EECS 112L ORG DIGITL COMP LAB

Mentor Graphics QuestaSim Tutorial

Prepared by: Nazanin (nghasemi@uci.edu)

1. Follow the MobaXterm Tutorial (SSH_client.pdf) for connecting to the server and logging into your account.
2. Use Notepad++ to edit the files (.sv format). Download Notepad++ for free from the below mentioned website

   https://notepad-plus-plus.org/download/v6.8.3.html

Simulations in Mentor Graphics QuestaSim

In this tutorial, you will learn to compile, optimize and simulate the design. The design and the testbench is written using System Verilog. There are steps to simulate these codes.

How to Simulate a design and its Testbench

The steps are explained by taking the example of the design file and testbench. alu_32.sv is a System Verilog code of a 32 bit ALU and alu_32_tb.sv is the testbench. The same files are used to explain the steps.

A. Design the System Verilog code for the 32 bit ALU
   We are considering the System Verilog alu_32.sv file as an example. We could also do different design and play with the language.

   System Verilog Code for ALU Design

   ```
   module alu_32(
   ```

```
input logic [31:0] A, B,
input logic [3:0] opcode,
output logic overflow,
output logic c_out,
output logic equal,
output logic [31:0] output1
);


always_comb
begin
    // Default values to be overwritten.
    // To prevent the accidental creation of latches when the values are
    // not assigned
    overflow = 1'b0;
    c_out = 1'b0;
    equal = 1'b0;
    output1 = 32'd0;

    case(opcode)
        4'b0000 :  // No OP
        begin
            output1 = 32'd0;
            c_out = 1'b0;
            overflow = 1'b0;
            equal = 1'b0;
        end
        4'b0001 :  // Add
        begin
            {c_out,output1} = A + B;
            if  (A[31] & B[31] & ~output1[31])
                overflow = 1'b1;
            else if  (~A[31] & ~B[31] & output1[31])
                overflow = 1'b1;
            else
                overflow = 1'b0;
            equal = 1'b0;
        end
```

```
4'b0010 :  // SUB
begin
    {c_out,output1} = A - B;
    if  (A[31] & ~B[31] & ~output1[31])
        overflow = 1'b1;
    else if  (~A[31] & B[31] & output1[31])
        overflow = 1'b1;
    else
        overflow = 1'b0;
    equal = 1'b0;
end
4'b0011 :  // COMP
begin
    output1 = 32'd0;
    c_out = 1'b0;
    overflow = 1'b0;
    if (A == B)
        equal = 1'b1;
    else
        equal = 1'b0;
end
4'b0101 :  // AND
begin
    output1 = A & B;
    c_out = 1'b0;
    overflow = 1'b0;
    equal = 1'b0;
end
4'b0110 :  // OR
begin
    output1 = A | B;
    c_out = 1'b0;
    overflow = 1'b0;
    equal = 1'b0;
end
4'b0111 :  // NOT
begin
    output1 = ~A;
```

```
                c_out = 1'b0;
                overflow = 1'b0;
                equal = 1'b0;
            end
            4'b1000 :   // XOR
            begin
                output1 = A ^ B;
                c_out = 1'b0;
                overflow = 1'b0;
                equal = 1'b0;
            end
            4'b1001 :   // SLL
            begin
                {c_out,output1} = $unsigned(A) << $unsigned(B);
                overflow = 1'b0;
                equal = 1'b0;
            end
            4'b1011 :   // MOV
            begin
                output1 = A;
                c_out = 1'b0;
                overflow = 1'b0;
                equal = 1'b0;
            end
            default :
            begin
                output1 = 32'd0;
                c_out = 1'b0;
                overflow = 1'b0;
                equal = 1'b0;
            end
        endcase
    end
```

a. Open the notepad++ tool.
b. Design using System Verilog code.
c. Save the file using .sv format. (Example: alu_32.sv)

d. Upload the file to your account in the server using MobaXterm

B. Design the Testbench using System Verilog
We will design the Testbench using the System Verilog Language. We are considering the alu_32_tb.sv as an example.

System Verilog Testbench for testing alu_32_tb.sv

```
module alu_32_tb;
   logic[31:0] A;
   logic[31:0] B;
   logic[3:0] opcode;
   wire overflow;
   wire c_out;
   wire equal;
   wire[31:0] output1;

alu_32 L1(
              .A(A)
             , .B(B)
             , .opcode(opcode)
             , .overflow(overflow)
             , .c_out(c_out)
             , .equal(equal)
             , .output1(output1)
             );

initial begin
/*arithmetic operation*/
             A = 32'b01010101010101010101010101010101;
             B = 32'b10101010101010101010101010101010;
             opcode = 4'b0000;
             #100;
             opcode = 4'b0001;        /*-- ADD*/
             #100;
             opcode = 4'b0010;        /*--SUB*/
             #100;
             opcode = 4'b0011;        /*--COMP*/
```

```
            #100;
            opcode = 4'b0101;       /*--AND*/
            #100;
            opcode = 4'b0110;       /*--OR*/
            #100;
            opcode = 4'b0111;       /*--NOT*/
            #100;
            opcode = 4'b1000;       /*--XOR*/
            #100;
            opcode = 4'b1001;       /*--SLL*/
            #100;
            opcode = 4'b1011;       /*--MOV*/
            #100;

/*round 2*/
            A = 32'b10101010101010101010101010101010;
            B = 32'b01010101010101010101010101010101;
            opcode = 4'b0000;
            #100;
            opcode = 4'b0001;       /*-- ADD*/
            #100;
            opcode = 4'b0010;       /*--SUB*/
            #100;
            opcode = 4'b0011;       /*--COMP*/
            #100;
            opcode = 4'b0101;       /*--AND*/
            #100;
            opcode = 4'b0110;       /*--OR*/
            #100;
            opcode = 4'b0111;       /*--NOT*/
            #100;
            opcode = 4'b1000;       /*--XOR*/
            #100;
            opcode = 4'b1001;       /*--SLL*/
            #100;
            opcode = 4'b1011;       /*--MOV*/
            #100;
```

```
/*round 3*/
            A = 32'b00101010101110111010101011111110;
            B = 32'b00101010101110111010101011111110;
            opcode = 4'b0000;
            #100;
            opcode = 4'b0001; /*-- ADD*/
            #100;
            opcode = 4'b0010; /*--SUB*/
            #100;
            opcode = 4'b0011; /*--COMP*/
            #100;
           opcode = 4'b0101; /*--AND*/
           #100;
           opcode = 4'b0110;/*--OR*/
           #100;
           opcode = 4'b0111;/*--NOT*/
           #100;
           opcode = 4'b1000;/*--XOR*/
           #100;
           opcode = 4'b1001;/*--SLL*/
           #100;
           opcode = 4'b1011;/*--MOV*/
           #100;
/*round 4*/   /* Test SLL */
           A = 32'b00101010101110111010101011111110;
           B = 32'b00000000000000000000000000000011;
           opcode = 4'b1001;
           #100;
end
endmodule
```

a. Now, design the testbench using the system verilog code using notepad++.
b. Save the file using .sv format. (Example: alu_32_tb.sv)
c. Upload the file to your account in the server using MobaXterm

C. Copy the below mentioned files to your account in the server
Make few modifications to these files before uploading them to your online

account in the server.

Now upload all the below mentioned files to the same location in the server where the design and testbench were uploaded before.

1. pre_compile.csh
2. setup.csh
3. rtl.cfg
4. tb.cfg
5. sim.do

D. Go to the folder location in your online account in the server, where the files are uploaded. Use the following Linux commands to do the compilation, optimization, simulation and to view the waveform.

Note: All these Linux commands should be executed in the folder location where these files are uploaded in the server.

1. **source setup.csh**
   This source command will load the file into the command prompt. It reads and executes the commands from the file setup.csh

2. **source pre_compile.csh**
   This source command will load the file into the command prompt. It reads and executes the commands from the file pre_compile.csh

3. **Compile the System Verilog design file: alu_32.sv**
   Run **vlog -64 -sv -f rtl.cfg** on the command line.
   Here,
   - ✓ **vlog** is a command which compiles the Verilog source code/System Verilog extensions (Here it is alu_32.sv)
   - ✓ **-64** represents that vlog uses 64-bit executable
   - ✓ **-sv** is used to enable the System Verilog features and keywords
   - ✓ **-f** specifies the argument file with command lines arguments, which allows to use the complex arguments once again without retyping.
   - ✓ **rtl.cfg** is the file which has the name of the VHDL file: alu_32.vhd

4. **Compile the System Verilog file: alu_32_tb.sv**
   Run **vlog -64 -sv -f tb.cfg -work work** on the command line.
   Here,
   - ✓ **vlog** is a command which compiles the Verilog source code/System Verilog extensions (Here it is alu_32_tb.sv)
   - ✓ **-64** represents that vlog uses 64-bit executable
   - ✓ **-sv** is used to enable the System Verilog features and keywords
   - ✓ **-f** specifies the argument file tb.cfg , which allows to use the complex arguments to be used once again without retyping.
   - ✓ **tb.cfg** is the file which has the name of the System Verilog Testbench: alu_32_tb.sv
   - ✓ **-work work**. The first work is a command option which specifies a logical name or pathname of a library that is to be mapped to the logical library work. Here the name of this specified library is work.

5. **Optimize the System verilog design: alu_32.sv**
   Run **vopt -64 alu_32_tb -o tb_top_opt +acc -work work** on the command line
   Here,
   - ✓ **vopt** is used to do the global optimization on the design after the compilation has been done using vcom (for VHDL design) or vlog
   - ✓ **-64** represents that vopt uses 64-bit executable
   - ✓ **-o** allows us to designate the name of the optimized design file
   - ✓ **+acc** provides visibility into the design for debugging purposes
   - ✓ **-work work**. The first work is a command option which specifies a logical name or pathname of a library that is to be mapped to the logical library work. Here the name of this specified library is work.

6. **Simulate the Design: alu_32.sv using the Testbench: alu_32_tb.sv**
   Run **vsim -64 -c tb_top_opt -do sim.do** on the command line.
   Here,
   - ✓ **vsim** command is used to simulate the VHDL design or an entity/architecture pair or a verilog module/system verilog extension or an optimized design.
   - ✓ **-64** represents that vsim uses 64-bit executable
   - ✓ **-c** specifies that the simulator will run in command-line mode

✓ **tb_top_opt** this is the name of optimized design which we want to simulate.

✓ **-do sim.do** tells the vsim to use the commands specified in the do file.

7. View the Waveform

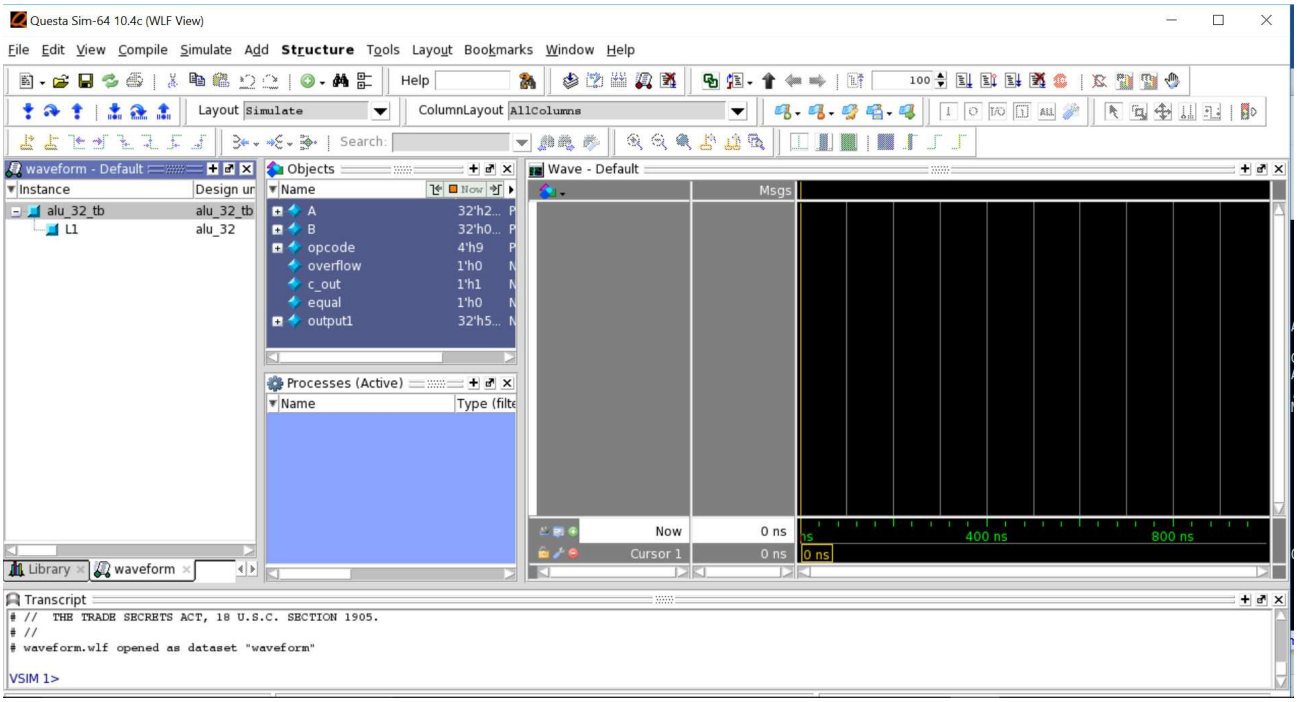
✓ Run **vsim -64 -gui -view waveform.wlf**

Here,

✓ **vsim** is also used to view the results of the previous simulation run when **-view** switch is invoked

✓ **waveform.wlf** is the simulation file. It is opened to view the waveform.

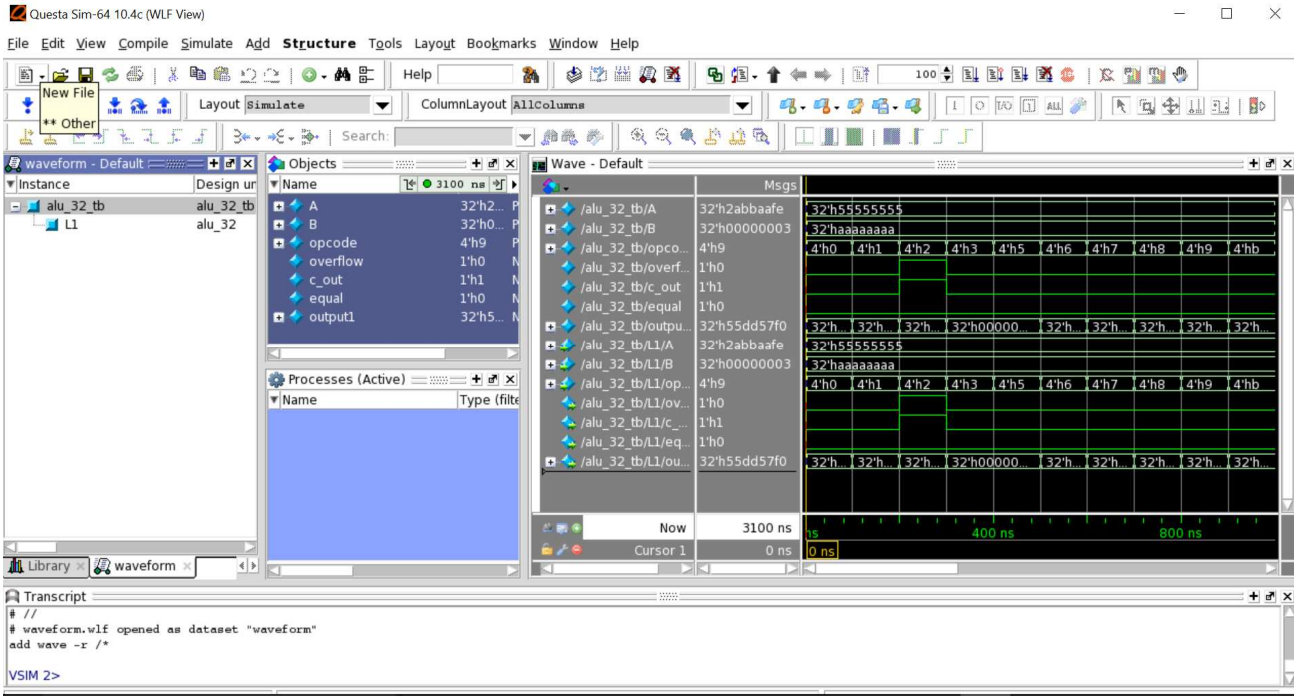
After running the command, a window will pop up. That window is shown below



8. Step to add waveforms in the simulation window

✓ Use the mouse and right click on the alu_32_tb in the waveform default tab. Select Add to/Wave /all items in design.

✓ Simulated waveforms should appear on the right side of the screen. It is shown below.



✓ Check the waveforms based on the desired functionality. If the waveforms do not follow the desired functionality, then make the changes in the design or testbench. Do the compilation and further steps which follows.
If you get the desired results and functionality, then your design is good enough to be proud of your efforts!