Toby Chappell (tchappell@chapman.edu)
Dana Davidson (ddavidson@chapman.edu)
Donner Hanson (hanso127@mail.chapman.edu)

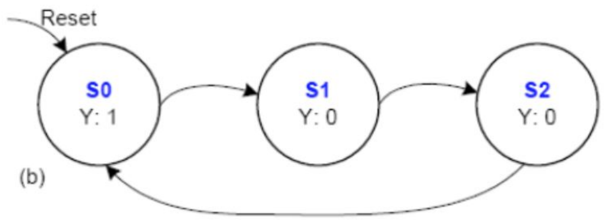
Introduction

For this assignment, we simulated a decoder and a FSM counter using Vivado.

Expected Truth Table for Decoder

sw	led
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

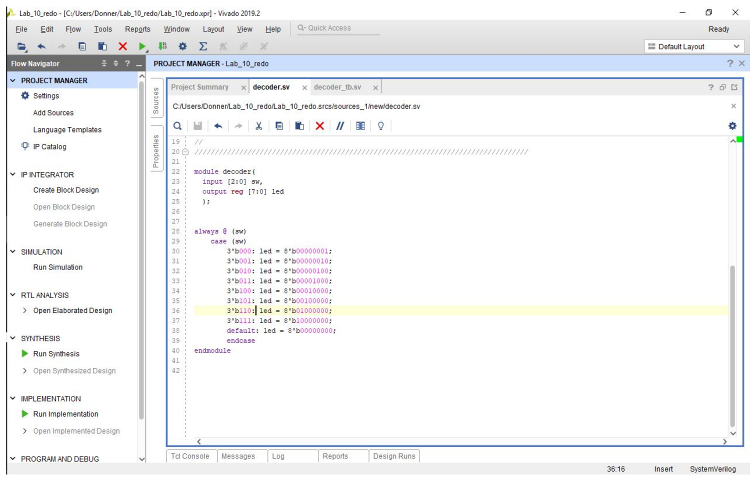
Expected State Transition Diagram for FSM Counter



Procedure

Decoder

To simulate the decoder we specified 1 3-bit input (sw) and one 8-bit output (led). We then assigned led according to the truth table given above

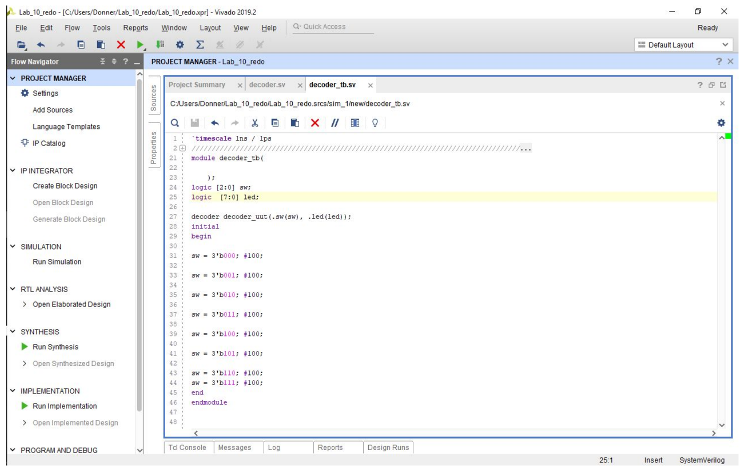


C:/Users/Donner/Lab_10_redo/Lab_10_redo.srcs/sources_1/new/decoder.sv

```
//
////////////////////////////////////////////////////////////////////////////////

module decoder(
  input [2:0] sw,
  output reg [7:0] led
  );


always @ (sw)
    case (sw)
        3'b000: led = 8'b00000001;
        3'b001: led = 8'b00000010;
        3'b010: led = 8'b00000100;
        3'b011: led = 8'b00001000;
        3'b100: led = 8'b00010000;
        3'b101: led = 8'b00100000;
        3'b110: led = 8'b01000000;
        3'b111: led = 8'b10000000;
        default: led = 8'b00000000;
        endcase
endmodule
```

Decoder Test Bench

We then created a test bench for our decoder to simulate our design.

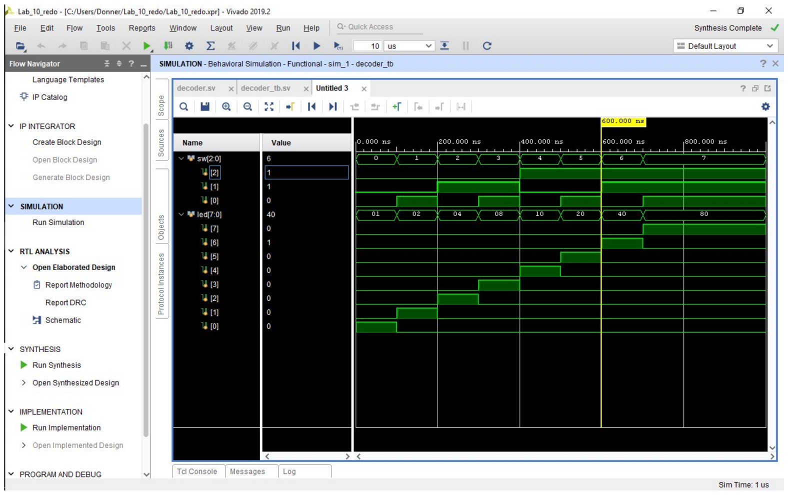


C:/Users/Donner/Lab_10_redo/Lab_10_redo.srcs/sim_1/new/decoder_tb.sv

```
`timescale 1ns / 1ps
///////////////////////////////////////////////////////////////////////////////...
module decoder_tb(

    );
logic [2:0] sw;
logic  [7:0] led;

decoder decoder_uut(.sw(sw), .led(led));
initial
begin

sw = 3'b000; #100;

sw = 3'b001; #100;

sw = 3'b010; #100;

sw = 3'b011; #100;

sw = 3'b100; #100;

sw = 3'b101; #100;

sw = 3'b110; #100;
sw = 3'b111; #100;
end
endmodule
```

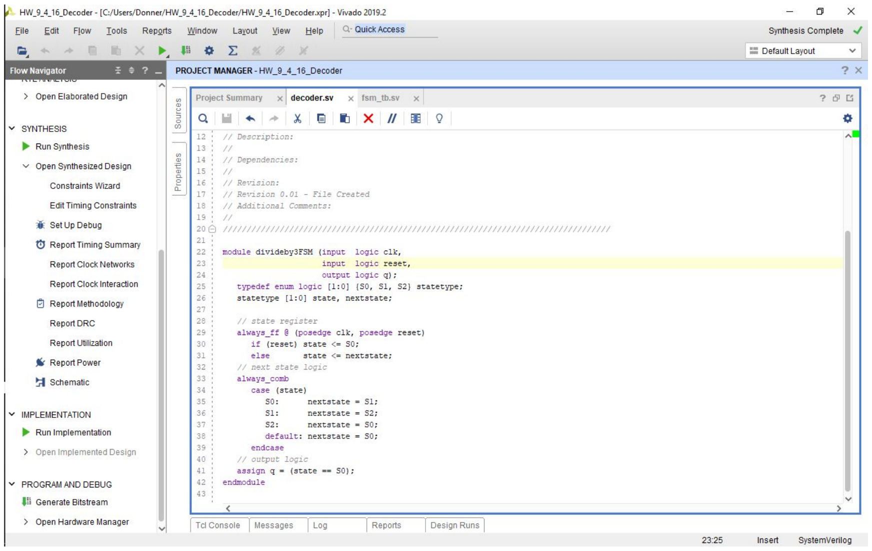
Decoder Waveform

We then ran the simulation of our decoder and came up with the following waveform.


Lab_10_redo - [C:/Users/Donner/Lab_10_redo/Lab_10_redo.xpr] - Vivado 2019.2
SIMULATION - Behavioral Simulation - Functional - sim_1 - decoder_tb
decoder.sv
decoder_tb.sv
Untitled 3
sw[2:0]
led[7:0]
600.000 ns
Sim Time: 1 us

FSM 3 Cycle Counter

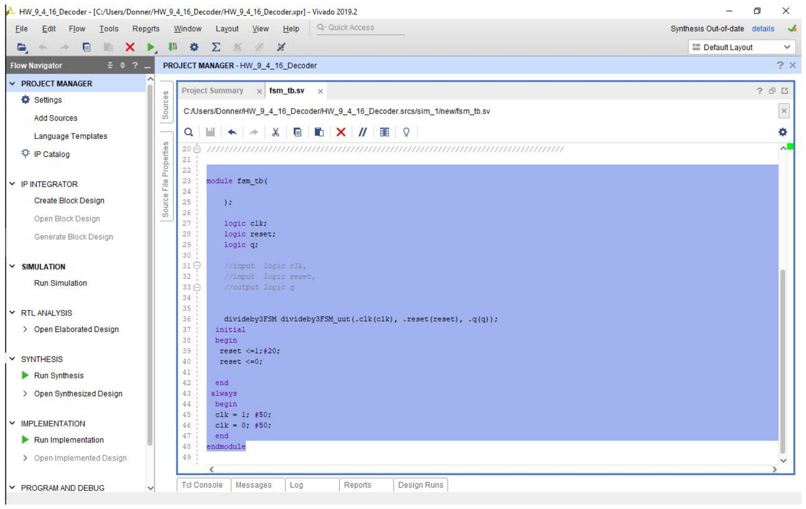
To simulate the FSM we specified a clk and a reset as inputs and a q as output. We then assigned q according to the truth table given above


HW_9_4_16_Decoder - [C:/Users/Donner/HW_9_4_16_Decoder/HW_9_4_16_Decoder.xpr] - Vivado 2019.2
PROJECT MANAGER - HW_9_4_16_Decoder
Project Summary
decoder.sv
fsm_tb.sv

```
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module divideby3FSM (input  logic clk,
                     input  logic reset,
                     output logic q);
  typedef enum logic [1:0] {S0, S1, S2} statetype;
  statetype [1:0] state, nextstate;

  // state register
  always_ff @ (posedge clk, posedge reset)
     if (reset) state <= S0;
     else       state <= nextstate;
  // next state logic
  always_comb
     case (state)
        S0:      nextstate = S1;
        S1:      nextstate = S2;
        S2:      nextstate = S0;
        default: nextstate = S0;
     endcase
  // output logic
  assign q = (state == S0);
endmodule
```

FSM 3 Cycle Counter Test Bench

We then created a test bench for our FSM to simulate our design.

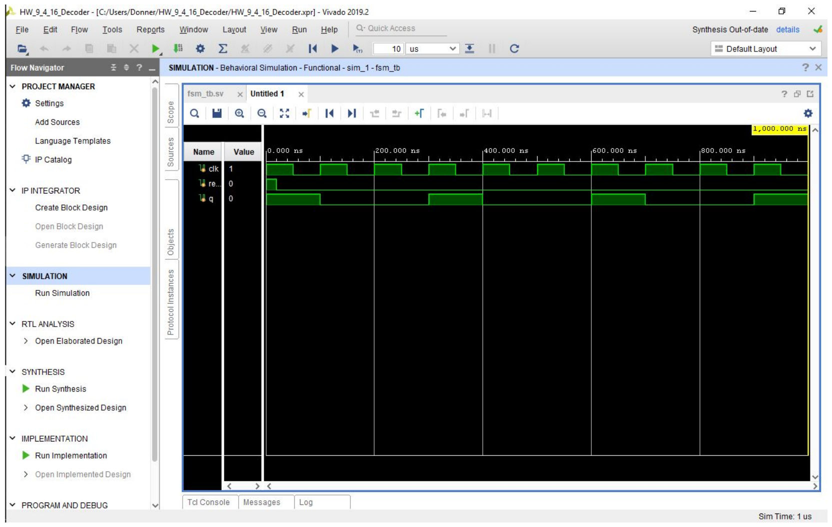


C:/Users/Donner/HW_9_4_16_Decoder/HW_9_4_16_Decoder.srcs/sim_1/new/fsm_tb.sv

```
//////////////////////////////////////////////////////////////////////////////////


module fsm_tb(

    );

    logic clk;
    logic reset;
    logic q;

    //input  logic clk,
    //input  logic reset,
    //output logic q


    divideby3FSM divideby3FSM_uut(.clk(clk), .reset(reset), .q(q));
  initial
  begin
   reset <=1;#20;
   reset <=0;

  end
 always
  begin
  clk = 1; #50;
  clk = 0; #50;
  end
endmodule
```

FSM 3 Cycle Counter Waveform

We then ran the simulation of our FSM and came up with the following waveform.



Conclusion

Based on our results for the decoder, our design works correctly. Each value of led corresponds to the correct sw value.

Based on our results for the FSM, our design works correctly. The value of q is high for 1 out of every 3 clock cycles.

Text of code for Decoder:
decoder.sv

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
///
// Company:
// Engineer:
//
// Create Date: 05/12/2020 04:23:06 PM
// Design Name:
// Module Name: decoder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///

module decoder(
  input [2:0] sw,
  output reg [7:0] led
  );

always @(sw)
        case (sw)
        3'b000: led = 8'b00000001;
        3'b001: led = 8'b00000010;
        3'b010: led = 8'b00000100;
        3'b011: led = 8'b00001000;
        3'b100: led = 8'b00010000;
        3'b101: led = 8'b00100000;
        3'b110: led = 8'b01000000;
        3'b111: led = 8'b10000000;
        default: led = 8'b00000000;
        endcase
endmodule
```

#########################

decoder_tb.sv

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
///
// Company:
// Engineer:
//
// Create Date: 05/12/2020 04:32:09 PM
// Design Name:
// Module Name: decoder_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
///
module decoder_tb(

        );
logic [2:0] sw;
logic  [7:0] led;

decoder decoder_uut(.sw(sw), .led(led));
```

```
initial
        begin
        sw = 3'b000; #100;
        sw = 3'b001; #100;
        sw = 3'b010; #100;
        sw = 3'b011; #100;
        sw = 3'b100; #100;
        sw = 3'b101; #100;
        sw = 3'b110; #100;
        sw = 3'b111; #100;
        end
endmodule


###############################################################
#############
decoder.xdc

# Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
        set_property IOSTANDARD LVCMOS33[get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
        set_property IOSTANDARD LVCMOS33[get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
        set_property IOSTANDARD LVCMOS33[get_ports {sw[2]}]

# LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
        set_property IOSTANDARD LVCMOS33[get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
        set_property IOSTANDARD LVCMOS33[get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
        set_property IOSTANDARD LVCMOS33[get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
        set_property IOSTANDARD LVCMOS33[get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
        set_property IOSTANDARD LVCMOS33[get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
        set_property IOSTANDARD LVCMOS33[get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
        set_property IOSTANDARD LVCMOS33[get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
        set_property IOSTANDARD LVCMOS33[get_ports {led[7]}]
```

Text of code for FSM:

dividebyFSM.sv

```
`timescale 1ns / 1ps
 //////////////////////////////////////////////////////////////////////////////////
 /
 // Company:
 // Engineer:
 //
 // Create Date: 05/13/2020 07:39:47 PM
 // Design Name:
 // Module Name: decoder
 // Project Name:
 // Target Devices:
 // Tool Versions:
 // Description:
 //
 // Dependencies:
 //
 // Revision:
 // Revision 0.01 - File Created
 // Additional Comments:
 //
 //////////////////////////////////////////////////////////////////////////////////
 /
 module divideby3FSM (input  logic clk,
             input  logic reset,
              output logic q);
  typedef enum logic [1:0] {S0, S1, S2} statetype;
  statetype [1:0] state, nextstate;
  // state register
  always_ff @ (posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;
  // next state logic
  always_comb
    case (state)
      S0:     nextstate = S1;
      S1:     nextstate = S2;
      S2:     nextstate = S0;
      default: nextstate = S0;
    endcase
  // output logic
  assign q = (state == S0);
endmodule
############
fsm_tb.sv
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/
// Company:
// Engineer:
//
// Create Date: 05/13/2020 07:43:13 PM
// Design Name:
// Module Name: fsm_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
/
module fsm_tb(
  );

  logic clk;
  logic reset;
  logic q;

  divideby3FSM divideby3FSM_uut(.clk(clk), .reset(reset), .q(q));

initial
 begin
   reset <= 1; #20;
   reset <= 0;
end
always
  begin
```

```
        clk = 1; #50;
        clk = 0; #50;
    end
endmodule
```