LAB 3: Hierarchical Circuits, Timing, and Using the VGA Port

Objective: The first part of this lab will demonstrate how to build combinational arithmetic circuits in Verilog and how to automatically verify small combinational circuits using a testbench. The second will serve as a brief introduction to using the VGA port on the DE10-Lite board.

Prelab
Write your solutions to the following:

1. Write a behavior Verilog module for a full adder.
2. Draw the block diagram for an 8-bit unsigned adder.
3. Draw the block diagram for an 8-bit unsigned multiplier. Use 8-bit ripple-carry adders instead of full adders in your diagram. Use bus notation where appropriate. To simplify your diagram, you may only use one AND gate per row. The implicit notation here is that \((A[7:0] \text{ AND } B[0])\) implies that \(B[0]\) is AND’ed bitwise with \(A[7:0]\). Write major bus names on your diagram.
4. For the 8-bit unsigned multiplier, give the minimum and maximum value of the input operands and the minimum and maximum value of the output product. How many bits are needed at the output of the multiplier to ensure that all possible products can be fully represented?
Part 1 – Building a Combinational Multiplier

In this section, you will design, implement, and verify a combinational 8-bit unsigned multiplier. The multiplier you will be creating works essentially the same way you would multiply two numbers by hand. A block diagram for the multiplier you are to design is shown in Figure 1. The modules labeled “FA” are Full-Adders.

One way to design a multiplier involves instantiating each full-adder manually and explicitly connecting each component together. While this approach will work, it quickly becomes tedious and error prone as the size of the circuit increases. For example, a 4-bit unsigned multiplier like the one shown in Figure 1 only requires 12 full-adders. However, an 8-bit unsigned multiplier like the one you will be creating requires 56 full-adders. A 64-bit multiplier would require 4032 adders!

Instead, we will approach the multiplier design hierarchically. First, we will design a single full-adder circuit. Next, that full-adder circuit will be used to create an 8-bit Ripple Carry Adder (RCA) module. Finally, the RCA module will be used to create the multiplier.

A) Design an 8-bit Ripple Carry Adder

A full-adder (FA) has inputs \(a, b,\) and \(c_i\) (carry-in) and produces outputs \(s\) (sum) and \(c_o\) (carry-out). Its Boolean expressions are as follows:

\[
    s = a \oplus b \oplus c_i \\
    c_o = ab + ac_i + bc_i
\]

A RCA is used to add multi-bit numbers and involves instantiating multiple full-adders. The block diagram for a 4-bit RCA is shown in Figure 2.

Perform the following steps using ModelSim:

1. Write a behavioral module for a full-adder in Verilog. (Hint – an assign statement can be used to compactly describe each output.)
2. Create a module in Verilog that **structurally** implements an 8-bit RCA by instantiating 8 full-adder circuits. The 8-bit RCA should have the following ports:

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Number of Bits</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>8</td>
<td>Input</td>
<td>Input operand</td>
</tr>
<tr>
<td>b</td>
<td>8</td>
<td>Input</td>
<td>Input operand</td>
</tr>
<tr>
<td>ci</td>
<td>1</td>
<td>Input</td>
<td>LSB Carry-In</td>
</tr>
<tr>
<td>s</td>
<td>8</td>
<td>Output</td>
<td>Output Sum</td>
</tr>
<tr>
<td>co</td>
<td>1</td>
<td>Output</td>
<td>MSB Carry-out</td>
</tr>
</tbody>
</table>

3. Write a testbench that simulates and verifies your design for all possible inputs, assuming ci is tied to 0. Since each operand is 8-bits, this means that there are \(2^{16} = 65536\) possible input combinations. This is close to the biggest circuit we can verify by testing all input combinations. If the circuit were much larger, we would have to rely on hand-selected or random input test vectors to verify the correct functionality of our circuit.

In Lab 2 you created a self-checking testbench by precomputing the expected result. For a circuit this large, trying to generate all 65536 possible outputs by hand would take a very long time. Instead, use the ‘+’ operator in Verilog to generate the expected output. Your testbench should include two nested for loops – one loop for each of the input operands.

**B) Design an 8-bit Multiplier**

Perform the following steps using ModelSim:

1. Use Verilog to describe an 8-bit multiplier by instantiating seven of your 8-bit RCAs. On page 225 of your textbook, the authors show how to do this for a 4-bit multiplier. Use their design as a template for your own design. The multiplier should have the following ports:

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Number of Bits</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>8</td>
<td>Input</td>
<td>Input Operand</td>
</tr>
<tr>
<td>b</td>
<td>8</td>
<td>Input</td>
<td>Input Operand</td>
</tr>
<tr>
<td>p</td>
<td>16</td>
<td>Output</td>
<td>Output Product</td>
</tr>
</tbody>
</table>

2. Modify your testbench from Part 1A to exhaustively simulate and verify your multiplier module. The only thing that should change is the method of verification. Verify the outputs of your multiplier using the Verilog ‘*’ operator.

**C) Implementation and Verification on the DE10-Lite**

Once you have verified your multiplier via functional simulation, it’s time to test it on the DE10-Lite. Some wrapper code has been provided to help you interact with your design on the board, verify the results, and to allow Quartus to perform accurate timing simulation. Download **lab3.zip** from the course web page. To use the provided code, perform the following steps which are also described in a little more detail on the 180B web page:

http://www.ece.ucdavis.edu/~bbaas/180b/lab/downloading.to.board.html

1. Run SystemBuilder. Name the project “lab3_multipler”.

3
2. Open the project you just created using Quartus. Replace the code in the top-level module with the code in `lab3_multiplier.v`. Add the following files to your Quartus Project:
   - `bcd.v` – (sequential module for converting a binary number into binary-coded decimal (BCD). This module allows for easier interpretation of the multiplier inputs and outputs)
   - `SEG7_LUT.v` – (Simple 4-bit binary to 7-segment display conversion.)
   - All the design files you used to implement your 8-bit multiplier. Note: do not include any testbench files in the Quartus Project.

3. In the correct place in the top-level module, instantiate your multiplier. A dummy instantiation is already in place. Make sure you change the module name in the instantiation to match that of your multiplier.

4. Compile the design and program the DE10-Lite board.

Figure 3 shows the architecture of the provided wrapper code.

![Figure 3: Top-level Architecture for the multiplier circuit.](image)

Here’s how the top-level module works. Register b samples the switches as long as KEY[0] is not pressed. If KEY[0] is pressed, the value of the switches will be stored in register a. This is how you will apply input operands to your multiplier. The product register p_r samples the product of the multiplier every clock cycle. The input operands and output product are fed into BCD converters, which are in turn converted into 7-segment display-compatible signals. The switch SW[9] selects between displaying the input operands (in decimal) or the output product (in decimal). Get familiar with the design and verify that your multiplier works correctly on the board. **Demonstrate the working circuit to your TA.**

**D) Timing Analysis of the Multiplier Circuit**

As part of the compilation process, Quartus performs a timing analysis on the post place-and-routed design. You can view the results of the timing analysis by going to Quartus’ Compilation Report and expanding the menu under “TimeQuest Timing Analyzer”. Quartus provides several timing reports at different operating conditions. For this lab, we’re interested in the most pessimistic (slowest) model which is the “Slow 1200mV 85C Model”. Expand the menu under this heading and click on the “Fmax Summary” item. The reported clock frequency is the fastest that the current design can be clocked without violating the setup-time of any flip flops.

Now, this number does not quite represent the absolute fastest the circuit can be clocked. To understand this, see the explanation in Section 4 of:

[http://www.ece.ucdavis.edu/~bbaas/180b/lab/downloading.to.board.html](http://www.ece.ucdavis.edu/~bbaas/180b/lab/downloading.to.board.html)

**Task:** Increase the constrained clock frequency (i.e. the “50 MHz”) in your design's SDC file until Quartus cannot meet timing constraints. You’ll know when this happens because the folder “Slow 1200mV 85C
Model” in the Compilation Report will be highlighted in red. Record the highest operating frequency for your design.

Next, replace the code in the top level module that instantiates your multiplier with the expression:

\[
\text{assign } p = a\times b;
\]

Recompile your design and note what you observe about the maximum operating frequency.

Part 2 – VGA Display

For this part of the lab, you will be using the VGA port on the DE10-Lite boards. The zip file you downloaded earlier includes a prepared Quartus Prime project that generates the necessary control signals to drive a VGA display. Open the project in Quartus, compile it, and program your DE10-Lite board. The monitors in the computer lab should have an extra VGA cable; connect the free end of the cable to the VGA connector on your board.

Using SW[2:0], you should be able to change the color of the whole screen. Have a look through the code in lab3_vga.v and vga_controller.v.

The VGA controller is currently configured to drive a screen that is 640 pixels wide and 480 pixels tall (quite low resolution by today’s standards). In the top level module lab3_vga.v, you will see two signals col and row.

- **col** – is the column of the current pixel being drawn. This will be between 0 and 639 inclusive.
- **row** – is the row of the current pixel being drawn. This will be between 0 and 479 inclusive.

**Task:** Right now, the module colors the entire 640x480 screen. You will modify the code in the marked section of lab3_vga.v to select which quadrant of the screen is colored using SW[8] and SW[9]. All pixels outside the specified region should be black. The color of the region should be selected using SW[2:0] and match the color palette of the original design. The table below indicates the pixel boundaries for the colored regions of the screen:

<table>
<thead>
<tr>
<th>SW[9:8]</th>
<th>Lower Column</th>
<th>Upper Column</th>
<th>Lower Row</th>
<th>Upper Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>319</td>
<td>0</td>
<td>239</td>
</tr>
<tr>
<td>01</td>
<td>320</td>
<td>639</td>
<td>0</td>
<td>239</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>319</td>
<td>240</td>
<td>479</td>
</tr>
<tr>
<td>11</td>
<td>320</td>
<td>639</td>
<td>240</td>
<td>479</td>
</tr>
</tbody>
</table>

*Table 1: For Part 2, only the pixels in the rectangular regions above should be colored. All other pixels should be black.*

**Hint:** Use the “>” and “<” (or “<=” and “>=”) operators in Verilog and the signals col and row to determine when to draw color to the screen.
Submitted Work

[20 pts] Prelab – Solutions to prelab exercises.

[50 pts] Lab Checkoffs

Part 1

A) [10 pts] Demonstrate the testbench for your 8-bit RCA and show that your design successfully simulates.

B) [10 pts] Demonstrate the testbench for your 8-bit multiplier. Show that your design successfully simulates.

C) [15 pts] Demonstrate the working multiplier on the DE10-Lite Board.

Part 2

A) [15 pts] Demonstrate your module successfully coloring the selected quadrant of the screen.

[30 pts] Lab Report – Include of your Verilog source and testbench code for Part 1. For Part 2, turn in all files that you modified. In your report, also include the answers to the following questions.

1. [7 pts] What is the maximum clock frequency for the combinational multiplier you designed in Part 1C? You should determine this by slowly increasing the frequency of the clock MAX10_CLK1_50 in the SDC file for your design until Quartus cannot meet timing constraints. Report your answer to the nearest 5 MHz.

2. [7 pts] When you replace your module instantiation with a single assign statement using the ‘*’ operator, what is the new maximum operating frequency? You may have to repeat the process of constraining the clock in the SDC file again.

3. [8 pts] Why are the two clock frequencies above so different?

4. [8 pts] In Part 2, you were asked to use the “<” and “>” operators. At some point, these operations have to get implemented in digital logic. What logic would you use to implement a “<” operation between two unsigned binary integers?