BME 580 Microcontrollers I

Lab 4: ADC – Analog-to-Digital Conversion *Ross Rucho*

In this lab you will continue to build upon previous labs using the OLED display. You will learn to use the on-board analog-to-digital converter (ADC) in the PIC16F1829. Many microcontrollers have internal analog-to-digital converters. Analog-to-digital conversion is among the most commonly used functions in embedded systems.

Before you begin this lab please read the documents listed below. These will help you understand what you are doing in the lab, and this material will be covered on the final examination at the end of the term.

Reading assignment:

Overview of Analog-to-digital conversion:

https://en.wikipedia.org/wiki/Analog-to-digital_converter

Overview of potentiometers:

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

The CCS C User Manual is available on the lab workstations, but a more recent version of the CCS C Compiler Manual is available from my dropbox folder:

https://www.dropbox.com/s/90kctnurjew9lwf/PCWH%20May-2014.pdf?dl=0

Once you have saved a copy of the CCS C Manual on your laptop, read the following sections:

"ADC" READ this section thoroughly

Also read these sections: "set_adc_channel" "setup_adc(mode)" "setup_adc_ports()"

BME 580 Microcontrollers I

OTHER DOCUMENTS AND FILES YOU WILL NEED FOR THIS LAB:

CCS C User Manual A copy is on the lab computer and you should save a copy for yourself.

16F1829.h This is the header file that PICC compiler uses to understand how to compile code specifically for the PIC16F1829 microcontroller. Every microcontroller is different: the input/output pins are at different memory addresses, internal functions are configured and used differently, etc. So every time you write source code for a microcontroller you need to include the appropriate device header file to tell the compiler the details about the hardware configuration of the microcontroller you are programming. The device header file is also a very useful document when writing source code because it lists every special function that can be called in source code for the specific microcontroller that you are using. It will show you the basic syntax for each special command. You should copy and save this header file as a text file so that you can refer to it easily when you are writing source code because it contains all of the internal functions that your microcontroller can use, such as setup_adc(), setup_adc_ports(), etc.

A copy of this header file is on the lab workstation. Find and save a copy as a *.txt file on your laptop.

You will also need to be sure you have the datasheet for the PIC16F1829 microcontroller. If you do not have it already, you can find it at this link:

http://ww1.microchip.com/downloads/en/DeviceDoc/40001440E.pdf

or from: https://www.dropbox.com/s/qsfxvfyr4wbb81y/DATASHEET_PIC16F1829.pdf?dl=0

QUESTIONS FROM THE READINGS:

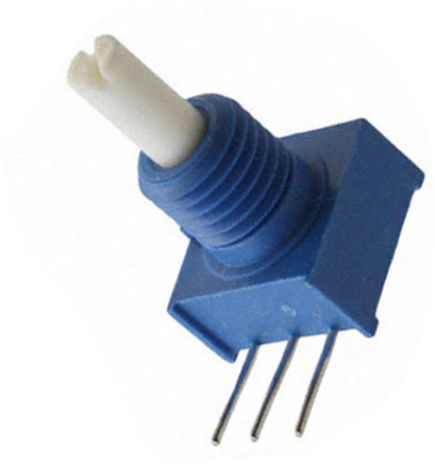
1- How many ADC channels are available on the PIC16F1829? Check the datasheet to find out.
There are up to 15 ADC channels available on the PIC16F1829. The available channels are labeled: AN<11:0>, Temperature Indicator, DAC Output, and FVR Output.

2- What pin numbers on the PIC16F1829 allow connection to an ADC channel?
All pins that are labeled with an ANX, that is, anything from AN0 through AN11. These labeled pins correspond with the following pin numbers: 19, 18, 17, 3, 16, 15, 14, 7, 8, 9, 13, 12.

3- What is the dynamic range of the ADCs on the PIC16F1829 (that is, how many bits: 8, 10, 12, 16?)
The dynamic range of the ADC's on the PIC16F1829 is either 8 or 10 bits.

BME 580 Microcontrollers I

NEW COMPONENTS TO ADD TO YOUR PCB

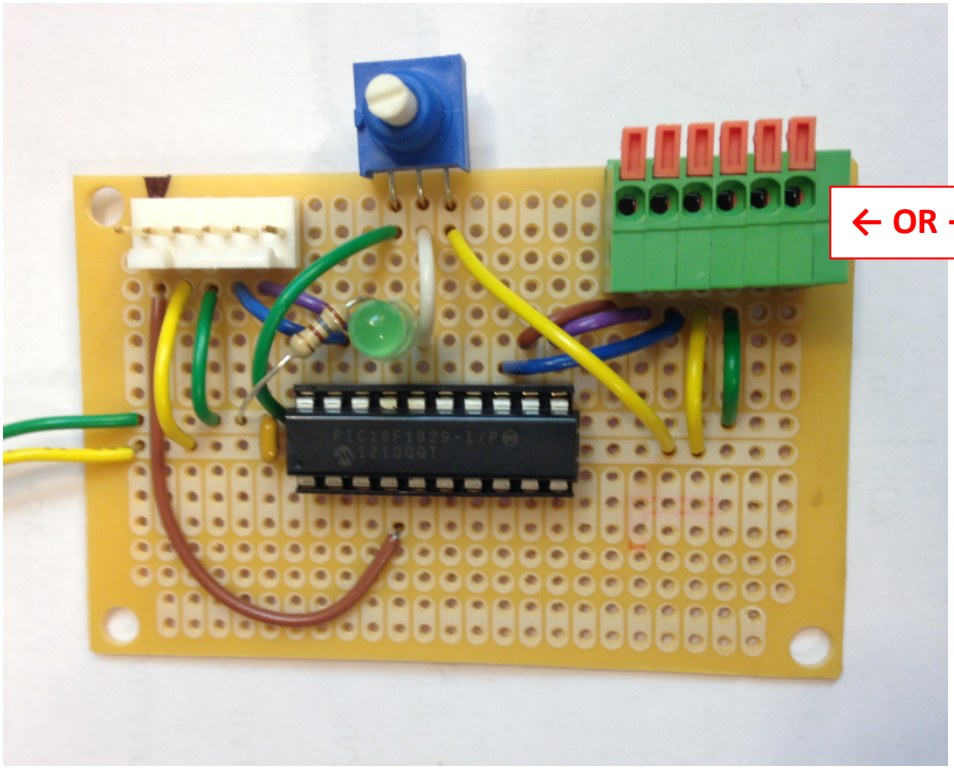


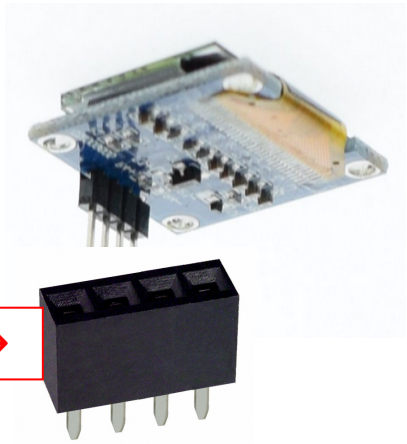
Use the volt meter in your toolkit to measure the resistance of the potentiometer. You do this by measuring the resistance between the outer two pins. Generally you should measure resistance of components before you solder them into the circuit. The resistance to the middle pin depends upon the rotation of the potentiometer shaft, but the resistance between the outer two pins remains constant. It should be somewhere in the range of 1 K to 10 K Ohms.

Solder the potentiometer to your PCB as shown below.

Connect the center pin on the potentiometer (called the "wiper") to pin #16 of the microcontroller (RC0). In the photograph below this is done using a white wire. Then connect the other two pins of the potentiometer to +5V (yellow wire) and ground (green wire), as shown.

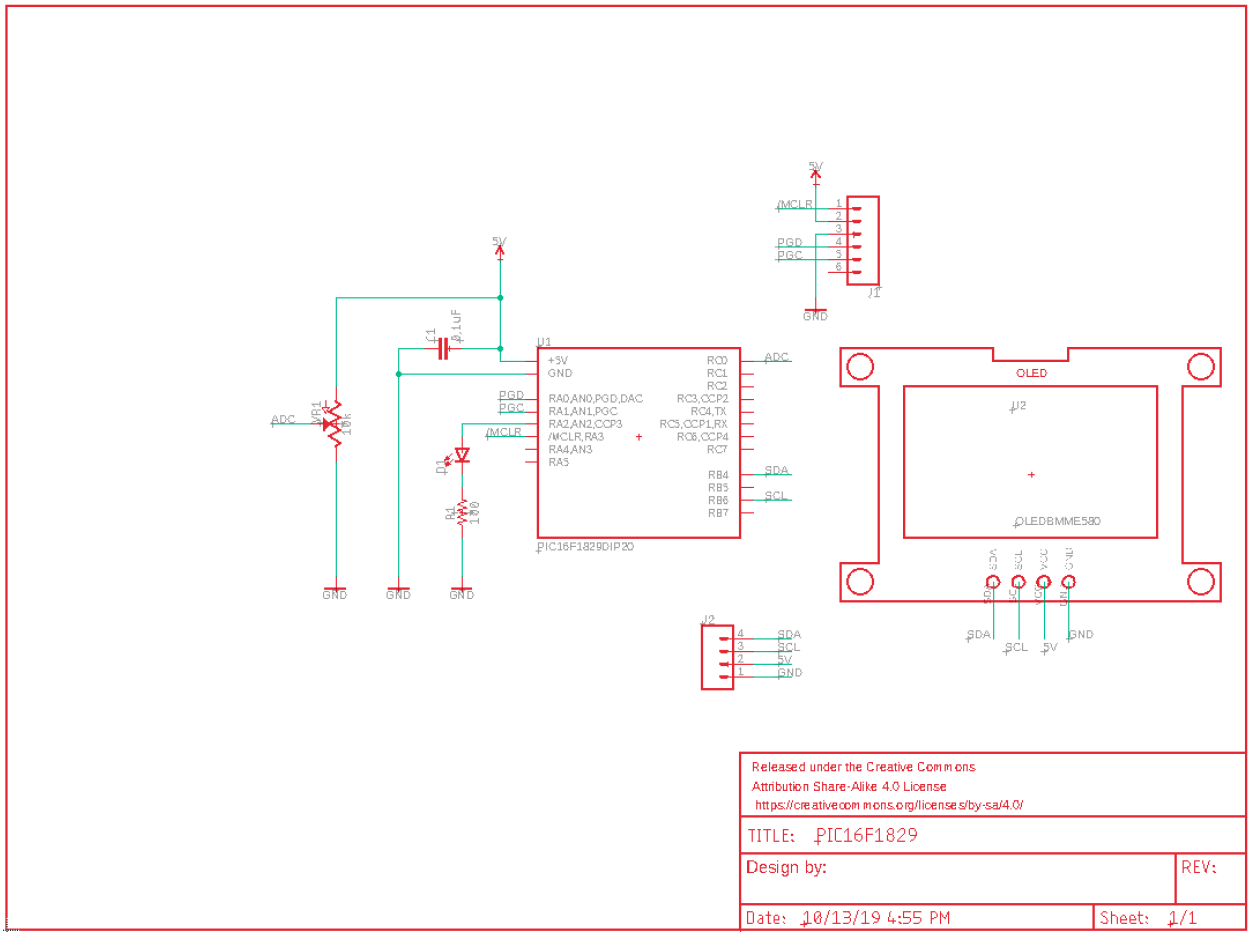
Then modify your EagleCAD schematic to include the new potentiometer.





BME 580 Microcontrollers I

Your EagleCAD version of the modified circuit schematic with potentiometer:



TITLE: PIC16F1829

Design by:

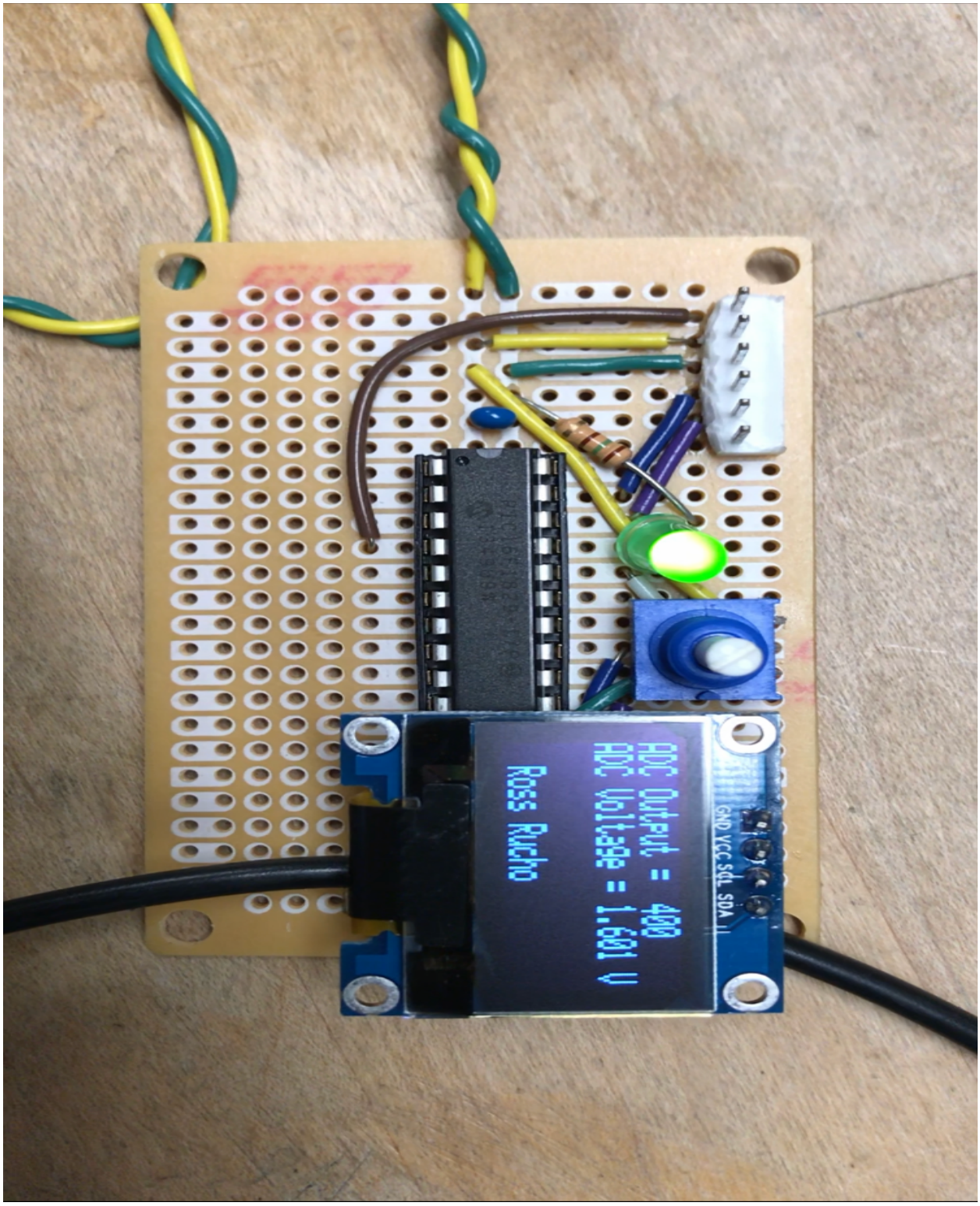
REV:

Date: 10/13/19 4:55 PM

Sheet: 1/1

BME 580 Microcontrollers I

Photograph of your completed circuit:



BME 580 Microcontrollers I

FIRMWARE HINTS for setting up and using the ADC

You will be doing a lot of setting up the ADC in this lab, and the bits of sample code you need can all be found in the CCS C User Manual plus the command and constant syntax in the 16F1829.h header file. To help point you in the right direction see the sample code below, with comments on each line for explanation:

Near the very top of your code you need to define the number of bits returned by the internal ADC:

```
#DEVICE ADC=8    // this is your only option if the internal ADC is limited to 8-bits
#DEVICE ADC=10   // use this if you want the full 10 bits available from the internal ADC
```

Then, down in your source code where you are setting up I/O ports, set up the fixed voltage reference:
Note that this sets up the FVR, but you have not explicitly used it yet. It is just ready to use when needed.

```
setup_vref(VREF_ADC_4v096);   // enable the internal fixed voltage reference (FVR = 4.096 volts)
```

Then setup the ADC clock. Look at the 16F1829.h file for all of the options for this command:

```
setup_adc(ADC_CLOCK_INTERNAL);     // this makes the ADC use the internal clock
```

Then setup the ADC ports. Look at the 16F1829.h file for all of the options for this command:
NOTE, you should only use one line for this command, so choose one from the four examples below.
Also note that the last example shows how to use external voltage references for the internal ADC.

```
setup_adc_ports(sAN4);                  // setup the ADC to function on one pin, RC0, analog channel 4
setup_adc_ports(sAN4|sAN7|sAN11); // setup the ADC to use three channels: #4, #7, and #11
setup_adc_ports(sAN4, vss_fvr);         // setup ADC channel #4 and the voltage range: 0V to FVR
setup_adc_ports(sAN4|sAN5, vref_vref);// setup ADC channels 4 & 5, voltage range Vref-low to Vref-high
```

You can mix and match constants in the setup_adc_ports() command to get what you need. Look at the 16F1829.h file for all of the options for this command. Look in the CCS C User Manual for sample code.

Before you start reading analog voltages on the ADC, first select the ADC channel then add a delay:

```
set_adc_channel(4);  // start by setting the ADC channel.  You need to do this again if you change the channel.
delay_us(10);          // a short delay is required after every time you change ADC channels
```

Then, read the analog voltage on the ADC and assign the returned value to an appropriate variable:

```
volts_adc=read_adc(); // this assumes you have declared the variable volts_adc as a long unsigned int
                      // or int16 or some other appropriate type for the value returned from the ADC
```

There are other things you can do to control the ADC, but these examples should get you started.

BME 580 Microcontrollers I

Step-by-step instructions for this lab

Begin by reviewing the source code for the firmware from Lab #3. Then follow these steps:

1- Before you change your firmware, check to make sure your OLED display is still working properly using the firmware you installed for Lab #3.
2- Use the DATASHEET for the PIC16F1829 to find which analog channel is connected to Pin #16 (RC0)
3- Make sure you have looked at the CCS User Manual and have read the ADC section.
4- The CCS C User Manual includes snippets of example code, and the PICC C compiler folder on the lab workstation also has many example files. You can also search the internet to find example code for PICC. You should search around a bit to figure out how to write the code for yourself.
5- Modify the source code from Lab #3 to setup the ADC channel and to make analog-to-digital conversions. You will need to reference both the CCS C User Manual as well as the 16F1829.h (header) file to get the syntax of the commands correct.
6- HINT: be careful about what types of variables you use to store the results from the ADC. Initially the conversion will yield an unsigned 8-bit value (0-255), but later when you change to 10-bits the returned value will be too large for an 8-bit variable. So, I suggest you use unsigned 16-bit variables for ADC values.
7- While looking at the 16F1829.h file, find the command you need to set the ADC voltage references to from +5V supply to ground (Vdd to Vss). What this means is that you are setting the scale of the ADC conversion so that it corresponds to a set range of voltages: supply voltage to ground. Note that you can also select internal fixed voltage references (FVRs) with the ADC.
8- Make sure the ADC is initially read as an 8-bit value.
9- Add source code including simple “while” or “for” loops to run the ADC to measure the potentiometer voltage about once every second.
10- The result will be an 8-bit integer.
11- Display the ADC result on the OLED display. It will be an integer number from 0 to 255.
12- Calculate the conversion of the ADC value into a real voltage (0 to +5.0 volts). This will be a floating point number.
13- Display the real voltage (0.000 to 5.000) on the OLED, one line below the ADC value (0 to 255)
14- Turn the shaft of the potentiometer to make sure you can get voltage readings all the way from 0.000V to 5.000V. You may find that you cannot quite get the readings to go all the way in one or both directions. You have just built a simple voltmeter.
15- Verify that your power supply is set to exactly 5.0V output. If the power supply is not accurate, then your voltage readings will not be accurate.
16- Now, change the source code to read the full 10 bits of the ADC. To do this you will need to make sure that you define the device as 10 bits rather than 8 bits, and you will need to be sure to use variables large enough to hold at least 10 bits. See the commands discussed above.
17- Display the new 10-bit values on the OLED. You should verify that you get the full dynamic range of the ADC: integer values from 0 to 1023 as you turn the shaft of the potentiometer.
18- Change your floating point conversion so that the real voltages read from 0.000 to 5.000 volts
19- Set the power supply to as close to exactly 5.000 volts as you can, then adjust the potentiometer shaft and then use a volt meter to read the voltage on the ADC channel #4 (pin #16) and build a table:

BME 580 Microcontrollers I

TABLE 1: ADC vs volt meter using power supply rails as ADC reference voltages

Power supply set to 5.00V

Digital value	OLED -Displayed voltage	Volt meter reading
0	*0.000 V*	*0.0 mV*
200	*0.977 V*	*0.987 V*
400	*1.955 V*	*1.976 V*
600	*2.932 V*	*2.93 V*
800	*3.910 V*	*3.91 V*
1023	*5.000 V*	*5.02 V*

20- Adjust your power supply to about 4.50V instead of 5 volts, BUT DO NOT CHANGE YOUR CODE (the point is to see what happens when power supply voltage changes) and build a new table:

TABLE 2: ADC vs volt meter using power supply rails as ADC reference voltages

Power supply set to 4.50V

Digital value	OLED -Displayed voltage	Volt meter reading
0	*0.000 V*	*0.0 mV*
200	*0.977 V*	*0.885 V*
400	*1.955 V*	*1.776 V*
600	*2.932 V*	*2.63 V*
800	*3.910 V*	*3.52 V*
1023	*5.000 V*	*4.52 V*

21- Now, find out about Fixed Voltage References (FVR). Read about FVRs in the following documents:

a. https://en.wikipedia.org/wiki/Voltage_reference
b. The DATASHEET for the PIC16F1829
c. The CCS Compiler User Manual
d. Most importantly: the 16F1829.h header file

BME 580 Microcontrollers I

22- Modify your source code to reconfigure your ADC to use the internal 4.096V FVR instead of the power supply rail, which is about 5.0V but can change if the power supply is altered, or if other electronic loads in your circuit pull the power rail down a bit, or in the presence of other transient voltage changes.

23- Repeat the measurements for the two tables above, but using the 4.096 volt fixed voltage ref.

24- First, set the power supply to as close to exactly 5.000 volts as you can, then adjust the potentiometer shaft and then use a volt meter to read the voltage on the ADC pin (analog channel 4 on pin #16) and build a table:

TABLE 3: ADC vs volt meter using 4.096V FVR as the ADC reference voltage

Power supply set to 5.00V

Digital value	OLED -Displayed voltage	Volt meter reading
0	*0.000 V*	*0.0 mV*
200	*0.800 V*	*0.805 V*
400	*1.601 V*	*1.58 V*
600	*2.402 V*	*2.38 V*
800	*3.203 V*	*3.19 V*
1023	*4.096 V*	*4.09 – 5.04 V*

25- Then, adjust your power supply to about 4.50V instead of 5 volts BUT DO NOT CHANGE YOUR CODE (the point is to see what happens when power supply voltage changes), and build a new table:

TABLE 4: ADC vs volt meter using 4.096V FVR as the ADC reference voltage

Power supply set to 4.50V

Digital value	OLED -Displayed voltage	Volt meter reading
0	*0.000 V*	*0.1 mV*
200	*0.800 V*	*0.803 V*
400	*1.601 V*	*1.58 V*
600	*2.402 V*	*2.38 V*
800	*3.203 V*	*3.19 V*
1023	*4.096 V*	*4.09 – 4.49 V*

BME 580 Microcontrollers I

Your source code using the ADC with 4.096V FVR:

```
#include <16F1829.H>
#device ADC=10               // Preprocessor directive to define the size of the ADC return value
// #include <stdlib.h>


//#device adc=8

#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)         //

#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 t = 0;                // floating point variable for "temperature"
float v = 0;                // floating point variable for "voltage"

// 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

////////////////////////////////////////
// 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(sAN4, VSS_FVR);          // ADC sets all ports to ANALOG and specifies the range to be 0 V to FVR
   set_adc_channel(analog4);                // ADC reads from channel analog4
   delay_us(10);                            // A short delay is required after every time you change ADC channels

   // Setup the reference voltage
   setup_vref(VREF_ON | VREF_ADC_4v096);    // FVR is turned on and set to ADC 4.096 V
}



////////////////////////////////////////
// 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


//  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);
```

```
   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){

      // Control ADC to only read a new value at about 1 Hz
       ADCout = read_adc();
       v = ADCout * (max_voltage / (num_states-1) ); // Calculate the analog voltage
       delay_ms(800);



      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,"ADC Output = %4Lu ", ADCout);  // Displays the digital representation of the voltage


   oled_gotoxy(1,0);
 printf(oled_printchar,"ADC Voltage = %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);

   t = t + 1.2;      //  increment the "temperature" variable
   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
   }
 }
      restart_wdt();

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

BME 580 Microcontrollers I

Interfacing microcontroller ADCs with the real world

The internal ADCs of microcontrollers work pretty well when measuring voltages that fall between the power rails, typically 0V to 5V, but what if you need to measure a voltage outside of this range?

In general you will need to add some analog circuitry if you want to work with real world signals. Using your electronics skills and EagleCAD, design an analog circuit that you could use to convert a generic analog signal in the range of -10V to +10V into a signal that can be read by the internal ADC of your microcontroller, and also using the internal 4.096V FVR so that your readings are not so sensitive to the power supply voltage. Don't build the physical circuit, just design a schematic of it using EagleCAD.

The use of analog circuitry to prepare an analog signal for conversion is sometimes called a "front end", or a "pre-amplifier" if it is a low-level analog signal, or sometimes it is called a "signal conditioner". Biopotential signals tend to require a lot of front-end amplification, buffering, and voltage offsetting before you can send them to a microcontroller for analog-to-digital conversion.

You will probably need to create a new op-amp component library for EagleCAD, and then you need to add this analog conditioning circuit to a new schematic. But you don't need to build this circuit. Just design it in EagleCAD. Just design the schematic, not the PCB layout.

Your EagleCAD version of the circuit schematic with analog signal conditioning: