Lab 1 - FPGA Design Basics

Name: Jason Xie
NSID: xix277
Student Number: 11255431

1. ***Explain the synthesis difference between using case statements and if/else statements.***

   Using case statements for synthesis will create one N:1 multiplexer whereas if/else statements use 2:1 multiplexers chained/cascaded together.

2. ***Describe in your own words how your design can be optimized for performance, and how it can be optimized for (minimum) size/area.***

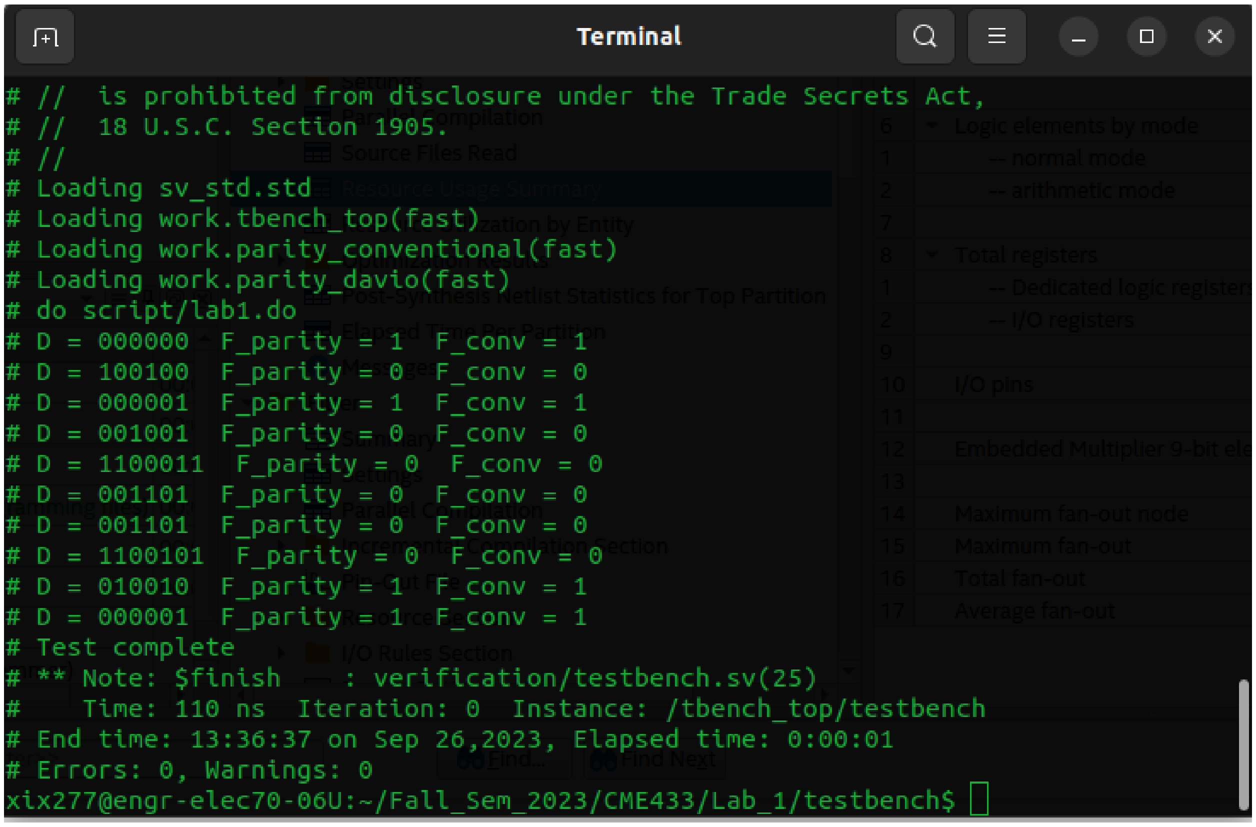
   In terms of optimizing performance, my design can be improved using parallelism, pipelining, and prediction. Parallelism to allow for different parts of a task to be run at the same time. Pipelining to allow for different finished phases of a task to be moved onto the next phase while also starting another phase of a new task instead of waiting for each stage to be completed. Prediction to allow for guessing of what may come next and to start the task as long as the recovery is not too costly.

   In terms of optimizing size/area, I would implement logic optimization to reduce the number of logic gates used. This means getting rid of redundant logic to minimize area and allow for the sharing of certain operations.

3. ***The VHDL and Verilog code for parity_conventional will be provided along with a template for parity_davio. Based on the conventional implementation, complete the HDL file for the Davio expansion (only required to do one of VHDL or Verilog). Refer to the Davio expansion theorem in your class ppt file.***

   See Appendix for Davio Expansion Code.

4. ***Verify that the Davio expansion code is functionally equivalent to the conventional method. (Use Modelsim/Questasim to prove correctness). Include screenshots that show your verification. Hint: make sure you actually show/prove that your design works.***



```
# //  is prohibited from disclosure under the Trade Secrets Act,
# //  18 U.S.C. Section 1905.
# //
# Loading sv_std.std
# Loading work.tbench_top(fast)
# Loading work.parity_conventional(fast)
# Loading work.parity_davio(fast)
# do script/lab1.do
# D = 000000  F_parity = 1  F_conv = 1
# D = 100100  F_parity = 0  F_conv = 0
# D = 000001  F_parity = 1  F_conv = 1
# D = 001001  F_parity = 0  F_conv = 0
# D = 1100011  F_parity = 0  F_conv = 0
# D = 001101  F_parity = 0  F_conv = 0
# D = 001101  F_parity = 0  F_conv = 0
# D = 1100101  F_parity = 0  F_conv = 0
# D = 010010  F_parity = 1  F_conv = 1
# D = 000001  F_parity = 1  F_conv = 1
# Test complete
# ** Note: $finish    : verification/testbench.sv(25)
#    Time: 110 ns  Iteration: 0  Instance: /tbench_top/testbench
# End time: 13:36:37 on Sep 26,2023, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
xix277@engr-elec70-06U:~/Fall_Sem_2023/CME433/Lab_1/testbench$ 
```

From the testbench, random inputs were generated and input to the Davio and conventional circuit and then compared to ensure the outputs were correct. See appendix for testbench code.

5. ***Try changing the optimization strategy (Area/Power/Performance/Balanced) and explain your results (Assignments→Settings→Compiler Settings). What does the synthesis tool adjust to get the best 'area' or 'performance' or 'power'? How does the Davio expansion compare to the conventional method in terms of synthesis (explain your result)?***

After changing the compiler settings for Area/Power/Performance/Balanced, there seemed to be no changes in the number of LE's that were used. This could be because Quartus already pre-optimizes it somehow in the background or just that the way the code is and how simple the circuit is that no optimization needs to be made.

However, comparing the conventional parity to the Davio parity, the Davio parity used 7 LE's whereas the conventional used 8 LE's. Utilizing technology mapping allowed for one less LE to be used for the Davio circuit allowing for cost savings.

Appendix

Davio Expansion Code:

```
module parity_davio( D, F );

input [6:0] D;
output F;

wire f6_0;
wire f6_2;
wire f5_0;
wire f5_2;

assign f6_0 = (~D[0] & D[1] & ~D[2] & D[3] & D[5]) ^
                        (D[2] & ~D[3] & D[5]) ^
                        (D[0] & ~D[1] & ~D[2] & ~D[3]) ^
                        (D[0] & D[1] & D[2] & ~D[3] & D[5]) ^
                        (D[0] & ~D[1] & ~D[3]) ^
                        (~D[0] & D[1] & D[3] & D[5]) ^
                        (~D[0] & D[2] & D[3] & D[5]) ^
                        (~D[0] & D[2]) ^
                        (~D[0] & D[1] & ~D[2]) ^
                        (~D[1] & ~D[2] & ~D[3]);

assign f6_2 = (D[0] & ~D[1] & ~D[2] & ~D[3]) ^
                        (~D[0] & D[1] & D[3] & D[5]) ^
                        (~D[0] & D[2] & D[3] & D[5]) ^
                        (~D[0] & D[1] & ~D[2]) ^
                        (~D[0] & D[1] & D[2] & ~D[3] & ~D[5]);

assign f5_0 = (D[0] & ~D[1] & ~D[2] & ~D[3] & ~D[6]) ^
                        (~D[0] & D[1] & D[2] & ~D[3] & D[6]) ^
                        (~D[1] & ~D[2] & ~D[3]);

assign f5_2 = (~D[0] & D[1] & D[2] & ~D[3] & D[6]);

assign F = ((f6_0 ^ (D[6] & f6_2)) ^ (D[4] & (f5_0 ^ (D[5] & f5_2))));

endmodule
```

Testbench Top:

```
module tbench_top();

//wires
wire [6:0] D; // Parity Input
wire F_davio, F_conv;



// instantiate testbench module
testbench testbench(
    .D(D), .F_conv(F_conv), .F_davio(F_davio)
     );

// instantiate dut module
parity_conventional dut_conv(
    .D(D), .F(F_conv)
);

parity_davio dut_davio(
    .D(D), .F(F_davio)
);



endmodule
```

Testbench:

```
module testbench(
    input F_davio, F_conv,
    output  reg [6:0] D

);

int counter, limit;

initial begin
    D = 7'b0;
    counter = 0;
    limit = 10;
    #10;

    while(counter < limit) begin
        #10;
        if (F_davio != F_conv)
            $display("Error! Parity Davio and Parity Conventional did not match, F_davio =
%b   F_conv =  %b", F_davio, F_conv);

        $display("D = %6b  F_parity = %b  F_conv = %b", D, F_davio, F_conv);
        D = $random();
        counter ++;
    end
    $display("Test complete");
    $finish;
end



endmodule
```