Rensselaer EE EPE CSE
Department Of Electrical, Computer, & Systems Engineering

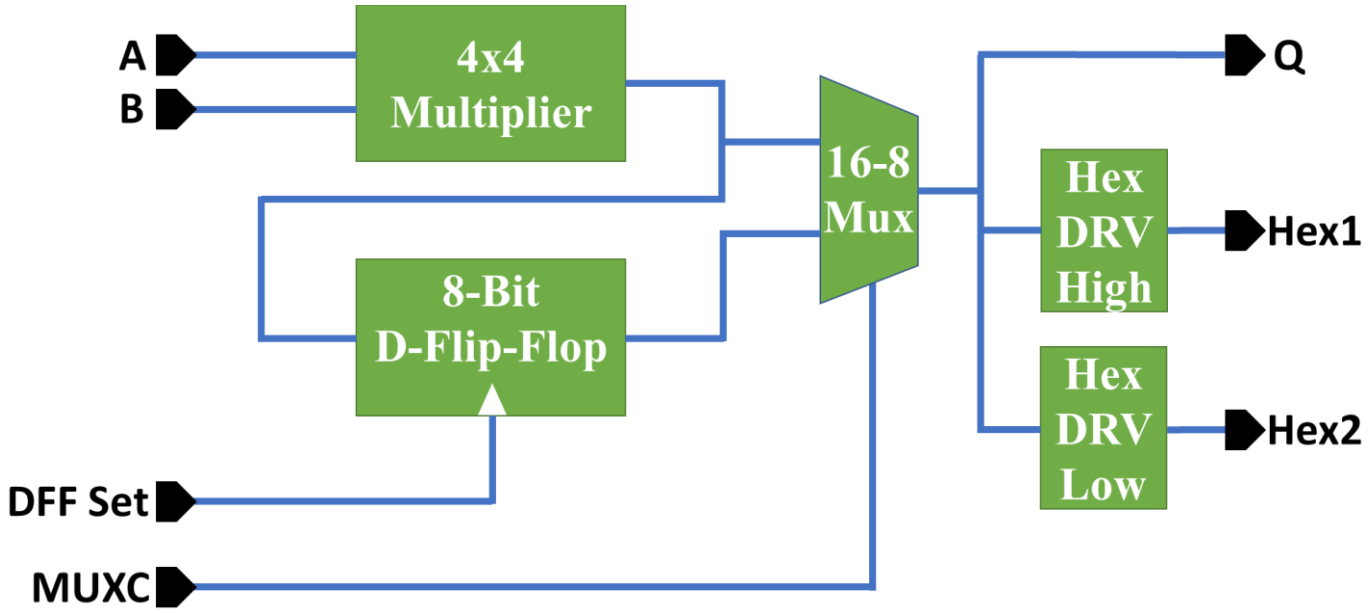
ECSE 4770 Computer Hardware Design: Lab 1-Verilog Fundamentals

Prepared By: Russell Kraft & Kurt English

Rev A

1. INTRODUCTION

This introductory lab assumes you have successfully completed the IVerilog D-Flip-Flop (DFF) simulation and Tutorial 1 Quartus Setup. We will now create and instantiate multiple Verilog modules within a Quartus schematic (.bdf) file to create a 4x4 multiplier with output save and display capability. This lab will demonstrate the fundamental Verilog **assign** and **always** logic definition methods, for creating combinational and sequential logic respectively. Students will use these operators to define behavior for a 4x4 mult iplier, an 8-bit DFF, and an 16-8 multiplexor. Hex display encoding Verilog files will be provided.



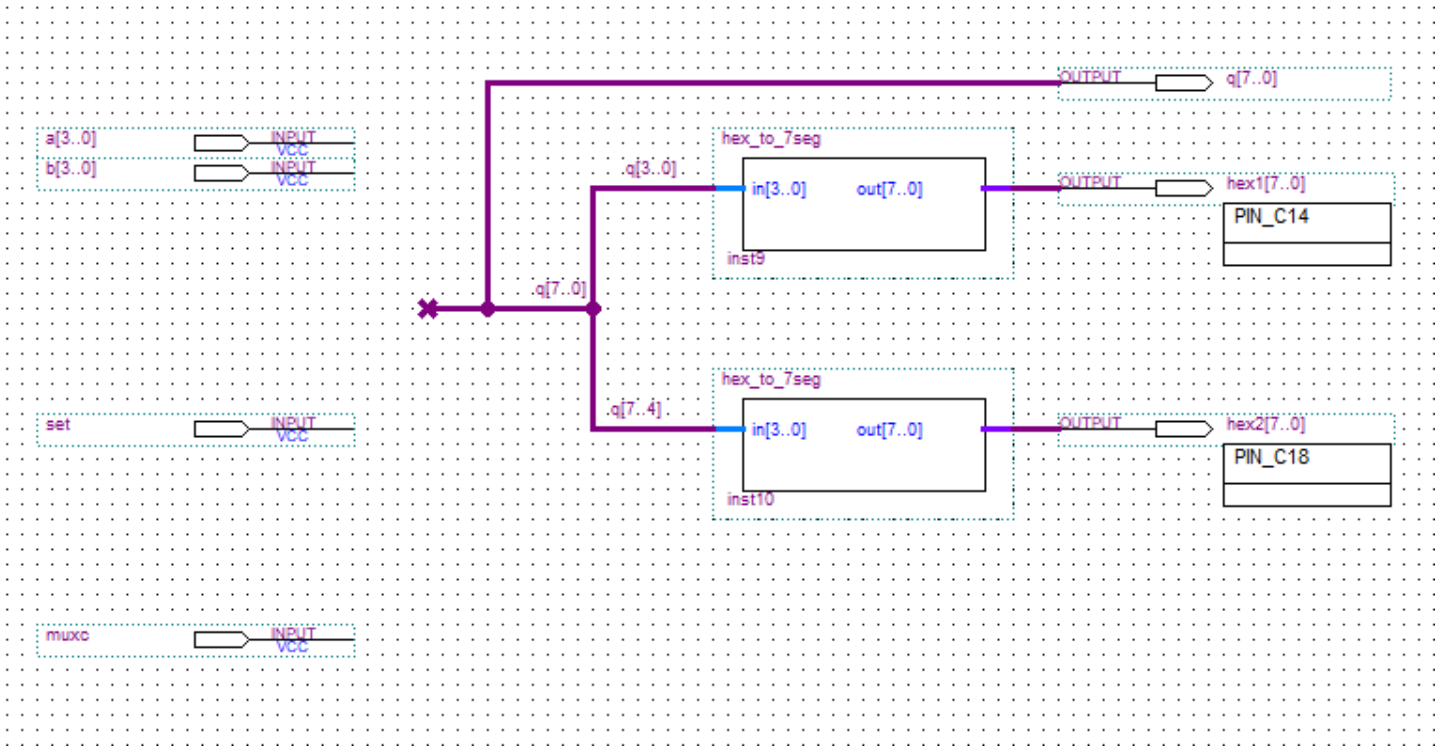
2. LOGIC CIRCUIT REQUIREMENTS & PROCEDURE

Students are expected to implement this design on their DE10-Lite FPGA boards, where multiplier operands **A** and **B** are input from the slide switches, and pushbuttons set the multiplexor control (**MUXC**) output select, as well as the output save (**DFF Set**) input. As the output can be 8 bits long, two hex displays are required to display the output.

If students do not have access to a DE10-Lite FPGA board or an FPGA with slide switch inputs and hex display outputs, notify TAs immediately so that alternative requirements can be arranged.

Begin the lab assignment by restoring the **lab1_student.qar** Quartus archive project located on LMS. It is recommended that students first modify the Lab 0 DFF to meet this lab's requirements, then create and place the symbol. Proceed creating Verilog for all 3 Verilog logic blocks, then wire the logic blocks together.

A partial top level .bdf schematic file is already provided. Students are to add and route the missing multiplier, DFF, and m ux logic blocks. Additionally, students must assign pins to the input signals so that they function correctly with the DE10-Lite FPGA board. Hex display output pin assignments are already provided.



Write Verilog for the multiplier, DFF, and multiplexor. Symbol files are to be generated and routed into the top level .bdf schematic file. Input and output connections are the following:

Signal Name	Width	Function
a	4	Multiplier input 1
b	4	Multiplier input 2
q	8	Simulation output
muxc	1	Select multiplier or DFF output for hex displays
set	1	Update DFF with current multiplier output
hex1	8	Lower nibble hex display control
hex2	8	Upper nibble hex display control

A brief report is required that lists your Verilog code with functionality explained, a timing diagram, and an analysis showing that your additions meet lab specifications. Further details are located at the end of this document.

3. VERILOG OVERVIEW

Verilog is a Hardware Description Language (HDL) with C-like syntax for implementing high level hardware descriptions into FPGA boards and fabricated logic circuits. Individual Verilog circuits are defined in **modules**, where combinational logic declared within modules are defined with **assign** statements, and sequential logic with **always** statements.

3.1 Verilog modules

Verilog module definitions obey the following syntax:

```
module "module name" ("input/output 1", "input/output 2", "...")
        "logic definitions"
endmodule
```

From Lab 0, your DFF module definition was as follows:

```
module dff(input d, input clk, input reset, output reg q, output q_b);
        "logic definitions"
endmodule
```

In the above syntax, the module name is "dff". The five arguments within the parenthesis declare the inputs and outputs for the module, where **d**, **clk**, and **reset** are inputs, and **reg q**, and **q_b** are outputs.

3.2 always block sequential logic

From lab 0, the behavioral description within the dff module contains the following **always** block:

```
always @(posedge clk, posedge reset) begin
        if (reset)
                q<=0;
        else
                q<=d;
end
```

The command **always @** states that this is a sequentially logic block. What follows within the parenthesis is the sensitivity list, which are the conditions that trigger an update of logic within the always block. In the example above, either a positive edge on **clk** or **reset** will trigger an update on **q**.

Operations within an **always** block can occur sequentially, or in parallel. This is denoted by the use of **<=** vs **=**, where **<=** denotes parallel operations and **=** sequential operations. For example:

```
x = 0;
y = 1;
```

x and y are set sequentially.

```
x <= 0;
y <= 1;
```

x and y are set simultaneously.

It is generally best to use the parallel **<=** when possible, as this is simpler to implement on the FPGA with less logic required.

Remember from the **Verilog modules** section that the dff from Lab 0 had an output named **reg q**. This is important, as outputs set from an **always** block must be registered, or else it is not possible to save their state.

always blocks can output to internal registers as well as output registers. To define a register, use the following syntax:

reg "register name";

3.3 assign statement combinational logic

While **always** blocks output to registers on a criteria met from the sensitivity list, logic defined from **assign** statements output to wires, and will update immediately on and change in input. Inputs to assign statements can either be other wires or registers. Recall the following **assign** statement from Lab 0:

assign q_b = ~q;

In the above, **q_b** is an input wire, and **q** is the **q** output register. The **~** is a bitwise invert operation. Because **q_b** was defined as an unregistered output, it is automatically considered to be a wire, which is valid output for the **assign** statement. In the event that **q_b** was not an unregistered output, it would have been necessary to define it as a **wire** with the following syntax:

wire q_b;

It is possible to define conditional statements within an **assign** statement. This is very useful for logic such as multiplexors. The following is the syntax for a conditional **assign**:

assign "output" = "conditional" ? "action if condition true" : "action if condition false";

The key characters above are the **?** and **:**. The **?** signifies that there is a conditional statement, and a following action for the true and false cases. The **:** signifies the separation between the true and false actions. Below is an example:

assign q = sel ? a : b;

In the above, **q** is assigned to the value of **a** if **sel** is true, or **q** is assigned to **b** if **sel** is false.

3.4 Verilog Buses

In Verilog inputs, outputs, wires, and registers can all be multi-bit bus connections. In all cases this is done with a square bracket syntax. Below is an example of a module named **logic** with a 4-bit bus input, and a 2-bit bus output:

module logic(input [3:0] a, output [1:0] b);

Next is a wire declaration for a 4-bit wire bus named **buswire**:

wire [3:0] buswire;

Logic operations can operate on busses, or subsets of busses. Below is an example where an assign statement sets only the lower 4 bits of a wider bus:

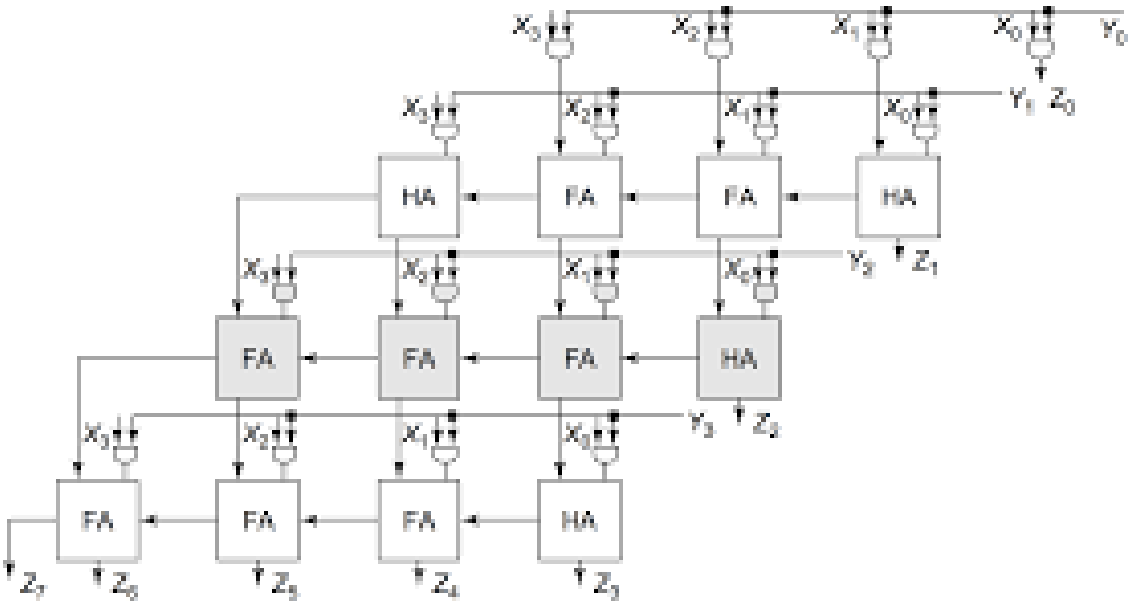
assign q[3:0] = a;

If the full bus width results in a matching width operation, there is no need to specify the bus width. In the example above, **a** at full width is only 4 bits, so it does not need to have selected bits defined.

3.5 Multiplier, DFF, Mux Verilog Guidelines

3.5.1 Multiplier

In traditional logic design a 4x4 multiplier would be implemented with a Wallace multiplier, requiring 16 AND gates, 8 full-adders, and 4 half-adders.



Adapted from University of Pennsylvania Department of Electrical and System Engineering

In an HDL language like Verilog, we can define 4x4 multiplication with the following, where Z is an 8-bit wire, and X and Y are 4-bit wires:

```
assign Z = X * Y;
```

3.5.2 DFF

The DFF from Lab 0 can be adapted for this assignment. Reset and inverted Q output can be deleted. Input **d** and output **q** must be expanded to 8 bits.

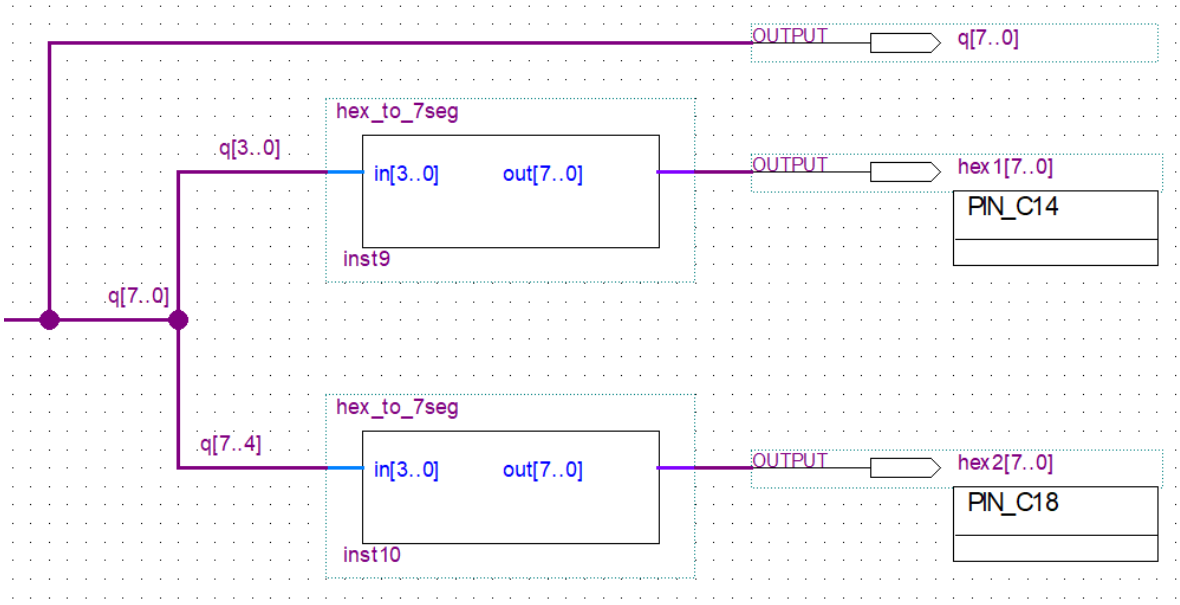
3.5.3 Multiplexor

The 16-8 bit multiplexor can be implemented using the **? :** conditional **assign** syntax as previously mentioned.

4. QUARTUS

4.1 Bus Routing

When connecting cells together with data paths wider than one bit, it is necessary to use the **Orthogonal Bus Tool** for routing instead of the **Orthogonal Wire Tool** as used in Tutorial 1. An example of this is provided in the initial Lab 1 .bdf file, where buses are already present for the provided hex display output.



Labeling busses is necessary when signals are split off from a wider bus. In the above image you can see hex display 1 receives **q** output bits 0 through 3, while hex display 2 receives **q** output bits 4 through 7.

4.2 Verilog File Symbol Creation

Creation of a .bsf symbol file from Verilog code is identical to that of creating a symbol from a .bdf schematic as should in Tutorial 1. With the Verilog file selected, go to **File → Create / Update → Create Symbol Files for Current File.**

5. REPORT SUBMISSION

Write a report with the following content:

- Brief introduction/abstract and description of the lab objective
- Complete top level circuit schematic
    - Quartus screenshot is sufficient
- Complete Verilog code for the following:
    - 4x4-bit multiplier
    - 8-bit D-Flip-Flop
    - 16-8-bit Multiplexor
- For each Verilog file, explain in 2-3 sentences how your Verilog functions and meets the required behavior
- Correct simulation waveform output demonstrating the following:
    - Correct multiplier output with non-zero operands
    - Correct DFF behavior
        - Storage
        - Read
    - Correct multiplexor behavior
        - Multiplier output
        - DFF output
- Two images of correct FPGA output
    - Correct multiplier output with non-zero operands
    - Correct DFF output with non-zero operands
        - Do not need to show setting the DFF, just the output

Submit your report as a PDF on LMS under the Assignments → Lab 1 Report submission section.

6. ADDITIONAL RESOURCES

An excellent external online resource for how to write Verilog is located at the following link:

http://www.asic-world.com/verilog/index.html

This link contains many helpful tutorials and examples on writing Verilog, which extend well beyond examples presented in this lab assignment.

Additionally, the DE10-Lite FPGA documentation contains many additional resources, most notable for this lab is the pin routing for the switches, pushbuttons, and hex displays:

https://lms.rpi.edu/webapps/blackboard/execute/content/file?cmd=view&mode=designer&content_id=_185891_1&course_id=_7160_1