Lab 5: 32-bit Single-cycle Datapath and Controller Design

Overview: In a series of laboratory exercises you will incrementally implement a 32-bit processor that supports a basic set of operations. In Lab4 you implemented the necessary datapath components for the processor. In this lab, you will put together a datapath using these components and design a state machine that controls this datapath. Datapath will be configured by the controller to execute a specific operation in a single clock cycle. This way the datapath will be able to execute different operations in each cycle. You will eventually execute matrix multiplication code on your datapath to validate the functionality of your datapath.

Lab 5 - Task 1 Total points 15 + 15 + 370 = 400 pts

Demo: Show your ***post-synthesis*** waveform simulation for each component to your TAs to get points.
Due date: Demo part 2 (especially Task 1 page 3) within the first hour of the lab session that you will attend during the week of April 3, 2017.
References:

Lecture Notes: **Verilog slides**

Objectives:

- Practice writing Datapath and Controller and the testbench
- Learn to use the ***post-synthesis simulation*** for your design

Assignments for Lab 5 Task 1:

1) (15 pts) **32-bit ALU** (Continued)

- Use the comments in the given .v file in the folder "ALU32Bit" to implement (write Verilog code) two more operations in the 32-bit ALU

// C**LO: Count the number of leading ones in a word.**
// Bits 31..0 of the input "A" are scanned from most significant to least significant bit.
Examples:
if A = -15 = 32'b1111 1111 1111 1111 1111 1111 1111 0001
then ALUResult = 28 (there are 28 1's starting from most significant bit (MSB))
if A = -1 = 32'b1111 1111 1111 1111 1111 1111 1111 1111
then ALUResult = 32 (there are 32 1's starting from MSB)
if A = 32'b1100 0000 0000 0000 0000 0000 0000 0011
then ALUResult = 2 (there are 2 1's starting from MSB)
if A = 32'b0000 0000 0000 0000 0000 0000 0000 0011
then ALUResult = 0

// **CLZ: Count the number of leading zeros in a word.**
// Bits 31..0 of the input "A" are scanned from most significant to least significant bit.
Examples:
if A = 32'b0000 0000 0000 0000 0000 0000 0000 0011
then ALUResult = 30 (there are 30 0's starting from MSB)
if A = 32'b0000 1000 0000 0000 0000 0000 0000 0011
then ALUResult = 4 (there are 4 0's starting from MSB)

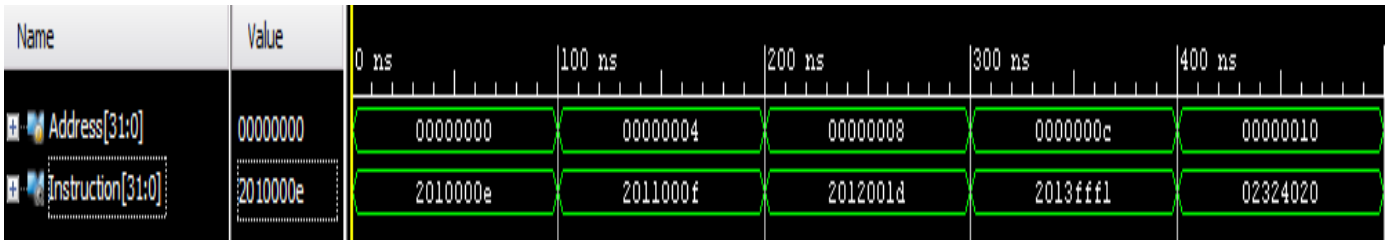
if A = 32'b1100 0000 0000 0000 0000 0000 0000 0011
then ALUResult = 0
if A = 0
then ALUResult = 32 (there are 32 0's)

- Use the same testbench for the ALU to test these two operations using examples given above (there should be 8 input cases in your waveform). Run Post-synthesis Functional simulation on your component.
  Note comment out all the previous test cases.

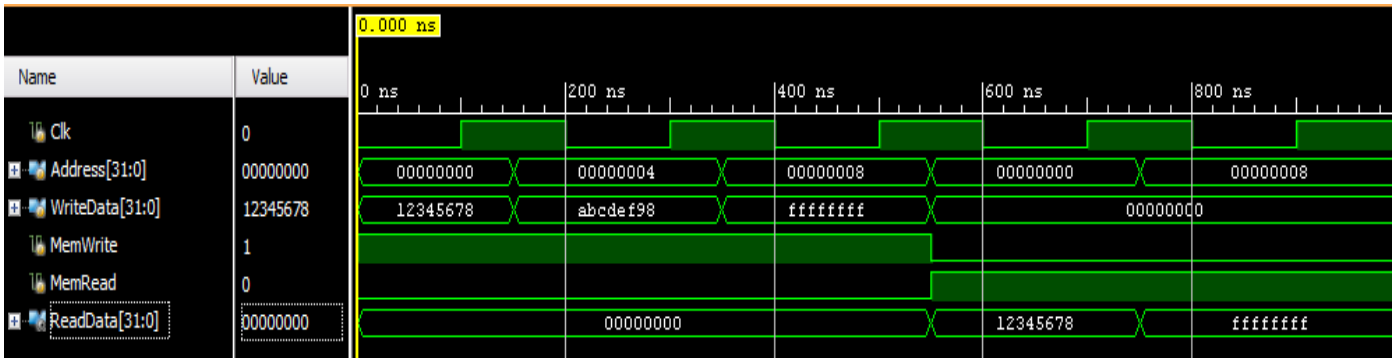
The templates (.v files and testbench files) are provided on D2L ("DatapathComponents_Part2" zipped file) to run Post-synthesis Functional simulation:

2) (15 pts) Show the post synthesis simulation waveforms for both Instruction and Data memories, to TA.

Instruction memory: It is used to keep the machine language (bianary sequence of instructions that we want the processor to execute). The initial part of the code already contains the code that you will use later on in part 2 of lab 5.
Use the given .v file in the folder "InstructionMemory" and complete the provided testbench (use the waveform shown below for inputs) and run Post-synthesis Functional simulation on your component.



Data memory: It will be used to keep the result from ALU (used in the later parts of lab 5). Use the given .v file and complete the provided testbench (use the waveform shown below for inputs) and run Post-synthesis Functional simulation on your component.



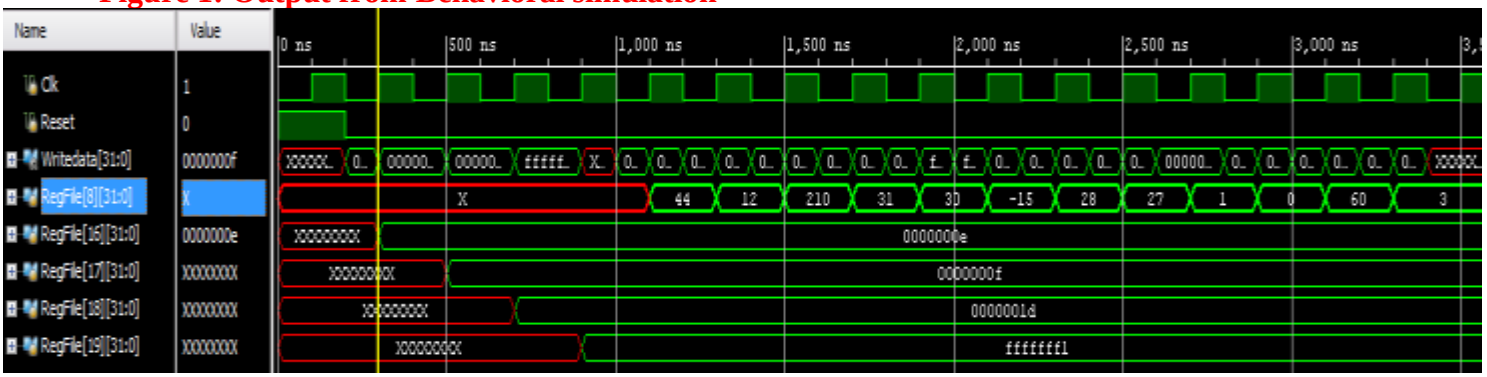
3) (410 pts) See Task 1 in the PowerPoint slides "single_cycle_datapth_Task1"

- (55 pts – 5 pt for each operation) Complete Slides 9 and 11 of "single_cycle_datapth_Task1"
  Read slides 1 – 8 and complete the table on Slide 9 (Exercise 1) – show it to your TA.
  Read slide 10 and complete the table on Slide 11 (Exercise 1 cont.) – show it to your TA

- (315 pts) Write Verilog codes (Slide 12 of "single_cycle_datapth_Task1" ppt) to implement
- Controller: Using the tables generated in Exercise 1 (ppt slides 9 and 11), design and implement (write Verilog code) a controller for the given Datapath.
  The controller has 2 inputs: 6-bit opcode and 6-bit func. Its outputs are control signals listed in the table of Exercise 1 (ppt slides 9 and 11).
- Datapath: Using slide 5, Write the Verilog code to implement Datapath (structural way – connecting several modules that you already implemented together as shown on Slide 5). It has two inputs: Clock, Reset and one output: the output from the rightmost 2x1 32-bit mux on slide 5. The above Datapath is NOT complete (it does not support shift operations yet) – read slide 13 and complete the Datapath
- Integrate Datapath and Controller: Call/Instantiate Controller in the Datapath code.
- Run behavioral simulation
  Before you can run the simulation,
  - Read slide 14 and make sure that your Instruction memory has what is required as stated in slide 14
  - Write a testbench
  Make sure that RegFile[8] shows the correct answers for each instruction in the given code before continue to the next step
  If your processor is working, you should have the following waveform (see Figure 1)
- Run Post-synthesis Functional simulation. Read Page 4

Note: See point distribution on slides 15 and 16.
30% deduction if your circuit work only in the behavioral simulation

Figure 1: Output from Behavioral simulation

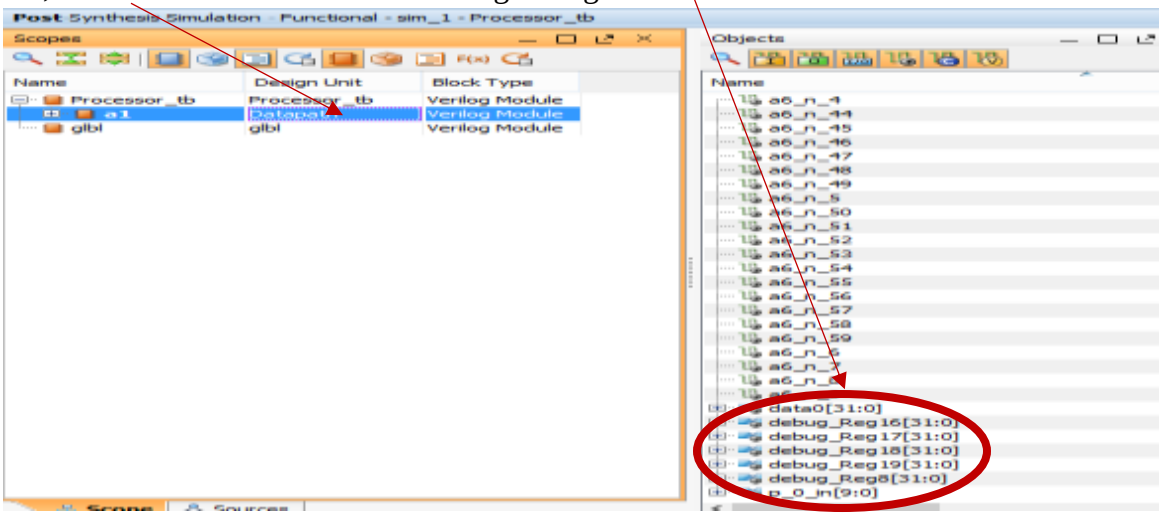


If your waveform show 0 (in place of X) - it is okay.

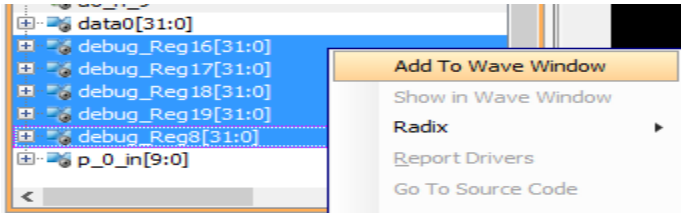
For **Post-synthesis Functional simulation**, we have to retain the signals such as RegFile8, 16, 17, 18 and 19 so that we can see their waveforms.

Download the zipped folder "retaining_signals_in_post_synthesis_simulation_guide"

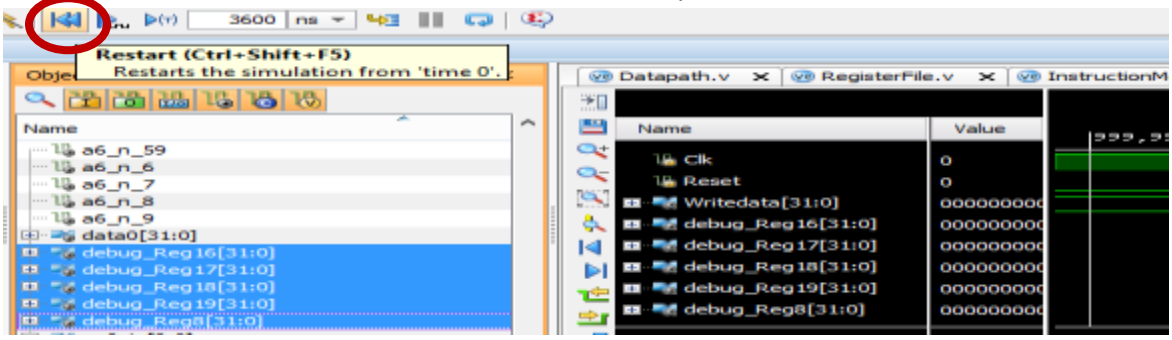
- Read "README" file
- Open "RegisterFile" in the folder to modify your RegisterFile.v file accordingly.
- Open "Datapath_top" in the folder to add the necessary signals to your Datapath..v file
- Run synthesis -> Run Post-synthesis Functional simulation
- After you run it, you should be able to see the following signals under Objects window. If not, click on and then look for those signals again



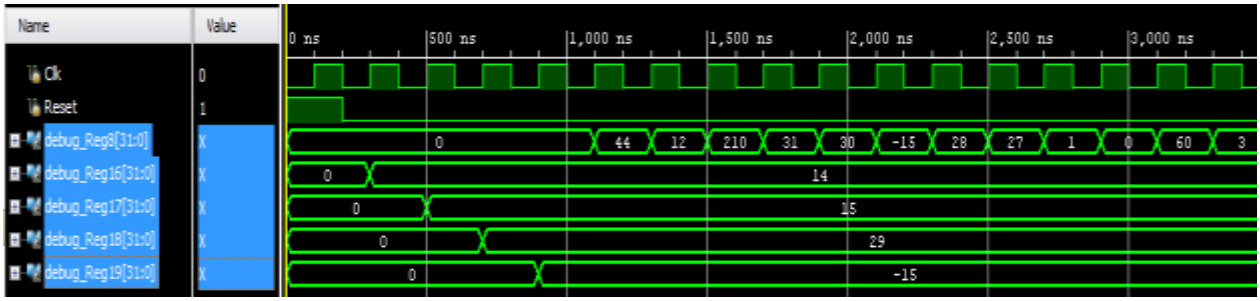
Then select all 5 signals (or one at a time) and add them to the waveform window. You can drag them to the waveform window or use "Add to Wave Window".



Then click on **Restart** button  -> then  to rerun your simulation.



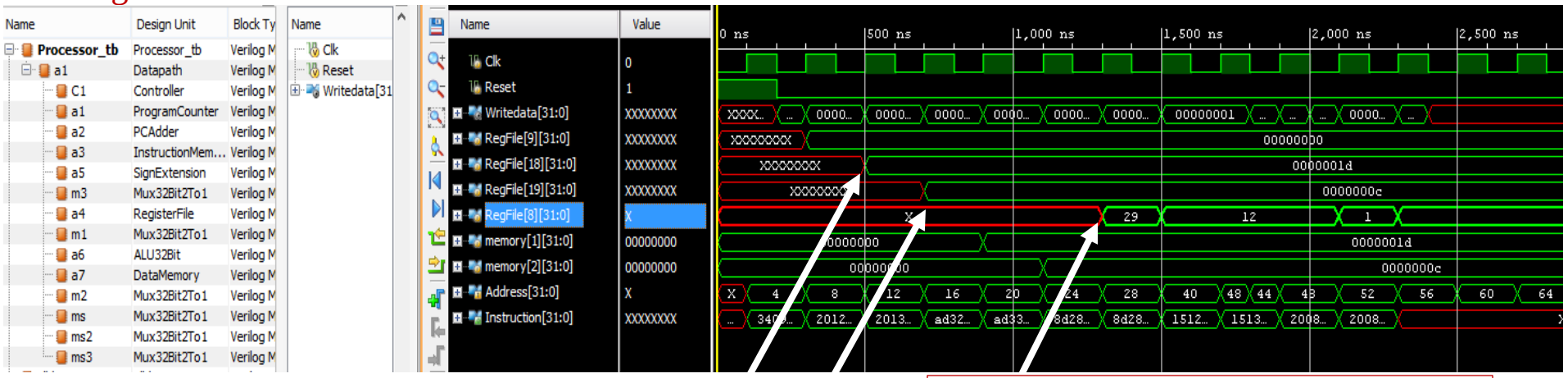
If working, you should see the following for the correct answer for part 2.


Name
Value
0 ns
500 ns
1,000 ns
1,500 ns
2,000 ns
2,500 ns
3,000 ns
Clk
0
Reset
1
debug_Reg8[31:0]
X
0
44
12
210
31
30
-15
28
27
1
0
60
3
debug_Reg16[31:0]
X
0
14
debug_Reg17[31:0]
X
0
15
debug_Reg18[31:0]
X
0
29
debug_Reg19[31:0]
X
0
-15

Lab 5 - Task 2 (200 points)

- Download "single_cycle_datapth_Task2" folder. Complete slides 4 and 7 of the PowerPoint slides single_cycle_datapth_Task2. (15pts)
- Follow slide 8 of the PowerPoint slides single_cycle_datapth_Task2
- Demo the working behavioral simulation to your TA. See Figure 1 (next page) for the working behavioral simulation waveform (110pts)
- Demo the working Post-Synthesis simulation to your TA. See Figure 2 (next page) for the working behavioral simulation waveform (75pts)
  Note: See point distribution on slide 10.

Figure 1: Behavioral Simulation



Note: RegFile[8] is the main outputs that we want to look at. Also

- show RegFile[9], [18], [19] on your waveform
- memory[1] and [2] from Data memory
- Address[31:0] from ProgramCounter
- Instruction[31:0] from InstructionMemory

Reg[9] still changes to 29 at the same time

Reg[19] still changes to 0xc at the same time

Reg[18] still changes to 0x1d at 500 ns

Figure 2: Post-synthesis Simulation