VLSI DESIGN LAB. ASSIGNMENT -6

SAURAV GUPTA

ROLL No. – 153070046

Dept.- Electrical Engineering

Specialisation – Microelectronics (TA)

Batch - 2015-17

Teacher concerned – VIRENDRA SINGH

Problem Statement:

We want to design a data compression circuit using run length encoding. It replaces continuously repeated occurrences of a byte by a repeat count and the byte value. The circuit receives a fresh byte at every positive transition of an externally supplied clock. We shall use the ESC character (code = 1BH) to signal the use of a repeat count. Therefore, if the ESC character itself appears in the input stream, it has to be handled in a special way. The output is byte wide and every data byte being output is signalled by a rising transition on a Data Valid line.

- **If any character 'c' repeats n times in the input stream with n > 2, we output the three byte sequence ESC n c.**
- **If n ESC characters arrive contiguously in the input stream, we output the 3 byte sequence ESC n ESC, where n can be from 1 to 255.**
- **Otherwise, we just output the received characters without any change.**
- **If the repeat count is more than 255, we handle the first 255 characters as above and treat the 256th occurrence on wards as if a new character has been received.**

Write a synthesizable behavioural description of the above circuit in Verilog. The external system provides a 'fast clock' and a 'data clock' which is at ¼ the frequency of fast clock. Fresh data is input/output at rising edge of 'data clock'. Since the output data rate may be less or more than the data rate over short periods, provision must be made for buffering the data and providing a 'data valid' (Data Valid line.) output which becomes 1 in a clock signal in which valid data is being output.

A single entity---architecture pair or module is to be described, in synthesizable behavioural style. No component instantiation is permitted. Standard Logic arithmetic libraries can be used and integer type signals are considered synthesizable. A clear block diagram must accompany the description in your report.

Sol:

Data is received at every data clock . Since fast clock is 4 times faster than data clock we have to take necessary decision and be ready with the output (if required) at every fourth fast clock.

FSM of Run Length Encoding

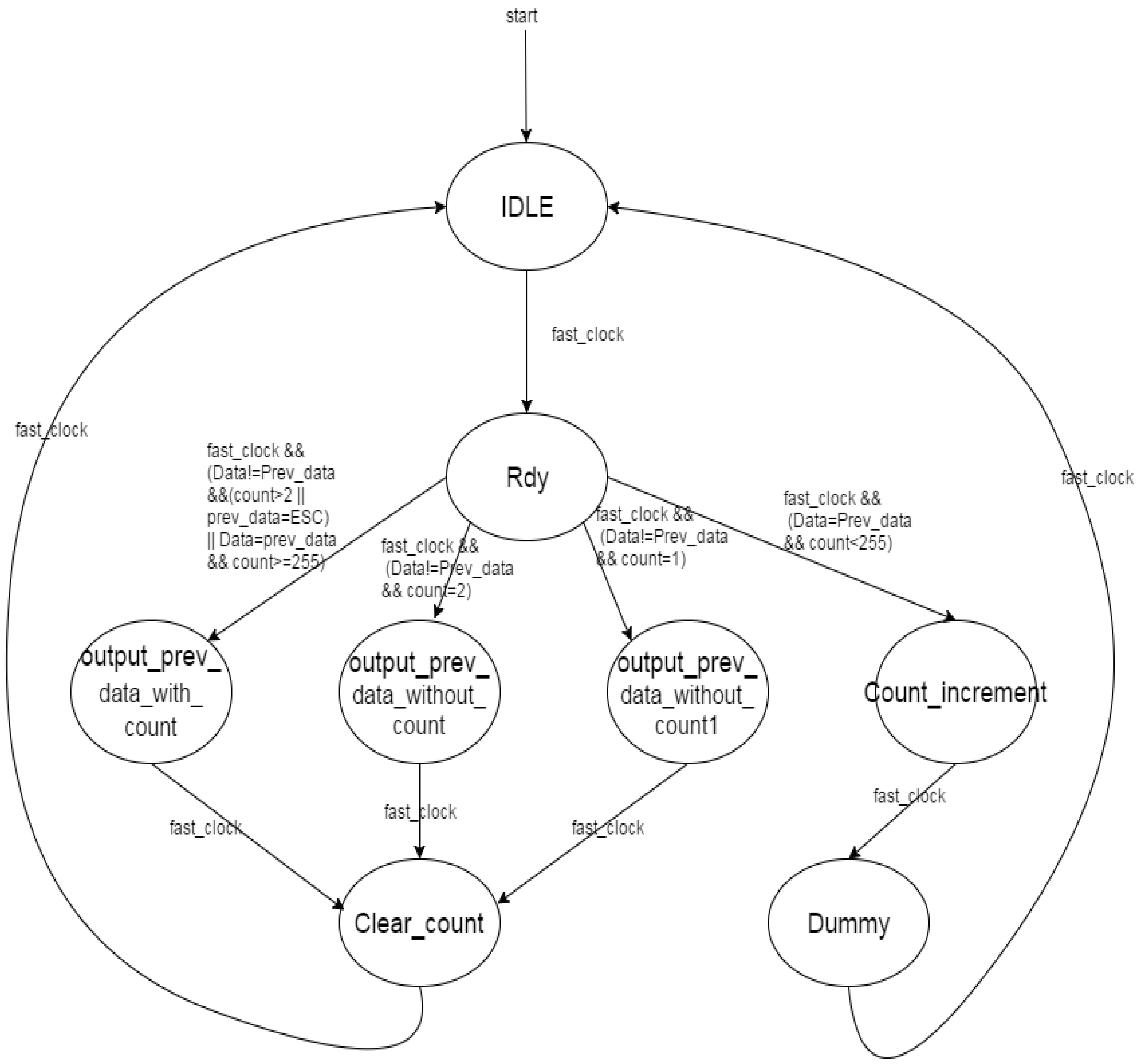
start
IDLE
fast_clock
Rdy
fast_clock
fast_clock && (Data!=Prev_data &&(count>2 || prev_data=ESC) || Data=prev_data && count>=255)
fast_clock && (Data!=Prev_data && count=2)
fast_clock && (Data!=Prev_data && count=1)
fast_clock && (Data=Prev_data && count<255)
fast_clock
output_prev_ data_with_ count
output_prev_ data_without_ count
output_prev_ data_without_ count1
Count_increment
fast_clock
fast_clock
fast_clock
fast_clock
Clear_count
Dummy

fig.1 FSM of RLE

VERILOG CODE:

Run Length Encoding

```
module
RLE(Data_In,Enable,fast_clock,data_clock,start,Data_out,Data_valid);
     input [7:0] Data_In;
     input Enable;
     input fast_clock;
     input data_clock;
     input start;

     output reg [7:0] Data_out;
     output reg Data_valid;

     reg [39:0] Data_buffer=40'h0000000000;
     reg Data_push;
     reg [7:0] prev_Data,prev_prev_Data;
     reg [7:0] count=8'h00,count_temp;
     reg countgt2,counteq255,Data_eq_Prev,Data_buffer2,Data_buffern;

     parameter ESC = 8'h1B;
     parameter IDLE=0, Rdy=1, count_increment =2,
output_prev_data_with_count=3, output_Buffer_data_without_count=4,
output_Buffer_data_without_count1=5, clear_count=6, Dummy=7;
     reg[2:0] state_signal=0, next_state_var;

     always @(posedge data_clock)
     begin
          if(start==1'b1)
               Data_eq_Prev=1'b1;
          else if(Data_In==prev_Data)
               Data_eq_Prev=1'b1;
          else
               Data_eq_Prev=1'b0;
     end

     always @(posedge data_clock)
     begin
          if(count==8'hFF)
               counteq255 = 1'b1;
          else
               counteq255= 1'b0;
     end

     always
@(Enable,Data_eq_Prev,state_signal,countgt2,counteq255,prev_prev_Data,
count)
     begin
     if(Enable==1'b1) begin
```

```
          case(state_signal)
          IDLE: begin
                    next_state_var<= Rdy;
          end

          Rdy: begin
               if(Data_eq_Prev==1'b1 && counteq255==1'b0)
                    next_state_var<= count_increment;
               else if((Data_eq_Prev==1'b0 &&
(countgt2==1'b1||prev_prev_Data == ESC)) || (Data_eq_Prev==1'b1 &&
counteq255==1'b1))
                    next_state_var<= output_prev_data_with_count;
               else if(Data_eq_Prev==1'b0 && count==8'h01)
                    next_state_var <=
output_Buffer_data_without_count1;
               else
                    next_state_var<=
output_Buffer_data_without_count;
          end

          count_increment:begin
               next_state_var<= Dummy;
          end

          output_prev_data_with_count: begin
               next_state_var<= clear_count;
          end

          output_Buffer_data_without_count: begin
               next_state_var<= clear_count;
          end

          output_Buffer_data_without_count1: begin
               next_state_var<= clear_count;
          end

          clear_count: begin
               next_state_var<= IDLE;
          end

          Dummy: begin
               next_state_var<=IDLE;
          end
          endcase
     end

     else
          next_state_var<=IDLE;

     end

     always@(posedge fast_clock)
```

```
begin
      state_signal<=next_state_var;
end

always@(state_signal)
begin
      case(state_signal)
      IDLE: begin
            Data_push<= 1'b0;
      end

      Rdy: begin
            Data_push<= 1'b0;
            prev_Data = Data_In;
            if(count>8'h02)
                  countgt2=1'b1;
            else
                  countgt2=1'b0;
      end

      count_increment:begin
            Data_push<= 1'b0;
            count<=count+8'h01;

      end

      output_prev_data_with_count: begin
            Data_push<= 1'b1;
            Data_buffern<=1'b1;
            Data_buffer2 <= 1'b0;
      end

      output_Buffer_data_without_count: begin
            Data_push<= 1'b1;
            Data_buffern<=1'b0;
            Data_buffer2 <= 1'b1;
      end

      output_Buffer_data_without_count1: begin
            Data_push<= 1'b1;
            Data_buffern<=1'b0;
            Data_buffer2 <= 1'b0;
      end

      clear_count: begin
            count_temp= count;
            count=8'h01;
      end

      Dummy: begin
      end
```

```
            endcase
      end


      always @(posedge data_clock)
      begin
      if(Data_buffer[15:0]==16'h0000)begin
            Data_out =  Data_buffer[23:16];
            Data_buffer = Data_buffer >> 24;
      end
      else if(Data_buffer[7:0]==8'h00)begin
            Data_out =  Data_buffer[15:8];
            Data_buffer = Data_buffer >> 16;
      end
      else begin
            Data_out =  Data_buffer[7:0];
            Data_buffer = Data_buffer >> 8;
      end

      if(Data_push==1'b1)begin
            if(Data_buffern==1'b1)
                  Data_buffer[39:16] = {prev_prev_Data,count_temp,ESC};
            else if(Data_buffer2==1'b1) begin
                  if(Data_buffer[31:16]==8'b0000)
                        Data_buffer[31:16]=
{prev_prev_Data,prev_prev_Data};
                  else
                        Data_buffer[39:24]=
{prev_prev_Data,prev_prev_Data};
            end
            else begin
                  if(Data_buffer[23:16]==8'b00)
                        Data_buffer[23:16]= prev_prev_Data;
                  else if(Data_buffer[31:24]==8'b00)
                        Data_buffer[31:24]= prev_prev_Data;
                  else
                        Data_buffer[39:32]= prev_prev_Data;
            end
      end
            if(Data_out==8'h00)
                  Data_valid = 1'b0;
            else
                  Data_valid = 1'b1;

            prev_prev_Data = prev_Data;
      end
endmodule
```

Test-Bench

```
`timescale 1us/1ps
module RLE_tb;
     wire [7:0] Data_out;
     wire Data_valid;

     reg [7:0] Data_In;
     reg Enable;
     reg fast_clock;
     reg data_clock;
     reg start;

     integer i;

     RLE
DUT(Data_In,Enable,fast_clock,data_clock,start,Data_out,Data_valid);

     initial
     begin
          $dumpfile("run.vcd");
          $dumpvars(0,RLE_tb);
          data_clock=1'b1;
          fast_clock=1'b1;
          Enable=1'b1;
          start=1'b1;
          Data_In= 8'b01100011; //c
          #40 start=1'b0;
          #40 Data_In= 8'b01100011;  //c
          #80 Data_In= 8'b01100011;  //c
          #80 Data_In= 8'b01100011;  //c
          #80 Data_In= 8'b01100010;  //b
          #80 Data_In= 8'b01100010;  //b
          #80 Data_In= 8'b01100001;  //a
          #80 Data_In= 8'b01100110;  //f
          #80 Data_In= 8'b01100110;  //f
          #80 Data_In= 8'b01100101;  //e
          #80 Data_In= 8'b00011011;  //ESC
          #80 Data_In= 8'b00011011;  //ESC
          #80 Data_In= 8'b01100111;  //g
          //for (i=1;i<=255;i=i+1)
               //#80 Data_In= 8'b01101000; //h
          //#80 Data_In= 8'b01101000; //h
          #80 Data_In= 8'b01101000;  //h
          #80 Data_In= 8'b01101000;  //h
          #80 Data_In= 8'b01101000;  //h
          #80 Data_In= 8'b01101000;  //h
          #80 Data_In= 8'b00011011;  //ESC
          #80 Data_In= 8'b00011011;  //ESC
          #80 Data_In= 8'b01101001;  //i
          #80 Data_In= 8'b01101010;  //j
          #80 Data_In= 8'b00011011;  //ESC
          #80 Data_In= 8'b01101011;  //k
          #80 Data_In= 8'b01101011;  //k
```

```
        #80 Data_In= 8'b00000000;   //NULL
        #640 $finish;
  end

  always
  begin
        data_clock = #40 ~data_clock;
  end

  always
  begin
        fast_clock = #10 ~fast_clock;
  end

endmodule
```

Sample Output

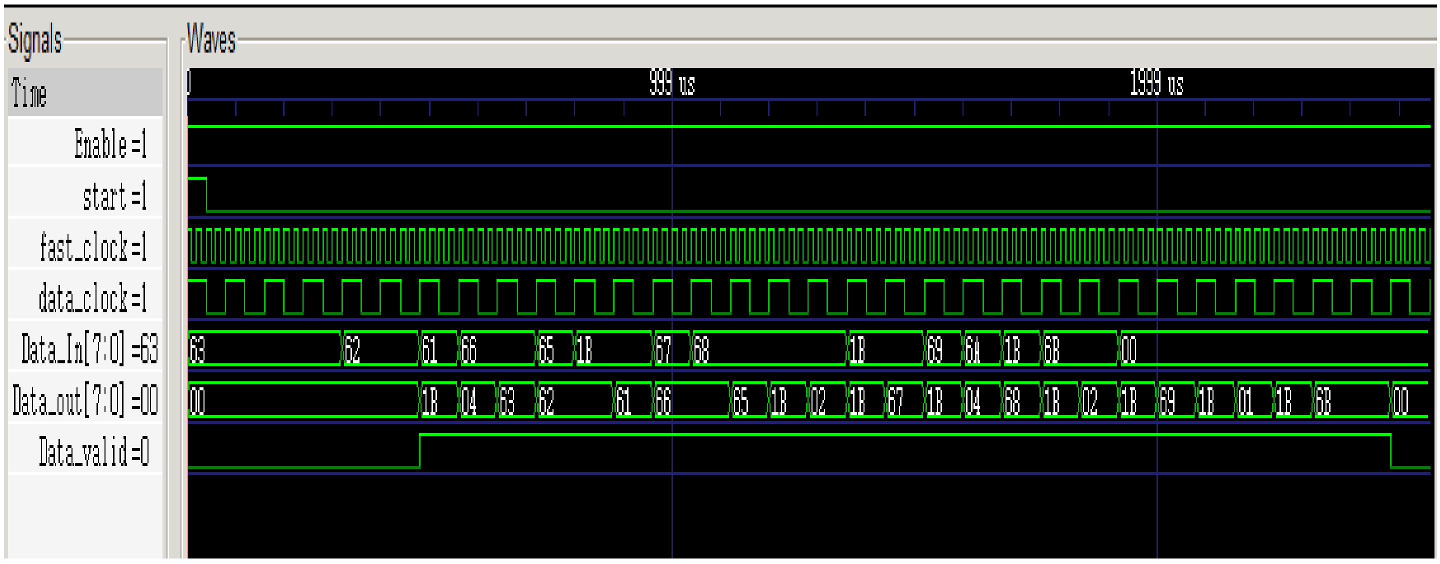

fig. 2 Sample output for given testbench