Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

RAPORT

la Programarea aplicațiilor incorporate și independente de platformă

Lucrare de laborator Nr.3

Tema:”Transformarea signalului Analog in Digital.Conectarea unui sensor de temperatura.”

A efectuat st. gr. FAF-141: Mihai Trofim

A verificat lect. univ.: Andrei Bragarenco

Chişinău 2017

Topic

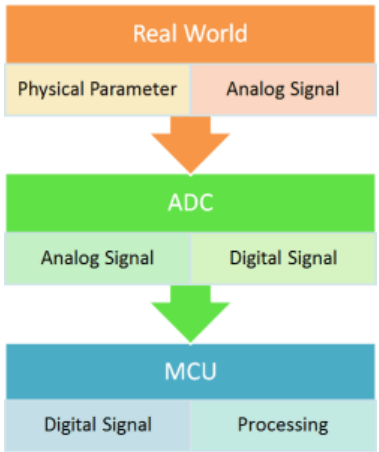
Converting Analog to Digital signal. Connecting temperature sensor to MCU and display temperature to display.

Task

Write 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.

Domain

Analog to Digital Conversion



Most real world data is analog. Whether it be temperature, pressure, voltage, etc, their variation is always analog in nature. For example, the temperature inside a boiler is around 800°C. During its light-up, the temperature never approaches directly to 800°C. If the ambient temperature is 400°C, it will start increasing gradually to 450°C, 500°C and thus reaches 800°C over a period of time. This is an analog data.\

Now, we must process the data that we have received. But analog signal processing is quite inefficient in terms of accuracy, speed and desired output. Hence, we convert them to digital form using an Analog to Digital Converter (ADC).

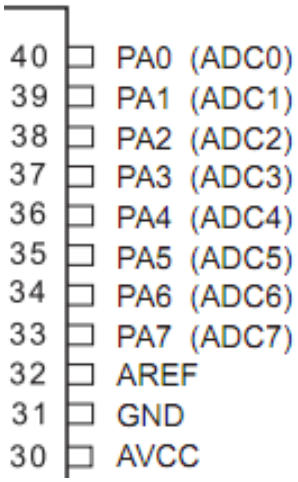
Signal Acquisition Process

In general, the signal (or data) acquisition process has 3 steps.

In the Real World, a sensor senses any physical parameter and converts into an equivalent analog electrical signal.

For efficient and ease of signal processing, this analog signal is converted into a digital signal using an **Analog to Digital Converter (ADC).**

This digital signal is then fed to the **Microcontroller (MCU)** and is processed accordingly.



Interfacing Sensors

In general, sensors provide with analog output, but a MCU is a digital one. Hence we need to use ADC. For simple circuits, comparator op-amps can be used. But even this won't be required if we use a MCU. We can straightaway use the inbuilt ADC of the MCU. In ATMEGA16/32, PORTA contains the ADC pins.

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 Registers

ADMUX – ADC Multiplexer Selection Register

The ADMUX register is as follows.

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The bits that are highlighted are of interest to us. In any case, we will discuss all the bits one by one.

ADCSRA – ADC Control and Status Register A

The ADCSRA register is as follows.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The bits that are highlighted are of interest to us. In any case, we will discuss all the bits one by one.

ADCL and ADCH – ADC Data Registers

The result of the ADC conversion is stored here. Since the ADC has a resolution of 10 bits, it requires 10 bits to store the result. Hence one single 8 bit register is not sufficient. We need two registers – ADCL and ADCH (ADC Low byte and ADC High byte) as follows. The two can be called together as ADC.

Used Resources

LM20 –Temperature Sensor

Characteristics

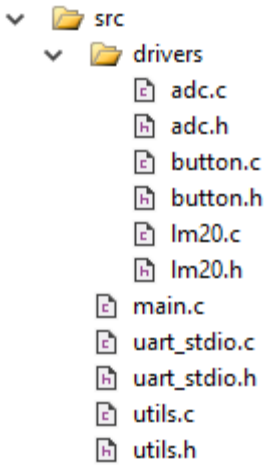
- Rated for −55°C to 130°C Range
- Suitable for Remote Applications
- Accuracy at 30°C ±1.5 to ±4°C (Maximum)
- Accuracy at 130°C and −55°C ±2.5 to ±5°C
- Power Supply Voltage Range 2.4 V to 5.5 V
- Current Drain 10 µA (Maximum)
- Output Impedance 160 Ω (Maximum)

Applications

- Cellular Phones
- Computers
- Power Supply Modules
- Battery Management
- FAX Machines
- Printers
- HVAC
- Disk Drives
- Appliances

Solution

Project Structure looks in this way



adc.h / adc.c

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

1. void ADC_Init();
2. int ADC_GetData();
3. float ADC_GetVoltage();

1) Initializes driver

2) Get data from initialized ADC. It's a 10 bit value, which can give us value from range 0..1023

3) Get Voltage from ADC but converts it to range 0 to 5 Volts.

lm20.h /lm20.c

Contains declaration and implementation of LM20 Driver. It depends on ADC driver. It has 2 methods:

1. void LM20_Init();
2. int LM20_GetTemperature();

1) Initializes driver. Initialization is done in ADC also.

2) Uses ADC driver to get digital value and converts to Celsius temperature.

lm20.h /lm20.c

Contains useful methods used in entire project.

float ConvertValueFromToRange(float val,float minFromRange,float maxFromRange,float minToRange,float maxToRange);

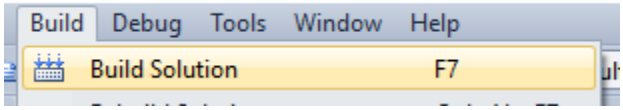
This method is used by ADC and LM20 drivers in order to transform value from [0..1023] to [0..5V] and for converting [0.3..2.4V] to Celsius [-55..130].This method uses Euqation of a line, for conversion form one line to another.

Main Program

Main Program is responsible for :

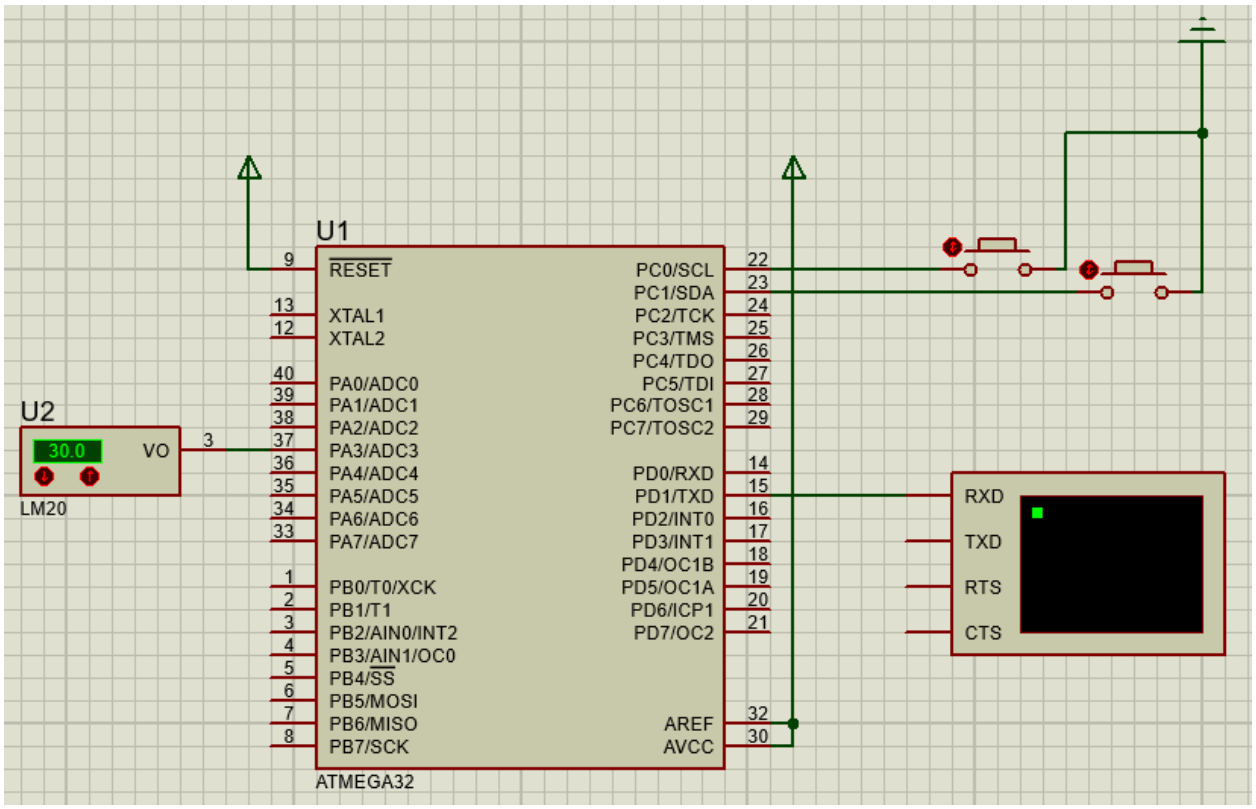
- Initialization of 2 buttons , which switch from Celsius to Fahrenheit and Kelvin
- Initialization of UART , for displaying temperature
- Initialization of LM20 Sensor
- Every 1 second program Receives temperature from Sensor and converts it to needed conversion, which is decided by button push

After code implementation, we should now **Build Hex** which will be written to MCU ROM.



Schematics

For our laboratory work we need only simple ATMega32 MCU,2 Push Buttons, LM20 sensor and peripheral UART device, which in our case is virtual terminal.

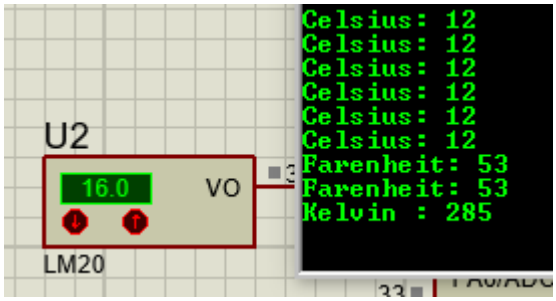


We need to make sure that our MCU is connected to Virtual Terminal. Because we use only data transmission on one direction (OUTPUT) we need to make sure that our MC **Tx** is connected to Peripheral **Rx**.

MCU is transmitter and peripheral is receiver. No vice versa connection because we don't need it in our laboratory work.

Our LM20 Sensor was connected to ADC3 and Push Buttons to port C , pin 0 and 1.

Simulation Result



We can see that our temperature is close to original. Error rate appears because we used proportional distribution for transformation to temperature. There is a formula on LM20 Datasheet for precise transformation.

Conclusion

This laboratory work gave us a basic concepts about ADC. We connected sensor to our MCU and have written Drivers for ADC and LM20 which actually prepares hardware and get's Analog data from sensor then converts it to Digital Data.

Most difficult part of this laboratory work was ADC driver. Actually initialization was one of hardest thing. Rest of application is quite easy for whose who have basic C knowledge. Also I used linear conversion for getting Temperature from device, but it could be better to use formula from LM20 datasheet. This improvement will make our temperature more precise.

Appendix

Main.c

```
#include <avr/io.h>
#include <avr/delay.h>
#include "uart_stdio.h"
#include "drivers/button.h"
#include "drivers/lm20.h"

#define DELAY_TIME_MS 1000

int CelsiusToKelvin(int celsius){
	return celsius+273;
}

int CelsiusToFarenheit(int celsius){
	return celsius*18/10 + 32;
}

void main() {
	struct ButtonDevice button1;
	struct ButtonDevice button2;
	button1.ddr = &DDRC;
	button1.pin = &PINC;
	button1.pinIndex = 0;

	button2.ddr = &DDRC;
	button2.pin = &PINC;
	button2.pinIndex = 1;

	// Initialization
	uart_stdio_Init();
	LM20_Init();
	ButtonInit(&button1);
	ButtonInit(&button2);

	int temperature;

	while(1){
		temperature = LM20_GetTemperature();
		if(ButtonPressed(&button1)){
			temperature = CelsiusToKelvin(temperature);
			printf("Kelvin : %d\n",temperature);
		} else {
			if (ButtonPressed(&button2))
			{
				temperature = CelsiusToFarenheit(temperature);
				printf("Farenheit: %d \n",temperature);
			} else {
				printf("Celsius: %d\n",temperature);
			}
		}
		_delay_ms(DELAY_TIME_MS);
	}
}
```

adc.h

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

void ADC_Init();
int ADC_GetData();
float ADC_GetVoltage();

#endif /* ADC_H_ */
```

adc.c

```
#include "adc.h"
#include "../utils.h"

void ADC_Init(){
	ADCSRA = (1 << ADEN)
			|(1 << ADSC)
			|(0 << ADATE)
			|(0 << ADIF)
			|(0 << ADIE)
			|(1 << ADPS2)
			|(1 << ADPS1)
			|(0 << ADPS0);

	ADMUX = 0x03;
}

int ADC_GetData(){
	ADCSRA |= (1 << ADSC);
	while(ADCSRA && ADIF == 0);

	return ADC;
}

float ADC_GetVoltage(){
	return ConvertValueFromToRange(ADC_GetData(),0,1023,0,5);
}
```

Lm20.h

```
#ifndef LM20_H_
```

```h
#define LM20_H_
#include "adc.h"

void LM20_Init();
int LM20_GetTemperature();

#endif /* LM20_H_ */
```

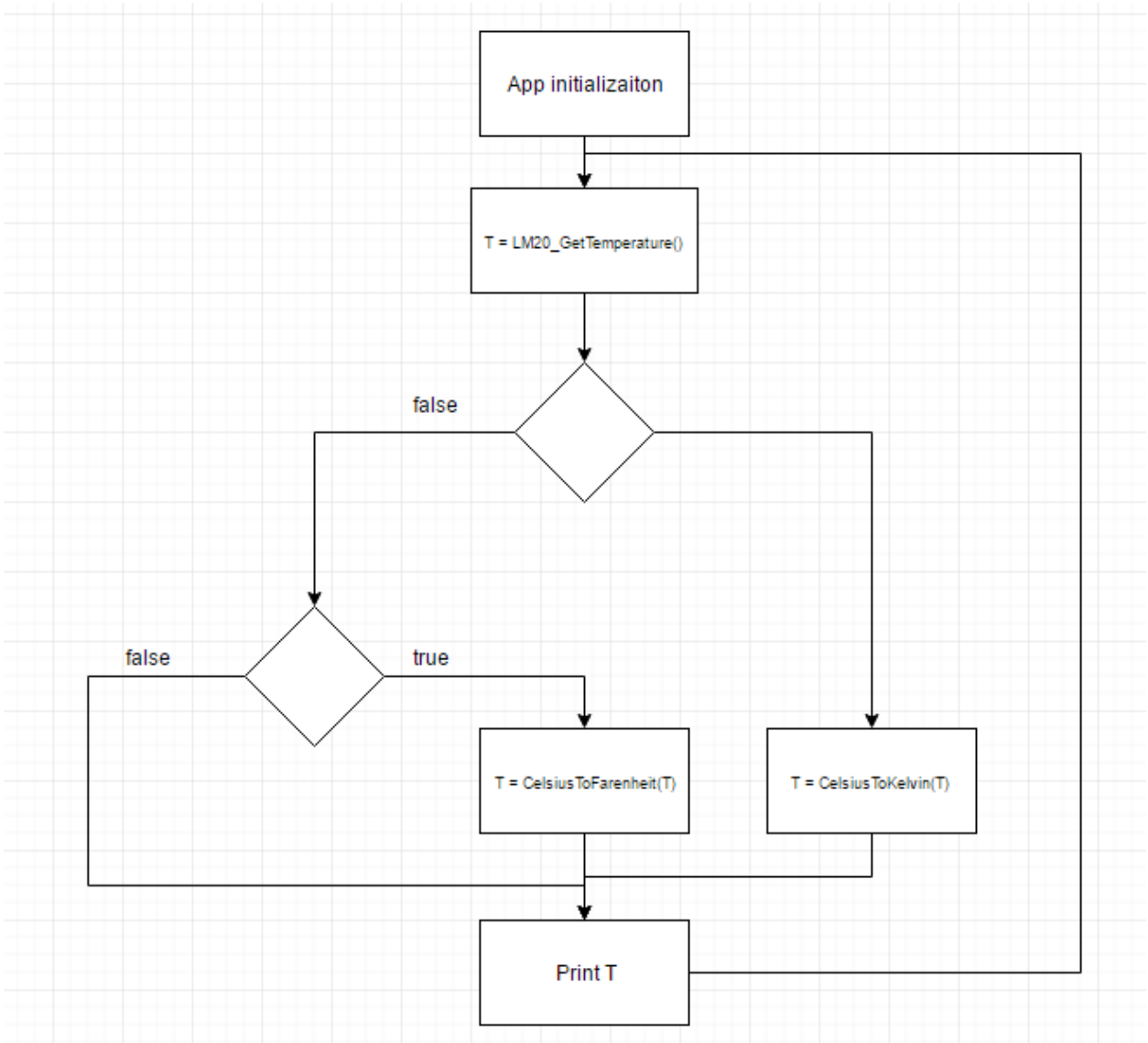
Lm20.c

```c
#include "lm20.h"
#include "../utils.h"

void LM20_Init(){
	ADC_Init();
}

int LM20_GetTemperature(){
	return ConvertValueFromToRange(ADC_GetVoltage(),0.3,2.48,130,-55);
}
```

FlowChart


App initializaiton
T = LM20_GetTemperature()
false
false
true
T = CelsiusToFarenheit(T)
T = CelsiusToKelvin(T)
Print T