Task 4 **Due: November 6, 2007**

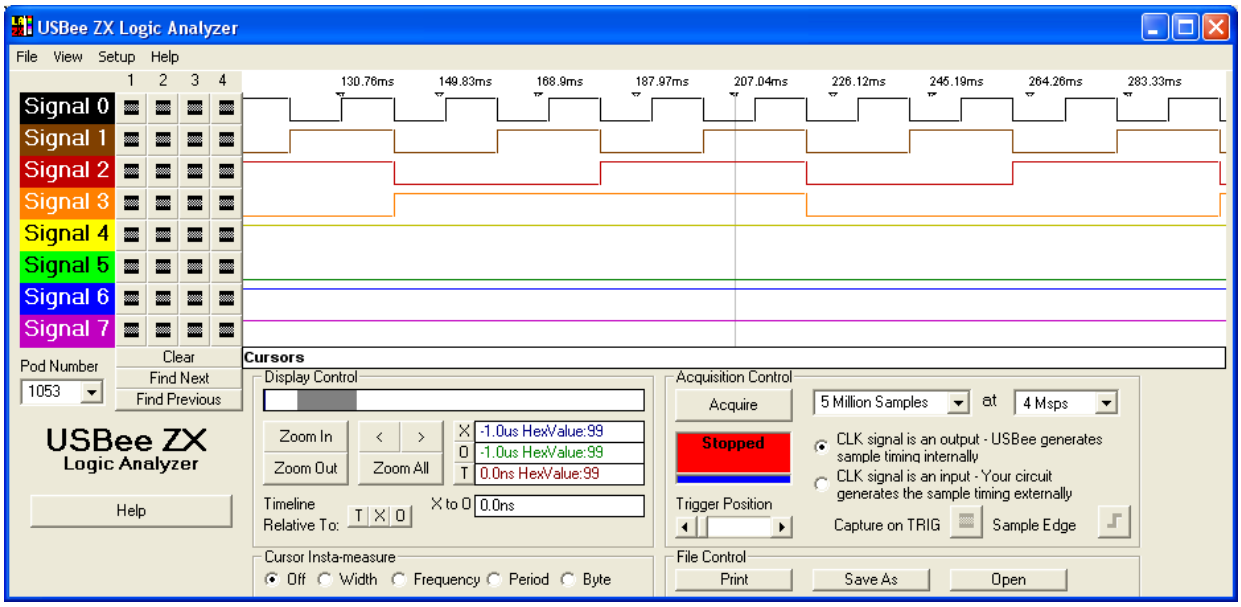
Task Purpose: **Using the Logic Analyzer**

Each team has, in its kit, a USB based logic analyzer. This device allows you to look at digital signals. The purpose of this task is to familiarize the team with the use of this device.

Video tutorials describing the use of the USBEE logic analyzer can be found on the company web site. The software should already be loaded on the computers in the ELC. Note that the software can be loaded on any machine, but only works in demo mode unless the test pod is plugged in. (The pod number will be listed as “demo” in this case.)

Note: Due to the permission scheme used in the ELC, if you plug your logic analyzer pod into a USB port and the computer starts through the “found new hardware” process, you will not be able to proceed using that port, as you don’t have permissions to install new hardware. Most of the USB ports in the ELC have had the hardware installed, but we still find some that don’t. Try using another USB port, and if that fails, another computer.

1. Write a program that sets port D to be an output port and increments the value continually, with a 10 msec delay between increments. Connect the leads 0-3 on the logic analyzer to the header pins for port d bits 0 – 3. Connect the logic analyzer pod ground pin to a pin on the microcontroller board labeled “gnd”. Verify that you can view the signals on the lower 4 bits of port D as shown below. (Notes: The colors of the wires from the pod follow the resistor code order for your convenience. Also, the leads from the logic analyzer should fit over the header pins without using the “E-Z Hooks”. This is a much more secure way to make connections to the logic analyzer.)



2. Using the cursor measurement tools, determine the period of the top square wave.

3. Repeat part 1, but without any added delay. (You should only need a single statement inside a while(1) construct.) Recapture the signal. *Can you visually determine that the LED's connected to port D changing? Copy a screen capture of this logic analyzer screen and include it in your task report.*

4. Repeat part 3 and acquire the logic trace with the sampling period set to 1M samples per second. Make multiple measurements of the width of the high and low portions of the highest frequency square wave (D0). Repeat with the sampling rate set to 24M samples per second. *What are your conclusions?*

5. Load the terminal program you used to test you serial I/O routines, and connect lead 0 of the logic analyzer to C6 (the transmit pin out of the onboard USART) and lead 1 to C7 (the receive pin going to the onboard USART.) Set the logic analyzer to trigger when the signal on C7 changes from high to low. When you hit acquire, the logic analyzer will indicate that it is waiting for the specified trigger. Type your name into HyperTerminal and view the logic analyzer output. You should see the characters you typed and the corresponding echo on the analyzer.

6. Use the serial decode function (under the view menu) to decode the characters you typed. *Copy a screen capture of this screen and include it in your task report.*

In more complex programs, it is often necessary to trigger the logic analyzer at some particular place in the code. One way to do this is to set some unused I/O pin to be an output, toggle it in the code at the desired spot, and use this signal to trigger the logic analyzer. For example, suppose port A1 is unused. It could be used to trigger the logic analyzer by doing the following:

```
/* setup bit a1 as an output and define
   a macro to toggle the bit */

volatile bit a1@PORTA.1;                    // name bit
#define toggle_a1 a1=1;nop();nop();a1=0;  // define toggle
trisa.1 = 0;                                // makeoutput
a1=0;                                       // start at zero
```

With this code included in your program, you can make bit 1 of port a toggle with the statement

```
toggle_a1;
```

You can use this bit to trigger the logic analyzer at the point in your code where you want to start looking at what the signals are doing.

This can be very useful in a number of situations. For example, if you want to know if you are getting an interrupt, you can place the **`toggle_a1`** statement in the interrupt service routine. If the logic analyzer doesn't trigger, you aren't getting an interrupt.

Report:

Please include answers to the questions above, as well as the requested screen captures in your task report.