**Introduction**

In this lab we are attaching a temperature sensor to the TM4C launch pad and reading in the values using an ADC.

The ADC (Analog to Digital converter) turns a voltage into a number. In our TM4C launch pads, the value is read in as an input voltage, and then is made into a 12-bit number and stored into a register. To do this, the maximum voltage is divided by the maximum value that the 12-bit register can hold. This gives us a resolution. The input voltage can be divided by this resolution to get the number it should be in the 12-bit register.

In this project, Alex worked on the temperature sensor, while Jake worked on the real time clock. Ammon worked on the real time clock as well.

The temperature sensor we are using the is TMP36. This temperature sensor has three pins, 5v in, ground and the analog output for the temperature. The sensor is rated to sense -40 to 125 degrees Celsius, with a conversion rate of 10 mv/degree and 750 mv at 25 degrees.

**Temperature sensor**

In this lab we used register writing for the temperature sensor. The following list of commands which were used to write to the appropriate registers.

ui32SysClkFreq = **SysCtlClockFreqSet**((SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_480), 120000000);

// Enable the clock to port D

SYSCTL\_RCGCGPIO\_R |= SYSCTL\_RCGCGPIO\_GPIOD;

// configuration of port D pin2

GPIO\_PORTD\_AHB\_DIR\_R |= PORTD\_DIR; //input set

GPIO\_PORTD\_AHB\_DEN\_R &= POTD\_DEN; //digital enable off

GPIO\_PORTD\_AHB\_AMSEL\_R |= PORTD\_AMSEL; //set alternate function registers

GPIO\_PORTD\_AHB\_AFSEL\_R |= PORTD\_AFSEL;

// Enable the clock for ADC0

SYSCTL\_RCGCADC\_R |= SYSCTL\_RCGCADC\_ADC0;

//turns off ADC0 for configuration

ADC0\_ACTSS\_R |= 0x0;

// delay

Delay (3);

//sets continuous triggering

ADC0\_EMUX\_R |= EMUX\_SS3\_DEFUALT;

// Select AN13 ( PD2 ) as the analog input

ADC0\_SSMUX3\_R |= SSMUX3\_AIN13\_SET;

// quarter conversion rate; 48\*Tadc periods pause

ADC0\_PC\_R |= PC\_WAIT48;

// 1st sample is end of sequence and source of interrupt

ADC0\_SSCTL3\_R |= SSCTL3\_INTEND;

// 16x oversampling and then averaged

ADC0\_SAC\_R |= SAC\_16X\_SET;

// Unmask ADC0 sequence 3 interrupt

ADC0\_IM\_R |= IM\_SS3\_ENABLE;

// Clear the interrupt for ADC0 sequencer 3

ADC0\_ISC\_R |= ISC\_SS3\_CLEAR;

// Enable ADC0 sequencer 3 interrupt in NVIC

NVIC\_EN0\_R |= NVIC\_SS3;

// Enable ADC0 module for sequencer 3

ADC0\_ACTSS\_R |= ACTSS\_SS3\_ENABLE;

// Initiate sequencer 3

ADC0\_PSSI\_R |= PSSI\_SS3\_START;

// wait certain time for ADC module do the conversion

Delay(100);

//loop for program to continue running

**while** (1) {

}

The first command used is,

ui32SysClkFreq = **SysCtlClockFreqSet**((SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_480), 120000000);

This command initiates the system clock, so that the ADC clock can be enabled and work properly.

The second command,

SYSCTL\_RCGCGPIO\_R |= SYSCTL\_RCGCGPIO\_GPIOD;

Enables the clock for the port D, by writing a 1 to its appropriate spot in the register.

The next set of commands

GPIO\_PORTD\_AHB\_DIR\_R |= PORTD\_DIR;

GPIO\_PORTD\_AHB\_DEN\_R &= POTD\_DEN;

GPIO\_PORTD\_AHB\_AMSEL\_R |= PORTD\_AMSEL;

GPIO\_PORTD\_AHB\_AFSEL\_R |= PORTD\_AFSEL;

Make the port D direction input, turn the digital mode off for port D as well as turn on the alternate function mode and enable the ADC circuitry.

The next command,

SYSCTL\_RCGCADC\_R |= SYSCTL\_RCGCADC\_ADC0;

Enables the clock for the ADC functionality, by writing a 1 to the part of the register for the ADC0 converter.

The following commands are both safety measures,

ADC0\_ACTSS\_R |= 0x0;

Delay (3);

The first disables the ADC0 temporarily so that we can change the setting without risking the possibility of it doing things, and the second is a short delay for after the clock enable so that everything is running before we try and change anything.

The next command,

ADC0\_EMUX\_R |= EMUX\_SS3\_DEFUALT;

Sets the ADC so that it continuously triggers, so that it is always taking another reading after it has completed the last. It does this by writing a 0xF000 to the appropriate register.

The next command,

ADC0\_SSMUX3\_R |= SSMUX3\_AIN13\_SET;

Sets the ADC sequencer to use AIN13 with corresponds with the GPIO pin 2.

The following command,

ADC0\_PC\_R |= PC\_WAIT48;

Sets a wait time after each sequence is taken, before the next sample starts.

The next command,

ADC0\_SSCTL3\_R |= SSCTL3\_INTEND;

Sets the end of sample to be the source of the interrupt, as well as making it trigger after only one sequence.

The next command,

ADC0\_SAC\_R |= SAC\_16X\_SET;

Sets the sequencer to do 16x sampling, which makes it read in 16 values, average them, then use that value to give to the 12-bit final value register.

The next commands,

ADC0\_IM\_R |= IM\_SS3\_ENABLE;

ADC0\_ISC\_R |= ISC\_SS3\_CLEAR;

Enable the ADC0 interrupt by writing a mask to it, as well as clearing the interrupt to make sure it is triggered currently.

The following command,

NVIC\_EN0\_R |= NVIC\_SS3;

Enables interrupt 17, which corresponds with the interrupt for ADC0.

The next commands,

ADC0\_ACTSS\_R |= ACTSS\_SS3\_ENABLE;

ADC0\_PSSI\_R |= PSSI\_SS3\_START;

Enable the ADCO sequencer 3, which we had disabled close to the begging in of the code. And starts the sequencer by writing the appropriate value to the starter.

The last commands

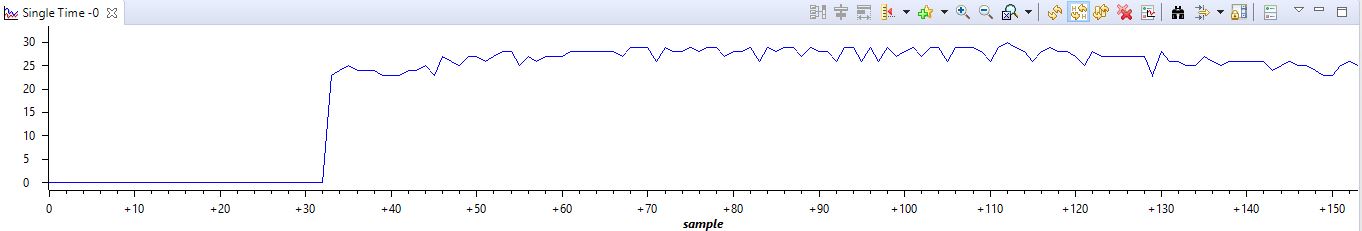
Delay(100);

**while** (1) {

}

Gives a delay for the sequencer to start working, then the while loop makes the program run indefinitely while it is taking in temperature data.

To make sure that our code was working we added the Data and Temperature variables to the watch list and turned on auto updating. We also made a graph for the temperature and turned on auto updating while the program was running. The graph we got from this is shown below.



This picture shows the temperature of the room and it changing when a finger is put on the sensor and then taken off.

To read the data from the destination register we used interrupts in the following code.

**void** **ADC0SS3\_Handler** (**void**) {

ADC0\_ISC\_R |= ISC\_SS3\_CLEAR; //clear interrupt

**int** DATA = 0; //input data

DATA = ((ADC0\_SSFIFO3\_R) & 0x0FFF); //read data in

Temperature = (((DATA\*.806) - 750)/10 + 25); //do conversions

ADC0\_ISC\_R |= ISC\_SS3\_CLEAR; //clear interrupt again, (for some reason this had to be done)

}

In this code the interrupt is cleared twice because that is what was found to work. The temperature conversion is done by getting the given value first in mV, which we can do by getting the resolution, which in this case is 3.3 V divided by the maximum value of the 12-bit register which is 4096. This gives us the resolution number of .000806. So, if we want our DATA value in mV, we multiply it by .806. Then to center it we minus 750 mV, so any leftover voltage is the error from 25 degrees. Then we divide by 10, because we get 10 mV per degree, now we have the error in degrees. Finally, we add 25 degrees to the error and we get our final temperature.