Faculty of Computers, Informatics and Microelectronics
Technical University of Moldova

REPORT

Laboratory work #3

Embedded Systems

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

Verified by:
Senior Lecturer, Andrei Bragarenco

Chișinău 2017

Topic:

Analog Digital Conversion. Temperature measurement using LM20 Sensor.

Objectives:

Retrieve data from ADC.
Analog to Digital Conversion.
Connect the LM20 Sensor to the ATMega32 MCU.

Task:

Write a program that will retrieve the data from a temperature sensor. The default value to be displayed will be in °C. There will be two buttons, used to switch from °C to °K or °F. Simulate the program on a scheme, constructed with Proteus.

Overview:

Microcontrollers are capable of detecting binary signals: is the button pressed or not? These are digital signals. When a microcontroller is powered from five volts, it understands zero volts (0V) as a binary 0 and a five volts (5V) as a binary 1. The world however is not so simple and likes to use shades of gray. What if the signal is 2.72V? Is that a zero or a one? We often need to measure signals that vary; these are called analog signals. A 5V analog sensor may output 0.01V or 4.99V or anything in-between. Luckily, nearly all microcontrollers have a device built into them that allows us to convert these voltages into values that we can use in a program to make a decision.

What is ADC?
An Analog to Digital Converter (ADC) is a very useful feature that converts an analog voltage on a pin to a digital number. By converting from the analog world to the digital world, we can begin to use electronics to interface to the analog world around us.

Not every pin on a microcontroller has the ability to do analog to digital conversions. On the Arduino board, these pins have an 'A' in front of their label (A0 through A5) to indicate these pins can read analog voltages.

ADCs can vary greatly between microcontroller. The ADC on the Arduino is a 10-bit ADC meaning it has the ability to detect 1,024 (210) discrete analog levels. Some microcontrollers have 8-bit ADCs (28 = 256 discrete levels) and some have 16-bit ADCs (216 = 65,535 discrete levels).

The way an ADC works is fairly complex. There are a few different ways to achieve this feat (see Wikipedia for a list), but one of the most common technique uses the analog voltage to charge up an internal capacitor and then measure the time it takes to discharge across an internal resistor. The microcontroller monitors the number of clock cycles that pass before the capacitor is discharged. This number of cycles is the number that is returned once the ADC is complete.

ADC Value to Voltage

The ADC reports a ratiometric value. This means that the ADC assumes 5V is 1023 and anything less than 5V will be a ratio between 5V and 1023.

$$\frac{Resolution\ of\ the\ ADC}{System\ Voltage} = \frac{ADC\ Reading}{Analog\ Voltage\ Measured}$$

Analog to digital conversions are dependent on the system voltage. Because we predominantly use the 10-bit ADC of the Arduino on a 5V system, we can simplify this equation slightly:

$$\frac{1023}{5} = \frac{ADC\ Reading}{Analog\ Voltage\ Measured}$$

Program memory in the form of Ferroelectric RAM, NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications consisting of various discrete chips.

Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys and other embedded systems. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes. Mixed signal microcontrollers are common, integrating analog components needed to control non-digital electronic systems.

Celsius to Fahrenheit Conversion:

The temperature T in degrees Fahrenheit (°F) is equal to the:

$$T_{(°F)} = T_{(°C)} \times 9/5 + 32$$

Celsius to Kelvin Conversion:

The temperature T in Kelvin (K) is equal to the:

$$T_{(K)} = T_{(°C)} + 273.15$$

Tools and techniques:

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.

lcd.h and lcd.c

These files are used to output information about temperature on the LCD screen.

button.h

Header file where are declared button related function like:

```
void initButtonOne();
void initButtonTwo();
int isButtonOnePressed();
int isButtonTwoPressed();
```

button.c

The file where functions defined in button header file are implemented.
First two are used to initialize twobuttons, meaning sets direction of DDRC to in (0).
The third and forth are used to return the value of PORTC0 and PORTC1, respectively.

lm20.h and lm20.c

The lm20 related files are used to convert obtained data to Celsius, Kelvin and Fahrenheit temperature data.

adc.h and adc.c

The adc related files are used obtain raw data about temperature.

lab3.c

1. In main file initially are included button.h, lm20.hand lcd.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 **`initButtonOne(), initButtonTwo(), initLM(), LCDInit(LS_BLINK|LS_ULINE)`** and **`uart_stdio_Init(),`** used to initialize 1st button, 2nd buton, lm20, LCD and uart_stdio, respectively.

3. Later, function LCDClear() is called user to clear LCD screen.

4. Program enters the infinite loop. In the infinite loop are checked conditions like:
   a. if(isButtonOnePressed()), if true the temperature is shown in Fahrenheit system of measurement, otherwise in Celsius.
   b. if(isButtonTwoPressed()), if the condition from a. is true is checked this condition and if this condition is also true the temperature is shown in Kelvin system of measurement.

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 1.

To be able to run the application we have to specify for ATmega32 the path to the hex file (see fig. 2).

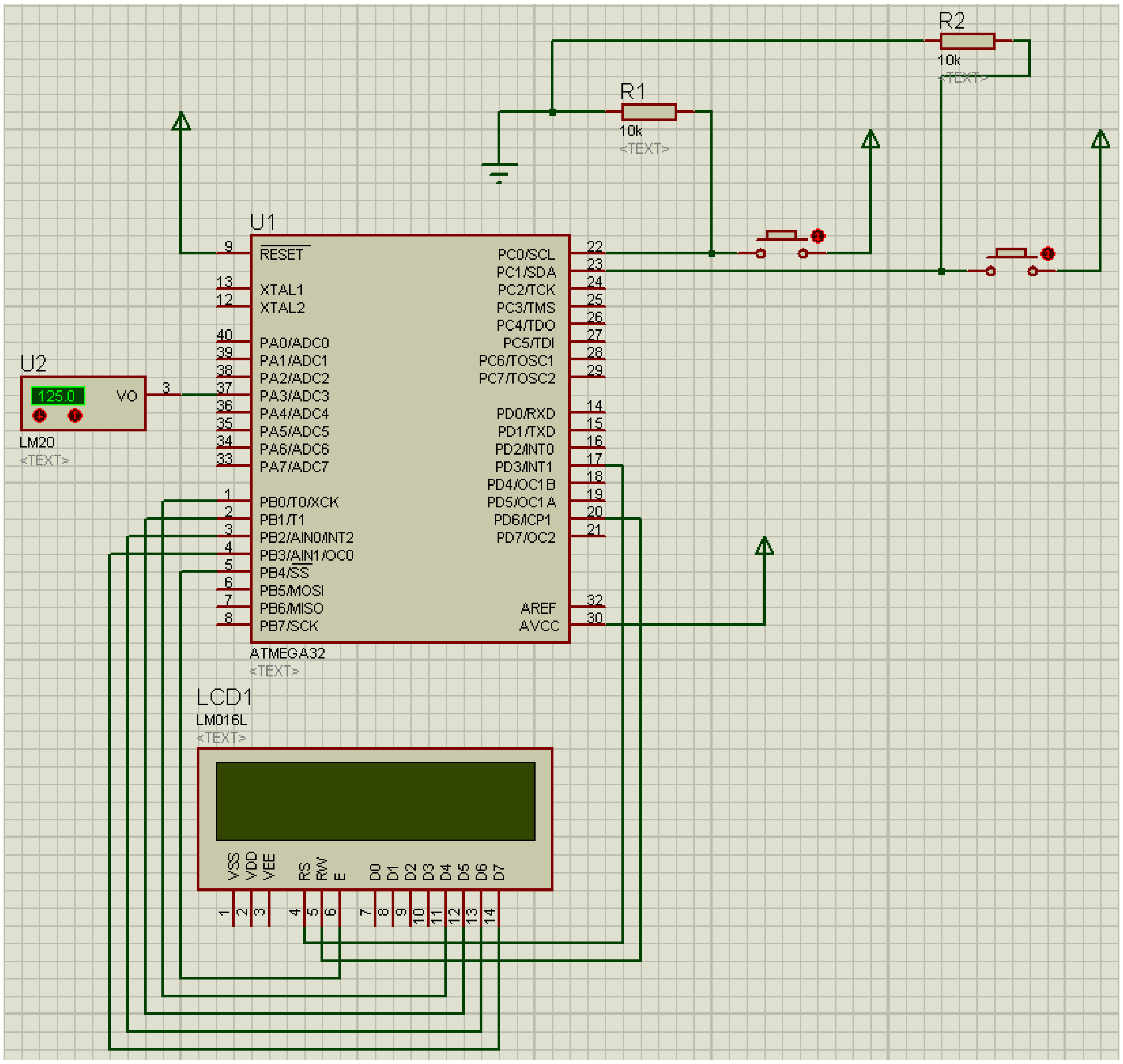
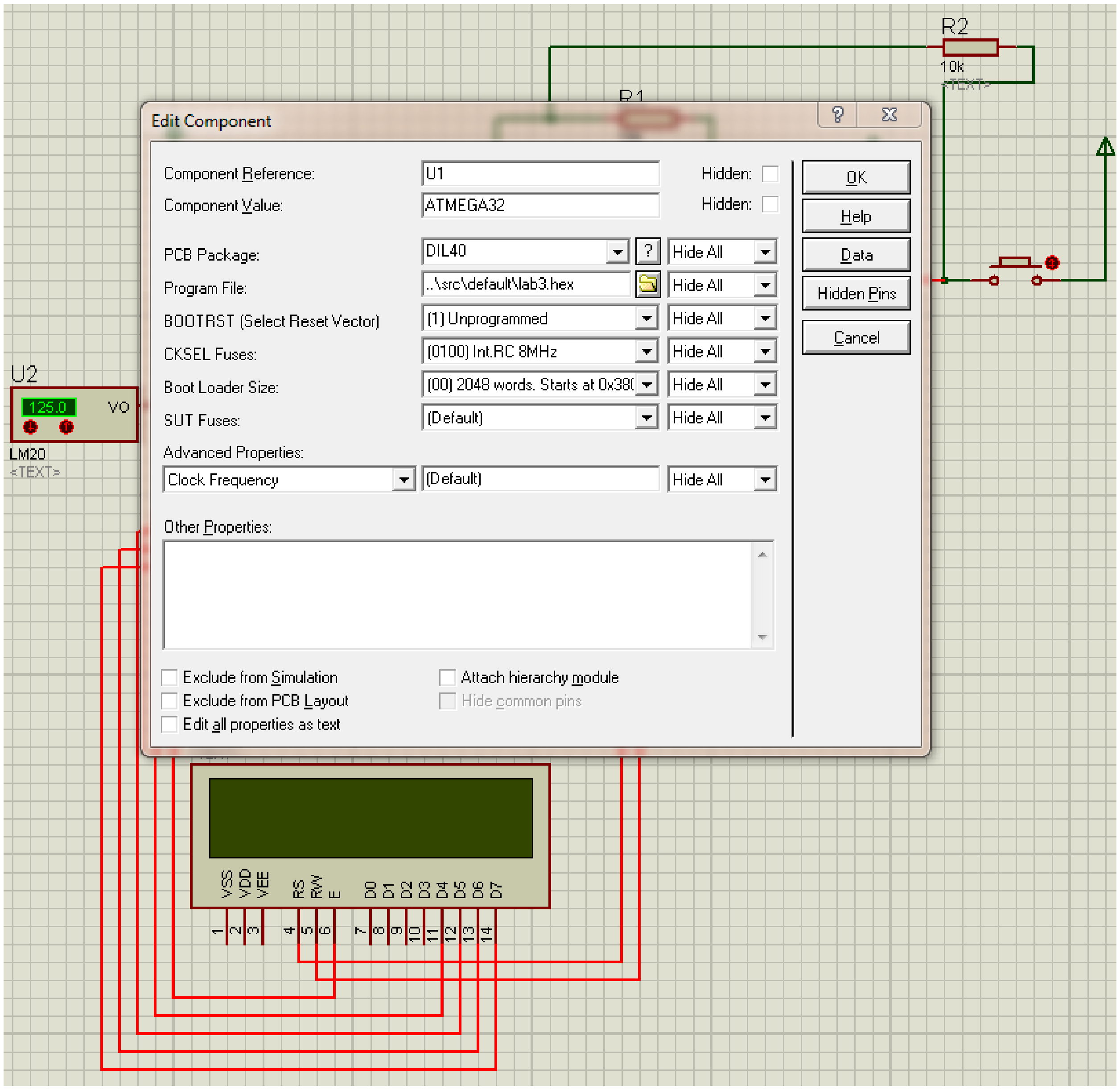


Figure 1. Scheme of the application in Proteus ISIS.


Edit Component

Component Reference: U1 Hidden: ☐

Component Value: ATMEGA32 Hidden: ☐

PCB Package: DIL40 Hide All

Program File: ..\src\default\lab3.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
☐ Exclude from PCB Layout
☐ Edit all properties as text
☐ Attach hierarchy module
☐ Hide common pins

Figure 2. Path to the hex file.

When none of the buttons are pressed the temperature is shown in Celsius (see fig. 3), if 1st button is pressed in Fahrenheit (see fig. 4) and if both buttons are pressed in Kelvin (see fig. 5).

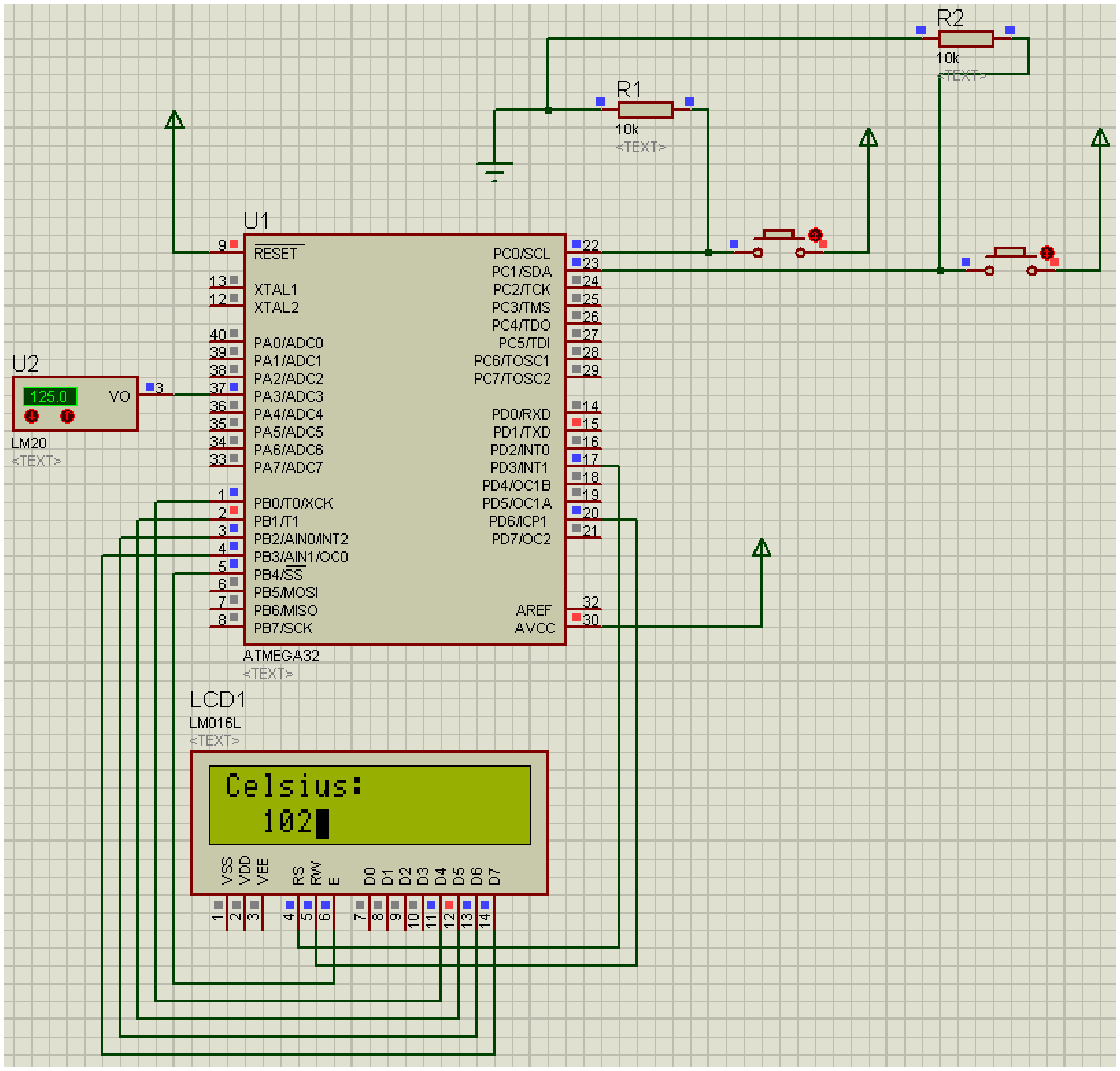


Figure 3. No buttons pressed.

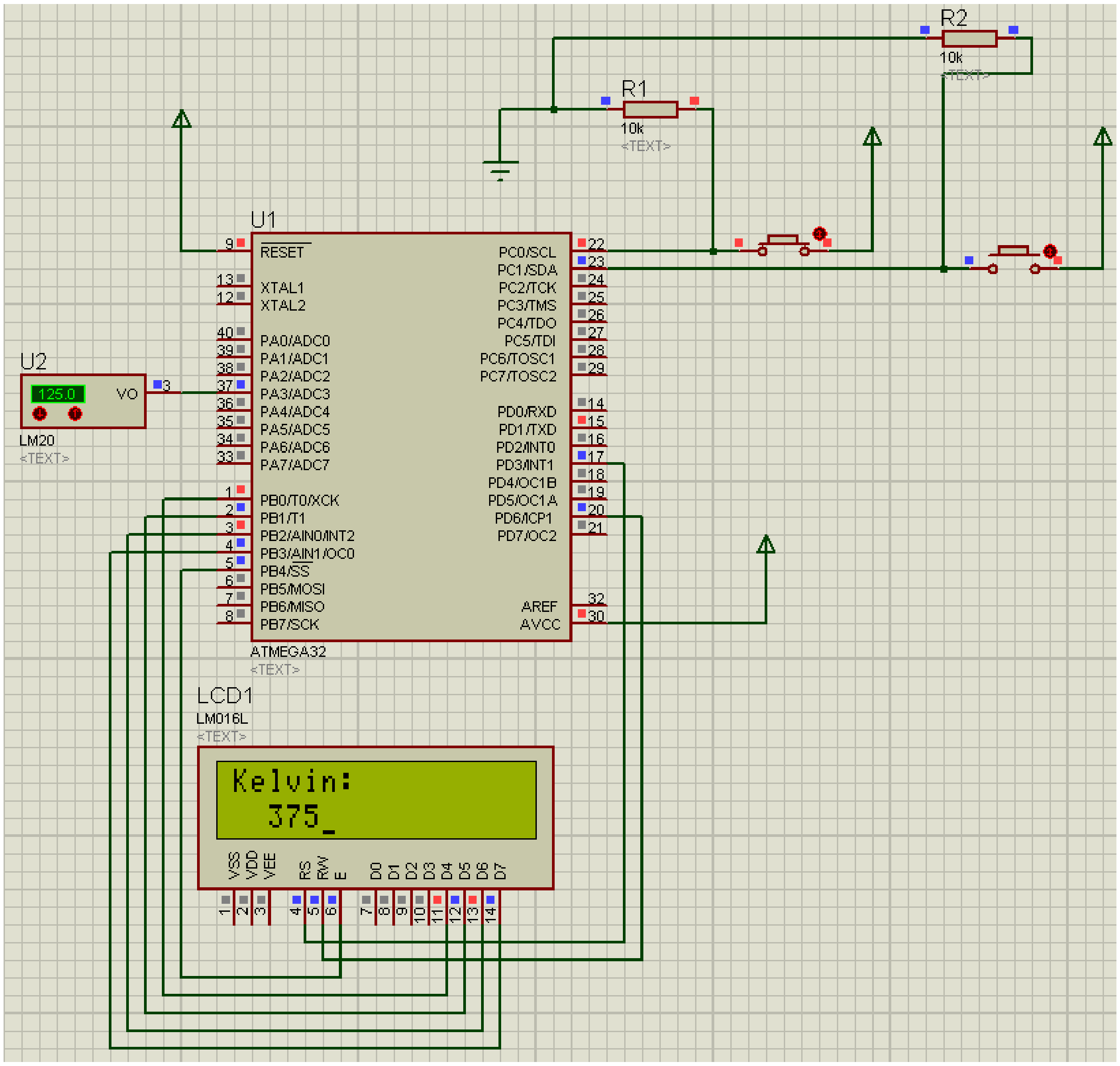


Figure 4. First button pressed.

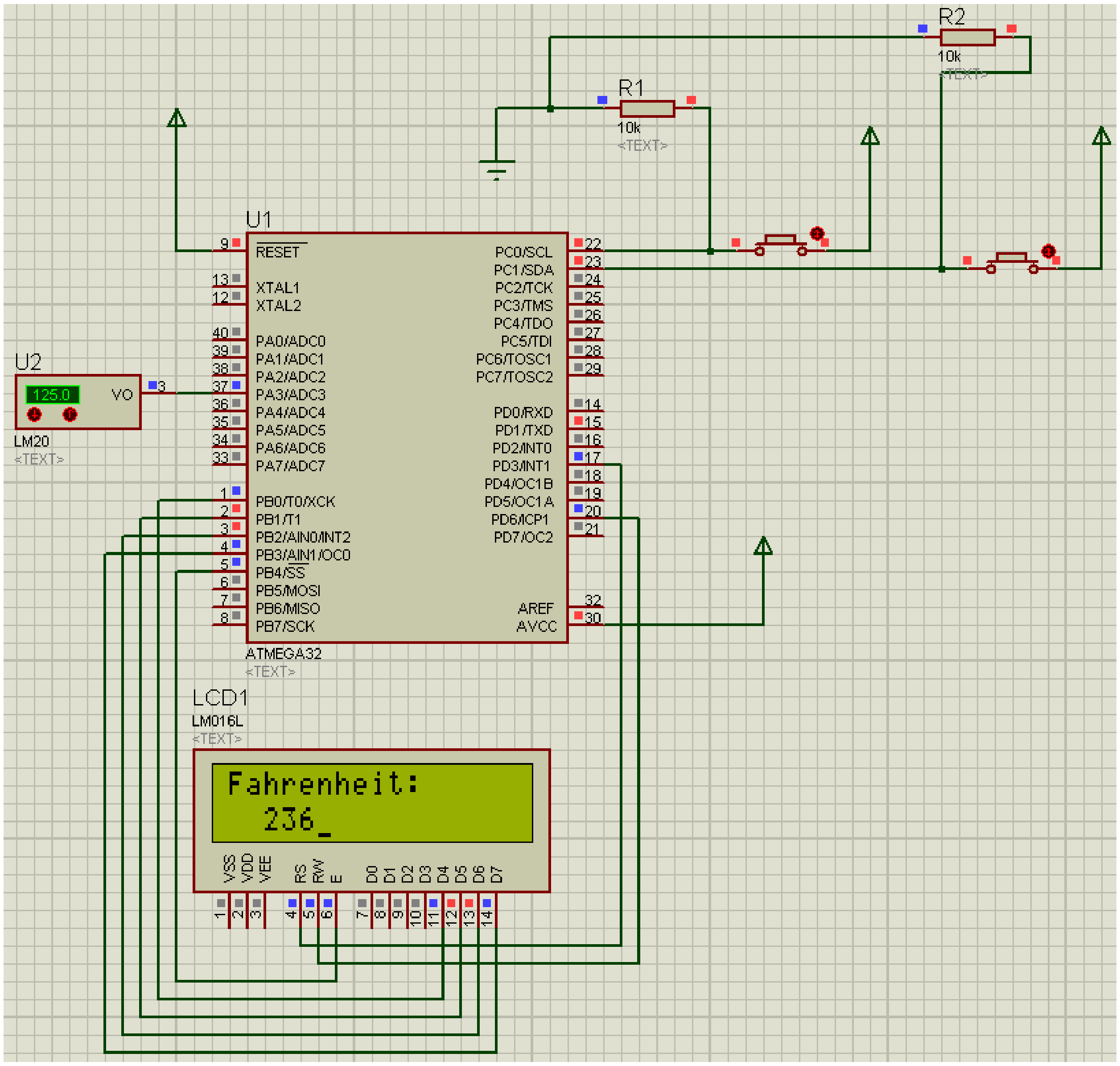


Figure 5. Both buttons pressed.

Conclusions:

The laboratory work #3 represented my first experience with retrieving data from ADC, analog to digital conversion and connecting the LM20 sensor to the ATmega32 MCU in AVR microcontroller programming. I learned the basics of working with ADC and LM20 sensor in AVR.

Appendix:

uart_stdio.h

```c
#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

```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;
}
```

lm20.h

```c
#ifndef LM20_H_
#define LM20_H_

#include "adc.h"

void initLM();
int getTemp();
int convertCelsiusToKelvin(int temp);



#endif /* LM20_H_ */
```

lm20.c

```c
#include "lm20.h"

int temp = 0;


void initLM() {
    initADC();

}

int getTemp() {
    temp = (382 - getData()) / 3;
    return temp;
}

int convertCelsiusToKelvin(int temp) {
    return temp + 273;
}

int convertCelsiusToFahrenheit(int temp) {
    return temp * 2 + 32;
}
```

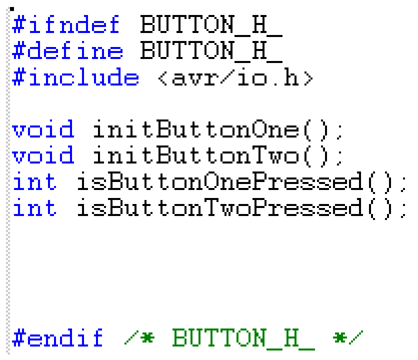
adc.h

```c
#ifndef ADC_H_
#define ADC_H_

void initADC();
int getData();
void toVoltage(int t);



#endif /* ADC_H_ */
```

adc.c

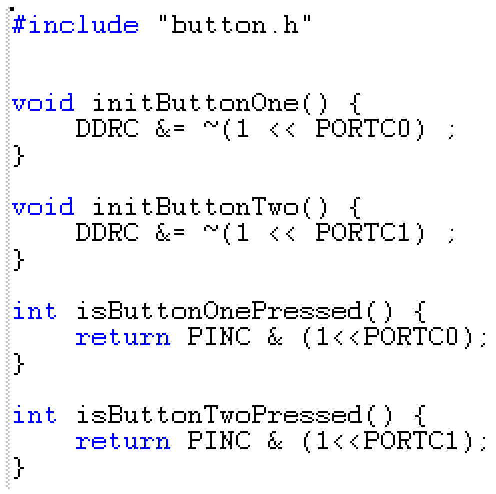
```c
#include "adc.h"
#include <avr/io.h>
int data;

void initADC() {

    ADMUX = (1 << REFS0);
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
}

int getData() {

    int adcData = 0;
    int port = 3;
    while(ADCSRA & 1 << ADSC);
    port &= 0x07;
    ADMUX = (ADMUX & ~(0x07)) | port;
    ADCSRA |= (1<<ADSC);
    while (ADCSRA & (1<<ADSC));
    adcData = ADC;
    return adcData;

}
```

button.h



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

void initButtonOne();
void initButtonTwo();
int isButtonOnePressed();
int isButtonTwoPressed();




#endif /* BUTTON_H_ */
```

button.c



```c
#include "button.h"


void initButtonOne() {
    DDRC &= ~(1 << PORTC0) ;
}

void initButtonTwo() {
    DDRC &= ~(1 << PORTC1) ;
}

int isButtonOnePressed() {
    return PINC & (1<<PORTC0);
}

int isButtonTwoPressed() {
    return PINC & (1<<PORTC1);
}
```

lab3.c

```c
#include <avr/io.h>
#include "button.h"
#include "lm20.h"
#include "lcd.h"
#include <avr/delay.h>


int main(void) {

    initButtonOne();
    initButtonTwo();
    initLM();
    uart_stdio_Init();

    //Initialize LCD module
   LCDInit(LS_BLINK|LS_ULINE);

   //Clear the screen
   LCDClear();


    while(1) {
        _delay_ms(1000);

        if(isButtonOnePressed()) {
            if(isButtonTwoPressed()) {
                LCDClear();
                LCDWriteString("Fahrenheit:");
                LCDWriteIntXY(1, 1, convertCelsiusToFahrenheit(getTemp()),3);
                printf("Fahrenheit: %d\n", convertCelsiusToFahrenheit(getTemp()));
            }else {
                LCDClear();
                LCDWriteString("Kelvin:");
                LCDWriteIntXY(1, 1, convertCelsiusToKelvin(getTemp()),3);
                printf("Kelvin: %d\n", convertCelsiusToKelvin(getTemp()));
            }

        } else {
            LCDClear();
            LCDWriteString("Celsius:");
            LCDWriteIntXY(1, 1, getTemp(),3);
            printf("Celsius : %d\n", getTemp());
        }
    }
}
```