**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 Task 0, you will start with implementing the datapath components required by the processor. Later you will put together a datapath using these components and design a controller 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 a specified application code on your datapath to validate the functionality of your datapath. For now, let’s start with the design and implementation of the basic datapath components described below.

**Lab 5 schedule overview (Tentative – subject to change)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Score** | **Start date** | **Due date** |
| Task 0 | 125 points | Week of  Oct 22 – Oct 26 | The **first 60 minutes** of **your lab session** during the week of **Oct 29 – Nov 2, 2018** |
| Pre-lab of Task 1 |  | Week of  Oct 29 – Nov 2 | The **first 30 minutes** of **your** **lab session** during the **week of Nov 5- Nov 9, 2018** |
| Task 1 | 325 points | As soon as you finish Task 0 | By 4.45 pm on Friday Nov 16th, 2018 |
| Task 2 | 150 points | As soon as you finish Task 1 | By 4.45 pm on Thursday Nov 29th, 2018 |
| Task 3 | 100 points | As soon as you finish Task 2 | By 11.45 am on Wednesday May 2nd, 2018 |
| **Total** | **700 points** |  |  |

**Lab 5 - Task 0 (125 points)**

**Demo:** Show your ***post-synthesis*** waveform simulation for each component to your TAs to get points.

**References:**

Lecture Notes: Verilog ppt slides and Verilog chapter on Zybook

**Objectives:**

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

**Assignments for Lab 5 Task 0:**

The templates (.v files and testbench files) are provided on D2L (“DatapathComponents” zipped file) to write eight following Datapath components. Show your TA to get the points for each component that you have the post-synthesis simulation work correctly.

1. (5 pts) **32-bit 2x1 Multiplexer (MUX)**: Complete the given .v file in the folder “Mux32Bit2to1” to implement 32-bit 2x1 mux. Use the provided testbench to run Post-synthesis Functional simulation on your 32-bit mux.

For simulation,

a) Run Behavioral simulation. If working, continue to b)

b) Under Synthesis, choose Run synthesis. Then under Simulation, choose Run Post-synthesis Functional simulation. It should provide the same output waveform as b).

c) Run Post-synthesis Timing simulation. You will now see that the change in output waveform occurs with some delays after the change in input waveform.

When you get the correct waveform, show your waveform simulation to your TA.

1. (5 pts) **PCAdder**: Complete the given .v file in the folder “PCAdder” to implement an adder that always adds one input by 4. Complete the provided testbench and run Post-synthesis Functional simulation on your PCAdder. Use at least 4 cases in your testbench – one case should be 0xFFFFFFFC. When you get the correct waveform, show your waveform simulation to your TA.
2. (5 pts) **ProgramCounter**: Complete the given .v file in the folder “ProgramCounter” to implement a 32-bit register with a synchronous reset. Complete the provided testbench and run Post-synthesis Functional simulation on your ProgramCounter. Use at least 3 cases in your testbench.
3. (5 pts) **SignExtension** module: Use the comments in the given .v file in the folder “SignExtension” to implement a sign extension module. Use the provided testbench and run Post-synthesis Functional simulation on your module.
4. (25 pts) **32x32 Register File**: Use the comments in the given .v file in the folder “RegisterFile” to implement the 32x32 register file. Use the provided testbench (Read carefully to understand all the test cases) and run Post-synthesis Functional simulation on your 32x32 Register File. In your .v file, write the following after the inputs and output declarations to initialize Reg0 (first register in the Register File) to 0.

initial begin

regFile[0] <= 32'h0;

end

Note: The Verilog\_lecture3\_dp ppt slides under Unit Verilog on D2L discusses the implementation of 4x32 Register file. You may first start by understanding that code and then modify it to implement this 32x32 register file – make sure that

a) this one is 32x32 (not 4x32); b) the signal names are different – use the ones given in the comments in the .v file; c) for the reading operation, it occurs at the falling edge of Clk (use negedge Clk) and there is no R\_en.

5

32

ReadRegister1 ReadData1

ReadRegister2 ReadData2

WriteRegister

WriteData

RegWrite

Clk

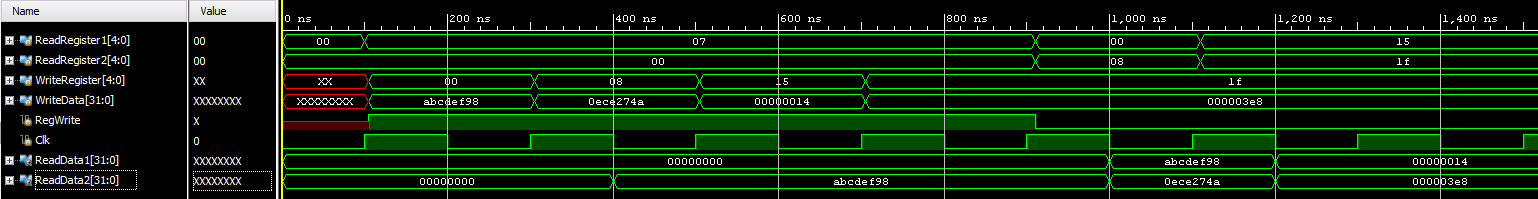
5

32

5

32

See next page for the waveform after the **post-synthesis functional simulation** of Register File



6) (60 pts) **32-bit ALU**:

(10 pts) Complete the calculations, of the table below, by hand (use a calculator, if needed) and show your answers to the TAs.

(25 pts) Use the comments in the given .v file in the folder “ALU32Bit” to implement (write the code for) the 32-bit ALU with 13 operations.

(25 pts) Complete the provided testbench using the table below (all of them should be in your testbench) and run Post-synthesis Functional simulation on your 32-bit ALU.

**ALU Control A B ALUResult Zero**

0000 (add) 0x000003E8 0x00000112 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0000 (add) 0xFFFF0000 0x1000000F \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0000 (add) 0xFFFFFFFF 0x00000001 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0000 (add) 0xFFFFFFFF 0xFFFFFFFF \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0001 (sub) 0x000003E8 0x00000112 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0001 (sub) 0x00000112 0x000003E8 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0001 (sub) 0xFFFF0000 0x0000000F \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0001 (sub) 0xFFFFFFFF 0x00000001 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0001 (sub) 0xFFFFFFFF 0xFFFFFFFF \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0010 (mul) 0x000003E8 0x00000112 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0010 (mul) 0xFFFF0000 0x0000000F \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0010 (mul) 0xFFFFFFFF 0xFFFFFFFF \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0011 (AND) 0x000003E8 0x00000112 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0011 (AND) 0xFFFF0000 0x0000000F \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0100 (OR) 0x000003E8 0x00000112 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0100 (OR) 0xFFFF0000 0x0000000F \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0101 (A<B) 0x000003E8 0x00000112 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0101 (A<B) 0x00000112 0x000003E8 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0101 (A<B) 0xFFFFFFFF 0xFFFFFFFF \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0110 (A==B) 0x000003E8 0x00000112 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0110 (A==B) 0xFFFFFFFF 0xFFFFFFFF \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0111 (A!=B) 0x000003E8 0x00000112 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

0111 (A!=B) 0xFFFFFFFF 0xFFFFFFFF \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

1000 (A<<B) 0x00000FED 0x00000001 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

1000 (A<<B) 0x00000FED 0x0000000A \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

1001 (A>>B) 0x00000FED 0x00000001 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

1001 (A>>B) 0x00000FED 0x0000000A \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

1010 (A ROTRB) 0x00000FED 0x00000001 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

1010 (A ROTRB) 0x00000FED 0x0000000A \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

Use the following cases to test your CLO and CLZ instructions

// 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

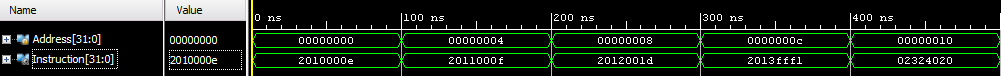
if A = 0

then ALUResult = 32 (there are 32 0’s)

7) (10 pts) **Instruction memory**

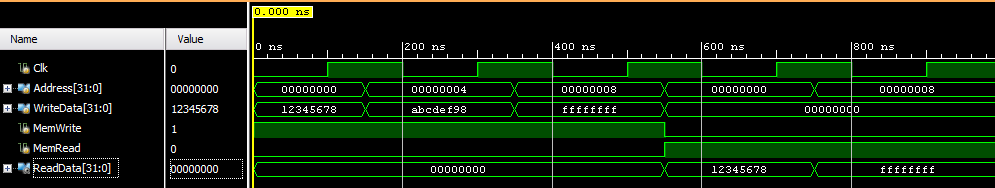
It is used to keep the machine language (binary sequence of instructions that we want the processor to execute). The “initial” part of the code already contains the code that you will use in the next task of lab 5.

First, read the given .v file in the folder “InstructionMemory” to understand how it should be used/how it functions. Then complete the provided testbench (use the waveform shown below for inputs) and run Post-synthesis Functional simulation on your component.



8) (10 pts) **Data memory**

It is used to keep the result from ALU (used in the later tasks of lab 5). First, read the given .v file in the folder “DataMemory” to understand how it should be used/how it functions. Then complete the provided testbench (use the waveform shown below for inputs) and run Post-synthesis Functional simulation on your component.



**Grading Criteria**

* Submit all the Verilog source files (.v only) including testbenches
* Complete the work effort survey for lab5 Task 0

**Penalty Conditions**

* + Percent work effort survey not completed (**20 pts penalty**)
  + Design only works in the behavioral/functional simulation but fails to synthesize (**maximum 40% of the assigned pts for each task**)
  + Design works in behavioral simulation, synthesizes with warnings but post-synthesis simulation fails (**maximum 60% of the assigned pts for each task**)
  + Late demo: **20% deductions, each day, of the assigned pts for the tasks that you/your team demo late.**