Introduction to Verilog

Exercise Manual

Software Requirements:

Intel® Quartus® Prime Standard Edition software v18.0

ModelSim®-Intel FPGA Edition simulation tool version 10.5b

Link to lab files (for going through labs offline):

https://www.intel.com/content/www/us/en/programmable/customertraining/ilt/ihdl120/ihdl12
0-lab-18-0-v3.zip

Lab Overview

Introduction to Verilog

Objective:

- *Build a sequential 8 x 8 multiplier*

The objective of the following exercises is to build an 8 x 8 multiplier. The input to the multiplier consists of two 8-bit multiplicands (dataa and datab) and the output from the multiplier is a 16-bit product (product8x8_out). Additional outputs are a done bit (done_flag) and seven signals to drive a seven-segment display (seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, & seg_g).

This 8 x 8 multiplier requires four clock cycles to perform the full multiplication. During each cycle, a pair of 4-bit portion of the multiplicands is multiplied by a 4 x 4 multiplier. The multiplication result of these 4-bit slices is then accumulated. At the end of the four cycles (during the 5th cycle), the fully composed 16-bit product can be read at the output.

The following equations illustrate the mathematical principles supporting this implementation:

$$result[15..0] = a[7..0] * b[7..0]$$
$$= ((a[7..4] * 2^4) + a[3..0] * 2^0)$$
$$* ((b[7..4] * 2^4) + b[3..0] * 2^0)$$
$$= ((a[7..4] * b[7..4]) * 2^8)$$
$$+ ((a[7..4] * b[3..0]) * 2^4)$$
$$+ ((a[3..0] * b[7..4]) * 2^4)$$
$$+ ((a[3..0] * b[3..0]) * 2^0)$$

Figure 1 (below) illustrates the top-level block diagram of the 8 x 8 multiplier.
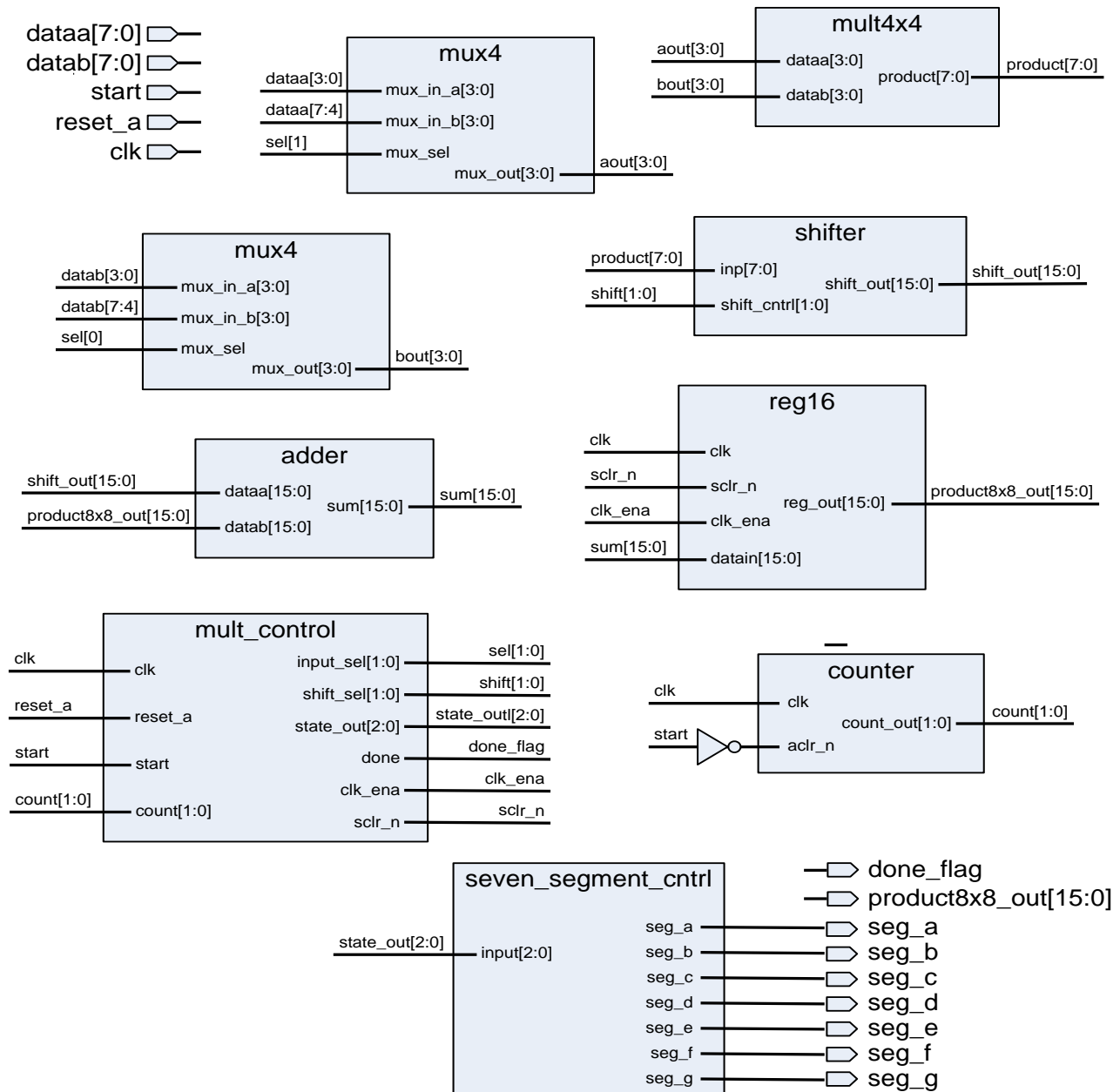
Figure 1 – 8 x 8 multiplier top level design block diagram

The labs are structured as a bottom-up design approach. In each of the first four exercises, you will use targeted features of the Verilog language to build the individual components of the 8 x 8 multiplier, compiling and simulating each component. Then, in Exercise 5 you will put everything together in a top-level design. You will then compile and simulate to verify the completeness of top-level design.

# Exercise 1a

Introduction to Verilog

Objectives:
- *Build a 16-bit adder using the '+' operator*
- *Practice coding basic module structure*

Step A: Unzip the exercise files and open an Intel Quartus Prime Project

____ 1. In a Windows Explorer window, navigate to the directory **C:\fpga_trn\VER**. If you see any subfolders already there, please delete them. They are from a prior class.

____ 2. Unzip the file named **ihdl120-lab-18-0-v3.zip** found in the **C:\fpga_trn\VER** directory. If you still cannot find this directory or file, ask your instructor for assistance. After double-clicking, in the WinZip dialog box, simply click **Unzip** to automatically extract the files into a newly created folder named **IVER10_1**. Close WinZip.

____ 3. In the Windows **Start** menu, go to the **Intel FPGA 18.0.0.614 Standard Edition** folder. Click **Quartus (Quartus Prime 18.0)** to start the software. Double-check from the **Help** menu (**About Quartus Prime**) that you are using the correct version of the tool.

____ 4. Open the adder project. From the **File** menu, choose **Open Project** (*not Open which only opens a single file instead of a project)*. Browse to the directory where you unzipped the labs and then descend into the **IVER10_1** subdirectory . This directory will be referred to as the **<Project Directory>** from now on.

____ 5. Navigate into the **lab1a** directory and select the file **adder.qpf** to open the project, click **Open**.

The project opens and you are ready to start coding your adder block

Step B: Write the code for a 16-bit adder
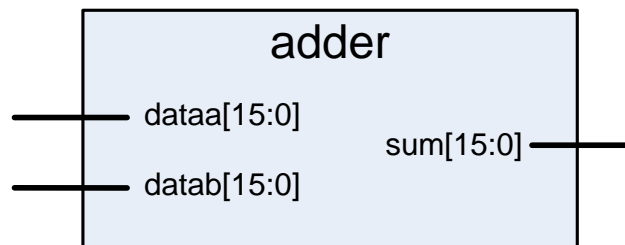
____ 1. Create a Verilog file using the Intel Quartus Prime text editor.

   a. From the **File** menu, select **New** or click on the ⬜ button.

   b. The **New File** dialog box will appear; select **Verilog HDL File**.

   c. Click **OK**.

____ 2.   Write the source code for a 16-bit adder using the '+' operator. Use the following information as a guide:

      a.   Use the names in the adder diagram to name your block and its ports (all lower-case)

      b.   Do not worry about rollover with this adder. This adder is already wide enough to account for all the values it will be adding together.

      c.   If you would prefer using a different text editor, please feel free to do so. Just make sure you save your Verilog file in the project directory.

____ 3.   Save the file as **adder.v**. From the **File** menu, select **Save** and save your Verilog file as **adder.v**. It should be located in the *<Project Directory>***lab1a** directory.

Step C: Synthesize the design & Check the code for correctness

____ 1.   Synthesize the design. From the **Processing** menu, select **Start** ⇒ **Start Analysis & Synthesis** OR click the   button.

This will perform checks on the source code to make sure it is using valid Verilog syntax as well as check the design for being synthesizable.

____ 2.   Correct any warnings and errors. Check the **Messages** window for any warning or error messages. Correct as needed.

*Be aware that most error messages point to the line number that is causing the error or it could be the line above it. You may also **right-click** on the message itself and choose **Help** to access the online help for clues on how to fix your warning or error. Of course, ask your instructor if you are unsure how to fix a particular warning or message. Repeat synthesis and error checking until the tool reports that "**Analysis & Synthesis was successful.**"*

Step D: Perform an RTL simulation

*You will now use the **ModelSim-Intel FPGA Edition** simulation tool to verify the functionality of your design. A Verilog testbench file (**adder_tb.v**) has already been created for you to provide the test vectors for your RTL simulation.*

____ 1.   Open the tool. In the Windows **Start** menu go to the **Intel FPGA 18.0.0.614 Standard Edition** folder. Click **ModelSim – Intel FPGA Edition 10.5b (Quartus Prime 18.0)** to start the program. Verify you are using the correct version of the tool.

____ 2.   Close the introductory window (if it appears).

____ 3.   Set the project directory. From the **ModelSim File** menu, select **Change Directory**. **Browse** to the location *<Project Directory>***lab1a.**

*A ModelSim macro file called a .DO file has been created for you. This file named **adder_tb.do** contains all of the steps to run the ModelSim tool and peform simulation. This includes:*

    a. *Creating a working library using the **vlib** command*

    b. *Compiling all of the Verilog files into the working library with the **vlog** command.*

    c. *Loading the simulator with the top-level testbench file using the **vsim** command.*

    d. *Opening the waveform window with the **wave** command.*

    e. *Adding target signals to the wave window (and formatting them) using the **add wave** command.*

    f. *Advancing simulation using the **run** command.*

*You may open the **adder_tb.do** file in a text editor if you wish to view the specific commands used.*

____ 4.   Execute the macro file. From the **ModelSim Tools** menu, select **Tcl ⇒ Execute Macro**. Select the file **adder_tb.do** and click **Open**.

*The **ModelSim** tool will now compile all of the Verilog files and start simulation. The waveform window will open (as a separate window) with the **dataa**, **datab**, and **sum** signals added so you can verify that your Verilog code is functioning correctly.*

____ 5.   Check simulation results for correct functionality. Bring the **Wave** window to the foreground. Right-click on the waveform and select **Zoom ⇒ Zoom Full**. Your results should look similar to the image below
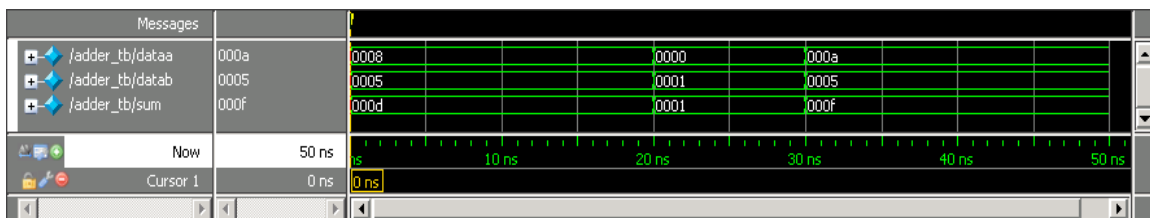
*If your simulation does not match the above, edit your Verilog code as needed and then save it. Re-run the **adder_tb.do** file (repeat #'s 4 and 5 above) to check your changes.*

____ 6.   End your simulation. From the **ModelSim Simulate** menu, select **End Simulation** OR type `quit -sim` in the **ModelSim Transcript** window. You will notice that when you select commands from the Modelsim pull down menus, the command is displayed in the transcript window. You can also use the up/down arrows to select prior comands to execute.

Exercise Summary

- *Coded a 16-bit adder block in Verilog using the '+' operator*
- *Simulated the Verilog code in the ModelSim tool to verify correct functionality*

END OF EXERCISE 1a

(Please continue to Exercise 1b)
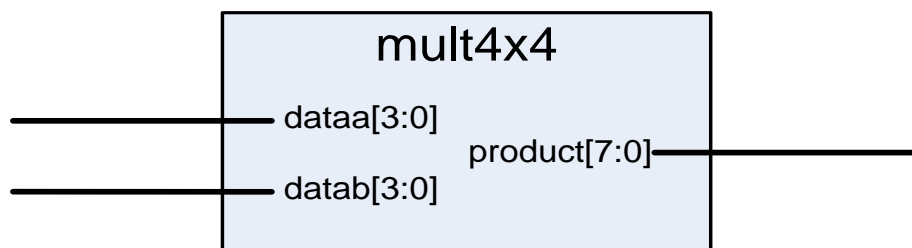
Exercise 1b

Introduction to Verilog

Objectives:

- *Build a 4x4 multiplier block using the '*' operator*
- *Synthesize and verify its operation*

Step A: Open a different project

____ 1.   Open the multiplier project. Back in the Intel Quartus Prime software, go to the **File** menu and choose **Open Project**. Browse to the directory ***<Project Directory>\lab1b*** and select the file **mult4x4.qpf**.

The project opens and you are ready to start coding your 4x4 multiplier block

Step B: Write the code for a 4x4 multiplier

```
                         mult4x4

              ─────── dataa[3:0]
                                   product[7:0]───────
              ─────── datab[3:0]
```

____ 1.   Create a Verilog file using the Quartus II text editor.

    a. From the File menu select New or click on the   button.

    b. The New File dialog box will appear; select Verilog HDL File.

    c. Click OK.

____ 2.   Write the source code for a **4x4 multiplier** using the **'*' operator**. Use the following information as a guide:

    a. The multiplier has two 4-bit multiplicand inputs and an 8-bit product output.

    b. Use the names in the diagram above to name your block and its ports (all lower-case)

____ 3.   Save the file as mult4x4.v. From the **File** menu, select **Save** and save your Verilog file as **mult4x4.v**. The file needs to be located in the ***<project directory>\lab1b*** directory.

Step 3: Synthesize the design & check the code for correctness

____ 1.   Synthesize the design. From the **Processing** menu, select **Start** ⇒ **Start Analysis & Synthesis** OR click the   button.

____ 2.   Correct any warnings and errors. Check the **Messages** window for any warning or error messages. Correct as needed using the message itself, the online help, the

class manual, and your instructor. Repeat synthesis and error checking until the tool reports that "**Analysis & Synthesis was successful**."

Step 4: Perform an RTL simulation

*You will now use the **ModelSim-Intel FPGA Edition** simulation tool to verify the functionality of your design. A Verilog testbench file (**mult4x4_tb.v**) has already been created for you to provide the test vectors for your RTL simulation.*

____ 1.   Set the project directory. From the **ModelSim File** menu, select **Change Directory**. **Browse** to the location *<Project Directory>*\lab1b.*

*Again, a ModelSim macro file named **mult4x4_tb.do** has been created for you to run the ModelSim tool and peform simulation.*

____ 2.   Execute the macro file. From the **ModelSim Tools** menu, select **Tcl ⇒ Execute Macro**. Select the file **mult4x4_tb.do** and click **Open**.

*The **ModelSim** tool will now compile all of the Verilog files and start simulation. the waveform window will open (as a separate window) with the **dataa**, **datab**, and **product** signals added so you can verify that your Verilog code is functioning correctly.*

____ 3.   Check simulation results for correct functionality. Bring the **Wave** window to the foreground. From the **View** menu, click **Zoom ⇒ Zoom Full**. Your results should look similar to the image below.
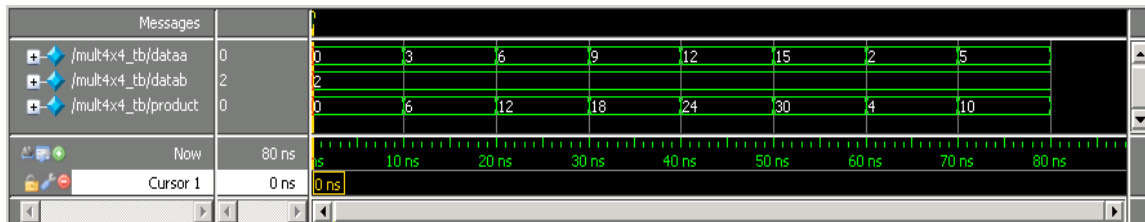
*If your simulation does not match the above, edit your Verilog code as needed and then save it. Re-run the **mult4x4_tb.do** file (repeat #'s 2 and 3 above) to check your changes.*

____ 4.   End your simulation. From the **ModelSim Simulate** menu, select **End Simulation** OR type `quit –sim` in the **ModelSim Transcript** window.

Exercise Summary
- *Coded a 4x4 multiplier block in Verilog using the '*' operator*
- *Simulated the Verilog code in the ModelSim tool to verify correct functionality*

END OF EXERCISE 1b

Exercise 2a

Introduction to Verilog

Objectives:

- *Build a 4-bit 2:1 multiplexer using the if-else statement*
- *Synthesize and verify its operation*

Step A: Open the Project

____ 1. Open the multiplexer project. In the Intel Quartus Prime software, go to the **File** menu and choose **Open Project**. Browse to the directory ***<Project Directory>\lab2a*** and select the file **mux4.qpf**.

The project opens and you are ready to start coding your 2-1 multiplexer block.

Step B: Write the code for a 2:1 multiplexer

____ 1. Create a Verilog file using the text editor:

   a. From the **File** menu select **New** or click the ⬜ button.

   b. The **New File** dialog box will appear; select **Verilog HDL File**.

   c. Click **OK**.

____ 2. Write the source code for a 4-bit 2:1 multiplexer using the if-else behavioral statement. Use the following information as a guide:

   a. The multiplexer has two 4-bit data inputs, a select line and a 4-bit output.

   b. Describe the following behavior: if **mux_sel is 0**, then choose **mux_in_a** for **mux_out**. if **mux_sel is 1**, then choose **mux_in_b** for **mux_out**.

   c. Use the names in the diagram above to name your block and its ports (all lower-case)

   d. Coding with behavioral statements requires using a procedural (e.g. always) block.

   e. All inputs to the procedural block (anything on the right-hand side (RHS) of an assignment operator within) should be listed in the procedural block's sensitivity list. The reasoning will be discussed later.

____ 3.   Save the file as **mux4.v**. From the **File** menu, select **Save** and save your Verilog file as **mux4.v**. It should be located in the ***<Project Directory>*\lab2a** directory.

Step C: Synthesize the design & check the code for correctness

____ 1.   Synthesize the design. From the **Processing** menu, select **Start ⇒ Start Analysis & Synthesis** OR click on the 🐌 button.

____ 2.   Correct any warnings and errors. Check the **Messages** window for any warning or error messages. Correct as needed using the message itself, the online help, the class manual, and your instructor. Repeat synthesis and error checking until the tool reports that "**Analysis & Synthesis was successful**."

Step D: Perform an RTL simulation

*You will now use the **ModelSim-Intel FPGA Edition** simulation tool to verify the functionality of your design. A Verilog testbench file (**mux4_tb.v**) has already been created for you to provide the test vectors for your RTL simulation.*

____ 1.   Set the project directory. From the **ModelSim File** menu, select **Change Directory**. **Browse** to the location ***<Project Directory>*\lab2a.**

*Use the ModelSim macro file named **mux4_tb.do**, created for you, to run the ModelSim tool and peform simulation.*

____ 2.   Execute the macro file. From the **ModelSim Tools** menu, select **Tcl ⇒ Execute Macro**. Select the file **mux4_tb.do** and click **Open**.

*The **ModelSim** tool will now compile all of the Verilog files and start simulation. The waveform window will open (as a separate window) with the **mux_in_a**, **mux_in_b, mux_sel**, and **mux_out** signals added so you can verify that your Verilog code is functioning correctly.*

____ 3.   Check simulation results for correct functionality. Bring the **Wave** window to the foreground. From the **View** menu, click **Zoom ⇒ Zoom Full**. Your results should look similar to the image below.
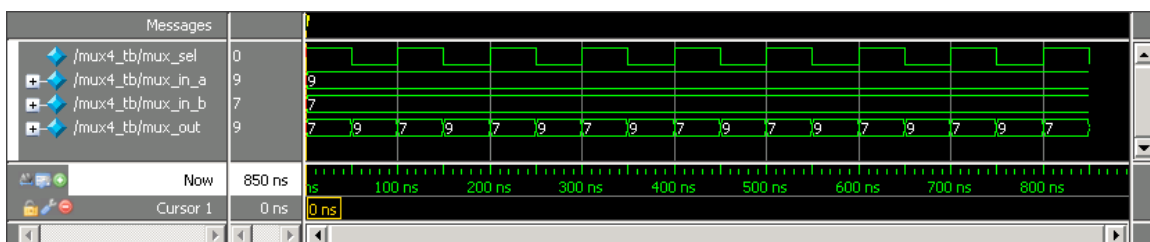
*If your simulation does not match the above, edit your Verilog code as needed and then save it. Re-run the **mux4_tb.do** file (repeat #'s 2 and 3 above) to check your changes.*

____ 4.   End your simulation. From the **ModelSim Simulate** menu, select **End Simulation**
OR type `quit -sim` in the **ModelSim Transcript** window.

Exercise Summary

- *Coded a 4-bit 2:1 multiplexer block in Verilog using the if-then behavioral statement*
- *Simulated the Verilog code in the ModelSim tool to verify correct functionality*

END OF EXERCISE 2a

(Please continue to Exercise 2b)

Exercise 2b

Introduction to Verilog

Objectives:

- *Build an 8-bit to 16-bit left shifter using the if-else statement*
- *Synthesize and verify its operation*

Step A: Open the Project

____ 1. Open the shifter project. In the Intel Quartus Prime software, go to the **File** menu and choose **Open Project**. Browse to the directory ***<Project Directory>*\lab2b** and select the file **shifter.qpf**.

The project opens and you are ready to start coding your 8-to-16 left shifter block.

Step B: Write the code for an 8-bit to 16-bit left shifter
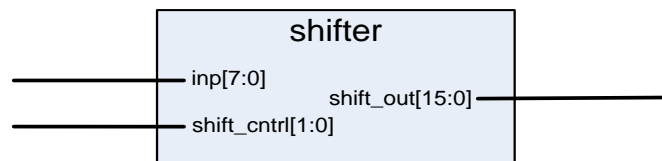
____ 1. Create a Verilog file using the text editor:

     a. From the **File** menu select **New** or click the ⬜ button.

     b. The **New File** dialog box will appear; select **Verilog HDL File**.

     c. Click **OK**.

____ 2. Write the source code for an **8-bit to 16-bit left shifter** using the **if-else behavioral statement**. Use the following information as a guide:

     a. The multiplexer has one 8-bit data input, a control line, and a 16-bit output.

     b. Describe the following behavior:

         i. When **shift_cntrl** is **0** or **3**, then <u>no shift</u> (i.e. **shift_out[7:0] equals input[7:0];** all other bits '0').

         ii. When **shift_cntrl** is **1**, then shift **input** to the left by <u>4 bits</u> within **shift_out** (i.e. **shift_out[11:4] equals input[7:0];** all other bits '0').

         iii. When **shift_cntrl** is **2**, then shift **input** to the left by <u>8 bits</u> within **shift_out** (i.e. **shift_out[15:8] equals input[7:0];** all other bits '0').

     c. When **shift_cntrl** is **2**, then shift **input** to the left by <u>8 bits</u> within **shift_out** (i.e. **shift_out[15:8] equals input[7:0];** all other bits '0').Use the names in the diagram above to name your block and its ports (all lower-case)

     d. Coding with behavioral statements requires using a procedural (e.g. always) block.

    e.   All <u>inputs</u> to the procedural block (anything on the RHS of an assignment operator within) should be listed in the procedural block's sensitivity list. The reasoning will be discussed later.

    f.   Try to simplify choices when possible, recognize similarities between choices.

____ 3.   Save the file as shifter.v. From the **File** menu, select **Save** and save your Verilog file as **shifter.v**. It should be located in the ***<Project Directory>\lab2b*** directory.

Step C: Synthesize the design & check the code for correctness

____ 1.   Synthesize the design. From the **Processing** menu, select **Start** ⇒ **Start Analysis & Synthesis** OR click the [icon] button.

____ 2.   Correct any warnings and errors. Check the **Messages** window for any warning or error messages. Correct as needed using the message itself, the online help, the class manual, and your instructor. Repeat synthesis and error checking until the tool reports that "**Analysis & Synthesis was successful**."

Step D: Perform an RTL simulation

*Use the **ModelSim-Intel FPGA Edition** simulation tool to verify the functionality of your design. A Verilog testbench file (**shifter_tb.v**) has been created for you to provide the test vectors for your RTL simulation.*

____ 1.   Set the project directory. From the **ModelSim File** menu, select **Change Directory**. **Browse** to the location ***<Project Directory>\lab2b.***

*Use the ModelSim macro file named **shifter_tb.do**, created for you, to run the ModelSim tool and peform simulation.*

____ 2.   Execute the macro file. From the **ModelSim Tools** menu, select **Tcl** ⇒ **Execute Macro**. Select the file **shifter_tb.do** and click **Open**.

*The **ModelSim** tool will now compile all of the Verilog files and start simulation. The waveform window will open (as a separate window) with the **shift_cntrl**, **input**, and **shift_out** signals added so you can verify that your Verilog code is functioning correctly.*

____ 3.   Check simulation results for correct functionality. Bring the **Wave** window to the foreground. From the **View** menu, click **Zoom** ⇒ **Zoom Full**. Your results should look similar to the image below.
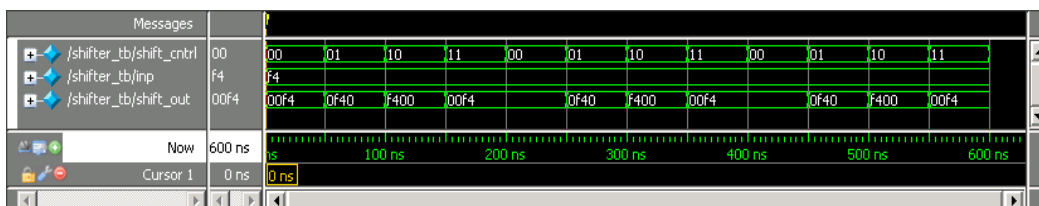
*If your simulation does not match the above, edit your Verilog code as needed and then save it. Re-run the **shifter_tb.do** file (repeat #'s 2 and 3 above) to check your changes.*

___ 4. End your simulation. From the **ModelSim Simulate** menu, select **End Simulation** OR type `quit –sim` in the **ModelSim Transcript** window.

Exercise Summary

- *Coded an 8-bit to 16-bit left shifter block in Verilog using the if-then behavioral statement*
- *Simulated the Verilog code in the ModelSim tool to verify correct functionality*

END OF EXERCISE 2b

Exercise 3

Introduction to Verilog

Objectives:

- *Build a 7-segment LED display controller using the case statement*
- *Synthesize and verify its operation*

Step A: Open the Project

____ 1.   Open the 7-segment LED display controller project. Back in the Intel Quartus Prime software, go to the **File** menu and choose **Open Project**. Browse to the directory ***<Project Directory>\lab3*** and select the file **seven_segment_cntrl.qpf**.

The project opens and you are ready to start coding your 7-segment LED display control block.

Step B: Write the code for a 7-segment LED display controller
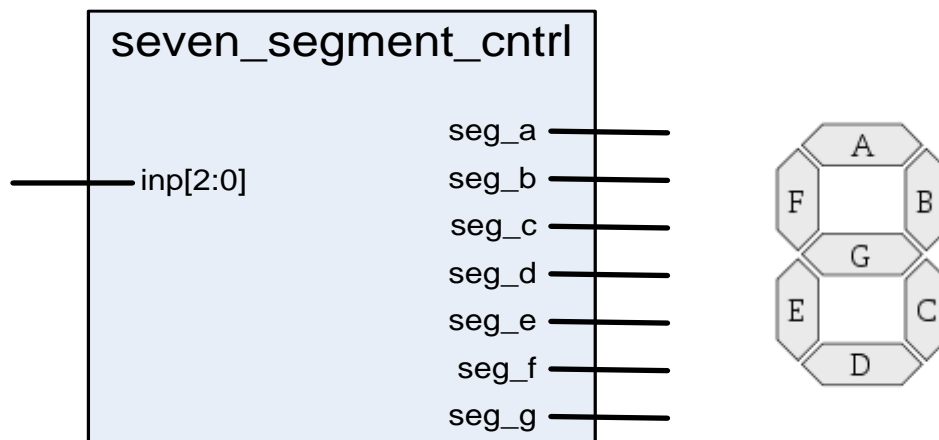
____ 1.   Create a Verilog file using the text editor:

    a.   From the File menu select New or click ⬜ button.

    b.   The **New File** dialog box will appear; select **Verilog HDL File**.

    c.   Click **OK**.

____ 2.   Write the source code for a **7-segment LED display controller** using the **case behavioral statement**. Use the following information as a guide:

    a.   The controller has one 3-bit data input and 7 single-bit outputs each controlling different segments of the 7-segment display

    b.   Describe the behavior as shown in the following table:

Inputs	Outputs							LED Display
inp [2:0]	seg_a	seg_b	seg_c	seg_d	seg_e	seg_f	seg_g	
000	1	1	1	1	1	1	0	0
001	0	1	1	0	0	0	0	1
010	1	1	0	1	1	0	1	2
011	1	1	1	1	0	0	1	3
All other values	1	0	0	1	1	1	1	E

   c. Use the names in the diagram above to name your block and its ports (all lower-case).

   d. Use the concatenation operator { } with the outputs to make assigning the output values easier.

   e. Coding with behavioral statements requires using a procedural (e.g. always) block.

   f. All inputs to the procedural block (anything on the RHS of an assignment operator within) should be listed in the procedural block's sensitivity list. The reasoning will be discussed later.

Step C: Synthesize the design & check the code for correctness

____ 1. Synthesize the design. From the **Processing** menu, select **Start** ⇒ **Start Analysis & Synthesis** OR click on the button.

____ 2. Correct any warnings and errors. Check the **Messages** window for any warning or error messages. Correct as needed using the message itself, the online help, the class manual, and your instructor. Repeat synthesis and error checking until the tool reports that "**Analysis & Synthesis was successful**."

Step D: Perform an RTL simulation

Use the ModelSim-Intel FPGA Edition simulation tool (or similar) to verify the functionality of your design. A Verilog testbench file (seven_segment_cntrl_tb.v) has been created for you to provide the test vectors for your RTL simulation.

____ 1. Set the project directory. From the **ModelSim File** menu, select **Change Directory**. **Browse** to the location ***<Project Directory>*\lab3**.

*Use the ModelSim macro file named **seven_segment_cntrl_tb.do**, created for you, to run the ModelSim tool and peform simulation.*

____ 2. Execute the macro file. From the **ModelSim Tools** menu, select **Tcl ⇒ Execute Macro**. Select the file **seven_segment_cntrl_tb.do** and click **Open**.

*The **ModelSim** tool will now compile all of the Verilog files and start simulation. The waveform window will open (as a separate window) with the **input, seg_a**, **seg_b**, **seg_c**, **seg_d**, **seg_e**, **seg_f**, and **seg_g** signals added so you can verify that your Verilog code is functioning correctly.*

____ 3. Check simulation results for correct functionality. Bring the **Wave** window to the foreground. From the **View** menu, click **Zoom ⇒ Zoom Full**. Your results should look similar to the image below.
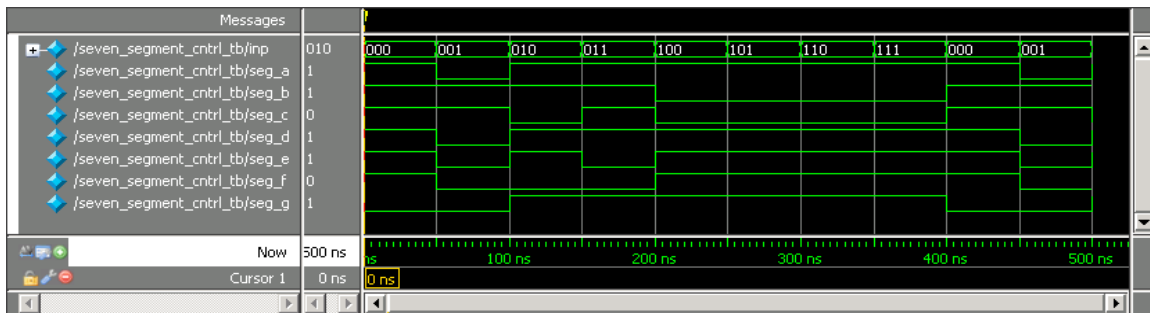
*If your simulation does not match the above, edit your Verilog code as needed and then save it. Re-run the **seven_segment_cntrl_tb.do** file (repeat #'s 2 and 3 above) to check your changes.*

____ 4. End your simulation. From the **ModelSim Simulate** menu, select **End Simulation** OR type `quit -sim` in the **ModelSim Transcript** window.

Exercise Summary

- *Coded a 7-segment LED display controller in Verilog using the **case** behavioral statement*
- *Simulated the Verilog code in the ModelSim tool to verify correct functionality*

## END OF EXERCISE 4

Exercise 4

Introduction to Verilog

Objectives:

- *Build a 16-bit register with synchronous control*
- *Synthesize and verify its operation*

Step A: Open the Project

____ 1.   Open the register project. In the Intel Quartus Prime software, go to the **File** menu and choose **Open Project**. Browse to the directory ***<Project Directory>\lab4a*** and select the file **reg16.qpf**.

The project opens and you are ready to start coding your 16-bit register block.

Step B: Write the code for a 16-bit register with synchronous control
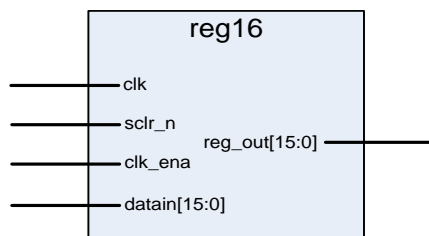
____ 1.   Create a Verilog file using the text editor:

    a. From the **File** menu select **New** or click the 🗋 button.

    b. The **New File** dialog box will appear; select **Verilog HDL File**.

    c. Click **OK**.

____ 2.   Write the source code for a **16-bit register with synchronous control** using a **sequential process**. Use the following information as a guide:

    a. The register has a clock, a 16-bit data input, a synchronous clear, a synchronous clock enable, and a 16-bit data output.

    b. Describe the following behavior:

        i. All transactions occur on the rising edge of a **clk**.

        ii. After a rising edge **clk** is detected, check to see if **clk_ena** is <u>high</u>.

        iii. If **clk_ena** is <u>high</u>, check to see if **sclr_n** is <u>low</u>. If so, then the register outputs are cleared.

        iv. If **clk_ena** is <u>high</u> and **sclr_n** is <u>not low</u>, then the register outputs are set equal to the register inputs.

        v. If **clk_ena** is low, do nothing.

    c. Use the names in the diagram above to name your block and its ports (all lower-case)

        d. Synchronous controls are not included in the sensitivity list.

        e. Unless explicitly changed, outputs to a procedural will retain their value from the previous execution.

____ 3.   Save the file as **reg16.v**. From the **File** menu, select **Save** and save your Verilog file as **reg16.v**. It should be located in the ***<Project Directory>\lab4a*** directory.

Step D: Perform an RTL simulation

*Use the **ModelSim-Intel FPGA Edition** simulation tool to verify the functionality of your design. A Verilog testbench file (**reg16_tb.v**) has been created for you to provide the test vectors for your RTL simulation.*

____ 1.   Set the project directory. From the **ModelSim File** menu, select Change Directory. Browse to the location **<Project Directory>\lab4a**.

*Use the ModelSim macro file named **reg16_tb.do**, created for you, to run the ModelSim tool and perform simulation.*

____ 2.   Execute the macro file. From the **ModelSim Tools** menu, select **Tcl ⇒ Execute Macro**. Select the file **reg16_tb.do** and click **Open**.

*The **ModelSim** tool will now compile all of the Verilog files and start simulation. The waveform window will open (as a separate window) with the **clk**, **sclr_n**, **clk_ena**, **datain**, and **reg_out** signals added so you can verify that your Verilog code is functioning correctly.*

____ 3.   Check simulation results for correct functionality. Bring the **Wave** window to the foreground. From the **View** menu, click **Zoom ⇒ Zoom Full**. Your results should look similar to the image below.
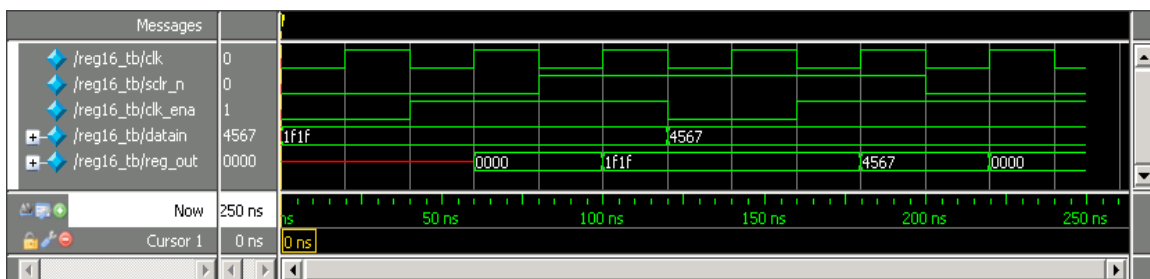
*If your simulation does not match the above, edit your Verilog code as needed and then save it. Re-run the **reg16_tb.do** file (repeat #'s 2 and 3 above) to check your changes.*

____ 4.   End your simulation. From the **ModelSim Simulate** menu, select **End Simulation** OR type `quit -sim` in the **ModelSim Transcript** window.

Exercise Summary

- *Coded a 16-bit register with synchronous controls in Verilog using a sequential procedural block.*
- *Simulated the Verilog code in the ModelSim tool to verify correct functionality*

END OF EXERCISE 4a

(Please continue to Exercise 4b)

Exercise 4b

Introduction to Verilog

Objectives:
- *Build a 2-bit counter with asynchronous control*
- *Synthesize and verify its operation*

Step A: Open the Project

____ 1.   Open the counter project. Back in the Intel Quartus Prime software, go to the **File** menu and choose **Open Project**. Browse to the directory ***<Project Directory>\lab4b*** and select the file **counter.qpf**.

The project opens and you are ready to start coding your 16-bit register block.

Step B: Write the code for a 2-bit counter with asynchronous control
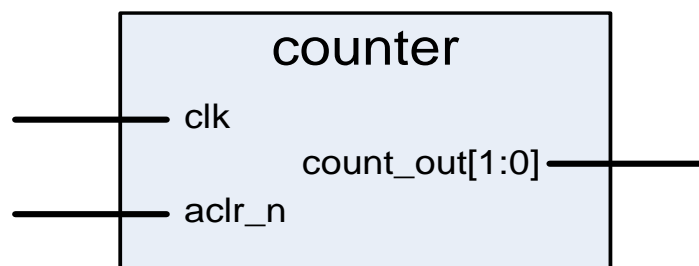
____ 1.   Create a Verilog file using the text editor:

   a.   From the **File** menu select **New** or click the ☐ button.

   b.   The **New File** dialog box will appear; select **Verilog HDL File**.

   c.   Click **OK**.

____ 2.   Write the source code for a **2-bit counter with asynchronous control** using a **sequential process**. Use the following information as a guide:

   a.   The register has a clock, an asynchronous clear, and a 2-bit data output.

   b.   Describe the following behavior:

      i.   The output of the counter goes to 00 immediately when **aclr_n** is <u>low</u>.

      ii.   If **aclr_n** is <u>not low</u>, then the output of the counter increments by 1 on every rising edge of **clk**.

   c.   Use the names in the diagram above to name your block and its ports (all lower-case)

   d.   Checking of asynchronous register control signals occurs <u>first</u> in a sequential procedural block.

   e.   Asynchronous controls <u>are</u> included in the sensitivity list.

____ 3.   Save the file as **counter.v**. From the **File** menu, select **Save** and save your Verilog file as **counter.v**. It should be located in the ***<Project Directory>\lab4b*** directory.

Step C: Synthesize the design & check the code for correctness

____ 4.   Synthesize the design. From the **Processing** menu, select **Start ⇒ Start Analysis & Synthesis** OR click on the 🛠 button.

____ 5.   Correct any warnings and errors. Check the **Messages** window for any warning or error messages. Correct as needed using the message itself, the online help, the class manual, and your instructor. Repeat synthesis and error checking until the tool reports that "**Analysis & Synthesis was successful**."

Step D: Perform an RTL simulation

*Use the **ModelSim-Intel FPGA Edition** simulation tool to verify the functionality of your design. A Verilog testbench file (**counter_tb.v**) has been created for you to provide the test vectors for your RTL simulation.*

____ 1.   Set the project directory. From the **ModelSim File** menu, select Change Directory. Browse to the location **<Project Directory>\lab4b**.

*Use the ModelSim macro file named **counter_tb.do**, created for you, to run the ModelSim tool and perform simulation.*

____ 2.   Execute the macro file. From the **ModelSim Tools** menu, select **Tcl ⇒ Execute Macro**. Select the file **counter_tb.do** and click **Open**.

*The **ModelSim** tool will now compile all of the Verilog files and start simulation. The waveform window will open (as a separate window) with the **clk**, **aclr_n**, and **count_out** signals added so you can verify that your Verilog code is functioning correctly.*

____ 3.   Check simulation results for correct functionality. Bring the **Wave** window to the foreground. From the **View** menu, click **Zoom ⇒ Zoom Full**. Your results should look similar to the image below.
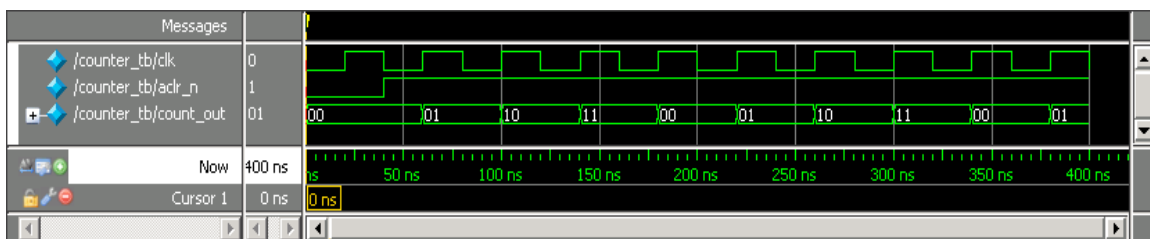
*If your simulation does not match the above, edit your Verilog code as needed and then save it. Re-run the **counter_tb.do** file (repeat #'s 2 and 3 above) to check your changes.*

____ 4.   End your simulation. From the **ModelSim Simulate** menu, select **End Simulation** OR type `quit –sim` in the **ModelSim Transcript** window.

Exercise Summary

- *Coded a 2-bit register with asynchronous control in Verilog using a sequential procedural block.*
- *Simulated the Verilog code in the ModelSim tool to verify correct functionality*
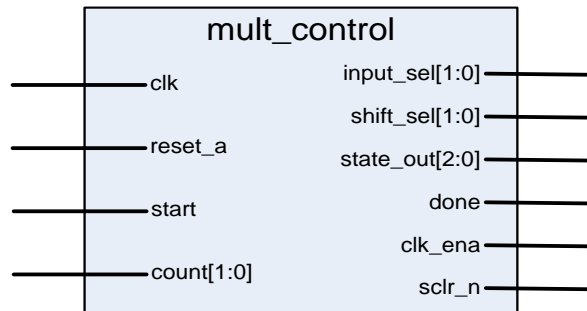
END OF EXERCISE 4b

(Please continue to Exercise 4b)

Exercise 5a

Introduction to Verilog

Objectives:

- *Examine a state machine implementation (no coding)*

You have now completed building all the components necessary to build the 8x8 multiplier, except for the controlling state machine.

*The project is in the lab5a folder called **mult_control**.*

*In this lab you are more on your own, there are no step-by-step instructions. The **mult_control.v** file contains the top level entity along with some other structures.*

*This state machine will manage all the operations that occur within the 8x8 multiplier using 6 defined states: **idle, lsb, mid, msb, calc_done**, and **err**. See the state diagram in Figure 5-1 for more definition of its behavior.*

*The state machine in the **LSB** state multiplies the lowest 4 bits of the two 8-bit multiplicands ((a[3..0] * b[3..0]) * 2^0). This intermediate result is saved in an accumulator.*

*The state machine in the **MID** state performs cross multiplication ((a[3..0] * b[7..4]) * 2^4) and ((a[7..4] * b[3..0]) * 2^4). This is done in successive clock cycles. The products of both multiply operations are added to the content of the accumulator as they are completed and clocked back into the accumulator.*

*The state machine in the **MSB** state multiplies the highest 4 bits of the two 8-bit multiplicands ((a[7..4] * b[7..4]) * 2^8). This product is added with the content of the accumulator and clocked back into the accumulator.*

This result is the final product:

```
result[15..0]   =     a[7..0] * b[7..0]
                =          ((a[7..4] * b[7..4]) * 2^8)
                +          ((a[7..4] * b[3..0]) * 2^4)
                +          ((a[3..0] * b[7..4]) * 2^4)
                +          ((a[3..0] * b[3..0]) * 2^0)
```

*The state machine in the **CALC_DONE** state asserts the **done_flag** output to indicate the final product has been calculated and is ready for reading by downstream logic.*

*The state machine in the **ERR** state indicates incorrect inputs have been received.*

*There are two inputs to the state machine: **start** and **count**. The **start** signal is asserted for once clock cycle to begin an 8x8 multiply operation on the next clock cycle. The **start** signal must only be asserted for one clock cycle. The **count** signal is used by the state machine to track the multiplication cycles.*

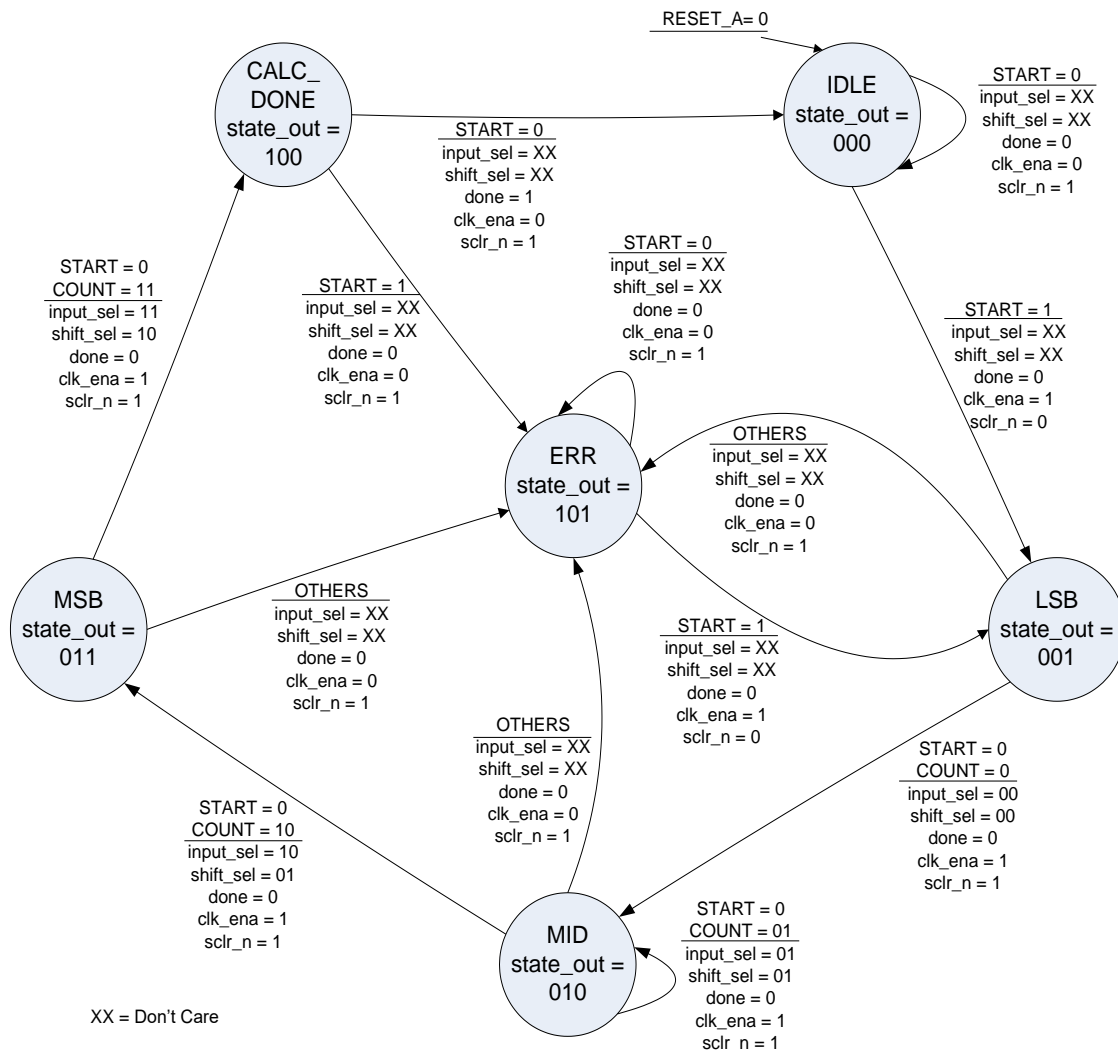*The outputs of **mult_control** state machine control the other blocks in the design from the prior labs.*

Figure 5-1: mult_control state machine state diagram

END OF EXERCISE 5a

(Please continue to Exercise 5b)

Exercise 5b

Introduction to Verilog

Objectives:

- *Complete the code to implement the 8x8 multiplier using Verilog structural modeling and instantiations*
- *Synthesize and verify its operation*

You now have all the building blocks necessary to complete the 8x8 multiplier.

Making use of the knowledge you have gained up to this point, you must now instantiate each component in a top-level design and connect all signals as shown in Figure 5-2. Once your design compiles and simulates successfully, you have completed the labs for the Introduction to Verilog class.
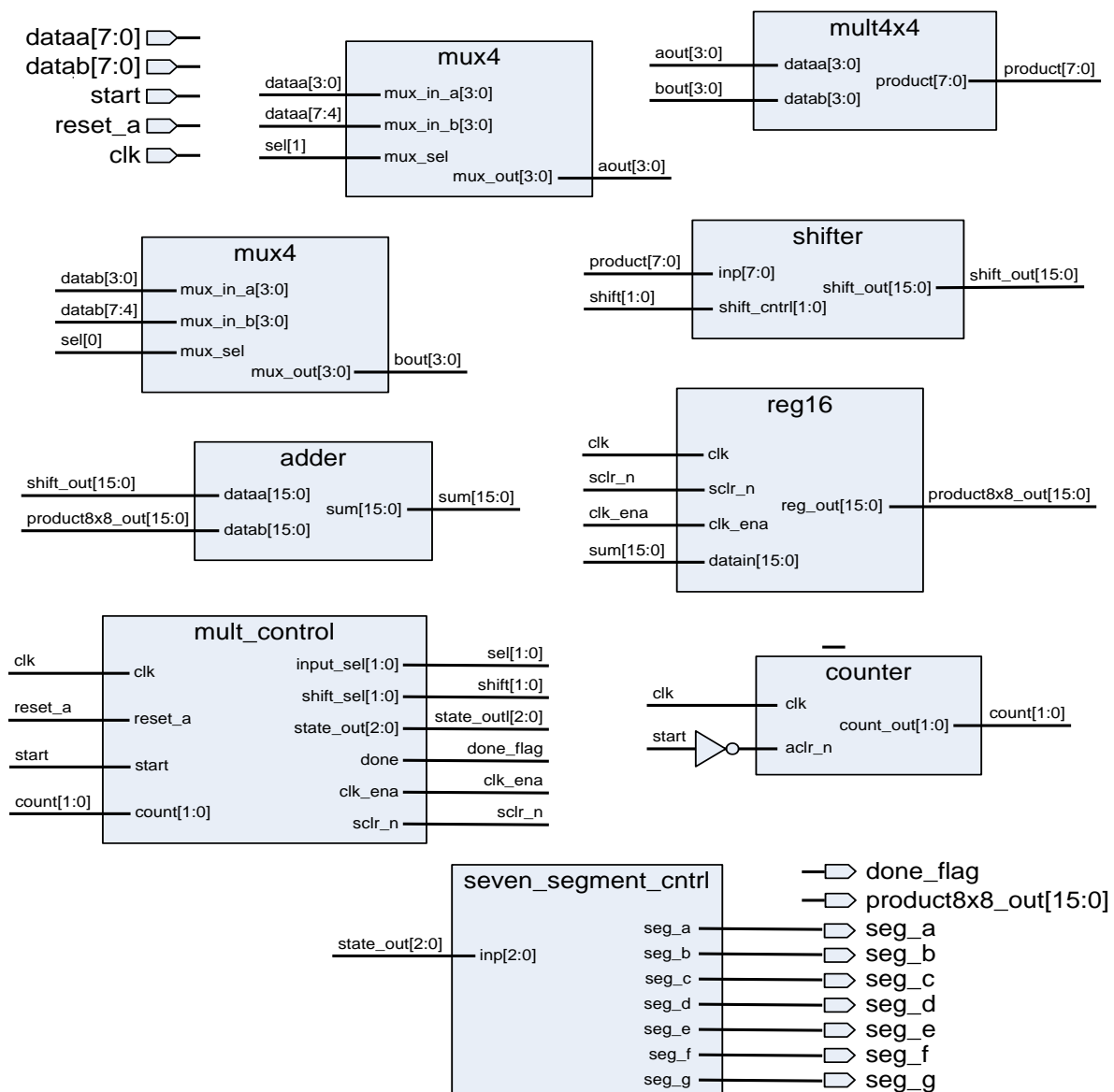

Figure 5-2 – 8 x 8 multiplier top level design block diagram

Step A: Open a Project

____ 1. Open the 8x8 multiplier project. In the Intel Quartus Prime software, go to the **File** menu and choose **Open Project**. Browse to the directory ***<Project Directory>\lab5b*** and select the file **mult8x8.qpf**.

The project opens and you can finish coding the 8x8 multiplier.

Step B: Finish the code for an 8x8 multiplier

____ 1. Create a Verilog file using the text editor:

    a. From the **File** menu select **New** or click the ▭ button.

    b. The **New File** dialog box will appear; select **Verilog HDL File**.

    c. Click **OK**.

____ 2. Write the source code to connect the **8x8 multiplier** using **Verilog structural instantiations**. Use the following information as a guide:

    a. Declare a top-level module named **mult8x8** with the top-level ports as shown in Figure 5-2.

    b. Create instantiations for **mux4** (<u>2</u> copies), **mult4x4**, **shifter**, **counter**, **mult_control**, **reg16**, **adder**, and **seven_segment_cntrl** based on the connections shown in Figure 5-2. Use **u6** as the instance name for mult_control. For other modules use any instance name you wish.

    c. This will require declaring new internal wires for connecting the blocks. Use the names from Figure 5-2.

____ 3. Save the file as **mult8x8.v**. From the **Quartus II File** menu, select **Save** and save your Verilog file as **mult8x8.v**. It should be located in the ***<Project Directory>\lab5b*** directory.

Step C: Synthesize the design & check the code for correctness

Since the other source files are in different directories, before synthesizing, you must directly add these other source files to this project.

____ 1. Add the source files from the other projects. From the **Project** menu, select **Add/Remove Files in Project**. Click on the button ▭ and browse to each of the subdirectories used for Exercises 1-5a to add each of the Verilog design files to this project. This includes:

    a. **adder.v (lab1a)**,

    b. **mult4x4.v (lab1b)**,

c. **mux4.v (lab2a)**,

d. **shifter.v (lab2b)**,

e. **seven_segment_cntrl.v (lab3)**,

f. **reg16.v (lab4a)**,

g. **counter.v (lab4b)**,

h. **mult_control.v (lab5a)**.

*MAKE SURE to click on the **Add** button each time so that the file appears in the **File name** list.*

____ 2. Synthesize the design. From the **Processing** menu, select **Start** $\Rightarrow$ **Start Analysis & Synthesis** OR click on the 🔺 button.

____ 3. Correct any warnings and errors. Check the **Messages** window for any warning or error messages. Correct as needed using the message itself, the online help, the class manual, and your instructor. Repeat synthesis and error checking until the tool reports that "**Analysis & Synthesis was successful**."

Step D: Perform an RTL simulation

*Use the **ModelSim-Intel FPGA Edition** simulation tool to verify the functionality of your design. A Verilog testbench file (**mult8x8_tb.v**) has been created for you to provide the test vectors for your RTL simulation.*

____ 1. Set the project directory. From the **ModelSim File** menu, select Change Directory. Browse to the location **<Project Directory>\lab5b**.

*Use the ModelSim macro file named **mult8x8_tb.do**, created for you, to run the ModelSim tool and perform simulation.*

____ 2. Execute the macro file. From the **ModelSim Tools** menu, select **Tcl** $\Rightarrow$ **Execute Macro**. Select the file **mult8x8_tb.do** and click **Open**.

*The **ModelSim** tool will now compile all of the Verilog files and start simulation. The waveform window will open (as a separate window) with all of the multiplier input and output signals added so you can verify that your Verilog code is functioning correctly. Dividers are used to separate different types of signals.*

____ 3. Check simulation results for correct functionality. Bring the **Wave** window to the foreground. Use the Zoom In tool to zoom until you can see the product output calculations. Your results should look similar to Figure 5-3 below. Note that each time **done_flag** goes high, the design is returning the product of the current inputs. Note that each time **done_flag** goes high, the design is returning the product of the current inputs.

*If your simulation does not match the above, edit your Verilog code as needed and then save it. Re-run the **mult8x8_tb.do** file (repeat #'s 2 and 3 above) to check your changes.*
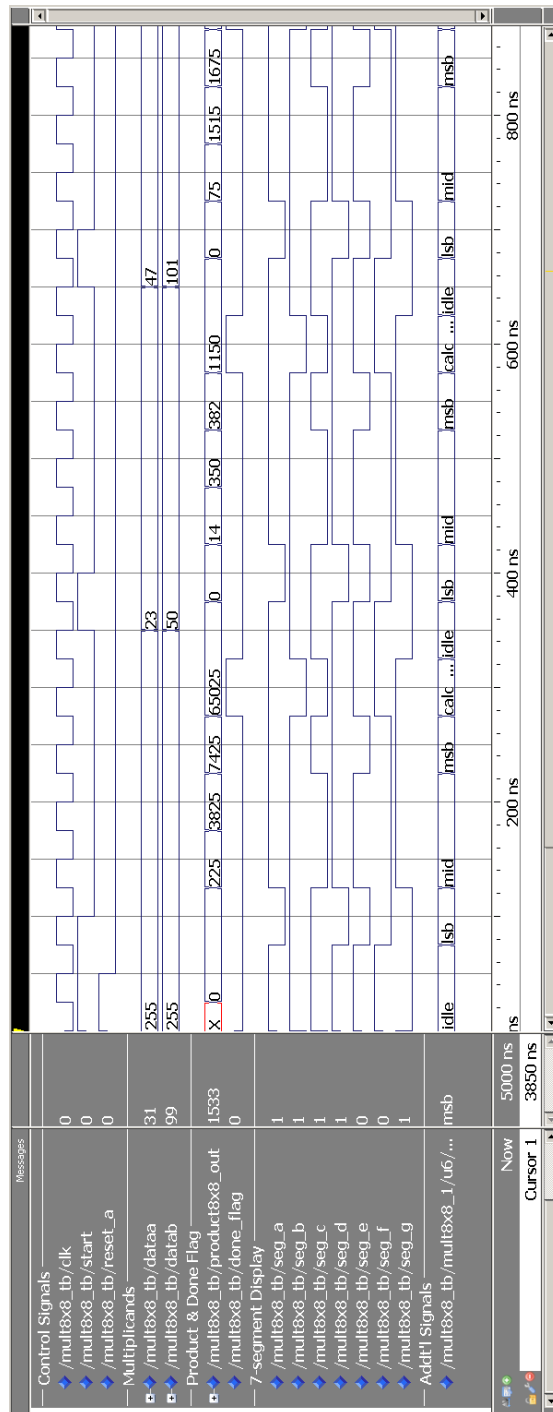

Figure 5-3: 8x8 multiplier simulation rsults (1st 900ns shown)

_____ 4.   End your simulation. From the **ModelSim Simulate** menu, select **End Simulation**
OR type `quit -sim` in the **ModelSim Transcript** window.

Exercise Summary

- *Finished coding the top-level 8x8 design by using structural modeling and component instantiations.*
- *Simulated the Verilog code in the ModelSim tool to verify correct functionality*

END OF EXERCISE 5b