DESIGN OF SYNCHRONOUS FIFO IN VERILOG

```
module synchronous_fifo #(parameter DEPTH=8, DATA_WIDTH=8) (
 input clk, rst_n,
 input w_en, r_en,
 input [DATA_WIDTH-1:0] data_in,
 output reg [DATA_WIDTH-1:0] data_out,
 output full, empty
);

 parameter PTR_WIDTH = $clog2(DEPTH);
 reg [PTR_WIDTH:0] w_ptr, r_ptr; // addition bit to detect full/empty condition
 reg [DATA_WIDTH-1:0] fifo[DEPTH];
 reg wrap_around;

 // Set Default values on reset.
 always@(posedge clk) begin
  if(!rst_n) begin
   w_ptr <= 0; r_ptr <= 0;
   data_out <= 0;
  end
 end

 // To write data to FIFO
 always@(posedge clk) begin
  if(w_en & !full)begin
   fifo[w_ptr[PTR_WIDTH-1:0]] <= data_in;
   w_ptr <= w_ptr + 1;
  end
 end

 // To read data from FIFO
 always@(posedge clk) begin
  if(r_en & !empty) begin
   data_out <= fifo[r_ptr[PTR_WIDTH-1:0]];
   r_ptr <= r_ptr + 1;
  end
 end

 assign wrap_around = w_ptr[PTR_WIDTH] ^ r_ptr[PTR_WIDTH]; // To check MSB of write and
read pointers are different

 //Full condition: MSB of write and read pointers are different and remainimg bits are same.
 assign full = wrap_around & (w_ptr[PTR_WIDTH-1:0] == r_ptr[PTR_WIDTH-1:0]);

 //Empty condition: All bits of write and read pointers are same.
 //assign empty = !wrap_around & (w_ptr[PTR_WIDTH-1:0] == r_ptr[PTR_WIDTH-1:0]);
 //or
```

```
 assign empty = (w_ptr == r_ptr);
endmodule
```

TESTBENCH

```
module sync_fifo_TB;
 reg clk, rst_n;
 reg w_en, r_en;
 reg [7:0] data_in;
 wire [7:0] data_out;
 wire full, empty;

 synchronous_fifo s_fifo(clk, rst_n, w_en, r_en, data_in, data_out, full, empty);

 always #2 clk = ~clk;
 initial begin
  clk = 0; rst_n = 0;
  w_en = 0; r_en = 0;
  #3 rst_n = 1;
  drive(20);
  drive(40);
  $finish;
 end

 task push();
  if(!full) begin
   w_en = 1;
   data_in = $random;
   #1 $display("Push In: w_en=%b, r_en=%b, data_in=%h",w_en, r_en,data_in);
  end
  else $display("FIFO Full!! Can not push data_in=%d", data_in);
 endtask

 task pop();
  if(!empty) begin
   r_en = 1;
   #1 $display("Pop Out: w_en=%b, r_en=%b, data_out=%h",w_en, r_en,data_out);
  end
  else $display("FIFO Empty!! Can not pop data_out");
 endtask

 task drive(int delay);
  w_en = 0; r_en = 0;
  fork
   begin
    repeat(10) begin @(posedge clk) push(); end
    w_en = 0;
```

```
    end
    begin
      #delay;
      repeat(10) begin @(posedge clk) pop(); end
      r_en = 0;
    end
  join
 endtask

 initial begin
  $dumpfile("dump.vcd"); $dumpvars;
 end
endmodule
```

OUTPUT WAVEFORM ON EDA PLAYGROUND