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

Problem Set 3

Problem Set Issued: March 3, 2006
Problem Set Due: March 15, 2006

Problem 1: Critical Path Timing Analysis

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

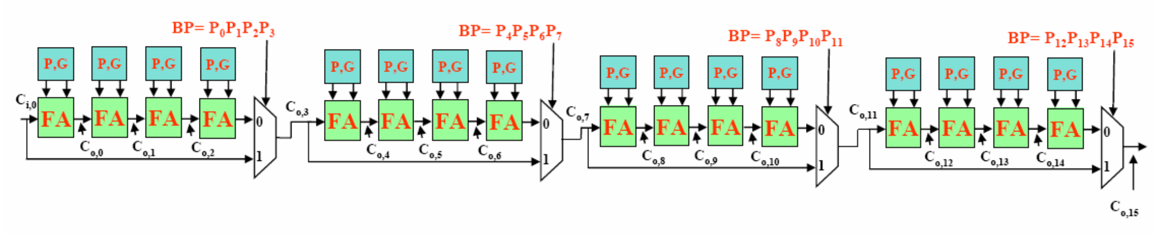

Figure 1: Carry-bypass adder

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 2: Two's Complement Multiplier

A 4x4 Two's Complement Multiplier was presented in lecture. In this problem, you will design a combinational 8x8 Two's 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 two's complement number Z. How many bits will Z have?

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

b) Implement a second twos complement multiplier in Verilog, only this time use the *signed* modifier and the * operator.

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 3: Generating Block RAMs

a) Generate a 16x16 Block RAM module using CoreGen. (Right click in the "Sources in Project" window of the Xilinx ISE and select "New Source…" When presented with options for what type of source to generate, select "IP (CoreGen & Architecture Wizard)". This will open up a core selection window. Browse through the tree structure to see all the different types of modules that can be generated. Find the option for "Single Port Block Memory" and click next, then finish. This opens a core generating wizard. Here, you can specify different parameters for the BRAM that you are going to create. All of the relevant module documentation is also available from this wizard. Adjust the width and depth parameters to specify a memory that is 16 bits wide, and uses 4 bits to address every location. Click generate to finish.

b) Design a module "**test_mem"** that writes data into one of the locations (pick any address) and then reads the data from the same location. Verify that the data were written correctly. Submit the Verilog code for your test module and a screenshot of your simulation.

Problem 4: 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://www-mtl.mit.edu/Courses/6.111/labkit/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 60 Hz 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 redraw and once per screen redraw, respectively. Thus, on the 640x480 display you will be using, the horizontal sync pulses (approximately) 480 times per vertical sync, and the vertical sync pulses (approximately) 75 times per second to specify a screen refresh rate of 75 Hz. 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 31.75 ns. This is another way of saying that our *pixel clock* is running at 31.5 MHz. You can generate this clock signal from the labkit's built-in 27MHz clock using a Digital Clock Manager (DCM) cell in the FPGA. Complete documentation for the DCM can be found in the Xilinx Libraries
Guide
(`http://toolbox.xilinx.com/docsan/xilinx7/books/docs/lib/lib.pdf`)

The FPGA you are using has several DCM's and these can be used to create signals with frequencies that are multiples of a reference signal. You can instantiate a DCM in your top level module with the following lines of code:

```
DCM pixel_clock_dcm (.CLKIN(clock_27mhz),.CLKFX(pixel_clock));
  // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
  // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
  // synthesis attribute CLKIN_PERIOD of pixel_clock_dcm is 37
  // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm is NONE
```

In this case, `clock_27mhz` is the lab kit's 27 MHz system clock and `pixel_clock` is the generated 31.5 MHz signal. The Xilinx compiler recognizes this primitive and configures one of the FPGA's DCM modules accordingly. It is worth noting that the comment lines that begin with `synthesis attribute` are actually *pseudo-comments*. That is, these lines have an effect on the synthesis of this module; they are functional Verilog code and must be included if the DCM is to function according to our specification. By multiplying the `CLKIN` signal by 7 and dividing by 6, we can generate a 31.5 MHz signal from our 27 MHz system clock.

Both the horizontal and vertical sync signals are high during active video period (this is the period of time where pixels are displayed onto the screen). After the 640 pixels in one row, we wait 16 more clock cycles before pulling the horizontal sync signal low. This signal stays low for 96 clock cycles, after which it should be set high again. We wait another 48 clock 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*.

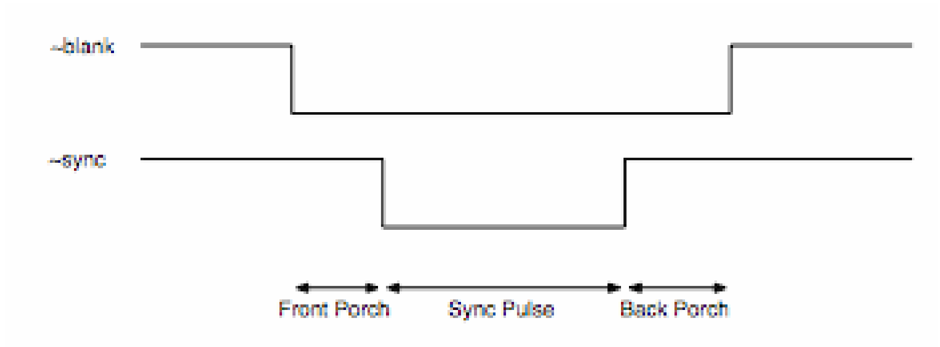

Figure 2: Generalized Timing Diagram for VGA Blank and Sync Signals

After the horizontal blanking period of the last line of pixels, the *vertical blanking period* begins. This sequence is similar to the horizontal blanking period except that this only happens once per screen refresh (i.e. every 480 lines) and the signal lengths are expressed in lines rather than pixels. The vertical blanking period has a front porch that consists of 11 lines (yes, 11*(640+16+96+48) pixels), a sync pulse that consists of 2 lines, and a back porch that is 32 lines in length. Using your 31.5 MHz clock, this leads to an approximate refresh rate of 75 Hz.

a) Using the timing specifications given above for 640x480 VGA video at 75Hz, write a verilog module that produces horizontal and vertical sync signals. As in Lab 2, you may find it useful to create 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.

Your generator should input a `reset` signal and a 31.5 MHz pixel clock, and should output the two sync signals, a pixel count, and a line count. The pixel and line counts can be used to keep track of which pixel of the screen is currently being displayed. The pixel count represents which pixel on the current line is being displayed. Similarly, the line count represents the current line that is being drawn onto the monitor. On reset, both sync signals should be set high and both counters should be zeroed. After reset, the generator should periodically pulse the sync signals according to the pixel clock and the pixel and line counts should increment appropriately.

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.

c) Let's use your new VGA video controller to draw something onto the monitor! We would like for you to instantiate your video controller, wire its inputs and outputs to the appropriate signals of your lab kit, and display a 10x10 checkerboard pattern onto the screen using two colors of your choice.

The 10x10 checkerboard pattern can be generated by using the pixel and line count outputs of your video controller.

The IC that you are interfacing with is the ADV7125. This circuit generates as output the appropriate analog signals that are needed to display red, green and blue as well as the correct blanking levels. Your module needs to provide several signals to the ADV7125 in order for it to generate a valid VGA signal: 8-bit red, green, and blue signals, a composite sync signal, a blank signal, and a pixel clock.

Send the inverted pixel clock that you implemented in part a to the ADV7125 by connecting it to the `vga_out_pixel_clock` signal in your top-level lab kit file.

The blank signal, like the horizontal and vertical sync signals, is active low. It should be pulled low whenever you want the screen to blank. In other words, the blank signal should be low during both the horizontal sync period and the vertical sync period. This can be implemented by providing the ADV7125 with the AND of the two blank signals. Connect this signal to `vga_out_blank_b`.

The composite sync signal is just the XOR of the horizontal and vertical sync signals. Connect the inverse of your composite sync signal to `vga_out_sync_b`. (connecting this signal to 1 will also work);

We mentioned that the ADV7125 is the IC that actually generates the analog RGB signals that can be used by a VGA monitor to display an image. Connect the RGB signals that generate a checkerboard pattern to the 8-bit signals `vga_out_red`, `vga_out_green`, `vga_out_blue`.

Note that the horizontal and vertical sync signals are generated directly by the FPGA, and do not pass through the ADV7125.This means that you need to provide the horizontal and vertical sync signals of your video controller as output in order to have a complete VGA signal. The last trick is that since your signals are not sent through the ADV7125, you need add a delay of two clock cycles (of your pixel clock, not your 27 MHz clock) to ensure that the sync signals are output at the same time that the corresponding RGB signals are generated by the IC. This delay is necessary because the IC is pipelined. Connect the delayed horizontal and vertical sync signals to `vga_out_hsync` and `vga_out_vsync` respectively.

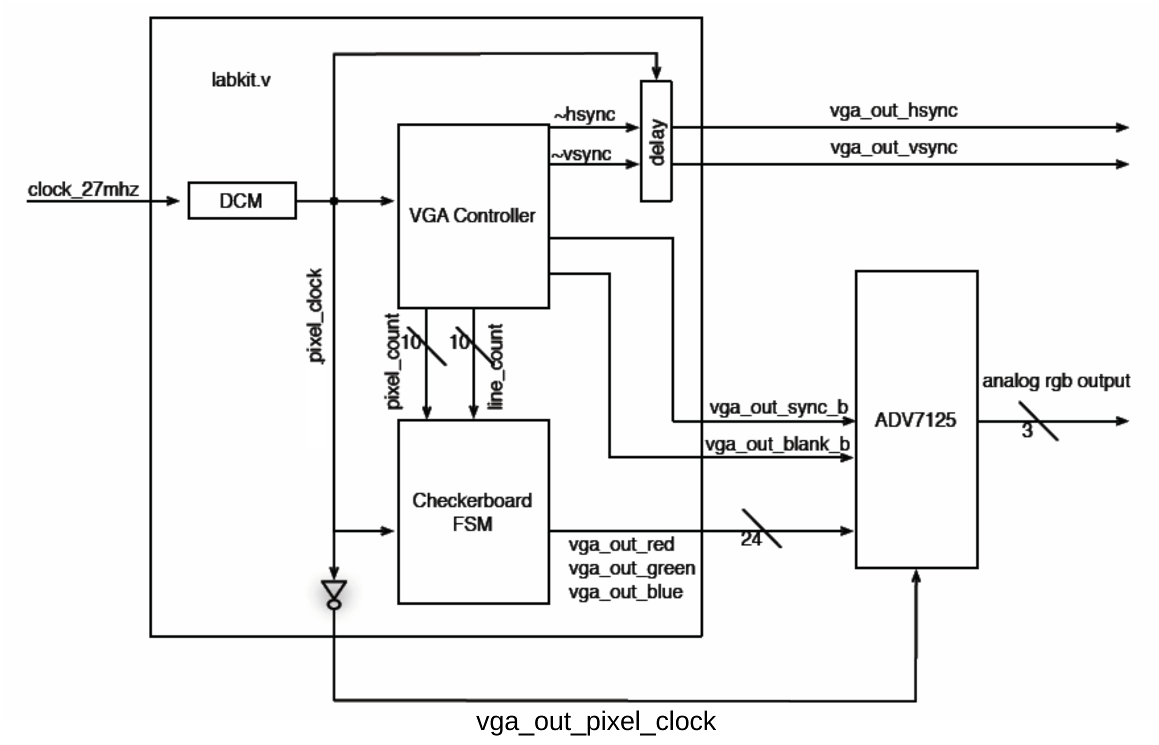

Figure 3: Block Diagram for Problem 4 part c