computers, Informatics and Microelectronics
Technical University of Moldova

REPORT

Laboratory work #2

Embedded Systems

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

Verified by:
Senior Lecturer, Andrei Bragarenco

Chișinău 2017

Topic:

User interface for a LED using a button.

Objectives:

Introduction to basics of peripheral module GPIO.
Basics of connecting LEDs and buttons to ATmega32.

Task:

Develop an application which would turn on a LED on pressing a button.

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.

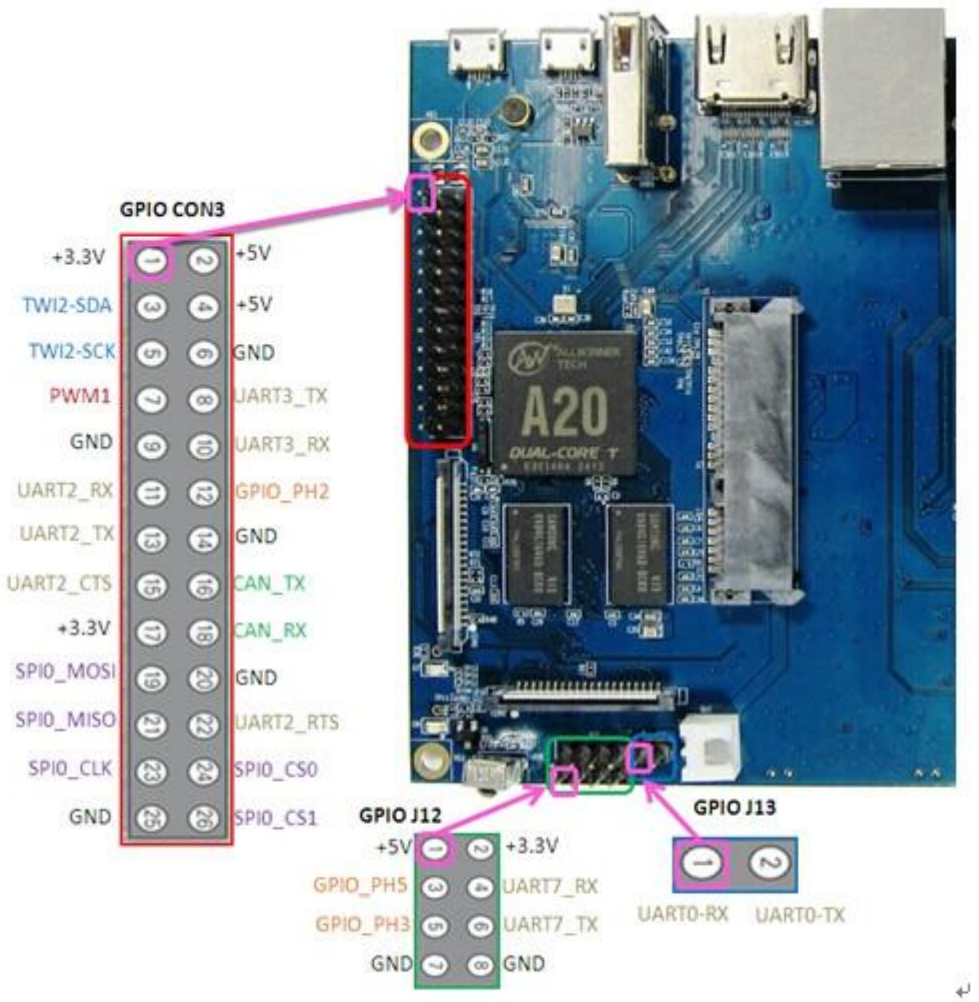


Figure 1. Male GPIO pins.

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

uart_stdio.h

Header file where are declared uart_stdio functions like:

```
void uart_stdio_Init(void);
int uart_stdio_PutChar(char c, FILE *stream);
```

uart_stdio.c

The file where functions defined in header file are implemented.
First one is used to initialize uart_stdio (increases baude rate* if needed, sets UART speed and enables RX and TX).
The second one is used to output a character.

led.h

Header file where are declared led related function like:

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

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 to out (1).
The second one is used to set value 1 for PORTB1.
The third one is used to set value 0 for PORTB1.

button.h

Header file where are declared button related function like:

```
void init();
int isPressed();
```

button.c

The file where functions defined in button header file are implemented.
First one is used to initialize button, meaning sets direction of DDRB to in (0).
The second one is used to set return the value of PORTA0.

lab2.c

1. In main file initially are included uart_stdio.h, led.h, button.h to be able to use its functions. Also, library <avr/delay.h> to be able to use delay function.

2. Main starts with calling **`init()`** and **`initLed()`**, used to initialize button and LED, respectively.

3. Program enters the infinite loop. In the infinite loop are called sequentially:
 a. if(isPressed()), checks if button is pressed. If true it is called ledOn(), otherwise ledOff().

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 (see fig. 3).

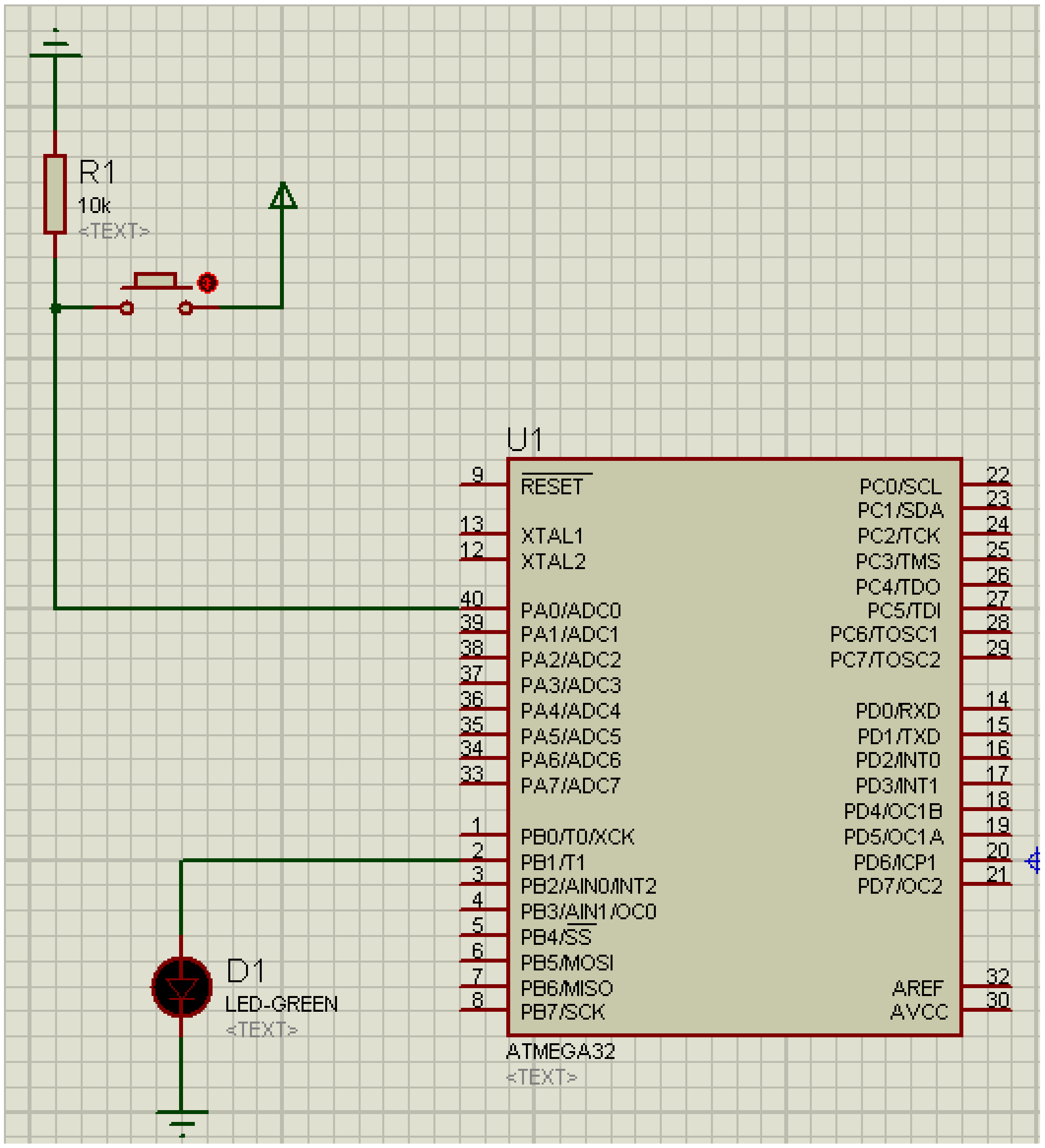
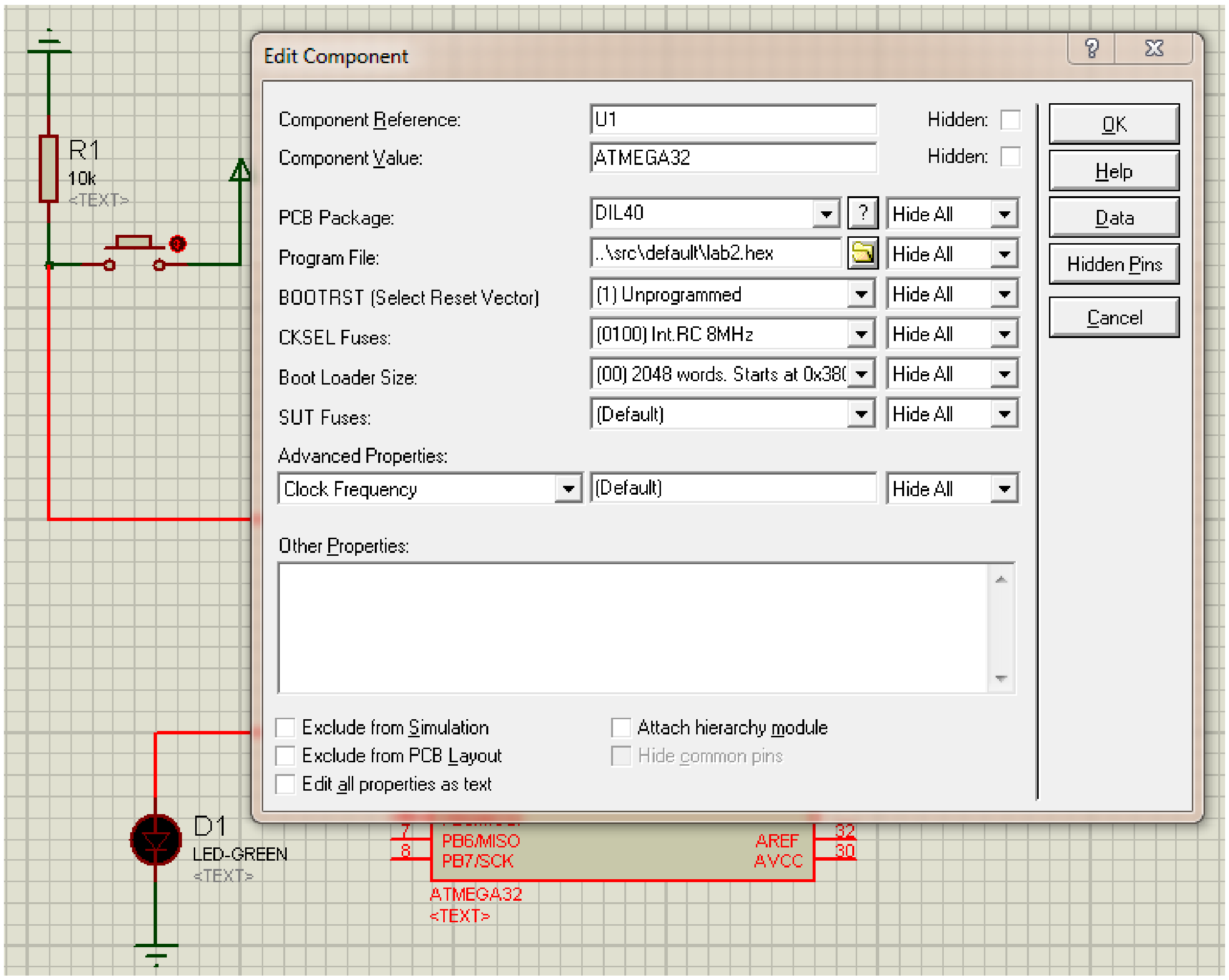


Figure 2. Scheme of the application in Proteus ISIS.



Component Reference:	U1	Hidden: ☐	
Component Value:	ATMEGA32	Hidden: ☐	
PCB Package:	DIL40	Hide All	
Program File:	..\src\default\lab2.hex	Hide All	
BOOTRST (Select Reset Vector)	(1) Unprogrammed	Hide All	
CKSEL Fuses:	(0100) Int.RC 8MHz	Hide All	
Boot Loader Size:	(00) 2048 words. Starts at 0x38(	Hide All	
SUT Fuses:	(Default)	Hide All	
Advanced Properties: Clock Frequency	(Default)	Hide All	

Other Properties:

☐ Exclude from Simulation ☐ Attach hierarchy module
☐ Exclude from PCB Layout ☐ Hide common pins
☐ Edit all properties as text

Figure 3. Path to the hex file.

When running the application we can observe that when button is pressed the LED is on, becomes green (see fig. 4).

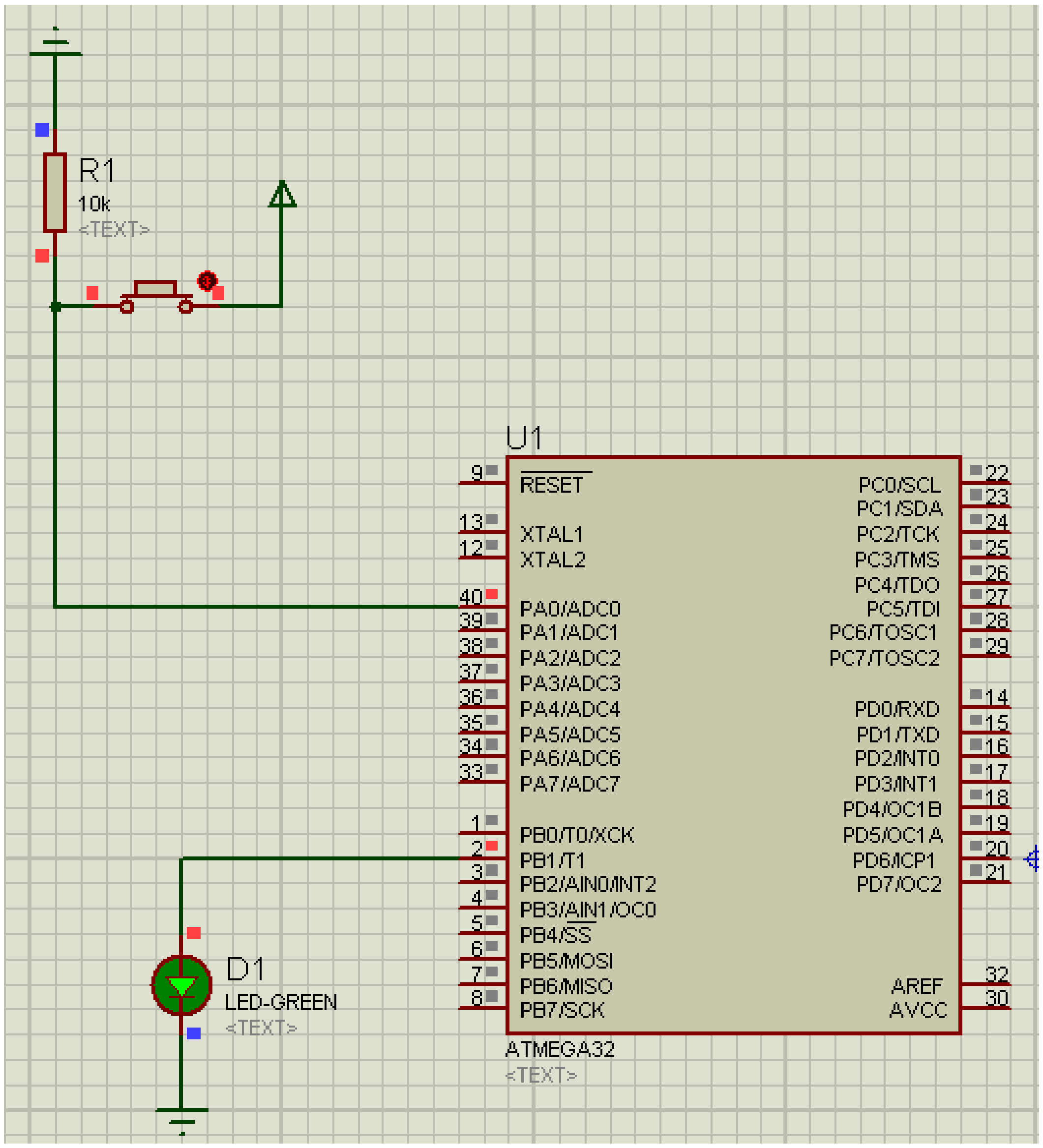


Figure 4. Button pressed.

Conclusions:

The laboratory work #2 represented my first experience with GPIO, LED and buttons AVR microcontroller programming. I learned the basics of GPIO programming in AVR. Also, I discovered that GPIO pins are grouped in 8, there are three definitions when working with pins: PINX – logical value to the physical pin; DDRX – used to configure the direction of pins and PORTX – generator of logical level for pins.

Appendix:

uart_stdio.h

```
#ifndef _UART_STDIO_H_
#define _UART_STDIO_H

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

    void uart_stdio_Init(void);
    int uart_stdio_PutChar(char c, FILE *stream);

#endif
```

uart_stdio.c

```
#include "uart_stdio.h"
#define UART_BAUD 9600

FILE uart_output = FDEV_SETUP_STREAM(uart_stdio_PutChar, NULL, _FDEV_SETUP_WRITE);

int uart_stdio_PutChar(char c, FILE *stream) {

  if (c == '\a') {                    /* checks if c is audible return character */
      fputs("*ring*\n", stderr);
      return 0;
    }

  if (c == '\n')                      /* checks if c is new line */
    uart_stdio_PutChar('\r', stream); /* takes control to first position in the line */
  while(~UCSRA & (1 << UDRE));
  UDR = c;

  return 0;
}


void uart_stdio_Init(void) {
    #if F_CPU < 2000000UL && defined(U2X)
      UCSRA = _BV(U2X);                      /* improve baud rate error by using 2x clk */
      UBRRL = (F_CPU / (8UL * UART_BAUD)) - 1;/* used to set UART speed*/
    #else
      UBRRL = (F_CPU / (16UL * UART_BAUD)) - 1;/* used to set UART speed*/
    #endif
      UCSRB = _BV(TXEN) | _BV(RXEN);           /* Enable RX and TX */

      stdout = &uart_output;
}
```

led.h

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

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



#endif /* LED_H_ */
```

led.c

```c
#include "led.h"

void initLed() {
    DDRB |= (1 << PORTB1); /* sets value of DDRB to out */
}

void ledOn() {
    PORTB |= (1 << PORTB1); /* sets value 1 for PORTB1 */
}

void ledOff() {
    PORTB &= ~(1 << PORTB1);/* sets value 0 for PORTB1 */
}
```

button.h

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

void init();
int isPressed();

#endif /* BUTTON_H_ */
```

button.c

```c
#include "button.h"

void init() {
    DDRA &= ~(1 << PORTA0) ;   /* sets direction of DDRB to in (0).*/
}

int isPressed() {

    return PINA & (1<<PORTA0); /* returns the value of PORTA0*/
}
```

lab2.c

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


int main() {

    init();
    initLed();

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


    return 0;
}
```