CMOS VLSI Design
Lab 2: Datapath Design and Verification

In this lab, you will begin designing an 8-bit MIPS processor. You will learn about datapath design by assembling and connecting wordslices into an ALU. As with all labs, read the whole writeup thoroughly before starting to avoid surprises.

I. Verilog Model RTL Simulation

In the lab directory find mips.sv and memfile.dat. Copy these files into your directory and rename them, adding your initials. mips.sv is System Verilog RTL for the 8-bit MIPS processer and memfile.dat contains test vectors. The processor is detailed in Chapter 1 of CMOS VLSI. The testbench for the processor is different from the previous lab.

Instead of testbench applying and asserting vectors, the external memory module exmemory loads a test program stored in memfile.dat. The program tests basic functionality of the processor and, if successful, writes a 7 to memory address 0x4C. testbench checks that the processor wrote the success value. The program is shown below; study it to see what it does.

```
# mipstest.asm
# 9/16/03 David Harris David_Harris@hmc.edu
#
# Test MIPS instructions.  Assumes little-endian memory was
# initialized as:
# word 16: 3
# word 17: 5
# word 18: 12

main:   #Assembly Code        effect                          Machine Code
        lb $2, 68($0)         # initialize $2 = 5             80020044
        lb $7, 64($0)         # initialize $7 = 3             80070040
        lb $3, 69($7)         # initialize $3 = 12            80e30045
        or $4, $7, $2         # $4 <= 3 or 5 = 7              00e22025
        and $5, $3, $4        # $5 <= 12 and 7 = 4            00642824
        add $5, $5, $4        # $5 <= 4 + 7 = 11              00a42820
        beq $5, $7, end       # shouldn't be taken            10a70008
        slt $6, $3, $4        # $6 <= 12 < 7 = 0              0064302a
        beq $6, $0, around    # should be taken               10c00001
        lb $5, 0($0)          # shouldn't happen              80050000
around:slt $6, $7, $2         # $6 <= 3 < 5 = 1               00e2302a
        add $7, $6, $5        # $7 <= 1 + 11 = 12             00c53820
        sub $7, $7, $2        # $7 <= 12 - 5 = 7              00e23822
        j end                 # should be taken               0800000f
        lb $7, 0($0)          # shouldn't happen              80070000
end:    sb $7, 71($2)         # write adr 76 <= 7             a0470047
        .dw 3                                                 00000003
        .dw 5                                                 00000005
        .dw 12                                                0000000c
```

Read through the testbench and exmemory modules and memfile.dat to see how the RTL works. Compile and simulate it. You should see `Simulation completely successful` if the RTL is working.

II. Library Organization

The MIPS processor design is split between multiple library files included in the lab directory: mips8.jelib, muddlib07.jelib, wordlib8.jelib, and muddpads13_ami05.jelib. Rename mips8.jelib to mips8_xx.jelib. Opening mips8_xx.jelib, which contains the main components of the 8-bit processor, opens all of the libraries because the processor depends on cells in the other libraries. You can select a different library when editing or creating a cell with the library dropdown in the respective dialogs. muddlib07 is a standard cell library. wordlib8 contains common 8-bit wordslices. Muddpads13_ami05.jelib contains version 1.3 of the input/output pad cells for the AMI 0.5 μm process. If prompted that the new library uses different project settings, click Use All New Settings.

III. Wordslices

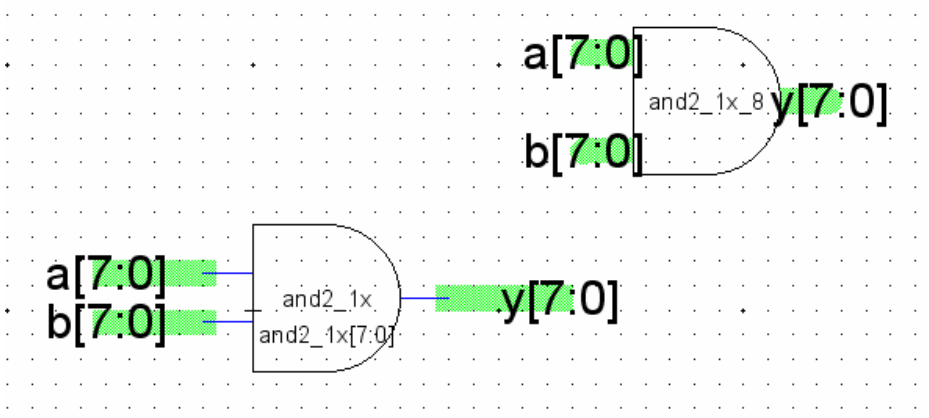
The Verilog and schematic contain functional units organized as 8-bit *wordslices*. This is a convenient way to group cells together. Wordslices can be connected with busses, which is much simpler than drawing eight separate wires. To see how a wordslice is created, select the 8-bit flopenr_1x_8{sch} (flip-flops with enable and reset) in wordlib8.jelib. Observe that it is formed from an array of eight flip-flops named flopenr_8[7:0] without having to draw each one. This part of the cell is called the datapath. Inputs and outputs are connected to 8-bit busses.

Datapath cells can factor out the inverters from select, clock, and enable signals because it is more efficient to place one inverter at the top of the datapath than one in each bit cell. These inverters are placed in a zipper at the top of the wordslice so that they can drive the entire slice. flopenr_1x_8 also has a zipper, made of inverters and buffers factored out of the individual one-bit flopenr_1x cells. 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 8-bit adder8{sch}, which is constructed from 8 full adders. Notice how the comma notation is used for the carry in and carry out signals in the schematic. This is much easier to draw than 8 separate full adders chained together.

The ALU includes an AND, OR, adder, and SLT. Your first step is to design a wordslice for an 8-bit AND that will be used in the ALU unit. Later in this lab you will design an 8-bit OR and hook the two up to the ALU, and then the ALU to the datapath. The two cells you will create do not have zippers because there are no circuits to factor out.

Create a new schematic called and2_1x_8 in wordlib8.jelib. When it is all done, it should look like the one below. Unless otherwise stated, use 1x cells in wordslices.



First instantiate an and2_ 1x{ic} from muddlib, either by choosing Cell • Place 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 and2_1x[7:0] to create 8 copies. In the components pane, click on the wide green bus symbol at the bottom. Draw busses for a, b, and y. Export the inputs and outputs as 8-bit signals. Create an attractive icon for the cell. Run DRC on the schematic; it sometimes catches errors with naming.

Once the schematic is finished, create a layout for the and2_1x_8. Place a single and2_1x{lay} within the cell. Remember to place the cell center two λ right from the left edge of the bottom GND rail. Rename the cell to and2_1x[0]. Add exports y[0], a[0], and b[0].

Instead of tediously placing eight and2_1x cells manually, you can use Electric's array and mimic-stich functions to speed repetitive layout. The array function produces multiple regularly spaced copies of a cell. Configure the array function by going to File • Preferences • General • Nodes, select "Duplicate/Array/Paste Copies Exports," and click OK to save the changes. Bit addresses within a bus will automatically increment, e.g. exports or names such as and2_1x[0] will be incremented to and2_1x[1], and2_1x[2], etc… Mimic-stitch, which will be used later in this lab, replicates the last wire drawn.

To make an array, click on the and2 cell to select it. Press F6 or go to Edit • Array to open the Array dialog box. Change Y repeat factor to 8, which is the number of repeated cells. Select "space by centerline distance" and enter 110 into Y centerline distance and 0 for X centerline distance. All other options should be left at their default. Click OK to finish the array. Zoom out and you should see eight and2 cells arranged vertically, spaced by 110 λ. Zoom in and verify the exports names are autoincremented. Export the power and ground rails with Export • Re-Export Power and Ground; you should see a message that 16 ports are exported (for the 8 powers and grounds).

Lastly, verify your design with DRC, NCC, and ERC. NCC should fail because of the multiple power and ground rails, which it expects to be connected. Since they will be connected later in a higher-level cell, click on Tool • NCC • Add NCC Annotation to Cell • Exports Connected by Parent vdd and Exports Connected by Parent gnd. Look to see that these annotation messages are added near the cell center. Recheck NCC and it should pass.

IV. OR Wordslice

Now that you know how to create a wordslice, design a schematic, icon, and layout for an 8-bit OR wordslice named or2_1x_8 using or2_1x cells. Verify that your design passes DRC, NCC, and ERC.

V. ALU Assembly

Open the alu{sch} cell in mips8. You'll see named buses for the inputs and outputs of the 8-bit AND/OR cells. Place and connect each. Electric treats excess pins as errors. Use Edit • Cleanup Cells • Cleanup Pins to clean up the excess pins and check the schematic with DRC.

Next, you will complete the alu{lay}. Use Cell * Expand Cell Instances * All the Way to see the details. You will see a space in the middle for the AND/OR wordslices. Place the AND wordslice on the left and the OR wordslice on the right.

Metal3-metal2 contacts are already provided to connect the wordslices to the bit lines. Align the cells and verify that the metal2-metal1 contacts are on an 8 λ pitch. Mimic stitch is handy to quickly connect, or stitch, the wordslices. It works by matching ports with names, sizes, and other properties similar to the last stitch. It is activated when you press F1 or go to Tool • Routing • Mimic-Stitch Now. You can modify the properties it matches in File • Preferences • Tools • Routing • Mimic Stitcher. There you can set certain restrictions about which connections are stitched. You can also enable interactive mimic stitching, which prompts before executing a set of stitches. For now, check the following while leaving the other options unchecked:

- No stitcher running
- Interactive mimicking
- Bus ports must have same width
- Node types must match
- Ignore if already connected elsewhere

First, connect the ground and power rails of the new cells to the existing cells. After connecting one wire, press F1 to invoke the mimic stitcher. Some of the stitches it suggests will be wrong, so only allow stitches for the power and ground rails. This takes some practice; save before you try it, and use the undo command if you don't like what happened. After adding all of the power and ground rails, run DRC and there should 8

errors all related to a metal3-metal2 via too close to a metal2-metal1 via, which you will soon connect. If you have more, check where they are and fix them.

Next, connect the wordslice exports to the metal3-metal2 vias that are attached to the metal3 bitlines (running horizontally). This is easiest to do by selecting the metal3-metal2 via, then clicking near the input or output on the gate. When multiple things are stacked up, hold the Ctrl key while clicking to successively select different items in the stack. Mimic-stitch the other seven copies of each connection. Again, some of its suggested stitches will be wrong. Use caution: it is quicker to add wires with mimic stitch than it is to delete them manually.

Verify the layout passes DRC, NCC, and ERC.

VI. Datapath Assembly

Electric includes an auto-stitcher, which will connect pins, wires, and ports, when they are placed on top of each other. Open datapath{lay} in mip8 and place the ALU near the end of the datapath. Align the ALU over the existing wires, check that the ALU is highlighted and press F2 (or go to Tool • Routing • Auto-stich Highlighted Now). Leave an extra 8 λ spacing between the ALU and the flopenr wordslice immediately to the left of it, to accommodate the extra wiring track of the zero detector.

After auto-stich completes, the layout should pass DRC, NCC, and ERC. Generate a Verilog deck and simulate it using the same MIPS testbench; check that you get the same results as from simulating the RTL. Note that the layout uses a two-phase clock but the RTL does not. Use 'clk' for the ph1 input of datapath and '~clk' (invert clk) for ph2. Do this in future labs as well. Adjust the inputs and outputs in the datapath module instantiation in the mips module as necessary to match the Verilog produced by Electric. Also remember that the original Verilog for the datapath used parameters, while the version from Electric does not. Therefore, you need to remove the #(WIDTH, REGBITS) instance parameters from the module instantiation.

VII. What to Turn In

Please provide a hard copy of each of the following items:

1. Please indicate how many hours you spent on this lab. This will not affect your grade, but will be helpful for calibrating the workload for the future.
2. A printout of your 8-bit AND wordslice schematic and layout.
3. A printout of your 8-bit OR wordslice schematic and layout.
4. A printout of your ALU schematic and layout
5. A printout of your datapath layout
6. What is the verification status of your layout? Does it pass DRC? ERC? NCC? Does it simulate successfully against the RTL?