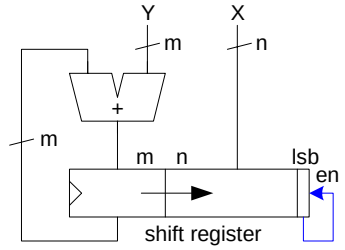
Introducton to CMOS VLSI Design (E158)

Lab 2: Multiplier Schematics

Harris

In this lab, we will design the schematics for a 32-bit serial multiply unit. You will learn to simulate the Verilog model with a self-checking test bench, then run the same model on your schematic. In the next lab, you will design the layout and assemble the unit into a chip.

The basic organization of the multiplier is shown below, with m = n = 32. The multiplier contains a 64-bit shift register and a 32-bit adder. At the start, X is loaded into the lower half of the shift register and the upper half is zeroed. On each subsequent cycle, Y is added to the upper half if the lsb is 1. In any case, the shift register shifts right by 1. After 32 steps, all 32 bits of X have been considered and shifted into the bit bucket. The shift register contains the 64-bit product X*Y. Note that X and Y are assumed to be unsigned numbers.



(a)

Step	Shift Reg	Notes
	0000|0101	initialize
0a	1100|0101	add 1*Y
0b	01100|010	shift right
1a	01100|010	add 0*Y
1b	001100|01	shift right
2a	111100|01	add 1*Y
2b	0111100|0	shift right
3a	0111100|0	add 0*Y
3b	00111100|	shift right

(b)

I. Verilog Model Synthesis and Simulation

Look in the E158/labs/lab2 directory on Charlie. You will find mult.v, multtest.v, and mult.tv. mult.v is a Verilog file describing the serial multiplier. multtest.v is a testbench, and mult.tv are some test vectors. Copy these files into your directory and rename them, adding your initials.

To understand the multiplier, you will synthesize it, simulate it, add more test vectors, and resimulate. In previous classes, you have invoked Synplify and Modelsim from Xilinx. However, these programs can be run directly and are much faster to use directly without the baggage of Xilinx, especially when you are debugging a complicated system. Moreover, for chip design, there is no reason to

Invoke Synplify Pro from the Start menu of the computer. From the File menu, create a new project called lab2syn_xx in your directory. From the Project menu, add mult_xx.v. From the options menu, choose Configure Verilog Compiler. Under the Device tab, choose Xilinx Virtex2p XC2VP30 with a speed grade of -7. Under the Constraints tab, set the frequency to 100 MHz. Under the Verilog tab, set the top level module to integrate. From the Run menu, choose Synthesize. In the bottom pane, look at the messages tab. You should see a variety of notes. There should be no warnings or errors, but if there are, fix them. Under the HDL Analyst menu, bring up the RTL view. Look at the schematic and make sure that it matches expectations.

Bring up the technology view and decipher how the schematic mapped to LUTs. Under the HDL Analyst menu, choose Show Critical Path. Browse through the path. The first number is the arrival time. The second number is the slack. As you move through the circuit, you should see the arrival time increasing.

Next, simulate your design. Invoke ModelSim from the Start menu. From the File menu, create a new project named lab2sim_xx in your directory. Add mult_xx.v and multtest_xx.v. From the Compile menu, choose Compile All. ModelSim and Synplify have different Verilog parsers, so it is not unusual to find a few more warnings or errors in ModelSim to fix if you had been writing the module from scratch.

Once your code compiles correctly, choose Simulate * Start Simulation. You will be prompted to choose which unit to simulate. Expand work, then scroll down and select multtest. The simulator will start and some windows will open so that you can explore your design. Select View * Debug Windows * Wave to open a waveform viewer; some of the other debug windows may be helpful too. In the objects pane is a list of the signals in the multtest module. Select them all and drag them into the waveform pane so that you can watch them during simulation.

It is possible to drive the inputs directly by hand. For example, you could (but shouldn't actually) type the following commands into the transcript pane at the bottom of the screen.

```
force reset 1
run 100
```

However, toggling a two-phase clock 32 times is very tedious. Moreover, checking your results by hand is tedious, especially if you have to rerun the simulation later after you have made changes. It is better to use a self-checking test bench that automatically applies the inputs and checks the results. Read through the multtest module and mult.tv test vector file to see how this works. Type `run 10000` to run through all of the test vectors. You should see that no errors are found. Learn to scroll in and out. Figure out how to display signals in binary, decimal, and hexadecimal form.

Make a habit of looking at the messages in the transcript window and learning what is normal. Warnings and errors should be taken seriously; they usually indicate real problems that will catch you later if you don't fix them.

The test vector set is rather skimpy. Add at least 5 nontrivial vectors of your own to more convincingly exercise the multiplier. Include some large numbers (bigger than 0x80000000).

To restart the simulation on your new vectors, type `restart` (or better yet, `restart -f` to avoid the annoying dialog box). Run the simulation. Check that your new vectors were run and that there are no errors.

You shouldn't have to correct the code, but if you did, you should choose Compile All again. Then type restart to begin the simulation again with the corrected code; there is no need to exit the simulator as you would have to in Xilinx.

II. Wordslice Design

Part of the multiplier schematics are already complete. Your objective in this lab is to finish the remainder and get it to simulate.

Copy the muddlib and mult.elib files from the Lab2 directory to your local directory. Rename mult.elib to mult_xx.elib. Look at mult{sch} and compare it to the Verilog. They should match up except that the prodlmux is missing in the schematic.

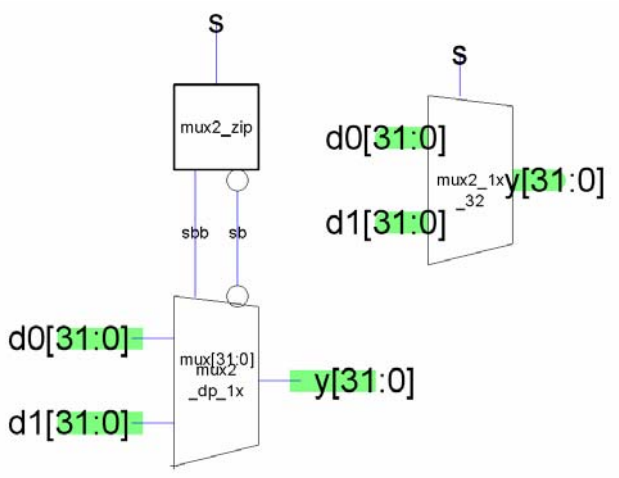
The Verilog and schematic contain functional units organized as 32-bit word slices. This is a convenient way to group things together. The units can be connected together with wide busses, which is much simpler than drawing 32 wires. Recall that datapath cells factor out the inverters from select, clock, and enable signals to improve density. These inverters are placed in a zipper at the top of the word slice so that they can drive the entire slice.

To see how a word slice is created, select one of the 32-bit flopenr_1x_32 (flip-flops with enable and reset) and choose Cell * Down Hierarchy (Ctrl-D). Observe that it is formed from a single flip-flop named flopenr[31:0] to create 32 copies without having to draw each one. This part of the cell is called the datapath. The input and output are connected to 32-bit busses. The cell also has a zipper, made of inverters and buffers factored out of the cell. In this cell, there is an inverter and buffer to drive the enable signal, an inverter to drive reset, and a pair of inverters and buffers to drive the two-phase clocks. The gates in the zipper are typically 4x normal size so that they can drive the entire wordslice in a timely fashion.

Also, look at the 32-bit adder. It is constructed from 32 full adders. Notice how the comma notation is used for the carry in and carry out signals. This is much faster to draw than 32 separate full adders chained together.

Your first step is to design a word slice for a 32-bit 2:1 multiplexer. Unless otherwise stated, use 1x cells in word slices.

Create a new schematic called mux2_1x_32. When it is all done, it should look something like the one below.



First instantiate a mux2_dp_1x{ic} from muddlib, either by choosing Cell * New Cell Instance or by dragging the icon from the Explorer pane on the left. Double-click on the icon to edit the properties. Change the name to mux[31:0] to create 32 copies.

In the components pane, click on the wide green bus symbol at the bottom. Draw busses for d0, d1, and y. Export the inputs and outputs as 32-bit signals.

Next, create the mux2_zip cell. It should contain two inv_4x cells. It takes a single input, s, and produces two outputs, s_out and sb_out, buffered and inverted versions of the input, respectively.

Create an icon for the mux2_zip. Instantiate it in the mux2_1x_32. Connect the select and select_bar lines. It is a good practice to label all signals in a schematic; for example, they may be called sbb and sb, respectively. Attach a wire to the s input of the zipper and export it as s. Create an attractive icon for the entire mux2_1x_32.

Instantiate the mux2_1x_32 in the mult schematic. Connect the inputs and outputs as specified by the Verilog. You will need comma notation to specify one of the inputs as a swizzle of bits; look at the prodh register for an example of this notation.

When you are done, it is a good idea to call Edit * Cleanup Cell * Cleanup Pins Everywhere to delete any stray pins that you may have created. Then run DRC on the schematic; it sometimes catches issues with naming.

III. Controller Design

The multcon cell has an icon but no schematic. Create the schematic, using the skills you have just learned. The schematic should match the Verilog. You may wish to create new cells for the 6-bit register and incrementer.

You will need a source of 0 and 1. To get these, place power and ground symbols, attach short wires, and label the wires zero and one. The mult{sch} has an example of a source of 0.

IV. Schematic Simulation

Choose Tools * Simulation(Verilog) * Write Verilog Deck to write a Verilog netlist of the entire design. Open it in ModelSim. Look through the file and see how it matches with the schematic. nMOS and pMOS transistors are netlisted as tranif1 and tranif0, respectively. Pay special attention to convince yourself that your multiplexer and multcon cells look correct.

Create a ModelSim project containing your mult.v netlist and the multtest.v test fixture. Compile the design and fix any errors. Start a simulation on multtest, just like you did on the original Verilog. Run until the simulation and check if there are errors. If you find errors, fix the schematic, generate a new netlist, recompile it, and restart the simulation. When you debug, you may find it helpful to use the Objects, Workspace(sim) and wave windows to add signals in the blocks that you have designed so that you can watch them during simulation.

What To Turn In

1) Your new mult.tv file containing a better set of test vectors.
2) Schematics for your mux2_zip, mux2_1x_32, and multcon cells.
3) Does the Verilog netlist of mult{sch} pass all of the test vectors?