Design Assignment 2 – Combinational Logic

1 PROCEDURE

1) Team sizes of 1 or 2 students, 1 report per team.
2) Review the instructions in the PowerPoint deck Introduction to Xilinx ISE (LP).pptx, uploaded to Canvas under the folder Design Assignments.
3) Create a new ISE Project, called eng312_proj2
4) Create and verify Verilog modules:
    a. full_adder
        i. Inputs: A, B, CARRY_IN
        ii. Outputs: SUM, CARRY_OUT
        iii. Functionality
            1. SUM, CARRY_OUT represent full adder
        iv. Verification
            1. Create testbench that steps through all 8 combinations of A, B, CARRY_IN. Verify that output waveforms match desired behavior.
    b. four_bit_adder
        i. Inputs: A[3:0], B[3:0]
        ii. Outputs: SUM[4:0]
            1. Functionality build a module to compute SUM=A+B based on four full_adder blocks. Write your Unit Under Test by computing:
                a. SUM0, CARRY0, as a function of A[0] and B[0]
                b. SUM1, CARRY1, as a function of A[1], B[1] and CARRY0
                c. SUM2, CARRY2, as a function of A[2], B[2] and CARRY1
                d. SUM3, CARRY3, as a function of A[3], B[3] and CARRY2
                e. SUM4 as CARRY3
        iii. Verification
            1. Create a testbench that steps through all 256 combinations of A and B. Verify that the output waveforms match desired behavior.
    c. three_bit_comparator
        i. Inputs: A[2:0], B[2:0]
        ii. Outputs: GT, LT, EQ
        iii. Functionality (write using Boolean equations)
            1. GT = ‘1’ if A>B, otherwise ‘0’
            2. LT = ‘1’ if B>1, otherwise ‘0’
            3. EQ = ‘1’ if A==B, otherwise ‘0’
        iv. Verification
            1. Create a testbench that steps through all 64 combinations of A and B. Verify that the output matches the desired behavior.
    d. dec_4_to_16
        i. Inputs: ADDR[3:0]
        ii. Outputs: DEC[15:0]

iii. Functionality
1. For each combination of the ADDR bits drive the addressed bit of DEC to '1', all other output bits of DEC should be '0'.

iv. Verification
1. Create testbench that steps through all 16 combinations of the four bits of ADDR. Verify that output waveforms match desired behavior.

e. priority_encoder
   i. Inputs: D[3:0]
   ii. Outputs: ENC[1:0], VLD
   iii. Functionality
      1. If any of the bits of D are '1', then ENC should reflect the index of the most significant '1' bit.
      2. VLD = '1' if any bits of D are '1', otherwise '0'
   iv. Verification
      1. Create testbench that steps through all 16 combinations of the four bits of D. Verify that output waveforms match desired behavior.

f. mux_four_to_one
   i. Inputs: DIN[3:0], SEL[1:0]
   ii. Outputs: DOUT
   iii. Functionality:
      1. DOUT=DIN[SEL]
   iv. Verification
      1. Create testbench that steps through all 64 combinations of the four bits of D and the two bits of SEL. Verify that output waveforms match desired behavior.

5) Report: organize your report by module. For each module include:
   a. Module source code, nicely formatted and commented, in a monospaced (e.g. Courier New) font.
   b. Model testbench code, similarly formatted and commented.
   c. Waveform snapshot
   d. Brief text description of the expected behavior, and how you verified the resulting waveform.