Education Ministry of Moldova

Technical University

Computer Science and Microelectronics

Report

Embedded Systems

laboratory work #3

Author:
Bîrcu Maxim

Supervisor:
Bragarenco Andrei

January 16, 2017

Introduction

Topic

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

Objectives

1. Initiation in Analog to digital converter (ADC)
2. Connecting Temperature sensor to MCU

Objectives

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.

1 What is the ADC?

1.1 Definition

Analog-to-digital conversion is an electronic process in which a continuously variable (analog) signal is changed, without altering its essential content, into a multi-level (digital) signal.

The input to an analog-to-digital converter (ADC) consists of a voltage that varies among a theoretically infinite number of values. Examples are sine waves, the wave forms representing human speech, and the signals from a conventional television camera. The output of the ADC, in contrast, has defined levels or states. The number of states is almost always a power of two – that is, 2, 4, 8, 16, etc. The simplest digital signals have only two states, and are called binary. All whole numbers can be represented in binary form as strings of ones and zeros.

1.2 Usage

Micro-controllers 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, micro-controllers make it economical to digitally control even more devices and processes. Mixed signal micro-controllers are common, integrating analog components needed to control non-digital electronic systems.

1.3 How does the micro-controller operate?

Even though there is a large number of different types of micro-controllers and even more programs created for their use only, all of them have many things in common. Thus, if you learn to handle one of them you will be able to handle them all. A typical scenario on the basis of which it all functions is as follows:

1. Power supply is turned off and everything is still the program is loaded into the micro-controller, nothing indicates what is about to come

2. Power supply is turned on and everything starts to happen at high speed! The control logic unit keeps everything under control. It disables all other circuits except quartz crystal to operate. While the preparations are in progress, the first milliseconds go by.

3. Power supply voltage reaches its maximum and oscillator frequency becomes stable. SFRs are being filled with bits reflecting the state of all circuits within the micro-controller. All pins are configured as inputs. The overall electronics starts operation in rhythm with pulse sequence. From now on the time is measured in micro and nanoseconds.

4. Program Counter is set to zero. Instruction from that address is sent to instruction decoder which recognizes it, after which it is executed with immediate effect.

5. The value of the Program Counter is incremented by 1 and the whole process is repeated several million times per second.

1.4 Special Function Registers (SFR)

Special function registers are part of RAM memory. Their purpose is predefined by the manufacturer and cannot be changed therefore. Since their bits are physically connected to particular circuits within the micro-controller, such as A/D converter, serial communication module etc., any change of their state directly affects the operation of the micro-controller or some of the circuits. For example, writing zero or one to the SFR controlling an input/output port causes the appropriate port pin to be configured as input or output. In other words, each bit of this register controls the function of one single pin.

1.5 Program Counter

Program Counter is an engine running the program and points to the memory address containing the next instruction to execute. After each instruction execution, the value of the counter is incremented by 1. For this reason, the program executes only one instruction at a time just as it is written. However the value of the program counter can be changed at any moment, which causes a "jump" to a new memory location. This is how subroutines

and branch instructions are executed. After jumping, the counter resumes even and monotonous automatic counting +1, +1, +1...

2 Resources

2.1 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.

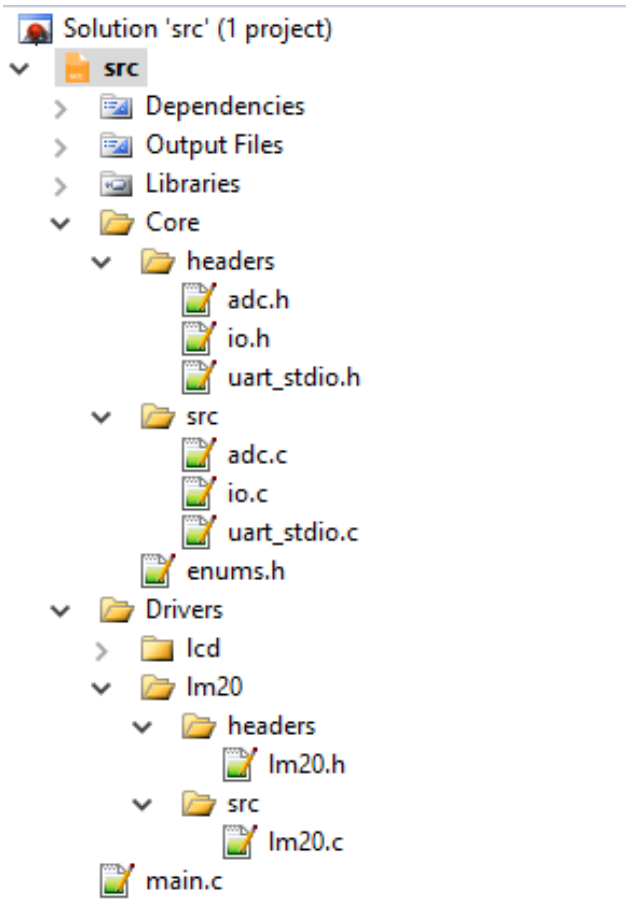
2.2 Proteus

The Proteus Design Suite is an Electronic Design Automation (EDA) tool including schematic capture, simulation and PCB Layout modules. It is developed in Yorkshire, England by Lab center Electronics Ltd with offices in North America and several overseas sales channels. The software runs on the Windows operating system and is available in English, French, Spanish and Chinese languages.

3 Solution

3.1 Project Structure

This is the project solution, I've added here a core folder where I've implemented a simple controller connection abstraction and some simple methods to deal with bitwise operations. Also I've added drivers for uart and lcd to output the temperature and implemented lm20 driver.

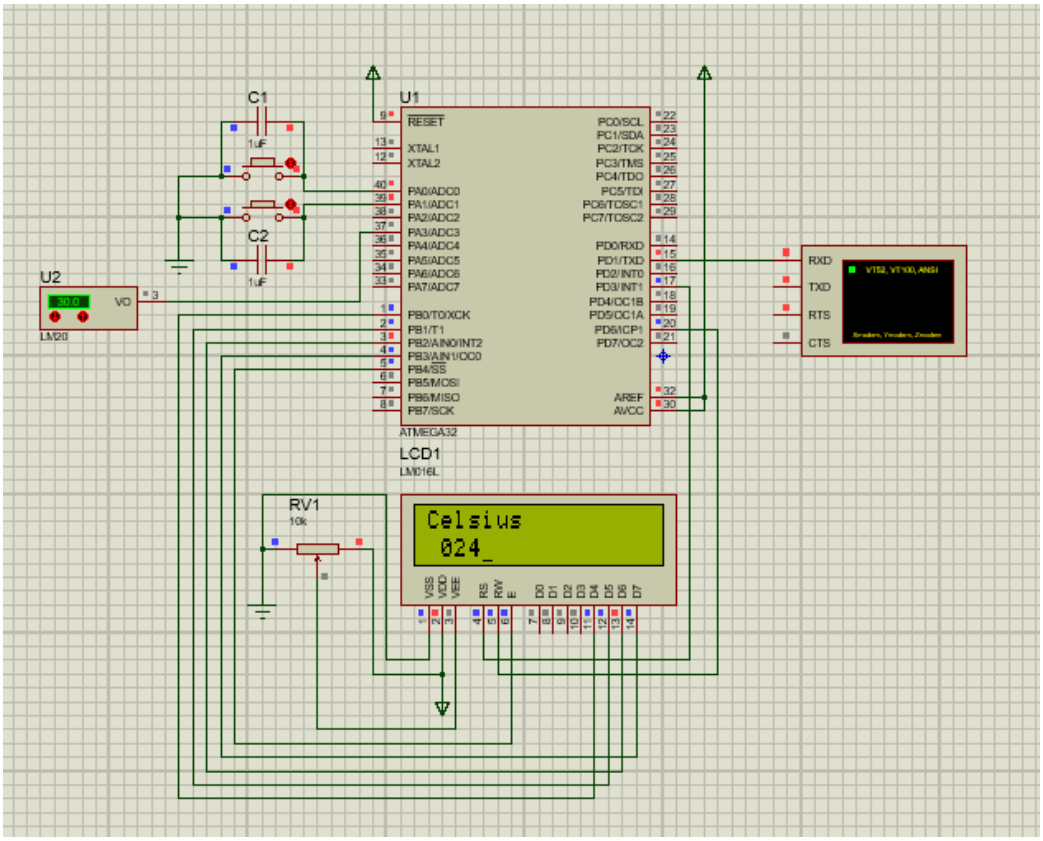


3.2 Main program flow

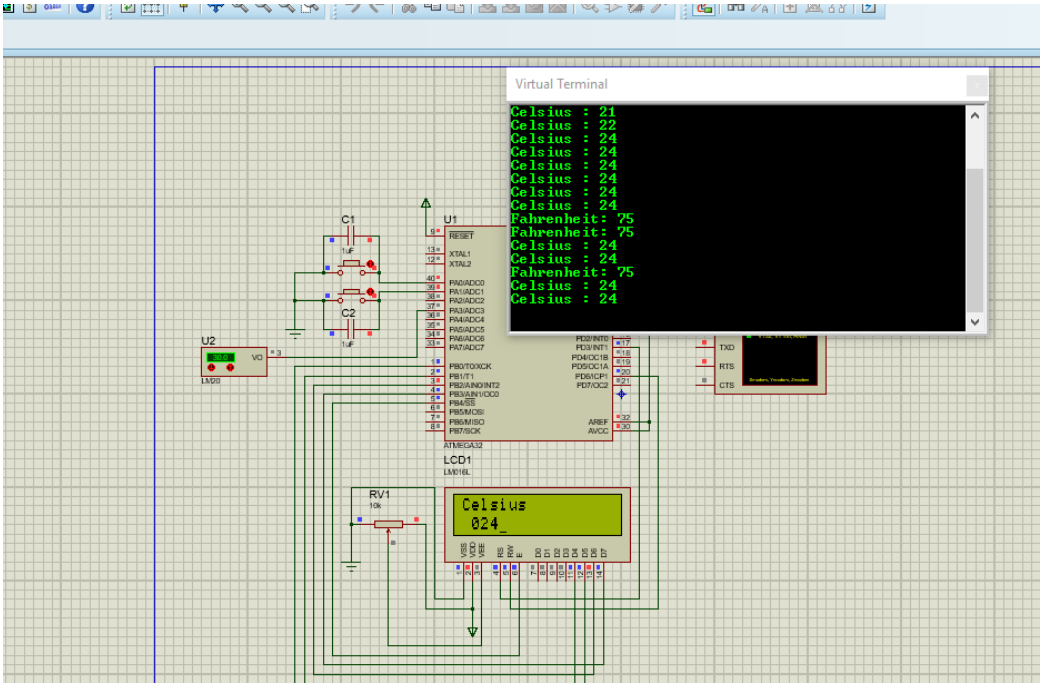
1. Global variable declarations.
2. button1 connection
3. button2 connection
4. lm20 init
5. LCD init
6. uart init
7. Start infinite loop (Controller life cycle)
   (a) get temperature from lm20
   (b) if button1 pressed then display kelvin
   (c) if button2 pressed then display celsius

3.3 Circuit in Proteus

I've connected MCU to Virtual Terminal. Because I use only data transmission on one direction (OUTPUT) we need to make sure that our MC Tx is connected to Peripheral Rx. Also I connected LM20 to ADC pin 3 and also connected LCD.



3.4 Simulation



Conclusion

In this laboratory work I've learned basic concepts of MCU programming in C language and building a simple printed circuit using Proteus. I've implemented a driver for LM20 and using this driver I've did a small embedded system that get the temperature from LM20 and displays it in a virtual terminal and also on LCD.

Apendix

Main

```
/*
 * main.c
 *
 * Created: 10/22/2016 8:39:04 PM
 *  Author: max
 */

#include <stdio.h>
#include "Core/headers/uart_stdio.h"
#include <util/delay.h>

int main(void)
{
    while(1)
    {
        uart_stdio_init();

        int i = 0;

        while(1){
                i++;
                printf("Counter %d \n",i);
                _delay_ms(1000);
        }
        return 1;
    }
}
```

Uart_stdio

```
/*
 * uart_stdio.c
 *
 * Created: 10/22/2016 8:42:39 PM
 *  Author: max
 */
#include <stdio.h>
#include <avr/io.h>

#define UART_BAUD  9600
#define F_CPU 1000000UL

int uart_stdio_putchar(char c, FILE *stream)
{
        if (c == '\n')
        uart_stdio_putchar('\r', stream);
        while(~UCSRA & (1 << UDRE));
        UDR = c;

        return 0;
}

FILE uart_str = FDEV_SETUP_STREAM(uart_stdio_putchar, NULL, _FDEV_SETUP_

void uart_stdio_init(){
        stdout = stdin = &uart_str;

        #if F_CPU < 2000000UL && defined(U2X)
        UCSRA = _BV(U2X);               /* improve baud rate error by usi
        UBRRL = (F_CPU / (8UL * UART_BAUD)) - 1;
        #else
        UBRRL = (F_CPU / (16UL * UART_BAUD)) - 1;
        #endif
        UCSRB = _BV(TXEN) | _BV(RXEN); /* tx/rx enable */
}
```