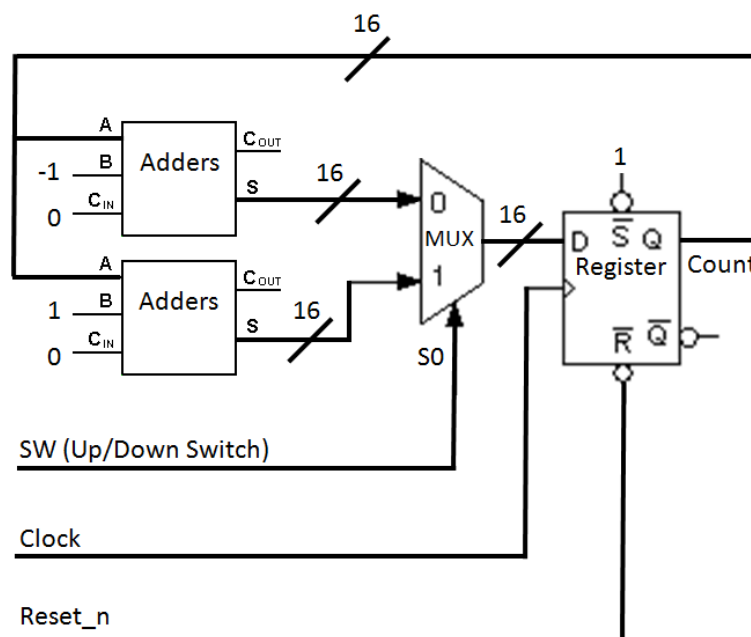


Verilog / ModelSim-Altera Tutorial and Lab Exercise

Objective: This tutorial covers the functional and timing simulation of a high-level Verilog design using ModelSim-Altera. The Verilog design is also compiled in Quartus II and programmed onto the Altera DE2 board.

The tutorial concludes with a lab exercise which extends the Verilog design covered in the tutorial and adds a combinational logic block. This new design is then simulated and implemented on the Altera DE2 board.

Up/Down Functional Block Circuit for Part I and Part II:




The up/down counter circuit you will be simulating is shown in the figure above. You will begin with the provided Verilog code that is the code representation of the counter circuit. The up/down counter consists of three inputs: Up/Down, Reset, and Clock inputs. Depending on the input combination, the multiplexer (MUX) will choose whether to increment or decrement the current count value stored on the D-Flip Flop Q output. The "addition" and "subtraction" statements are performed by Full Adders, "If..else" statements are represented by the 2-input MUX, and the count assignment or storage is performed by a memory device like a D-Flip Flop.

I. Functional Simulation in ModelSim-Altera

The first step in the design process is a functional simulation of your Verilog description of your circuit. We will use the ModelSim-Altera simulator in order to simulate and debug Verilog designs both interactively and using a test bench.

A. Create a ModelSim-Altera project

- 1) Run ModelSim-Altera. Click on the Windows Start icon and select All Programs. Open the Altera folder (or Altera Web Edition folder) and the ModelSim-Altera subfolder (or ModelSim-Altera Starter Edition subfolder) and click on the ModelSim-Altera program icon, . (Or double-click a desktop ModelSim-Altera icon, if there is one available.)



NOTE: Don't worry about initial program opening errors like "Unable to open project." or "File not found." This just means the last project or file opened by ModelSim-Altera was not found. We will create a new project for each lab.

- 2) Create a new directory in your work space, such as eec180b/lab0, and copy the design files into it. The design files are named "updown.v" and "tb_updown.v" and are posted on the course SmartSite.
- 3) Select **File > Change Directory** and change to the directory you created in step 2. Changing the directory will close the currently open project, if there is one.
- 4) Select **File > New > Project...** to create a new ModelSim project. In the "Create Project" dialog box, enter the following:
Set Project Name to: **lab3**
Set Project Location to: **the directory you created in step 2.**
Set Default Library Name to: **work**
Leave the default settings for the modelsim.ini path and the option Copy Library Mappings.
Click **OK**
- 5) A window titled "Add Items to the Project" will pop up on the successful creation of the project.
Double-click the **Add Existing File** icon.
Click the Browse button and select the updown.v design file in your project directory and click **Open**.
Select the option: **Reference from current location**
Leave the default settings for "Add file as type" and "Folder" option selections, the default options selected should be "default" and Top Level" respectively.
Click **OK**

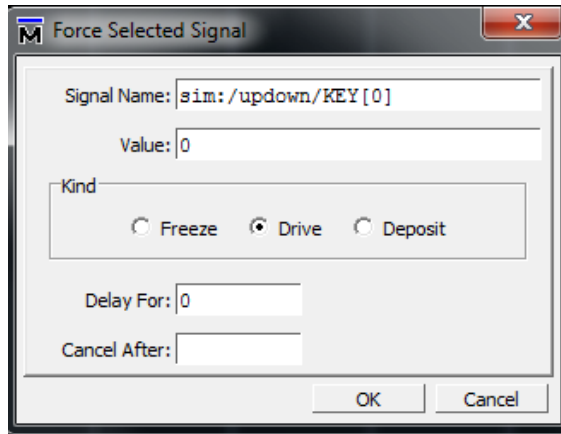
Click **Close** to close the “Add Items to the Project” window.

B. Compile and Simulate a Verilog Design in ModelSim-Altera

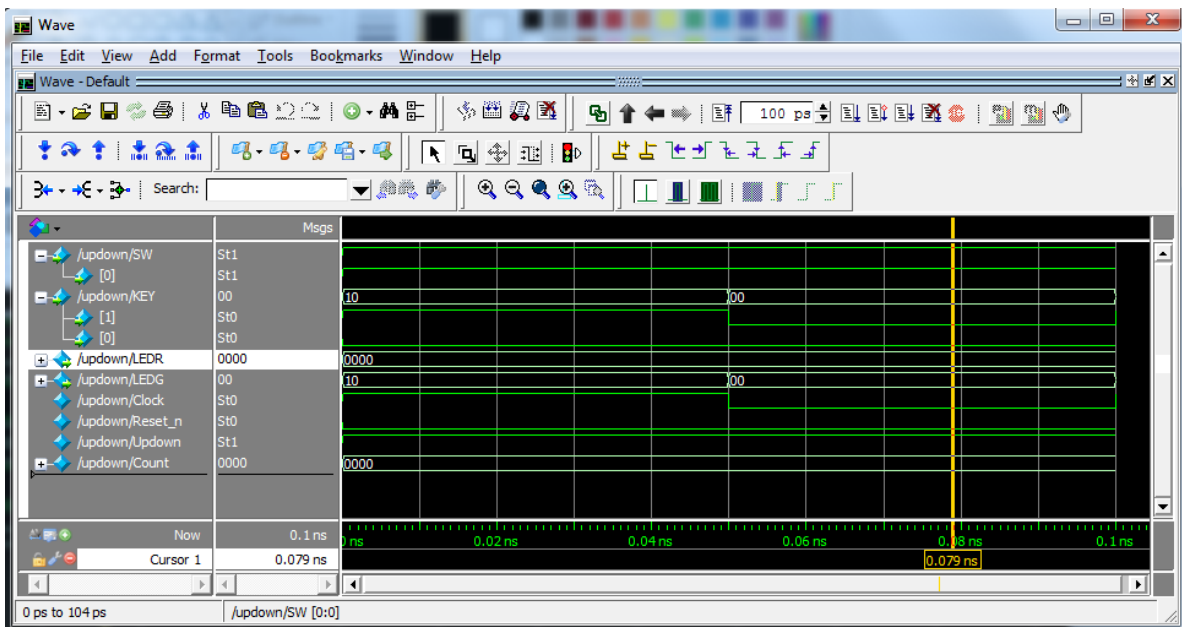
- 1) Select **Compile > Compile All** to compile updown.v. You should not get any errors.

NOTE: Notice before you compile any design you see a  next to your file in the status column. This question mark indicates the specific file in the project has not been compiled. Once the file is compiled, the question mark turns into a .

- 2) Select the **Library** tab at the bottom left of the workspace pane, next to the Project tab. Expand the **work** library by clicking the “+” box next to it. You should see the updown module and its path.
- 3) Select the updown module and select **Simulate > Start Simulation....** The Start Simulation dialog box will pop up. Select the Design tab, expand the work library and select the updown module as the Design Unit(s). Click **OK**. (Note: we do not need to add the cycloneii_ver library for a functional simulation.)
- 4) The sim-Default, Objects, and Processes (Active) windows will be loaded with design data from the updown.v design file. If the Wave window is not open, open it by selecting **View > Wave** from the pull-down menu.
- 5) In the Objects window, select the signals that you would like to view in the Wave window. You should definitely select **SW**, **KEY**, **LEDR**, and **LEDG** since these are the input and output signals. You might also want to select some internal signals such as **Clock**, **Reset_n**, **Updown**, and **Count**. Select **Add > To Wave > Selected Signals** to add the signals to the Wave window. (You can also drag and drop them from the Objects window to the Wave window.)
- 6) Change the radix for **LEDR** and **Count** to hex by selecting the signals in the Wave window, right-clicking and selecting **Radix > Hexadecimal**;
- 7) Expand the **KEY** signal by clicking on the ‘+’ next to it in the Wave window. Select **KEY[1]**, right-click and select **Clock....** Accept all the defaults and click **OK**. (This will give a default clock period of 100 ps.)
- 8) Select **KEY[0]**, right-click and select **Force....** Enter the value as ‘0’ and select the Kind as **Drive** as shown in the Figure below. Leave the **Delay For** parameter at the default value of 0.



- 9) Follow the same procedure to set the **SW[0]** signal to '1'. Now we have initialized all the inputs so we are ready to simulate.
- 10) Run the simulation for 100 ps. You may need to zoom-in and adjust the zoom settings to reproduce the figure below. You should see waveforms similar to those in the Figure below. Note that Reset_n is the same as KEY[0], Clock is the same as KEY[1] and Updown is the same as SW[0]. Since Reset_n is low, the Count has been reset to 0x0000.



- 11) Force input KEY[0] to logic '1' and run the simulation for several hundred ps. You should see the Count start increasing. Simulate until the Count reaches at least 0x0004.
- 12) Force input SW[0] to logic '0' so that the counter will count down instead of up. Simulate until the Count goes below 0x0000. Depending on how many times you

step forward in the simulation in the previous count up step, the count could increase by one before counting down. Once you have reached past 0x0000, the count will wrap around to 0xffff, which is the twos complement representation of -1.

13) Demonstrate your simulation waveforms to your TA and have your TA sign your verification sheet.

14) File > Quit ModelSim and close the program.

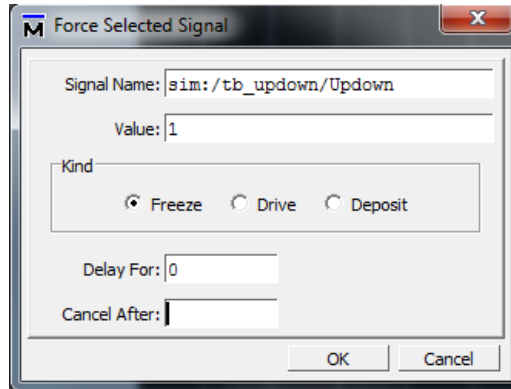
C. Simulate Using a Testbench

As you may have noticed, it is very tedious to configure inputs manually when simulating designs with multiple inputs. It is often more convenient to use a “testbench” program, also written in Verilog, to generate the inputs needed to simulate your design.

- 1) Run ModelSim-Altera and open your project that you used in Part IB. Notice ModelSim-Altera opens the last working project before ModelSim-Altera was closed previously. You will also see the updown.v file listed. However, if you closed the project before exiting ModelSim-Altera previously, your project will not automatically open and you will not see the updown.v file listed. To open the project, click File > Open and scroll to the directory where you stored the lab3 files. Select show ALL FILES in the file name selection box, and Select the <filename>.mpf file. The mpf extension indicates the file is a ModelSim Project File (MPF). Click Open.
- 2) Select **Project > Add to Project > Existing File** and click the **Browse...** button. Make sure that you have copied the tb_updown.v file into your project directory along with updown.v. Select **tb_updown**, click **Open** and then click **OK** to close the dialog box.
- 3) Select **Compile > Compile All** to compile both Verilog files. There shouldn't be any errors.
- 4) Select **Simulate > Start Simulation**. The Start Simulation dialog box will pop up. Select the Design tab, expand the work library and select **only** the tb_updown module as the Design Unit(s). (The updown module is an instantiated component of the testbench so it doesn't need to be included separately.) Click **OK**. (Note: we still do not need to add the cycloneii_ver library for a functional simulation.)
- 5) Add the signals you would like to observe to the Wave window as in Part IB. Change the radix for LEDR to hex.
- 6) Note that there are **initial** blocks (located in the "sim-Default" window under the tb_updown design unit) to generate the Clock and the Reset_n signals so now the

only signal that needs to be set manually is the Updown signal to control if the counter counts up or down. (We could easily write an **initial** block to generate the Updown signal if we want.)

- 7) Force **Updown** to '1', but this time leave the Kind as "**Freeze**" as shown in the Figure below.



- 8) Run the simulation for 100 ps. Force **Updown** to '0' and run for another 200 ps. Do your simulation waveforms make sense?
- 9) Demonstrate your simulation waveforms to your TA and have your TA sign your verification sheet.
- 10) Quit ModelSim.

II. Timing Simulation in ModelSim

To perform timing simulation we need to compile the design in Quartus II in order to generate the necessary timing information for ModelSim-Altera.

A. Interactive Timing Simulation

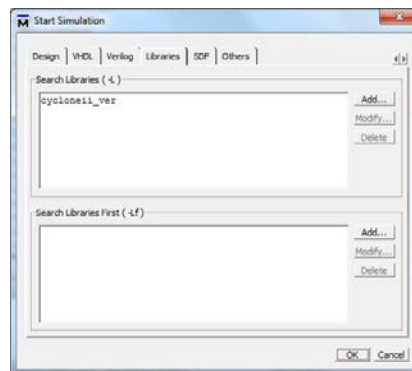
- 1) Create a new Quartus II project as you have for Labs 1 and 2. You can put the Quartus II project in the same folder that you used for ModelSim functional simulation, such as eec180b/lab0. Specify the name of the project and the top-level design entity as **updown**. Add the updown.v file to your Quartus project, but do **not** add the tb_updown file. Only the updown.v file is synthesizable Verilog that will produce a circuit that will run on the Altera DE2 board. Specify the device family as Cyclone II and the device type as EP2C35F672C6 as in previous labs. Specify ModelSim-Altera as the Simulation tool.
- 2) Import the Pin Assignments file by selecting **Assignments > Import Assignments...** and selecting the DE2_pin_assignments file.
- 3) Compile the design in Quartus II. It should compile without any errors.

- 4) In the simulation/modelsim subfolder of your project folder, you should see the Verilog netlist files, <filename>.vo. We will simulate the updown.vo file.
- 5) Create a **NEW** ModelSim project in the simulation/modelsim subfolder. Follow the same steps as in Part I to create the project. Add updown.vo to the project (but not tb_updown.v). If an old project is automatically loaded into ModelSim-Altera, click **File > Close Project**, and then create a new ModelSim project following the previous steps above. You can also just directly create a new project and ModelSim-Altera will ask whether you would like to close and save the existing project. Click Yes, and continue to create the new project.

NOTE: In the previous labs we did not need to create a new ModelSim project to simulate the designs from Quartus II. While we can definitely perform the same procedure here by clicking the "Gate Level Simulation" button in Quartus and save some steps, that "short-cut" method only works if we are **NOT** using a Test Bench. For ALL test bench simulations, you will need to create a new ModelSim-Altera project.

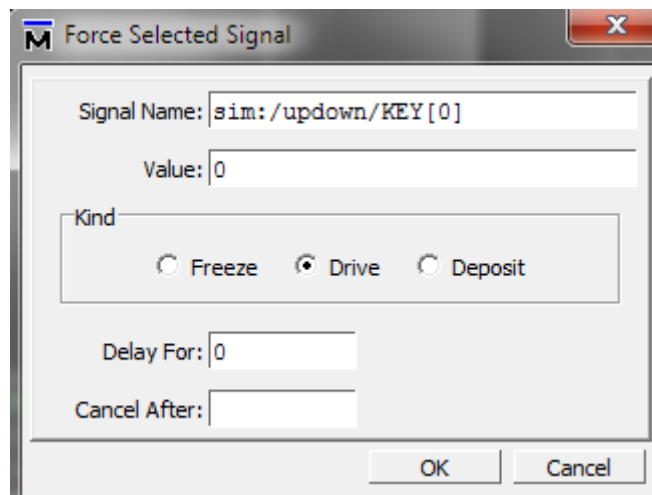
- 6) Select **Compile > Compile All** to compile updown.vo in ModelSim. You should not get any errors.
- 7) Select the **Library** tab at the bottom of the workspace pane. Expand the **work** library by clicking the "+" box next to it. You should see the updown module and its path.
- 8) Select the updown module and select **Simulate > Start Simulation....** The Start Simulation dialog box will pop up. Select the Design tab, expand the work library and select the updown module as the Design Unit(s).

Select the Libraries tab and add the library **cycloneii_ver** as shown in the Figure below.

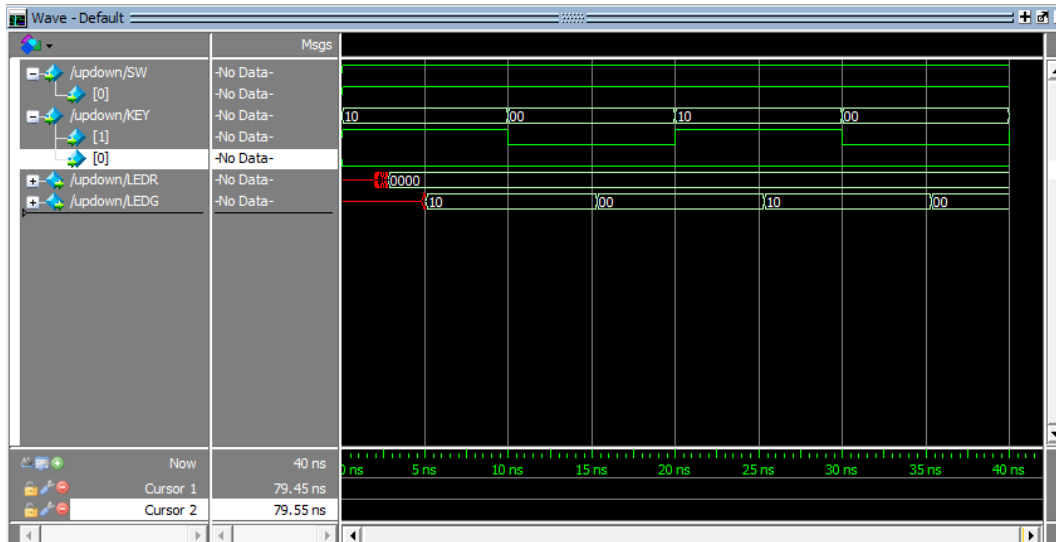


- 9) The sim-Default, Objects, and Processes (Active) windows will be loaded with design data from the updown.vo design file. If the Wave window is not open, open it by selecting **View > Wave** from the pull-down menu.

- 10) In the Objects window, select the signals that you would like to see in the Wave window. You should definitely select **SW**, **KEY**, **LEDR**, and **LEDG** since these are the input and output signals. Select **Add > To Wave > Selected Signals** from the pull-down menu to add the signals to the Wave window. (You can also right-click and select **Add > To Wave** to copy the signals into the Wave window.)
- 11) Change the radix for LEDR to hex by selecting it in the Wave window, right-clicking and selecting **Radix > Hexadecimal**;
- 12) Expand the **KEY** signal by clicking on the '+' next to it in the Wave window. Select **KEY[1]**, right-click and select **Clock....** Change the Period to **20 ns** and click **OK**. This will give a clock frequency of 25 MHz, which is reasonable for a timing simulation.
- 13) Select **KEY[0]**, right-click and select **Force....** Enter the value as '**0**' and select the Kind as **Drive** as shown in the Figure below. Leave the **Delay For** parameter at the default value of 0.



- 14) Follow the same procedure to set the **SW[0]** signal to '**1**'. Now we have initialized all the inputs so we are ready to simulate.
- 15) Run the simulation for 40 ns. You should see waveforms similar to those in the Figure below. Since **KEY[0]** is low, the **LEDR** value has been reset to 0x0000.



16) Force input KEY[0] to logic '1' and run the simulation for several clock cycles. You **MAY** need to select **No Force** first and then select Force to logic '1'. There are times when ModelSim-Altera changes do not materialize. You should see the LEDR value start increasing. Zoom in to see the delay between the rising edge of the clock (KEY[1]) and the change in the LEDR value.

17) Demonstrate your simulation waveforms to your TA and have your TA sign your verification sheet.

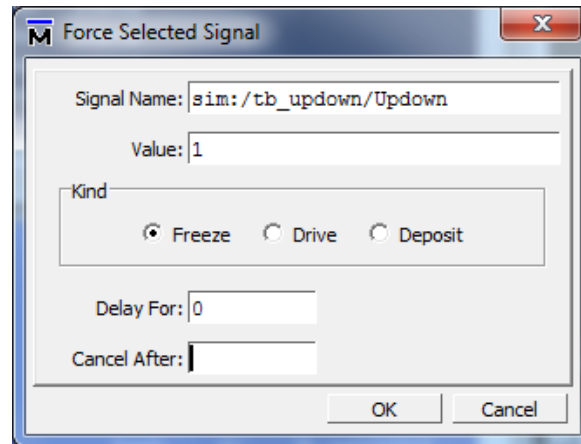
B. Timing Simulation Using a Testbench

- 1) Copy tb_updown.v into your ModelSim project folder for timing simulation (i.e. the simulation/modelsim subfolder).
- 2) Click on the Project tab at the bottom of the window. Select **Project > Add to Project > Existing File** from the pull-down menus and select tb_updown.v.
- 3) Add the following line (including the single quote mark) to the top of the tb_updown.v file so that the clock rate will be 20 ns rather than 20 ps:

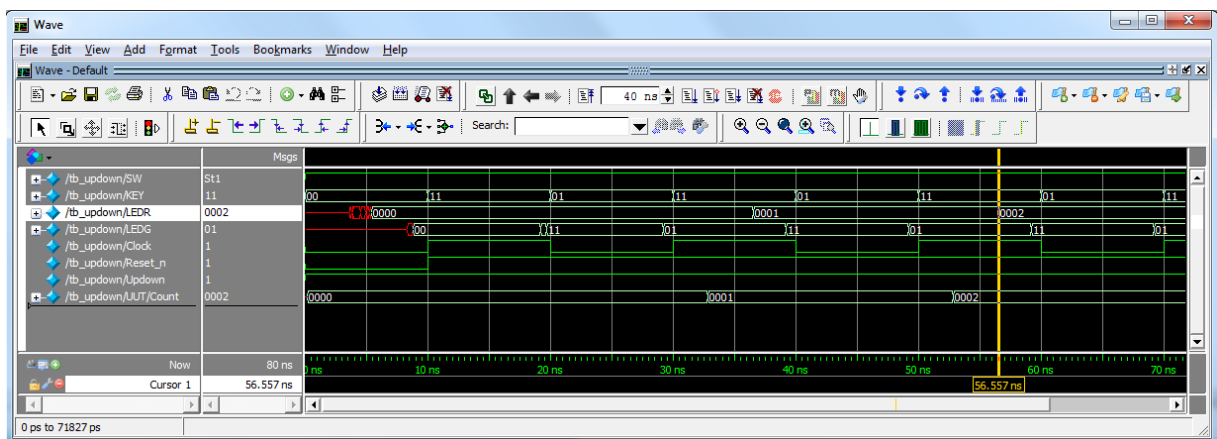
```
`timescale 1 ns/ 1 ns
```

- 4) Select **Compile > Compile All** from the pull-down menus. The files should compile with no errors.
- 5) Select **Simulate > Start Simulation** from the pull-down menus. From the Design tab, select **only** the tb_updown module in the work library as the Design Unit(s). From the Libraries tab, make sure that cycloneii_ver is added in the Search Libraries window, if it isn't already there. Click **OK**.

- 6) Open any windows that you need and aren't currently open by selecting them from the **View** pull-down menu.
- 7) Make sure to **select** and **highlight** the tb_updown Instance in the "sim - Default" window to update the Objects window with the signals from the tb_updown. Add signals to the Wave window and set the radix for LEDR to hex.
- 8) Force the signal Updown to a '1' as shown in the Figure below. Note that the Kind should be left on Freeze.



- 9) Run the simulation for 5 to 10 clock cycles. Zoom-In and view the delay between the rising edge of the KEY[1] (clock) signal and LEDR value change. The delay should be on the order of 6ns to 7ns.



- 10) Demonstrate your simulation waveforms to your TA and have your TA sign your verification sheet.

III. Testing Design on Altera DE2 Board

Once you have compiled the updown.v file with the correct pin assignments, you can download the design to the DE2 board just as you did after compiling a schematic design in the previous labs.

NOTE: Remember to import the pin assignments again in Quartus II every time you change something in the design. After importing the pin assignments remember to compile again.

A. Testing with Manual Clocking Using KEY[1]

- 1) Download the updown counter design to the DE2 board and verify that it works correctly. Demonstrate your working circuit to your TA and have your TA sign your verification sheet.

IV. Lab Exercises: Hex to Seven-Segment Decoder Circuits in Verilog

In this exercise, you will write Verilog code for a hex to 7-segment code converter that will be used to display the count value in hexadecimal on 7-segment displays on the Altera DE2 board. Figure 1 shows a block diagram of a hex to 7-segment display code converter. This code converter should be used to display hexadecimal characters (0-9, A-F) on the 7-segment display. The Altera DE2 board has eight seven-segment displays, named HEX0, HEX1,..., HEX7. Each segment is illuminated by driving its control signal to logic 0; that is, they are **active-low**. In the *DE2_pin_assignments.csv* file, the segments are identified by the indices 0 to 6 as shown in Figure 1. You should declare a 7-bit port for each HEX output, similar to:

output [0:6] HEX0, HEX1, ..., HEX7;

in your Verilog code so that the output names match the corresponding names in the pin assignments file.

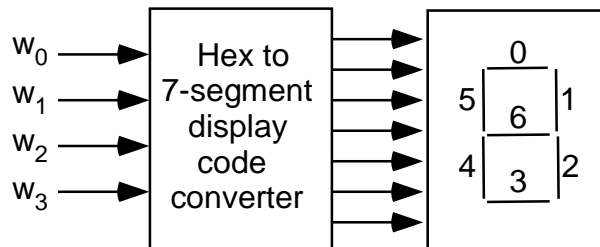


Figure 1. Code converter Block Diagram

A. Combinational Logic Design Using Assign Statements

Perform the following steps:

- 1) Generate a truth table showing w3 – w0 as inputs and segments seg0-seg6 as outputs. For example, the first couple of lines in the truth table are shown below. (Note: w3 is the most significant bit of the input signal.) The hex digits ‘B’ and ‘D’ will be displayed as lower-case ‘b’ and ‘d’ to avoid confusion with ‘8’ and ‘0’. The ‘6’ will have seg0 on to avoid confusion with ‘b’.

Hex input	w3	w2	w1	w0	seg0	seg1	seg2	seg3	seg4	seg5	seg6
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1

- 2) From the truth table, generate the Boolean equations for each output signal, seg0 – seg6. You may want to use a K-map to minimize the equation, but that is not required.
- 3) Write the Boolean equations into a Verilog module. A template is given below.

```

module hex_7seg (w, seg) ;
    input [3:0] w;
    output [0:6] seg;

    assign seg[0] = (~w[3] & ~w[2] & ~w[1] & w[0]) |
                  (~w[3] & w[2] & ~w[1] & ~w[0]) |
                  (w[3] & ~w[2] & w[1] & w[0]) |
                  (w[3] & w[2] & ~w[1] & w[0]);

    // assignments for seg[1] – seg[6]

endmodule

```

- 4) Do a functional simulation using ModelSim-Altera to verify the correctness of your design. Print the waveforms showing the code converter outputs for w[3..0] for values 0 through 15. Include the waveform in your lab report.
- 5) Change the up/down counter to a 32-bit counter by changing the parameter n from 16 to 32. Note that you will not be able to display the full 32-bit count on the red LEDs since the DE2 board doesn't have enough LEDs.
- 6) Instantiate 8, hex_7seg components into the updown counter and display the 32-bit count value on HEX7 – HEX0.
- 7) Compile your design in Quartus II and download the circuit to the DE2 board. Test your circuit using switches on the DE2 board. You should see the count in hexadecimal on the HEX7-HEX0 seven-segment displays. Demonstrate your working circuit to your TA and have your TA sign your verification sheet.

NOTE: If you used separate files for each module, remember to add all your Verilog files into the same Quartus II project. Otherwise, compile errors will arise since the compiler would not be able to resolve module to module call dependencies.

NOTE: In Quartus II, make sure the updown.v file is the Top-Level Entity, to do so, Click on Files tab in the project, Right Click on the updown.v file and select "Set as Top-Level Entity". Every project must have a top-level entity. The Top-Level Entity should be the highest level design file in your project.

NOTE: Remember to import the pin assignments again in Quartus II every time you change something in the design. After importing the pin assignments remember to compile again. If you do not do this step you will see the DE 2 board being programmed but will not function due to pin assignment mismatch.

B. Combinational Logic Design Using a Case Statement

1. Write a new hex_7seg module using a case statement as shown in the template below. The Verilog case statement allows you to directly specify the truth table of your logic function. Thus, you should have an easy time writing this code. Refer to the truth table you constructed in Part IV-A above.

```
module hex_7seg (  
    input [3:0] w,  
    output reg [0:6] seg  
);  
  
    //          012_3456 (segments are active-low)  
    parameter ZERO = 7'b000_0001;  
    parameter ONE  = 7'b100_1111;  
    // etc.  
    parameter F = 7'b011_1000;  
  
    always @(w)  
  
        case (w)  
            0: seg = ZERO;  
            1: seg = ONE;  
  
            // etc.  
            15: seg = F;  
        endcase  
  
endmodule
```

2. Perform a functional simulation of your Verilog design to verify your design's correctness. Simulate the entire design, including the updown counter with the

- hexadecimal output display. Modify the testbench to simulate your new design. Print a simulation waveform showing a complete counting cycle (0 – 15) and submit the printout with your lab report.
3. Compile your design in Quartus II and download the circuit to the DE2 board. Test your circuit using switches on the DE2 board. You should see the 32-bit count in hexadecimal on the HEX7-HEX0 seven-segment displays. Demonstrate your working circuit to your TA and have your TA sign your verification sheet.

Lab Report

Submit the following items for verification and grading:

- 1) Lab cover sheet with TA verifications for
 - 16-bit counter functional simulation (manual)
 - 16-bit counter functional simulation (testbench)
 - 16-bit counter timing simulation (manual)
 - 16-bit counter timing simulation (testbench)
 - 16-bit counter implementation on DE2 board using KEY[1] as the Clock
 - 32-bit counter implementation on DE2 with 7-segment displays (Assign statement implementation)
 - 32-bit counter implementation on DE2 with 7-segment displays (Case statement implementation)
- 2) Complete Verilog source code for Part IV, Lab Exercises. Submit your revised test bench module as well as the updown counter and hex to 7-segment display modules.
- 3) Functional simulation waveforms