DESIGN OF ASYNCHRONOUS FIFO IN VERILOG

```
module synchronizer #(parameter WIDTH=3) (input clk, rst_n, [WIDTH:0] d_in, output reg
[WIDTH:0] d_out);
 reg [WIDTH:0] q1;
 always@(posedge clk) begin
  if(!rst_n) begin
   q1 <= 0;
   d_out <= 0;
  end
  else begin
   q1 <= d_in;
   d_out <= q1;
  end
 end
endmodule

module wptr_handler #(parameter PTR_WIDTH=3) (
 input wclk, wrst_n, w_en,
 input [PTR_WIDTH:0] g_rptr_sync,
 output reg [PTR_WIDTH:0] b_wptr, g_wptr,
 output reg full
);

 reg [PTR_WIDTH:0] b_wptr_next;
 reg [PTR_WIDTH:0] g_wptr_next;

 reg wrap_around;
 wire wfull;

 assign b_wptr_next = b_wptr+(w_en & !full);
 assign g_wptr_next = (b_wptr_next >>1)^b_wptr_next;

 always@(posedge wclk or negedge wrst_n) begin
  if(!wrst_n) begin
   b_wptr <= 0; // set default value
   g_wptr <= 0;
  end
  else begin
   b_wptr <= b_wptr_next; // incr binary write pointer
   g_wptr <= g_wptr_next; // incr gray write pointer
  end
 end

 always@(posedge wclk or negedge wrst_n) begin
  if(!wrst_n) full <= 0;
  else      full <= wfull;
 end
```

```
  assign wfull = (g_wptr_next == {~g_rptr_sync[PTR_WIDTH:PTR_WIDTH-1],
g_rptr_sync[PTR_WIDTH-2:0]});

endmodule

module rptr_handler #(parameter PTR_WIDTH=3) (
  input rclk, rrst_n, r_en,
  input [PTR_WIDTH:0] g_wptr_sync,
  output reg [PTR_WIDTH:0] b_rptr, g_rptr,
  output reg empty
);

  reg [PTR_WIDTH:0] b_rptr_next;
  reg [PTR_WIDTH:0] g_rptr_next;

  assign b_rptr_next = b_rptr+(r_en & !empty);
  assign g_rptr_next = (b_rptr_next >>1)^b_rptr_next;
  assign rempty = (g_wptr_sync == g_rptr_next);

  always@(posedge rclk or negedge rrst_n) begin
    if(!rrst_n) begin
      b_rptr <= 0;
      g_rptr <= 0;
    end
    else begin
      b_rptr <= b_rptr_next;
      g_rptr <= g_rptr_next;
    end
  end

  always@(posedge rclk or negedge rrst_n) begin
    if(!rrst_n) empty <= 1;
    else       empty <= rempty;
  end
endmodule

module fifo_mem #(parameter DEPTH=8, DATA_WIDTH=8, PTR_WIDTH=3) (
  input wclk, w_en, rclk, r_en,
  input [PTR_WIDTH:0] b_wptr, b_rptr,
  input [DATA_WIDTH-1:0] data_in,
  input full, empty,
  output reg [DATA_WIDTH-1:0] data_out
);
  reg [DATA_WIDTH-1:0] fifo[0:DEPTH-1];

  always@(posedge wclk) begin
    if(w_en & !full) begin
```

```
    fifo[b_wptr[PTR_WIDTH-1:0]] <= data_in;
   end
  end
  /*
  always@(posedge rclk) begin
   if(r_en & !empty) begin
    data_out <= fifo[b_rptr[PTR_WIDTH-1:0]];
   end
  end
  */
  assign data_out = fifo[b_rptr[PTR_WIDTH-1:0]];
endmodule



module asynchronous_fifo #(parameter DEPTH=8, DATA_WIDTH=8) (
  input wclk, wrst_n,
  input rclk, rrst_n,
  input w_en, r_en,
  input [DATA_WIDTH-1:0] data_in,
  output reg [DATA_WIDTH-1:0] data_out,
  output reg full, empty
);

  parameter PTR_WIDTH = $clog2(DEPTH);

  reg [PTR_WIDTH:0] g_wptr_sync, g_rptr_sync;
  reg [PTR_WIDTH:0] b_wptr, b_rptr;
  reg [PTR_WIDTH:0] g_wptr, g_rptr;

  wire [PTR_WIDTH-1:0] waddr, raddr;

  synchronizer #(PTR_WIDTH) sync_wptr (rclk, rrst_n, g_wptr, g_wptr_sync); //write pointer to
read clock domain
  synchronizer #(PTR_WIDTH) sync_rptr (wclk, wrst_n, g_rptr, g_rptr_sync); //read pointer to write
clock domain

  wptr_handler #(PTR_WIDTH) wptr_h(wclk, wrst_n, w_en,g_rptr_sync,b_wptr,g_wptr,full);
  rptr_handler #(PTR_WIDTH) rptr_h(rclk, rrst_n, r_en,g_wptr_sync,b_rptr,g_rptr,empty);
  fifo_mem fifom(wclk, w_en, rclk, r_en,b_wptr, b_rptr, data_in,full,empty, data_out);

endmodule
```

TESTBENCH

```
module async_fifo_TB;

  parameter DATA_WIDTH = 8;
```

```
wire [DATA_WIDTH-1:0] data_out;
wire full;
wire empty;
reg [DATA_WIDTH-1:0] data_in;
reg w_en, wclk, wrst_n;
reg r_en, rclk, rrst_n;

// Queue to push data_in
reg [DATA_WIDTH-1:0] wdata_q[$], wdata;

asynchronous_fifo as_fifo (wclk, wrst_n,rclk, rrst_n,w_en,r_en,data_in,data_out,full,empty);

always #10ns wclk = ~wclk;
always #35ns rclk = ~rclk;

initial begin
 wclk = 1'b0; wrst_n = 1'b0;
 w_en = 1'b0;
 data_in = 0;

 repeat(10) @(posedge wclk);
 wrst_n = 1'b1;

 repeat(2) begin
  for (int i=0; i<30; i++) begin
   @(posedge wclk iff !full);
   w_en = (i%2 == 0)? 1'b1 : 1'b0;
   if (w_en) begin
    data_in = $urandom;
    wdata_q.push_back(data_in);
   end
  end
  #50;
 end
end

initial begin
 rclk = 1'b0; rrst_n = 1'b0;
 r_en = 1'b0;

 repeat(20) @(posedge rclk);
 rrst_n = 1'b1;

 repeat(2) begin
  for (int i=0; i<30; i++) begin
   @(posedge rclk iff !empty);
   r_en = (i%2 == 0)? 1'b1 : 1'b0;
```

```
      if (r_en) begin
        wdata = wdata_q.pop_front();
        if(data_out !== wdata) $error("Time = %0t: Comparison Failed: expected wr_data = %h,
rd_data = %h", $time, wdata, data_out);
        else $display("Time = %0t: Comparison Passed: wr_data = %h and rd_data = %h",$time,
wdata, data_out);
      end
     end
     #50;
   end

   $finish;
 end

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

OUTPUT WAVEFORM ON EDA PLAYGROUND