***CMOS VLSI Design Course***

**LAB 02**

**Datapath Design and Verification**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc48269483)

[1.1 Lab overview 1](#_Toc48269484)

[2 Learning Objectives 1](#_Toc48269485)

[3 SystemVerilog Model RTL Simulation 1](#_Toc48269486)

[4 Library Organization 2](#_Toc48269487)

[4.1 Add Libraries to library path 2](#_Toc48269488)

[5 Wordslices 3](#_Toc48269489)

[5.1 AND Wordslice 4](#_Toc48269490)

[5.1.1 Create a new schematic called and2\_1x\_8 in wordlib8 4](#_Toc48269491)

[5.1.2 Create a symbol for *and2\_1x\_8* 4](#_Toc48269492)

[5.1.3 Create a layout for *and2\_1x\_8* 5](#_Toc48269493)

[5.2 OR Wordslice 7](#_Toc48269494)

[6 ALU Assembly 7](#_Toc48269495)

[6.1 Add AND/OR wordslices schematic to ALU 7](#_Toc48269496)

[6.2 Add AND/OR wordslices layout to ALU 7](#_Toc48269497)

[6.3 Connect wordslices inputs and outputs 7](#_Toc48269498)

[7 Datapath Assembly 8](#_Toc48269499)

[8 Datapath Simulation 8](#_Toc48269500)

[8.1 Generate and edit datapath netlist 8](#_Toc48269501)

[8.2 Simulate datapath using NC-Verilog 10](#_Toc48269502)

[9 What to Turn In 10](#_Toc48269503)

# Introduction

## Lab overview

In this lab, you will begin designing a simplified 8-bit microprocessor. If you have not already read about the processor in Section 2 of the Getting Started Guide, do so now. You will review and simulate a Verilog model of the overall processor. You will learn about datapath design by assembling and connecting wordslices into an ALU.

# Learning Objectives

At the end of this lab, you should be able to:

* Simulate a SystemVerilog RTL model in NC-Verilog
* Draw schematic and layout for a wordslice in Cadence Virtuoso
* Assemble a datapath including the wordslice
* Verify the blocks with simulation, DRC, and LVS

# SystemVerilog Model RTL Simulation

In the **/courses/cmosvlsi/20/lab2** directory, find **processor\_multi.sv, memfile.s,** and **memfile.dat**.

* Make a subdirectory in your IC\_CAD directory.
* Copy these files into your subdirectory and rename them, adding your initials.

The file **processor\_multi.sv** is a System Verilog Register Transfer Level (RTL) code for the 8-bit microprocessor. The file **memfile.dat** contains test vectors. The file **memfile.s** is the human-readable version of memfile.dat.

|  |  |
| --- | --- |
| Lights On | The testbench for the processor is different from the previous lab. Instead of the testbench applying and asserting vectors, the external memory module mem 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 100. The testbench checks the memory address 100 to ensure the success value 7 is written by the processor. The program is shown in Figure 1 below; study it to see what it does.

; Assembly Instruction Comment Machine Language Code (Binary and Hexadecimal) Addr

MAIN SUB R0, R15, R15 ; R0 = 0 1110 000 0010 0 1111 0000 0000 0000 1111 E04F000F 0x00

ADD R2, R0, #5 ; R2 = 5 1110 001 0100 0 0000 0010 0000 0000 0101 E2802005 0x04

ADD R3, R0, #12 ; R3 = 12 1110 001 0100 0 0000 0011 0000 0000 1100 E280300C 0x08

SUB R6, R3, #9 ; R6 = 3 1110 001 0010 0 0011 0110 0000 0000 1001 E2436009 0x0c

ORR R4, R6, R2 ; R4 = 3 OR 5 = 7 1110 000 1100 0 0110 0100 0000 0000 0010 E1864002 0x10

AND R5, R3, R4 ; R5 = 12 AND 7 = 4 1110 000 0000 0 0011 0101 0000 0000 0100 E0035004 0x14

ADD R5, R5, R4 ; R5 = 4 + 7 = 11 1110 000 0100 0 0101 0101 0000 0000 0100 E0855004 0x18

SUBS R1, R5, R6 ; R1 <= 11 – 3 = 8, set Flags 1110 000 0010 1 0101 1000 0000 0000 0110 E0558006 0x1c

BEQ END ; shouldn’t be taken 0000 101 0000 0 0000 0000 0000 0000 1100 0A00000C 0x20

SUBS R1, R3, R4 ; R1 = 12 – 7 = 5 1110 000 0010 1 0011 0001 0000 0000 0100 E0531004 0x24

BHI AROUND ; should be taken 1010 101 0000 0 0000 0000 0000 0000 0000 8A000000 0x28

ADD R5, R0, #0 ; should be skipped 1110 001 0100 0 0000 0101 0000 0000 0000 E2805000 0x2c

AROUND SUBS R1, R6, R2 ; R1 = 3 – 5 = -2, set Flags 1110 000 0010 1 0110 0001 0000 0000 0010 E0561002 0x30

ADDLS R6, R5, #1 ; R6 = 11 + 1 = 12 1011 001 0100 0 0101 0110 0000 0000 0001 92856001 0x34

SUB R6, R6, R2 ; R6 = 12 – 5 = 7 1110 000 0010 0 0110 0110 0000 0000 0010 E0466002 0x38

STR R6, [R3, #84] ; mem[12+84] = 7 1110 010 1100 0 0011 0110 0000 0101 0100 E5836054 0x3c

LDR R2, [R0, #96] ; R2 = mem[96] = 7 1110 010 1100 1 0000 0010 0000 0110 0000 E5902060 0x40

ADD R15, R15, R0 ; PC <- PC + 8 (skips next) 1110 000 0100 0 1111 1111 0000 0000 0000 E08FF000 0x44

ADD R2, R0, #14 ; shouldn’t happen 1110 001 0100 0 0000 0010 0000 0000 0001 E280200E 0x48

B END ; always taken 1110 101 0000 0 0000 0000 0000 0000 0001 EA000001 0x4c

ADD R2, R0, #13 ; shouldn’t happen 1110 001 0100 0 0000 0010 0000 0000 0001 E280200D 0x50

ADD R2, R0, #10 ; shouldn’t happen 1110 001 0100 0 0000 0010 0000 0000 0001 E280200A 0x54

END STR R2, [R0, #100] ; mem[100] = 7 1110 010 1100 0 0000 0010 0000 0101 0100 E5802064 0x58

*Figure 1: memfile.s*

Read through the testbench and mem modules and **memfile.s** to see how the RTL works. Compile and simulate it by invoking

**sim-nc processor\_multi.sv.**

You should see ***Simulation completed successfully*** if the RTL is working. You may wish to run the simulation with the GUI (sim-ncg) and watch the top-level signals to observe the processor executing the program.

# Library Organization

The microprocessor uses an assortment of libraries: **muddlib11, wordlib8,** and **processor8**.

* **muddlib11** is the 2011 release of a simple standard cell library from Harvey Mudd College.
* **wordlib8** contains 8-bit wordslices used in the microprocessor datapath and potentially other datapaths.
* **processor8** contains cells unique to the 8-bit microprocessor processor.

Copy these libraries to your own directory so you have your own working versions to edit. (Just create a symbolic link for **muddlib11** because it can be read-only.)

From your cadence directory, enter the following commands

**ln -s /courses/cmosvlsi/20/lab2/muddlib11 .**

**cp –r /courses/cmosvlsi/20/lab2/wordlib8 .**

**cp –r /courses/cmosvlsi/20/lab2/processor8 .**

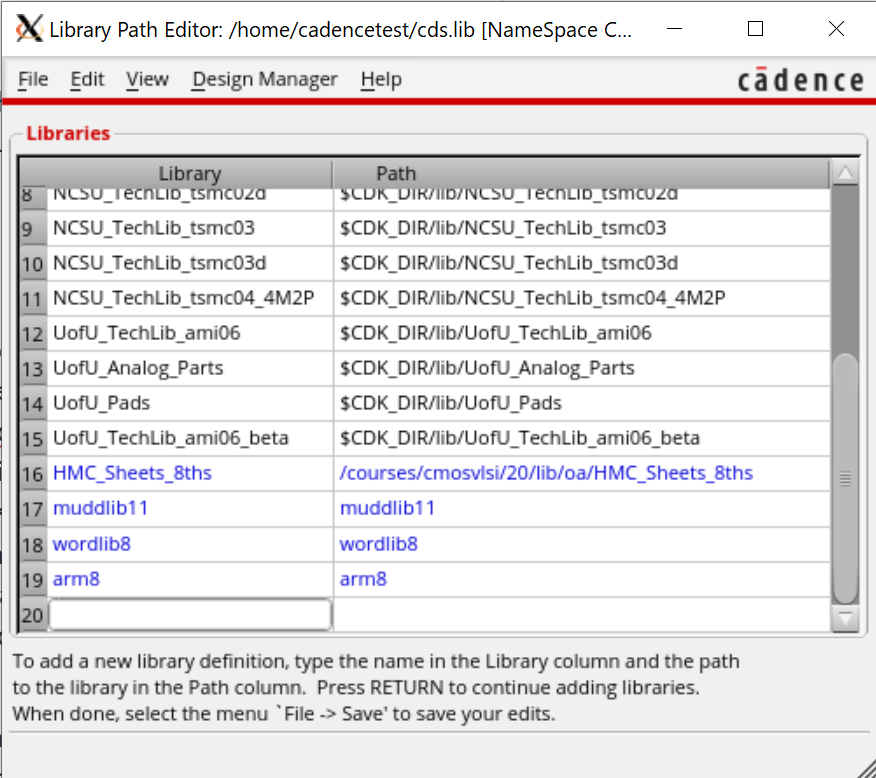
## Add Libraries to library path

Add these libraries to your library path by choosing

* Open you Library Manager window.
* **Go to:** Edit • Library Path in the.
* In a blank row at the bottom of the Library Path Editor, **enter**: **muddlib11** **muddlib11** as shown in Figure 2.
* Do the same for wordlib8 and processor8.
* Use File • Save to save the new path in your cds.lib file.

|  |  |
| --- | --- |
| Lights On | You could have also made these changes by editing the file directly. |

You should now be able to scroll down and see the new libraries in the Library Manager.



*Figure 2: Library Path Editor*

# 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, open the 8-bit flopenr\_1x\_8 (flip-flips with enable and reset) schematic in wordlib8. Observe that it is formed from an array of eight flip-flops named flopenr\_dp\_1x<7:0> without having to draw each one. This part of the cell is called the datapath. Inputs and outputs (d<7:0> and q<7:0>) are connected to 8-bit busses. For clarity, the busses are drawn with wide wires.

|  |  |
| --- | --- |
| Lights On | Cadence uses angle brackets (< >) rather than square brackets ([ ]) to represent busses and arrays |

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 4× normal size so that they can drive the entire wordslice in a timely fashion.

Also, look at the 8-bit adder8 schematic, 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, and adder. 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 in the datapath. The two cells you will create do not have zippers because there are no circuits to factor out.

## AND Wordslice

### Create a new schematic called and2\_1x\_8 in wordlib8

|  |  |
| --- | --- |
| Lights On | When it is all done, it should look like Figure 1. Unless otherwise stated, use 1x cells in wordslices. |



Figure 3: and2\_1x\_8 schematic

* First instantiate an **and2\_1x** from **muddlib11**.
* **Go to**: Edit • Properties • Object… (q) and change the Instance Name to and2\_1x<7:0> to create 8 copies.
* Create the input pins (a<7:0> and b<7:0>) and output pins (y<7:0>).
* **Go to**: Add • Wire (wide) (shift + w) and draw busses between the pins and the gate.
* The schematic should look like Figure 3. Check and save and ensure you have no errors.

### Create a symbol for *and2\_1x\_8*

The easiest approach is to use the Library Manager to copy the symbol from the and2\_1x in muddlib11 to the and2\_1x\_8 in wordlib8. Ignore any complaints about the prop.xx file.

#### Edit the new symbol

* **Go to**: Edit • Properties • Object to select each of the three pins and modify its name by adding <7:0> (e.g., changing a to a<7:0>).

|  |  |
| --- | --- |
| Lights On | Make sure you are editing the pin and not just the label next to the pin. Move the label to a good position. |

* Also edit the properties of each of the lines and modify the width to wide.

When you are done, the symbol should look like Figure 4. Check and save.



Figure 4: and2\_1x\_8 symbol

### Create a layout for *and2\_1x\_8*

Once the schematic is finished, create the layout for the and2\_1x\_8.

#### Add instance of and2\_1x\_8

* Press “i" to add an instance.
* Browse to select the **and2\_1x** layout from **muddlib11**.
* In the “Mosaic” area, change the number of rows to 8 and then click somewhere else in the window, “View,” for example, to make the change register.
* Then, change the delta Y to 33 so that each row is spaced 110 λ (33 μm) apart.
* Now place the instance the same way you would have placed a single cell.

There should now be eight copies of the and2\_1x layout that are all part of a single instance. Use Options • Display (e) and set the Stop Levels so that you can see the contents of the wordslice.

#### Create input and output pins

Next, create the pins.

* Select metal2 in the LSW.
* Open the create pin window (hotkey “ctrl+p”)
* Under Terminal Names, enter “a<0:7>”.

|  |  |
| --- | --- |
| Lights On | The order is reversed from usual (<7:0>) because we will be placing bit 0 at the bottom. |

* Select something else in the window to make the change register; for instance, make sure the I/O type is input and that “Create Label” is checked.

The boxes next to “X Pitch” and “Y Pitch” should no longer be grayed out.

* Change the Y Pitch to 33.
* Make sure metal2 is selected and draw the pin over the “a” pin on the bottom and2\_1x gate.

When you are placing the pin, it should display as “a<0:7>”, and once you’ve placed it, you should see the name change to just “a<0>”. Look at the other and gates, and you should see that the pins “a<1>” through “a<7>” were automatically placed over the other “a” pins.

Place the b and y pins. Make sure all the settings are correct before placing the pins. Immediately check that the pins were placed correctly because if they were not, or the I/O direction was wrong or the wrong metal was used, the easiest way to get rid of the pins is to “undo” (hotkey “u”). Otherwise, you will have to manually delete each pin.

#### Create power pins

* Now place a metal1 gnd! pin in the bottom row over the entire ground wire.
* Remember that gnd! and vdd! are **inputOutput** pins.

|  |  |
| --- | --- |
| Lights On | Since ground is not a bus, you will not be able to use the Y Pitch in the Create Pin. Instead you will make seven copies of the gnd! pin. |

To make copies of the gnd! pin, on the command interpreter window (CIW) that is Virtuoso window.

* **Go to**: Options • User Preferences and check “Options Displayed When Command Start” box.

|  |  |
| --- | --- |
| Lights On | You may not always want that option checked; however, it is necessary for this step. |

* Click on “Apply” or “OK” and go back to the layout.
* Select the gnd! pin you just created.
* Now copy the pin (hotkey “c”) and you should see a Copy window open.
* In the Copy window, make sure Snap Mode is set to “orthogonal.”
* Change Copies/step to 7.
* Set Delta X to 0, Delta Y to 33, and click Apply twice.

The gnd! pin should have been copied onto the seven other AND gates. Next, create the vdd! pins. When you are done, the layout should look something like Figure 5.



Figure 5: and2\_1x\_8 layout

#### Verify the layout

Finally, run DRC and Extract.

* Be sure to check “**Join Nets with Same Name**” option in the DRC and extract windows.
* Also run LVS.

|  |  |
| --- | --- |
| Lights On | Be careful when using “**Join Nets with Same Name**” option because while it is necessary for wordslices that are intentionally disconnected, it can hide disconnects in the final layouts. |

We will provide power and ground rings in Lab 4 that connect the supplies and make the “Join Nets with Same Name” option unnecessary.

## OR Wordslice

Now that you know how to create a wordslice, design a **schematic, symbol,** and **layout** for an **8-bit OR** wordslice named **or2\_1x\_8** using **or2\_1x** cells. Verify that your design passes DRC and LVS.

# ALU Assembly

## Add AND/OR wordslices schematic to ALU

Open the **alu schematic** in **processor8**. You’ll see named busses for the inputs and outputs of the **8-bit AND/OR cells**.

* Place and connect each. Check and save.

## Add AND/OR wordslices layout to ALU

Next, you will complete the **alu** **layout**. Change your display options so you can see all of the cells. Study the layout until you can relate it to the schematic.

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**.

## Connect wordslices inputs and outputs

The alu already has metal3 bitlines for a, b, andresult, and orresult in each of the 8 bitslices. It also has via2s conveniently located so that you can connect the inputs and outputs of the and2\_1x\_8 and or2\_1x\_8 gates to the bitlines using vertical metal2 wires. If you’ve placed your two wordslices properly, all you need to do is add six vertical metal2 wires in each bitslice to connect the wordslices to the metal3.

|  |  |
| --- | --- |
| Lights On | Even though **a** and **b** might seem symmetric in the schematic, be sure to connect the proper bitline to the proper input or LVS will complain. |

* Make sure to connect the **and2\_1x\_8** output to the **metal3** wire connected to the **d0 input** of the **mux3**.
* the ***or2\_1x\_8*** output to the ***metal3*** wire connected to the ***d1 input*** of the ***mux3***, as shown in the schematic.
* If you add the six wires to the bottom bitslice, use the copy command to create 7 more rows of the same thing.
* Verify the layout passes DRC and LVS. Make sure your LVS settings match those from Lab 1.

# Datapath Assembly

Next, open the datapath schematic and layout. Study them until you can identify the parts and understand why the layout looks as it does.

Compare the schematic to Figure 4 of the Getting Started Guide. Compare the datapath to the slice plan in Figure 10 of the Getting Started Guide. Observe how the datapath uses 8 horizontal metal3 tracks per bitslice for routing (centered at 10, 20, … , 80 λ) corresponding to the 8 tracks shown in the slice plan.

The alu was already a part of the datapath, so now that you have added the AND and OR wordslices, the datapath should be complete. Make sure it passes DRC and LVS. Here, you **do not** **need** to use the **“Join Nets with Same Name”** option because power and ground are internally connected.

# Datapath Simulation

Our final goal is to **simulate** the **datapath schematic** and verify that it is correct. Because generating a good set of test vectors for just the datapath would take a good deal of thought, an easier strategy is to resimulate the entire chip with the datapath schematic replacing the behavioral Verilog model of the datapath. If the entire chip works with the schematic, then the datapath is likely correct.

## Generate and edit datapath netlist

* Open the datapath schematic and generate a netlist in a new run directory such as **datapath\_run1**.

Poke around the run directory.

* Open the file **verilog.inpfiles**. This file contains a list of all the modules involved in the datapath and what subdirectory they were netlisted into, as shown in Figure 6.
* Identify where the datapath schematic has been netlisted (e.g., ihnl/cds39/netlist).
* Open this netlist in a text editor. Look at the structural Verilog that was produced by the netlister. Note the order of the inputs and outputs in the module declaration.

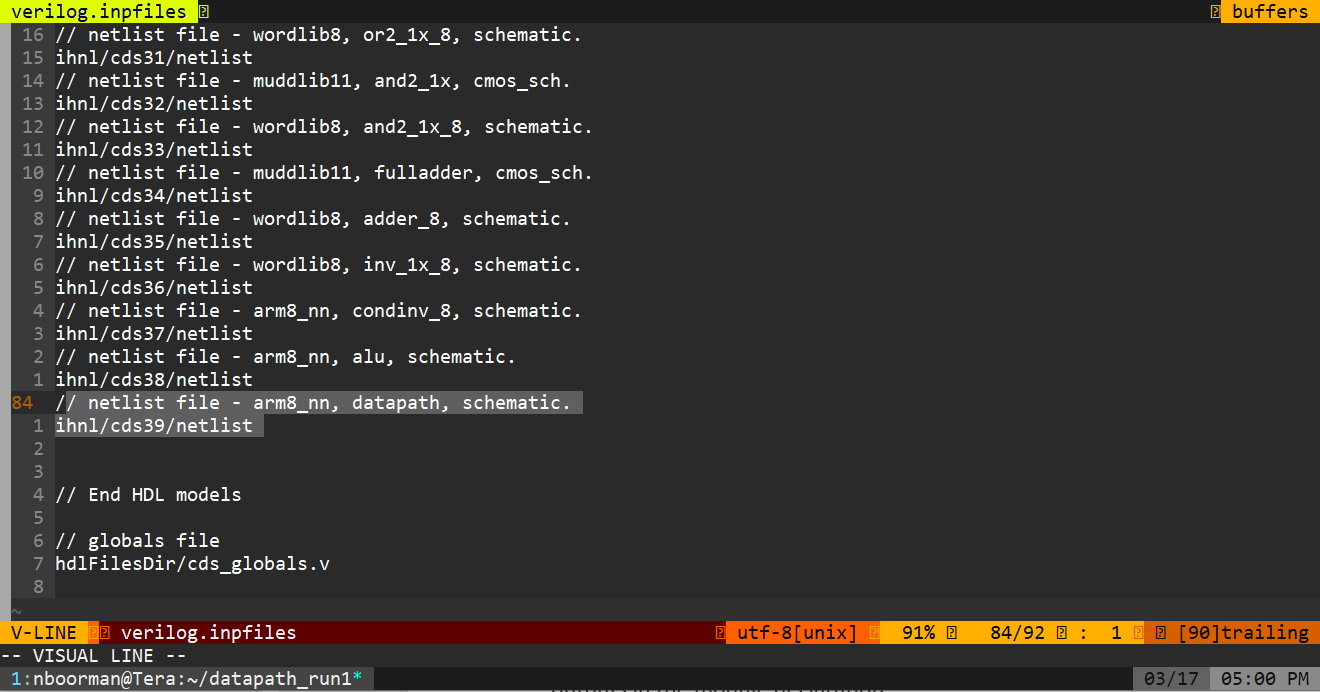


Figure 6: verilog.inpfiles listing

* Then, copy processor\_multi.sv and memfile.dat from where you simulated them in the first part of this lab. Our goal is to remove the datapath from processor\_multi.sv and replace it with the structural netlisted file.
* Open processor\_multi.sv in a text editor by typing gedit processor\_multi.sv.
* Comment out the datapath module and the alu module because we will be using the netlisted datapath and alu modules instead.
* Then, look in the processor8 module. It instantiates a controller, aludecoder, and datapath.
* We need to replace the datapath instantiation with a new one that puts the ports in the correct order. An easy way to do this is to comment out the old datapath and alu instantiations.
* Copy the datapath module declaration from the schematic netlist and paste it in and then comment it out as well. It will serve as a reference to get the order correct.
* Then, type in a new datapath call that puts the ports in the same order as they are expected must be in the netlist. When you are done, your processor8 module may look like the one below.

/\* ORIGINAL datapath instantiation, wrong port order for netlist

datapath dp(ph1, ph2, reset, Adr, WriteData, ReadData, Instr, Funct, Rd, ALUFlags,

PCWrite, RegWrite, IRWrite,

AdrSrc, RegSrc, ALUSrcA, ALUSrcB, ResultSrc,

ImmSrc, ALUControl); \*/

/\* Datapath module declaration from ihnl/cds39/netlist

Module datapath( ALUFlags, Adr, Funct, Instr[31:25], Rd, WriteData,

ALUControl, ALUSrcA, ALUSrcB, AdrSrc, IRWrite, ImmSrc, PCWrite,

ReadData, RegSrc, RegWrite, ResultSrc, ph1, ph2, reset );\*/

/\* New datapath instantiation with corrected port order \*/

datapath dp( ALUFlags, Adr, Funct, Instr[31:25], Rd, WriteData,

ALUControl, ALUSrcA, ALUSrcB, AdrSrc, IRWrite, ImmSrc, PCWrite,

ReadData, RegSrc, RegWrite, ResultSrc, ph1, ph2, reset );

## Simulate datapath using NC-Verilog

Invoke the simulation with the following command:

**sim-nc processor\_multi.sv –f verilog.inpfiles**

processor\_multi.sv is the first file to read. The ***–f*** options asks **NC-Verilog** to also load all the files specified by verilog.inpfiles. If all goes well, you will get a “Simulation completed successfully” message. If not, look for compilation errors and fix them (e.g., a typo in your new datapath instantiation). If that doesn’t find the error, fire up sim-ncg, add some key waveforms, and track down your error. It is often helpful to open two simulations simultaneously, with one showing the expected results using the golden behavioral Verilog module and the other showing the erroneous results with the actual design.

# 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. What is the verification status of your layout? Do and2\_1x\_8, or2\_1x\_8, alu, and datapath all pass DRC and LVS? Does the datapath pass DRC, LVS and simulate correctly?