Ministerul Educaţiei şi Tineretului al Republicii Moldova

Universitatea Tehnică a Moldovei

Departament „Informatica aplicată”

RAPORT

Lucrarea de laborator nr.3

LA DISCIPLINA”Programarea aplicațiilor incorporate și independente de platformă ”

A efectuat:

St. gr. FAF 141 Bega Valeria

A verificat:

Lect . supp Bragarenco Andrei

Chişinău 2017

Topic: Converting Analog to Digital signal. Connecting temperature sensor to MCU and display temperature to display.

Objectives:

- ADC of the AVR
- Analog to digital conversion
- Connecting Temperature Sensor to MCU

Task:

Write a driver for ADC and LM20 Temperature sensor. ADC will transform Analog to Digital data. LM20 driver will use data from ADC to transform to temperature regarding to this sensor parameters. Also use push button to switch between metrics Celsius, Fahrenheit and Kelvin

Overview of the work:

1. Analog to Digital Conversion

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.

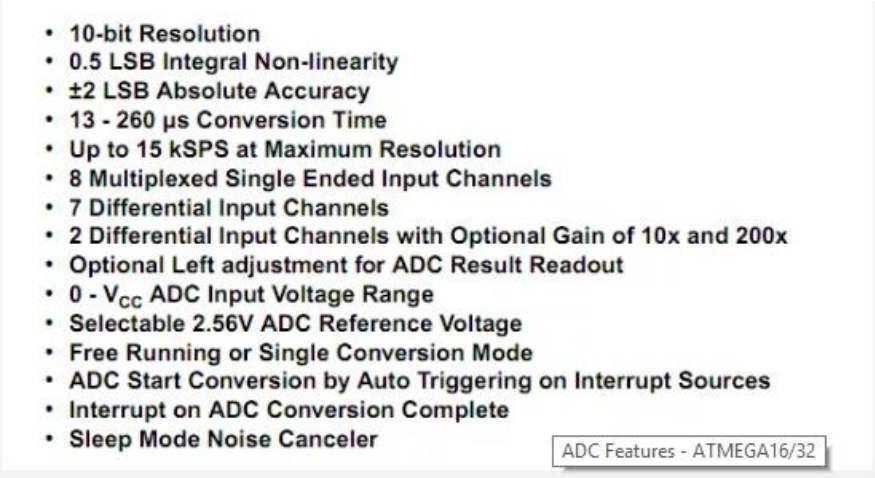
Not every pin on a microcontroller has the ability to do analog to digital conversions. On the Arduino board, these pins have an 'A' in front of their label (A0 through A5) to indicate these pins can read analog voltages.

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 = 65,536 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.

2. The ADC of AVR

The AVR features inbuilt ADC in almost all its MCU. In ATMEGA16/32, PORTA contains the ADC pins. Some other features of the ADC are as follows:



ADC Prescaler

The ADC of the AVR converts analog signal into digital signal at some regular interval. This interval is determined by the clock frequency. In general, the ADC operates within a frequency range of 50kHz to 200kHz. But the CPU clock frequency is much higher (in the order of MHz). So to achieve it, frequency division must take place. The prescaler acts as this division factor. It produces desired frequency from the external higher frequency. There are some predefined division factors – 2, 4, 8, 16, 32, 64, and 128. For example, a prescaler of 64 implies F_ADC = F_CPU/64. For F_CPU = 16MHz, F_ADC = 16M/64 = 250kHz.

Now, the major question is… which frequency to select? Out of the 50kHz-200kHz range of frequencies, which one do we need? Well, the answer lies in your need. There is a trade-off between frequency and accuracy. Greater the frequency, lesser the accuracy and vice-versa. So, if your application is not sophisticated and doesn't require much accuracy, you could go for higher frequencies.

ADC Initialization

The following code segment initializes the ADC.

```
void adc_init()
{
   // AREF = AVcc
   ADMUX = (1<<REFS0);    // ADC Enable and prescaler of 128
```

```
  // 16000000/128 = 125000
  ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}
```

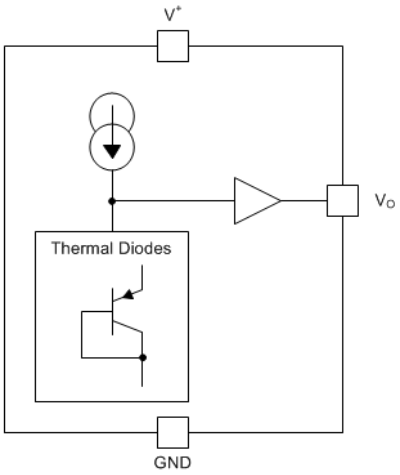
Reading ADC Value

The following code segment reads the value of the ADC. Always refer to the register description above for every line of code.

```
uint16_t adc_read(uint8_t ch)
{
  // select the corresponding channel 0~7
  // ANDing with ’7′ will always keep the value
  // of ‘ch’ between 0 and 7
  ch &= 0b00000111;  // AND operation with 7
  ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing

  // start single convertion
  // write ’1′ to ADSC
  ADCSRA |= (1<<ADSC);

  // wait for conversion to complete
  // ADSC becomes ’0′ again
  // till then, run loop continuously
  while(ADCSRA & (1<<ADSC));

  return (ADC);
}
```

3. LM20 Temperature Sensor

The LM20 is a precision analog output CMOS integrated-circuit temperature sensor that operates over −55°C to 130°C. The power supply operating range is 2.4 V to 5.5 V. The transfer function of LM20 is predominately linear, yet has a slight predictable parabolic curvature. The accuracy of the LM20 when specified to a parabolic transfer function is ±1.5°C at an ambient temperature of 30°C.



Applicable in :

- Cellular phones
- PCs
- Power Supply Modules
- Printers

Solution:

adc.h / adc.c

Contains declaration and implementation of ADC Driver.
It has 3 methods:

- void initADC(); // Initializes driver
- int getData(); // Get data from initialized ADC. It's a 10 bit value, which can give us value from range 0..1023
- void toVoltage(int t); //converts temperature to voltage

lm20.h /lm20.c

Contains declaration and implementation of LM20 Driver. It depends on ADC driver. It has 3 methods:

- void initLM(); // Initializes driver
- int getTemp(); // Uses ADC driver to get digital value
- int convertCelsiusToKelvin(int temp); //converts digital value to temperature

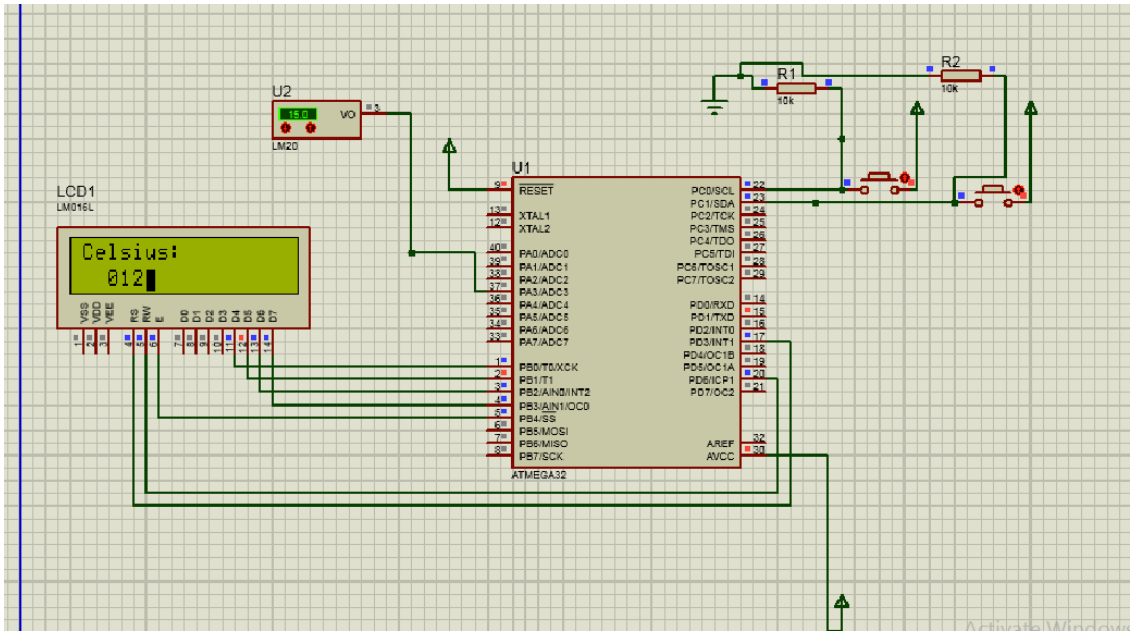
Main Program

Main Program is responsible for :

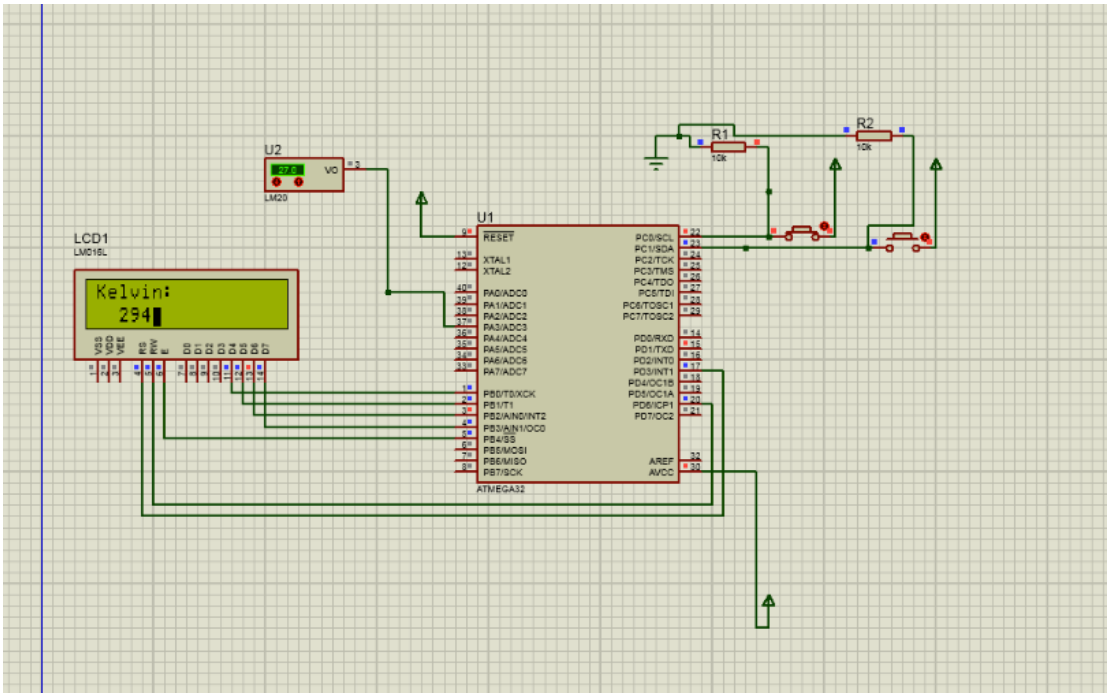
- Initialization of 2 buttons , which switch from Celsius to Fahrenheit and Kelvin
- Initialization of UART , for displaying temperature
- Initialization of LM20 Sensor
- program receives temperature from Sensor and converts it to needed conversion, which is decided by button push

We proceed with the Schemes in Proteus:

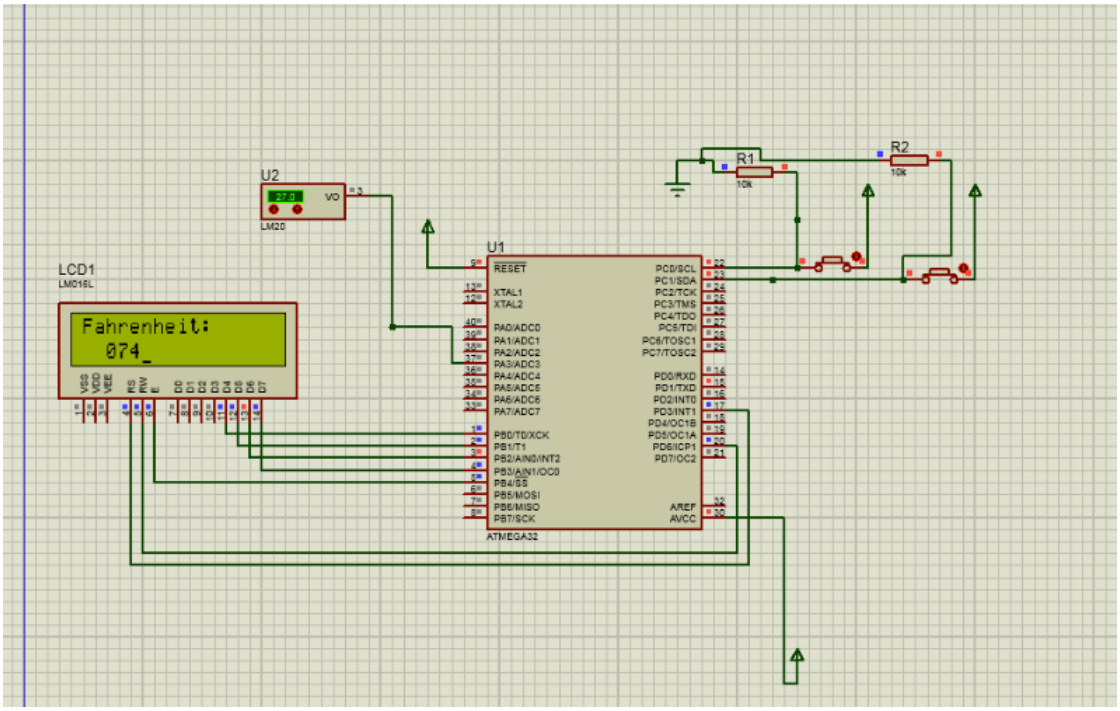
Celsius Representation:



Kelvin representation:



Fahrenheit Representation:



Conclusion:

During this laboratory work I learned how to get analog values from the environment and convert them into digital values by ADC. It was fun to work with temperature and learn more about how the ADC works in AVR. Analog and Digital data is important to understand, because its very often applicable in real life, so it was useful to learn all this new information.

Appendix

main.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());
                }
                }
}
```

adc.h

```
#ifndef ADC_H_
#define ADC_H_
void initADC(); int
getData();
void toVoltage(int t);

#endif /* ADC_H_ */
```

adc.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;

        }
```

lm20.h

```
#ifndef LM20_H_

#define LM20_H_


#include "adc.h"


void initLM(); int
getTemp();

int convertCelsiusToKelvin(int temp);


#endif /* LM20_H_ */
```

lm20.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;

}
```