In this laboratory we will build a rather fancy “digital voltmeter.” It will sample voltages and report them via a serial interface on the computer via a terminal emulator (hyperterminal.)

We’ll be using the ADC and the UART. We’ll be operating the ADC at its maximum speed. We’ll be operating the UART at 115,200 baud. To provide a connection between the microcontroller and the computer, you’ll need to use a special serial-to-USB converter, supplied in the lab.

Your job is to build an application which will sample the analog input (from AN0) and keep a running average of the last 512 samples and their sample standard deviation (the square root of the sample variance). You will output the decimal representation of the voltage to the host computer using the UART. To drive the analog signal, connect a 10K potentiometer between power and ground, with the “sweeper” connected to the AN0. You should sample the analog input as fast as possible, but (because of the serial interface) you can’t report the results that fast (see below).

The results should be printed as: ”X.XXXXV xxx.xmV” where the first number is the voltage and the secon number is the standard deviation (in millivolts). Note that, because we are taking samples in groups of 512, the resolution of our measurement would be .006295 millivolt. Our actual accuracy won’t be that good, of course – but it will be on the order of .1 mV, we hope.

After each result is printed, print a carriage return (hex 0x0D) but **no** line feed. (This means that the terminal will just display the current value at any instant. The next value will overwrite the last value.)

You should set up the ATD to use Vdd and ground as the reference voltages. Please note that the documentation is a little confusing about the precise timing for the ADC converter in the PIC24FJ64GA002. You should use the timer3 conversion trigger method, and assume a minimum sample period (including the acquisition time, the conversion time – 12 T_{ad} – and overhead) of 2.00 microseconds to 2.125 microseconds. That means it will take from 1.024 to 1.088 milliseconds.

We’re going to send a total of 16 characters for each result, which will take 1.38 milliseconds.

Here are the specifications:

1. The display must be as smooth as hyperterminal can make it. No pausing, jerking, or other weird displays of uneven operation.

2. The result you display must always represent the latest values of the voltage you could get. You are not allowed to grab 512 samples and then wait around, not sampling, until you can deliver them. In other words, what we want, at any time the the value is printed, is that it represents the “latest and most accurate” value we could get.

There are four tasks you have to accomplish: acquire, calculate and convert, and print. You can split these tasks among the various threads in your code, assuming you are using interrupts (and, if you are not, you might consider using them so you can have four threads to split the job among). You can do this any way you like, but some ways are going to be easier and work better. Give it some thought before starting to write your code.

To get credit for this exercise, demonstrate your working model to the TA.

Turn in answers to the following questions:

1. What is the resolution of your DVM?
2. What is the accuracy of your DVM?
3. What is the sampling rate of your DVM? (After averaging.)
4. What proportion of the time in each iteration does your application spend in each of the following tasks?
   (a) Handling the ADC (waiting for it, reading it, setting bits, whatever).
   (b) Handling the UART (etc.)
   (c) Computing the average and standard deviation.
   (d) Converting the value to ASCII.

How to attach the serial-to-USB adapter

The serial-to-USB adapter is a little board with a USB cable on one side and an IDC (Insulation Displacement Connector) on the other side, with a ribbon cable connected to another IDC connector on the other end. You connect the free end of the ribbon cable to your board. It uses three connections: Ground, TX, and RX (although we’re not going to use one of them.) The “RX” in this case is the one from the device on the little board, and that needs to connect to the “TX” on your chip, RP3, also known as pin 7. (You’ll need to make that happen with PPS magic.) The connector has 4 pins (OK, 8, but 4 are just duplicates of the other 4). They are numbered 1, 2, 3, 4 on the little board, from left to right as you look at the board with the connector toward you, and the chip facing up. The four pins are 1 - Unused, 2 - GND, 3 - TX, and 4 - RX.

You want to hook up pin 2 to your circuit GND and pin 4 to your TX (RP3). Note that the ribbon cables and connectors don’t come in the right size, but ones with 10 pins are readily available, so we’re using ones with 5 connectors in a row (and two rows, but we only use one row). If you look at the way they’re connected, you’ll notice that the fifth pair is just hanging over the edge.

The little connectors also have a little mark, shaped like a little triangle, which indicates pin 1, and the cable has a red stripe on the side that connects to pin 1. That is going to make it (we earnestly hope) very hard to hook these up backward, because these are not fool-proof. Be careful, double and triple check your circuit before applying power.

Connect your circuit, and then plug the USB cable in to the computer. Either hyperterminal, putty, or some other program should be installed, and you should be able to run that, bringing up a “terminal emulator” window which displays the output from your board. (The “COM port” this comes up as will be something like COM5, COM8, or COM12, or another in that range (thanks to the abomination known as “plug and play” in Windows, we cannot predict which COM port you’ll get.) Set the port to use 8 bits, no parity, baud rate 115,200 and do not use either hardware or software handshake.