University of California, Davis
Department of Electrical and Computer Engineering

EEC180 **DIGITAL SYSTEMS II** **Winter Quarter 2024**

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.

Prelab - Behavioral vs. Structural RTL (15pts)

1. What is the difference between behavioral and structural HDL code?

2. Provide one small example (10 lines of code maximum) of a behavioral RTL module, and one small example of a structural RTL module.

Part 1 - Ripple-carry Adder (30 pts)

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.

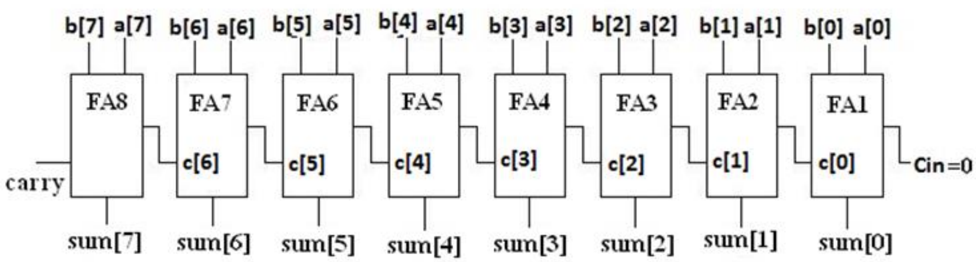

Figure 1 - 8 bit Ripple Carry Adder

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 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. **Demonstrate your testbench to your TA.**

<u>Part 2 - Multiplication (30 pts)</u>

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.

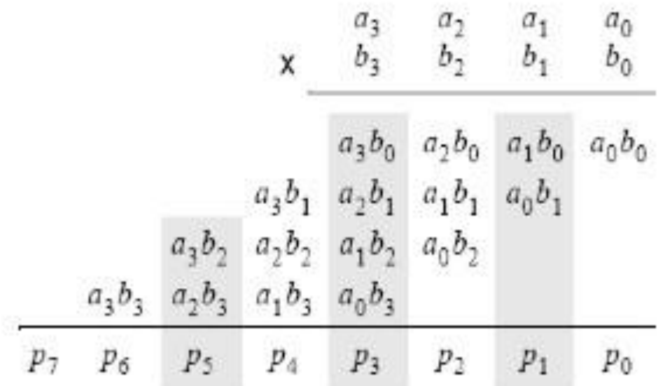

Figure 2 - Multiplication of Unsigned Binary Numbers

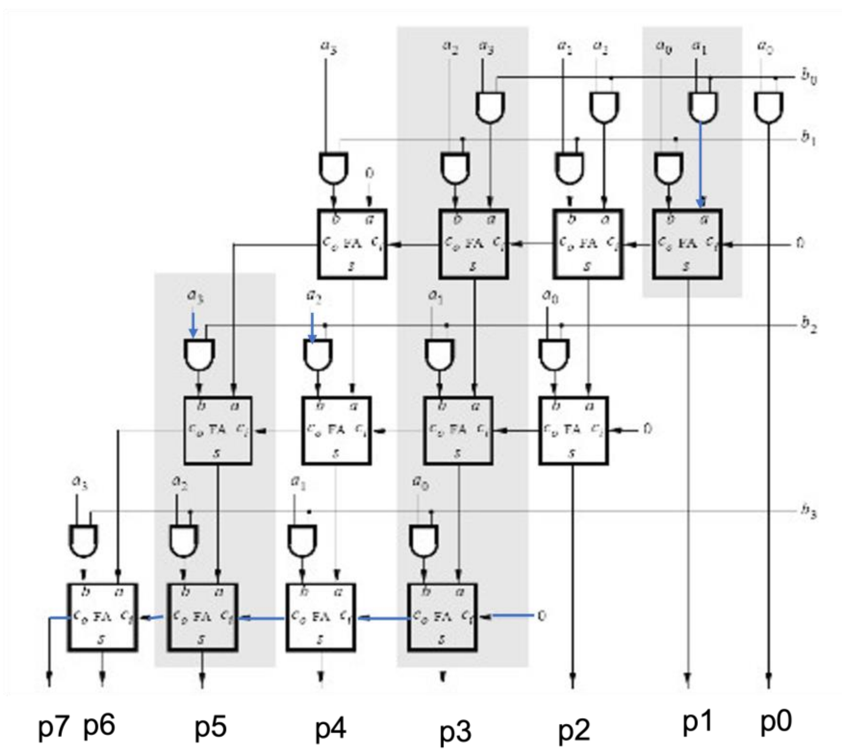

Figure 3 - Array Multiplier Circuit Block Diagram

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 hex value of the result P = A x B on HEX1-0.

2. Modify the testbench for the adder to test the multiplier design. **Demonstrate your testbench to your TA.**

3. Compile your design in Quartus. Download your design to the Terasic DE10-Lite board and test your circuit. **Demonstrate your circuit to your TA.**

<u>Part 3 - Generic Adder (25 pts)</u>

Verilog *parameters* are a powerful way to create flexible HDL code. You should include parameters in your code wherever possible to allow reuse of modules by simply modifying parameters at instantiation. Helpful reference: https://www.chipverify.com/verilog/verilog-parameters.

Write a *generic* adder using Verilog parameters and generate statement in Verilog. Your adder should take a parameter K. See https://www.chipverify.com/verilog/verilog-generate-block for examples of how to use parameters and generate statements.

Update your testbench as well so that it works for any value of K. **Demonstrate your testbench to your TA.**

The TA will test your design for any value of K, such as K=7 or 11.

You don't have to download the design to the hardware for this part of the lab. You must show that the design works correctly via functional simulation and testbench.

<u>Submission (100 pts total)</u>

Submit the following items for verification and grading:

- Report containing
    - Page 1: Lab cover sheet with TA verifications for
        1) Prelab Q1 & Q2
        2) Part I – HW demonstration & testbench
        3) Part II – HW demonstration & testbench
        4) Part III – testbench
    - Prelab solution
    - Outputs and/or waveforms from self-checking testbenches for Pt I-III.
- Complete Verilog source code for all lab exercises including
    - partI.v, tb_partI.v
    - partII.v, tb_partII.v
    - partIII.v, tb_partIII.v
    - Any additional design files required to compile your ModelSim/Quartus projects.

Appendix - Self Checking Testbench

*A half adder has 2 inputs, so for exhaustive verification we must 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
```