3. Question 3

3.1. Design code “single port ram”

```
module single_port_Ram #(
    parameter MEM_WIDTH = 16,           /* Data in/out and memory word width */
    parameter MEM_DEPTH = 1024,         /*  Memory depth */
    parameter ADD_SIZE = 10,            /* Address size based upon the memory depth */
    parameter ADDR_PIPELINE = "FALSE",  /* If "TRUE", the address should be pipelined before writing/reading the RAM, if
                                           "FALSE" then the address input will be assigned directly to the RAM's address port */
    parameter DOUT_PIPELINE = "TRUE",   /* If "TRUE" then the data out should be pipelined, if "FALSE" then the output will be
                                           out of the RAM directly */
    parameter PARITY_ENABLE = 1         /* If the parameter value is 1 then the parity should be calculated and assigned to
                                           parity_out port, if the parameter is 0 then the parity_out port should be tied to 0 */
)(
  /*--------------Inputs-------------*/
  input  [MEM_WIDTH-1:0]din,
  input  [ADD_SIZE-1:0]addr,
  input  addr_en,     /*  enable signal for the flipflop that pipelines the address */
  input  dout_en,     /*  enable signal for the flipflop that pipelines the data out */
  input  wr_en,       /*  enable signal for writing in ram */
  input  rd_en,       /*  enable signal for reading from ram */
  input  blk_select,  /*  enable signal to the ram */
  input  clk,         /*  clock signal input */
  input  rst,         /*  active high synchronous reset */

  /*--------------outputs------------*/
  output [MEM_WIDTH-1:0]dout,  /* data out of Ram */
  output parity_out   /*  calculates the even parity on the dout bus */
);

 reg[MEM_WIDTH-1:0]mem [MEM_DEPTH-1:0];

 wire [ADD_SIZE-1:0]addr_final;  /* the address I will use to  access the location of reading or writing */
 /* Address Pipeline stage */
 generate
    if(ADDR_PIPELINE=="TRUE")
        peline_stage_mod #(.WIDTH(ADD_SIZE),  // parameter defines the width of input and output
                           .RSTTYPE("SYNC") // parameter defines the type of rst, takes SYNC or ASYNC
                )ADDR_PIPELINE_reg (.in(addr),    // [Width-1:0]input
                                    .clk(clk),    // clock input
                                    .FF_en(addr_en), // clock enable
                                    .rst(rst),    // reset signal
                                    .sel(1),    // selction of the mux
                                    .out(addr_final)     // output of the mux
                                    );
    else if(ADDR_PIPELINE=="FALSE")
        peline_stage_mod #(.WIDTH(ADD_SIZE),  // parameter defines the width of input and output
                           .RSTTYPE("SYNC") // parameter defines the type of rst, takes SYNC or ASYNC
                )ADDR_PIPELINE_reg (.in(addr),    // [Width-1:0]input
                                    .clk(clk),    // clock input
                                    .FF_en(addr_en), // clock enable
                                    .rst(rst),    // reset signal
                                    .sel(0),    // selction of the mux
                                    .out(addr_final)     // output of the mux
                                    );
 endgenerate


 reg [MEM_WIDTH-1:0]dout_from_mem;  /* the direct data output of Ram */
 always @(posedge clk ) begin
    if(rst)
        dout_from_mem <= 0;
    else begin
        if(blk_select)begin
            if(wr_en)
                mem[addr_final] <= din;
            if(rd_en)
                dout_from_mem <= mem[addr_final];
        end
    end
 end

 /* Dout Pipeline stage */
  generate
    if(DOUT_PIPELINE=="TRUE")
        peline_stage_mod #(.WIDTH(MEM_WIDTH),  // parameter defines the width of input and output
                           .RSTTYPE("SYNC") // parameter defines the type of rst, takes SYNC or ASYNC
                )Dout_PIPELINE_reg (.in(dout_from_mem),    // [Width-1:0]input
                                    .clk(clk),    // clock input
                                    .FF_en(dout_en), // clock enable
                                    .rst(rst),    // reset signal
                                    .sel(1),    // selction of the mux
                                    .out(dout)     // output of the mux
                                    );
```

```
    else if(DOUT_PIPELINE=="FALSE")
        peline_stage_mod #(.WIDTH(MEM_WIDTH),  // parameter defines the width of input and output
                           .RSTTYPE("SYNC") // parameter defines the type of rst, takes SYNC or ASYNC
                  )Dout_PIPELINE_reg (.in(dout_from_mem),   // [Width-1:0]input
                                      .clk(clk),    // clock input
                                      .FF_en(dout_en), // clock enable
                                      .rst(rst),    // reset signal
                                      .sel(0),    // selction of the mux
                                      .out(dout)     // output of the mux
                                      );
 endgenerate

 /* Calculate parity check if the PARITY_ENABLE = 1 */
 generate
    if(PARITY_ENABLE==1)
        assign parity_out = ^dout;
    else if(PARITY_ENABLE==0)
        assign parity_out = 0;
 endgenerate

endmodule //Ram
```

3.2. Pipeline_stage_module

```
module peline_stage_mod (in,clk,FF_en,rst,sel,out);
 parameter WIDTH = 18 ;       /* Width of the input and the output */
 /* type of the reset is sync or async. It takes SYNC or ASYNC */
 parameter RSTTYPE = "SYNC";  /* default reset type is synchronous */

 input  [WIDTH-1:0]in;
 input  clk,FF_en,sel,rst;
 output [WIDTH-1:0]out;

 reg [WIDTH-1:0]in_r;
 /* generate block according to RSTTYPE prameter */
 generate
    if(RSTTYPE=="SYNC")begin
        always @(posedge clk) begin
            if(rst)
                in_r <= 0;
            else begin
                if(FF_en)
                    in_r <= in;
            end
        end
    end
    else if(RSTTYPE=="ASYNC")begin
        always @(posedge clk or posedge rst) begin
            if(rst)
                in_r <= 0;
            else begin
                if(FF_en)
                    in_r <= in;
            end
        end
    end
 endgenerate
 /* mux design using assignment statement */
 assign out =(sel)? in_r : in;
endmodule

/*
//------module instantiation------
peline_stage_mod #(.WIDTH(),  // parameter defines the width of input and output
                   .RSTTYPE() // parameter defines the type of rst, takes SYNC or ASYNC
                  )module_name (.in(),   // [Width-1:0]input
                                .clk(),    // clock input
                                .FF_en(), // clock enable
                                .rst(),    // reset signal
                                .sel(),    // selction of the mux
                                .out()     // output of the mux
                                );
*/
```

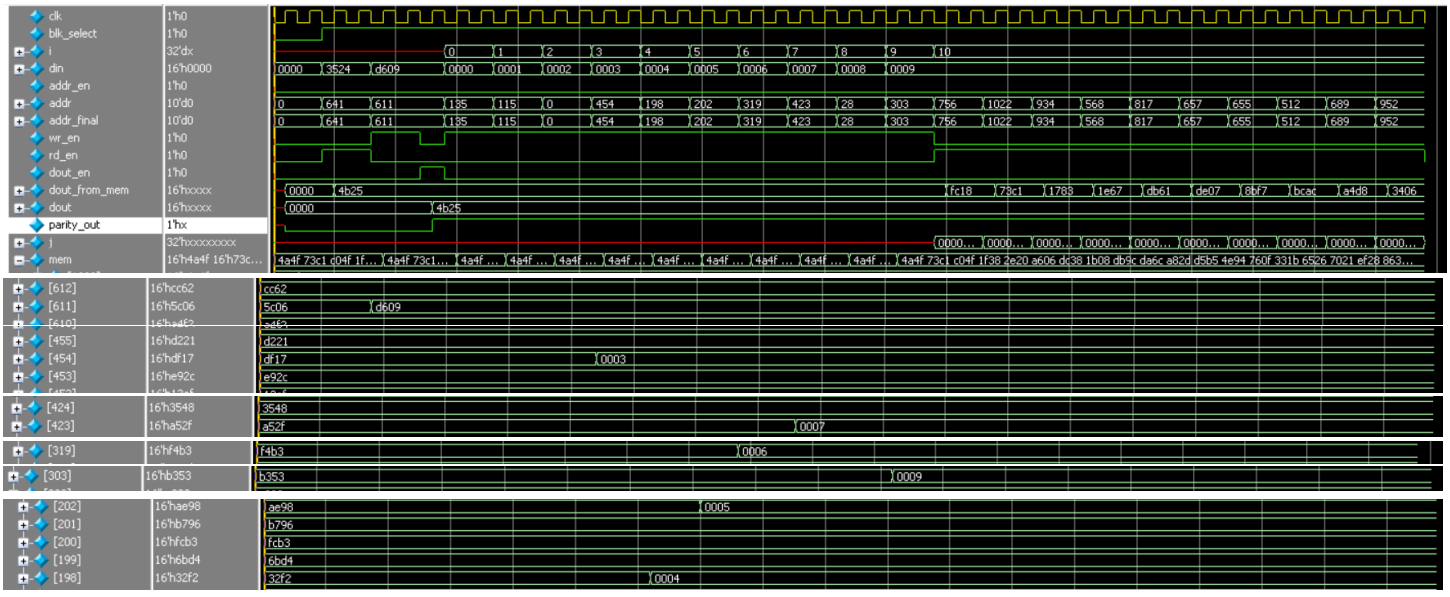
3.3. Testbench

```
module single_port_Ram_tb #(
    parameter MEM_WIDTH = 16,
    parameter MEM_DEPTH = 1024,
    parameter ADD_SIZE = 10,
    parameter ADDR_PIPELINE = "FALSE",
    parameter DOUT_PIPELINE = "TRUE",
    parameter PARITY_ENABLE = 1
) ();
  /*--------------Inputs--------------*/
  reg  [MEM_WIDTH-1:0]din;
  reg  [ADD_SIZE-1:0]addr;
  reg  addr_en;
  reg  dout_en;
  reg  wr_en;
  reg  rd_en;
  reg  blk_select;
  reg  clk;
  reg  rst;
  /*--------------outputs-------------*/
  wire [MEM_WIDTH-1:0]dout;
  wire parity_out;
  /*---Instantiation of DUT and Ref---*/
  single_port_Ram  DUT (.*);
  /*---------clock generation---------*/
  initial begin
    clk=0;
    forever begin
        #20; clk=~clk;
    end
  end
  integer i;
  integer j;
  /*--------Stimulus process---------*/
  initial begin
    $display("---START SIMULATION---");
    $readmemh("mem.dat",DUT.mem);
    /* intialize the inputs */
    rst = 0;
    addr_en = 0;
    dout_en = 0;
    wr_en = 0;
    rd_en = 0;
    blk_select = 0;
    din = 0;
    blk_select = 0;
    din = 0;
    addr = 0;
    /* enable reset signal */
    rst = 1;
    repeat(2) @(negedge clk);
    /* disable reset signal */
    rst = 0;
    /* enable blk_select */
    blk_select=1;
    /* enable read and disable write */
    rd_en = 1;      wr_en = 0;
    din =$random;
    addr=$random;
    repeat(2) @(negedge clk);
    /* disable read and enable write */
    rd_en = 0;      wr_en = 1;
    din =$random;
    addr =$random;
    repeat(2) @(negedge clk);
    wr_en = 0;
    /* Dout pipeline (if enabled) */
    dout_en = 1;
    @(negedge clk);
    dout_en = 0;
    /* Test Case 6: Write and Read multiple addresses */
    blk_select = 1;
    wr_en = 1;
    for ( i = 0; i < 10; i = i + 1) begin
        din = i;
        addr = $urandom_range(0,512);
        repeat(2) @(negedge clk);
    end
    wr_en = 0;
    rd_en = 1;
    for ( j = 0; j < 10; j = j + 1) begin
        addr = $urandom_range(512,1024);
        repeat(2) @(negedge clk);
    end
    rd_en = 0;

    $display("---END SIMULATION---");
    $stop;
  end

endmodule
```

3.4. Memory data file

[Mem.dat]

3.5. QuestaSim Waveform



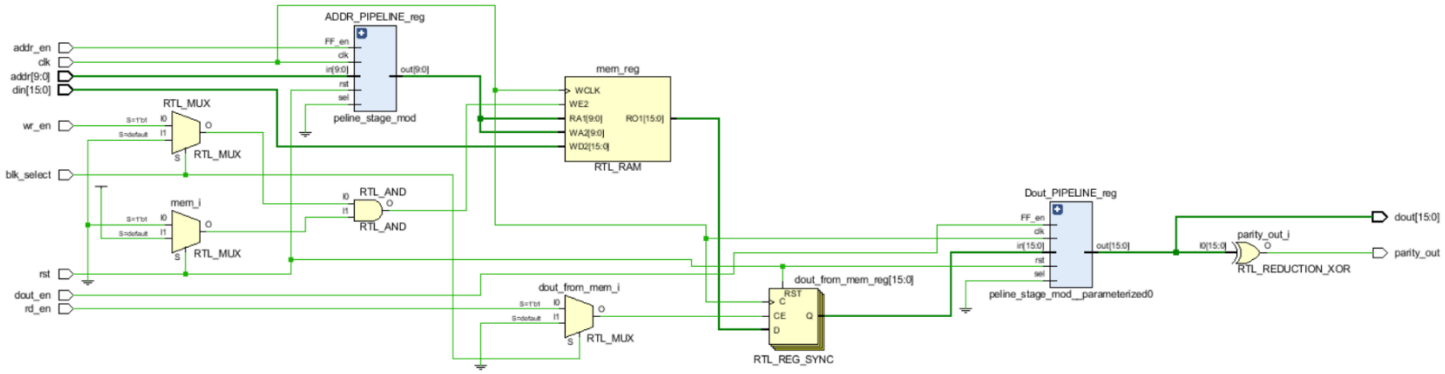
3.6. Do File

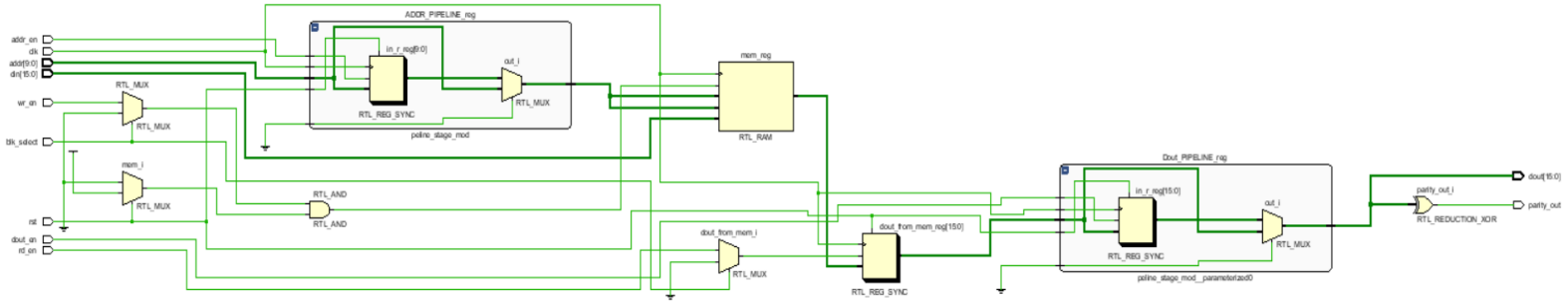
```
#create Work Folder
vlib work

#Compile files with names
vlog Single_Port_syn_Ram.v Single_Port_syn_Ram_tb.v

#simulate The TB file with module Name
vsim -voptargs=+acc work.single_port_Ram_tb

#add the variables and internal signals with specific order to notice them easily

add wave -position insertpoint  \
sim:/single_port_Ram_tb/DUT/MEM_WIDTH \
sim:/single_port_Ram_tb/DUT/MEM_DEPTH \
sim:/single_port_Ram_tb/DUT/ADD_SIZE \
sim:/single_port_Ram_tb/DUT/ADDR_PIPELINE \
sim:/single_port_Ram_tb/DUT/DOUT_PIPELINE \
sim:/single_port_Ram_tb/DUT/PARITY_ENABLE \
sim:/single_port_Ram_tb/DUT/rst \
sim:/single_port_Ram_tb/DUT/clk \
sim:/single_port_Ram_tb/DUT/blk_select \
sim:/single_port_Ram_tb/i \
sim:/single_port_Ram_tb/DUT/din \
sim:/single_port_Ram_tb/DUT/addr \
sim:/single_port_Ram_tb/DUT/addr_en \
\sim:/single_port_Ram_tb/DUT/addr_final \
sim:/single_port_Ram_tb/DUT/wr_en \
sim:/single_port_Ram_tb/DUT/rd_en \
sim:/single_port_Ram_tb/j \
sim:/single_port_Ram_tb/DUT/dout_from_mem \
sim:/single_port_Ram_tb/DUT/dout_en \
sim:/single_port_Ram_tb/DUT/dout \
sim:/single_port_Ram_tb/DUT/parity_out \
sim:/single_port_Ram_tb/DUT/mem

run -all

wave zoom full
```

3.7. Elaboration

3.7.1. Messages Tab

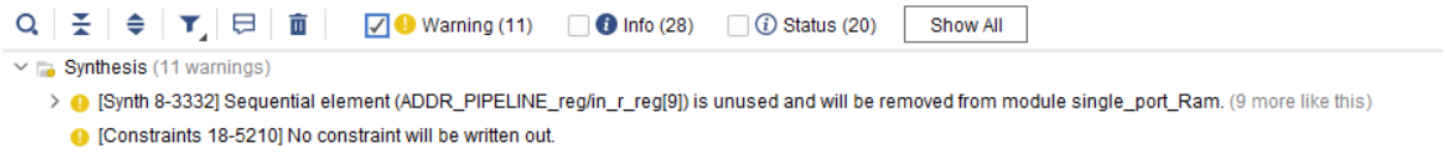
3.7.2. Schematic



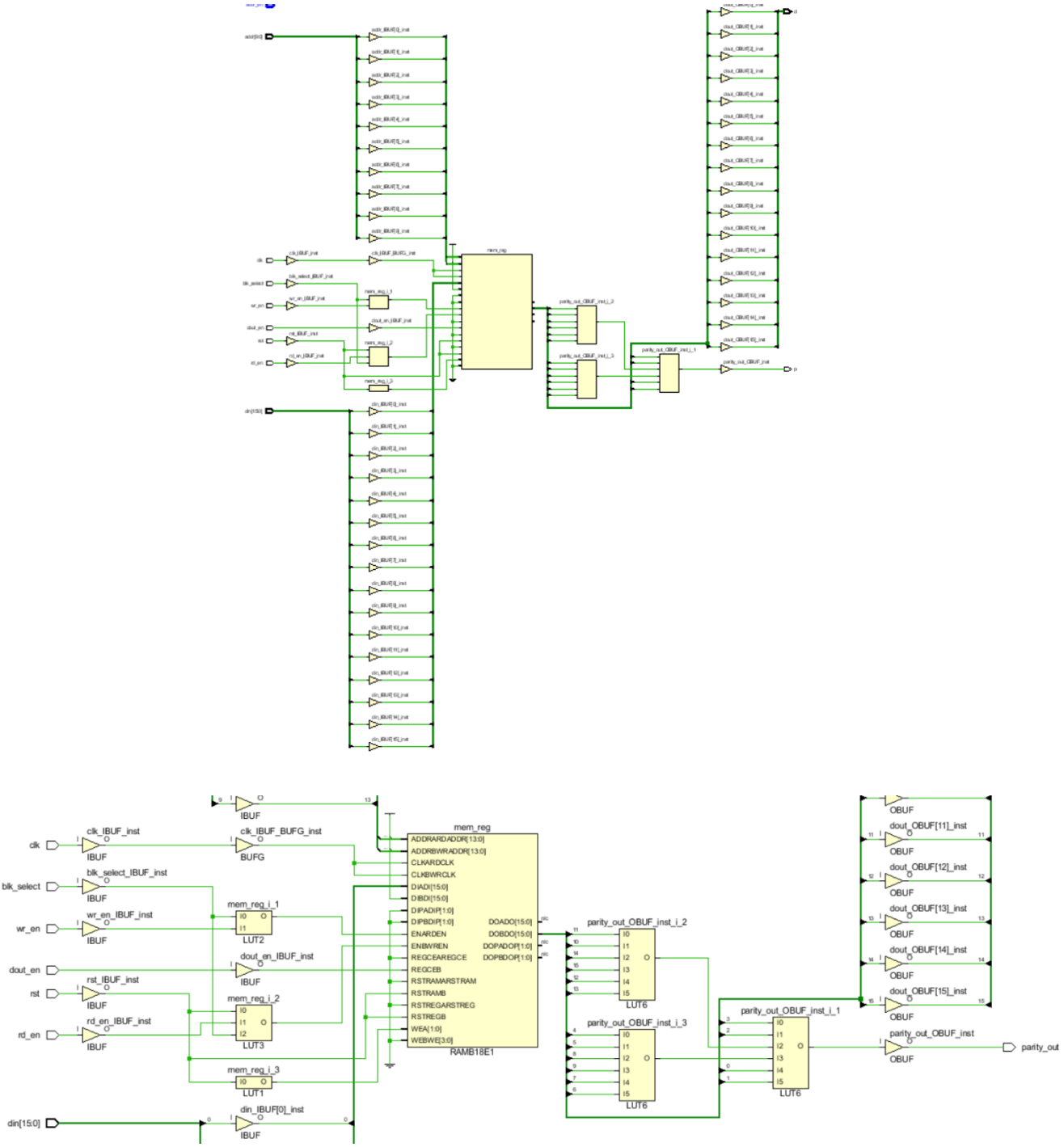


3.8. Synthesis

3.8.1. Messages Tab



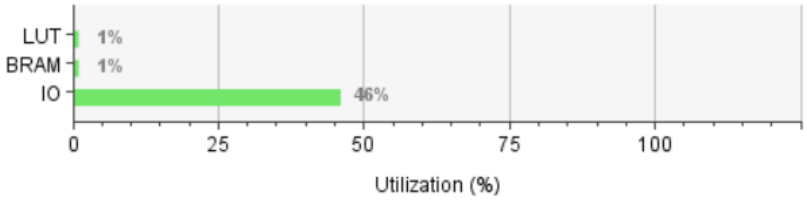
3.8.2. Schematic



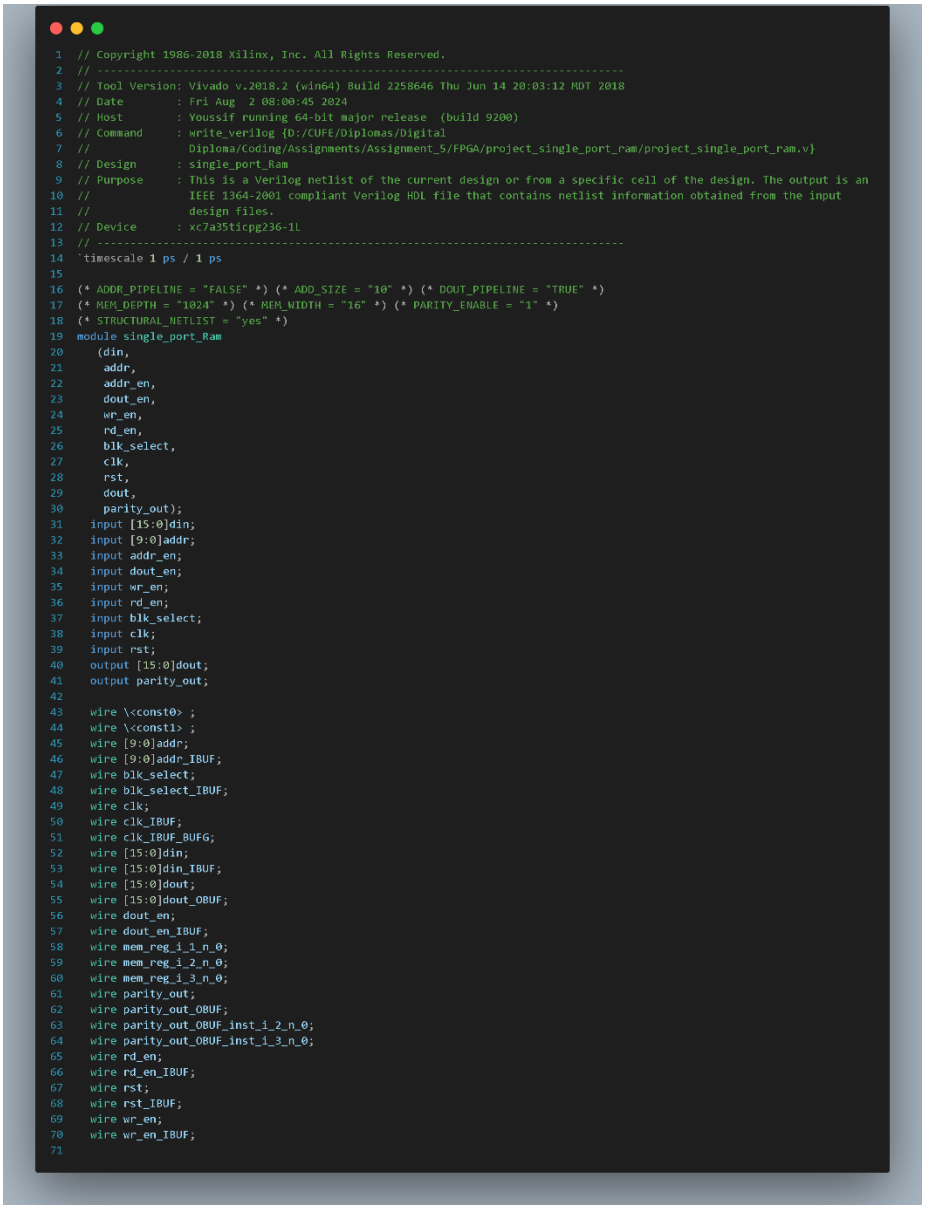
3.9. Utilization report summary

Summary

Resource	Utilization	Available	Utilization %
LUT	6	20800	0.03
BRAM	0.50	50	1.00
IO	49	106	46.23



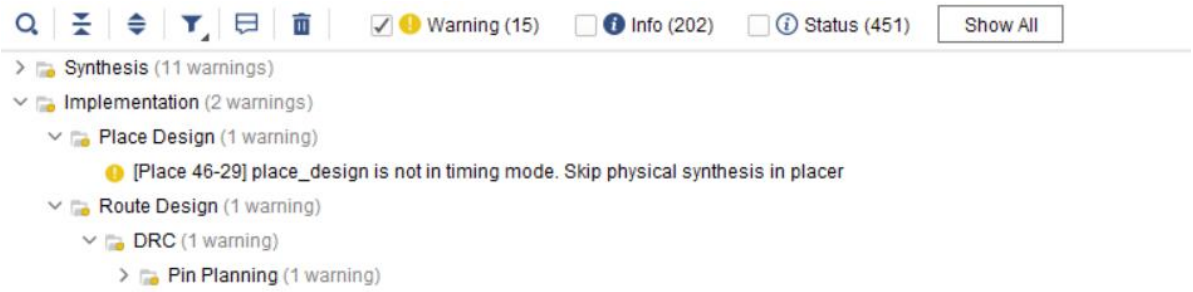
3.10. Snippet from Netlist file



```
// Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
// ----------------------------------------------------------------------------
// Tool Version: Vivado v.2018.2 (win64) Build 2258646 Thu Jun 14 20:03:12 MDT 2018
// Date        : Fri Aug  2 08:00:45 2024
// Host        : Youssif running 64-bit major release  (build 9200)
// Command     : write_verilog {D:/CUFE/Diplomas/Digital
//               Diploma/Coding/Assignments/Assignment_5/FPGA/project_single_port_ram/project_single_port_ram.v}
// Design      : single_port_Ram
// Purpose     : This is a Verilog netlist of the current design or from a specific cell of the design. The output is an
//               IEEE 1364-2001 compliant Verilog HDL file that contains netlist information obtained from the input
//               design files.
// Device      : xc7a35ticpg236-1L
// ----------------------------------------------------------------------------
`timescale 1 ps / 1 ps

(* ADDR_PIPELINE = "FALSE" *) (* ADD_SIZE = "10" *) (* DOUT_PIPELINE = "TRUE" *)
(* MEM_DEPTH = "1024" *) (* MEM_WIDTH = "16" *) (* PARITY_ENABLE = "1" *)
(* STRUCTURAL_NETLIST = "yes" *)
module single_port_Ram
   (din,
    addr,
    addr_en,
    dout_en,
    wr_en,
    rd_en,
    blk_select,
    clk,
    rst,
    dout,
    parity_out);
  input [15:0]din;
  input [9:0]addr;
  input addr_en;
  input dout_en;
  input wr_en;
  input rd_en;
  input blk_select;
  input clk;
  input rst;
  output [15:0]dout;
  output parity_out;

  wire \<const0> ;
  wire \<const1> ;
  wire [9:0]addr;
  wire [9:0]addr_IBUF;
  wire blk_select;
  wire blk_select_IBUF;
  wire clk;
  wire clk_IBUF;
  wire clk_IBUF_BUFG;
  wire [15:0]din;
  wire [15:0]din_IBUF;
  wire [15:0]dout;
  wire [15:0]dout_OBUF;
  wire dout_en;
  wire dout_en_IBUF;
  wire mem_reg_i_1_n_0;
  wire mem_reg_i_2_n_0;
  wire mem_reg_i_3_n_0;
  wire parity_out;
  wire parity_out_OBUF;
  wire parity_out_OBUF_inst_i_2_n_0;
  wire parity_out_OBUF_inst_i_3_n_0;
  wire rd_en;
  wire rd_en_IBUF;
  wire rst;
  wire rst_IBUF;
  wire wr_en;
  wire wr_en_IBUF;

```

3.11. Implementation

3.11.1. Messages Tab



3.11.2. Schematic