Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 - Introductory Digital Systems Laboratory (Spring 2009)

Laboratory 1 - Introduction to Digital Electronics and Lab Equipment (Logic Analyzers, Digital Oscilloscope, and FPGA-based Labkit)

Issued: February 4, 2009
Checkoff and Report Due in Class (1PM) on: February 18, 2009

Introduction

This lab assignment introduces you to important tools and devices that we will be using throughout the term. You will be introduced to the following:

- Logic analyzer and digital scope.
- 74LS TTL series chips, including the ‘00, ‘04, ‘163, ‘393, ‘74
- 74HC00, a CMOS NAND gate
- 1.8432 MHz Crystal Oscillator
- TTL and CMOS voltage levels
- Karnaugh Maps and Boolean Algebra
- Latches and Flip-flops, Simple Sequential and Asynchronous circuits
- Simple Verilog
- ModelSim (Verilog simulator), Xilinx FPGA labkit, Xilinx programming software

The following are relevant handouts that you should be using in conjunction with this lab:

- Lectures 1-5
- Safety Memo

Procedure

This lab is divided into several exercises to guide you through the design, construction, and debugging process. You will be asked to wire circuits for many of the exercises. Save all of these circuits until you have completed the lab as many of these circuits might be reused in subsequent parts of this lab.

1. Read and understand the whole assignment.
2. Design, build, test, debug, and fix each exercise in turn. Be sure to answer all the questions in the report template before you get checked off. You do NOT have to get checked-off on a exercise before proceeding to the next one. For each exercise, be sure to have the appropriate figures ready. Turn in the completed report template after your checkoff.
3. **Do not use the power supply from the labkit to power the external proto board. Use the bench power supply. Problems 1-7 should not use the FPGA labkit and should be done on the external breadboard.**

Exercise 1: TTL/CMOS Static DC Characteristics

The logic values of 1 and 0 are represented by voltage levels in the hardware implementation. The voltage levels and other electrical characteristics are not standardized from one logic family to another. 6.111 will use both TTL (Transistor-Transistor Logic) and CMOS (Complementary Metal-Oxide Semiconductor) logic. The voltage ranges for the two logic families are different.

In this exercise, you will first measure the electrical characteristics of a TTL and CMOS gate using the circuit in Figure 1. Wire up this circuit using a 74LS00 part. Do not forget to wire power and ground! These connections are usually omitted from logic diagrams, as the power and ground of the 74LS series are generally the top-right and bottom-left pin, respectively. Typically, the top of the chip has a small semi-circular cutout, or a white dot next to pin 1.

Ground the input of the inverter (the first NAND) and measure the output voltage using the oscilloscope. Be sure to be using the voltage markers on the oscilloscope. Connect the input to a logic '1' and repeat the measurement.

Figure 1: (a) Logic Level Measurement (Measure voltage at *OUT* node).
(b) Power Supply Wiring.

Next, use a 74HC00 and wire up the same circuit and hook V_{CC} to a +5V supply. Perform the same measurements and record your results.

Look up the valid input and output voltage ranges using the datasheet for a 74LS00 and a 74HC00. For each experiment, do your output values satisfy the range specified in the datasheets?

Consider interfacing a TTL inverter to a CMOS inverter and vice versa. Look at the datasheet titled "HCMOS Family Characteristics". Based on the +5V supply you used, find out the recommended input voltage for HCMOS inputs. Discuss potential issues when interfacing TTL and CMOS components? Run the two experiments (interface TTL to CMOS and vice versa) and make voltage measurements.

Exercise 2: Build Your Own Ring Oscillator

Important timing parameters associated with the speed of digital logic gates are the propagation delay time t_{PD}, and the output signal rise and fall times, t_r and t_f. Propagation delay is a measure of how much time is required for a signal to change state. It is measured as the time from the 50% point of the input to the 50% point of the output (Figure 2). It is often cited as the average of the high-to-low and low-to-high delays (corresponding to the two transitions). The rise and fall times represent the amount of time for a signal to change state. To measure rise and fall times, you should be using the 10% to the 90% point, or vice versa.

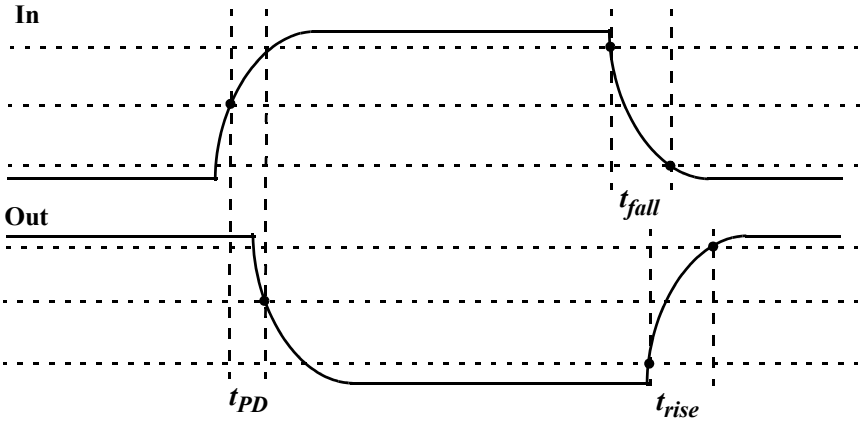

Figure 2: Timing Characteristics (10%, 50%, 90% marked).

Construct the ring oscillator shown in Figure 3 using a 74LS04 with as little wire as possible. From this circuit, determine the average propagation delay of a TTL inverter by measuring the period of oscillation by using the time markers on the oscilloscope. You can determine this by determining the number of gates a signal must travel through to complete a full period of oscillation.

What should the period of oscillation be with 3 inverters in the ring? Rewire the circuit and measure the period. Comment on the new result.

Insert a long piece of wire (about 3 feet) into the ring of 3 inverters. Observe how this extra length of circuit affects the signal. Can you explain the change?

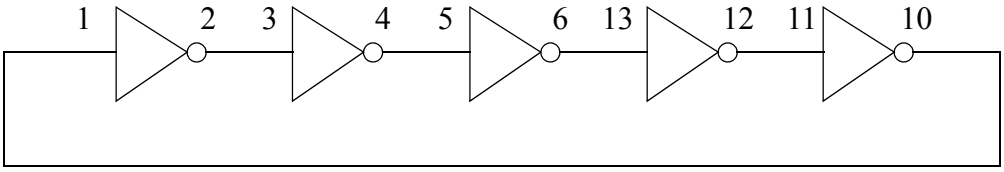

Figure 3: Ring Oscillator (using a 74LS04).

Finally, take a single inverter, and wire the output to the input. The voltage should be stable. If it isn't, connect a capacitor (0.01 μF) between this single node and ground to stabilize the voltage. Measure the voltage, and explain the significance of this voltage.

Exercise 3: Glitches

Wire up the circuit in Figure 4 using a 1.8432 MHz crystal oscillator and a 74LS00. Be sure to wire the crystal oscillator right side up. Pin 1 should be marked with a dot. The output *GLITCH* is a purposely glitchy output caused by the circuit. Using the oscilloscope, measure the width of the glitch.

Next, add *CLK* and *GLITCH* as signals to the logic analyzer and measure the width of the glitch (please refer to http://web.mit.edu/6.111/www/s2008/handouts/quick_la.html for a quick introduction to the logic analyzer). Is there a significant difference between your two different types of measurements? Why does this glitch occur, and what is the lesson learned from this exercise? Under what conditions is it a bad idea to use a glitchy signal as an input?

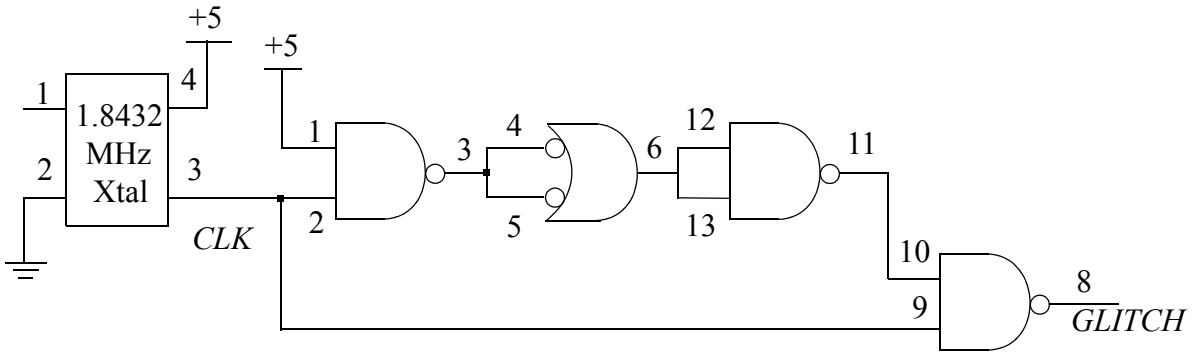

Figure 4: Glitch Measurement Circuit (74LS00).

Exercise 4: Asynchronous Counters

Wire up the 1.8432 MHz clock from the previous exercise to an 8-bit ripple counter as shown in Figure 5. You do not need to disconnect the previous circuit. In this exercise, we are concerned with how each output bit changes with respect to other bits. Since this counter increments its count every falling edge of the clock, each output bit must have a period twice that of the next significant bit. Because of this characteristic, counters make good clock dividers; if you want a slower frequency clock, it may be helpful to use a counter or a series of counters. Verify the counter's operation using the oscilloscope.

Trigger on either a rising or falling edge of the MSB, and measure the time between the falling-edge of *CLK* to edge of the MSB. Estimate the clk-to-q delay for each flip-flop in this chip.

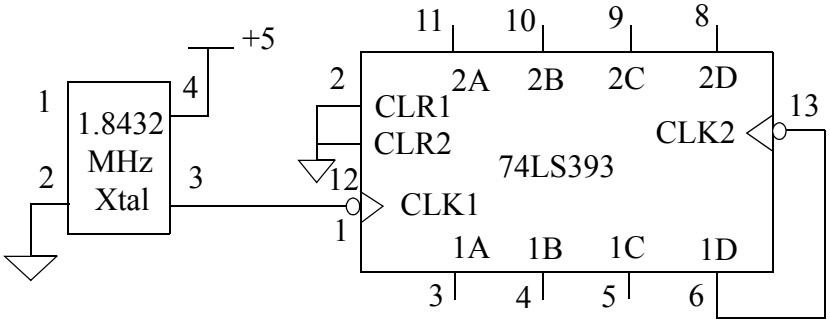

Figure 5: Clock and Ripple Counter.

Exercise 5: Synchronous Counter

One of the most useful of the 74LS series is the 74LS163. This is a 4-bit loadable synchronous counter with a synchronous reset. Wire two 74LS163s so that they count continuously as shown in Figure 6. Trigger your scope on the most significant bit (MSB) and verify its operation.

Explain why the *RCO* and *ENT* are connected between the two counters, and explain how they work. What is the difference between the *ENT* and *ENP* inputs on a 74LS163?

How long does it take, after the rising edge of the clock, for one of the flip-flops to change state? You should be triggering the oscilloscope on an appropriate output bit. Does it matter which output bit you choose?

Use the logic analyzer to capture the Q_A, Q_B, Q_C, Q_D and *RCO* outputs of the *low-order* counter. Display the values of *A*, *B*, *C* and *D* as a 4-bit hex number. For a quick introduction to the logic analyzer, please refer to http://web.mit.edu/6.111/www/s2008/handouts/quick_la.html.

Look at the *RCO* output of the *low-order* counter. Use the "Glitch Trigger" feature of the logic analyzer to see if you can locate any *RCO* glitches. Do you observe any glitches on the carry output? Glitches like these are hard to see as they are very short. The carry output does not always have glitches, just sometimes for particular chips. Try using the oscilloscope as well to look for glitches.

Be prepared to comment on the difference between the 74LS163 and the 74LS393 in terms of design and performance; in particular, the speed and area of the device. By area, we are asking you to consider how complicated it is to implement the counter. How many flip-flops and how much logic is required to implement each counter?

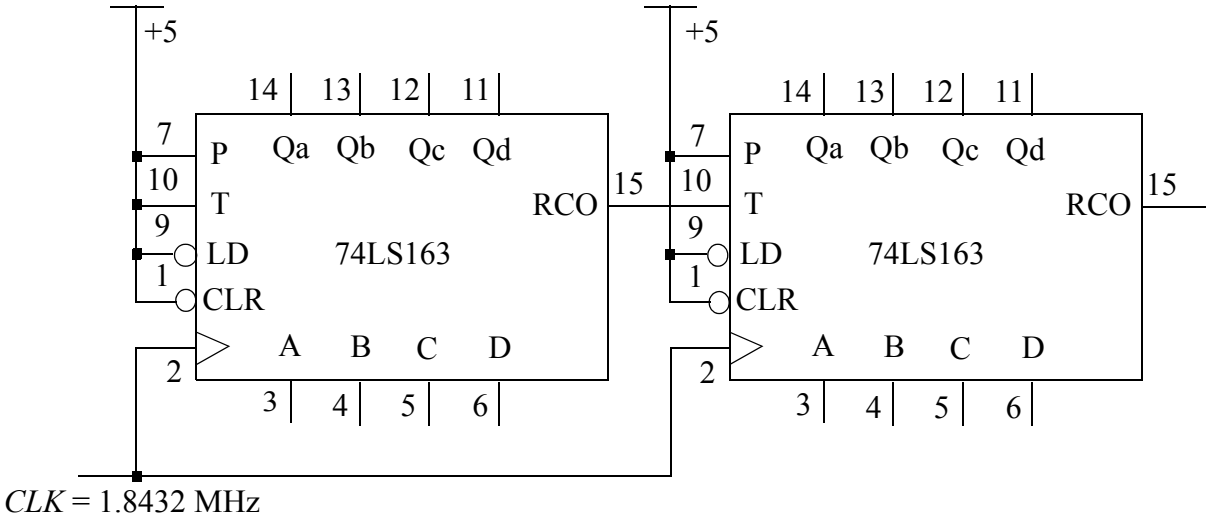

Figure 6: Synchronous Counter Wiring (RCO of the first chip is only connected to *T*, not +5).

Exercise 6: Construct a Set-Reset Latch with NAND gates

Build a clocked positive SR latch using NAND gates (use the setup shown in Figure 7). On a "set", the output Q should be high, and $\overline{Q}$ should be low. On a "reset", the output $\overline{Q}$ should be high, and Q should be low. However, since this is a clocked latch, the output can only change during the clk high phase; it is held at all other times. Explain how the SR latch works and demonstrate the functionality of the built circuit. Be prepared to explain what happens during the case when both S and R are high.

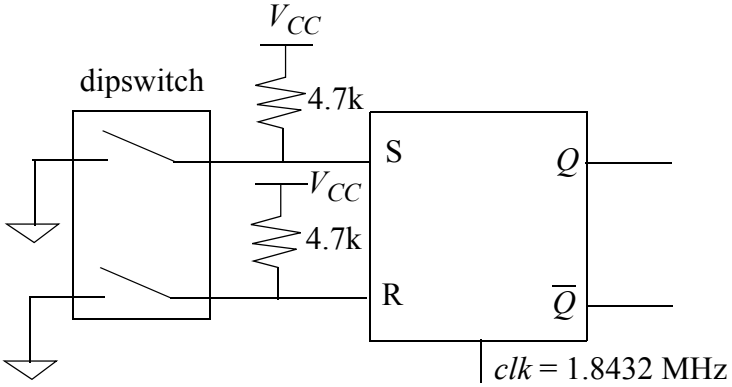

Figure 7: Setup for SR Latch.

Exercise 7: Setup Time Measurement of a *D Edge-triggered Register*

This is an open-ended problem. If you find a solution, be descriptive in your measurement methodology. If you can't find a solution, explain the different methods you did try and other possibilities that you thought may work.

Setup and hold times are important constraints to follow when implementing a digital system. With respect to a *D edge-triggered register*, explain what each of these constraints are, and explain types of failures that may occur if one of these constraints is violated. Using a 74LS74 part and any additional components necessary, develop a system that can measure the setup time of a *D register*. Remember that you need to test both when a *D register* should capture a 0 and when it should capture a 1. Be sure that the system only tests the setup time and does not change any other parameters, such as the rise and fall times of the inputs. You may use any method possible, but be sure to justify the method.

Find the published setup time on the datasheet for a 74LS74 (this number will vary from manufacturer to manufacturer and across chips) and compare to your measured time. Provide possible reasons why they might not match. Be prepared to demonstrate how you measured the setup time using a circuit diagram and an oscilloscope.

Exercise 8: Writing Combinational Verilog code (introduction to ModelSim, Xilinx Software and the Labkit)

In this exercise, you will design and implement a Verilog module that takes as input a 4-input binary coded decimal and outputs 7 bits that will properly illuminate a 7-segment display. Figure 8a (left) shows the 10 possible digits and Figure 8b shows the output mapping to segments.

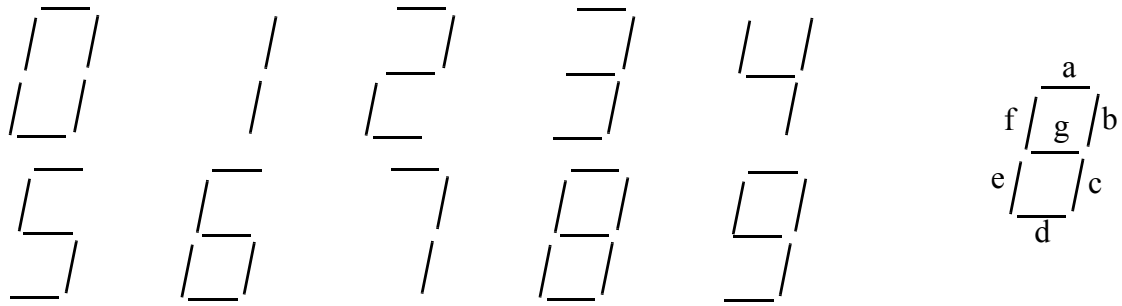

Figure 8: (a) The ten segmented digits. (b) shows the mapping of outputs a-g to segments.

Using Karnaugh Maps, design the combinational logic that encodes the 4-bit input to the 7-segment display. Fill out each Karnaugh Map, and make circles for either Minimal Sum-of-Products, or Minimal Product-of-Sums so that you minimize the required logic. Some segments will use MSP, and others will use MPS.

In Verilog, program the combinational logic. You do not have to use the minimized equations for programming the logic. Synthesis tools are powerful and will automatically optimize your logic equations. Download three source files from the course website: http://web.mit.edu/6.111/www/s2008/LABS/LAB1/{lab1_labkit.v, labkit.ucf, and seven_segment.v}. lab1_labkit.v instantiates the seven_segment conversion module as well as other circuitry to drive the dot-matrix display on the labkit (appropriate wrapper logic is included to make the dot-matrix display on the labkit look like a 7-segment display. **Please do not use the discrete 7-segment display chip given with the kit**). For this exercise, you should only modify the seven_segment.v source code and assign the correct logic function to the outputs.

Go to the labkit documentation on the course website, and follow through the 'Getting Started with ISE' and 'Programming the Labkit' tutorial. You can simulate your 7-segment decoder using ModelSim. Please read 'Simulating with ModelSim', which describes running ModelSim directly from Xilinx ISE. These tutorial sites can be found at: http://www-mtl.mit.edu/Courses/6.111/labkit/{ise.shtml, configuration.shtml, simulation.shtml}.

You should now be able to create a new Xilinx project that includes lab1_labkit.v, labkit.ucf, and seven_segment.v. Find where your 'segment_decoder' module is instantiated under lab1_labkit.v. Create a simple test bench called tb_7segment.v to verify that your 7-segment decoder works correctly (call the top level module tb_7segment). Simulate your 7-segment decoder using ModelSim. Observe the outputs of your test vectors on the Wave window. After you have verified that the logic is correct, you will compile and program the FPGA to display the ten digits. After you

load your project on the FPGA, use the low four switches as the input. Verify that the correct digit appears on the LED display.

You may also run ModelSim in stand alone mode for simulation (in general, we recommend you run ModelSim directly from the Xilinx ISE environment). Details of this can be found on the course website (http://web.mit.edu/6.111/www/s2009/handouts/ModelSim_tutorial.pdf).