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 141eng Schidu Luca

A verificat:

Lect . supp A. Bragrarenco

Chişinău 2016

<u>Topic:</u> ADC of AVR Microcontroller

<u>Task:</u> Write an application that will read an analog value from ADC and convert this value to the temperature. The result will be displayed to the standard output interface LCD or to virtual terminal.

<u>Theoretical notes</u>

Most of the physical quantities around us are continuous. By continuous we mean that the quantity can take any value between two extreme. For example the atmospheric temperature can take any value (within certain range). If an electrical quantity is made to vary directly in proportion to this value (temperature etc) then what we have is Analogue signal. Now we have we have brought a physical quantity into electrical domain. The electrical quantity in most case is voltage. To bring this quantity into digital domain we have to convert this into digital form. For this a ADC or analog to digital converter is needed. Most modern MCU including AVRs has an ADC on chip. An ADC converts an input voltage into a number. An ADC has a resolution. A 10 Bit ADC has a range of 0-1023. (2^10=1024) The ADC also has a Reference voltage(ARef). When input voltage is GND the output is 0 and when input voltage is equal to ARef the output is 1023. So the input range is 0-ARef and digital output is 0-1023.

Inbuilt ADC of AVR

The ADC is multiplexed with PORTA that means the ADC channels are shared with PORTA. The ADC can be operated in single conversion and free running more. In single conversion mode the ADC does the conversion and then stop. While in free it is continuously converting. It does a conversion and then start next conversion immediately after that.

ADC Prescaler.

The ADC needs a clock pulse to do its conversion. This clock generated by system clock by dividing it to get smaller frequency. The ADC requires a frequency between 50KHz to 200KHz. At higher frequency the conversion is fast while a lower frequency the conversion is more accurate. As the system frequency can be set to any value by the user (using internal or externals oscillators)(In xBoard™ a 16MHz crystal is used). So the Prescaler is provided to produces acceptable frequency for ADC from any system clock frequency. System clock can be divided by 2,4,16,32,64,128 by setting the Prescaler.

ADC Channels

The ADC in ATmega32 has 8 channels that means you can take samples from eight different terminal. You can connect up to 8 different sensors and get their values separately.

ADC Registers.

As you know the registers related to any particular peripheral module(like ADC, Timer, USART etc.) provides the communication link between the CPU and that peripheral. You configure the ADC according to need using these registers and you also get the conversion result also using appropriate registers. The ADC has only four registers.

- ADC Multiplexer Selection Register – ADMUX : For selecting the reference voltage and the input channel.
- ADC Control and Status Register A – ADCSRA : As the name says it has the status of ADC and is also use for controlling it.
- The ADC Data Register – ADCL and ADCH : The final result of conversion is here.

LM20

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. The temperature error increases linearly and reaches a maximum of ±2.5°C at the temperature range extremes. The temperature range is affected by the power supply voltage. At a power supply voltage of 2.7 V to 5.5 V, the temperature range extremes are 130°C and −55°C. Decreasing the power supply voltage to 2.4 V changes the negative extreme to −30°C, while the positive extreme remains at 130°C.

The LM20 quiescent current is less than 10 μA. Therefore, self-heating is less than 0.02°C in still air. Shutdown capability for the LM20 is intrinsic because its inherent low power consumption allows it to be powered directly from the output of many logic gates or does not necessitate shutdown.

Working process

We have to configure the ADC by setting up ADMUX and ACSRA registers.

The ADMUX has following bits.

Bit Nr.	7	6	5	4	3	2	1	0
Bit Name	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
Initial Val	0	0	0	0	0	0	0	0

REFS1 REFS0 selects the reference voltage.

We will go for 2nd option, Our reference voltage will e VCC(5v).

REFS1	REFS0	Voltage reference Selection
0	0	Aref internal Vref Turned off
0	1	AVCC
0	1	Reserved
1	1	Internal 2.56 Voltage reference

```
ADMUX = (1 << REFS0);
```

The ADCSRA Register:

Bit Nr.	7	6	5	4	3	2	1	0
Bit Name	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Initial Val	0	0	0	0	0	0	0	0

- ADEN – Set this to 1 to enable ADC
- ADSC – We need to set this to one whenever we need ADC to do a conversion.
- ADIF – This is the interrupt it this is set to 1 by the hardware when conversion is complete. So we can wait till conversion is complete by polling this bit like:

```
while(ADCSRA & 1 << ADSC);
```

The loop does nothing while ADIF is set to 0, it exits as soon as ADIF is set to 1, when conversion is complete.

- ADPS2-ADPS0 – These selects the Prescaler for ADC. Frequency must be between 50KHz to 200 KHz.

ADPS2	ADPS1	ADPS0	Division factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

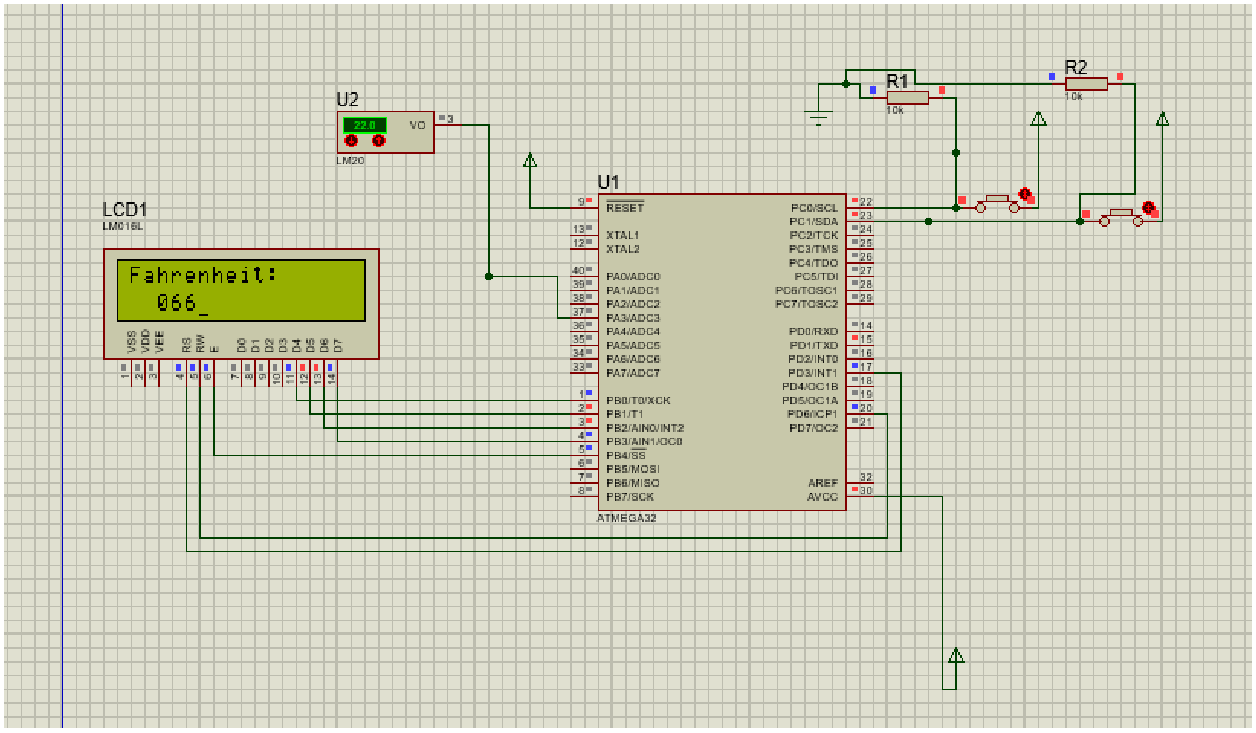
We need to select division factor so as to get an acceptable frequency from out 16Mhz clock. We select division factor as 128. So ADC clock frequency 16000000/120 = 125000 = 125 KHz. So we set ADCSR as

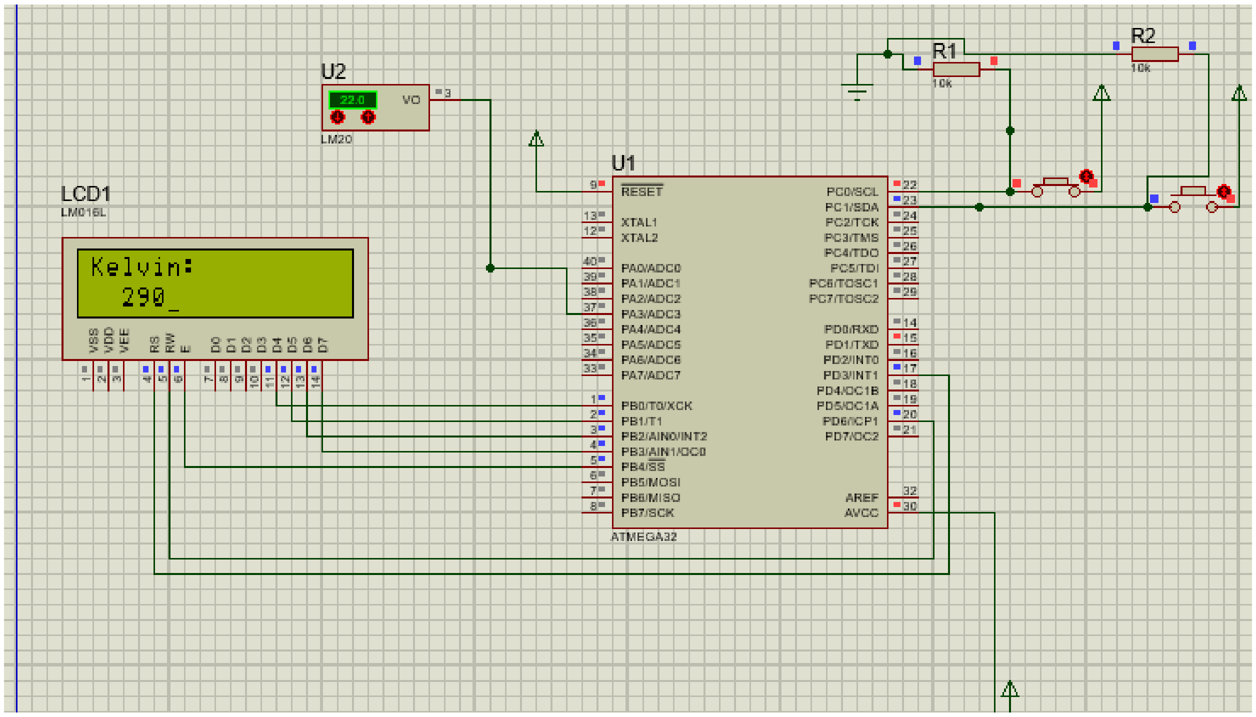
```
ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
```

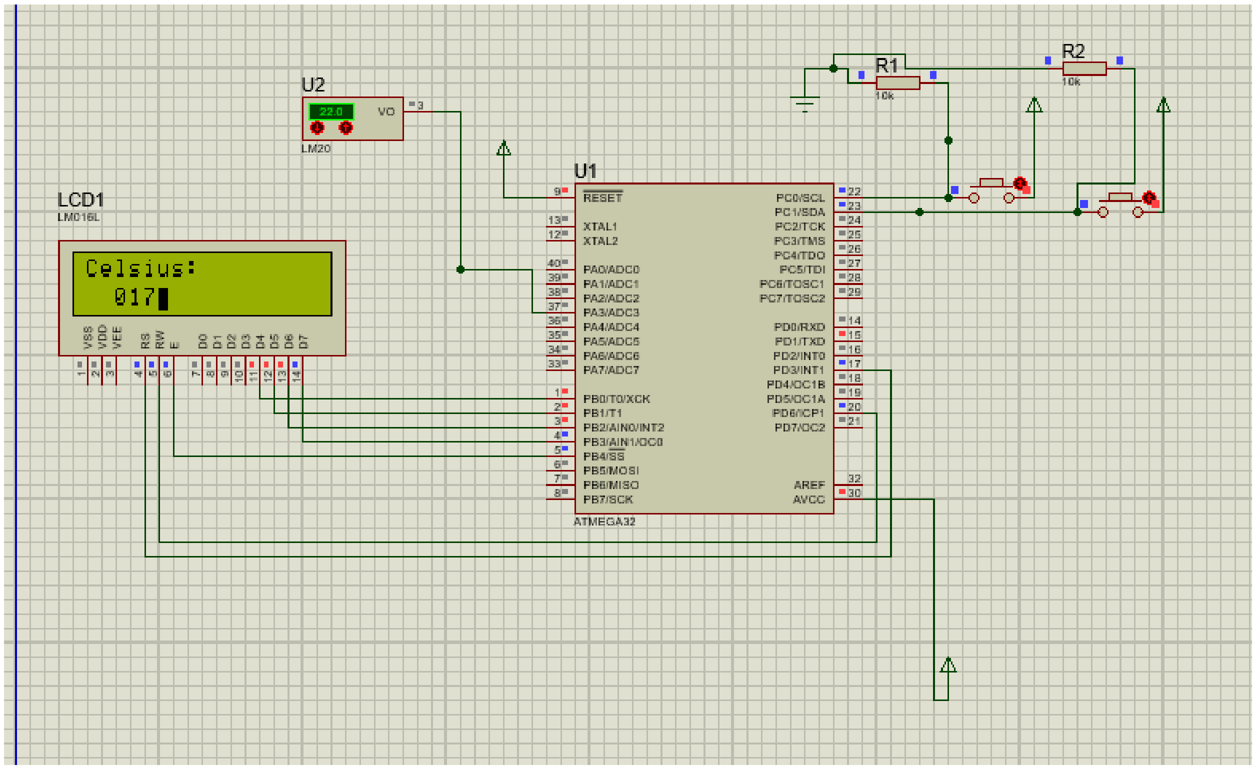
Now we can get data from ADC. First we wait until ADC is busy, after we normalize the channel, leave last 3 bits LSB. After we apply the channel to the ADMUX with protection of configuration bits. Start conversion. Wait until conversion is complete then return adcData.

```
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;

}
```

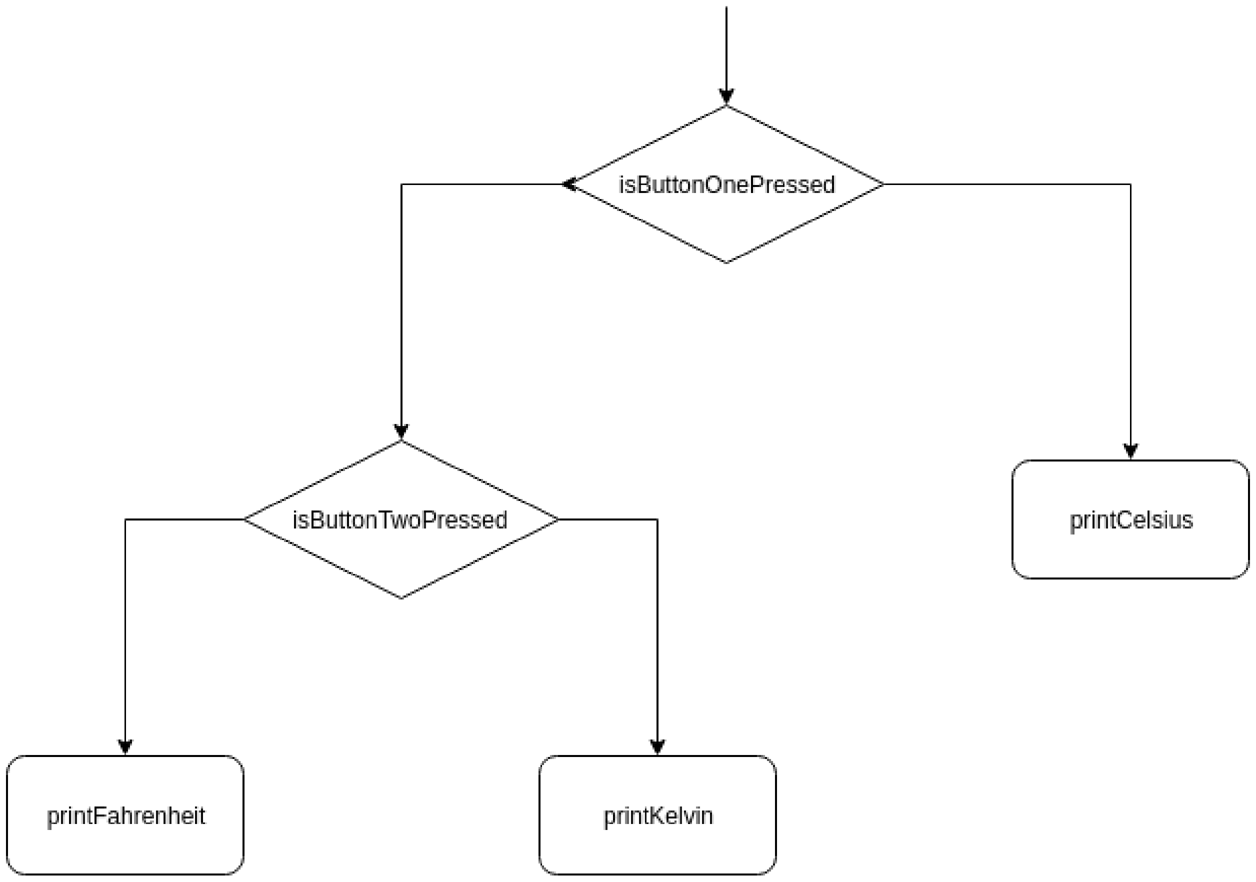
Results


U2
22.0
VO
LM20
LCD1
LM016L
Fahrenheit:
066_
U1
ATMEGA32
R1
10k
R2
10k
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7


U2
22.0
VO
LM20
U1
LCD1
LM016L
Kelvin:
290_
R1
10k
R2
10k
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
ATMEGA32


U2
22.0
VO
LM20
U1
LCD1
LM016L
Celsius:
017
R1
10k
R2
10k
ATMEGA32

Flowchart diagram



Conclusion

During this laboratory work I learned how to get analog values from the environment and convert them into digital values by ADC. I also learned how to connect a lcd to the Atmega32 microcontroller and display some messages on it.

Source code :

```
/*
 * adc.c
 *
 * Created: 10/4/2016 11:40:30 PM
 *  Author: schid
 */
#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;

}
```

```
/*
 * adc.h
 *
 * Created: 10/4/2016 11:40:40 PM
 *  Author: schid
 */

#ifndef ADC_H_
#define ADC_H_

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

#endif /* ADC_H_ */
```

```
/*
 * LAB3.c
 *
 * Created: 10/4/2016 11:14:51 PM
 *  Author: schid
 */

#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());

                }

    }

}
```