Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.111 – Introductory Digital Systems Laboratory

Problem Set 3

Problem Set Issued: February 25, 2005
Problem Set Due: March 11, 2005

Problem 1: Twos Complement Multiplier

In Lecture #8 a 4x4 Twos Complement Multiplier was presented. In this problem, you will design an 8x8 Twos Complement Multiplier in Verilog, and validate your design using a testbench.

Your multiplier should take as input the two's complement numbers X [7:0] and Y [7:0], and give as output a twos complement number Z. How many bits will Z have?

Code the multiplier you designed above in Verilog and validate it using a testbench.

For this problem, turn in Verilog code for your multiplier and testbench. We also ask that you submit a screen capture of your simulation.

Problem 2: Critical Path Timing Analysis

The Figure below is the 16-bit Carry-Bypass Adder from Lecture 8 (see notes for a clear block diagram).

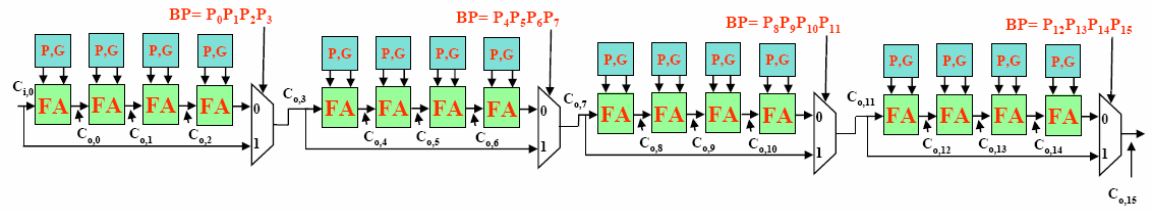

Figure 1: Carry-bypass adder

Assume the following delay for each gate:

Producing P_i, G_i from A_i, B_i: 1 unit
P_i, G_i, C_i to C_o or Sum for a FA: 1 unit
2:1 mux delay: 1 delay unit
BP: It takes 1 delay unit to generate BP from the propagate signals.

What is the worst case propagation delay for the 16-bit adder?

Problem 3: Introduction to Video

In this problem you will build part of a video controller, and use ModelSim to verify its correct operation. To get you started, a brief overview of VGA video generation follows. For additional guidance, refer to the URL:

http://web.mit.edu/6.111/www/s2004/NEWKIT/vga.shtml

To maintain a stable image on a monitor, a video controller must repeatedly output the entire contents of the screen, one pixel at a time, at the desired screen refresh rate (usually 60Hz or above). Usually this is accomplished by filling a memory with the desired screen image, and reading from the memory in a cyclic fashion.

The screen is redrawn one pixel row at a time, from left to right. Rows are drawn from top to bottom to form a complete image. To specify how quickly the image should be redrawn, displays require horizontal and vertical sync signals that pulse once per row and one per screen redraw, respectively. Thus, on the 800x600 display you will be using, the horizontal sync pulses (approximately) 600 times per vertical sync, and the vertical sync pulses (approximately) 72 times per second to specify a screen refresh rate of 72Hz. The horizontal and vertical sync are active low; their default state is a 1, and their periodic pulse is a 0.

In this video mode, one pixel is drawn every 20ns. This is another way of saying that our *pixel clock* is running at 50 MHz. After the 800 pixels in one row, we wait 56 more clock cycles before pulling the horizontal sync signal low. This pulse stays low for 120 clock cycles, and after setting it high again, we wait another 64 cycles before starting to draw the next line. The delays before and after the sync pulse are called the (horizontal) *front porch* and *back porch*, respectively. Together with the sync pulse itself, they form the *horizontal blanking period*.

After the back porch following the final row, the *vertical blanking period* begins. This begins with 37 blank lines, 1040 clock cycles each, which form the (vertical) front porch. The vertical sync signal is then brought low for 6 lines, and the 23 lines after it is reset to high, but before we start drawing the next frame, form the (vertical) back porch. With a 50 MHz clock, this leads to an approximate refresh rate of 72Hz.

Part (a) Using the timing specifications given above for 800x600 VGA video at 72Hz, write a verilog module that produces horizontal and vertical sync signals. As in Lab 2, you may find it useful to couple an FSM with one or more counters. If you take this approach, make sure your states change after exactly the right number of clock cycles; off-by-one errors will cause trouble. To facilitate this, you may wish to have your counters count upwards.

Also, use `parameter` statements to specify all of the constants in your code (porch lengths, screen dimensions, etc.). This will make it easy to test and adapt to different screen modes.

Your generator should input a `reset` signal and a 50 MHz pixel clock, and should output the two sync signals. On reset, both outputs should be set high. After reset, the generator should periodically pulse the sync signals according to the pixel clock.

Part (b) Write a testbench for your generator, and use it to verify that its behavior is correct. Be sure to set the `reset` signal a few times to verify the reset behavior. As mentioned above, you will find it useful to change the timing constants to something much smaller for simulation. Turn in screenshots of your simulation, and your final code listings.