Ministerul Educatiei al Republicii Moldova
Universitatea Tehnica a Moldovei
Filiera Anglofona

Report

at Embedded Systems

Laboratory Work #3

Topic: ADC - Analog Digital Conversion. Temperature measurement using LM20 Sensor.

Performed by: Diana ARTIOM

Verified by: Andrei
BRAGARENCO

Chisinau 2016

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 inbetween. 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.

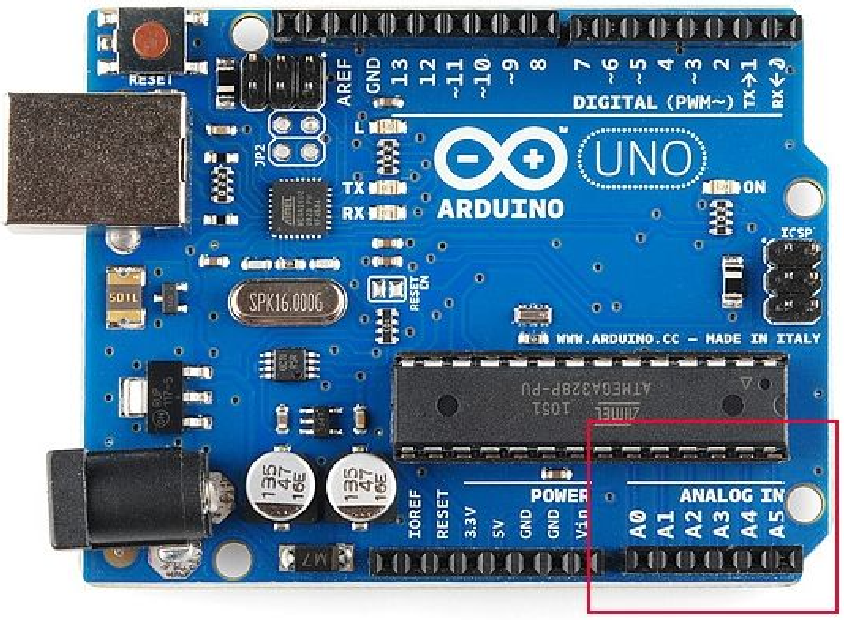

Figure 1: Physical representation of ADC pins on Arduino

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 dependant 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 temperature *T* in degrees Celsius (°C) times 9/5 plus 32:

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

Celsius to Kelvin Conversion

The temperature *T* in Kelvin (K) is equal to the temperature *T* in degrees Celsius (°C) plus 273.15:

$$T_{(K)} = T_{(°C)} + 273.15$$

Tools and Technologies used:

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

Proteus is a Virtual System Modelling and circuit simulation application. The suite combines mixed mode SPICE circuit simulation, animated components and microprocessor models to facilitate co-simulation of complete microcontroller based designs. Proteus also has the ability to simulate the interaction between software running on a microcontroller and any analog or digital electronics connected to it. It simulates Input / Output ports, interrupts, timers, USARTs and all other peripherals present on each supported processor.

Solution:

In this laboratory work I had to deal with Analog Digital Conversion. Temperature measurement using LM20 Sensor.

Before proceeding to explain which is what, I will first include here the project structure of the project:

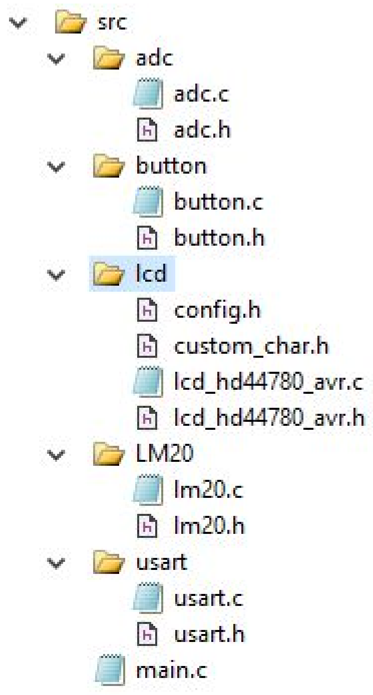
```
src
  adc
    adc.c
    adc.h
  button
    button.c
    button.h
  lcd
    config.h
    custom_char.h
    lcd_hd44780_avr.c
    lcd_hd44780_avr.h
  LM20
    lm20.c
    lm20.h
  usart
    usart.c
    usart.h
  main.c
```

Figure 3: Project structure

Implementations

Button

In order to make the program efficient and elegant, I have chosen to represent each connected device to a port of the MCU with a button struct:

```
struct Button {
        uint8_t pinNr;
        volatile uint8_t *ddr;
        volatile uint8_t *ioReg;
};
```

`pinNr` - is the index of the pin at some specific port(A, B, C or D)
`ddr` - configuration on input or output of the whole port
`ioReg`- pin or port - in dependence of the configuration (input or output)

LCD

For LCD interfacing I used a library found on internet - written by **eXtreme Electronics India.** For more info, check the link Extreme Elecrtonics.

ADC

For LCD interfacing I used a library found on internet - written by **eXtreme Electronics India.** For more info, check the link Extreme Elecrtonics.

LM20

For LCD interfacing I used a library found on internet - written by **eXtreme Electronics India.** For more info, check the link Extreme Elecrtonics.

main

Main function is the entry point of the program. It works in the following way:

1) Initializes the lcd, button and LM20 :

```
initButtons();
USARTInit();
LM20_Init();
LCDInit(LS_NONE);
```

2) Initializes the LCD:

```
LCDInit(LS_BLINK);
```

3) Enters the infinite while loop:

   With a frequency of 50 ms (`_delay_ms(50);`)

   (a) Gets the celsius value from ADC:

   ```
   value = LM20_GetCelsiusValue(value);
   ```

   (b) Converts it to display on LCD:

   ```
   itoa(value, buffer, 10);
   ```

(c) Checks whether the convert to Fahrenheit or Kelvin conversion buttons were clicked.

```
checkTConversion(buffer, value);
```

The full code can be found in Appendix.

Preparing for Proteus Simulation

After implementing the solution in terms of C code, I compiled it using the **Build Solution** option in Atmel Studio. The path for the program to run on MCU in Proteus, is the file with extension **.hex**, found in .../lab3/lab3/debug folder, with the name lab3.hex:

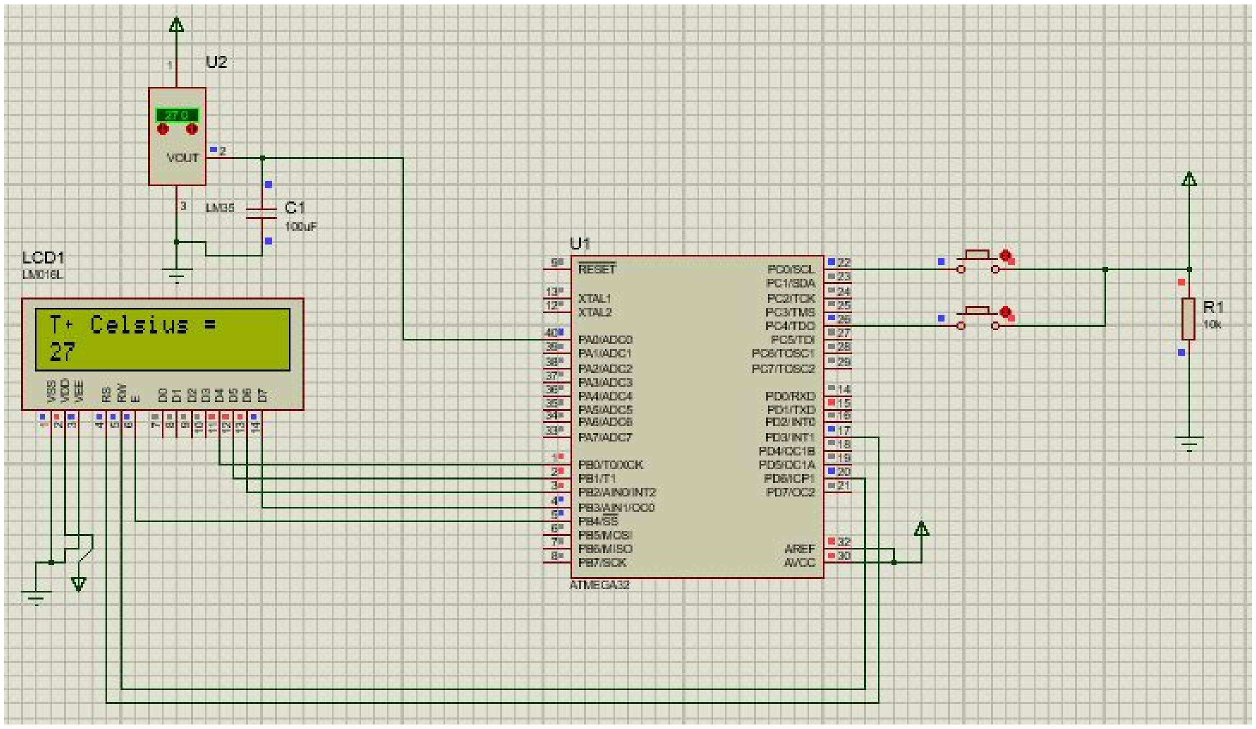

Figure 4: Construction of the scheme - Default Celsius value displayed

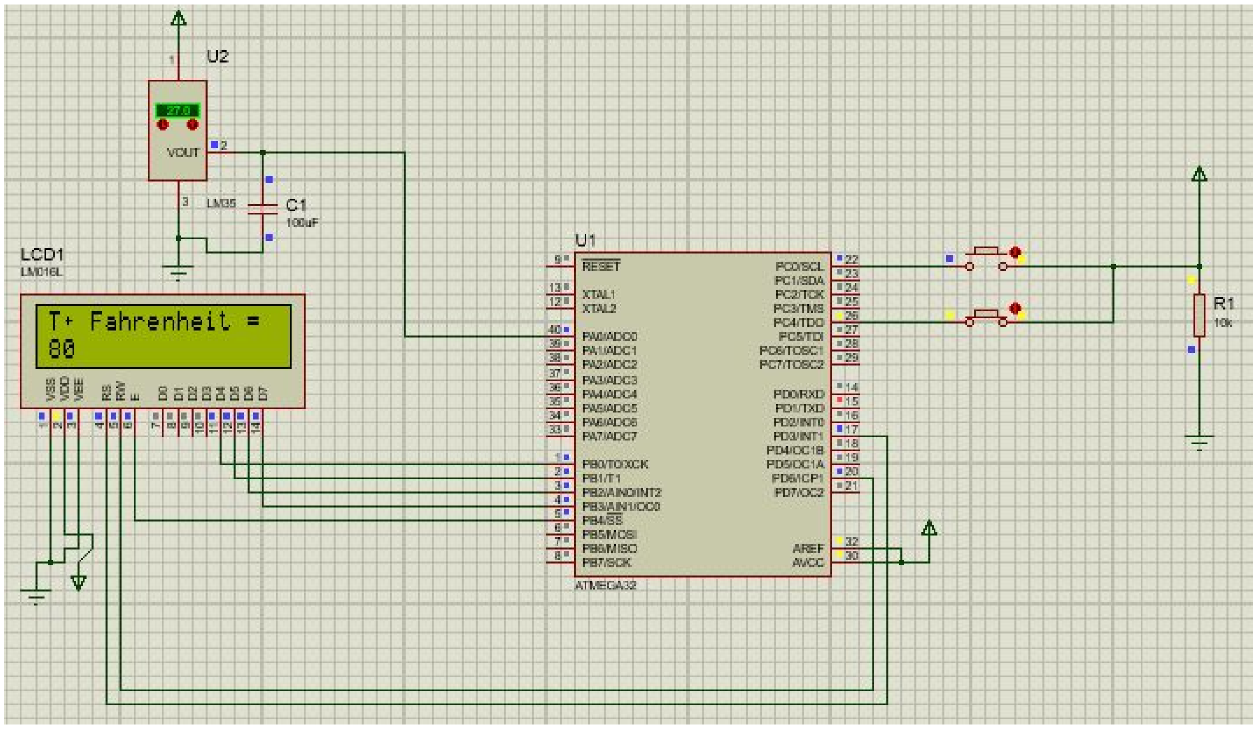

Figure 4: Construction of the scheme -Fahrenheit conversion button pressed

Conclusion:

Appendix:

main.c

```
#include <avr/delay.h>
#include "lcd/lcd_hd44780_avr.h"
#include "usart/usart.h"
#include "adc/adc.h"
#include "LM20/lm20.h"

struct Button *btnKelvin;
struct Button *btnFarenheit;

void initButtons();
void outputOnLCD(char *msg, char *buffer, int value);
void checkTConversion(char *buffer, int value);

int main(void) {

       unsigned int value;
       char buffer[3];
       initButtons();
       USARTInit();
       LM20_Init();
       LCDInit(LS_NONE);

       while(1) {
              value = LM20_GetCelsiusValue(value);
              itoa(value, buffer, 10);
              _delay_ms(50);
              checkTConversion(buffer, value);
       }
}


void checkTConversion(char *buffer, int value) {

       if (isButtonPressed(&btnKelvin)) {
              value = convertToKelvin(value);
              outputOnLCD("T%0 Kelvin = ", buffer, value);
       } else if (isButtonPressed(&btnFarenheit)) {
              value = convertToFarenheit(value);
              outputOnLCD("T%0 Fahrenheit = ", buffer, value);
       } else {
              outputOnLCD("T%0 Celsius = ", buffer, value);
       }
}


void initButtons() {
       initButton(&btnKelvin, PINC0, &DDRC, &PINC); // init button
       setButtonDDR(&btnKelvin);

       initButton(&btnFarenheit, PINC4, &DDRC, &PINC); // init button
       setButtonDDR(&btnFarenheit);

}


void outputOnLCD(char *msg, char *buffer, int value) {
       itoa(value, buffer, 10);
       LCDGotoXY(0,0);
       LCDWriteString(msg);
       LCDGotoXY(0,1);
       LCDWriteString("    ");
       LCDGotoXY(0,1);
       LCDWriteString( buffer);
}
```

lm20.h

```
#ifndef LM20_H

#define LM20_H


void LM20_Init(void);
```

```c
unsigned int LM20_GetCelsiusValue(int value);

int convertToKelvin(int value);

int convertToFarenheit(int value);


#endif
```

lm20.c

```c
#include "lm20.h"

void LM20_Init(void) {
       ADC_init();
}


unsigned int LM20_GetCelsiusValue(int value) {
value = ADC_read(0x00);
value = value * 500/1024;
return value;
}

int convertToFarenheit(int value) {
       return((int) value *9/5 + 32);
}

int convertToKelvin(int value) {
       return value + 273;
}
```

adc.h

```c
#ifndef ADC_H

#define ADC_H


#define ADC_VREF_TYPE 0x40


#include <avr/delay.h>

#include <avr/io.h>


void ADC_init(void);

unsigned int ADC_read(unsigned char adc_input);

#endif
```

adc.c

```
#include "adc.h"

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


unsigned int ADC_read(unsigned char adc_input) {
	ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
	// Delay needed for the stabilization of the ADC input voltage
	_delay_us(10);
	// Start the AD conversion
	ADCSRA|=0b01000000;
	// Wait for the AD conversion to complete
	while ((ADCSRA & 0x10)==1);
	ADCSRA|=0b00000;
	return ADCW;
}
```

button.h

```
#ifndef BUTTON_H_

#define BUTTON_H_



#include <stdint.h>

#include <avr/io.h>


struct Button {

uint8_t pinNr;

volatile uint8_t *ddr;

volatile uint8_t *ioReg;

};


void initButton(struct Button *obj,

uint8_t _pinNr,

volatile uint8_t *_ddr,

volatile uint8_t *_ioReg );


char isButtonPressed(struct Button *obj);

void setButtonDDR(struct Button *obj);
```

```
#endif
```

button.c

```
#include "button.h"


char  isButtonPressed(struct Button *obj) {
       if((*(obj->ioReg))&(1<<obj->pinNr))
              return 1;
       return 0;
}


void setButtonDDR(struct Button *obj) {
       *(obj->ddr) |= 1<<obj->pinNr;
}


void initButton(struct Button *obj,
       uint8_t _pinNr,
       volatile uint8_t *_ddr,
       volatile uint8_t *_ioReg ) {
              obj->pinNr = _pinNr;
              obj->ddr = _ddr;
              obj->ioReg = _ioReg;
       }
```

usart.h

```
#ifndef USART_H
#define USART_H

#include <avr/io.h>

void USARTInit();
char USARTReadChar();
void USARTWriteChar(char data);

#endif
```

usart.c

```
#include "usart.h"

void USARTInit() {
       UCSRA=0x00;
```

```
        UCSRB=0x18;
        UCSRC=0x86;
        UBRRH=0x00;
        UBRRL=0x33;
}


char USARTReadChar() {

        while(!(UCSRA & (1<<RXC))) { }
        return UDR;
}


void USARTWriteChar(char data) {


        while(!(UCSRA & (1<<UDRE))) { }
        UDR=data;
}
```