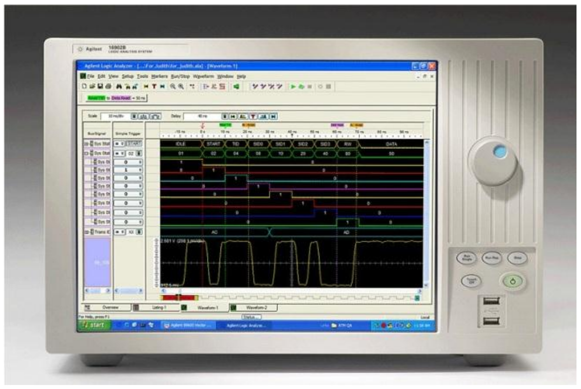
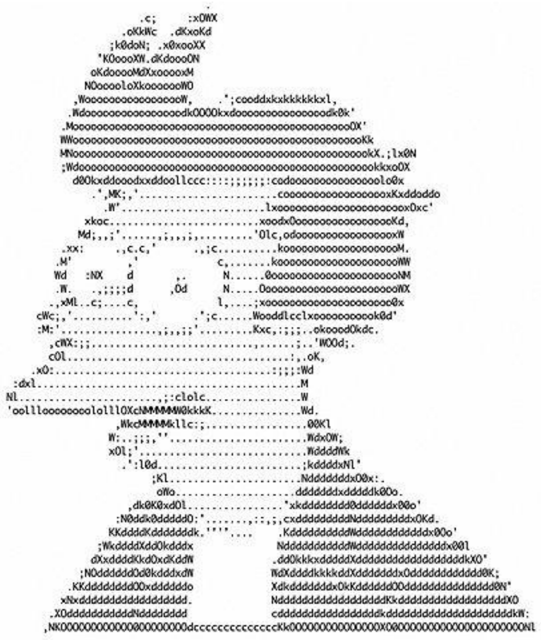
ECE 2036 Mbed Programming Assignment 2
Build your own mymbedDAQ

Most modern electronic test instruments such as multimeters and oscilloscopes are actually embedded devices that contain a computer with firmware, an LCD display, and high-speed Analog to Digital conversion hardware. Some high-end instruments such as Logic Analyzers even contain a PC with an operating system. Several demonstrations will be developed in this assignment that illustrate the basics of how these devices work using only the hardware available on the mbed chip and three jumper wires.



Unfortunately, the mbed module does not have a high resolution graphics display. For the display output on mbed, a VT100 or ANSI terminal character set will be used in this assignment. Primitive graphics can be produced using only ASCII characters (also called ASCII Art). An example of an ASCII art image is shown below. There are even automatic image conversion programs and ASCII video players. We will not be doing anything quite this complex.



Futurama’s Fry in ASCII Art

VT100 cursor commands are also used in text editors such as vi and emacs to move the cursor directly to any location on the screen. A C++ Terminal class for mbed that supports VT100 commands is available in the mbed cookbook. Two useful member functions include a fast clear screen, *cls()*, and cursor position control, *locate(col,row)*.

Most Terminal application programs like the one used in the earlier mbed lab to display *printf* output, support VT100 commands by default. In RealTerm, click the display tab and you must select ANSI (VT100) and not ASCII to turn this feature on and the number of rows should also be increased from 16 to 24 ("^" on center bottom). *Cout* can also be used but it requires a significant amount of the limited RAM memory on mbed.

Basic ASCII characters use only seven bits, but extended ASCII uses 8-bits to include 128 new special graphics characters by turning on the high bit and not using it for a parity bit. In C/C++ you can generate one of these characters using *char(xxx)*, where *xxx* is the digital value of the graphics character. A handy table of these graphics characters is available at http://en.wikipedia.org/wiki/Code_page_437 for IBM PCs. On Macs, a different selection than used in the example code for those characters would generate improved graphics since Macs use Unicode UTF-8 characters. Instructions for Mac users can be found at the end of http://mbed.org/users/4180_1/notebook/ascii-graphics-characters/. These graphics characters will be used to draw lines later in C/C++ using *printf* and *putc*. It is also possible to read a character on mbed coming from the PC's terminal application using *getc*. *Getc* will be used to select menu items in the demo program using the keyboard.

A function generator would typically be used to generate test signals. Instead of hooking up a function generator, the mbed module will also generate real signals for the test instrument to read in and monitor. It contains a D/A that can be used to generate an analog output. The mbed class for the D/A is AnalogOut. It is scaled from 0 to 1.0 just like AnalogIn. Mbed's PWM output hardware can generate a square wave automatically without the processor once it is activated. PwmOut is the mbed's C/C++ class that controls the 6 PWM output pins. PWM is widely used for motor speed control and dimming LEDs. For the logic analyzer test signal, a serial output pin will be connected to a DigitalIn pin.

Tektronix Function Generator

The maximum speed of the mbed's D/A is 1 Mhz and the A/D is 200Khz. With C++ classes and without writing some rather complex code that directly sets up the hardware for full speed it is quite a bit slower. To make it easier, for the lab we will sample data in the 10Khz range. A real oscilloscope or logic analyzer has orders of magnitude faster A/D and sampling hardware up to the Ghz range, but then they also can cost tens of thousands of dollars.

Breadboard Wiring

First to setup the breadboard for the code, three jumper wires will be required as shown in the table below:

Mbed jumper wire	Purpose
P18 to P16	Connects the D/A output to an A/D input (used for sine wave)
P12 to P13	Connects Serial out to a DigitalIn (used as digital test signal)
P21 to P15	Connects PWM out to an A/D input (used for square wave)

These jumpers connect the mbed generated output test signals to the mbed input pins used for the test instrument code.

Project Template Code

To save time, template code is provided for this assignment and comments in the code "//ADD CODE HERE" show where code must be added to complete the assignment. Comments in each section describe what the code needs to do. The template code project can be imported into your mbed compile project window by **first logging onto the mbed site** and **typing this URL**:

http://mbed.org/users/4180_1/code/mymbedDAQ/

On the web page that opens, click the **import this program** link. It will then open up the mbed compiler window and ask where to download all of the project files. The mbed cloud compiler template code project already includes the other libraries and routines needed (i.e., Terminal_VT100, FFT). Everything that needs to be changed is in the project's main.cpp file. **Even though much of the code is provided, you will be expected to understand all of it.**

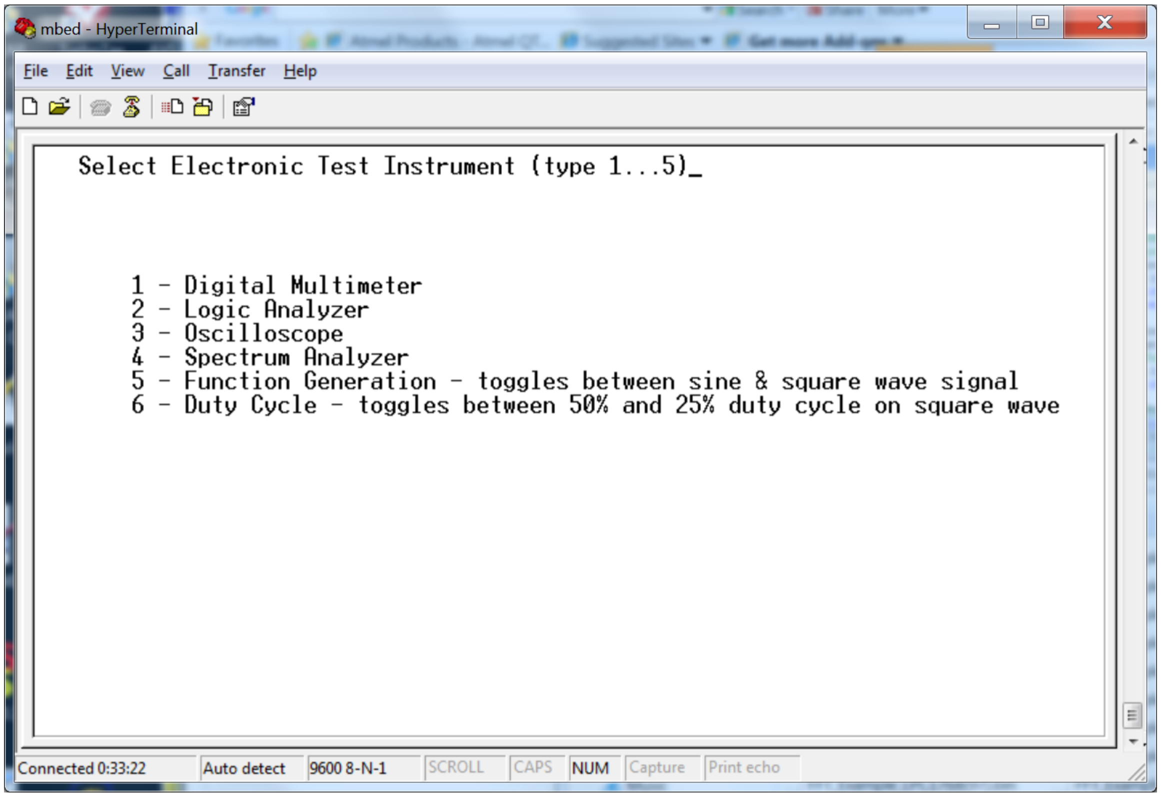
An executable only demo version can be downloaded to mbed and run at:

http://users.ece.gatech.edu/~hamblen/2036/handouts/Demo_mymbedDAQ_LPC1768.bin

Part 1 (20%): Menu, user input, and DMM

The initial screen display is shown below. The user selects the instrument by typing 1…5 in the terminal application and reading it in the mbed code with *PC.getc*. A switch statement using the PC input character calls a function for each instrument.

A *while* loop inside each instrument demo checks *PC.readable* to decide when to exit and return to the menu. *PC.readable* indicates that another character is available for *PC.getc*, but it does not wait for or actually read the character. It's a handy way to check for character input without waiting. Just about any computer has a similar feature to check for character input without waiting, but it might be difficult to find in the documentation.



```
 Select Electronic Test Instrument (type 1...5)_



     1 - Digital Multimeter
     2 - Logic Analyzer
     3 - Oscilloscope
     4 - Spectrum Analyzer
     5 - Function Generation - toggles between sine & square wave signal
     6 - Duty Cycle - toggles between 50% and 25% duty cycle on square wave
```

```
Connected 0:33:22   Auto detect   9600 8-N-1   SCROLL   CAPS   NUM   Capture   Print echo
```

VT100 Instrument Selection Menu

On the PC, the user moves the mouse to the terminal application window and types 1 through 5 to control the instrument demos. Hitting any key will exit the current demo once a complete display is redrawn.

The code for each instrument is in a separate C/C++ function in main.cpp. It makes more sense to implement them in the order 1-4 shown in the menu since the complexity increases and additional features are added in that order. The function generator code needs to be inside the sampling code provided in the template for the oscilloscope and spectrum analyzer since timing is critical.

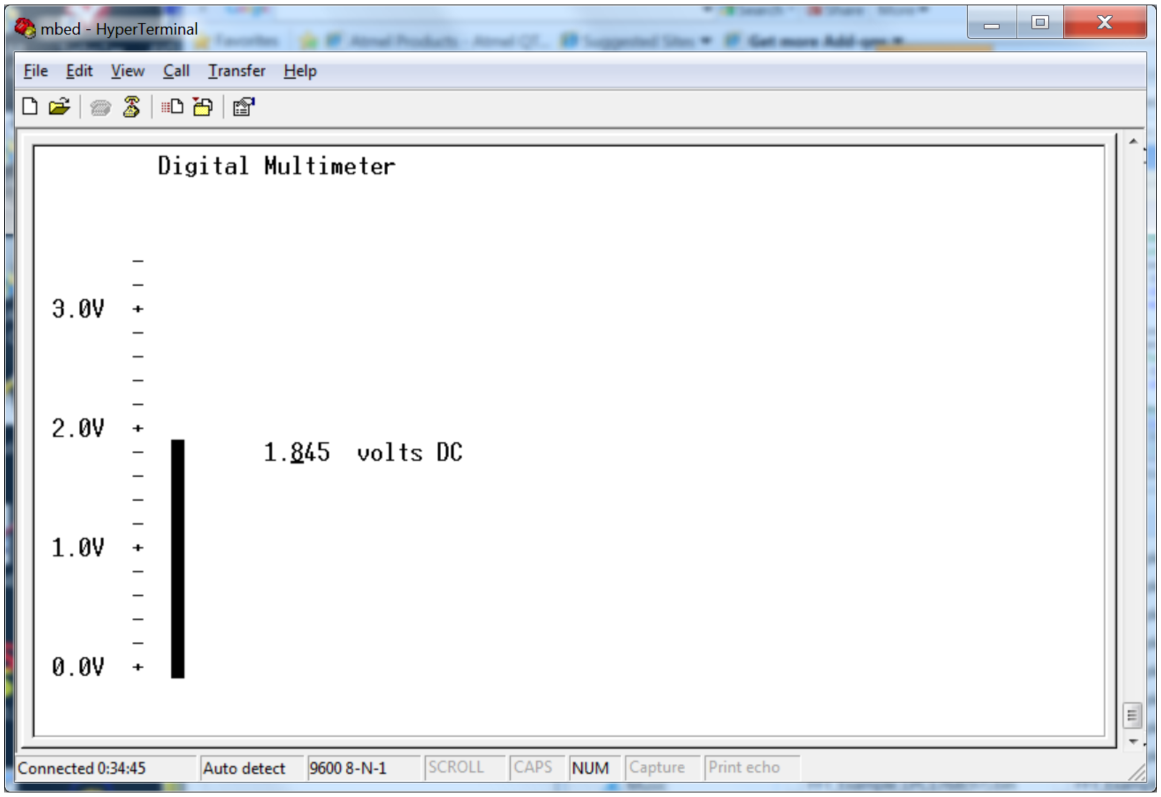
A Digital Multimeter (DMM)

A handheld DMM is perhaps the most common electronic test instrument and they are typically used to read analog voltage levels. The mbed Digital multimeter demo will read in an Analog voltage signal generated by the D/A. A *for* loop runs through the voltage from low (0.0V) to high(3.3V) . Recall that in C/C++ *AnalogIn* and *AnalogOut* are scaled from 0 to 1.0, but the actual external voltage level ranges from 0 to 3.3V. Note that many DMMs such as the one seen below also have an analog level bar.

A Fluke Digital Multimeter

In the mbed demo, the voltage will be printed out and using the graphics characters a vertical analog level bar will be generated. The VT100 commands to draw the screen are provided in the template code, but you must add code to output an analog signal using the D/A and read it back in using the A/D. Study this code example and make sure that you understand how *PC.cls*, *PC.locate* and *char(xxx)* are used to draw and re-draw the screen since these features will be needed later on in the code you need to write.

A special sequence of several characters is used in *PC.locate(col,row)* to move the cursor. The upper left corner is (0,0) and the bottom right is (79,23). To change a character in the image, move the cursor to that character position and overwrite it with a new character. An ASCII space or blank ‘ ‘ or char(32) will also clear out an existing character. *Locate* is faster than clearing and redrawing the entire image when only a few characters change. If you are printing characters from left to right on the same line, the normal sequence of *printfs* or *putcs* is significantly faster than using *locate* to move the cursor to print each character on a line (since fewer characters are sent out).



```
        Digital Multimeter


          -
          -
 3.0V     +
          -
          -
          -
          -
 2.0V     +
          -   █       1.845  volts DC
          -   █
          -   █
          -   █
 1.0V     +   █
          -   █
          -   █
          -   █
          -   █
 0.0V     +   █
```

VT100 DMM display with an analog input signal generated by the D/A

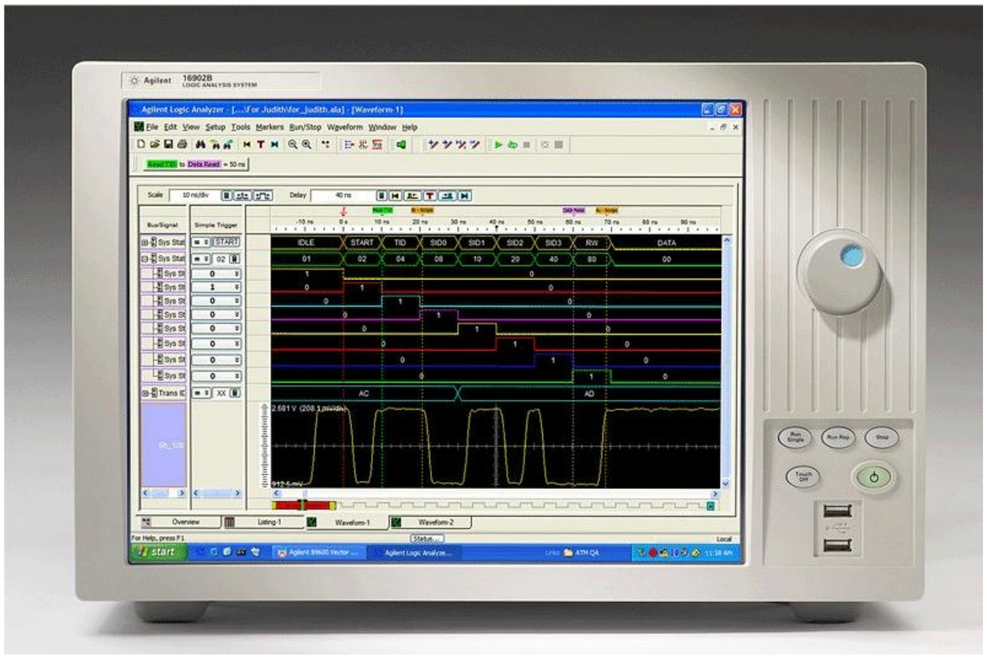
Once it is working, the jumper wire could be removed and it could be used to read in any external analog voltage input as long as it was less than 3.3V and greater than 0V. A small 1.5V battery could be used. If you try this, do not forget to connect the ground signals first. Larger voltages would need to be scaled down by adding an external switch selectable resistor network or an analog circuit with programmable gain. Autoscaling DMMs contain this analog circuit. These analog circuits also need high input impedance to avoid any loading on the signal (i.e. changing it when measuring it)

Cautionary Note: Do not connect any external inputs to the A/D that can be greater than 3.3V. It could blow out the A/D input circuits and even perhaps the processor as there are no special overvoltage protection circuits. Also remember that a signal ground is required for an external input or to monitor an output pin from mbed on an actual laboratory instrument such as an oscilloscope.

Part 2 (20%): A Logic Analyzer

Logic Analyzers are used to read in high-speed digital signals. They contain multiple high-speed digital inputs and large memory buffers. On mbed, a slow logic analyzer will be implemented by reading a *DigitalIn* pin in a loop into a vector. A 1D array might actually be a bit faster than a vector, but we will be sampling a slower signal. In fact, a *wait* will be added in the sampling loop to slow it down so that the test data scales and fills the screen properly.

Perhaps the most common way to use a logic analyzer is to read in a digital signal and display its value as it changes over time (i.e. a timing diagram) as seen in the logic analyzer below. Since it is a digital signal, analog voltage levels are not displayed. Just long sequences of 0's and 1's are stored in the large internal memory buffers.

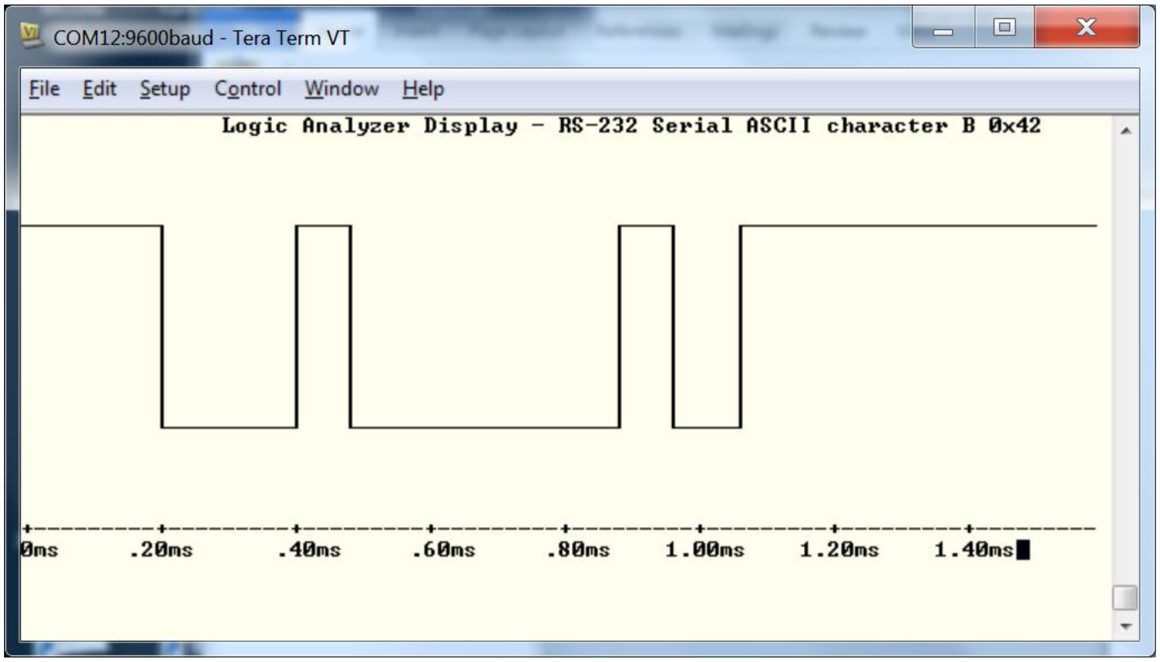

Agilent Logic Analyzer used to display digital signals

For a test signal, we will read in a serial output bit coming from mbed. RS-232 serial ports (called COM ports on a PC) take an 8-bit ASCII character and shift it out one bit at a time. A low start bit is added that signals a new character is coming. It is followed by the eight data bits in low to high order, and finally a high stop bit is added. So each 8-bit character actually requires 10-bits.

The baud rate is the rate at which the bits change. No clock is sent out, rather both ends must be setup for exactly the same baud rate and the hardware receiving the serial bit stream figures it out by checking for the stop and start bits. The idle state is high. On an actual RS-232 device, the voltage levels are not 0 and 3.3V and a special voltage level conversion IC must be added.

When a new character is sent to the serial port hardware (i.e. a UART) using *putc*, it immediately starts shifting out the bits at the current baud rate. The default baud rate of 9600 is used (9600 bits per second).

The desired output is shown below. Look for the low start bit, the eight low to high data bits, and the high stop bit. The upper right label shows the character being sent out for capture. In this example, it is probably faster to use *cls* and redraw the screen each time. Special graphics characters are used to draw the signal. Two characters are used to draw a horizontal or vertical line and also four special right angle corner characters to connect horizontal and vertical line segments. The template code shows the characters you will need to use and don't forget that *char(xxx)* will generate the special graphics characters in C/C++. Drawing the timing diagram from left to right, and minimizing the use of *locate* when possible will result in a faster display update.


COM12:9600baud - Tera Term VT
File Edit Setup Control Window Help

```
                    Logic Analyzer Display - RS-232 Serial ASCII character B 0x42

+---------+---------+---------+---------+---------+---------+---------+---------
0ms      .20ms      .40ms      .60ms      .80ms     1.00ms     1.20ms     1.40ms
```

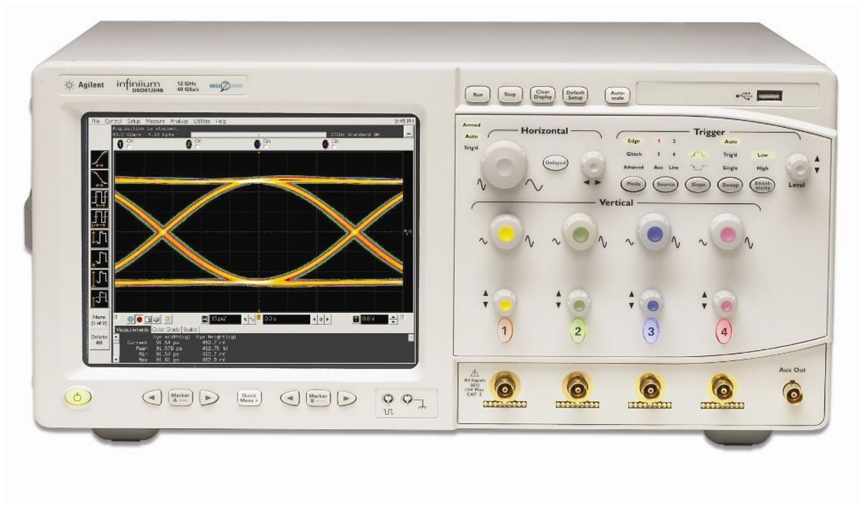
The VT100 Logic Analyzer Display for a character sent out as a serial bit stream.

A vector must be used to sample and hold the digital data, so that you have some experience using vectors. Experimentation was required to determine the time duration of the *wait* for each sample delay, so that an entire character filled the screen. The labels on the X axis are also approximate. With Tera Term, a bad character shows up occasionally on a high to low corner. This did not happen with the other terminal application programs. The mbed might be a bit too fast for the older Tera Term software.

The demo code automatically outputs the next ASCII character, samples, and generates a new display in a loop until a key is hit on the keyboard.

Part 3 (30%): An Oscilloscope

Oscilloscopes display an analog voltage input versus time. Early oscilloscopes used a cathode ray tube like old TVs. Modern digital oscilloscopes contain an A/D and the converted digital signal is stored in a memory buffer to generate the display. Mid-range scopes often run an embedded version of Linux and many of the high-end scopes are PCs running Windows with special A/D hardware and analog front ends.


An Agilent High-Speed Oscilloscope

Sampling Data

The mbed also has hardware timers that can be used with the *Ticker* or *Timer* API. This approach will be used in the next two test instruments. When sampling data, always try to use the minimum number of operations in the sample loop to maximize the sample rate. Scaling and other calculations can be done later outside of the sampling loop after obtaining an entire sample sequence.

Using a loop, data from the A/D is read in and stored in an array of floating point values. The sample period is carefully controlled using a programmable timer that counts down and generates an interrupt. With the mbed *ticker* class, the interrupt activates a C/C++ function that sets a flag and quickly returns. This flag signals the sample loop (waiting or “spinning” in an empty *while* loop) that it is time to take another sample. This produces more accurate uniform sample periods than using carefully adjusted time delays using *waits*. The actual conversion time delays on A/Ds also varies depending on the voltage level making it almost impossible to accurately control the sample period without a timer.

When accessing and modifying global or shared data with more than one process (i.e., in this case a main and interrupt routine) mutual exclusion is needed to avoid errors due to data inconsistency. This can happen when switching back and forth between processes while a new global or shared variable value is being changed in a register and has not yet been written back to memory. With an OS, synchronization primitives such as a mutual exclusion lock would normally be used to solve this critical code section problem. On a single processor system with no OS, mutual exclusion for critical sections can be provided by disabling interrupts. In the ARM C/C++ compiler used for mbed, the *volatile* keyword in C on simple variable declarations (i.e. char, int, bool and not arrays or objects) solves this problem by making basic operations on that variable atomic (i.e., non-interruptible). It makes it atomic by placing special instructions to disable interrupts and enable interrupts around operations on the variable. The sample flag is such a variable used in both the timer interrupt routine and the sample function.

For analog test signals, a function generator would typically be used in the laboratory. On mbed, two approaches can be used to generate sine and square waves. For a sine wave, values for a sine wave are pre-computed and stored in an array to save time. The sine wave is scaled and a DC bias is added so that it is always between 0 and 1.0 for the *AnalogOut* API. The array is then sent out to the D/A with the correct sample period to generate a sine wave.

PWM hardware on mbed can be used to generate a square wave automatically in hardware. One PwmOut C/C++ method sets the period and another the duty cycle. Hitting "6" on the main menu toggles the duty cycle on the square wave between high 50% of the time to high 25% of the time.

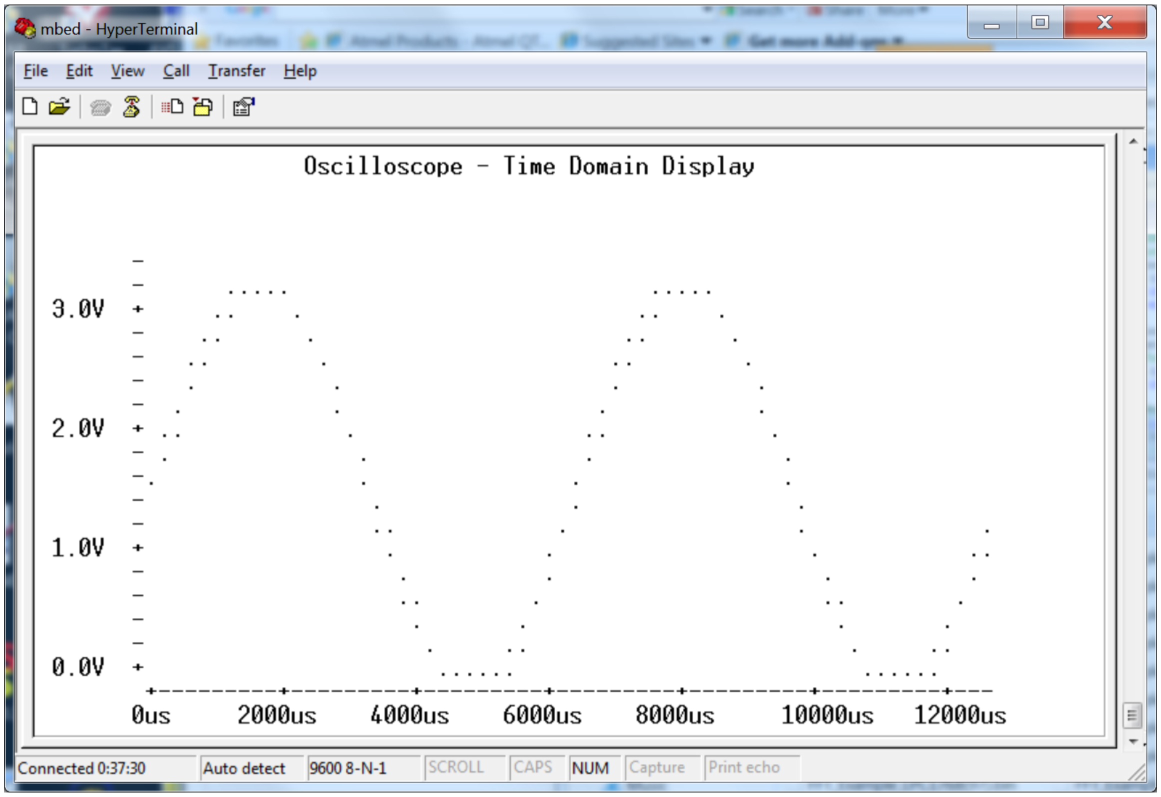
Both of these output test signals could be checked with a real oscilloscope. The sine output code would need to be modified to be constantly running to catch it on an oscilloscope as it is only active during data sampling in the current code. The PWM square wave would be a bit easier to see since it is hardware based and continuously running.

One A/D input channel is used for the sine wave from the D/A and another input channel is used for the PWM square wave. Hitting a "5" in the main menu sets a flag that selects which input channel to use in the sampling routine (i.e, sine or square).

The oscilloscope calls a helper function to display the data samples. After clearing the screen and drawing the X and Y axis tick marks, to generate the data display, a loop moves through the array of samples from left to right. Each analog sample is scaled to fit the screen height and printed at the correct position using *PC.locate* followed by *PC.putc('.")*. Two examples display are seen on the next page.

The demo code automatically increases the frequency of both the sine and square wave input, samples, and generates a new display in a loop until a key is hit on the keyboard.

Just like the earlier DMM example, external analog circuits would be needed to properly buffer and scale any external analog inputs in a real instrument. Faster more expensive A/D hardware could also increase the sampling rate by orders of magnitude.

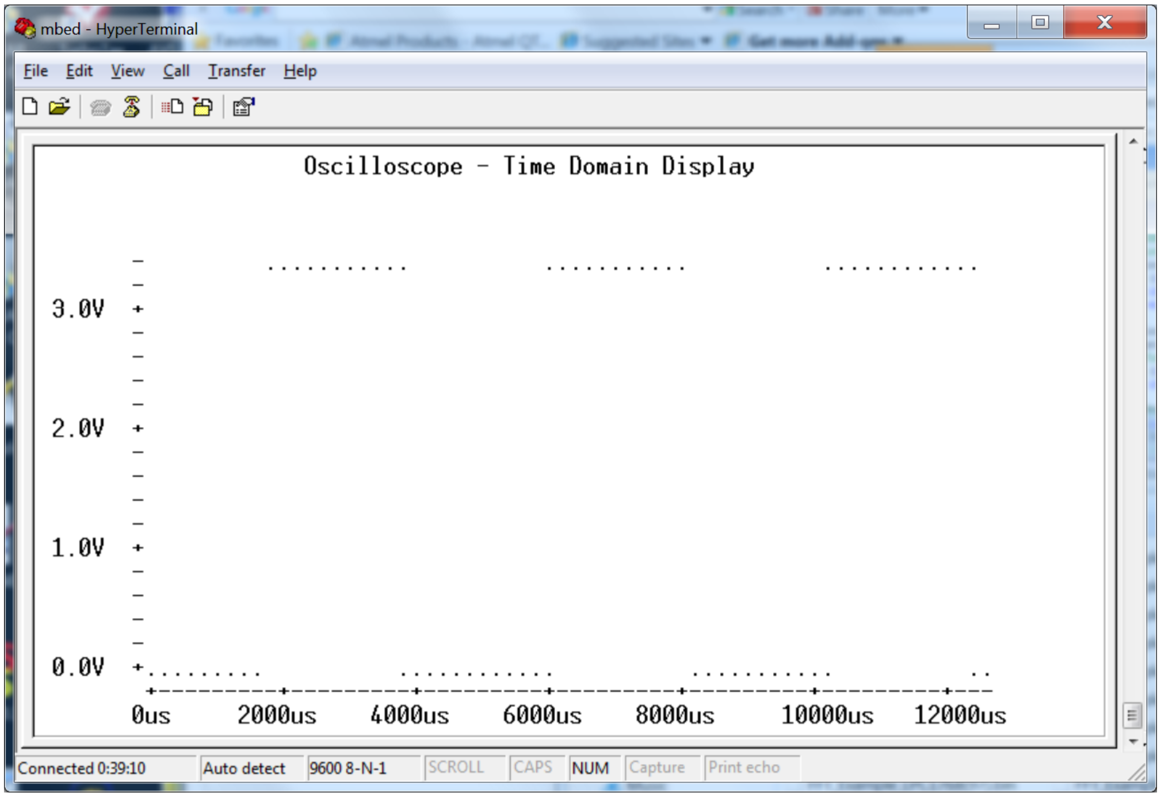


```
                 Oscilloscope - Time Domain Display

 3.0V  +
 2.0V  +
 1.0V  +
 0.0V  +
       +---------+---------+---------+---------+---------+---------+---
       0us     2000us    4000us    6000us    8000us    10000us   12000us
```

Connected 0:37:30 | Auto detect | 9600 8-N-1

VT100 Oscilloscope display using sine wave source generated by D/A



```
                 Oscilloscope - Time Domain Display

 3.0V  +
 2.0V  +
 1.0V  +
 0.0V  +
       +---------+---------+---------+---------+---------+---------+---
       0us     2000us    4000us    6000us    8000us    10000us   12000us
```

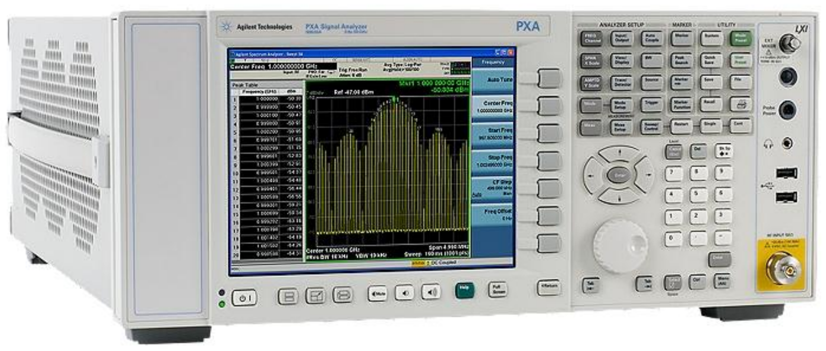
Connected 0:39:10 | Auto detect | 9600 8-N-1

VT100 Oscilloscope display using square wave generated by PWM hardware.

Part 4 (30%): A Spectrum Analyzer

A spectrum analyzer measures the magnitude of an input signal versus frequency. Mathematically by using Fourier analysis, many things are easier to examine in the frequency domain than in the time domain. By analyzing the spectra of electrical signals, dominant frequency, power, distortion, harmonics, bandwidth, and other spectral components of a signal can be observed that are not easily detectable in time domain waveforms. These parameters are useful in the characterization and design of filters, radio transmitters and receivers, control systems, and other electronic devices.

The display of a spectrum analyzer has frequency on the horizontal axis and the amplitude displayed on the vertical axis as seen in the picture below. An FFT-based Spectrum Analyzer uses A/D hardware to sample an analog signal, stores it in memory buffers, and then computes an FFT to switch over the frequency domain for display.



An Agilent Spectrum Analyzer displaying the frequency content of a signal

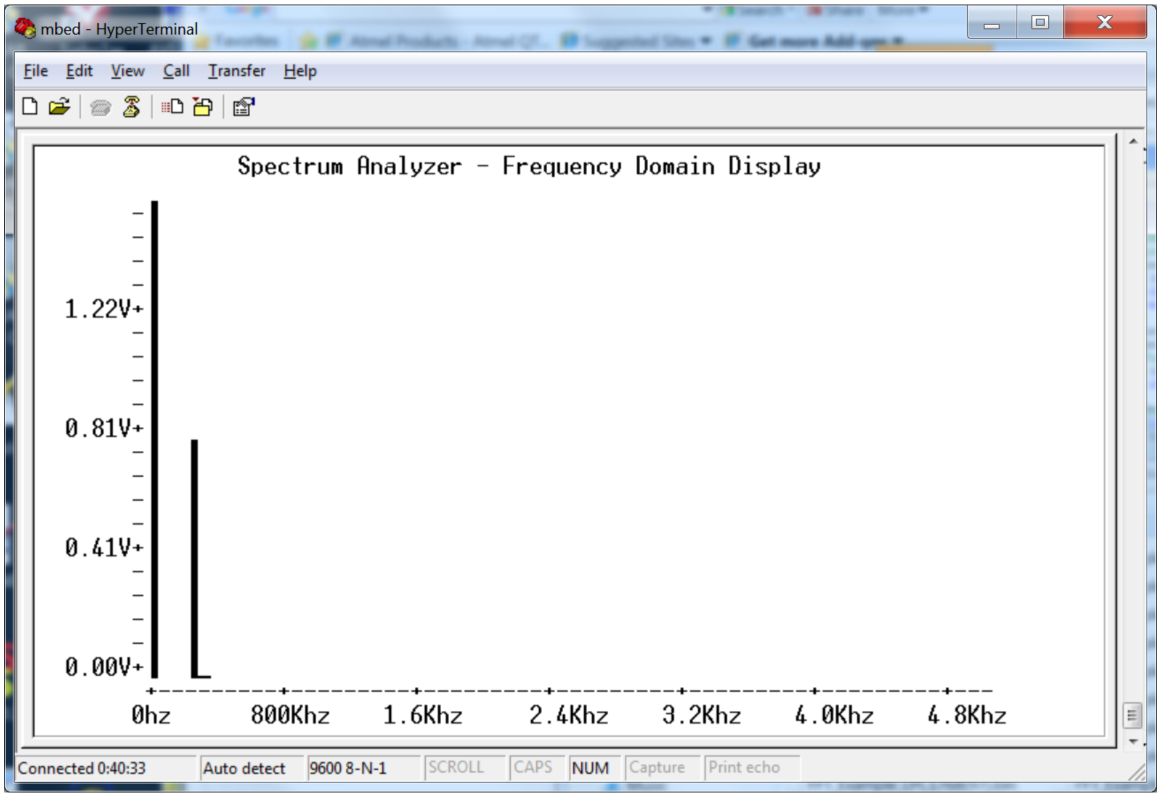
For the mbed demo, an FFT function and another function to compute the magnitude of the complex frequency response from the FFT are provided in the template code. The magnitude is used for the display and only the positive portion of the frequency is displayed.

It is then a matter of scaling and displaying the frequency response data from the array as a series of vertical lines. The large bar at zero indicates the DC offset level of the original signal. Since the signal was scaled from 0 to 1 by *AnalogIn*, the signal does have a large DC level. In an FFT, the first bar at zero is the DC level. It varies quite a bit depending on the test signal so autoscaling on the Y axis will result in an improved display. It autoscales by first searching the FFT magnitude array for the largest value and using the maximum value to scale the bars to the size of the vertical display area.

The Spectrum Analyzer uses the same sampling setup as the oscilloscope. It then calls the oscilloscope's time domain display function first so that the signal can be observed first in the time domain and then in the frequency domain with another display spectrum function. A sine wave’s FFT should have one pulse at its base frequency. A square wave’s FFT has pulses that diminish in amplitude at odd harmonics. A square wave with only a 25% duty cycles starts looking like a sync function with all harmonics, but forth harmonic terms should be near zero.

The demo code automatically increases the frequency of both the sine and square wave input, samples, and generates a new time and frequency display in a loop until a key is hit on the keyboard.

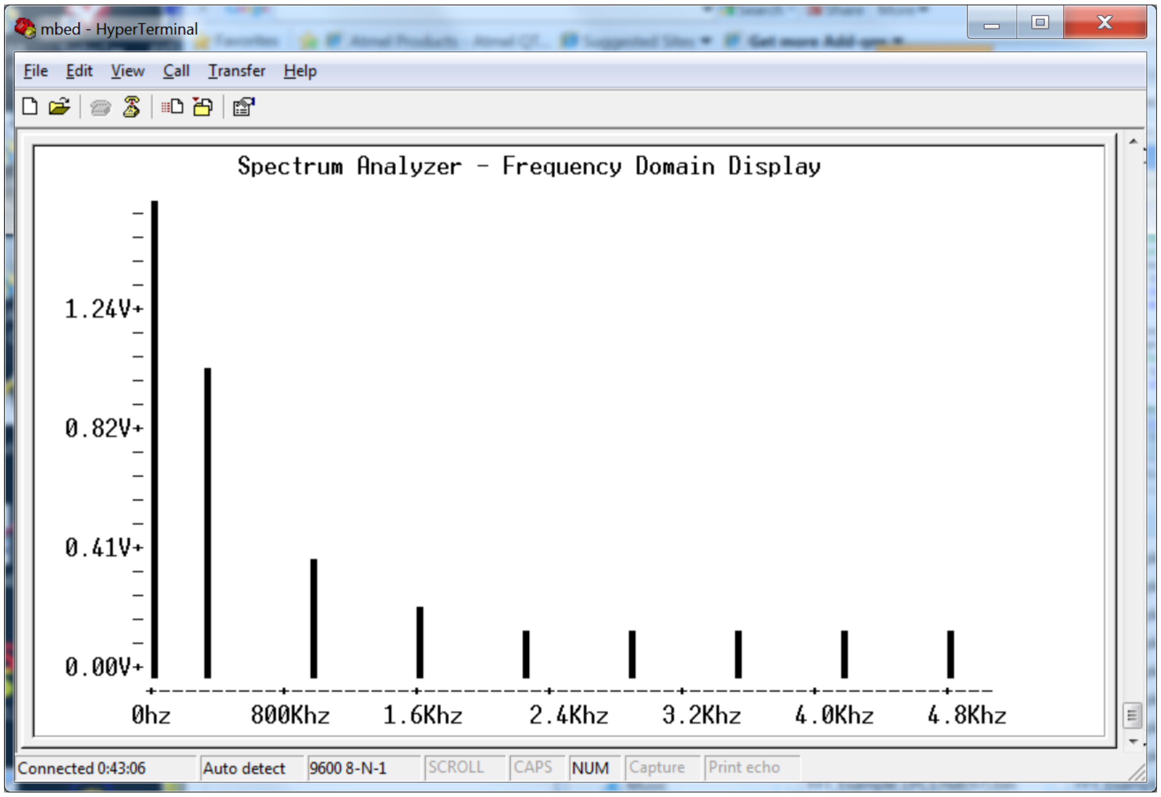
Somewhat improved signals could be obtained with the use of filters (i.e., the “ideal low pass filter”). Without the filter aliasing can occur, but for the test signals the results are not bad and what one would expect.



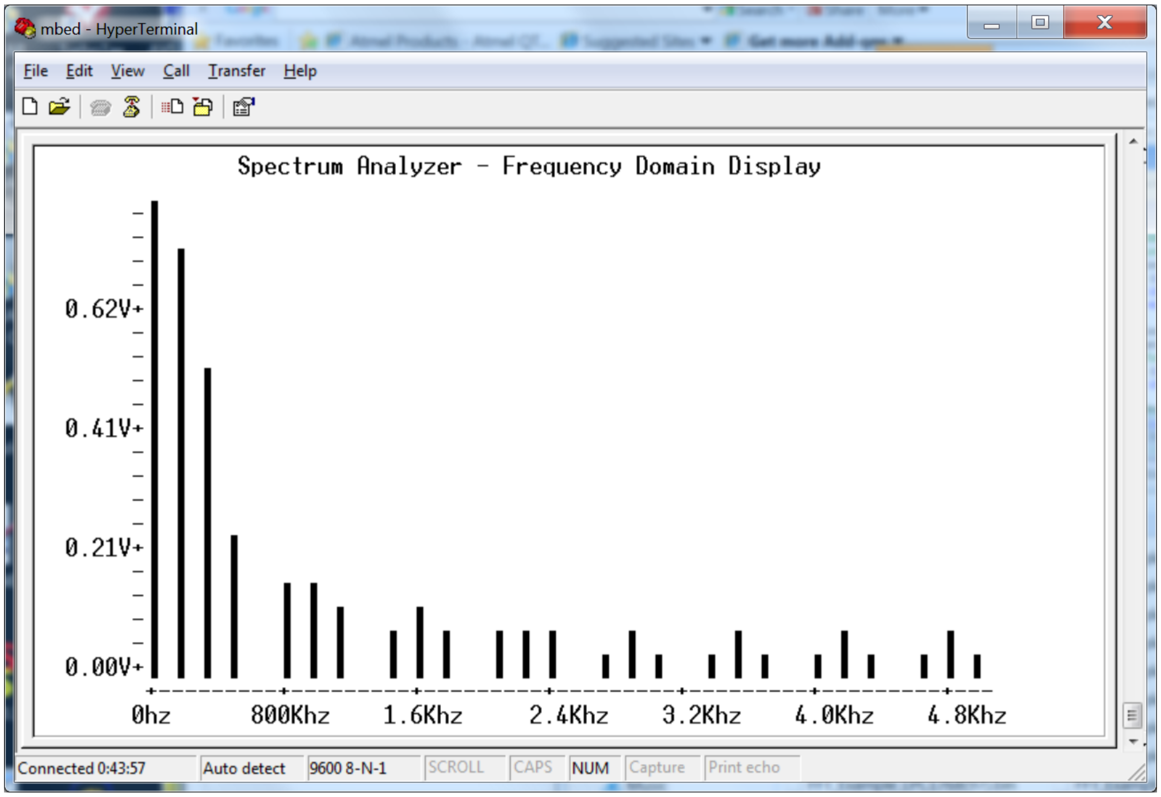
VT100 Spectrum Analyzer display using the sine wave input signal

The signals are not synchronized to always start at a zero crossing at the beginning and end of a cycle of a square wave in the sample sequence, so occasionally the pulses may spread out just a bit and not be as sharp as the images here on the square wave.

Note that only the odd harmonics appear in the FFT of a square wave in the next screen capture. That is correct for a square wave.



VT100 Spectrum Analyzer display using the square wave input signal.



VT100 Spectrum Analyzer using the square wave input signal with a 25% duty cycle.

Note that all harmonics appear in the FFT of a square wave with a 25% duty cycle and the amplitude dies off like a sync function, but every 4th harmonic is near zero. Also note that the DC level (first bar) is lower (i.e., it is only high 25% of the time). So the Y axis auto scaling feature and FFT appears to be working.

Ideas for Further Enhancements

1. Increase the baud rate for faster display updates.

2. Special ANSI character sequences can be used to add color

3. Increase the number of channels on the logic analyzer and oscilloscope.

4. Calibrate the instruments by checking them with a real instrument

5. Add an external clock input feature to the logic analyzer using interrupts.

6. Add triggering features to the logic analyzer and oscilloscope.

7. Add time autoscaling to the oscilloscope.

8. Add a color graphics display to your mbed hardware. Two interesting options are a Nokia LCD or a uVGA controller.

9. Talk to hardware directly for faster sampling rates.

10. Build an external analog circuit to scale down larger voltage level inputs.

11. Instead of a VT100 terminal window, write a custom window's application for the PC that reads the data values from mbed over the USB Virtual com serial port and displays the data using hi-res graphics on the PC.