University of California, Davis
Department of Electrical and Computer Engineering

EEC 180

Winter Quarter 2021

LAB 2: Combinational Logic Design Using Verilog

Objective: The purpose of this lab is to use Verilog to design combinational arithmetic circuits. You will also learn how to write self-checking testbenches.

I. Ripple-carry Adder

A *full adder* (FA) has inputs a, b and c_i (carry in) and produces outputs s (sum) and c_0 (carry out).

$$c_0 = a.b + a.\,c_i + b.c_i$$
$$s = a \oplus b \oplus c_i$$

An n-bit ripple-carry adder can be designed by connecting full-adders in a chain with the carry in for a given stage being the carry out of the previous stage and the carry in of the least significant bit being a 0.

Perform the following steps:

1. Write a **behavioral** model for a full adder in Verilog. (Hint: an assign statement can describe each output equation.)
2. Instantiate your full adder subcircuit in order to build an *8-bit ripple-carry adder.* Add logic to produce an *Overflow* output, which should be set to 1 whenever the sum produced by the adder does not provide the correct signed (twos complement) value.
3. Write a testbench to simulate your design for all possible input combinations. Note that for an 8-bit adder there are 2^{16} (256 x 256) possible inputs. See Appendix at the end of this document for an example of how to construct a testbench using a **for loop** in Verilog.

II. Multiplication

Figure 2 shows the traditional procedure for performing the multiplication P=A x B, where A and B are 4-bit unsigned binary numbers. Since each bit in B is either 1 or 0, the summands are either shifted versions of A or 0000. The Boolean AND operation can be used to multiply any two binary bits. Figure 3 shows an array multiplier circuit that implements P = A x B, where A and B are 4-bit unsigned binary numbers.

$$
\begin{array}{cccccccc}
 & & & \times & a_3 & a_2 & a_1 & a_0 \\
 & & & & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\
 & & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\
 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

Figure 2. Multiplication of Unsigned Binary Numbers

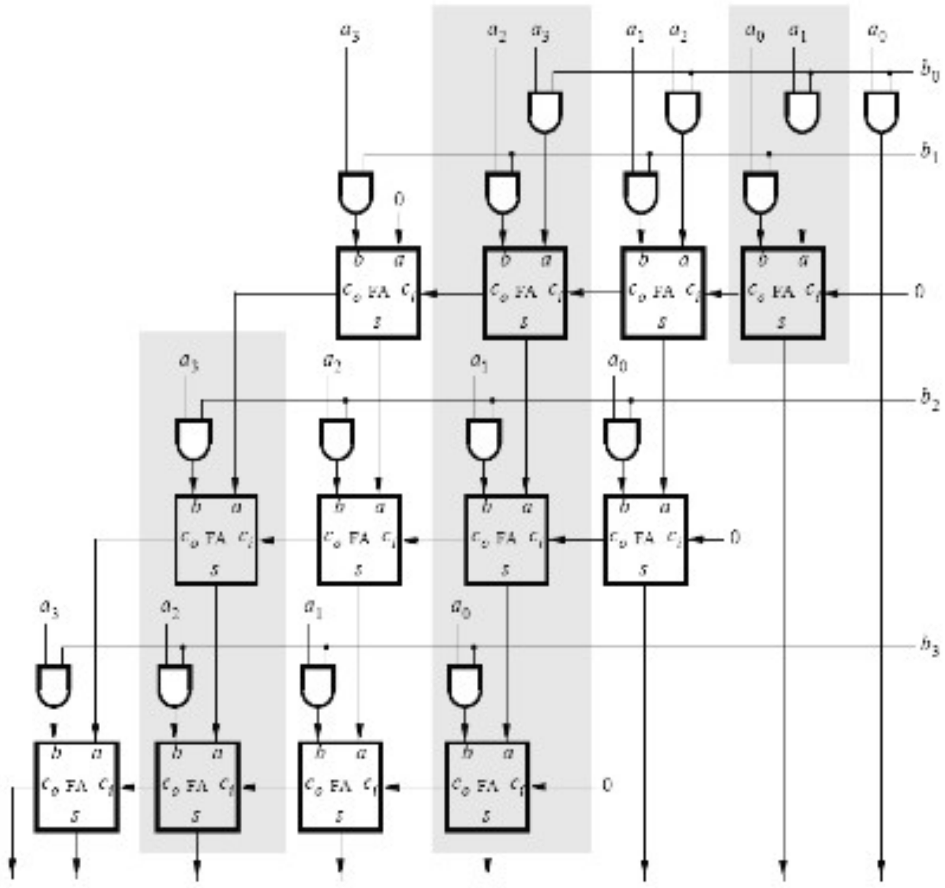


Figure 3. Array Multiplier Circuit Block Diagram

Perform the following steps:

1. Write a **structural** model in Verilog that describes an 4x4 unsigned array multiplier that can be implemented on the Altera DE10-Lite board. Use switches

SW$_{7\text{-}4}$ to represent the number A and switches SW$_{3\text{-}0}$ to represent the number B. Display the hex value of A on HEX3 and the hex value of B on HEX2. Display the result P = A x B on HEX1-0.

2. Compile your design in Quartus. Download your design to the Altera DE10-Lite board and test your circuit. **<u>Demonstrate your circuit to your TA</u>.**

3. Estimate the performance of your circuit in terms of the critical path using the timing analysis tool. The following is a step by step procedure to obtain the propagation delay between the input and output ports of your design:

   Step 1. Timing netlist
   In this section, we will create a timing netlist, specify the delay model and operating conditions for the multiplier design.

   - Go to Netlist section on top and click on **Create Timing Netlist.** The corresponding Timing Netlist window will pop-up.

   - Select Fast-corner delay model and click OK.

   - Next, go to Netlist -> Set Operating Conditions, select MIN-Fast-1200mV-0c operating condition and click OK.

   Step 2. Constraints
   In this section, we will set the maximum propagation delay constraints for the design.

   - Go to Constraints -> Set Maximum Delay and Set Maximum Delay window will pop-up.

   - Click the name finder icon and that will open up the Name Finder window.

   - Select *get_ports* in collection and click *List* to list out all I/O ports.

   - First, select all input ports, SW0, SW1..SW7 and transfer them to the second column using '>', which locks all the selected ports. Next, click ok to complete the *From* section in *Set Maximum Delay* window.

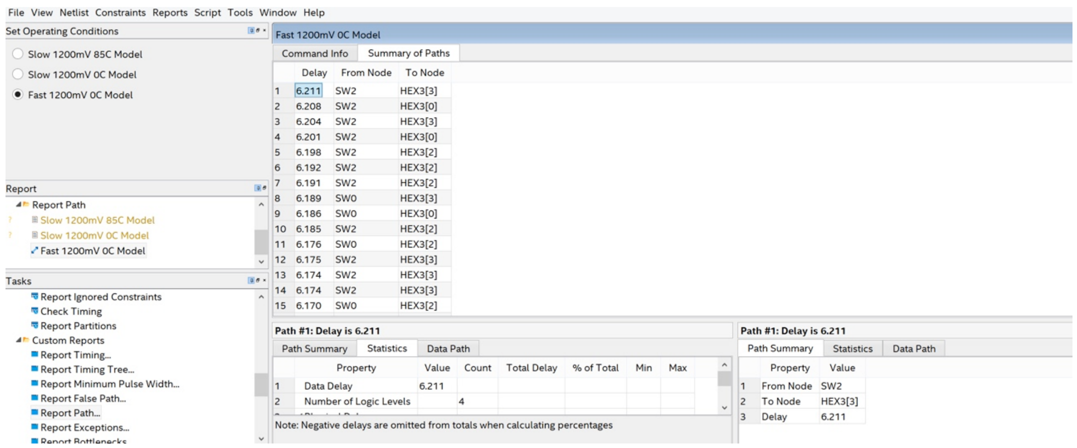
   - Similarly, select all output ports HEX0[0:6] .. HEX3[0:6] and complete the *To* section in *Set Maximum Delay* window.

   - Next, set a ***Delay value of (say 30ns)*** and click *Run.*

- Finally, click on *Update Timing Netlist* in Tasks section to consider the timing constraint and re-generate timing reports.

Step 3. Reports
In this section, we will generate and examine the timing reports.

- Go to *Tasks -> Reports -> Custom reports -> Report Path*
- In Report Path window, fill up the I/O ports in *From* and *To* section like the previous section.
- Set Report number of paths as 50 and click on *Report Path*.
- Next, you can see the delay across each path and the critical path delay of the design being the top one in the list (e.g. 6.211ns for an example design as shown below with critical path being SW2 -> HEX3[3])



	Delay	From Node	To Node
1	6.211	SW2	HEX3[3]
2	6.208	SW2	HEX3[0]
3	6.204	SW2	HEX3[3]
4	6.201	SW2	HEX3[0]
5	6.198	SW2	HEX3[2]
6	6.192	SW2	HEX3[2]
7	6.191	SW2	HEX3[2]
8	6.189	SW0	HEX3[3]
9	6.186	SW0	HEX3[0]
10	6.185	SW2	HEX3[2]
11	6.176	SW0	HEX3[2]
12	6.175	SW2	HEX3[3]
13	6.174	SW2	HEX3[3]
14	6.174	SW2	HEX3[3]
15	6.170	SW0	HEX3[2]

Path #1: Delay is 6.211

	Property	Value	Count	Total Delay	% of Total	Min	Max
1	Data Delay	6.211					
2	Number of Logic Levels		4				

Note: Negative delays are omitted from totals when calculating percentages

Path #1: Delay is 6.211

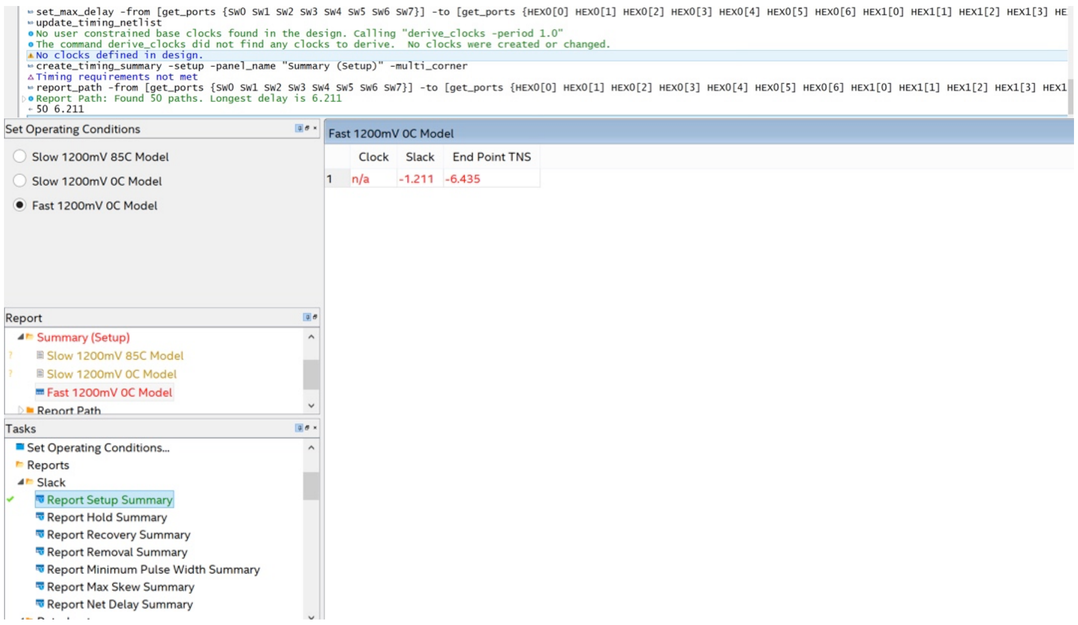
	Property	Value
1	From Node	SW2
2	To Node	HEX3[3]
3	Delay	6.211

- Next, go to Tasks -> Reports -> Slack -> Report Set up Summary to see a **Positive Slack**, which indicates the design has met the timing constraint (Propagation delay) of 30ns.
- Finally, we can do an experiment to find whether the design considered for an example can meet a **propagation delay timing constraint of 5ns**. To do so, we can proceed with the steps listed earlier. However, to do things faster we can run some of the **tcl** commands on console as shown below.
  - *set_max_delay -from [get_ports {SW0 SW1 SW2 SW3 SW4 SW5 SW6 SW7}] -to [get_ports {HEX0[0] HEX0[1] HEX0[2] HEX0[3] HEX0[4] HEX0[5] HEX0[6] HEX1[0] HEX1[1] HEX1[2] HEX1[3] HEX1[4] HEX1[5] HEX1[6] HEX2[0] HEX2[1] HEX2[2] HEX2[3] HEX2[4] HEX2[5] HEX2[6] HEX3[0] HEX3[1] HEX3[2] HEX3[3] HEX3[4] HEX3[5] HEX3[6]}] 5*
  - *update_timing_netlist*

- *create_timing_summary -setup -panel_name "Summary (Setup)" -multi_corner*
- *report_path -from [get_ports {SW0 SW1 SW2 SW3 SW4 SW5 SW6 SW7}] -to [get_ports {HEX0[0] HEX0[1] HEX0[2] HEX0[3] HEX0[4] HEX0[5] HEX0[6] HEX1[0] HEX1[1] HEX1[2] HEX1[3] HEX1[4] HEX1[5] HEX1[6] HEX2[0] HEX2[1] HEX2[2] HEX2[3] HEX2[4] HEX2[5] HEX2[6] HEX3[0] HEX3[1] HEX3[2] HEX3[3] HEX3[4] HEX3[5] HEX3[6]}] -npaths 50 -panel_name {Report Path} -multi_corner*

- Running those commands generates the Path Reports and directly show the propagation delay of 6.211ns on the console with a message ***Timing requirement not met*** as shown below.

- The Set up summary notifies **a negative slack of -1.22ns** as shown below, which means that the design has not met the timing constraint (Propagation delay of 5ns) and slack of -1.22ns is basically the difference of constrained propagation delay and actual propagation delay in the design.



```
set_max_delay -from [get_ports {SW0 SW1 SW2 SW3 SW4 SW5 SW6 SW7}] -to [get_ports {HEX0[0] HEX0[1] HEX0[2] HEX0[3] HEX0[4] HEX0[5] HEX0[6] HEX1[0] HEX1[1] HEX1[2] HEX1[3] HE
update_timing_netlist
No user constrained base clocks found in the design. Calling "derive_clocks -period 1.0"
The command derive_clocks did not find any clocks to derive.  No clocks were created or changed.
No clocks defined in design.
create_timing_summary -setup -panel_name "Summary (Setup)" -multi_corner
Timing requirements not met
report_path -from [get_ports {SW0 SW1 SW2 SW3 SW4 SW5 SW6 SW7}] -to [get_ports {HEX0[0] HEX0[1] HEX0[2] HEX0[3] HEX0[4] HEX0[5] HEX0[6] HEX1[0] HEX1[1] HEX1[2] HEX1[3] HEX1
Report Path: Found 50 paths. Longest delay is 6.211
50 6.211
```

Set Operating Conditions

- ○ Slow 1200mV 85C Model
- ○ Slow 1200mV 0C Model
- ● Fast 1200mV 0C Model

Fast 1200mV 0C Model

	Clock	Slack	End Point TNS
1	n/a	-1.211	-6.435

Report the critical path and propagation delay for the 4x4 multiplier.

4. Design and write a Verilog model for 8x8 unsigned array multiplier.

5. Perform a functional simulation of your 8x8 array multiplier design and print the simulation results.

III. **Lab Report**

For your lab report, include the following:

- Lab Cover Sheet with signed TA verification for successful simulations and implementation in the Altera board.
- Design and implementation of the overflow circuit in part I of the lab.
- Complete Verilog test bench for each part. The testbench should test your design exhaustively, i.e., for all possible input combinations and should be self-checking as shown in the appendix.

Grading Guidelines

Part 1 – 50 points
Part 2 – 50 points

IV. Appendix: Self Checking Testbench

A half adder has 2 inputs, so for exhaustive verification we have to check the output for all the four input combinations, which can be done by the ***for loop****.*

The testbench is self-checking – it examines the outputs and prints out an error message if the output is wrong.

```
module halfAdd (sum, cOut, a, b);
        output  sum, cOut;
        input   a, b;

        xor             (sum, a, b);
        and             (cOut, a, b);
endmodule

module tBench;
        wire    sum, co;
        reg [2:0] test;

        // design under test;
 halfAdd        HA (sum, co, test[1], test[0]);
        // stimulus and verification that the output is correct
initial begin
        for (test = 0; test < 4; test = test + 1)
        begin
          #10;
            if ({co, sum} !=   (test[1] + test[0]) )
```

```
                $display("ERROR:  a=%b b=%b sum=%b cout=%b", test[1], test[0], sum,
co);
              end
            #10 $finish;
           end
endmodule
```
