**CECS 341 – LAB 1 – Multiplexer (Simulation)**

**Objective: To become familiar with Xilinx software** and to simulate a simple Multiplexor using Verilog behavioral Hardware Description Language (HDL).

In case you are asking what is a multiplexer, multiplexer is a fundamental building block used in Logic Design and you probably came across it in your first digital course. If you need refreshing, see pages in Appendix A of the 341 text called Basics of Logic Design, page A10 for a very basic review.

The Big(?) Goal (looking at the big picture):

Our goal is to “build” or “construct” the ARM processor (actually a trivial version of it sarcastically called LEG processor) seen in Figure 4.1 in text (p.258). We will not build it in hardware but in software – in Hardware Description Language. We will **simulate** what we “**build**” to confirm that it works as designed. Figure 4.1 is somewhat sketchy. The details get added in Figures 4.2, 11, 17, 19, 20, 21 and 23. But we need not fear – the pieces are added **one at a time** and the bigger unit is checked before proceeding to add another piece. See Figures 4.5, 6, 7, 8, 9 and 10.

We use Verilog as the Hardware Description Language. In case you are asking what Verilog is, you probably covered it in the first digital course. There is also a brief review in the 341 text in its Appendix A, Section A4, pages A20 to A25. There is also a reference to a tutorial on Verilog in the companion web site. There are also million (that is an exaggeration) resources in the web for Tutorials on Verilog.

First, about the Xilinx tool.

There are two major steps in this Lab of familiarizing us with the Xilinx tool**:** They are **(1)** **Design Entry** and **(2) Simulation or Verification**. Design entry for hardware can be done in many ways – by a schematic diagram or by Hardware Description Language (HDL) or even by a State Diagram in the case of Sequential circuits. The first and third are graphical but the second is textual. Simulation also can be performed at two levels – Behavioral Simulation or Timing Simulation. Behavioral simulation simply checks whether the design will behave or function as expected. It does not bother about timing issues (It is also called functional simulation by some). Timing simulation is obviously more involved. In this Lab, we will enter the design textually in Verilog and perform only behavioral simulation using a Verilog “test fixture”. Ideally we should go for the third step also which is Implementation on a test kit and confirm that the design works as planned. We will skip this step in this lab.

The first simulation project is a simple 4 to 1 **Multiplexor** or Data Selector. A “mux” selects one of the many input lines to appear at the output depending on the selection inputs. The block diagram is shown below:

In0

..

...

Out

..

In3

Sel

Read the brief description of a 2 to 1 1 bit multiplexor found in p.A.10 in Appendix A (The Basics of Logic Design). Page A.25 describes a 4 to 1 32 bit multiplexor. We will use the code given in Figure A 4.2 on p.A.25 of the text. Instead of 32 bit vectors for In and Out, we will use only 1 bit scalar. This is trivial but the idea of this lab is to familiarize ourselves with the Verilog HDL and the Xilinx waveform simulator.

**Starting ISE and Opening a New Project:**

To start the Xilinx ISE tools, click on the Windows icon at the left bottom. In the listed programs, select **Xilinx ISE Design Suite 14.7 > ISE Design Tools > 64 bit Project Navigator)**.  You may close the tip window.

ISE may come up with the last project opened by the tool but a new project needs to be started.  To create a new project, click above the console pane which has messages, on the Start tab. Select **New Project**.  This will bring up a small dialog box called “New Project Wizard – Create New Project”. We need to enter information about the new project.  In the "Project Name" box, type lab1 or mux. Now, select an appropriate location for all of the project files in the "Project Location" box, preferably say, a flash drive. C is not writable. You may choose or create a folder called 341labs. Type the location as say, D:\341labs\

For the ‘top level source type’ at the bottom, select **HDL**. Click next. This will open a dialog box called “New Project Wizard, Project Settings”. We need to specify the Device details (though we will not implement the multiplexer using a Field Programmable Gate Array (FPGA) test kit). Use the drop down menu for each item and select. For the Product Category, you may select General Purpose. The FPGA "Device Family" we are going to use is the **Spartan3E** and the "Device" is **XC3S500E** and the "Package" is **FG320.** The "Speed Grade" may be left at **-5**. For Synthesis Tool, select **XST (VHDL/Verilog)**. (XST stands for Xilinx Synthesis Technology and VHDL and Verilog are the names of two Hardware Description Languages (HDLs)). For the Simulator, in the pull down menu, choose ISim (VHDL/Verilog). This is Xilinx’s simulator tool. For Preferred, select Verilog. You may leave others as they are. Click Next. The New Project Wizard may give you a project summary. Click Finish.

**Part A - DESIGN ENTRY IN HDL:**

On top of the console at the bottom, choose Design tab (if not already). In the View line above the Hierarchy pane on the left, choose Simulation (not implementation). For every project, we have to add **“Sources”** and for that source, go through different **“Processes”**. There is an icon with yellow star called New Source (to the left of the word View). Or you may choose in the Project menu, choose New Source. In the “New Source Wizard – Select Source Type” dialog box, select Source Type as **Verilog Module**. You may type the filename as mux. “.v” for Verilog will be appended later. Location will default. Let Add to Project be checked. Click next. In the Define Module dialog box, you may enter the module name (if not already done) as mux. We can specify inputs and outputs (to reduce our work of declaring them in the Verilog module). Start with Port Name. Enter In0 in the first row. It is an input and is a scalar (1 bit only). So we need not choose Bus (for multi-bit) and we need not enter MSB value and LSB value. Hit Enter and in the next row, enter In1 etc. Enter key takes you to the next row. After In3, for Sel which a 2 bit vector input, we need to check the Bus button and enter MSB as 1 and LSB as 0. Hit the Enter key. In the next row, enter Out and change the direction as output (It is one bit only). Click next. It gives the summary for the New Source. Click finish. In the Hierarchy pane on the left, you should see mux.v and on the right, a skeletal Verilog file will appear. We need to fill in. It gives the module name with port list and lists the inputs and outputs. We need to declare Out as reg since it is assigned (<=) inside the case statements that we will be adding. So in the port list, the statement for output will be ‘output **reg** Out’. Enter the rest of the statements (modified version of p.25 of Appendix A) given below:

module mux(

input In0,

input In1,

input In2,

input In3,

input [1:0] Sel,

output **reg** Out

);

**always @(In0, In1, In2, In3, Sel)**

**case (Sel)**

**0: Out <= In0;**

**1: Out <= In1;**

**2: Out <= In2;**

**3: Out <= In3;**

**endcase**

endmodule

Click on the Save in the File menu or click on the save icon. If there are errors, they will appear pink in the console. We need to correct them. Better to do a syntax check. Select (highlight) mux.v (box with V in it) in the Hierarchy pane. In the Processes pane below the Hierarchy pane, expand ISim simulator (clicking the + sign). Double Click on Behavioral Check Syntax. A blue diamond starts rotating to indicate that the syntax is being checked. A green tick mark should appear and a message saying successful completion of checking syntax will appear. You can print the file from the File menu.

**Part B: BEHAVIORAL SIMULATION**:

Simulation involves developing two files. One is the module we just developed (mux.v). This describes the behavior of the unit. Normally this is easy. Another file to develop is the “test bench” – another Verilog file (with the extension “.vtf” for Verilog test fixture) which will exercise the object module. All possible combinations of the input signals must be given to ensure that the module we built gives the correct output signals as we desired. This is a bit more challenging. Hopefully the simulator’s outputs are as expected. (Let Simulation (not implementation) be selected). Make sure that mux.v is highlighted in Hierarchy box. We have to create a Verilog test fixture (.vtf) file as a **source** to associate with the mux project for behavioral simulation. Right click on mux.v and choose New Source. In the dialog box called the ‘New Source Wizard – Select Source Type’ which asks us to select a source type, select the type “**Verilog Test Fixture**". Give it a file name as muxvtf. The extension .v will be added later. Location will default. Click next. When the "New Source Wizard - Associate Source" window appears, mux should be selected already (it will be the only available selection). Click on "Next". It summarizes New Source Information. Click "Finish".

A skeletal file in Verilog Hardware Description Language should open on the right and we need to flesh it out.

In any simulation, as mentioned already, creating a test is more challenging sometimes than creating the HDL object module which describes the behavior of the unit. For the mux ‘unit under test’ (uut), we need to verify that the output obeys the selection commands. So we need to go through each combination of the 2 bit selection lines. In addition, it will be good to make sure that the output can become either 1 or 0 depending on the input applied. It should be able to change. Thus these are tested typically by doing the “walking 1” and the “walking 0” test. In the walking 1 test, for each Sel combination, we change the *corresponding* In*i* line to 1 and check that 1 appears at the Out line. In3 to In0 lines will get 0001, then 0010, then 0100 and 1000 as Sel cycles from 00 to 11. We confirm that Out line gives 1 and is not “stuck at 0”. Then we conduct the “walking 0” test. Now In3 to In0 lines will get 1110, 1101, 1011 and 0111. Out line should give us 0, confirming that it is not “stuck at 1”. “Stuck at” faults are the most common type of faults since wires in the circuit layout may get connected by mistake to adjacent power or ground wires.

We have 8 cases to test (8 input combinations) and so we should choose 800 ns as our total time (assuming 100 ns as our time slot). Let us make it as say, 1100 ns with two extra time slots at the beginning and one extra time slot on the end.

The template already lists the inputs and outputs. (inputs become type ‘reg’ and outputs become type ‘wire’). It will “instantiate” or refer to the Verilog source module that we developed earlier by giving the module name, an arbitrary instance name (say, uut) and port list. We have to supply the stimuli. Time is specified with # and time is calculated *cumulatively*. Default time unit is ns. On the top, you will see a ‘directive’ with the prefix “`” giving the time scale (as 1 ns) as well as the iteration interval for the simulator (as 1 ps). Let us input the 8 cases of input combinations as follows. After the line “ //Add stimulus here”, add the extra lines shown.

module muxvtf;

// Inputs

reg In3;

reg In2;

reg In1;

reg In0;

reg [1:0] Sel;

// Outputs

wire Out;

// Instantiate the Unit Under Test (UUT)

mux uut ( .In0(In0),

.In1(In1),

.In2(In2),

.In3(In3),

.Sel(Sel),

.Out(Out)

);

initial

begin

// Initialize Inputs

In0 = 0;

In1 = 0;

In2 = 0;

In3 = 0;

Sel = 0;

// Wait 100 ns for global reset to finish

#100; //#sign gives elapse of time

// Add stimulus here

//………………………………………………

**// Checking “walking 1”**

**#100 Sel = 0; In0 = 1;// We need to tell only changes**

**#100 Sel = 1; In0 = 0; In1 = 1;**

**#100 Sel = 2; In1 = 0; In2 = 1;**

**#100 Sel = 3; In2 = 0; In3 = 1;**

**// Checking "walking 0"**

**#100 Sel = 0; In0 = 0; In1 = 1; In2 = 1;//In3 is already 1,only In0 be 0**

**#100 Sel = 1; In0 = 1; In1 = 0; //In2 is already 1, Only In1 to be 0**

**#100 Sel = 2; In1 = 1; In2 = 0; //In0 is already 1, Only In2 to be 0**

**#100 Sel = 3; In2 = 1; In3 = 0; //Only In3 to be 0**

**//Let us zero everyone**

**#100 Sel = 0; In0 = 0; In1 = 0; In2 = 0; //In3 is already 0**

end

endmodule

Make sure muxvtf.v is highlighted. Save the file by clicking on the disc icon (**Save)**.  Better to check the syntax by double clicking the Behavioral Check Syntax in the Processes box. Correct the errors if any and Save and recheck syntax for the green tick mark.

Double click on the Simulate Behavioral Model in the Processes window pane to start ISim. Cancel a message about Windows Firewall in case it appears. We will see the new window with the waveform. Notice that the time scale is in ps (picoseconds) by default. The time box may say 1 us (1,000,000 ps). Let us thin out the Name and Value columns to see more of the waveform. Zoom out to see the relevant 800 ns with one time slot on either side (200 ns to 1000 ns). There is a drop down box with time as 1.00 us. Change it to 1.10 us. In the Name column on left, you can select Sel and move it all the way to the top since it is the master controller. If Out is on top, with the mouse, bring it to below all In*i*. You can click on zoom in (+sign) or zoom out (-sign).You can click on “Zoom to full view” icon (a lens with a circle at the center and four inward arrows). It can be a bit challenging to get the desired range of the waveform to be displayed in full. You can move the horizontal scroller at the bottom appropriately. If needed, we can even separate the inputs and outputs by providing dividers with appropriate headings. Right click on the signal name pane, choose New Divider and write in ‘Inputs’. Move it to the top. Similarly we can create a divider for Output. Remember that the waveform should have all inputs first on top and the resulting outputs below. If you prefer, select Sel in the Value column and right click. Select radix and choose decimal if you prefer.

Try to move the bottom scroller to left to start from say, 100 ns. We can inspect by moving the Time Marker at various times in the waveform. The value column gives the values at the time Marker. Check for the 8 time slots. In the Set Up menu for print, choose the time range as 100 ns to 1100 ns. If this was a test, you need to print the waveform (To print, Landscape must be already selected). and you must *annotate* (confirm) the result, i.e., for each time slot, you must state what the theoretically expected result was and what the simulator output came out to be. For example, for the first time slot (200 – 300 ns), you may write underneath it, Sel = 0; In0 = 1; In1 =0; In2 = 0; In3 = 0; **Out = In0** = 1 (Th), Out = 1 (Simulator Output). √ .

Save the waveform as mux.wcfg. Exit ISim by File -> Exit. Exit the Project Navigator.

You can install Xilinx “ISE Web Pack” in your desktop or notebook by going to Xilinx.com. It is an involved process – includes creating an account with Xilinx and doing an extra step for license management (get a file ‘license.dat’ to include it in an indicated file). It appears that one can download Web Pack 14.7 choosing Developer Zone, then Hardware Developer Zone -> ISE Design Suite -> Web Pack Edition -> Download -> for Windows etc. etc. …..Left side bar will have different version numbers etc…. mc,s17