SPARTAN6 - DSP48A1

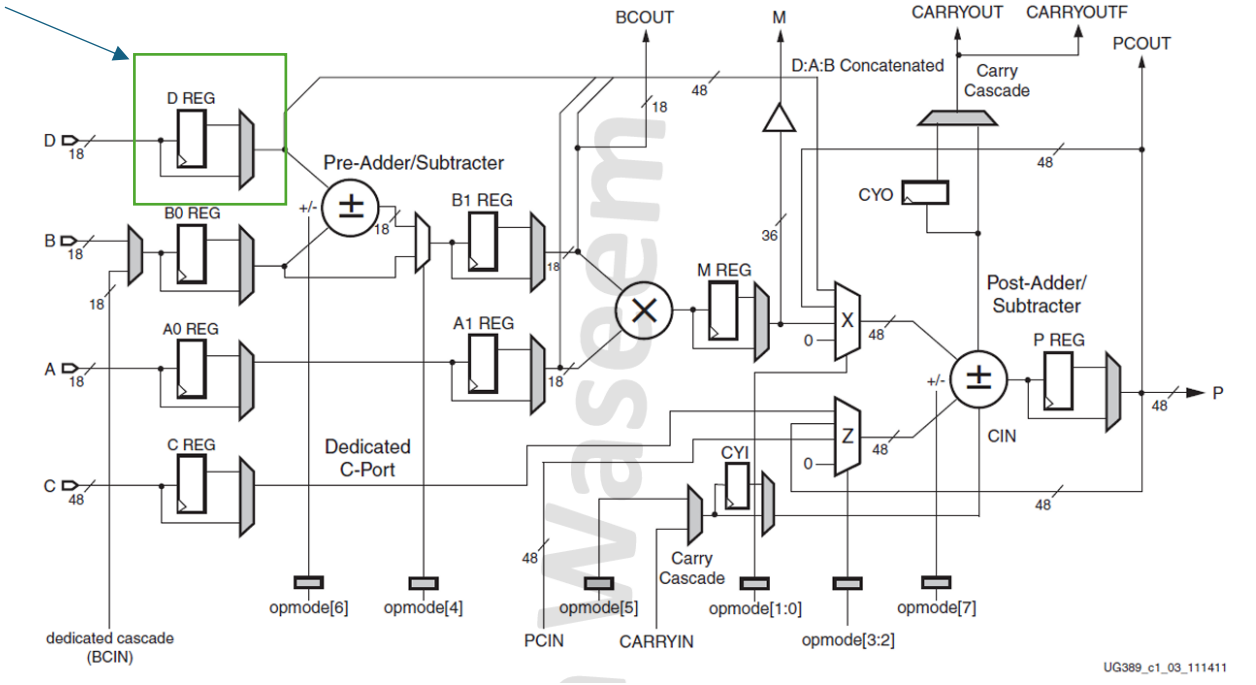
Project 1

Saeed Maher Saeed

Table of Contents

Introduction

The design of the DSP is divided into instantiation of mux register as in the square of the schematic and then implement the other parts of the design.



The file will be divided into 2 parts one for the mux register and the other part for the main design each part will have its schematic from lint and vavido to make sure it is right also its testbench with the design code.

You can find here a link with the codes and do file to run their testbenches also a text file called description to guide you each do file do what. The link

Register and MUX part:

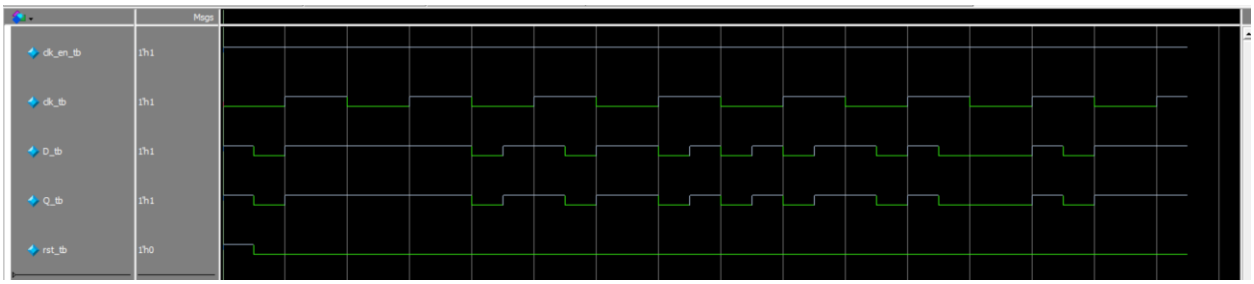
The RTL code design:

```
module REG_MUX(clk,rst,clk_en,D,Q);
parameter register = 1;
parameter TYPE = "SYNC";
parameter WIDTH = 1;
input [WIDTH-1:0]D;
input clk,rst,clk_en;
output reg [WIDTH-1:0]Q;
generate
  if(register == 1) begin
        if(TYPE == "SYNC") begin
        always @(posedge clk) begin
            if(rst) begin
              Q<=0;
            end
            else if(clk_en) begin
              Q<=D;
            end
        end
        end
      else begin
        always @(posedge clk or posedge rst) begin
            if (rst) begin
              Q<=0;
            end
            else begin
              if(clk_en) begin
                Q<=D;
              end
            end
            end
        end
  end
  else begin
    always @(*) begin
      Q=D;
    end
  end
endgenerate
endmodule
```

The testbench code with waveform:

For register = 0 & TYPE = “SYNC”:

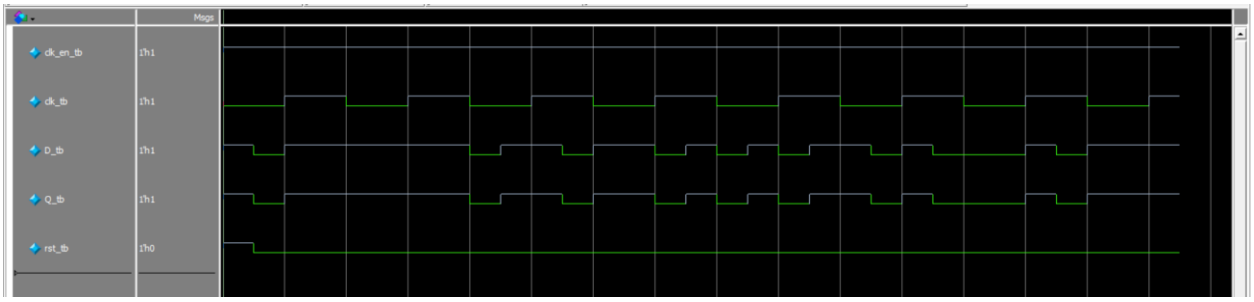
```
module REG_MUX_tb1();
reg D_tb,clk_tb,rst_tb,clk_en_tb;
wire Q_tb;

REG_MUX #(.register(0), .TYPE("SYNC")) dut (clk_tb,rst_tb,clk_en_tb,D_tb,Q_tb);

initial begin
    clk_tb=0;
    forever begin
        #10
        clk_tb = ~clk_tb;
    end
end

initial begin
    rst_tb=1;
    D_tb=1;
    clk_en_tb=1;
    #5

    rst_tb=0;
    repeat (30) begin
      D_tb=$random;
      #5;
    end

    $stop;
end

endmodule
```



since the reset signal doesn't affect it and also the output is changing with D immediately without waiting the clock then it is passed.

For register = 0 & TYPE = "ASYNC"":

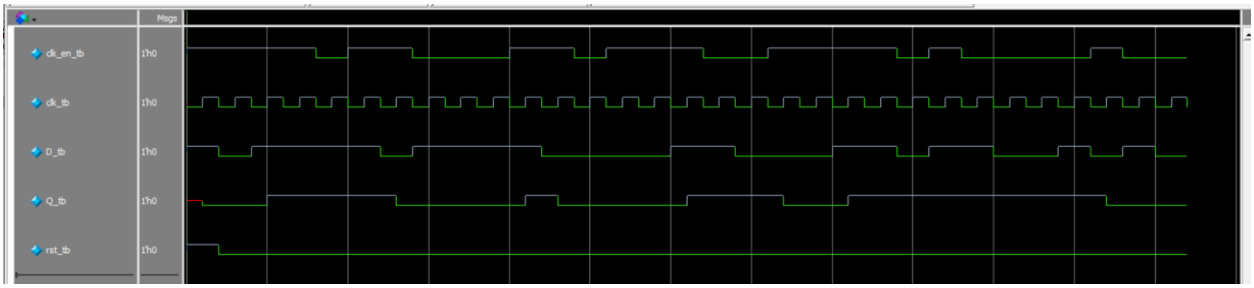
```
module REG_MUX_tb2();
reg D_tb,clk_tb,rst_tb;
wire Q_tb;

REG_MUX #(.register(0), .TYPE("ASYNC")) dut (clk_tb,rst_tb,D_tb,Q_tb);

initial begin
    clk_tb=0;
    forever begin
        #10
        clk_tb = ~clk_tb;
    end
end

initial begin
    rst_tb=1;
    D_tb=1;
    #5

    rst_tb=0;
    repeat (30) begin
      D_tb=$random;
      #5;
    end

    $stop;
end

endmodule
```



since the reset signal doesn't affect it and also the output is changing with D immediately without waiting the clock then it is passed.

For register = 1 & TYPE = "SYNC":

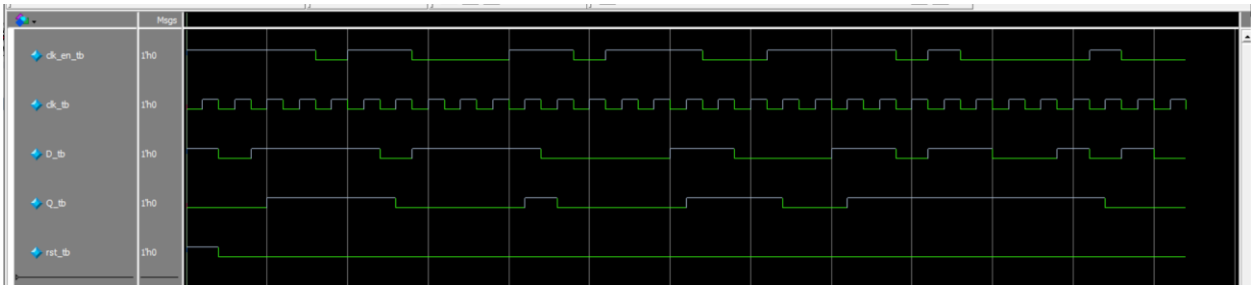
```
module REG_MUX_tb3();
reg D_tb,clk_tb,rst_tb,clk_en_tb;
wire Q_tb;

REG_MUX #(.register(1), .TYPE("SYNC")) dut (clk_tb,rst_tb,clk_en_tb,D_tb,Q_tb);

initial begin
    clk_tb=0;
    forever begin
        #10
        clk_tb = ~clk_tb;
    end
end

initial begin
    rst_tb=1;
    D_tb=1;
    clk_en_tb=1;
    @(negedge clk_tb);

    rst_tb=0;
    repeat (30) begin
      D_tb=$random;
      clk_en_tb=$random;
      @(negedge clk_tb);
    end

    $stop;
end

endmodule
```



Since that the output appears at the edge of the clock when the clock enable signal is high and that the reset signal waits for the clock and not reset the output immediately then it is passed.

For register = 1 & TYPE = “ASYNC”:

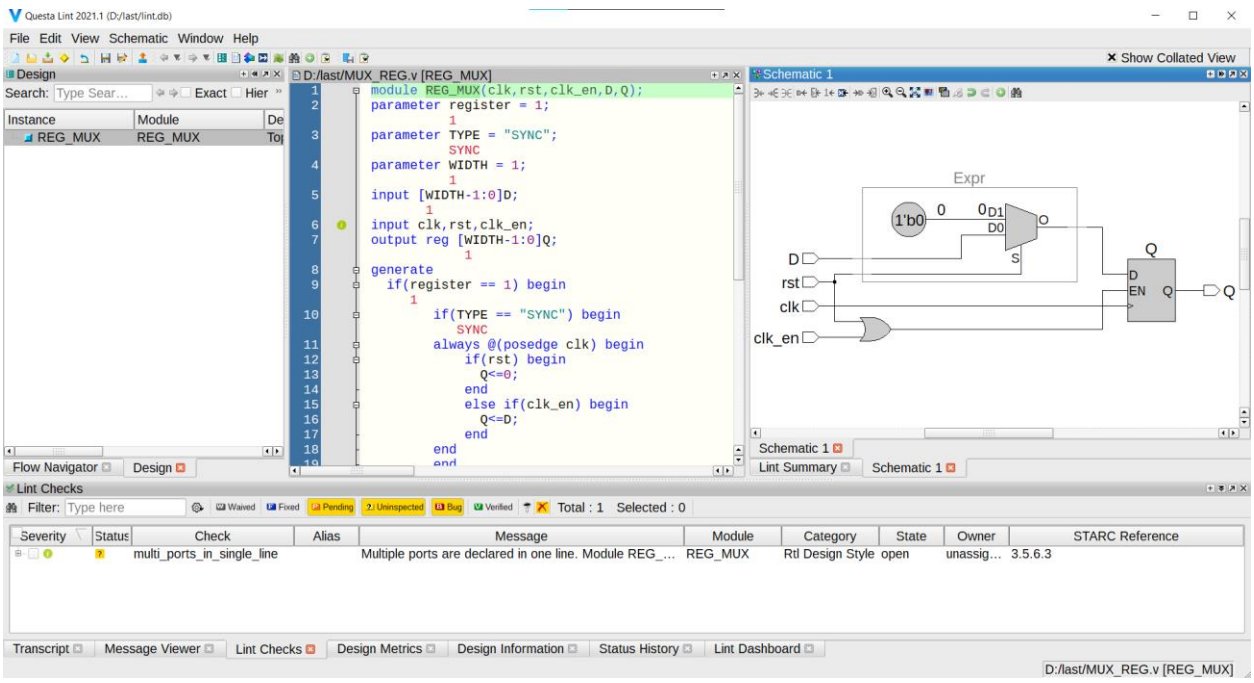
```
module REG_MUX_tb4();
reg D_tb,clk_tb,rst_tb,clk_en_tb;
wire Q_tb;
REG_MUX #(.register(1), .TYPE("ASYNC")) dut (clk_tb,rst_tb,clk_en_tb,D_tb,Q_tb);

initial begin
    clk_tb=0;
    forever begin
        #10
        clk_tb = ~clk_tb;
    end
end

initial begin
    rst_tb=1;
    D_tb=1;
    clk_en_tb=1;
    @(negedge clk_tb);

    rst_tb=0;
    repeat (30) begin
      D_tb=$random;
      clk_en_tb=$random;
      @(negedge clk_tb);
    end
    $stop;
end
endmodule
```

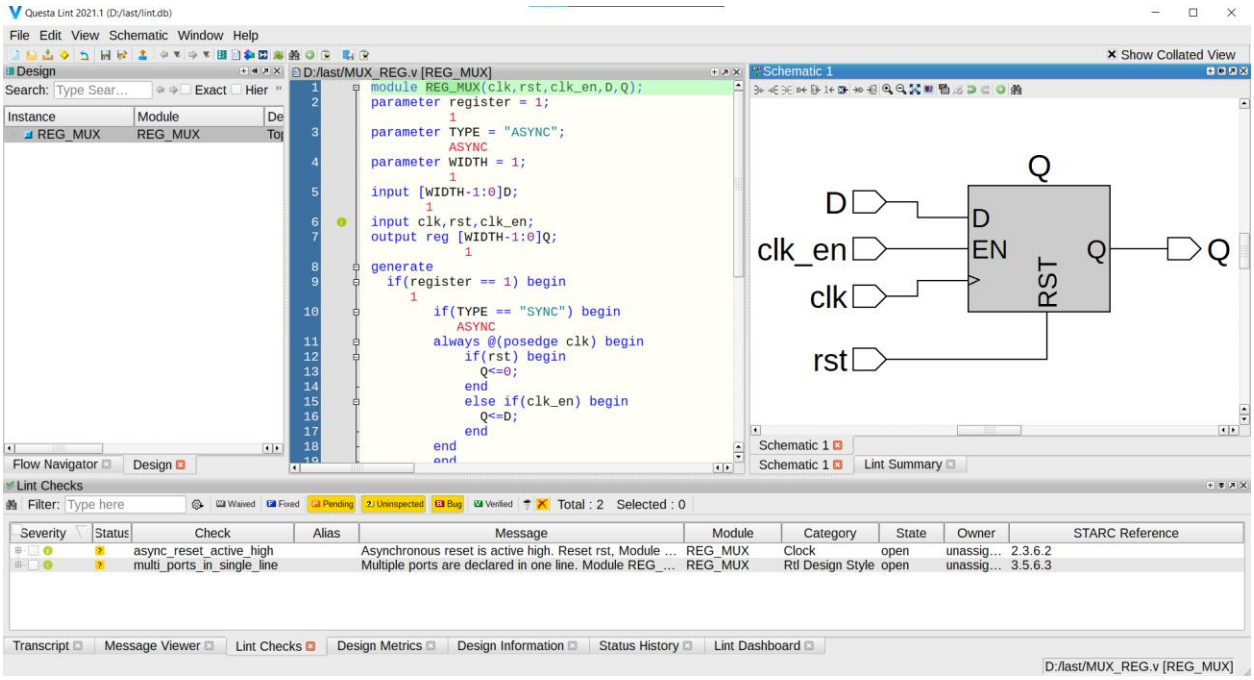


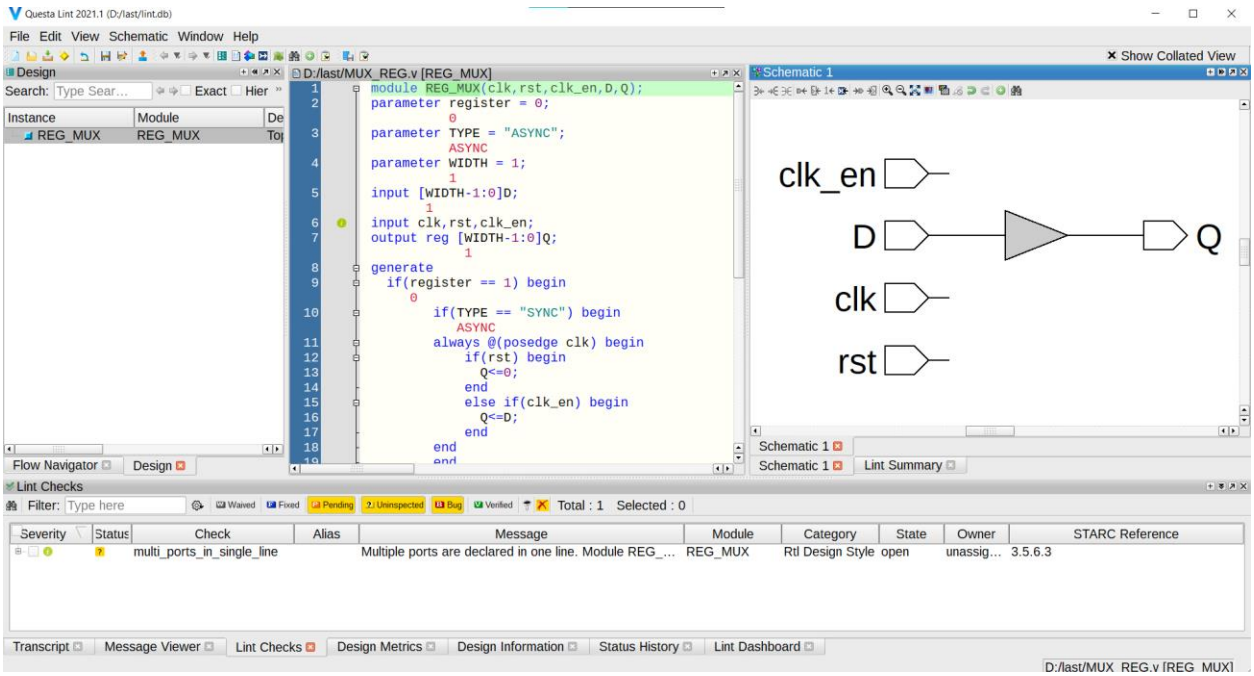
Since that the output appears at the edge of the clock and that the reset signal doesn't wait for the clock and resets the output immediately then it is passed.

The lint tool:

In this part, it is check that this module is synthesizable or not also if it shows the correct schematic.







From the above snippets, it appears that the schematic is right and synthesizable.

Main DSB part:

The RTL code design:

```
module
DSB(A,B,C,D,BCIN,CARRYIN,M,P,CARRYOUT,CARRYOUTF,clk,opmode,CEA,CEB,CEC,CECARRYIN,
CED,CEM,CEOPMODE,CEP,RSTA,RSTB,RSTC,RSTCARRYIN,RSTD,RSTM,RSTOPMODE,RSTP,BCOUT,PCO
UT,PCIN);
// parameters
parameter A0REG = 0;
parameter A1REG = 1;
parameter B0REG = 0;
parameter B1REG = 1;
parameter CREG = 1;
parameter DREG = 1;
parameter MREG = 1;
parameter PREG = 1;
parameter CARRYINREG = 1;
parameter CARRYOUTREG = 1;
parameter OPMODEREG = 1;
parameter CARRYINSEL = "OPMODE5";
parameter B_INPUT  = "DIRECT";
parameter RSTTYPE = "SYNC";
// data ports
input [17:0]A,B,D,BCIN;
input [47:0]C;
input CARRYIN;
output [35:0]M;
output [47:0]P;
output CARRYOUT,CARRYOUTF;
// control input ports
input clk;
input [7:0]opmode;
// clock enable input ports
input CEA,CEB,CEC,CECARRYIN,CED,CEM,CEOPMODE,CEP;
// Reset Input Ports
input RSTA,RSTB,RSTC,RSTCARRYIN,RSTD,RSTM,RSTOPMODE,RSTP;
// cascade ports
output [17:0]BCOUT;
output [47:0]PCOUT;
input [47:0]PCIN;
// internal wires connection
wire [17:0] B0_reg_in;
wire[7:0]OPMODEREG_out;
wire [17:0] A0REG_out,B0REG_out,DREG_out;
```

```
wire [47:0] CREG_out;
wire [17:0] A1REG_out;
wire [17:0]pre_AS_result; // Pre adder/subtractor result
wire [17:0] B1_REG_in,B1REG_out;
wire [35:0]mult,MREG_out;
wire CYI_in,CYI_REG_out;
reg [47:0]X_MUX,Z_MUX;
wire [47:0]post_AS_result; // post adder/subtractor result
wire CYO_in;

assign B0_reg_in = (B_INPUT == "DIRECT")? B:
                   (B_INPUT == "CASCADE")? BCIN:0;

 REG_MUX #(.WIDTH(8), .register(OPMODEREG), .TYPE(RSTTYPE)) REG_OPMODE
(clk,RSTOPMODE,CEOPMODE,opmode,OPMODEREG_out);
 REG_MUX #(.WIDTH(18), .register(A0REG), .TYPE(RSTTYPE)) REG_A0
(clk,RSTA,CEA,A,A0REG_out);
 REG_MUX #(.WIDTH(18), .register(B0REG), .TYPE(RSTTYPE)) REG_B0
(clk,RSTB,CEB,B0_reg_in,B0REG_out);
 REG_MUX #(.WIDTH(48), .register(CREG), .TYPE(RSTTYPE)) REG_C
(clk,RSTC,CEC,C,CREG_out);
 REG_MUX #(.WIDTH(18), .register(DREG), .TYPE(RSTTYPE)) REG_D
(clk,RSTD,CED,D,DREG_out);
 REG_MUX #(.WIDTH(18), .register(A1REG), .TYPE(RSTTYPE)) REG_A1
(clk,RSTA,CEA,A0REG_out,A1REG_out);

assign pre_AS_result = (OPMODEREG_out[6])? (DREG_out-
B0REG_out):(DREG_out+B0REG_out);
assign B1_REG_in = (OPMODEREG_out[4])? pre_AS_result:B0REG_out;
 REG_MUX #(.WIDTH(18), .register(B1REG), .TYPE(RSTTYPE)) REG_B1
(clk,RSTB,CEB,B1_REG_in,B1REG_out);

assign mult = B1REG_out*A1REG_out;
 REG_MUX #(.WIDTH(36), .register(MREG), .TYPE(RSTTYPE)) REG_M
(clk,RSTM,CEM,mult,MREG_out);

assign CYI_in = (CARRYINSEL == "CARRYIN")? CARRYIN:
                   (CARRYINSEL == "OPMODE5")? OPMODEREG_out[5]:0;

 REG_MUX #(.WIDTH(1), .register(CARRYINREG), .TYPE(RSTTYPE)) REG_CYI
(clk,RSTCARRYIN,CECARRYIN,CYI_in,CYI_REG_out);

always @(*) begin
    case (OPMODEREG_out[1:0])
        0: X_MUX = 0;
```

```
        1: X_MUX = {12'b0,MREG_out};
        2: X_MUX = P;
        3: X_MUX = {DREG_out[11:0],A1REG_out,B1REG_out};
    endcase
end

always @(*) begin
    case (OPMODEREG_out[3:2])
        0: Z_MUX = 0;
        1: Z_MUX = PCIN;
        2: Z_MUX = P;
        3: Z_MUX = CREG_out;
    endcase
end

assign M = MREG_out;

assign {CYO_in,post_AS_result} = (OPMODEREG_out[7])? (Z_MUX-
(X_MUX+CYI_REG_out)):(Z_MUX+X_MUX+CYI_REG_out);

 REG_MUX #(.WIDTH(1), .register(CARRYOUTREG), .TYPE(RSTTYPE)) REG_CYO
(clk,RSTCARRYIN,CECARRYIN,CYO_in,CARRYOUT);
 REG_MUX #(.WIDTH(48), .register(PREG), .TYPE(RSTTYPE)) REG_P
(clk,RSTP,CEP,post_AS_result,P);

assign PCOUT=P;
assign CARRYOUTF=CARRYOUT;
assign BCOUT=B1REG_out;
endmodule
```

The testbench code with waveform:

➢ Code:

```
module DSP_tb();
reg [17:0]A_tb,B_tb,D_tb,BCIN_tb;
reg [47:0]C_tb;
reg CARRYIN_tb;
wire [35:0]M_tb;
wire [47:0]P_tb;
wire CARRYOUT_tb,CARRYOUTF_tb;
reg clk_tb;
reg [7:0]opmode_tb;
reg CEA_tb,CEB_tb,CEC_tb,CECARRYIN_tb,CED_tb,CEM_tb,CEOPMODE_tb,CEP_tb;
```

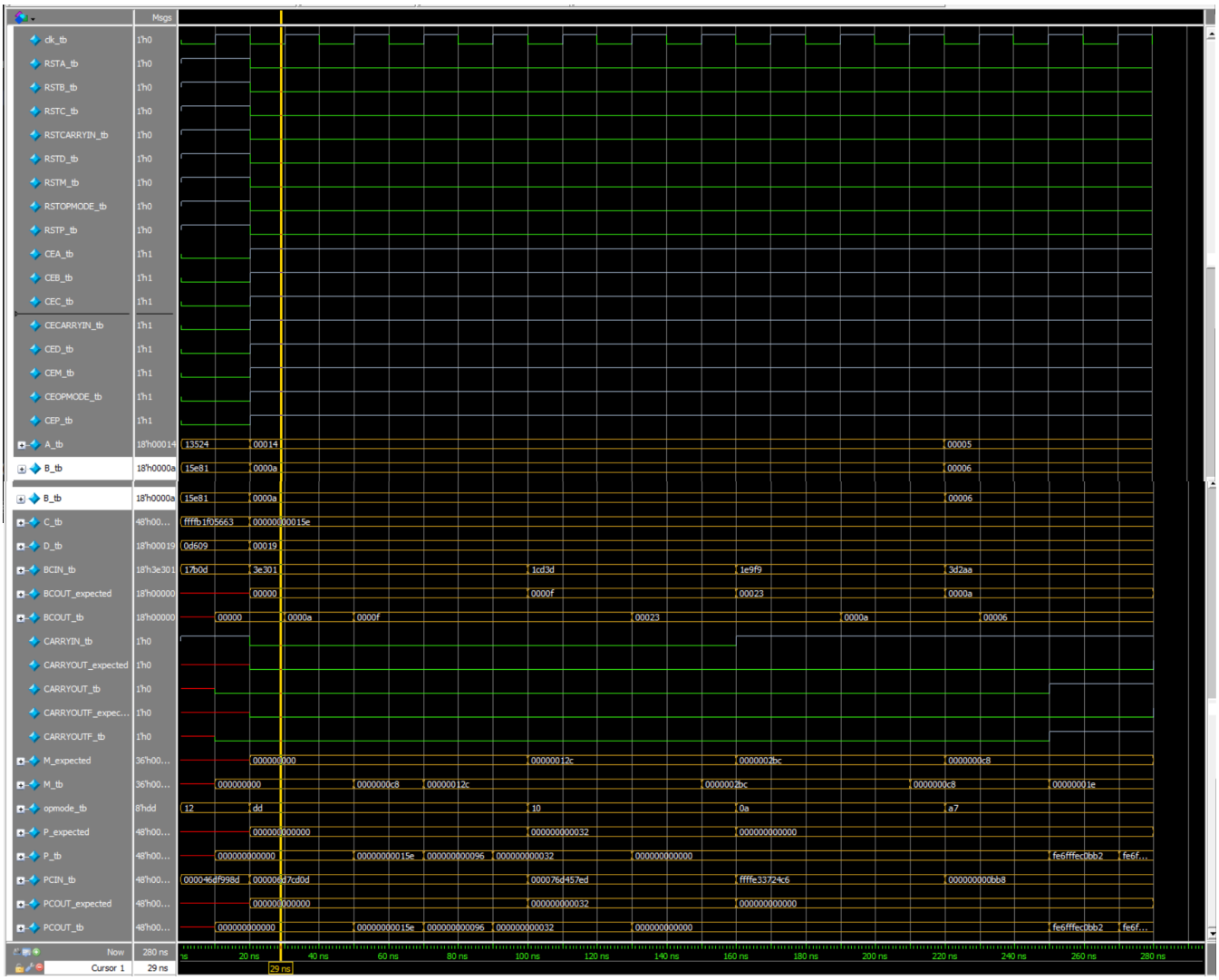
```
reg RSTA_tb,RSTB_tb,RSTC_tb,RSTCARRYIN_tb,RSTD_tb,RSTM_tb,RSTOPMODE_tb,RSTP_tb;
wire [17:0]BCOUT_tb;
wire [47:0]PCOUT_tb;
reg [47:0]PCIN_tb;
// self checking regs
reg [35:0]M_expected;
reg [47:0]P_expected;
reg [17:0]BCOUT_expected;
reg [47:0]PCOUT_expected;
reg CARRYOUT_expected,CARRYOUTF_expected;

DSP dut
(A_tb,B_tb,C_tb,D_tb,BCIN_tb,CARRYIN_tb,M_tb,P_tb,CARRYOUT_tb,CARRYOUTF_tb,clk_tb
,opmode_tb,CEA_tb,CEB_tb,CEC_tb,CECARRYIN_tb,CED_tb,CEM_tb,CEOPMODE_tb,CEP_tb,RST
A_tb,RSTB_tb,RSTC_tb,RSTCARRYIN_tb,RSTD_tb,RSTM_tb,RSTOPMODE_tb,RSTP_tb,BCOUT_tb,
PCOUT_tb,PCIN_tb);

initial begin
    clk_tb=0;
    forever begin
        #10
        clk_tb = ~clk_tb;
    end
end

initial begin
    // reset check
    RSTA_tb=1;
    RSTB_tb=1;
    RSTC_tb=1;
    RSTCARRYIN_tb=1;RSTD_tb=1;
    RSTM_tb=1;
    RSTOPMODE_tb=1;
    RSTP_tb=1;
    A_tb=$random;
    B_tb=$random;
    D_tb=$random;
    C_tb=$random;
    BCIN_tb=$random;
    PCIN_tb=$random;
    CARRYIN_tb=$random;
    opmode_tb=$random;
    CEA_tb=0;
    CEB_tb=0;
    CEC_tb=0;
```

```
    CECARRYIN_tb=0;
    CED_tb=0;
    CEM_tb=0;
    CEOPMODE_tb=0;
    CEP_tb=0;
    @(negedge clk_tb);
    M_expected=0;
    P_expected=0;
    BCOUT_expected=0;
    PCOUT_expected=0;
    CARRYOUT_expected=0;
    CARRYOUTF_expected=0;
    if (M_expected != M_tb || P_expected != P_tb || BCOUT_expected != BCOUT_tb ||
PCOUT_expected != PCOUT_tb || CARRYOUT_expected != CARRYOUT_tb ||
CARRYOUTF_expected != CARRYOUTF_tb) begin
      $display("Error in reset");
      $stop;
    end
    RSTA_tb=0;
    RSTB_tb=0;
    RSTC_tb=0;
    RSTCARRYIN_tb=0;RSTD_tb=0;
    RSTM_tb=0;
    RSTOPMODE_tb=0;
    RSTP_tb=0;
    CEA_tb=1;
    CEB_tb=1;
    CEC_tb=1;
    CECARRYIN_tb=1;
    CED_tb=1;
    CEM_tb=1;
    CEOPMODE_tb=1;
    CEP_tb=1;

    // path 1 check
    A_tb=20;
    B_tb=10;
    C_tb=350;
    D_tb=25;
    BCIN_tb=$random;
    PCIN_tb=$random;
    CARRYIN_tb=$random;
    opmode_tb=8'b1101_1101;
    repeat(4) @(negedge clk_tb);
    M_expected='h12c;
```

```
    P_expected='h32;
    BCOUT_expected='hf;
    PCOUT_expected='h32;
    CARRYOUT_expected=0;
    CARRYOUTF_expected=0;
    if (M_expected != M_tb || P_expected != P_tb || BCOUT_expected != BCOUT_tb ||
PCOUT_expected != PCOUT_tb || CARRYOUT_expected != CARRYOUT_tb ||
CARRYOUTF_expected != CARRYOUTF_tb) begin
      $display("Error in path 1");
      $stop;
    end

    // check path 2
    BCIN_tb=$random;
    PCIN_tb=$random;
    CARRYIN_tb=$random;
    opmode_tb=8'b0001_0000;
    repeat(3) @(negedge clk_tb);
    M_expected='h2bc;
    P_expected='h0;
    BCOUT_expected='h23;
    PCOUT_expected='h0;
    CARRYOUT_expected=0;
    CARRYOUTF_expected=0;
    if (M_expected != M_tb || P_expected != P_tb || BCOUT_expected != BCOUT_tb ||
PCOUT_expected != PCOUT_tb || CARRYOUT_expected != CARRYOUT_tb ||
CARRYOUTF_expected != CARRYOUTF_tb) begin
      $display("Error in path 2");
      $stop;
    end

    // check path 3
    BCIN_tb=$random;
    PCIN_tb=$random;
    CARRYIN_tb=$random;
    opmode_tb=8'b0000_1010;
    repeat(3) @(negedge clk_tb);
    M_expected='hc8;
    BCOUT_expected='ha;
    if (M_expected != M_tb || P_expected != P_tb || BCOUT_expected != BCOUT_tb ||
PCOUT_expected != PCOUT_tb || CARRYOUT_expected != CARRYOUT_tb ||
CARRYOUTF_expected != CARRYOUTF_tb) begin
      $display("Error in path 3");
      $stop;
    end
```

```
    // check path 4
    A_tb=5;
    B_tb=6;
    C_tb=350;
    D_tb=25;
    BCIN_tb=$random;
    PCIN_tb=3000;
    CARRYIN_tb=$random;
    opmode_tb=8'b1010_0111;
    repeat(3) @(negedge clk_tb);
    M_expected='h1e;
    BCOUT_expected='h6;
    P_expected='hfe6fffec0bb1;
    PCOUT_expected='hfe6fffec0bb1;
    CARRYOUT_expected=1;
    CARRYOUTF_expected=1;
    if (M_expected != M_tb || P_expected != P_tb || BCOUT_expected != BCOUT_tb ||
PCOUT_expected != PCOUT_tb || CARRYOUT_expected != CARRYOUT_tb ||
CARRYOUTF_expected != CARRYOUTF_tb) begin
      $display("Error in path 4");
      $stop;
    end

$stop;
end

endmodule
```

➢ Waveform:

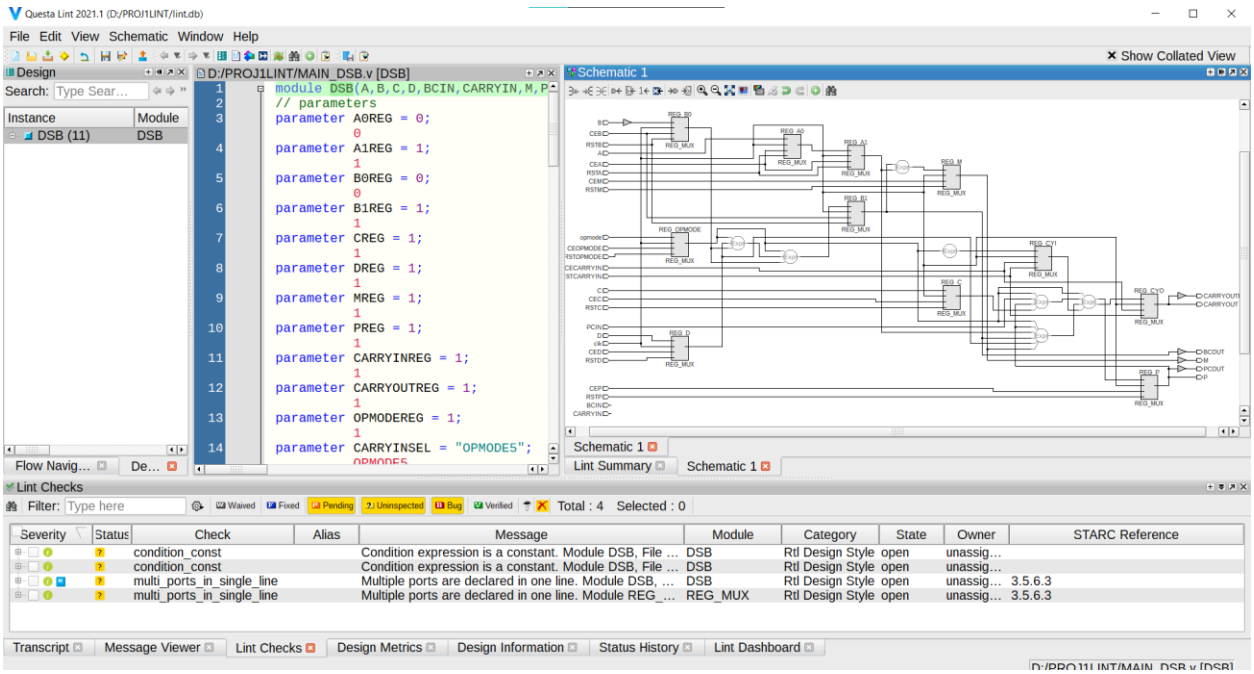


Note: the waveform is too long to be in one picture so I divided it into 2 and tried to make them under each other for easier tracing.

DO file:

```
vlib work
vlog MUX_REG.v MAIN_DSP.v DSP_tb.v
vsim -voptargs=+acc work.DSP_tb
add wave *
run -all
#quit -sim
```

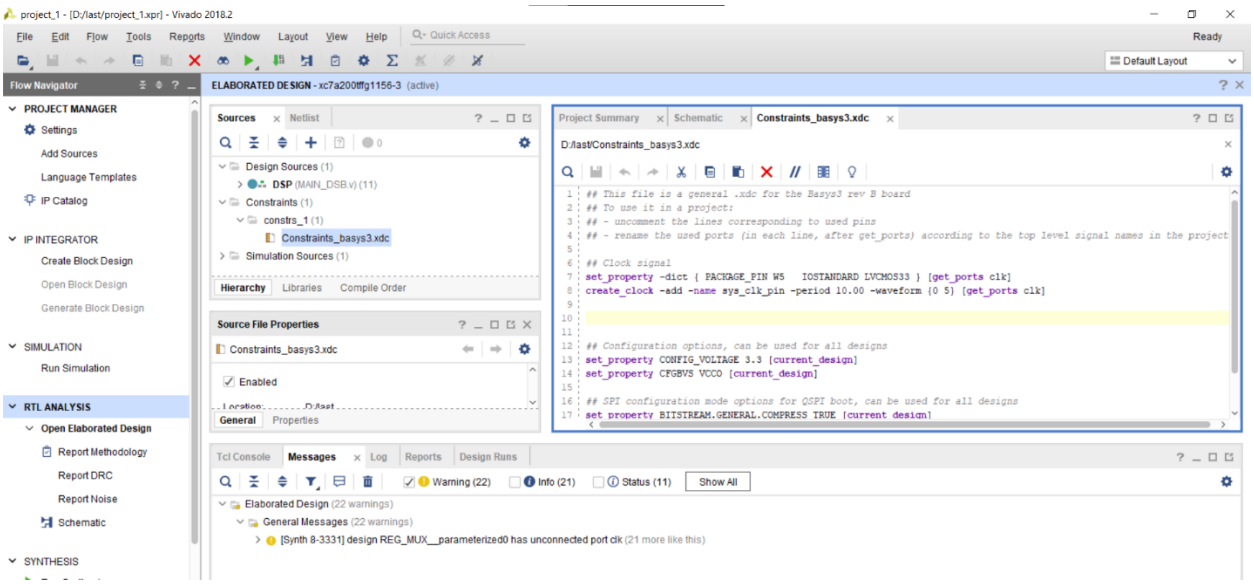
The lint tool:



The synthesis tool “Vavido”:

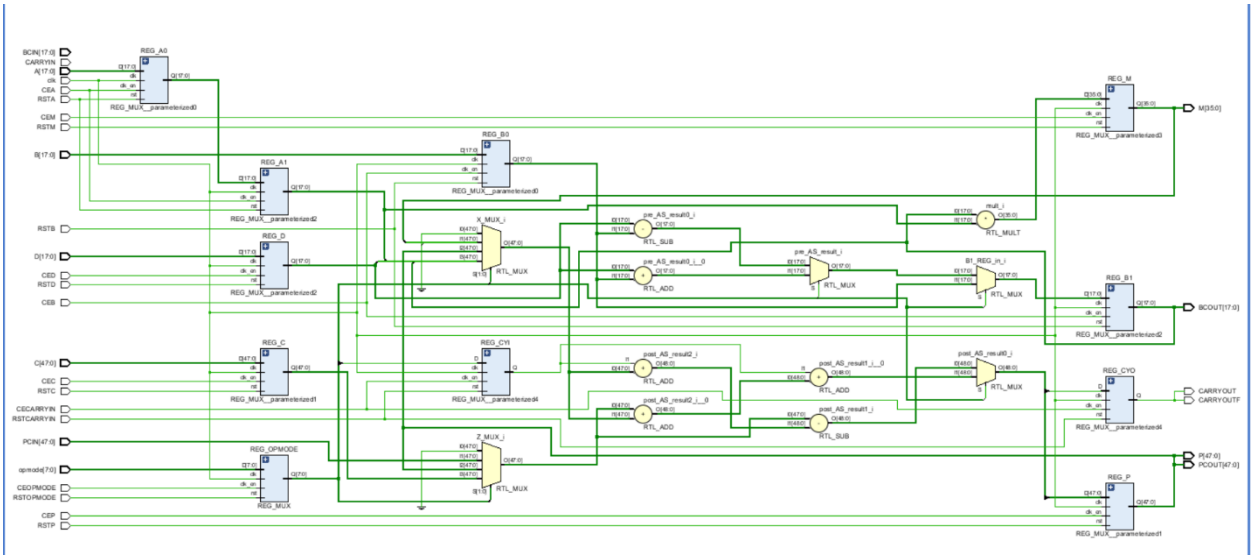
➢ Elaboration:

1. MSG of no errors or critical warning:



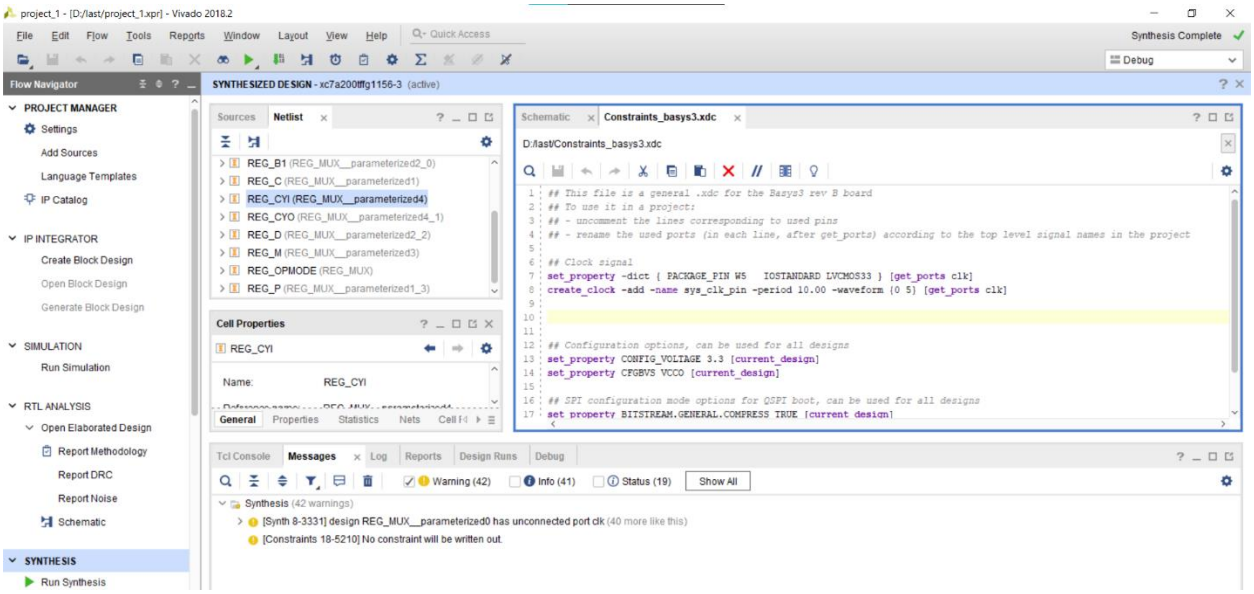
Note: the warnings is that the BCIN and the CARRYIN are not used due to the pre defined MUX_REG as in description nothing more than that.

2. Schematic:

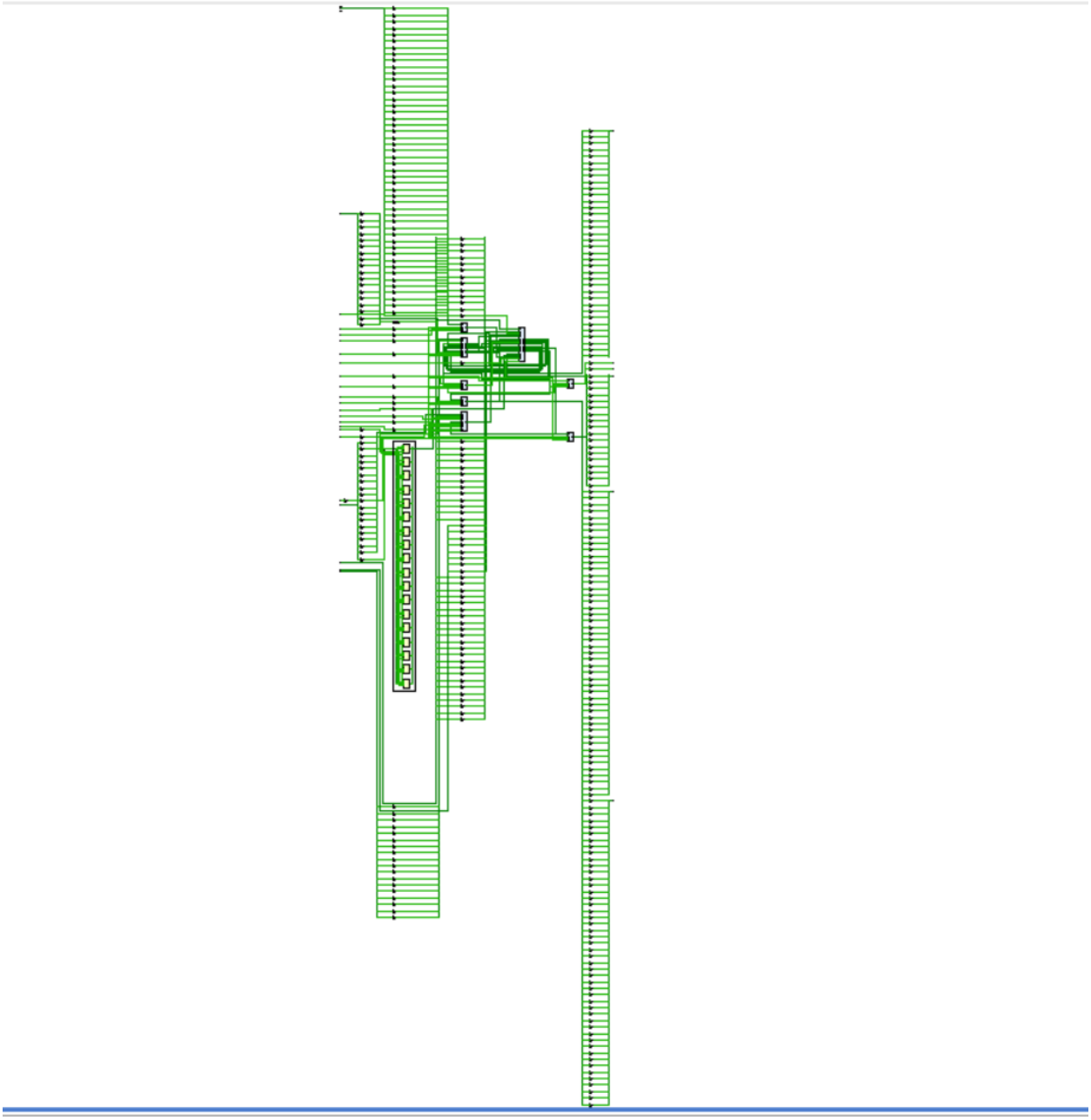


➢ Synthesis:

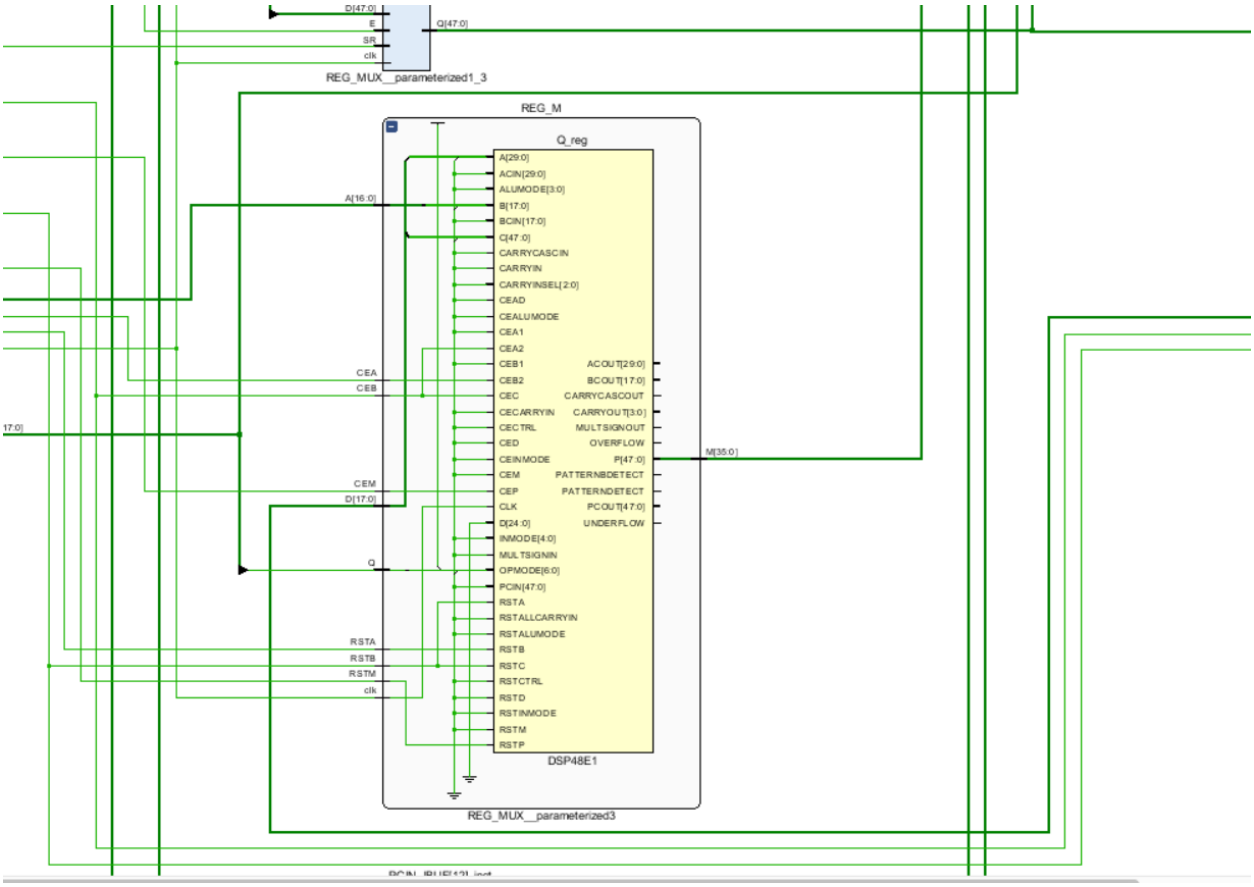
1. MSG of no errors or critical warning:



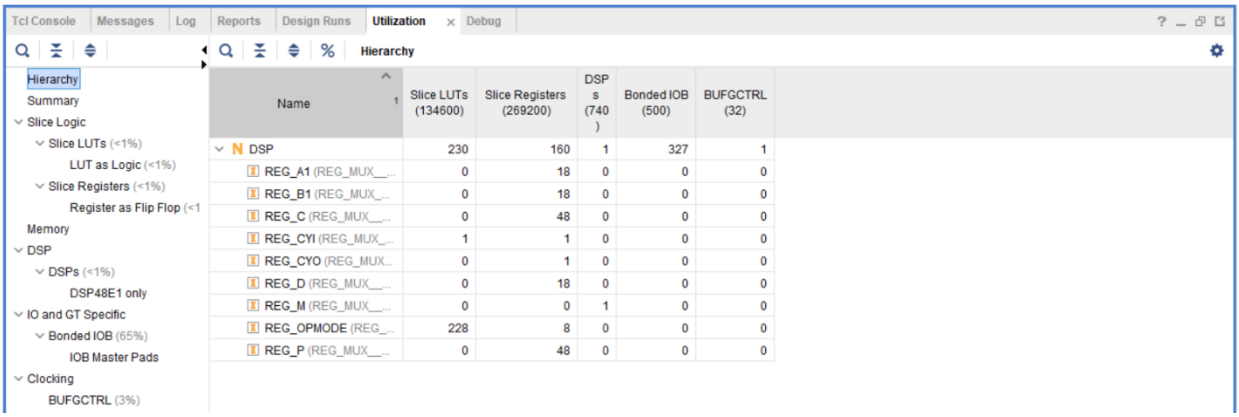
2. Schematic:



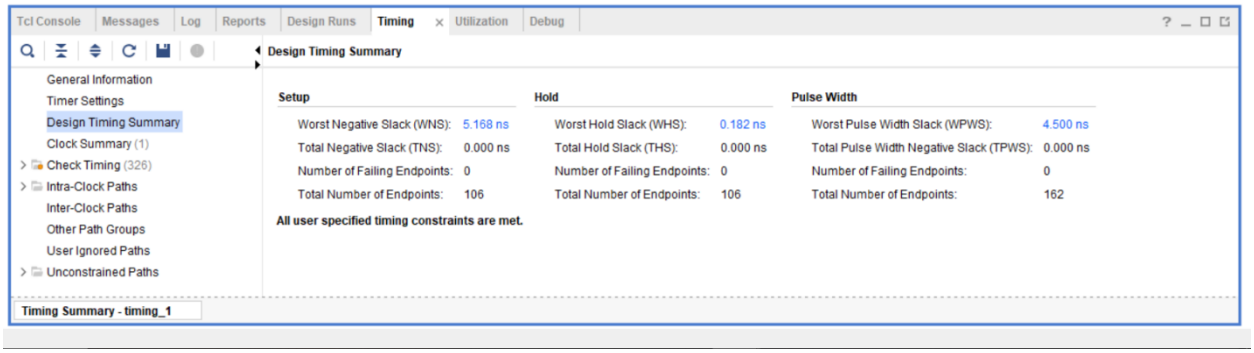
The DSP block would be found inside the REG_M as shown in the following figure:



3. Utilization report:

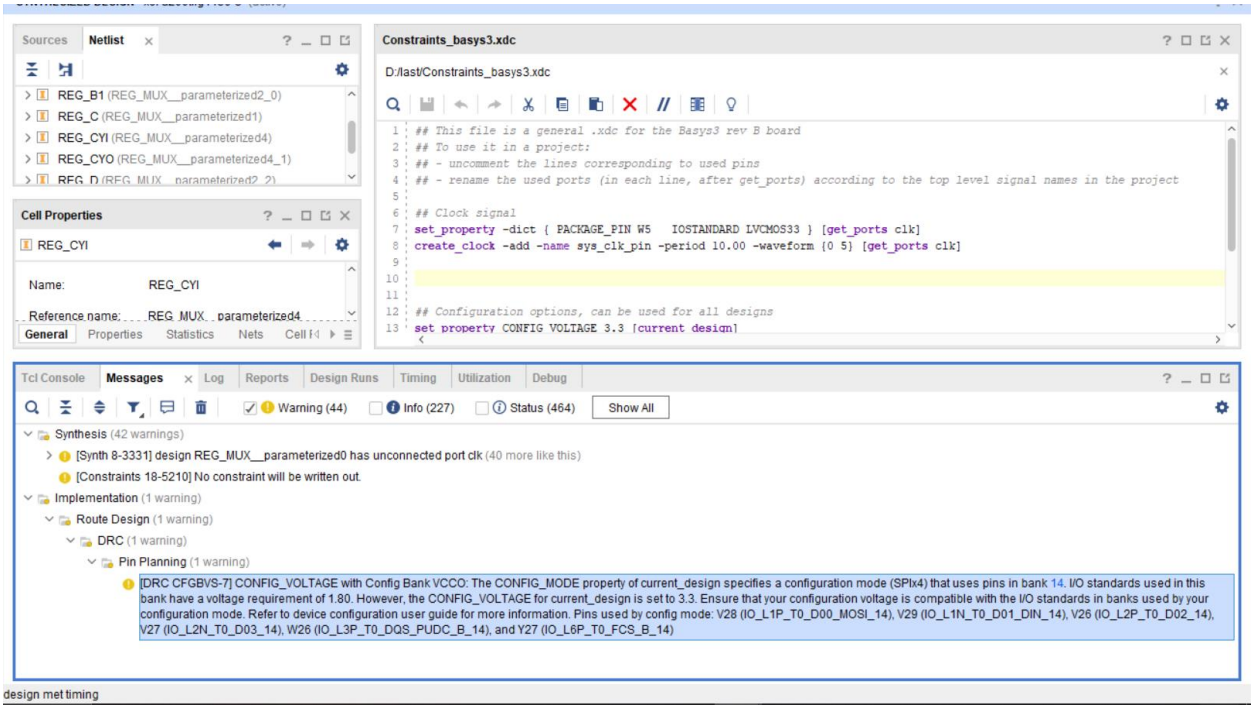


4. Timing report:

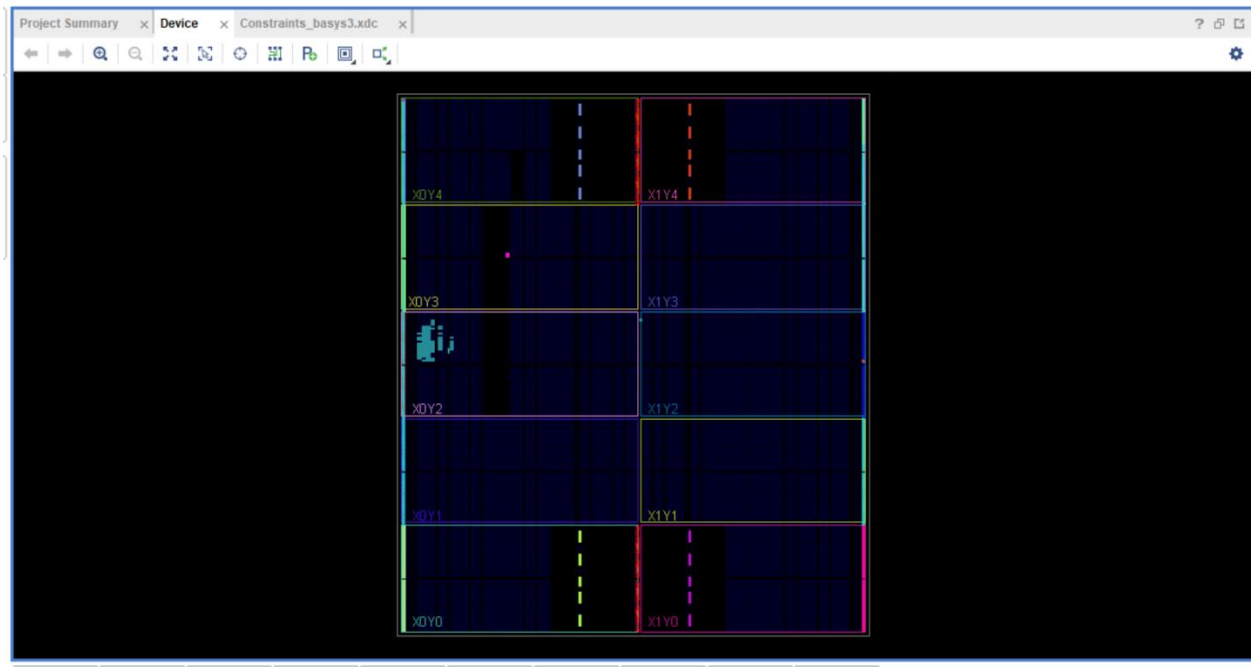


➢ Implementation:

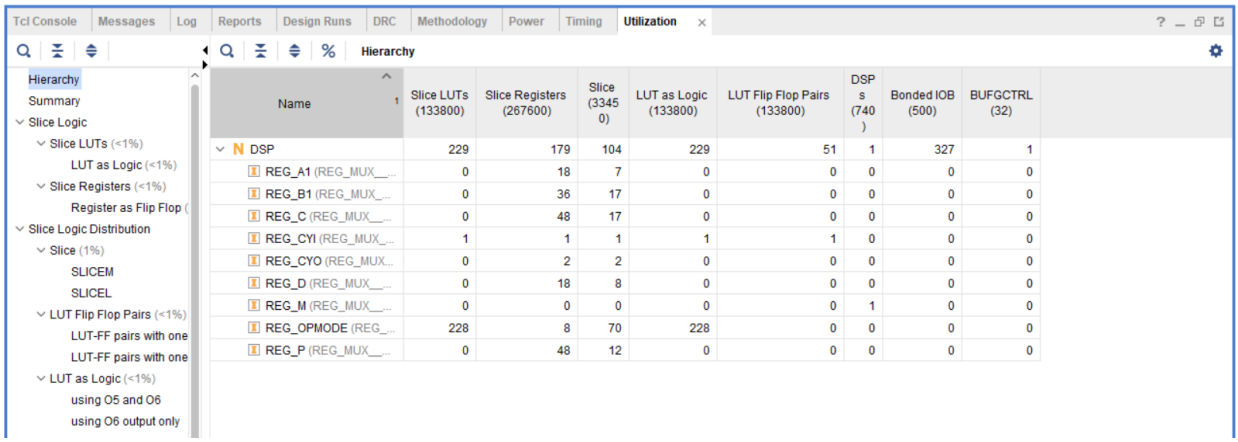
1. MSG of no errors or critical warning:



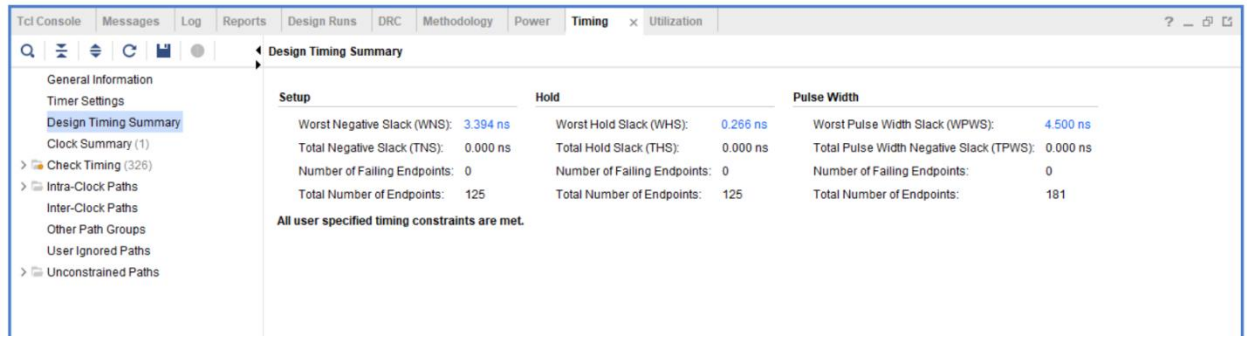
2. Device:



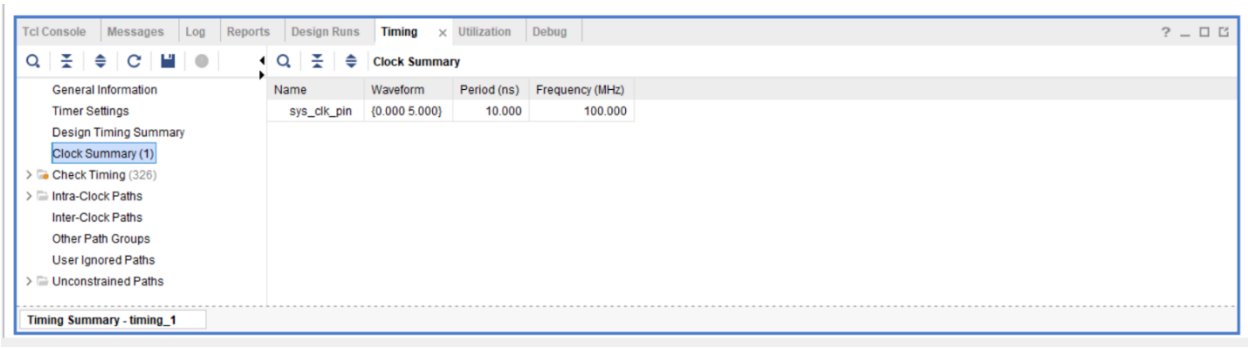
3. Utilization report:



4. Timing report:



➢ Clock:



The constraint file:

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top
level signal names in the project

## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```