Atmel Microcontroller in Data Acquisition and Device Control

Nathan Nguyen (nhnguyen2997@eagle.fgcu.edu)
An Nguyen (avnguyen@eagle.fgcu.edu)
Professor: Dr. Janusz Zalewski
# Table of Contents

Section 1: Introduction .......................................................... 2
Section 2: Problem Specification ........................................... 6
Section 3: Solution .............................................................. 12
Section 4: Implementation .................................................... 15
Section 5: Experimental Testing ........................................... x
Section 6: Conclusion .......................................................... x
Section 7: References .......................................................... x
Appendix ................................................................................. x
1. Introduction

1.1 Project Overview

Microcontrollers exist in every part of the world. Having components of a processor, memory, and peripherals with the ability of providing real time response to physical events (basically a microcomputer), they are mainly used as an embedded system. They are embedded in several types of machinery, including automobiles, telephones, appliances (kitchen or basic calculator), and peripherals for computer systems. The purpose of this project is to extend on the previous projects by previous students. In the previous projects, they have calibrated the temperature sensor into the microcontroller and the STK500 sent a temperature reading as an output every time the user sends an input.

Figure 1.1 STK500 Micro Controller, first connected to a computer where a program that connects to the controller is used. The code is written either in the program or a separate editor, and the program is later used to send the data to the controller where it runs it until the user resets the board.

Microcontrollers are used in a wide array of applications, ranging from automobiles, kitchen appliances, larger embedded systems, smartphones, digital cameras, microwave ovens, and much more. Every stoplight you drove through contained a microcontroller that controls the lights, or when it receives an input from
another microcontroller that makes the first one change the lights. The potential of the Microcontroller has been utilized since its creation in 1971, and it has expanded since then.

Life would definitely be harder without these microcontrollers. Their existence even extends to a refrigerator[1], where the temperature is controlled by the microcontroller or when the user sends an input to the microcomputer. However, the limit of this technology has definitely not been reached, and further expansion to possible better and smaller microcontrollers can lead to a very different future. A very practical and cheap technology, the microcontroller’s influence cannot be belittled in their help to advance technology and provide an easier life for people everywhere.

The Atmel STK500 is a starter development kit for microcontrollers. It’s a basic board, meant to help us understand Atmel products, and microcontrollers in general. The STK500 is composed of three sections: Control, Target Socket, and User Interface. The control section provides communication between the Host computer and the board using USB serial cables. The Target Socket section contains plastic dual in-line package (PDIP) sockets, programming headers, and configuration jumpers, and the User Interface section includes LEDs, switches and I/O headers. The control section includes the driving force for the entire board, the MCU. STK500 uses ATmega8535L. We need to program the devices installed on the board and control the functionality of the board.

![Modern Refrigerators, using embedded controllers](image)

**Figure 1.2 Modern Refrigerators, using embedded controllers [2].**

### 2. Problem Description
In this project, we will be extending the previous groups' works on the microcontroller. In the earlier projects, they had their boards programmed to read the current room temperature the board was in either through the input on the board, or from their website, where they were able to send an input from any location to the board, and send an output to the same location that sent the input. Figure 2.1 shows the previous group’s system diagram, and Figure 2.2 shows their configurations. Our **primary objective** is to control the temperature reading/outputs, control the board through user inputs, and create an idle-time process for the board. Figure 2.1.7 is a simple depiction of our objective. Secondary objectives include giving a complete “Dummy guide for basic microcontroller board users”, and helping the readers understand hardware and software interactions.

### 2.1 Pre-Experimentation

Before the start of the project, one of the Atmel board was broken, and we couldn’t tell until we found that the other board worked with the same connection, and setup. The other problems we encountered were the computer’s abilities to detect ports, and displayed ports were not found sometimes. There are many input pins/ports, and we had a hard time understanding what the pins were used for. The stk500 manual may be there, but it is very hard to understand since we are actually hardware beginners. We have encountered several more issues, which will be discussed throughout this manual. Understanding things like the STK500 diagram shown in 2.1.6 is hard to understand at first.

### 2.2 Understanding the Software Needed

We used CodeVisionAVR (CVAVR) Evaluation version 3.0 to program our board. The free software offered excellent user-friendly libraries and settings and also generated source files needed to program the board. Originally, AVR Studio 4.16 was the software used to program the board, but the software lacked the ability to generate codes/files based on the settings chosen and needed for board configuration, and we lacked the abilities and time to create it ourselves. We knew however, that we will be using C programming, or specifically, embedded C. However, do not be afraid of the different terminology as the syntax is the same for both, but with different library requirements for both.
Figure 2.1 Previous Group’s software diagram, Nicolas Barragan and Andres Santamaria. It actually looks similar to our diagram, but the truth of the matter was that it was not completely user-controlled inputs, which we are working on in this draft. We have reused some of their codes (although standard) to shorten our total amount of codes.

Figure 2.2 Previous Group’s board configuration, Nicolas Barragan and Andres Santamaria.
2.1 Required Materials

- (1) AC adapter (shown in figure 2.1.1)
- (1) 9-pin RS-232 cable (shown in figure 2.1.2)
- (1) USB to RS-232 serial adapter (shown in figure 2.1.3)
- (1) STK500 starter kit evaluation board with ATMega8515L chip (shown in figure 2.1.5.1, figure 2.1.5.2)
- (1) CodeVisionAVR Evaluation 3.0
  Available at http://www.hpinfotech.ro/html/cvavr.htm
- (1) STK500 jtag adapter
- (1) TIP3055 Transistor NPN silicon
- (1) Micro motor
- (1) myTWI Temperature-Sensor (shown in figure 2.1)
- (1) 6-wire cable for In-System Programming (shown in figure 2.1.4)
- (2) 10-wire cables for I/O ports and parallel mode programming (shown in figure 2.1.4)
- (4) 2-wire cable for UART and DataFlash connections (shown in figure 2.1.4)

The minimum hardware and software requirements are:

- 486 processor (Pentium® is recommended)
- 16 MB RAM
- 12 MB free hard disk space (AVR Studio)
- Windows® 95/98/2000/ME/Windows NT® 4.0/or higher/XP/Vista/7/8
- 115200 baud RS-232 port (COM port)
- 10 - 15V DC power supply, 500 mA min.

Additional Resources:

ATMEL STK500 User Guide
Figure 2.1 AC Adapter [3]

Figure 2.1.1 AC Adapter [3]

Figure 2.1.2 9-pin RS-232 serial cable [3], two serial cables recommended

Figure 2.1.3 USB to RS-232 serial adapter [3]
Figure 2.1.4 from left to right. 6-wire cable, 10-wire cable, 2-wire cable [3]

Figure 2.1.5.1 STK50 starter kit evaluation board [3]

Figure 2.1.5.2 ATmega8515L-8PU installed on a STK500 PCB at target socket SCKT3000D3 (the ATmega8515L-8PU is an 8-bit RISC MCU in PDIP form)
Figure 2.1.6 Components of the STK500. It is recommended to look at the diagram several times to understand.

Figure 2.1.7 Basic UML diagram of what is expected at the conclusion of research. ATMEIL Micro-Controller will receive data from Temperature Sensor then converts to the temperature specified by user and sends to terminal the temperature, all through user input. Other user controlled inputs is turning on/off the motor. Another button allows user to play with lights, but excluded in this diagram.
3. Solution

Our solution is to code the ports in the stk500 board. Port A will be Switches, Port C will be for the LEDS. To program the stk500 we have to connect SPROG3 to the ISP6PIN. PORTE is a special port, and it allows us to connect the temperature sensor to the board. Port B will be used for terminal communication. We actually do not need the use of Port C for LEDS, but we included a fun gimmick for our board. Figure 3.1.1 is a more developed diagram of our system for development.

3.1 Development

At first, we had actually no clue on how to proceed with our project. So the most obvious thing to do would be to read the manual for the stk500 board, avrstudio 4.16, and about the ATMega chip itself. More about these specs will be included in the Implementation and Experimental Testing sections. Our first step was to connect the board to the computer. When we were given the equipment and the box of the board, the required supplies were insufficient, such as the power supply, or half of it, so expect missing equipment. Most of the time gathering information was from the published drafts of teams who also experimented on the STK500 board, but nearly all of them were incomplete, albeit the first draft by Vincent Rosa, which covers several aspects extensively. But even with the information, understanding the hardware and setup requires time and patience. This draft also provides example codes for test and basic understandings.

Figure 3.1.1 A more developed software diagram of our system based on Figure 2.2. LEDs are not included because they are primarily used for testing for all users (can be called a hidden element). The main component of our software design is only included. However, we have included the LED portion of the code in the Appendix.
3.2 Code Design

Below is a pseudo code of what we were planning to code so that everything becomes user based input. Basically, we have 3 ports, PORTA is for inputs, mostly for the user, PORTC is for outputs (we actually do not need PORTC, but it’s a good testing material), and PORTE, where it is a special port, and we can query for temperature readings from the myTWI sensor. I will explain what PIN is later, but think of Port IN.

```c
int TEMP = 0;

while (1) { // forever loops, main function where everything is running in void main
    PORTA input OPEN
    PORTC output OPEN
    PORTE activate TEMPERATURE SENSOR

    if(PINA.SWITCH=1) //CELSIUS READING
        TEMP=READ_TEMP();
        printf("Temperature: %d", TEMP)

    if(PINA.SWITCH=2)
        TEMP=READ_TEMP();
        TEMP=FAHRENHEIT_EQU;
        printf("Temperature: %d", TEMP);

    if(PINA.SWITCH=3)
        PINB= output OPEN;
        PINB TURN ON MOTOR;

    if(PINA.SWITCH=4)
```
PINB= output OPEN;

PINB TURN OFF MOTOR;

}  //end while loop
4. Implementation

To understand the board further, a short program was written to understand the output and input of the PORTS on the board. The board has to be set up a certain way if you program it in a different way. The need to understand the PORTS are very crucial to understanding the board. Figure 4.1 shows a physical representation of the Ports and the board’s interaction with it.

![Physical representation of the board's interaction with Woodward](image)

**Figure 4.1 Physical representation on how the board will connect/relate the pins, and the user controls those pins.**

4.1 Description

What the program basically do, is what was shown in the pseudo code in Section 3. The user sends a request to the board, and micro-controller will do what is requested upon the user pressing a button. However, the problem occurs when then pin provides voltage to the equipment. Although is a total of 5V per pins, the ampere is actually only up to 0.25 to 0.5. So for the motor, we had to provide a separate power source. If we didn’t, two things can happen, the motor drains too much power from the board, rendering it disabled, or the motor barely has power to rotate, so it’s actually important to consider other outside power source. Figure 4.4.1 shows an slightly updated version when we included the LED flashing lights.
4.2 Set Up

We will provide you with the exact steps on setting up the board through CodeVisionAVR if you have configured the board correctly. Below is a figure on how to connect the board, and temperature sensor, without the motor. The connection of a motor is more difficult to create because a complete circuit is needed to power the motor, since we are using a power source.

Here are the steps to connect the board without the motor:

- First connect the power source AC to the wall, and connect the other plug to the board
- Connect a serial cord USB to the computer and the other serial wire to the board insert plug on the right, where the right represents the computer to board programming and the left insert represents the communication from the board to computer
- Get a 6 pin wire and connect that between ISP6PIN and SPROG3
- Get two 10 pin wires, and connect one from PORT A to SWITCHES, and PORT C to LEDS.
- Connect a 2 wire pin from RS232 SPARE to PORTB SW0 and SW1
- Grab the myTWI and two 2 wire pins, and refer to Figure 4.4.2. Holding the myTWI front and forward, connect a 2 wire pin to the left most side, and connect that to PORTE PE0, PE1, and another 2 wire pin and connect that to 4 pins to the right of the earlier 2 wire pins, and connect that to the PORTE GND, VTG.
- Turn on the board, if the led status is red, switch the color placement for the PORTE GND and VTG. If it is still red, check the voltage amount of your power supply, and refer to Section 2.2

Figure 4.2.1 This is the exact configuration used for testing the board and myTWI sensor, motor testing is later. We used two serial communication wires for the Terminal and to send our program to the board. It’s not required, but it saves time, if you have two.

1. To start, first install CodeVisionAVR Evaluation 3.0, online by searching for the download in Google. After that, follow the Figures 4.2.2-4.2.16 in the images to connect to the board after you have the same configurations in Figure 4.2.1.
1) Open CodeVisionAVR
2) Click on File > New > Project

3) Click Yes

4) Pick AT90
5) Click OK

Figure 4.2.2 Create a new Project file.

Figure 4.2.3 Click yes

Figure 4.2.4 Based on what chip your using, but we are using ATmega.
Figure 4.2.5 Make sure you choose ATmega8515L, and the clock is 3.68 MHz, very important.

Figure 4.2.6 The reason why we change these settings is to make it easier to program, and use.

6) Chip: ATmega8515L
7) Clock: 3.68 MHz
8) Click Ports
9) Click the small box from Bit 0 to 7 to change the value from T to P
10) Click Port C tab.
Atmel Microcontroller in Data Acquisition and Device Control | 4/19/2013

Figure 4.2.7 Very important that PORTC is output, and value is already set to 1 for output.

11) Change all Data Direction from Bit 0-7 listbox from In to Out
12) Change all Pullup/Output Value from Bit 0-7 listbox from 0 to 1
13) Click on USART

Figure 4.2.8 The baud rate is 115200, this is very important in any basic boards because it defines rate of data speed, in receiving and sending.

14) Check Receiver, and Transmitter checkbox
15) Change Baud Rate: 115200
16) Click Bit-Banged I2C Bus Interface
17) Change I2C Port: PORTE
18) Check Enable in LM75 tab
19) Click Generate, Save, and Exit. Choose your folder, and the same saves files name

Figure 4.2.9 This setting is to include libraries for the myTWI sensor, and specifically its chip, LM75.

20) Code template generate, and a place to code the STK500 board

Figure 4.2.10 Just a small outlook when you finish generating your codes, this is what AVR Studio lacks
Figure 4.2.11 Very important, you must click on communication port box, and choose it directly so it is set, otherwise it assumes default, which you do not want.

Figure 4.2.12 Now to configure the project so that you can program the board.
24) Check Program the Chip
25) Click OK on the popup
26) Click OK

Figure 4.2.13 You always have to do this setting for any projects so that when you build the project, you can program the board.
Figure 4.2.14  This is another important aspect of the settings. Temperature output only appears in terminal, and CVAVR has its own terminal feature. Like for the Programmer settings, make sure you choose the port again, or it will be default and wrong. MAKE SURE THE BAUD RATE IS 115200, otherwise you get indistinguishable character. Test it so you can understand. Baud rates do not matter much in newer boards though.

29) Click Setting > Terminal
30) Port: COM(x) STK500 location
31) Click OK

Figure 4.2.15 To open Terminal Window, click on ‘Tools’ top tab, and click on ‘Terminal’. It should open up if you configured the terminal settings right, regardless if you choose the same ports for programmer and terminal.

32) Terminal Windows will appear
33) Click Connect to enable terminal communication
4.4 Setting up the Motor and Information about the myTWI Sensor

The main reason why the motor has its own section is because it deals mainly with hardware, rather than software. Understanding circuits is required, and additional equipment is highly likely to set up the motor. In our project, we used a separate power source so the board does not have to power the motor. The temperature only requires a small amount of power to work, but the motor is different because it requires a lot more amperes than what the board can provide per pins. From an observational view, the motor will move very slowly if powered by the board, and the voltage power has to be consistent. In figure 4.4.1, a diagram on how to connect the motor is shown. Each PD shown on the ports represent 1 pin, and each pin draws an amount of 5V when on. So for the motor, we only need 1 pin, while the transistor connected to the board connects to the 1 pin. The transistor plays a very important part because it is the ‘on/off’ switch for the power source to the motor. For instance, when the pin is on, the board sends a small amount of amps through the wire which is connected to the transistor. If enough power (5V needed for transistor) is sent through, the transistor activates and the emitter allows the power source to send power to the motor. This is how we powered the board. Soldering wires and black tape might be required to close the wire connections. Refer to the appendix section and check the codes for the comments our groups provided for each function and how to power on and off the motor independently.
Figure 4.4.1 A diagram on how to set the circuit up with the motor, power source, the STK500, and transistor.

The temperature sensor is simpler to connect as there are direct connections from board to myTWI. Figure 4.4.2 shows where and why we connect two 2 wire cables to myTWI to the board.

Figure 4.4.2 A diagram representing the myTWI. Important information about this is pins 13 and 14, which represent a circuit and connection for power. 17 and 16 represent UART, or communication with the board.
4.5 Notes about the Set Ups and Additional Options

1. For the terminal, we can use other programs to open it. Examples include Putty, HyperTerminal (for Windows XP and below), etc.

2. As stated before, two serial ports are recommended, but not required. For instance, you can send the program first, and then reconnect the serial cable to the terminal communication port, and then activate terminal.

3. RXD to PD0 and TXD to PD1 is necessary to connect (using two pin connectors) so that we can send data from the board to the computer, or the receiver. Without it, we cannot receive any data whatsoever from the board. (Read STK500 for further information)

4. It is not necessary to use CVAVR for the Atmel board, but it is extremely recommended for its useful libraries and user-friendly user interactions, or GUI.

5. A very important note about setting up the hardware is to make sure you follow the wire color and they are inserted the correct way, not reversed. Figure 4.5.1 provides an example of what the board looks like if the board ‘crashes’. It happens due to your program, and your hardware set up as it happened due to the equipment drawing too much power from the board. To know if the board has crashed, the only LED light on is the Power LED.

6. Going back on 5, if you only see the Power light LED light up, than your power supply does not supply enough power to the board. The required voltage is 10-15.5 V. This is also shown in the Section 2.1 Project Requirements.

7. Make sure the serial communication cables you are using is compatible with your OS system, we had this problem when we didn’t check the equipment we were buying.
5. Experimenting Testing

In the beginning, we tested the board responses by playing the LEDs. A very simple test was to configure PORTA and PORC so they are set for input and output respectively in Figure 5.1.

Below is an example of a 10 wire cable connected from LED to PORTB and another 10 wire cable from SWITCHES to PORTD. Reconfigure your board so that it follows the description given, and test this program.

//The code below will turn on every light except switch led itself
unsigned char human; // temporary variable
DDRD = 0x00; // set PORTD for inputs
PORTD = 0xFF; // turn on portD
DDRB = 0xFF; // set PORTB for outputs
while(1) // the condition will run forever
{
    human = ~PIND; // switches are controlled the user (human variable)
    switch (human) {
        if (human == 0x01) // if SW0 is hold
            PORTB =~0xFF; // then all leds should be light on
        else if (human == 0x02) // if SW1 is hold
            PORTB = ~0xBF; // this will light all the leds except LED1
        // if none of the button is hold, then PORTB(LEDS) will turn off
        else PORTB = ~ 0x00;
    }
}

This is a very good example program that can help you further understand ports and pins. Use this code to test the board.
Figure 5.1  Testing board response and temperature sensor, this picture is not part of the code above

We had a huge amount of trouble trying to figure out why we couldn’t get any output from the terminal. When we tested our codes to get the temperature, we had no clue we needed to connect the USART and the terminal, so we had to read the manual and the drafts to learn how to obtain any output. Figure 5.2 shows this, and later on when we did connect it, we had no idea that the baud rate was so important, shown in Figure 5.3.

When we finally figured it out, it was time to test the motor, which was the hardest part of this project. I even created another section just for the motor. Basically the problem was that the motor never turned on, and when I configured it right, the board crashes and a red status is shown for the board.

That’s when we decided to use a transistor and a separate power source, but it took awhile before we actually did decide to change our plan to power the board, shown in figure 5.4.

After getting the transistor and power source, we had trouble learning how to make a connection. We didn’t really know what GND meant on the board, but it means ‘Ground’ which is like a base for connections to go through, which connected our wires and made a circuit, shown in figure 4.4.1. Figure 5.5 shows our final configuration afterwards. But a problem with consistent voltage and amps flowing through became a small problem, but it was fixed with cutting wires, twisting them, and taping them so power is consistent.
Figure 5.2 We had no idea we needed to connect the serial cable to the right side of the board for communication, but we eventually got it.

Figure 5.3 The baud rate was supposed to be 115200 for the USART and the terminal. If they were different, undistinguishable characters were shown.
Figure 5.2 Terminal output when board queries the myTWI temperature sensor, the values are converted first.

Figure 5.5 Final configurations of the boards and wires.
After finishing our program that allowed us to do user inputs, we had another test for idle processing. Basically, when the board is doing nothing, it is constantly reading the temperature. When the temperature pasts a certain limit, the motor will turn on indefinitely until the temperature is lowered, and then it will turn off. During idle time, the board is still able to play with LEDs, read temperature, and stop motor. However, to represent RTS for the board, when the motor is running indefinitely because of the temperature, the motor can only be stopped by holding SW4, otherwise if it is turned on manually, than it can be stop with a push of a button. It took us some time to understand programming the board, and we finally figured out that we can program the board like a regular program, taking into process the sequential process the board has and making use of functions. Previous groups used gotos, but those are really hard to understand and functions are more readable.
6. Conclusion

We were able to further understand embedded system by working with a basic Atmel board that users have to configure hardware and software features to use. Although all boards have different design and specifications, all boards also have general concepts. Such concepts are PINS, I/OS, sufficient power, USART communications, and connectors. Our STK500 is able to query for the temperature and convert values (Celsius and Fahrenheit) all through a push of a button, where the temperature is sent to the terminal and it displays the value. Our board is able to turn on the motor and turn it off with a push of a button. Our board also contains an extra feature that plays repetitive LEDs for awhile, although the value is not randomized. This concludes the simple features of the board.

This project is no joke. It was a hellish experience for my group since we had no experience in hardware, and that took us a lot of time understanding circuits, connecting wires, etc. However, it was fun when we finally got the board to work, and buying equipments also came as a learning experience. I don’t recommend this board for future use. Reasons why is that the libraries are probably outdated, there are more BASIC and recent boards that can be used to help others understand embedded systems and work hands on with better libraries and equipment. Heck, there are boards where they have libraries for direct communication with computer, rather than using terminal, where if you wanted the information to go in a computer file, or web, you had to create your own program for it. There were no libraries for STK500 board and the ATmega8515L chip. So forget about the Atmel board; instead, get an Arduino board and learn how to configure and program it.

The future is looking bright however, since this is a very basic and old microcontroller. Look at everything that surrounds us now, and almost all the electronics, home appliances, automobiles, and infrastructures are embedded systems, and they are still improving them. A very strong example of a advanced microcontroller would be the iPhone, or Samsung Galaxy, where they further improved the technology with direct touching with the human fingers, and that was years ago. There are endless possibilities for mankind through embedded systems, and doing this project is one step towards understanding embedded system and utilizing this tool and knowledge for the future.
7. References


http://www.loghome.com/modern-conveniences-built-into-refrigerators/


http://itech.fgcu.edu/faculty/zalewski/projects/files/

APPENDIX

/*******************************************************************************
Project : Atmel Microcontroller
Version : 2.5 Final Version
Date : 4/21/2013
Author : Nathan Nguyen, An Nguyen
School : Florida Gulf Coast University
Chip type : ATmega8515L
Program type : Application
AVR Core Clock frequency: 3.686400 MHz
Memory model : Small
External RAM size : 0
Data Stack size : 128
*******************************************************************************/

#include <mega8515.h>
// I2C Bus functions
#include <i2c.h>
// LM75 Temperature Sensor functions
#include <lm75.h>
// Standard Input/Output functions
#include <stdio.h>
// Contains "printf()" and sprintf()
#include <delay.h> // Contains "delay_ms()"
#include <stdlib.h>

// Contains "abs()"

// Delay time in milliseconds (ms)

void motorIdle(int, int, int);
void motorStop();
void motorRun();

void main(void)
{
    // Declare your local variables here
    int tempC_x10=0;
    int i;
    int j;
    int temp;
    int limit=25;

    // Input/Output Ports initialization
    // Port A initialization
    // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
    // State7=P State6=P State5=P State4=P State3=P State2=P State1=P State0=P
    PORTA=0xFF;
    DDRA=0x00;

    // Port B initialization
    // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
    // State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
    DDRB=0xff;
    PORTB=0x00;
/ Port C initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=1 State6=1 State5=1 State4=1 State3=1 State2=1 State1=1 State0=1
PORTC=0xFF;
DDRC=0xFF;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0xff;
DDRD=0x00;

// Port E initialization
// Func2=In Func1=In Func0=In
// State2=T State1=T State0=T
PORTE=0x00;
DDRE=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// InputCapture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
EMCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity

// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 115200
UCSRA=0x00;
UCSRB=0x18;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x01;
// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
// I2C Bus initialization
i2c_init();
// LM75 Temperature Sensor initialization
// thyst: 75°C
// tos: 80°C
// O.S. polarity: 0
lm75_init(0,75,80,0);

while(1)
{
/*
 * The "int lm75_temperature_10(unsigned char chip)" function
 * returns the temperature in degrees C times 10 retrieved from
* the LM75 sensor with the address "chip".

* CAUTION: A 300ms delay must be present between two successive calls to this function.

*/// Get the temperature as an integer in degrees Celcius times 10

DDRA=0x00;

for(i=0;i<100;i++) {

    tempC_x10 = lm75_temperature_10(0);
    temp = tempC_x10/10; //records temp for while process

    while(tempC_x10>limit) { //THIS MUST RUN FIRST FOR IDLE PROCESS
        motorIdle(tempC_x10, temp, limit);
        break;
    }
    motorStop();
}

for(j=0;j<1000;j++) {
    if(PINA.1==0x00) {
        for(i=0; i<5; i++) {
            tempC_x10 = lm75_temperature_10(0);
            printf("Temperature: ");
            printf("%-i.%-u(C)", tempC_x10/10, abs(tempC_x10%10)); //needs to divide temperature by 10 to give real temperature
            printf("%-.1,%.u(C)\r", tempC_x10/10, abs(tempC_x10%10)); //needs to divide temperature by 10 to give real temperature
            delay_ms(300); //output slowed down
// IF SW 2 IS PUSHED, GENERATES TEMPERATURE IN Fahrenheit

if(PINA.2 == 0x00) {
  for(j=0; j<5; j++) {
    tempC_x10 = lm75_temperature_10(0);
    printf("Temperature: ");
    temp = tempC_x10/10, abs(tempC_x10%10);
    temp = (( temp* 1.8 ) + 32); // equation to convert the TEMP
    printf("%-i.%-u(F)\r", temp);
    delay_ms(300);
  }
}

// this runs if SW 3 IS PUSHED, causes motor to turn on

if(PINA.3 == 0x00) {
  motorRun();
}

// this runs if SW 4 IS PUSHED, causes motor to turn off

if(PINA.4 == 0x00) {
  motorStop();
}

// fun gimmick, plays with LEDs when SW 5 IS PUSHED

if(PINA.5 == 0x00) {
DDRC=0xFF;

for(i=0;i<=3;i++){
    PORTC=0x10;
    delay_ms(300);
    PORTC=0x1E;
    delay_ms(300);
    PORTC=0xD3;
    delay_ms(300);
    PORTC=0x01;
    delay_ms(300);
    PORTC=0x02;
    delay_ms(300);
    //the ports accept hexadecimal figures, so if you want different lights, change the last //two numbers/letters in the '0xNN', and you can test the LEDs.
    PORTC=0xFF;
}
// IF SW 1 IS PUSHED, GENERATES TEMPERATURE IN CELSIUS

void motorIdle(int tempC_x10, int temp, int limit) {
    int i=0;
    int j=0;
tempC_x10 = lm75_temperature_10(0);
temp=tempC_x10/10;
while(temp>limit) {
    tempC_x10 = lm75_temperature_10(0);
temp=tempC_x10/10;
    DDRB=0xFF;
    PORTB.2 =0xff;
    if(PINA.4 == 0x00) {
        DDRB=0xFF;
        PORTB.2 =0x00;
    }
    if(PINA.1==0x00) {
        for(i=0; i<5; i++) {
            tempC_x10 = lm75_temperature_10(0);
            printf("Temperature: ");
            printf("%-i.%-u(C)\r",tempC_x10/10,abs(tempC_x10%10)); //needs to divide temperature by 10 to give real temperature
            // the abs() is to get the decimal temperature, which gets combined with the real temperature for true temperature
            delay_ms(300);//output slowed down
        }
    }
    if(PINA.2 == 0x00) {
        for(j=0; j<5; j++) {
tempC_x10 = lm75_temperature_10(0);
printf("Temperature: ");
temp = tempC_x10/10,abs(tempC_x10%10);
temp = (( temp* 1.8 ) + 32); // equation to convert the TEMP
printf("%-i.%-u(F)\r",temp);
delay_ms(300);
}
}

if(PINA.5 == 0x00){
    DDRC=0xFF;
    for(i=0;i<=3;i++) {
        PORTC=0x10;
delay_ms(300);
        PORTC=0x1E;
delay_ms(300);
        PORTC=0xD3;
delay_ms(300);
        PORTC=0x01;
delay_ms(300);
        PORTC=0x02;
delay_ms(300);
        PORTC=0xFF;
    }
}

// the ports accept hexadecimal figures, so if you want different lights, change the last // two numbers/letters in the '0xNN', and you can test the LEDs.

PORTC=0xFF;
void motorStop() { // stops motor
    DDRB = 0xFF;
    PORTB.2 = 0x00;
}

void motorRun() {
    while (PINA.4 != 0x00) {
        DDRB = 0xFF;
        PORTB.2 = 0xff;
    }
}