

ECE 218

EXERCISE 3. ANALOG TO DIGITAL CONVERSION

This exercise will introduce you to the PIC24 ADC and some of the functions in the textbook library. You will interface a voltage divider circuit, implemented with a potentiometer, to an analog input and write a program to read the value and convert it to a voltage value. You will gain more experience with the debugger, and if there is time, you will write a function to convert the real voltage value to an ASCII code.

SET UP HARDWARE

We will apply a fixed voltage to pin AN1 (pin 3 of the PIC24 chip) using a 10K potentiometer connected as a voltage divider, as shown in Figure 1. We will use the V_{DD} (3.3V) and GND (0V) signals that are available on the Microstick II board. The GND signal is available from pin 8, and the 3.3V signal is available from the header on the Microstick II board that we soldered on.

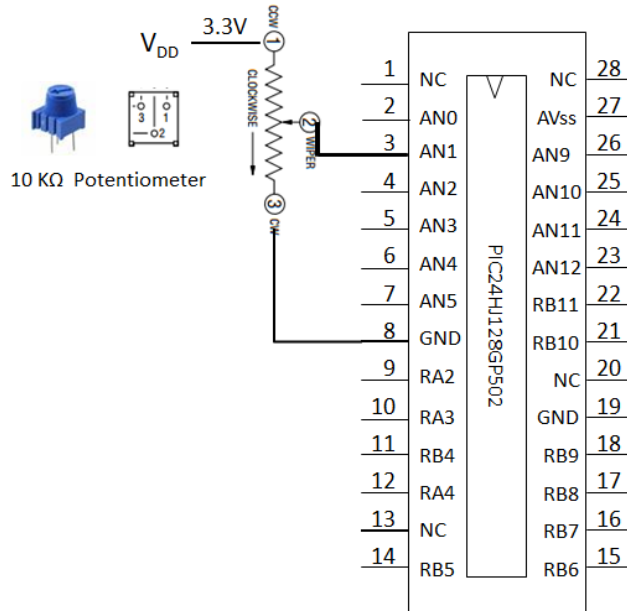


Figure 1. Voltage Divider Circuit Connected to PIC Analog Input 0

Once the voltage divider circuit is constructed on the breadboard, measure the voltage between the wiper node and GND to verify that it varies between the values 3.3V and 0. Remember that the maximum range of the ADC on the Microstick II board is GND(0V) – V_{DD} (3.3V).

Now connect the voltage divider circuit output (the wiper node) to the analog pin AN1 on the Microstick II.

SOFTWARE SETUP – CREATE NEW PROJECT

Using the steps from the previous 2 exercises, start a new project in the MPLAB X IDE. Name the project Ex3_ADC. Be sure to do all the steps to add the textbook library.

Open the `main_template.c` file and use the following guidelines and the prep slides on Nexus to write your program.

The only necessary code before the main program for this exercise is the inclusion of the textbook library that is already given in the template.

In the main program, we will declare two local variables. One of type `uint16_t` to hold the result of the A/D conversion, and one of type `float` to hold the corresponding voltage value you will calculate.

In addition to the `configClock()` function that is included in the template, there are two configurations needed to set up pin 3 (RA1) to be analog, and to configure the ADC for manual conversion using channel 0, using pin RA1, a 31 cycle hold time, and 10-bit conversion.


In the main program while loop there are just 3 things to do.


1. Use the `convertADC1()` function to do a conversion and assign it to the `u16` variable.
2. Convert the digital value to the voltage it corresponds to.
3. Delay for 100 ms. (arbitrary delay between conversions)

RUN PROGRAM

Measure and note the value of the voltage on AN1 at the current position of the potentiometer.

1.74 V _____

Build the project and download to the board in **debug** mode .

Pause the program, , and set a breakpoint at the `DELAY_MS` function. Run the code to the delay function breakpoint and observe the local variables by going to the Variables tab in the status window. It should look something like this:

Variables x					Call Stack	Breakpoints	Output
Name	Type	Address	Value				
<Enter new watch>							
f_voltage	float	0x9B2	1.65				
u16_adcValue	uint16_t	0x9B0	0x0200				

The Address is where your variables are stored in memory, and the Value is the current value of the variable. You can right click on the value to display in a different format if you wish.

Run the code to the breakpoint and check the value of the `adcvalue` and `volts` variables. **Does it match the expected value for your input?_____ If not, how much is it off by?_____**

Rotate the potentiometer and check that your variables are changing as expected. **Write down the measured value at the pin (voltage(V)), the ADC value, and the calculated voltage value (volts) for 8 different potentiometer settings, somewhat evenly spaced from 0 to 3.3V, to show the linear relationship.**

<i>voltage (V)</i>	<i>adcvalue</i>	<i>volts</i>	<i>voltage(V)</i>	<i>adcvalue</i>	<i>volts</i>

The main part of the exercise is complete when you have filled in this table, documented your code, and uploaded it to Nexus.

CHALLENGE

In the first project you will need to convert a real number to the ASCII character codes for each digit or decimal point. For example:

Real number: 1.63

Characters: '1', '.', '6', '3'

ASCII code: 0x31, 0x2E, 0x36, 0x33 [note that for the digits $n = 0-9$, the ascii code is $0x30 + n$]

This process takes several lines of code, so it is a good idea to encapsulate the code in a function. Define the function in the "GLOBAL VARIABLE AND FUNCTION DEFINITIONS" part of the template. Below is one algorithm you can use. If there is time, write the function and call it from your main program, and verify it works by using the debugger. Note that in the Variables window, only local variables are shown, but you can double-click on <enter new watch> to add your global variable to display the array of ASCII characters.

One algorithm for converting real numbers with 1 integer digit and 2 fractional digits to ASCII:

1. Declare a global character array of size 4 in the main program to hold result, including decimal point.
2. In the function, define an unsigned 16-bit integer, Nint
3. Initialize the 4 character array elements to 0x30, 0x2E, 0x30, 0x30 (ASCII for 0.00).
4. Multiply the real number by 100 to convert to an integer, Nint.
 - a. use a loop that counts the number of 100's in Nint (using subtraction) to find the unit digit, incrementing the 0th place of the character array each time.
 - b. use a loop that counts the number of 10's in the remaining Nint (using subtraction) to find the tenth's digit, incrementing in 2nd place of character array each time.
 - c. the remaining Nint is the hundredth's digit, so convert to ASCII by adding it to the 3rd place of character array.

Here is an example of the results of the algorithm steps above on the example real number 2.15.

1. chararr[4]
2. Nint
3. chararr: 0x30, 0x2E, 0x30, 0x30
4. Nint: 215
 - a. chararr[0]: 0x32
 - b. chararr[2]: 0x31
 - c. chararr[3]: 0x35