BME 580 Microcontrollers I

Lab 6: SENSORS and DATA LOGGING

Ross Rucho

In this lab you will continue to build upon previous labs by learning about the use of external and internal sensors in embedded systems and data logging. For this lab you will need to find your own temperature sensor. We have them in the lab and they look like this →

They are actually pretty small, generally smaller than an old school pencil eraser. This makes them convenient for placing into small spaces or inside a container.

Find a temperature sensor in the lab:

Steve ☺ keeps temperature sensors in the lab. All you have to do is find them. The three types we typically stock are: LM34DZ, LM35DZ, and MCP9700A-E/TO. All three are available from digikey.com. You can find datasheets for all three of these at digikey.com by searching for the component and then clicking on the link for the datasheet on the component page. We generally stock only the TO package shown above, though the same sensor does come in several different physical packages. At any given time we are unlikely to have all three of these available. We may only have one or two of these three types. You just need to find one.

Also, you already have a temperature sensor in your circuit. The PIC16F1829 has an *Internal Temperature Indicator* (ITI). This internal device is also sometimes called a die temperature sensor, as it directly senses the temperature of the integrated circuit die within the microprocessor chip. For this lab you will use one external temperature sensor (ETS) and the internal temperature indicator (ITI) on the PIC16F1829.

READING ASSIGNMENT

https://en.wikipedia.org/wiki/Sensor

PIC16F1829 DATA SHEET – Section 15.0: Temperature Indicator Module

Additional datasheets you will need to find on the Internet:

AN1333 – This is an "Application Note" (AN) published by Microchip that gives technical details on their products. The world of electronics is filled with this type of document, which is also sometimes called a Technical Note or Technical Brief. Basically they want to give you some information about how cool their device is so you will use it in your designs. Read through this 12-page document to see how to calibrate and use the internal temperature indicator on the PIC16F1829.

LM34DZ – Find and download the datasheet for this device.

LM35DZ – Find and download the datasheet for this device.

MCP9700A-E/TO – Find and download the datasheet for this device.

Also read: https://en.wikipedia.org/wiki/Data_logger

https://en.wikipedia.org/wiki/Temperature_data_logger

BME 580 Microcontrollers I

QUESTIONS FROM THE READINGS:

1- Use the information in the datasheets to build a table. List all three types of external temperature sensor and the internal temperature indicator on the microcontroller (PIC16F1829). Include the following data for each temperature sensor or indicator:
   a. Temperature range (maximum and minimum)
   b. Operating voltage range
   c. Calibration factor: mV/degree
   d. Accuracy
   e. Cost per device from digikey.com (assume you only buy one of them, not 100 or 1000)

Temperature Sensor	Temperature Range (MAX and MIN)	Operating Voltage Range (V)	Calibration Factor (mV/degree)	Accuracy (typical)	Cost Per Single Device From digikey.com ($)
LM34DZ	212 – 32 (°F)	30 – 4	10	± 1.2°F	3.45
LM35DZ	100 – (0) (°C)	30 – 4	10	± 0.6°C	2.05
MCP9700A	125 – (-40) (°C)	5.5 – 2.3	10.0	± 1°C	0.31
Internal	85 – (-40) (°C)	(3.6 or 1.8)	Determined By Calibration	Depends On Calibration	1.80

2- Find in the lab and select one of the external temperature sensors that you will use for this lab.
I will use the LM34DZ Precision Fahrenheit Temperature Sensor for this lab.

3- Based upon the technical specifications for the external temperature sensor you have selected, assume you plan to use that temperature sensor over its full linear range of temperatures.
   a. What range of output voltages corresponds to the full linear range of temperatures for the external temperature sensor you have selected?

      The LM34DZ has a transfer function equivalent to $V_{out} = 10.0 \frac{mV}{°F} * (°F)$.

      Therefore, over the entire temperature range of 32 – 212 (°F), the device will output a voltage from 0 – 1.8 V.

   b. What would be the best internal FVR (fixed voltage reference) value to select for your internal ADC voltage reference so that you get the maximum temperature resolution over the linear temperature range? You will need to look at the PIC16F1829 datasheet and/or the pic16f1829.h header file to determine what values of internal FVR are available on the PIC16F1829.

      The best internal FVR value would be 2.048 V because this is the smallest voltage that is larger than the greatest output voltage of the temperature sensor. This means that the entire range of voltages can be represented with the greatest resolution.

BME 580 Microcontrollers I

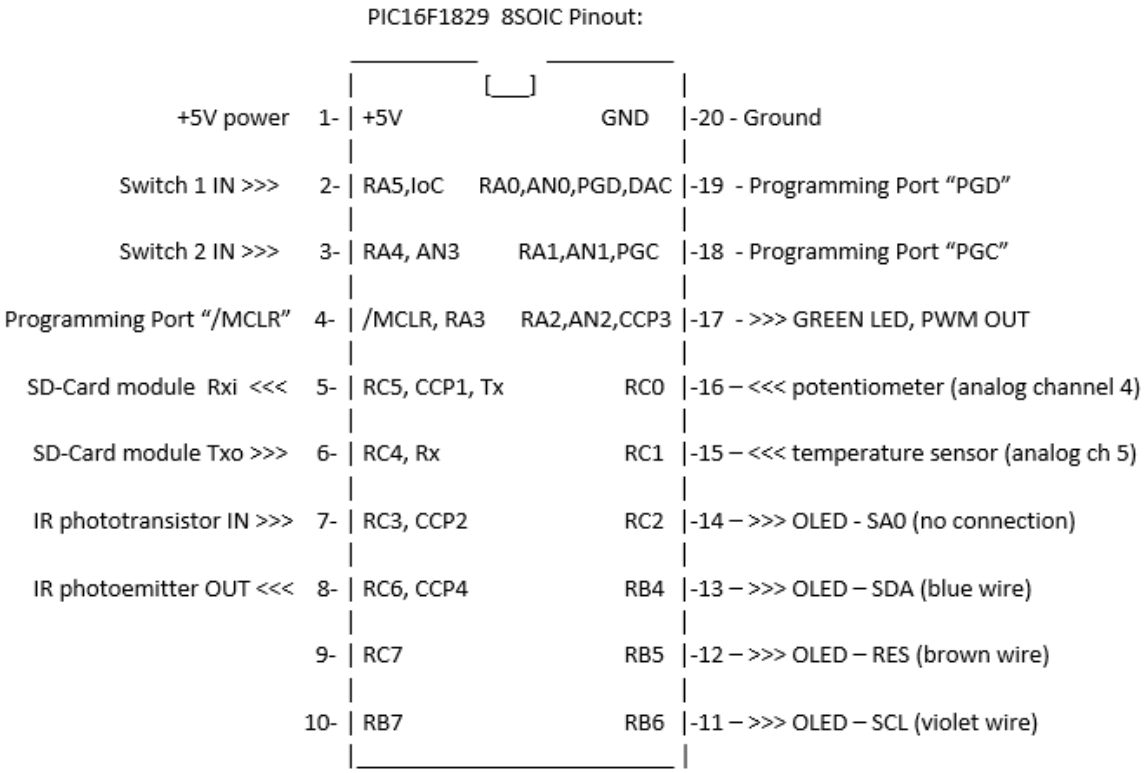
c. What is the temperature resolution using the optimal FVR and a 10-bit ADC?
Using an FVR of 2.048 V and a 10-bit ADC, voltage steps of approximately 2 mV can be measured. Since the temperature sensor has a calibration factor of 10 (mV/°F), the temperature resolution of our digitized signal is 0.2 °F.

BME 580 Microcontrollers I

Microcontroller Connections:

Building upon your earlier circuits, add a signal wire for the temperature sensor to pin # 15 of the PIC16F1829 as shown below. Don't forget to also add the power and ground line to the temperature sensor as indicated on the datasheet for the external temperature sensor you have decided to use.

PIC16F1829 8SOIC Pinout:

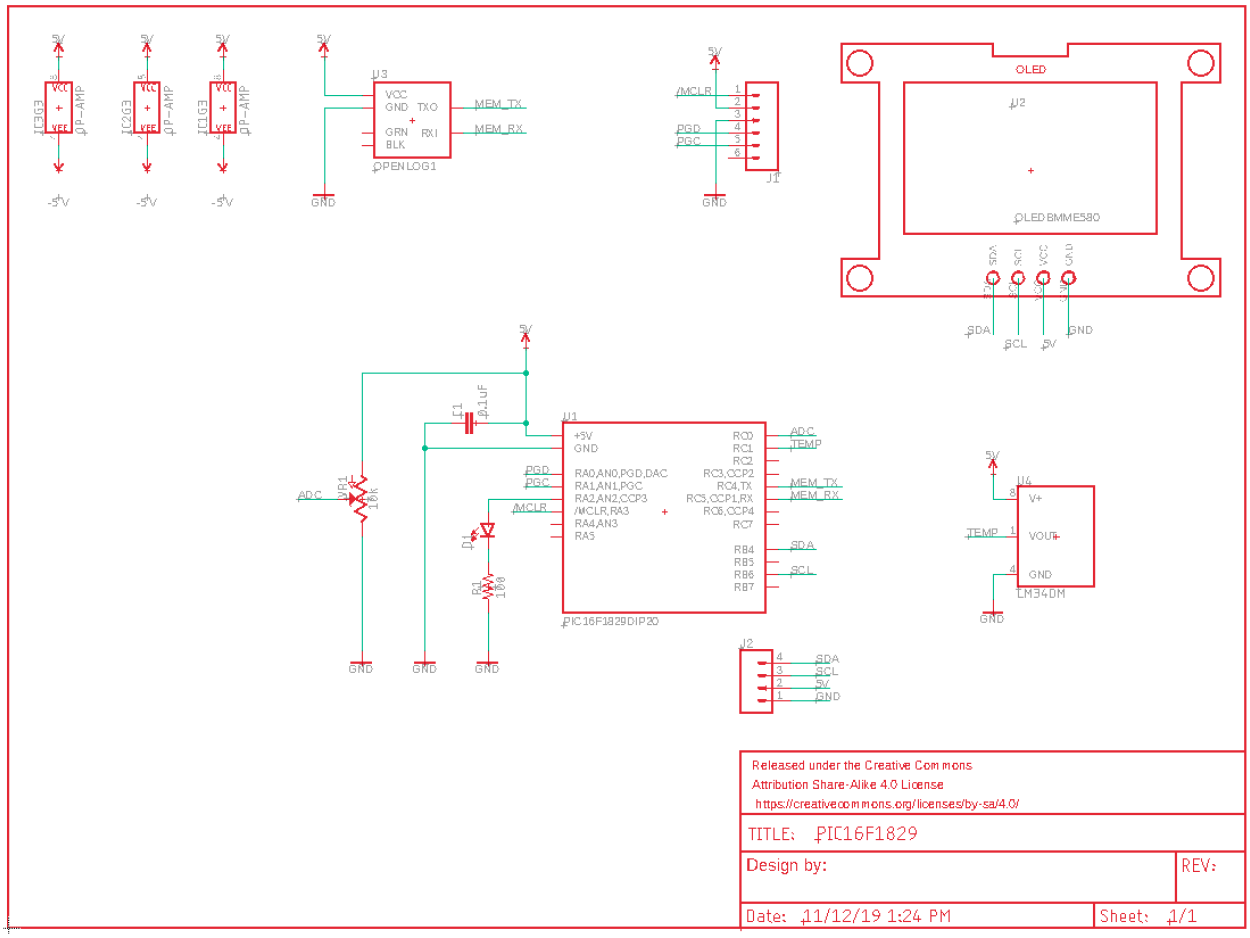


This time you will not use a spring terminal connector. Just hard wire the temperature sensor to your prototype PCB. Make the wires about 6 to 8 inches long so you can put the sensor into locations a little bit away from your PCB. These temperature sensors are inexpensive, so you do not have to return them at the end of the term.

Then modify your EagleCAD schematic to include the new component (external temperature sensor).

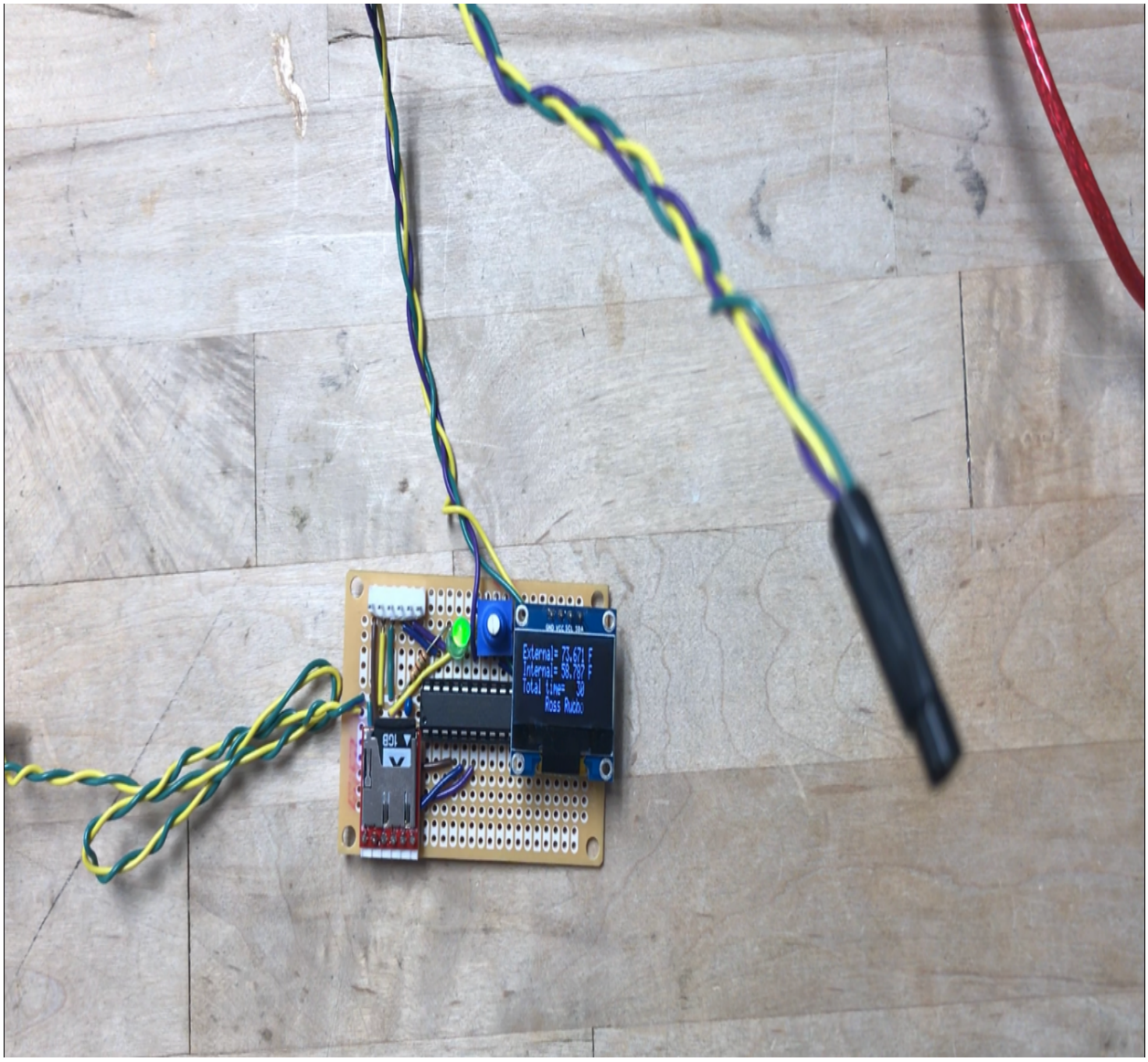
BME 580 Microcontrollers I

Your EagleCAD version of the modified circuit schematic with external temperature sensor (ETS):



BME 580 Microcontrollers I

Photograph of your completed circuit:



BME 580 Microcontrollers I

FIRMWARE:

Before you change your firmware, check to make sure your OLED display is still working properly using the firmware you installed since Lab #3. Also make sure that you can still read and display the voltage on the potentiometer as it changes when you rotate the potentiometer shaft, from Lab #4. Finally, make sure your SD Card Module is still able to log ASCII strings (text data). Make sure all three of these work at the same time, as you did in the previous lab (Lab #5: Memory)

For this lab you will be converting the temperature signal from one external temperature sensor (ETS) and the internal temperature indicator (ITI) into digital data using two different ADC channels. The microcontroller only has one internal ADC, so to make measurements of different analog signals you need to switch ADC channels before measuring the voltage. The ADC value (an integer) will be converted into a standard length floating point number to represent temperature, either Fahrenheit or Centigrade. You decide which units you want to use.

For the external temperature sensor, you will measure the analog voltage from the ETS on one of the analog input pins of the PIC16F1829 microcontroller. That means you will need to add source code to setup the new analog channel (see code examples from Lab #4: ADC), and then during code execution you will need to read that channel, so you will need to use the command:

```
set_adc_channel();
```

Do not forget to also add a short delay (10 to 20 microseconds) to allow the new channel to stabilize before trying to read the voltage on the new channel.

For the internal temperature indicator you also need to change the ADC channel and then wait a short period before reading the voltage on the internal temperature indicator. You need to use the following command:

```
set_adc_channel(TEMPERATURE_INDICATOR);

delay_us(20);          //  the delay after setting an ADC channel needs to be ~ 10 to 20 micro seconds.
```

For both measurements you should use the full 10-bit dynamic range available in the internal ADC.

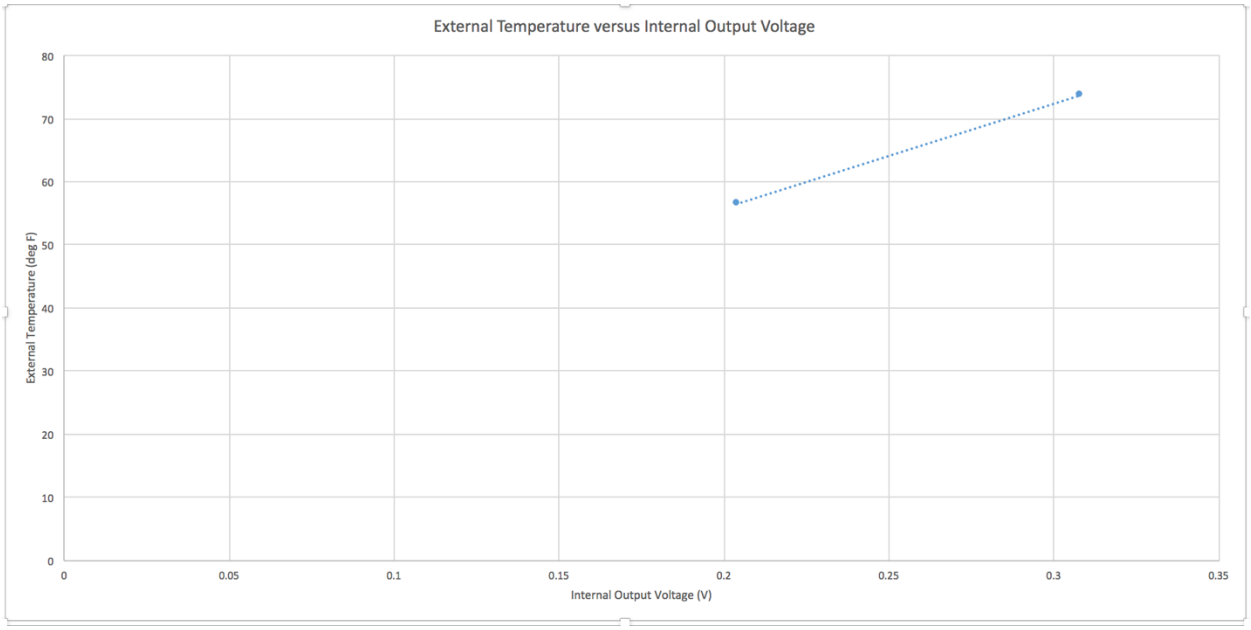
BME 580 Microcontrollers I

Step-by-step laboratory instructions:

1- Begin by simply displaying the integer value for the external temperature sensor (ETS) on the OLED Display. Let the ETS just hang in room air.
2- Update the temperature value once every second and display the new integer value. Watch this value for a minute or so to make sure it is stable, changing only a digit or two over time.
3- Now, while the circuit is running, cup the ETS in your hand tightly and watch the temperature rise. The integer value should begin to increase almost immediately.
4- If your ETS is working properly, write some code to convert and display the temperature using a floating point number (with proper units: °C or °F) on the next line of the OLED display.
5- Repeat the experiment above: first display the ETS at room temperature for a minute or two, then cup the ETS in your hand tightly to watch the temperature rise.
6- Now, add some source code to also read the value of the internal temperature indicator (ITI).
7- Simultaneously display the integer value for the ITI and both values (integer and floating point) for the ETS at room temperature. Watch both values for a minute or so at room temperature. Take note of the two integer values and the floating point value.
8- Now, cup the PCB and the ETS between your hands with the OLED hanging out so you can see it. Warm the ETS and the PCB with your breath between your cupped hands.
9- After a short while when the temperature has increased, try to stabilize it for a few seconds and take note of the two integer values and the floating point value.
10- NOTE: Sometimes when you use more than one ADC channel, the analog signals “leak” between analog channels when you switch between channels quickly. To minimize this you may need to allow more time for the analog signal to settle. Sometimes, after you change to a new analog channel the best thing to do is to take an analog value and just throw it away, then immediately take another value and keep the second one. This allows you to clear and reset the analog sampling capacitor, and allows time for the analog multiplexer to completely switch over to the new analog channel. So, if you find that your internal temperature indicator (ITI) data drifts up when you heat up the external temperature sensor, then you might need to try these additional strategies.
11- Use these two sets of values to calibrate the ITI so it gives the same temperature reading as the ETS when converted to a floating point value. If you want to, you can just store all of these temperature values to the SD Card to use in your calibration. You can plot the temperature data on a spreadsheet and pick two or more points where you think the temperature of the ETS and ITI have stabilized. Basically you can do the calibration very simply in your source code by making a linear calibration calculation based on at least two temperature points. The two numbers you need to calculate are the slope (also called the gain or calibration coefficient) and the offset. This will take a while, but basically this is what you need to do any time you want to calibrate a sensor: measure its output at two or more points and equate this to a reference standard. In this case you are using the ETS as your reference standard (not the best choice, but its handy). In a laboratory you would use a calibrated thermometer or the phase change of water under specified conditions to establish two fixed points for temperature, and you would be a lot more careful to make sure that the ITI and the ETS were stabilized at the same temperature before recording the two points. But the general calibration procedure is otherwise basically the same.
12- NOTE: if you can think of a more clever and stable way to do this two point temperature calibration then go ahead and try it.

BME 580 Microcontrollers I

Your calibration curve for the internal temperature indicator (ITI):



BME 580 Microcontrollers I

13- GENERAL COMMENTS: You can do pretty amazing things when you combine a microcontroller with sensors. To begin with, you can monitor and record the sensor output. If a measurement exceeds a threshold your microcontroller can take specific action, such as sounding an alarm (over/under temperature), turning on a motor or other device, cutting power to the system in case of an emergency, or even start to take more detailed data and store it during a crisis or accident for later analysis. But you can also do a lot of data manipulation and signal processing to greatly increase the quality of the raw signal from the sensor. For example, say the sensor is extremely stable, giving the same measurement every time, but that measurement has a constant offset, something like reading too high by 3.4 °C for example. With a microcontroller, you can adjust for errors of offset by adding or subtracting a constant. This process is called *calibration*. If the sensor is linear but the transfer function slope is inaccurate, you can also compensate for errors in the calibration constant or gain by multiplying (or dividing) the signal by the necessary correction factors. You can even use a microcontroller to "linearize" a non-linear sensor signal. This is very common practice in industrial controls and scientific instruments. Say, for example, the sensor generates a signal that is characterized by a non-linear arbitrary function. You can either develop a correction function to cancel out the non-linearity, or you can generate a "look-up table": when the sensor gives a reading of *x*, that means the pressure (or temperature or whatever) is *y*. Then you tabulate these values, store them in memory, and whenever you get a sensor value you just look up the corresponding result in the table. For very large tables one strategy is to use a smaller table and then use linear interpolation to calculate points between the values in the table. A process like this is sometimes called "piecewise linearization": breaking down a non-linear curve into a series of linear segments. You can also apply oversampling and filtering. It is usually very easy to take more samples than needed and apply simple filters (such as a rolling average), or complex filters, or even non-linear filters, such as the very popular and very powerful median filter for removing digital noise. Microcontrollers or microprocessors that are optimized for doing these functions are often called DSPs (digital signal processors):
https://en.wikipedia.org/wiki/Digital_signal_processor

14- Back to the lab exercises: write some code to record the temperature data to the SD card. Log the temperature data once every second. Record each temperature point as a floating point number, and separate the two data points at each time point (the ETS and ITI temperature values, one of each at each time point) into two columns using a comma as a separator.

15- CALCULATING THE AVERAGE TEMPERATURE: In your source code, define two floating point variables that you will use to sum up all of the temperature data, one FLOAT for each temperature sensor. Also define a 32-bit unsigned integer to count the number of time points.

16- At each time point, calculate the new average temperature for all data points collected, one average for the ETS, and another for the ITI temperature data.

17- At each time point, once every second, display the average temperatures (for all accumulated data) and the number of time points you have collected, on the OLED.

18- Check that the time point temperature data (not the average temperature data, which is only displayed, not logged) is properly logged on the SD Card before continuing.

19- Now you are ready to think about data logging temperature data…

BME 580 Microcontrollers I

Your source code using the internal and external temperature sensors:

```
#include <16F1829.H>
#device ADC=10                    // Preprocessor directive to define the size of the ADC return value
#BYTE FVRCON=getenv("SFR:FVRCON")  // An attempt to define an 8 bit variable that is stored at the memory address where the FVRCON special function register is located
// #include <stdlib.h>

#use delay (clock=8000000)    //  8 MHz, 0.5 us instruction cycle, this MUST come before the #include <oled_16F1829_i2c.h> to define delays in header file such as "delay_us(1)"
//   #use fast_io (A)

#use rs232(baud=9600, xmit=PIN_C5, rcv=PIN_C4, stream=COM_MEM)

#include <oled_16F1829_i2c_Edit.h>

#fuses WDT, INTRC_IO, PUT     // internal oscillator operation with RA6 and RA7 as digital I/O, ref 16F1829.h, enable Power Up Timer
#fuses NOMCLR                 // disable /MCLR pin
#fuses NOPROTECT              // disable code protection
// #fuses NOLVP

// Bob's pin definitions for PIC16F1829 for I2C communication with OLED are in oled_16F1829_i2c.h

#define led pin_a2          // BE SURE TO REDEFINE THE LED PIN TO MATCH YOUR CIRCUIT

unsigned int i = 0;
unsigned int j = 0;

float tempE;            // floating point variable for external temperature
float tempI;            // floating point variable for internal temperature
float AVG_tempE = 0;        // floating point variable for the average external temperature
float AVG_tempI = 0;        // floating point variable for the average internal temperature
float tempAVG_tempE;     // floating point variable for the average external temperature
float tempAVG_tempI;     // floating point variable for the average internal temperature
float v;                // floating point variable for "voltage"
float vE;               // floating point variable for "voltage" stored in EEPROM

// Ross' definitions for PIC16F1829 for ADC channels
#define TEMPERATURE_INDICATOR 29
#define ANALOG5 5

// Ross' definitions for PIC16F1829 for ADC setup and usage
#define ANALOG4 4           // define channel 4 of the ADC to be analog4
#define max_voltage 4.096   // define the maximum voltage to be Vss (This needs to be chagned if the voltage range changes)
#define num_states 1024     // Number of states for the given bit depth "ADC=8" (This needs to be changed if the bit depth changes

unsigned int16 ADCout;      // 16 bit integer value for storing ADC output from the potentiometer
unsigned int16 ADCoutE;     // 16 bit integer value for storing ADC output from EEPROM
unsigned int8 lower_byte;   // 8 bit integer value for storing lower 8 bits of ADCout
unsigned int8 upper_byte;   // 8 bit integer value for storing upper 8 bits of ADCout
unsigned int16 externalTemp;  // 16 bit integer value for storing the ADC output from the external temperature pin
unsigned int16 internalTemp;  // 16 bit integer value for storign the ADC output from the internal temperature pin
unsigned int16 garbage;     // 16 bit integer value for clearing the ADC between reads
unsigned int32 totalTime=0;  // 32 bit integer value for counting the total number of elapsed time points

////////////////////////////////////////
// SUBROUTINES

void init_ports(void) {

   SETUP_OSCILLATOR(OSC_8MHZ|OSC_INTRC);
   SETUP_WDT(WDT_8S);                       //  WDT set to 8 second period

   // Setup the ADC
   setup_adc(ADC_CLOCK_INTERNAL);           // ADC uses the internal clock
   setup_adc_ports(sAN5, VSS_FVR);          // ADC sets specific ports to ANALOG and specifies the range to be 0 V to FVR

   // Setup the reference voltage
   setup_vref(VREF_ON | VREF_ADC_2v048);    // FVR is turned on and set to ADC 2.048 V
   FVRCON = 227;                            // Attempt to ensure that the TSEN bit is 1
}

////////////////////////////////////////
// PIC12F1829 goes here at RESET

void main()
{
init_ports();               //  Initialize ports
restart_wdt();
delay_ms(50);               //  insert a short delay before starting the system up
```

BME 580 Microcontrollers I

```
fprintf(COM_MEM, "BMME 580");
fputs("", COM_MEM);  // Adds a reliable carriage return for OpenLog data streams
fprintf(COM_MEM, "Ross Rucho");
fputs("", COM_MEM);  // Adds a reliable carriage return for OpenLog data streams
fprintf(COM_MEM, "Data Logging from PIC16F1829 to SparkFun OpenLog");
fputs("", COM_MEM);  // Adds a reliable carriage return for OpenLog data streams

// the "bit bang" code below and in the *.h file to emulate I2C
// to run the OLED Display was developed by Kenny Donnelly

output_low(res);
  delay_ms(10);
  output_high(res);
  delay_ms(10);

initialise_screen();
  delay_ms(10);
  clear_screen();
  delay_ms(10);
  fill_screen();
  delay_ms(10);
  clear_screen();
  delay_ms(10);


CYCLE:

i = 0;  // counter for testing "moving text"
clear_screen();  // clear OLED screen

while(true)
{
  // Increment total time
  totalTime = totalTime + 1;

  // Change ADC channel to read from the internal temperature sensor
  set_adc_channel(TEMPERATURE_INDICATOR);  // ADC reads from channel 29
  delay_ms(10);
  garbage = read_adc();
  delay_ms(10);
  garbage = read_adc();
  delay_ms(10);

  internalTemp = read_adc();

  // Calculate analog internal temperature in fahrenheit
  tempI = ((73.671-56.455)/(0.304-0.204))*((internalTemp*(max_voltage/(num_states-1))) - 0.304) + 73.671;  // This calculation includes the linear interpolation from analog voltage to temperature
  AVG_tempI = AVG_tempI + tempI;
  tempAVG_tempI = AVG_tempI/totalTime;

  // Change ADC channel to read from channel 5
  set_adc_channel(ANALOG5);   // ADC reads from channel 5
  delay_ms(10);
  garbage = read_adc();
  delay_ms(10);
  garbage = read_adc();
  delay_ms(10);

  externalTemp = read_adc();

  // Calculate analog external temperature in fahrenheit
  tempE = (externalTemp*(max_voltage/(num_states-1)))*100;  // This calculation includes the conversion from analog voltage to temperature for the LM34DZ temperature sensor
  AVG_tempE = AVG_tempE + tempE;
  tempAVG_tempE = AVG_tempE/totalTime;

  // Change ADC channel to read from the internal temperature sensor
  // This is necessary to prevent crosstalk between ADC channels
  set_adc_channel(TEMPERATURE_INDICATOR);
  delay_ms(10);
  garbage = read_adc();
  delay_ms(10);
  garbage = read_adc();
  delay_ms(10);

  // Write the analog voltage to external memory
  fprintf(COM_MEM, "%2.3f,%2.3f", tempE, tempI);
  fputs("", COM_MEM);  // Adds a reliable carriage return for OpenLog data streams
```

```
  restart_wdt();

  oled_write_command(0xb0);
  oled_write_command(0x00);
  oled_write_command(0x10);

  // uncomment the following line to zoom in to text (top four lines of text only)
  oled_zoom();


  // the following command places the next character on line 0 (the 8 text lines are numbered 0 to 7)
  // and column 42 (there are 128 pixels from left to right on this 128 x 64 pixel display)
  // note that text character placement on the OLED display is therefore:
  //       128 columns (individual pixels) numbered 0 to 127
  //         8 lines, numbered 0 to 7, where each line is one character tall
  //           each character is 8 pixels tall
  //           8 x 8 = 64 pixels = the full display height
  // By placing the next character at this location the text "BMME580" is centered at the top
  // question: how many characters fit into the 128 characters across the OLED display

  oled_gotoxy(0,0);
  printf(oled_printchar, "External= %2.3f F", tempAVG_tempE);

  oled_gotoxy(1,0);
  printf(oled_printchar, "Internal= %2.3f F", tempAVG_tempI);

  oled_gotoxy(2,0);
  printf(oled_printchar, "Total time= %4Lu", totalTime);

  //oled_gotoxy(0,0);
  //printf(oled_printchar, "Stored Volts= %2.3f ", vE);  // Displays the last recorded voltage from the previous session

  //printf(oled_printchar, "V");

  //oled_gotoxy(1,0);
  //printf(oled_printchar, "ADC Output= %4Lu ", ADCout);  // Displays the digital representation of the voltage

  //oled_gotoxy(1,0);
  //printf(oled_printchar, "ADC Output= %4Lu ", ADCoutE);  // Displays the digital representation of the voltage from EEPROM

  //oled_gotoxy(2,0);
  //printf(oled_printchar, "New Volts= %2.3f ", v);  // see the PIC_C User Manual under "printf()" to format numbers in the display
  //printf(oled_printchar, "V");  // notice how the "V" displays directly after the last printed character

  //oled_gotoxy(2,0);
  //printf(oled_printchar,"temperature = %1.1f $F", t);

  // look in the oled_16F1829_i2c.h file to see why the "$" character actually
  // displays a the temperature degree symbol "∞"
  // answer: I "remapped" the pixels in the BYTE const TABLE5 for "$".
  // So I changed the ASCII Character Table for the OLED Display in the *.h file

  oled_gotoxy(3,0);

  for (j=0; j<i; j++) printf(oled_printchar," ");

  printf(oled_printchar,"Ross Rucho");


  OUTPUT_HIGH(LED);  // RA5 - turn ON green LED
  delay_ms(100);
  OUTPUT_LOW(LED);  // RA5 - turn OFF green LED
  delay_ms(100);

  i = i + 1;  // nudge test to the right one character

  if(i>22)
  {
    i = 0;  // prevent moving text from scrolling down to next row on OLED display
    clear_screen();  // clear OLED screen
  }

  delay_ms(800);
}

restart_wdt();

goto CYCLE;  //  cycle indefinitely until battery is removed or battery runs out of power
} // main
```

BME 580 Microcontrollers I

Run your circuit for a few minutes to generate a good set of data, then paste your data set (the resulting text file on the SD card) here:

73.671,24.780	73.671,56.489	73.271,57.178	72.871,57.178	72.470,57.178
73.671,75.790	73.671,57.178	73.271,57.178	72.871,56.489	72.470,56.489
73.671,56.489	73.671,57.178	73.271,56.489	72.871,56.489	72.470,56.489
73.671,57.178	73.671,56.489	73.271,57.178	72.871,57.178	72.470,57.178
73.671,57.178	73.671,56.489	72.871,57.178	72.871,57.178	72.470,56.489
73.671,56.489	73.671,56.489	73.271,57.178	72.871,56.489	72.871,57.178
73.671,56.489	73.671,57.178	73.271,57.178	72.871,57.178	72.470,75.100
73.671,57.178	73.671,56.489	72.871,57.178	72.470,57.178	72.470,77.168
73.671,57.178	73.671,57.178	72.871,57.178	72.470,57.178	72.470,56.489
73.671,57.178	73.671,57.178	72.871,56.489	72.470,56.489	72.470,57.178
73.671,56.489	73.671,57.178	72.871,56.489	72.871,57.178	72.470,57.178
73.671,56.489	73.271,56.489	72.871,57.178	72.470,90.265	72.470,57.178
73.671,56.489	73.271,56.489	72.871,57.178	72.470,56.489	72.470,56.489
73.671,56.489	73.271,55.800	72.871,57.178	72.871,57.178	72.470,56.489
73.671,57.178	73.271,57.867	72.871,57.178	72.470,57.178	72.470,57.178
73.671,56.489	73.271,57.178	72.871,90.265	72.470,56.489	72.470,56.489
73.671,57.178	73.271,57.867	72.871,57.178	72.470,57.178	72.470,57.178
73.671,57.178	73.271,57.178	72.871,56.489	72.470,57.178	72.470,56.489
73.671,57.178	73.271,57.178	72.871,57.178	72.470,57.178	72.470,56.489
73.671,56.489	73.271,90.265	72.871,57.178	72.470,57.178	72.470,56.489
73.671,56.489	73.271,57.178	72.871,56.489	72.871,57.178	72.470,56.489
73.671,75.790	73.271,57.178	72.871,56.489	72.470,57.178	72.470,56.489
73.671,75.790	73.271,57.178	72.871,57.178	72.470,57.178	72.470,56.489
73.671,89.576	73.271,56.489	72.871,57.178	72.470,57.178	72.470,57.178
73.671,56.489	72.871,56.489	72.871,57.178	72.470,57.178	72.470,56.489
73.671,57.867	73.271,57.178	72.871,57.178	72.470,56.489	72.470,56.489
73.671,57.178	73.271,57.178	72.871,57.178	72.470,56.489	72.470,57.178

BME 580 Microcontrollers I

DATA LOGGING

Now, you will build a temperature data logger. Most data loggers are compact units that can be placed in remote locations, usually not plugged in to the wall. This usually means that a data logger has to be battery powered. That single consideration, batteries, is usually one of the major drivers of data logger design.

CONTINUED Step-by-step laboratory instructions:

1- First, you need to know how much power your data logger uses during normal operation. To do this you will need to measure the current (milliamps) that the circuit is drawing while it is operating. Use your bench meters and your digital volt meter to measure the current drawn by your circuit while it is logging data.

According to the readings from the benchtop power supplies, my circuit appears to draw approximately 20 mA of current during operation.

2- Next, you need to learn about batteries. Batteries have a rated capacity, usually stated in units of mAh (milli-Amp hours). You can read very short articles on battery energy capacity at:

https://en.wikipedia.org/wiki/Ampere-hour

and

https://en.wikipedia.org/wiki/Battery_%28electricity%29#Capacity_and_discharge

The implicit assumption is that a battery is a constant voltage source (approximately), so one can assume the fundamental units of battery *energy* capacity are VAh (volt-amp-hours), which have units of power (VA) multiplied by time (h) which yields units of energy (power · time = energy). It turns out that for batteries of the same internal chemistry, the amount of energy they contain is pretty much linearly related to their overall volume: larger batteries contain more available energy.

So, you can approximate how long a device will operate on a set of batteries by knowing the current drain of the device (mA) and the capacity of the batteries (mAh). If you divide capacity by current drain you end up with units of time (hours) which tells you approximately how long your device will operate continuously on a battery of a known capacity assuming the battery starts with a full charge. A lot of things can cause this approximation to vary, but in general it turns out that the more slowly you drain a battery the more energy it can deliver. This is known as Peukert's Law:

https://en.wikipedia.org/wiki/Peukert's_law

Since data logging systems are designed to be low drain systems (to maximize battery life), the simple approximation of battery life based upon measured drain rate and battery capacity is usually conservatively accurate; the battery may do better than estimated. Of course in critical applications you can always test this empirically, but for design purposes this quick calculation is usually reliable. The biggest factor turns out to be variability in manufacturing: batteries are quite crude chemical reactors and they are subject to variability that I have measured to be about +/- 20%, even for batteries from the very same package fresh from the manufacturer. So, using the simple calculation, estimate how long your circuit will run if powered by AA batteries with 2000 mAh capacity:

My circuit will operate for approximately: $\frac{2000mAh}{20mA}$ = 100 hours.

BME 580 Microcontrollers I

3- Decide whether or not you want to try to build a real data logger or just simulate one on the lab bench. For the real data logger you will need AA batteries and a battery holder. Three AA batteries would be about perfect since you would not need to regulate the voltage (it would be about 4.5 volts total). If you use 4 or more AA batteries in series, the voltage will exceed the maximum for your microcontroller, so you will likely destroy your circuit if you are not careful. In that case you would need to add a voltage regulator to your circuit, which will stabilize the voltage but will also drain more current, thus shortening the life of your batteries. So if you decide to try to use batteries to build a real data logger, consider using 3 AA batteries and the internal FVR to assure that the reference voltage does not change over the time period you are logging data.
4- Before you take a lot of data, make sure that the data format you log to your SD card is in a format that can readily be exported into a spread sheet to give you columns of numbers, not just text or gibberish. A few seconds of data logging will verify this for you. Usually you just want comma-delimited numbers represented as ASCII character strings, each column representing a different sensor or conversion, each row representing one time point of data.
5- Once you are sure your system is logging readable temperature data, set your data logger up on a benchtop or better yet if you are using batteries put it somewhere thermally interesting, like inside your car on the dashboard, or in a sunny window area, or in a refrigerator. You could get very interesting physiological data: just wrap it up and put it in a pocket, wear it all day, then sleep on it all night to track your diurnal skin temperature cycle. Be creative but be careful of course. Also keep in mind the temperature range limits of your sensor, so logging the temperature in a freezer or in an oven might not be such a great Idea (Steve 😐 would not approve). Log the temperature every second for at least 24 hours. You should end up with about 86,400 time points of data on your SD card.
6- BEFORE STOPPING YOUR DATA LOGGER: Note the average temperatures and the total number of time points being displayed.
7- Export the data to a spreadsheet as comma-delimited numerical values.
8- USE THE SPREADSHEET to calculate the average temperature values for each sensor for all accumulated data points.
9- Is there a difference between the average values displayed on the OLED (calculated internally on the microcontroller) versus the spread sheet?

There does not appear to be a significant difference between the average values calculated on the microcontroller versus the average values calculated from a spreadsheet.

DATA SUMMARY:

89196 data points

External = 63.651 °F ; Internal = 50.090 °F

External = 63.653 °F ; Internal = 50.094 °F

10- Use the spreadsheet temperature data to generate a line graph showing both the ETS and ITI temperatures vs time on the same graph. Make sure each axis is labeled properly.

BME 580 Microcontrollers I

Photograph of your data logging system in action (either battery powered or on the bench):

BME 580 Microcontrollers I

Import your text data on the SD card into a spreadsheet such as Excel. Plot temperature vs time for both the ITI and ETS: