Cover Page
CPE 324-02: Digital Hardware Design Fundamentals

Lab 5

Simulation of Finite State Machine Designs

Submitted by: Nolan Anderson

Date of Experiment: 28 March 2021

Report Deadline: 28 March 2021

1. Introduction:

1.1 - What is to be studied, what is the purpose, and how is this purpose accomplished?

This lab is split into 3 separate parts, all implemented in the Modelsim program. We are to implement a (2.1) behavioral simulation of a Mealy FSM, a (2.2) Structural Implementation of a Mealy FSM, and (2.3) a Behavioral simulation of a Moore FSM. We are to implement these 3 different models for a two's complement serial adder. We do this by feeding the implementations with a testbench file that has predefined values passed into the FSM. I will first cover the theory behind these three implementations in section 2 and share my implementation in section 3. Section 4 will cover what I learned running the experiment, and section 5 will provide an overview of the outputs for each of the simulations.

2. Experiment Description

Theory, analysis, and purpose:

2.1 Two's complement adder

A two's complement adder uses a DFF in its design and two bit-wide inputs to produce a result S. This output S will be the result of the circuit. One thing to note, is that the output depends on the value the DFF produces. So there are technically 3 inputs to the Full Adder. Figure 2.1 shows a functional overview.

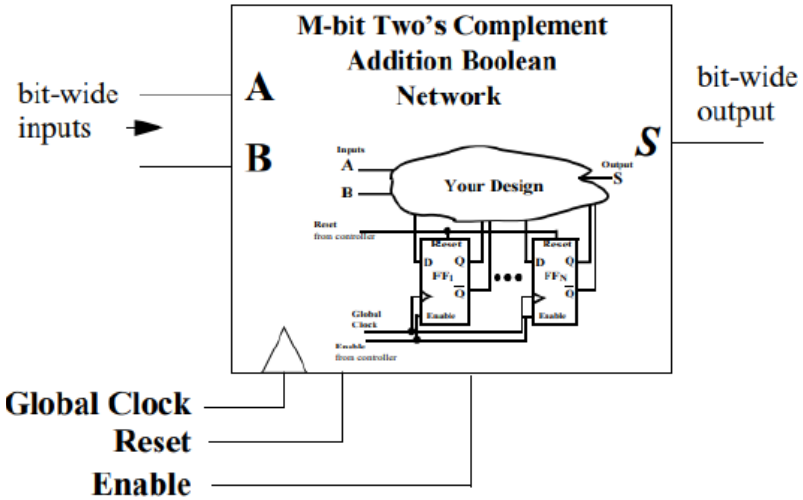
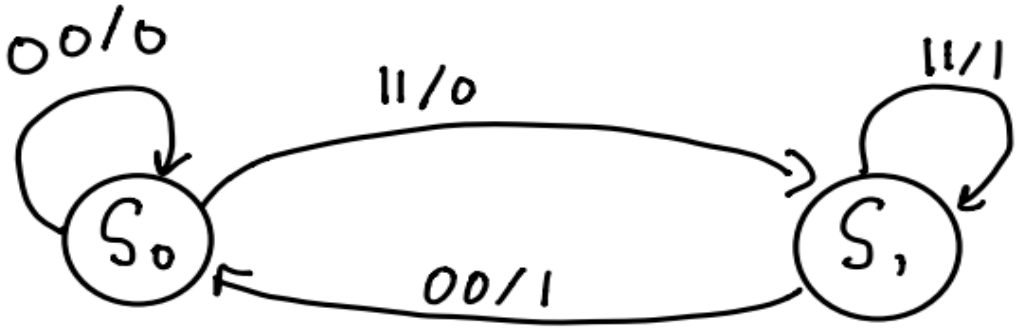


Figure 1: Two's complement adder network

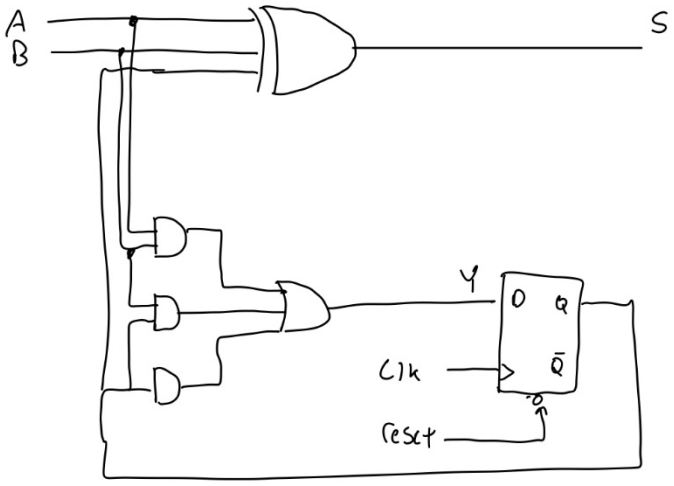
2.2 Mealy FSM Behavioral Simulation

The image below shows a state graph for the Mealy FSM that is implemented in Appendix 1. It is a Mealy Machine because its outputs depends on the current inputs.



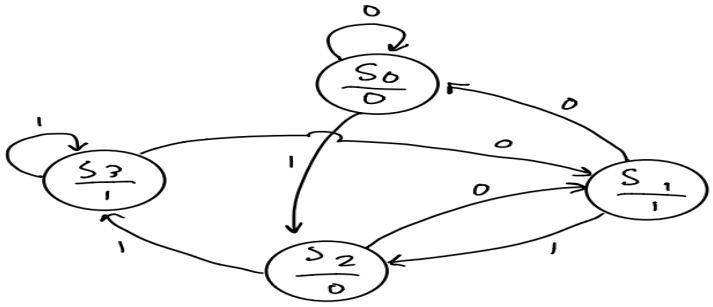
2.3 Mealy FSM Structural Simulation

Like section 2.2, this will have the same state graph. The only different here for the code is how it is implemented. The output produced will be identical and can be seen in section 5. The next figure shows the design for this circuit.



2.4 Moore FSM Behavioral Simulation

The image below shows a state graph for the Moore FSM that is implemented in Appendix 3. We can see that it is a Moore machine as its outputs only depends on the current state.



2.4 Structural vs Behavioral

As shown in the appendix, the main difference between the behavioral and structural implementations of Verilog code is structural uses module instances and gates / flip flops. Behavioral refers to using always @ blocks. This is important to note as we are implementing part 1 in two different ways, just one is with structural and the other is behavioral.

3. Demonstration

3.1 Implementation method:

3.1.1 Mealy Behavioral:

See appendix 1.

3.1.2 Mealy Structural:

See appendix 2.

3.1.3 Moore Behavioral:

See appendix 3.

3.2 Video Link:

Video uploaded to shared drive.

4. Experimental Results

4.1 Observations:

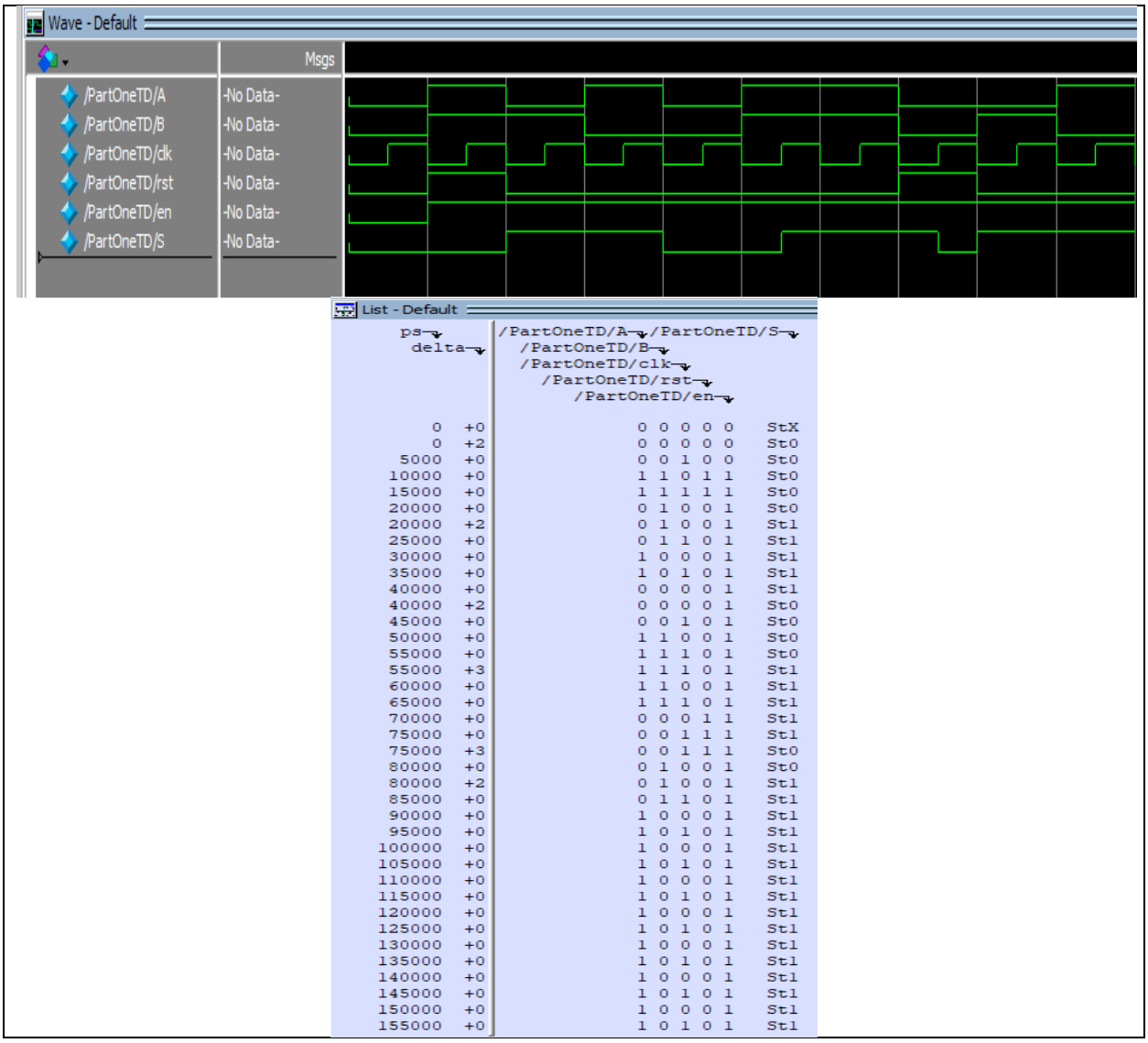
When observing the outputs from the different waveforms (see section 5) from part one and part 2, we can see that the outputs are the same. Even though the implementation method is completely different, the output is the same. This is the power of structural vs behavioral models. They do the same thing but are fundamentally different. Lastly, looking at the Moore model, we can see the results are as expected.

5. Conclusions

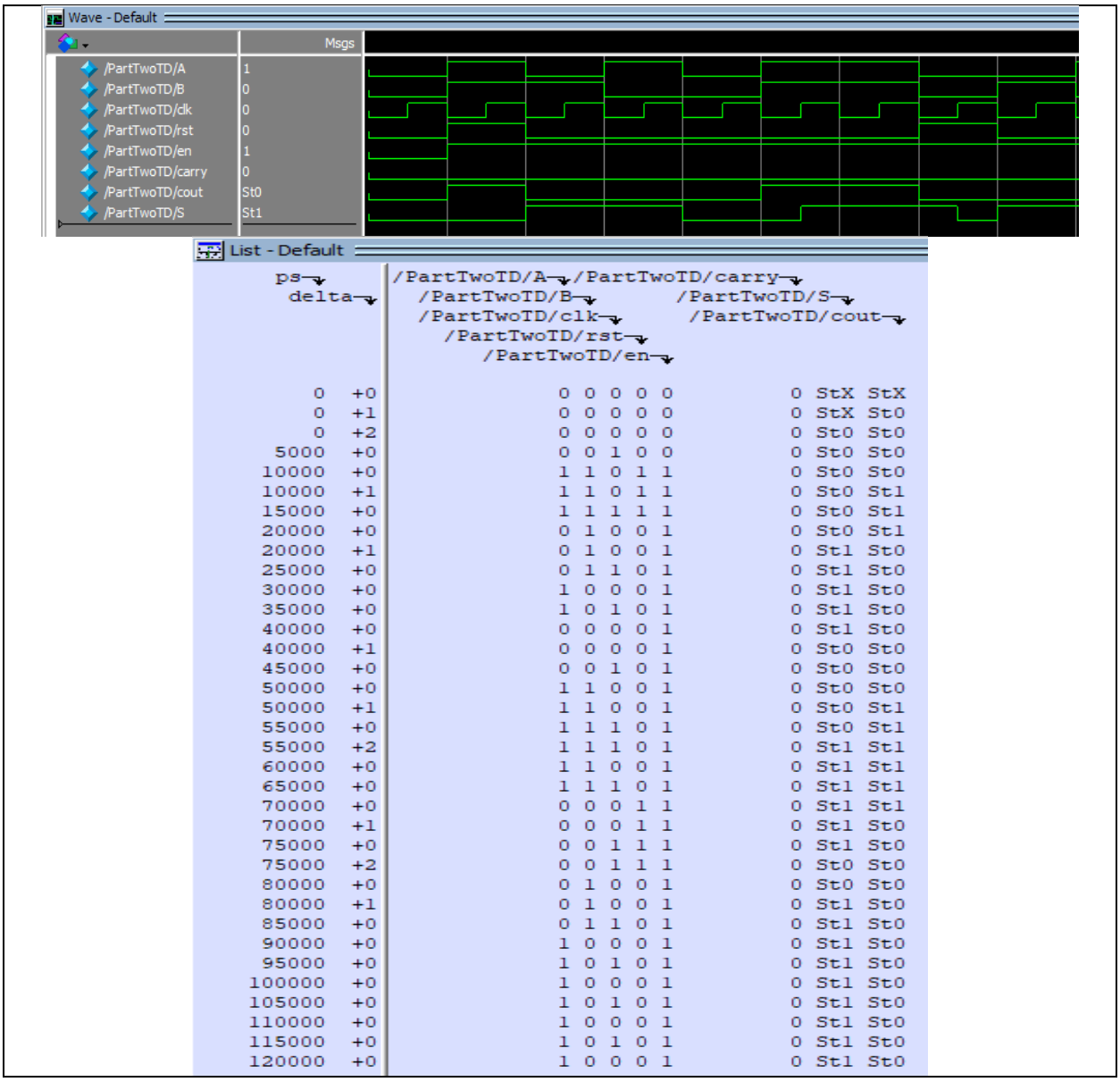
5.1 - Results and lessons learned:

Sections 5.2-5.4 will show the outputs produced from the implementation methods. Overall, the results found in modelsim line very closely with the results expected. In this lab I learned how to use modelsim and all its features, along with a better understanding of Verilog code.

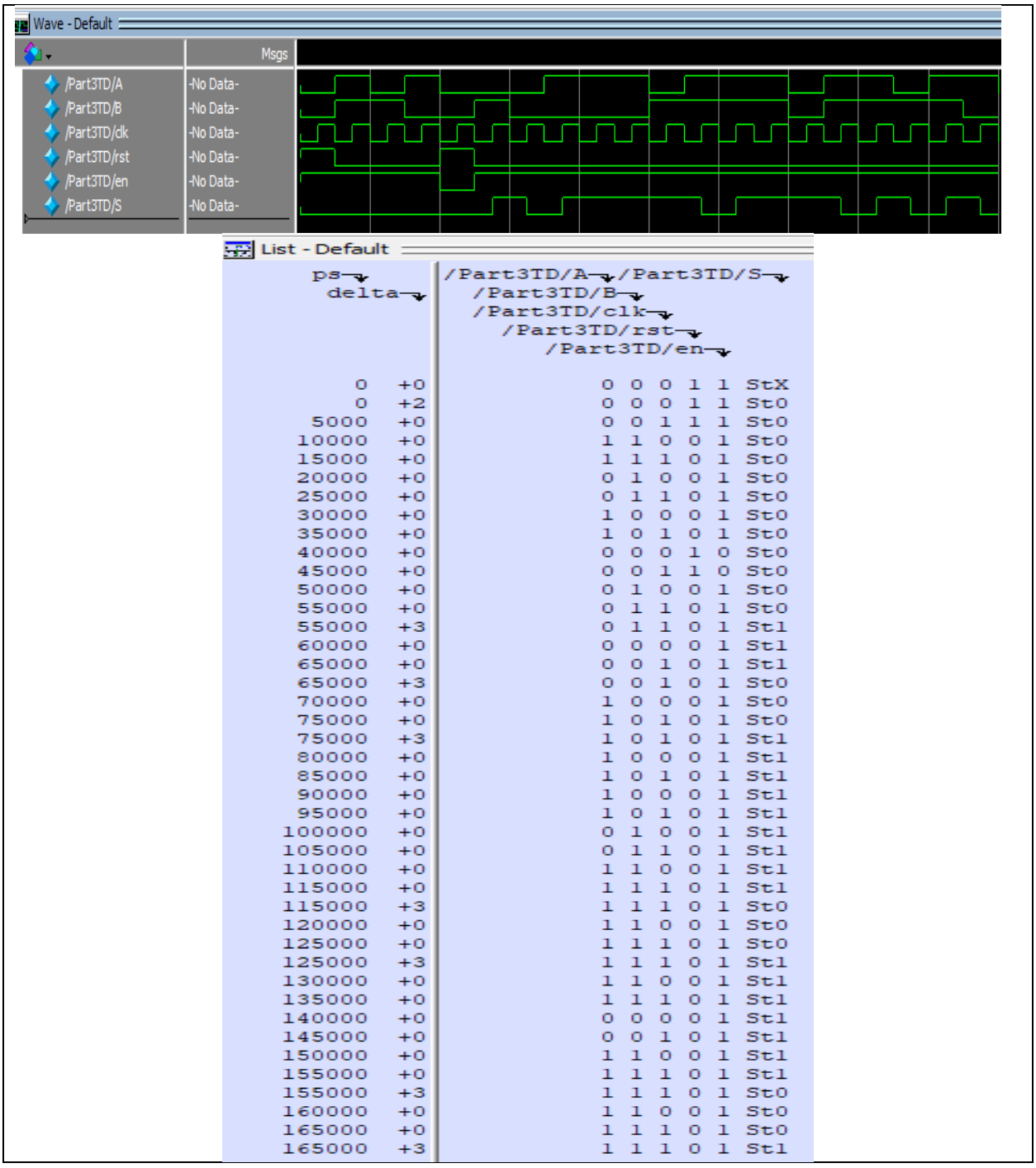
5.2 – Mealy Behavioral Output



5.3 – Mealy Structural Output



5.4 – Moore Behavioral Output



ps	delta	/Part3TD/A	/Part3TD/B	/Part3TD/clk	/Part3TD/rst	/Part3TD/en	/Part3TD/S
0	+0	0	0	0	1	1	StX
0	+2	0	0	0	1	1	St0
5000	+0	0	0	1	1	1	St0
10000	+0	1	1	0	0	1	St0
15000	+0	1	1	1	0	1	St0
20000	+0	0	1	0	0	1	St0
25000	+0	0	1	1	0	1	St0
30000	+0	1	0	0	0	1	St0
35000	+0	1	0	1	0	1	St0
40000	+0	0	0	0	1	0	St0
45000	+0	0	0	1	1	0	St0
50000	+0	0	1	0	0	1	St0
55000	+0	0	1	1	0	1	St0
55000	+3	0	1	1	0	1	St1
60000	+0	0	0	0	0	1	St1
65000	+0	0	0	1	0	1	St1
65000	+3	0	0	1	0	1	St0
70000	+0	1	0	0	0	1	St0
75000	+0	1	0	1	0	1	St0
75000	+3	1	0	1	0	1	St1
80000	+0	1	0	0	0	1	St1
85000	+0	1	0	1	0	1	St1
90000	+0	1	0	0	0	1	St1
95000	+0	1	0	1	0	1	St1
100000	+0	0	1	0	0	1	St1
105000	+0	0	1	1	0	1	St1
110000	+0	1	1	0	0	1	St1
115000	+0	1	1	1	0	1	St1
115000	+3	1	1	1	0	1	St0
120000	+0	1	1	0	0	1	St0
125000	+0	1	1	1	0	1	St0
125000	+3	1	1	1	0	1	St1
130000	+0	1	1	0	0	1	St1
135000	+0	1	1	1	0	1	St1
140000	+0	0	0	0	0	1	St1
145000	+0	0	0	1	0	1	St1
150000	+0	1	1	0	0	1	St1
155000	+0	1	1	1	0	1	St1
155000	+3	1	1	1	0	1	St0
160000	+0	1	1	0	0	1	St0
165000	+0	1	1	1	0	1	St0
165000	+3	1	1	1	0	1	St1

6. Appendix

Appendix 1: Mealy Behavioral Implementation and Test Bench

```
module PartOne(input A, B,  clk, rst, en, output reg S);

        reg state = 0;
        reg next = 0;

        always @( A,B,state)begin
                case(state)
                        0 : if (A == 0 && B == 0) begin next = 0; S= 0; end
                                else if (A ==1 && B == 1) begin next = 1; S = 0; end
                                else begin next = 0; S = 1; end
                        1 : if (A == 0 && B == 0) begin next = 0; S= 1; end
                                else if (A ==1 && B == 1) begin next = 1; S = 1; end
                                else begin next = 1; S = 0; end
                        endcase
        end

        always @(posedge  clk)begin

                if (rst)begin
                        state = 0;
                end
                else begin
                        if (en) begin state = next; end
                end
         end
endmodule
-------------------------------------------------------------------------------------------------------------
`timescale 1ns/100 ps
module PartOneTD;
        reg A = 0, B = 0, clk = 0, rst = 0, en = 0;
        wire S;

        initial
          begin

        #10      // 0ns
        rst = 1; A = 1; B = 1;  en = 1;

        #10      // 10ns
        rst = 0; A = 0; B = 1;  en = 1;
```

```
        #10 // 20ns
        rst = 0; A = 1; B = 0;  en = 1;

        #10 // 30ns
        rst = 0; A = 0; B = 0;  en = 1;

        #10 // 40ns
        rst = 0; A = 1; B = 1;  en = 1;

        #10 // 50ns
        rst = 0; A = 1; B = 1;  en = 1;

        #10 // 60ns
        rst = 1; A = 0; B = 0;  en = 1;

        #10 // 70 ns
        rst = 0; A = 0; B = 1;  en = 1;

        #10 // 80 ns
        rst = 0; A = 1; B = 0;  en = 1;
        end

  always
     begin
     clk = #5 ~clk;
     end

   PartOne uut (.A(A),.B(B),.clk(clk),.en(en),.rst(rst),.S(S));

endmodule
```

Appendix 2: Mealy Structural Implementation and Test Bench

```
module Mealy(input A, B, carry, en, rst, clk, output S, cout);
    wire cin, x1, x2, x3;
    wire w0, w1, w2, w3, w4, w5, w6;

          //  Cin value
    DFF A0(carry, clk, rst, en, cin);

          // XOR A and B
    and(w0, A, B);
    or (w1, A, B);
    not(w2, w0);
    and(w3, w2, w1);
```

```
        // XOR A, B, and CIN
    and(w4, w3, cin);
    or (w5, w3, cin);
    not(w6, w4);
    and(S, w6, w5);
        // Perform operation on inputs
    and A2(x1, A, B);
    and A3(x2, A, cin);
    and A4(x3, B, cin);
    or A5(cout, x1, x2, x3);
endmodule


module DFF(input D, clk, rst, en, output reg Q);
    initial
    begin
    Q <= 0;
    end
    always @(posedge clk) begin
    if(rst) Q <= 0;
    else if(en) Q <= D;
    end
endmodule
```

```
`timescale 1ns/100 ps

module PartTwoTD;
        reg A = 0, B = 0, carry = 0, clk = 0, rst = 0, en = 0;
        wire S, cout;

        initial
          begin

        //0ns
        rst = 1; A = 1;  B = 1;  en = 1;

        #10     //10ns
        rst = 0; A = 0;  B = 1;  en = 1;

        #10     //20ns
        rst = 0; A = 1;  B = 0;  en = 1;

        #10     //30ns
        rst = 0; A = 0;  B = 0;  en = 1;

        #10     //40ns
        rst = 0; A = 1;  B = 1;  en = 1;
```

```
        #10     //50ns
        rst = 0; A = 1;  B = 1;  en = 1;

        #10 //60ns
        rst = 1; A = 0;  B = 0;  en = 1;

        #10     //70ns
        rst = 0; A = 0;  B = 1;  en = 1;

        #10     //80ns
        rst = 0; A = 1;  B = 0;  en = 1;
        end


  always
     begin
     clk = #5 ~clk;
     end

    Mealy uut (.A(A),.B(B),.carry(cout),.en(en),.rst(rst),.clk(clk),.S(S),.cout(cout));

endmodule
```

Appendix 3: Moore Behavioral Implementation and Test Bench

```
module Part3(input A,B,clk,rst,en, output reg S);

        reg [1:0] state = 0;
        reg [1:0] next = 0;

        always @(A, B, state)begin
                case(state)
                        0 : begin    S = 0;
                                if (A == 0 && B == 0) begin next = 0; end
                                else if (A ==1 && B == 1) begin next = 2; end
                                else begin next = 1; end
                           end
                        1 : begin S = 1;
                                if (A == 0 && B == 0) begin next = 0; end
                                else if (A ==1 && B == 1) begin next = 2; end
                                else begin next = 1; end
                           end
                        2 : begin S = 0;
                                if (A == 0 && B == 0) begin next = 1; end
```

```
                                else if (A ==1 && B == 1) begin next = 3; end
                                else begin next = 2; end
                            end
                        3 : begin S = 1;
                                if (A == 0 && B == 0) begin next = 1; end
                                else if (A ==1 && B == 1) begin next = 3; end
                                else begin next = 2; end
                            end
                        endcase
        end

        always @(posedge clk)begin

                if (rst)begin
                        state = 0;
                end
                else begin
                        if (en) begin state = next; end
                end
         end

endmodule
```

```
`timescale 1ns/100 ps

module Part3TD;
        reg A = 0, B = 0, clk = 0, rst = 0, en = 0;
        wire S;

        initial
          begin

        // 0ns
        rst = 1; A = 0; B = 0;  en = 1;

        #10 // 10ns
        rst = 0; A = 1; B = 1;  en = 1;

        #10 // 20ns
        rst = 0; A = 0; B = 1;  en = 1;

        #10 // 30ns
        rst = 0; A = 1; B = 0;  en = 1;

        #10 // 40ns
        rst = 1; A = 0; B = 0;  en = 0;
```

```
#10 // 50ns
rst = 0; A = 0; B = 1;  en = 1;

#10 // 60ns
rst = 0; A = 0; B = 0;  en = 1;

#10 // 70ns
rst = 0; A = 1; B = 0;  en = 1;

#10 // 80ns
rst = 0; A = 1; B = 0;  en = 1;

#10 // 90 ns
rst = 0; A = 1; B = 0;  en = 1;

#10      //100ns
rst = 0; A = 0; B = 1;  en = 1;

#10 //110ns
rst = 0; A = 1; B = 1;  en = 1;

#10 // 120ns
rst = 0; A = 1; B = 1;  en = 1;

#10 // 130ns
rst = 0; A = 1; B = 1;  en = 1;

#10 // 140ns
rst = 0; A = 0; B = 0;  en = 1;

#10 // 150ns
rst = 0; A = 1; B = 1;  en = 1;

#10 // 160ns
rst = 0; A = 1; B = 1;  en = 1;

#10 // 170ns
rst = 0; A = 0; B = 1;  en = 1;

#10 // 180ns
rst = 0; A = 1; B = 1;  en = 1;

#10 // 190ns
rst = 0; A = 1; B = 0;  en = 1;

#10 // 200ns
```

```
        rst = 0; A = 0; B = 0;  en = 1;
        end


  always
     begin
     clk = #5 ~clk;
     end

    Part3 uut (.A(A),.B(B),.clk(clk),.rst(rst),.en(en),.S(S));

endmodule
```