Sequence Detector (Verilog) – Case Study

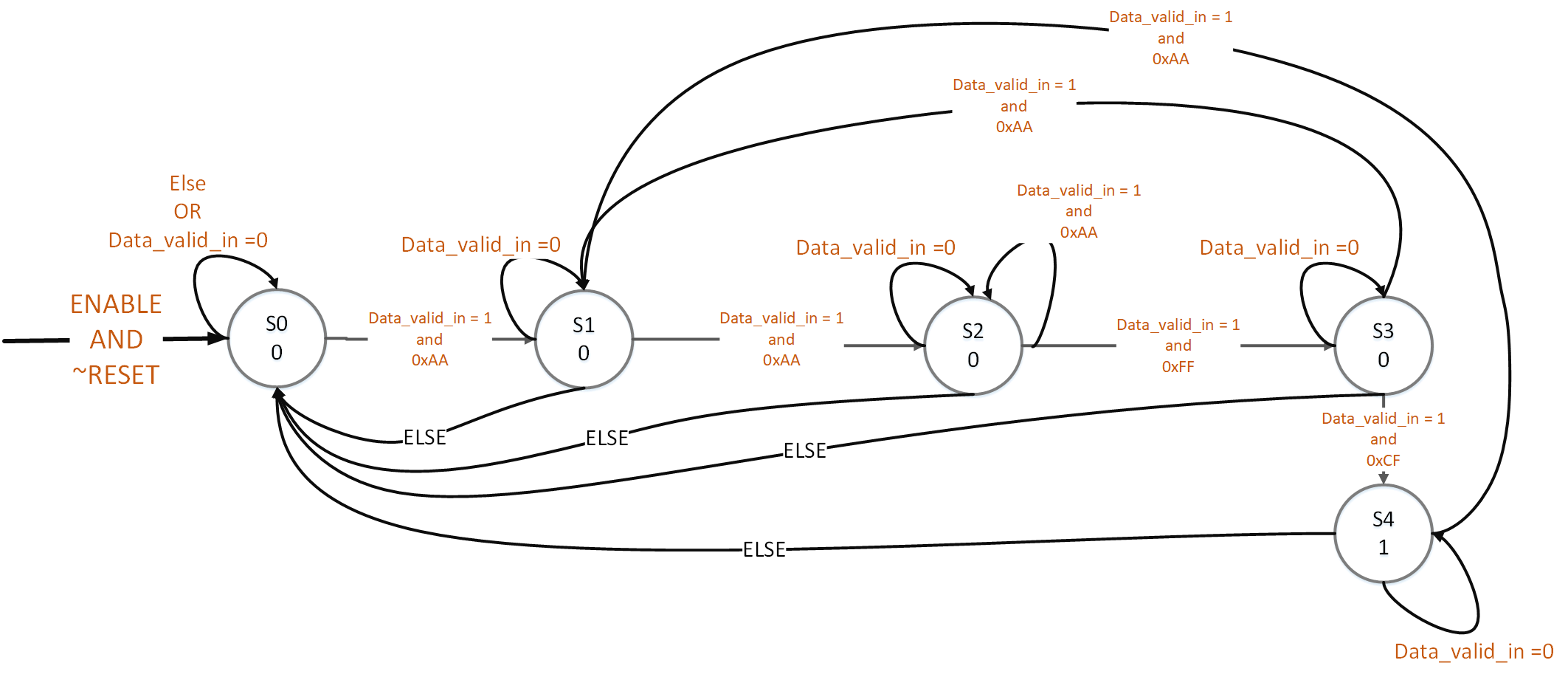
Report

G.S. Umair Ahmadh

[umaireravur@gmail.com](mailto:umaireravur@gmail.com)

15.04.2020

1) Sequence diagram for the design



2) Verilog Design file

|  |
| --- |
| //Sequence detection (case study)  `timescale 1ns / 1ps  module SQD(  //defining input ports  input[7:0] data\_in,  input reset,  input clk,  input enable,  input data\_valid\_in,  //output ports  output reg detected\_out = 0,  output reg data\_valid\_out  );  //states  parameter S0 = 3'b000,  S1 = 3'b001,  S2 = 3'b011,  S3 = 3'b010,  S4 = 3'b110;  //registers to track current and next states  reg [2:0] current\_state, next\_state;  //For each clocks, for the purpose of updating the state, if the reset goes high, current state will be S0  //if enable goes low, state will go to S0  //Else: the next state will be updated as the current state  always @ (posedge clk, posedge reset, posedge enable)  begin  if(reset==1 | enable==0)  current\_state <= S0;  else  current\_state <= next\_state;  end  //wheneever the current state or the data\_in or data\_valid\_in changes, it has to determine  //which state is going to be the next  always @(current\_state, data\_in, data\_valid\_in)  begin  if(enable ==1)  if(data\_valid\_in ==1)  case(current\_state)  S0:begin  if(data\_in == 8'hAA)  next\_state <= S1;  else  next\_state <= S0;  end    S1:begin  if(data\_in ==8'hAA)  next\_state <= S2;  else  next\_state <= S0;  end    S2:begin  if(data\_in ==8'hFF)  next\_state <= S3;  else if(data\_in ==8'hAA)  next\_state <= S2;  else  next\_state <= S0;  end    S3:begin  if(data\_in ==8'hCF)  next\_state <= S4;  else if(data\_in ==8'hAA)  next\_state <= S1;  else  next\_state <= S0;  end    S4:begin  if(data\_in ==8'hAA)  next\_state <= S1;  else if(data\_in ==8'hAA)  next\_state <= S1;  else  next\_state <= S0;  end  default: next\_state<=S0;  endcase  else next\_state <= current\_state;    else next\_state <= S0;  end    //checking for output    always @(current\_state)  begin  case(current\_state)  S0: begin data\_valid\_out <= 0; detected\_out <= 0; end  S1: begin data\_valid\_out <= 0; detected\_out <= 0; end  S2: begin data\_valid\_out <= 0; detected\_out <= 0; end  S3: begin data\_valid\_out <= 0; detected\_out <= 0; end  S4: begin data\_valid\_out <= 1; detected\_out <= 1; end  default: begin data\_valid\_out <= 0; detected\_out <= 0; end  endcase  end  endmodule |

3. Self-checking test bench

|  |
| --- |
| `timescale 1ns / 1ps  module testbench;  //inputs  reg [7:0] data\_in;  reg reset;  reg clk;  reg enable;  reg data\_valid\_in;  //outputs  wire detected\_out;  wire data\_valid\_out;    //Unit under Test  SQD uut (  .data\_in(data\_in),  .reset(reset),  .clk(clk),  .enable(enable),  .data\_valid\_in(data\_valid\_in),    .detected\_out(detected\_out),  .data\_valid\_out(data\_valid\_out)    );    initial begin  clk = 0;  forever #5 clk =~clk;  end  initial  begin  data\_valid\_in =0;  data\_in = 8'h00;  reset = 1;  enable = 0;    #30;  reset = 0;  enable = 1;  data\_valid\_in =1;  #40;  data\_in = 8'hAA;  #20;  data\_in = 8'hFF;    #10;  data\_valid\_in =0;  data\_in = 8'hCF;    #10;  data\_valid\_in =1;  data\_in = 8'hCF;  #10;  data\_in = 8'hEF;  #10;  data\_in = 8'hFF;  #10;  data\_in = 8'hAA;  #20;  data\_in = 8'hFF;  #10;  data\_in = 8'hCF;  #10;    data\_in = 8'hAA;  #20;  data\_in = 8'hFF;  #10;  //trying to reset in the middle of the sequence  reset = 1;  data\_in = 8'hCF;  #10;  reset = 0;  data\_in = 8'hCF;  #10;  data\_in = 8'hAA;  #20;  data\_in = 8'hFF;  #10;  data\_in = 8'hCF;  #10;      #10 $finish;  end    endmodule |

4) Test Result (Behavioral Simulation)

