Memories

1)

```
module spr_sync_en(clk, din, dout, rst, wr_en, rd_en, blk_select, addr, addr_en, dout_en, parity_out);
    parameter MEM_WIDTH = 16;
    parameter MEM_DEPTH = 1024;
    parameter ADDR_SIZE = 10;
    parameter ADDR_PIPELINE = "FALSE";
    parameter DOUT_PIPELINE = "TRUE";
    parameter PARITY_ENABLE = 1;

    input  clk, rst, wr_en, rd_en, blk_select, addr_en, dout_en;
    input  [MEM_WIDTH-1:0] din;
    input  [ADDR_SIZE-1:0] addr;
    output [MEM_WIDTH-1:0] dout;
    output parity_out;

    reg [MEM_WIDTH-1:0] mem [MEM_DEPTH-1:0];
    wire tmp_parity;
    wire [ADDR_SIZE-1:0] addr_in;
    reg  [ADDR_SIZE-1:0] addr_pipe;
    reg  [MEM_WIDTH-1:0] dout_stg0, dout_stg1;

    parity_out #(MEM_WIDTH) m1(dout, tmp_parity);

    assign addr_in = (ADDR_PIPELINE)? addr_pipe: addr;
    assign dout    = (DOUT_PIPELINE)? dout_stg1: dout_stg0;

    //Read/Write Operation
    always @(posedge clk) begin
        if(rst) begin
            addr_pipe <= 'b0;
            dout_stg1 <= 'b0;
            dout_stg0 <= 'b0;
        end else begin
            if (addr_en)
                addr_pipe <= addr;
            if (dout_en)
                dout_stg1 <= dout_stg0;
```

```
            if (blk_select) begin
                if (wr_en) begin
                    mem [addr_in] <= din;
                end
                if (rd_en) begin
                    dout_stg0 <= mem[addr_in];
                end
            end
        end
    end

    assign parity_out = (PARITY_ENABLE)? tmp_parity: 1'b0;

endmodule
```

```
module spr_sync_en_tb();
  reg wr_en, rd_en, blk_select, addr_en, dout_en ,clk, rst;
  reg [15:0] din;
  reg [9:0] addr;
  wire [15:0] dout;
  wire parity_out;

  spr_sync_en m1(clk, din, dout, rst, wr_en, rd_en, blk_select, addr, addr_en, dout_en, parity_out);

  integer i;

  //clock generation
  initial begin
    clk = 0;
    forever
      #1 clk = ~clk;
  end

  //reset and initial values for inputs
  initial begin
    $readmemh ("mem.dat", m1.mem);
    rst = 1;
    addr = 0;
    addr_en = 0;
    din = 0;
    rd_en = 0;
    blk_select = 0;
    dout_en = 0;
    #50
    rst = 0;
    #100;
```

```
    //test Write operation only
    for (i = 0; i < 10000; i=i+1) begin
      #2
      blk_select = $random;
      din = $random;
      addr = $random;
      wr_en = $random;
      addr_en = $random;
      dout_en = $random;
    end
    //Read operation only
    wr_en = 0;
    for (i = 0; i < 10000; i=i+1) begin
      #2
      blk_select = $random;
      addr = $random;
      rd_en = $random;
      addr_en = $random;
      dout_en = $random;
    end
    //Test both read and write
    rd_en = 1;
    wr_en = 1;
    for (i = 0; i < 10000; i=i+1) begin
      #2
      blk_select = $random;
      addr = $random;
      rd_en = $random;
      wr_en = $random;
      din = $random;
      addr_en = $random;
      dout_en = $random;
    end
    $stop;
  end

  //Test monitor and results
  initial begin
    $monitor("din = %b, dout = %b, parity_out = %b, wr_en = %b, rd_en
  end
endmodule
```

2)

```
module FIFO(data_in, wr_en, rd_en, clk_a, clk_b, rst_n, data_out, full, empty);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 512;
parameter ADDR_SIZE = 9;
input [FIFO_WIDTH-1:0] data_in;
input clk_a, clk_b, rst_n, wr_en, rd_en;
output reg [FIFO_WIDTH-1:0] data_out;
output full, empty;

reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];

reg [ADDR_SIZE:0] wr_ptr, rd_ptr;

// Write Operation
always @(posedge clk_a) begin
    if (!rst_n) begin
        // reset
        wr_ptr <= 0;
    end
    else if (wr_en && full != 1) begin
        mem[wr_ptr[ADDR_SIZE-1:0]] <= data_in;
        wr_ptr <= wr_ptr + 1;
    end
end

// Read Operation
always @(posedge clk_b) begin
    if (!rst_n) begin
        // reset
        rd_ptr <= 0;
    end
    else if (rd_en && empty != 1) begin
        data_out <= mem[rd_ptr[ADDR_SIZE-1:0]];
        rd_ptr <= rd_ptr + 1;
    end
end
```

```
//Output Assignments
assign full = ( (wr_ptr[ADDR_SIZE] != rd_ptr[ADDR_SIZE]) && (wr_ptr[ADDR_SIZE-1:0] == rd_ptr[ADDR_SIZE-1:0]))? 1 : 0;
assign empty = (wr_ptr == rd_ptr)? 1 : 0;

endmodule
```

```
module FIFO_tb();
  reg [15:0] data_in;
  reg wr_en, rd_en, clk_a, clk_b, rst_n;
  wire [15:0] data_out;
  wire full, empty;

  FIFO f1(data_in, wr_en, rd_en, clk_a, clk_b, rst_n, data_out, full, empty);
  integer i;

  initial begin
    clk_a = 0;
    clk_b = 0;
    forever begin
      #1 clk_a = ~clk_a;
      #1 clk_b = ~clk_b;
    end
  end

  //reset and initial values for inputs
  initial begin
    rst_n = 0;
    #50
    rst_n = 1;
    #100;
    rd_en = 0;
    //test Write operation only
    for (i = 0; i < 10000; i=i+1) begin
      #2
      data_in = $random;
      wr_en = $random;
    end
    //Read operation only
    wr_en = 0;
    for (i = 0; i < 10000; i=i+1) begin
      #2
      rd_en = $random;
    end
```

```
    end
    //Test both read and write
    rd_en = 1;
    wr_en = 1;
    for (i = 0; i < 10000; i=i+1) begin
      #2
      rd_en = $random;
      wr_en = $random;
      data_in = $random;
    end
    $stop;
  end
endmodule
```