I. Prelab

Complete the following and submit your work at the beginning of your lab session.

1. [15 pts]  Complete a preliminary design of your add3.v, seg7, test1, and test2 code (a draft of your verilog code)

II. A Combinational 3-input Adder

Design and implement the following modules in verilog:

1. Write a module fa.v for a full-adder in Verilog using either wires or regs. A full-adder has inputs $a$, $b$, and $ci$ (carry-in) and produces outputs $s$ (sum) and $co$ (carry-out). Its Boolean expressions are as follows:
   
   $s = a \oplus b \oplus ci$

   $co = ab + aci + bci$

2. Create a module add2.v in Verilog that implements an 8-bit ripple-carry adder by instantiating eight full-adder circuits. The adder is built with a ripple-carry adder design (consult a reference if you are rusty with the details). An example block diagram for a 4-bit ripple-carry adder is shown:

   Each ripple-carry adder is built with eight Full Adders (FA). The 8-bit adder has 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/Output</td>
<td>Input operand</td>
</tr>
<tr>
<td>$b$</td>
<td>8</td>
<td>Input/Output</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. Create a combinational adder module add3.v that adds three 6-bit inputs into one 8-bit output sum. add3.v works by first adding two inputs and then adding this sum to the third input, using two instances of add2.v. The two internal $ci$ signals are tied low, and the two internal $co$ signals are left unattached so there are only four ports as listed:
<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Number of Bits</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in1</td>
<td>6</td>
<td>Input</td>
<td>Input operand</td>
</tr>
<tr>
<td>in2</td>
<td>6</td>
<td>Input</td>
<td>Input operand</td>
</tr>
<tr>
<td>in3</td>
<td>6</td>
<td>Input</td>
<td>Input operand</td>
</tr>
<tr>
<td>sum</td>
<td>8</td>
<td>Output</td>
<td>Output Sum</td>
</tr>
</tbody>
</table>

The MSB and MSB–1 of each input into the internal add2.v adders are tied low inside add3.v; in other words, three of the four 8-bit add2.v inputs would appear as `{2'b00, in1}` where in1 is 6 bits wide. The other two inputs are handled in the same way. The bus between the two add2 adders is a full 8 bits. add3.v's inputs are 6 bits wide and can range from only 0 to +63. The output is 8 bits wide and can range from 0 to +255 and therefore the adder can never overflow.

### III. A Hexadecimal display driver

Design and implement a combinational 7-segment display driver module seg7.v in verilog that inputs a 4-bit signal and drives 7 outputs that correspond to 0–9, A–F with a low output turning on an LED segment and a high output turning the corresponding segment off. The letters A–F may be upper or lower case. Two of the outputs must be wires and five as regs. As much as possible, use different coding styles for each of the seven outputs. Since an output port may be only a wire or reg, the module's output must be a concatenation of the two wires and five regs.

![7-segment display](image)

### IV. Testbench Design and Simulation

Design and write a testbench called test1 with the following features:

1. It instantiates one copy of add3.v
2. add3.v's input ci is tied to 1'b0
3. It calculates the sum of the three inputs connected to add3.v using two (or three for ForceError) “+” operators in verilog
4. It automatically compares the output sum of add3.v with the verilog sum using “+”
5. It tests add3 over all possible input combinations
6. It prints an error message and halts the simulation if there is a mismatch of the two sums
7. A single-bit reg called ForceError causes a sum error in the verilog sum. This is used to verify the comparison code is working correctly.

[20 pts] Once your design is working correctly in simulation, demonstrate it exercising your module correctly, and forcing an error, to your TA and have it checked off.
V. Implementation and Verification on the DE10-Lite

Design and write a test module called test2 with the following features:

1. It instantiates one copy of add3.v
2. add3.v’s input $ci$ is tied to $1'b0$
3. Three of the DE10-Lite’s switches [SW2, SW1, SW0] select one of eight tests where each test causes a unique set of inputs to be driven into the three inputs of the adder.
4. Test $\{0,0,0\}$ causes the three inputs to be $6'b000000$, $6'b000000$, $6'b000000$.
5. Test $\{0,0,1\}$ causes the three inputs to be $6'b000001$, $6'b000001$, $6'b000001$.
6. Test $\{0,1,0\}$ causes the three inputs to be $6'b000010$, $6'b000010$, $6'b000010$.
7. Test $\{0,1,1\}$ causes the three inputs to be $6'b111111$, $6'b111111$, $6'b111111$.
8. Choose interesting test inputs for the other four test cases.
9. Two of the DE10-Lite’s 7-segment displays are driven with two instances of seg7.v showing the 8-bit sum in hexadecimal format.

Use SystemBuilder to create a new Quartus project. Download your design onto the DE10-Lite board and verify it works correctly.

[50 points] Demonstrate it compiling, downloading, and operating to your TA. Your TA will record a certutil hash of your top-level files and this must match the hash of the files you upload to canvas so no file modifications are possible after your demo.

VI. Submitted Work [100 pts total]

[15 pts] Prelab

[20 pts] Lab Checkoff: Simulation

[50 pts] Lab Checkoff: FPGA board

[15 pts] Lab Report

Submit all Verilog hardware and testbench code that you wrote. Do not include any code that you did not write such as files generated by Quartus or IP components.

a) Upload a copy to Canvas by performing the following steps by the end of your lab session—this is essential to receive any credit for the entire lab.

1. Make a folder on your computer
2. Copy all verilog files you wrote into the folder—only the ones you wrote
3. “zip” the folder into a single .zip file
4. Log onto Canvas, click Assignments, find the correct lab number
5. Upload the .zip file

2019/04/15 Posted
2019/04/26 Added minor clarification