Ministry of Education of Moldova
Technical University of Moldova
Faculty "Computers, Informatics and Microelectronics"

REPORT

Laboratory work No.3
On Embedded systems

Topic: ” ADC of AVR Microcontroller”

Performed by st. gr. FAF-141 : Botnari Nicolae

Verified by : Andrei Bragarenco

Chişinău 2016

Topic: ADC of AVR Microcontroller

Task: 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.

Theory

Using the ADC (Analog to Digital Converter) of PIC Microcontroller

Many electrical signals around us are Analog in nature. That means a quantity varies directly with some other quantity. The first quantity is mostly voltage while that second quantity can be anything like temperature, pressure, light, force or acceleration. For example in LM35 temperature sensor the output voltage varies according to the temperature, so if we could measure voltage, we can measure temperature.

But most of our computer (or Microcontrollers) are digital in nature. They can only differentiate between HIGH or LOW level on input pins. For example if input is more than 2.5v it will be read as 1 and if it is below 2.5 then it will be read as 0 (in case of 5v systems). So we cannot measure voltage directly from MCUs. To solve this problem most modern MCUs have an ADC unit. ADC stands for analog to digital converter. It will convert a voltage to a number so that it can be processed by a digital systems like MCU.

This enables us to easily interface all sort of analog devices with MCUs. Some really helpful example of analog devices are

Light Sensors.
Temperature Sensors.
Accelerometers.
Touch Screens.
Microphone for Audio Recording.
And possibly many more.

Specifications of ADCs

Most important specification of ADCs is the resolution. This specifies how accurately the ADC measures the analog input signals. Common ADCs are 8 bit, 10 bit and 12 bit. For example if the reference voltage(explained latter) of ADC is 0 to 5v then a 8 bit ADC will break it in 256 divisions so it can measure it accurately up to 5/256 v= 19mV approx. While the 10 bit ADC will break the range in 5/1024 = 4.8mV approx. So you can see that the 8 bit ADC can’t tell the difference between 1mV and 18mV. The ADC in PIC18 are 10 bit.

Other specification include (but not limited to) the sampling rate, that means how fast the ADC can take readings. Microchip claims that pic18f4520′s ADC can go as high as 100K samples per second.

Solution

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 of
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.

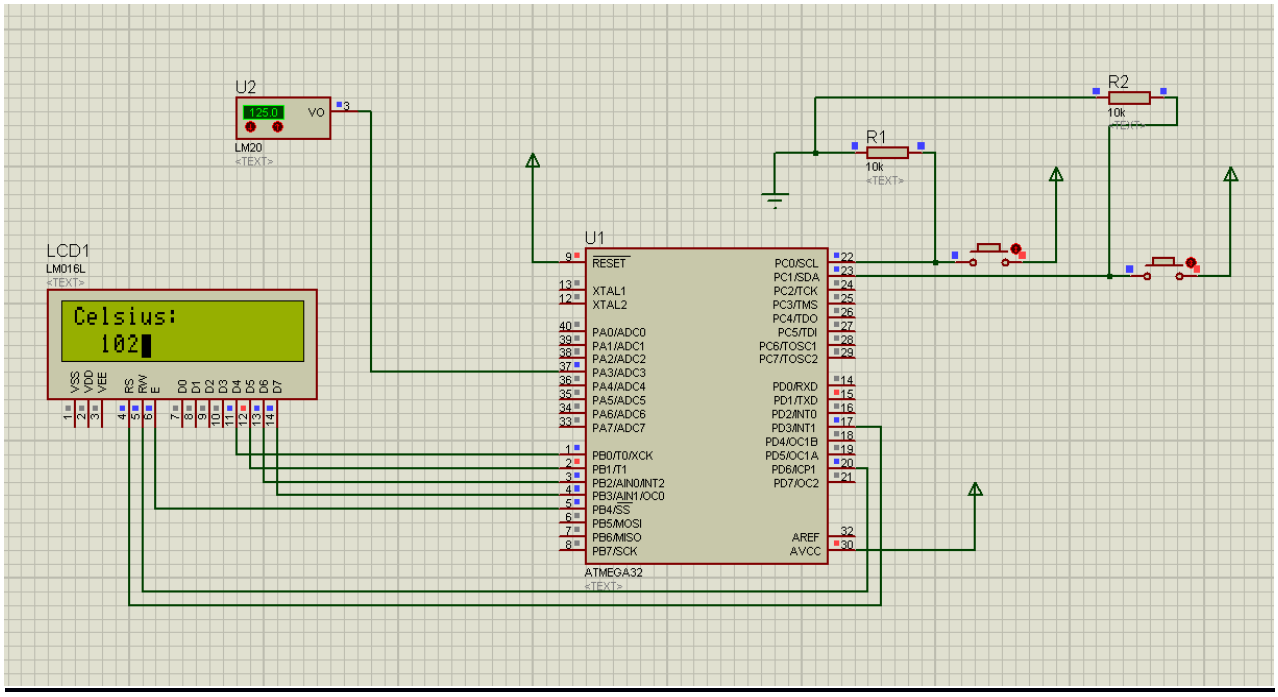
ADPS2	ADPS1	ADPS0	Division factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

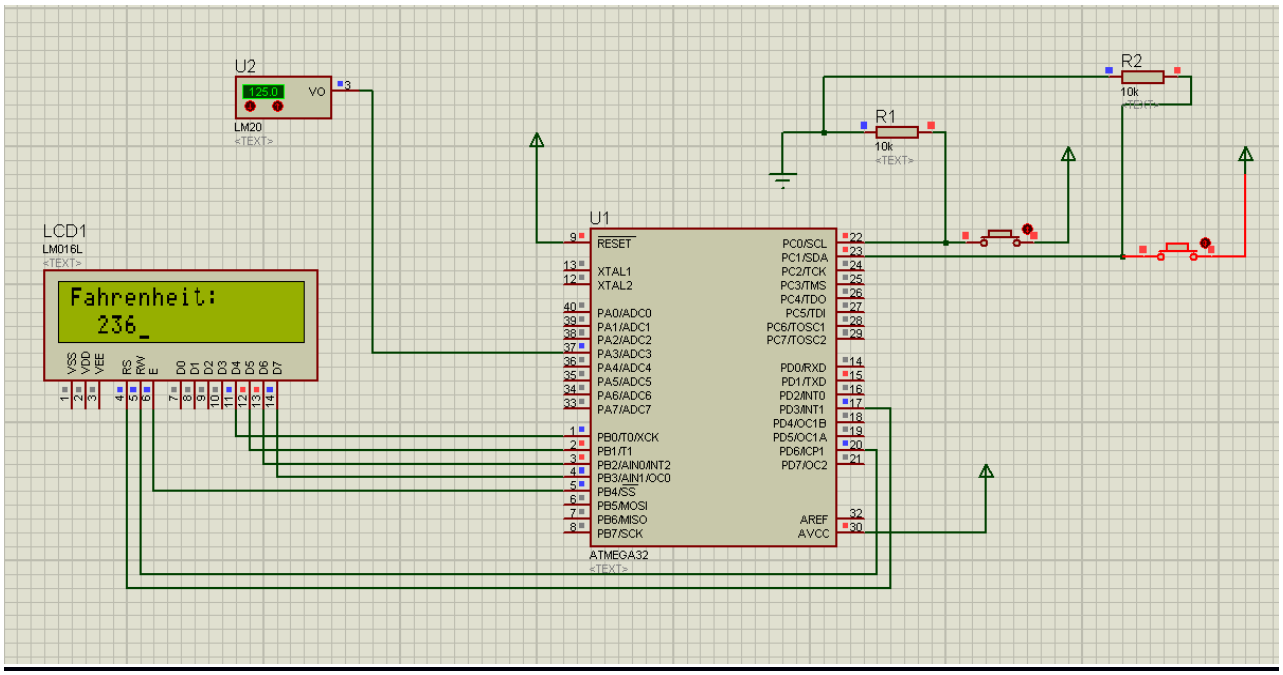
We need to select division factor so as to get an acceptable frequency from out 16Mhz clock. We select division factor as 128. So ADC clock frequency 16000000/120 = 125000 = 125 KHz. So we set ADCSR as

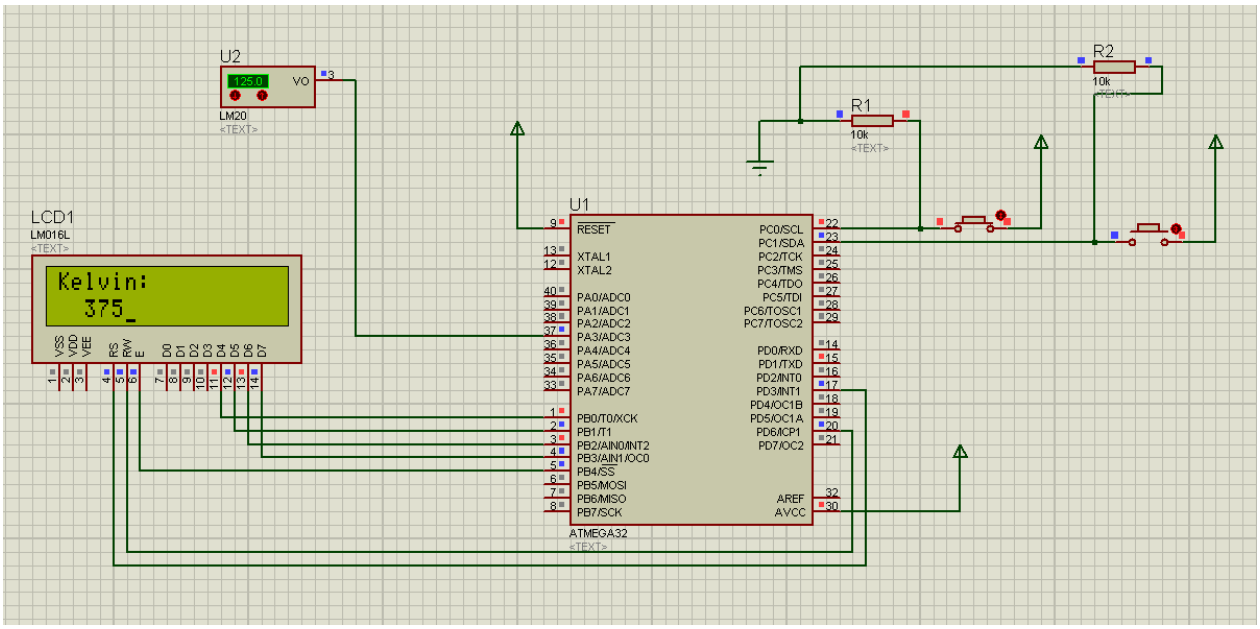
```
ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
```

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.

Result


U2
125.0
VO
LM20
<TEXT>
R2
10k
R1
10k
<TEXT>
LCD1
LM016L
<TEXT>
Celsius:
102
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
U1
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
<TEXT>


U2
125.0
VO
LM20
<TEXT>
R2
10k
R1
10k
<TEXT>
LCD1
LM016L
<TEXT>
Fahrenheit:
236
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
U1
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
<TEXT>



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.

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

```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;
}
```