Project_1

Spartan6 - DSP48A1

Supervisor: Eng. Kareem Waseem

Created by: Abdelrahman Mahmoud Abdelhamed

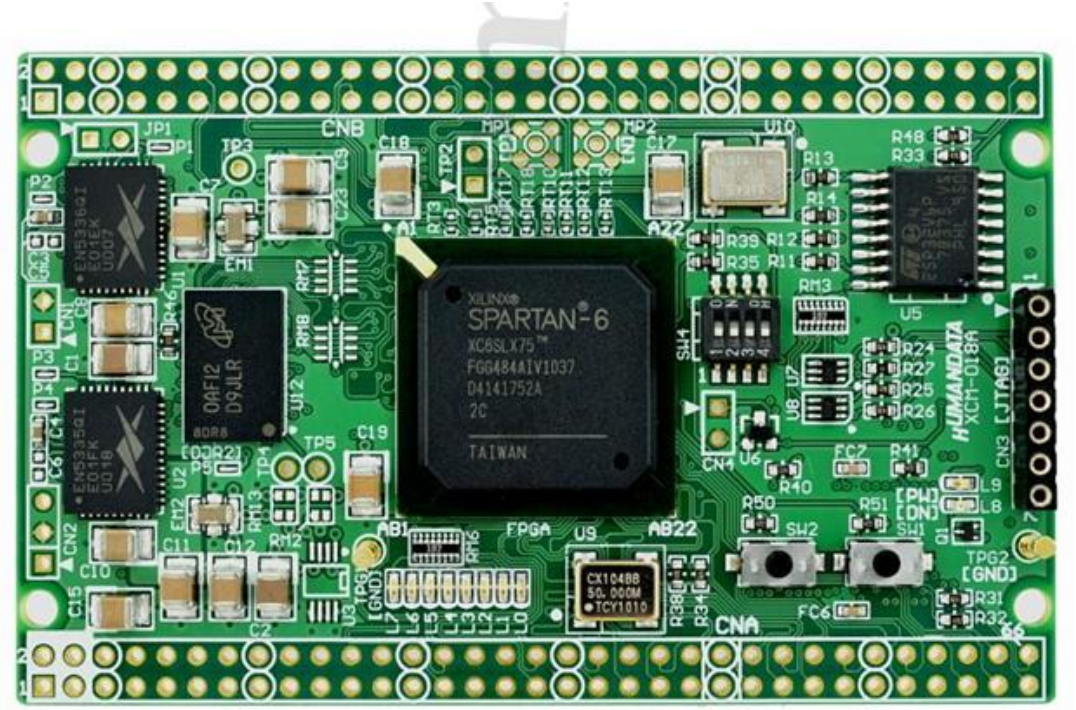


Table of Contents

RTL CODE:

Register pipeline code:

```
module register( clk, rst, cenable, reg_in, reg_out);

    parameter WIDTH = 18 ;
    parameter A0REG = 1 ;
    parameter RSTTYPE = 1 ;                                  // 1 FOR SYNC & 0 FOR ASYNC

    input clk, rst, cenable ;
    input [WIDTH-1:0] reg_in ;
    output reg  [WIDTH-1:0] reg_out ;

    reg  [WIDTH-1:0] dff ;

    generate
          if (RSTTYPE == 1'b0) begin                         // ASYNC RS
               always @(posedge clk or posedge rst) begin
                   if (rst) begin
                       dff <= 0 ;
                   end
                   else if (cenable) begin
                       dff <= reg_in ;
                   end
               end
           end
           else if (RSTTYPE == 1'b1)  begin                  // SYNC RST
                 always @(posedge clk ) begin
                       if (rst) begin
                          dff <= 0 ;
                       end
                       else if (cenable) begin
                           dff <= reg_in ;
                       end
                   end
           end
    endgenerate
           always @(*)begin
               if (A0REG) begin
                   reg_out = dff ;
               end
               else begin
                   reg_out = reg_in ;
               end

           end

endmodule
```

Code:

```
module DSP48 (
    input [17:0] A ,
    input [17:0] B ,
    input [17:0] BCIN ,
    input [17:0] D ,
    input [47:0] C ,
    input [47:0] PCIN ,
    input [7:0] opmode ,
    input CARRYIN ,
    input CLK ,
    input CEA ,
    input CEB ,
    input CEC ,
    input CED ,
    input CEM ,
    input CEP ,
    input CEOPMODE ,
    input CECARRYIN ,
    input RSTA ,
    input RSTB ,
    input RSTC ,
    input RSTD ,
    input RSTM ,
    input RSTP ,
    input RSTOPMODE ,
    input RSTCARRYIN ,

    output [17:0] BCOUT ,
    output [35:0] M ,
    output [47:0] P ,
    output [47:0] PCOUT ,
    output CARRYOUT ,
    output CARRYOUTF
);

    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 = 1 ;  // 1 FOR OPMODE[5] & 0 FOR CARRYIN
    parameter B_INPUT = 1 ;     // 1 FOR DIRECT & 0 FOR CASCADED
    parameter RSTTYPE = 1 ;     // 1 FOR SYNC & 0 FOR ASYNC


    // A pipeline 1st stage
    wire [17:0] A_stage1 ;
    register #(.WIDTH(18), .A0REG(A0REG), .RSTTYPE(RSTTYPE)) REG_A1
    (
        .clk(CLK), .rst(RSTA), .cenable(CEA), .reg_in(A), .reg_out(A_stage1)
    );


    // B pipeline 1st stage
    wire [17:0] B_stage1 ;
    wire [17:0] B_mux ;
    assign B_mux = ((B_INPUT == 1'b1) ? B : BCIN) ;
    register #(.WIDTH(18), .A0REG(B0REG), .RSTTYPE(RSTTYPE)) REG_B1
    (
        .clk(CLK), .rst(RSTB), .cenable(CEB), .reg_in(B_mux), .reg_out(B_stage1)
    );


    // C pipeline 1st stage
    wire [47:0] C_stage1 ;
    register #(.WIDTH(48), .A0REG(CREG), .RSTTYPE(RSTTYPE)) REG_C1
    (
        .clk(CLK), .rst(RSTC), .cenable(CEC), .reg_in(C), .reg_out(C_stage1)
    );


    // D pipeline 1st stage
    wire [17:0] D_stage1 ;
    register #(.WIDTH(18), .A0REG(DREG), .RSTTYPE(RSTTYPE)) REG_D1
    (
        .clk(CLK), .rst(RSTD), .cenable(CED), .reg_in(D), .reg_out(D_stage1)
    );


    // OPMODE pipeline stage
    wire [7:0] opmode_stage ;
    register #(.WIDTH(8), .A0REG(OPMODEREG), .RSTTYPE(RSTTYPE)) REG_OPMODE
    (
        .clk(CLK), .rst(RSTOPMODE), .cenable(CEOPMODE), .reg_in(opmode), .reg_out(opmode_stage)
    );


    // PRE_ADD/SUB
    wire [17:0] pre_addsub_result ;
    assign pre_addsub_result = ((opmode[6] == 0) ? (D_stage1 + B_stage1) : (D_stage1 - B_stage1)) ;

```

```
// A pipeline 2nd stage
    wire [17:0] A_stage2 ;
    register #(.WIDTH(18), .A0REG(A1REG), .RSTTYPE(RSTTYPE)) REG_A2
    (
        .clk(CLK), .rst(RSTA), .cenable(CEA), .reg_in(A_stage1), .reg_out(A_stage2)
    );


    // B pipeline 2nd stage
    wire [17:0] B_stage2 ;
    wire [17:0] B_mux2 ;
    assign B_mux2 = ((opmode_stage[4] == 1'b1) ? pre_addsub_result : B_stage1 ) ;
    register #(.WIDTH(18), .A0REG(B1REG), .RSTTYPE(RSTTYPE)) REG_B2
    (
        .clk(CLK), .rst(RSTB), .cenable(CEB), .reg_in(B_mux2), .reg_out(B_stage2)
    );


    //X MULTIPLIXER & BCOUT
    wire [35:0] mult_result ;
    assign mult_result = (A_stage2 * B_stage2);
    assign BCOUT = B_stage2 ;


    // M pipeline stage
    wire [35:0] m_stage ;
    register #(.WIDTH(36), .A0REG(MREG), .RSTTYPE(RSTTYPE)) REG_M
    (
        .clk(CLK), .rst(RSTM), .cenable(CEM), .reg_in(mult_result), .reg_out(m_stage)
    );

    // M assignment
    assign M = m_stage ;


    // carryin pipeline stage
    wire cin_stage ;
    wire carryin ;
    assign carryin = (CARRYINSEL == 1'b1) ? opmode_stage[5] : CARRYIN ;
    register #(.WIDTH(1), .A0REG(CARRYINREG), .RSTTYPE(RSTTYPE)) REG_CARRYIN
    (
        .clk(CLK), .rst(RSTCARRYIN), .cenable(CECARRYIN), .reg_in(carryin), .reg_out(cin_stage)
    );


    wire [47:0] X ;
    wire [47:0] Z ;

    assign X = (opmode[1:0] == 2'b00) ? 48'b0 :
               (opmode[1:0] == 2'b01) ? {12'b0,m_stage} :
               (opmode[1:0] == 2'b10) ? PCOUT : { D_stage1[11:0], A_stage2[17:0], B_stage2[17:0]} ;

    assign Z = (opmode[3:2] == 2'b00) ? 48'b0 :
               (opmode[3:2] == 2'b01) ? PCIN  :
               (opmode[3:2] == 2'b10) ? PCOUT : C_stage1 ;

    // post_add/sub
    wire [47:0] post_addsub_result;
    wire cout ;
    assign {cout, post_addsub_result} = ((opmode_stage[7] == 0) ? (X + Z + cin_stage) : (Z - (X + cin_stage))) ;


    // CARRYOUT pipeline stage
    wire cout_stage ;
    register #(.WIDTH(1), .A0REG(CARRYOUTREG), .RSTTYPE(RSTTYPE)) REG_COUT
    (
        .clk(CLK), .rst(RSTCARRYIN), .cenable(CECARRYIN), .reg_in(cout), .reg_out(cout_stage)
    );

    assign CARRYOUT = cout_stage ;
    assign CARRYOUTF = cout_stage ;


    // P pipeline stage
    wire [47:0] P_stage ;
    register #(.WIDTH(48), .A0REG(PREG), .RSTTYPE(RSTTYPE)) REG_P
    (
        .clk(CLK), .rst(RSTP), .cenable(CEP), .reg_in(post_addsub_result), .reg_out(P_stage)
    );


    assign P = P_stage ;
    assign PCOUT = P ;


endmodule
```

Test bench code:

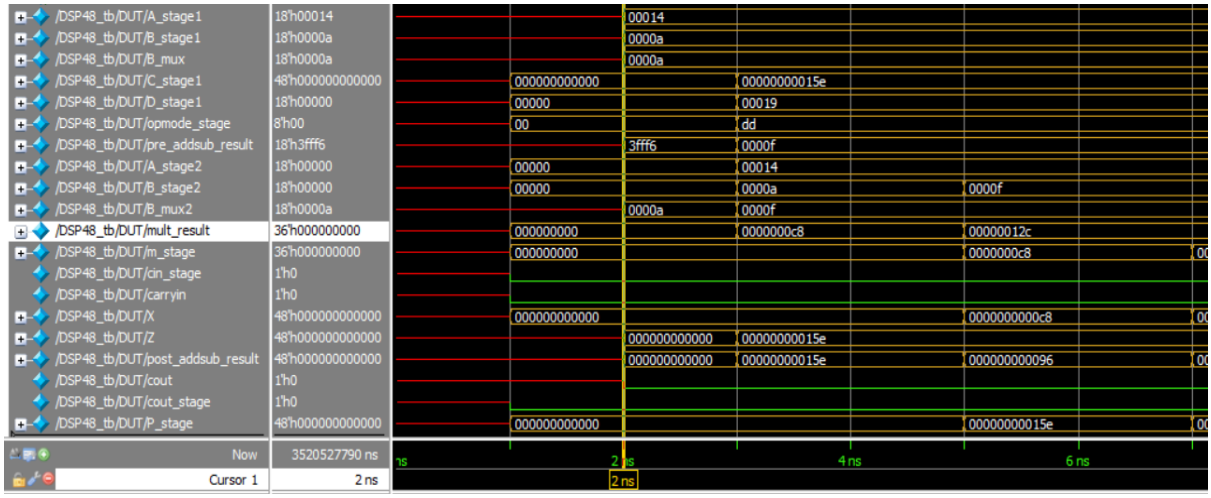
2.1. Verify Reset Operation:

Code:

```
module DSP48_tb();

    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 = 1 ;  // 1 FOR OPMODE[5] & 0 FOR CARRYIN
    parameter B_INPUT = 1 ;     // 1 FOR DIRECT & 0 FOR CASCADED
    parameter RSTTYPE = 1 ;     // 1 FOR SYNC & 0 FOR ASYNC

    reg [17:0] A ;
    reg [17:0] B ;
    reg [17:0] BCIN ;
    reg [17:0] D ;
    reg [47:0] C ;
    reg [47:0] PCIN ;
    reg [7:0] opmode ;
    reg CARRYIN ;
    reg CLK ;
    reg CEA ;
    reg CEB ;
    reg CEC ;
    reg CED ;
    reg CEM ;
    reg CEP ;
    reg CEOPMODE ;
    reg CECARRYIN ;
    reg RSTA ;
    reg RSTB ;
    reg RSTC ;
    reg RSTD ;
    reg RSTM ;
    reg RSTP ;
    reg RSTCARRYIN ;
    reg RSTOPMODE ;
    wire [47:0] PCOUT ;
    wire CARRYOUTF ;

    wire [47:0] P_dut ;
    wire [17:0] BCOUT_dut ;
    wire [35:0] M_dut ;
    wire CARRYOUT_dut ;
```

```
initial begin
        CLK = 0;
        forever begin
            #1 CLK = ~CLK;
        end
    end


    initial begin
    // Verify Reset Operation
        RSTA = 1 ;
        RSTB = 1 ;
        RSTC = 1 ;
        RSTD = 1 ;
        RSTM = 1 ;
        RSTP = 1 ;
        RSTCARRYIN = 1 ;
        RSTOPMODE = 1 ;
        repeat(1)begin
            @(negedge CLK);
            if (M_dut != 0 || P_dut != 0 || BCOUT_dut != 0 || CARRYOUT_dut !=0) begin
                $display ("RESET FAILED");
                $stop;
            end
            else begin
                $display ("reset passed");
            end
        end

```

RESULTS:




```
# M_dut=xxxxxxxxx, BCOUT_dut=xxxxx, CARRYOUT_dut=x, P_dut=xxxxxxxxxxxx
# M_dut=000000000, BCOUT_dut=00000, CARRYOUT_dut=0, P_dut=000000000000
# reset passed
```

2.2. Verify DSP Path 1:

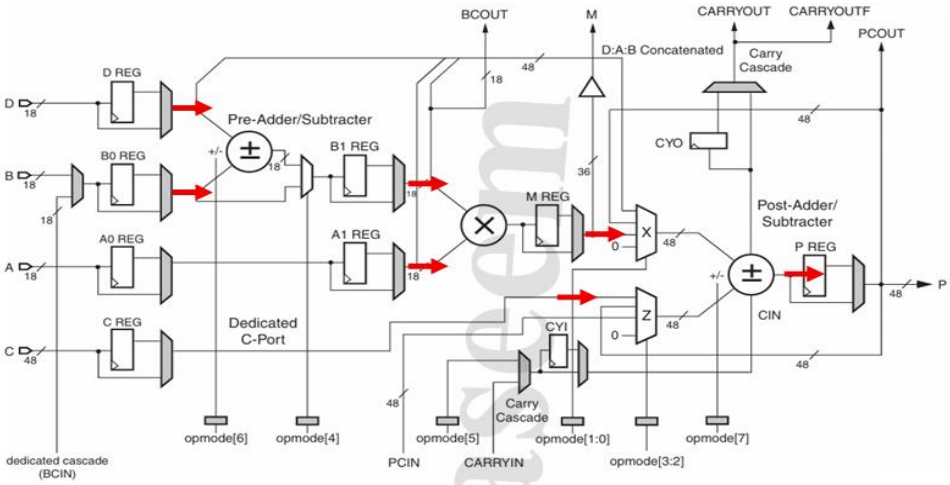


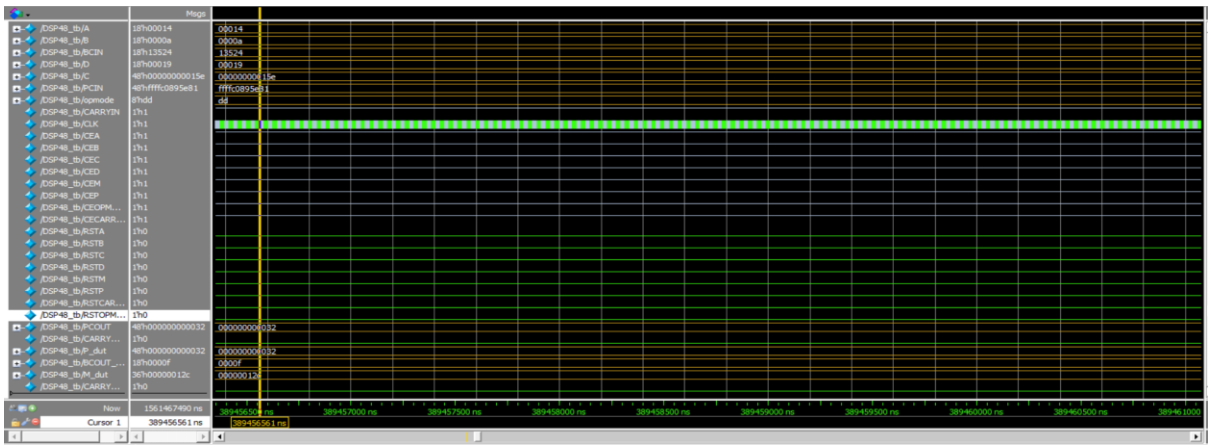
Figure 1: Path 1 Data Flow

Code:

```
//Deassert all reset signals
        RSTA = 0 ;
        RSTB = 0 ;
        RSTC = 0 ;
        RSTD = 0 ;
        RSTM = 0 ;
        RSTP = 0 ;
        RSTCARRYIN = 0 ;
        RSTOPMODE = 0 ;
        // assert all clock enable signals
        CEA = 1 ;
        CEB =1 ;
        CEC = 1 ;
        CED = 1 ;
        CEM = 1 ;
        CEP = 1 ;
        CEOPMODE = 1 ;
        CECARRYIN = 1 ;

        //Verify DSP Path 1
        opmode = 8'b11011101 ;
        A = 20 ;
        B = 10 ;
        C = 350 ;
        D = 25 ;
```

```
    BCIN = $random ;
           PCIN = $random ;
           CARRYIN = $random ;
           repeat(4) @(negedge CLK);
           if ( P_dut != 'h32) begin
               $display ("PATH_1 'P' FAILED");
               $stop;
           end
           else if (BCOUT_dut != 'hf) begin
               $display ("PATH_1 'BCOUT' FAILED");
               $stop;
           end
           else if (CARRYOUT_dut !=0) begin
               $display ("PATH_1 'CARRYOUT' FAILED");
               $stop;
           end
           else if (M_dut != 'h12c ) begin
               $display ("PATH_1 'M' FAILED");
               $stop;
           end
           else begin
               $display ("PATH_1 passed");
           end
   end
```

Results:



```
# M_dut=000000000, BCOUT_dut=0000a, CARRYOUT_dut=0, P_dut=000000000000
# M_dut=0000000c8, BCOUT_dut=0000f, CARRYOUT_dut=0, P_dut=00000000015e
# M_dut=00000012c, BCOUT_dut=0000f, CARRYOUT_dut=0, P_dut=000000000096
# M_dut=00000012c, BCOUT_dut=0000f, CARRYOUT_dut=0, P_dut=000000000032
# PATH_1 passed
```

2.3. Verify DSP Path 2

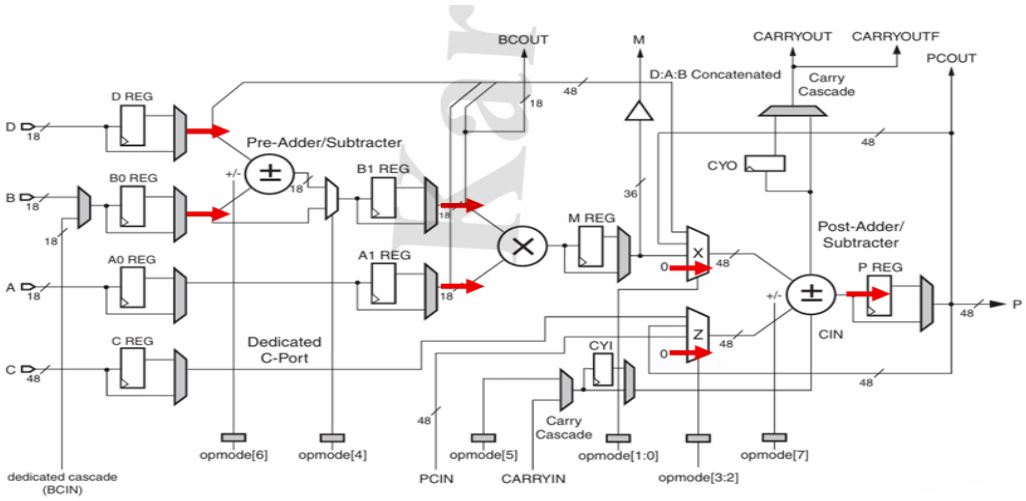


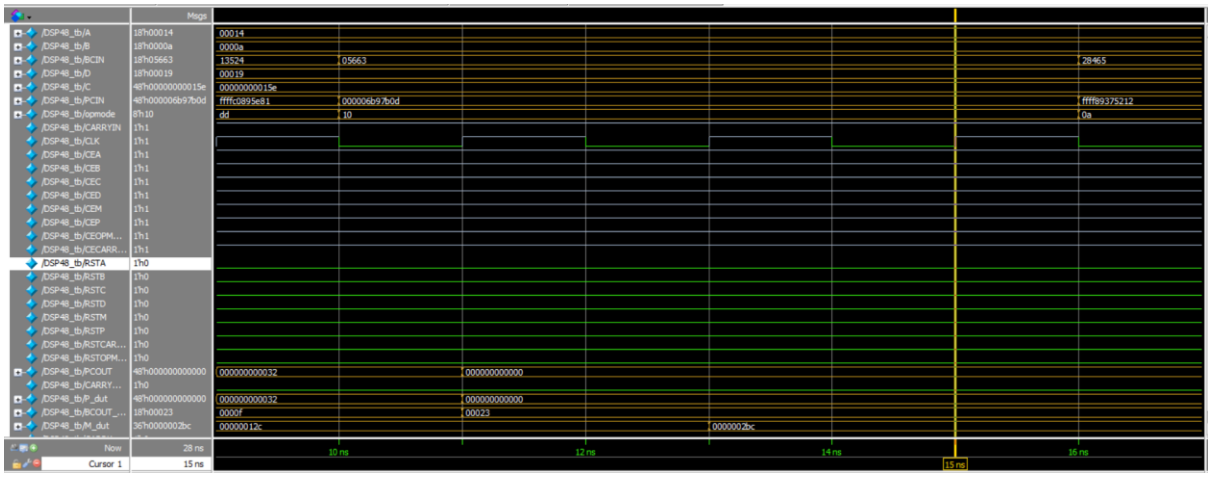
Figure 2: Path 2 Data Flow

Code:

```
// 2.3. Verify DSP Path 2
        opmode = 8'b00010000 ;
        A = 20 ;
        B = 10 ;
        C = 350 ;
        D = 25 ;

        BCIN = $random ;
        PCIN = $random ;
        CARRYIN = $random ;
        repeat(3) @(negedge CLK);
        if ( P_dut != 'h0) begin
            $display ("PATH_2 'P' FAILED");
            $stop;
        end
        else if (BCOUT_dut != 'h23) begin
            $display ("PATH_2 'BCOUT' FAILED");
            $stop;
        end
        else if (CARRYOUT_dut !=0) begin
            $display ("PATH_2 'CARRYOUT' FAILED");
            $stop;
        end
        else if (M_dut != 'h2bc ) begin
            $display ("PATH_2 'M' FAILED");
            $stop;
        end
        else begin
            $display ("PATH_2 passed");
        end
```

Results:

```
# M_dut=00000012c, BCOUT_dut=00023, CARRYOUT_dut=0, P_dut=000000000000
# M_dut=0000002bc, BCOUT_dut=00023, CARRYOUT_dut=0, P_dut=000000000000
# PATH_2 passed
```



2.4. Verify DSP Path 3

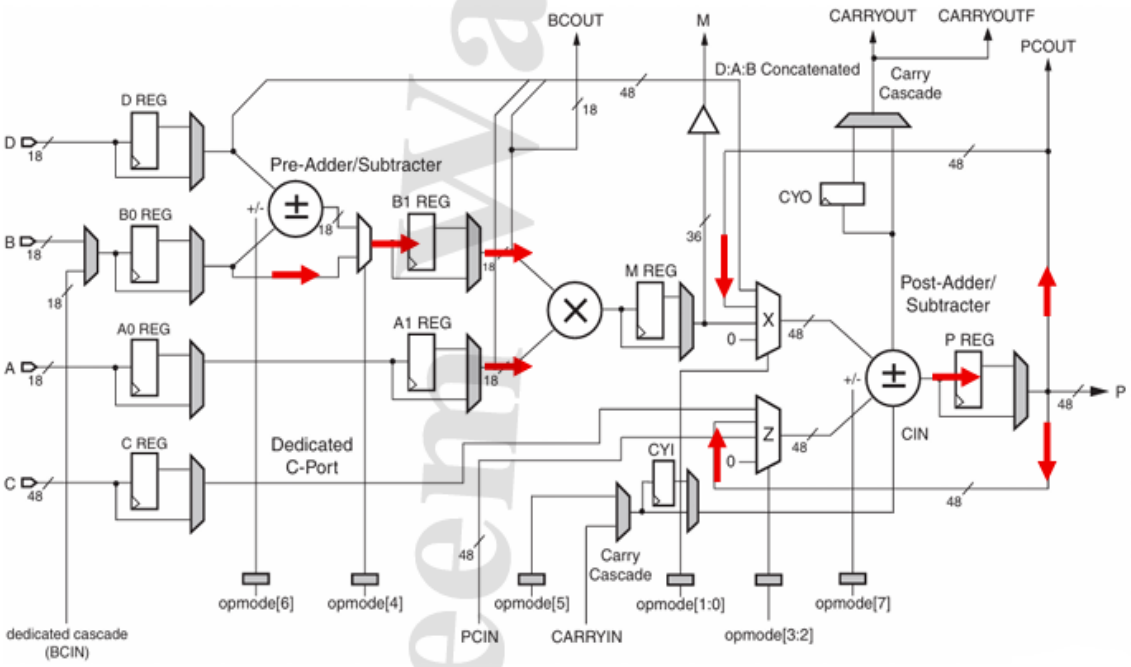


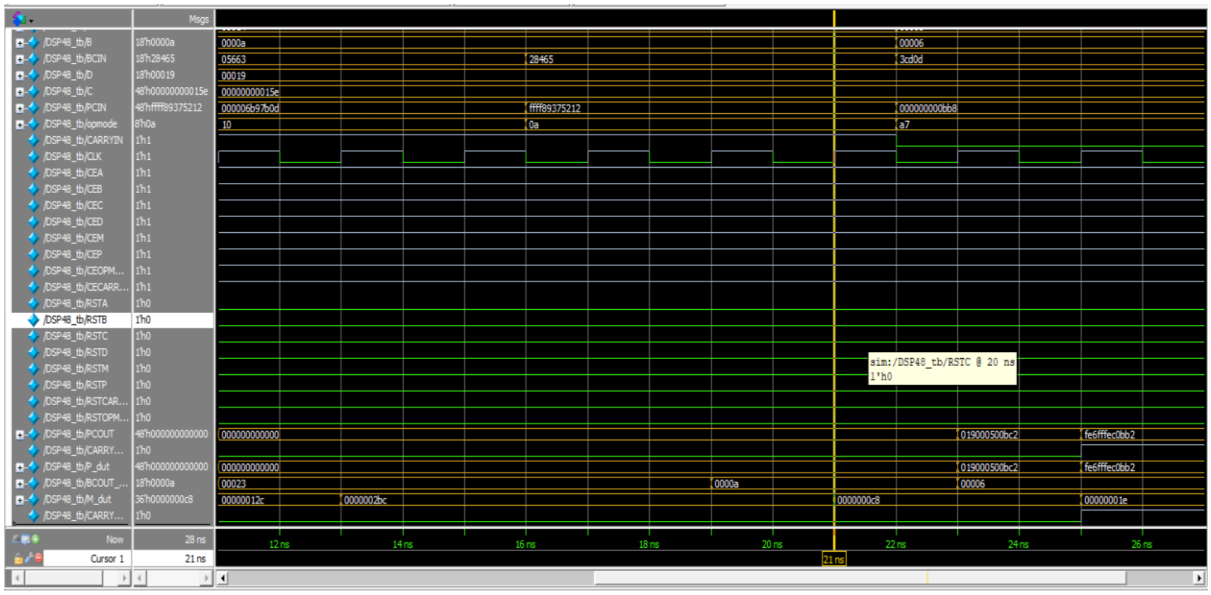
Figure 3: Path 3 Data Flow

Code:

```
// 2.4. Verify DSP Path 3
        opmode =  8'b00001010 ;
        A = 20 ;
        B = 10 ;
        C = 350 ;
        D = 25 ;

        BCIN = $random ;
        PCIN = $random ;
        CARRYIN = $random ;
        repeat(3) @(negedge CLK);
        if ( P_dut != PCOUT) begin
            $display ("PATH_3 'P' FAILED");
            $stop;
        end
        else if (BCOUT_dut != 'ha) begin
            $display ("PATH_3 'BCOUT' FAILED");
            $stop;
        end
        else if (CARRYOUT_dut != CARRYOUTF) begin
            $display ("PATH_3 'CARRYOUT' FAILED");
            $stop;
        end
        else if (M_dut != 'hc8 ) begin
            $display ("PATH_3 'M' FAILED");
            $stop;
        end
        else begin
            $display ("PATH_3 passed");
        end
```

Results:

```
# M_dut=0000002bc, BCOUT_dut=0000a, CARRYOUT_dut=0, P_dut=000000000000
# M_dut=0000000c8, BCOUT_dut=0000a, CARRYOUT_dut=0, P_dut=000000000000
# PATH_3 passed
```



2.5. Verify DSP Path 4

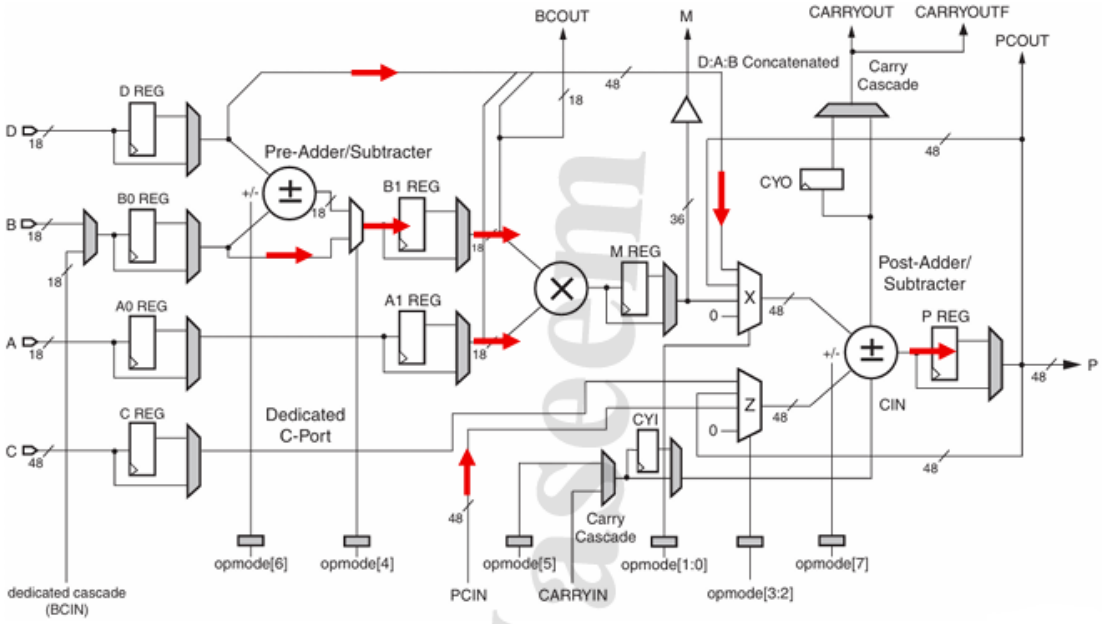


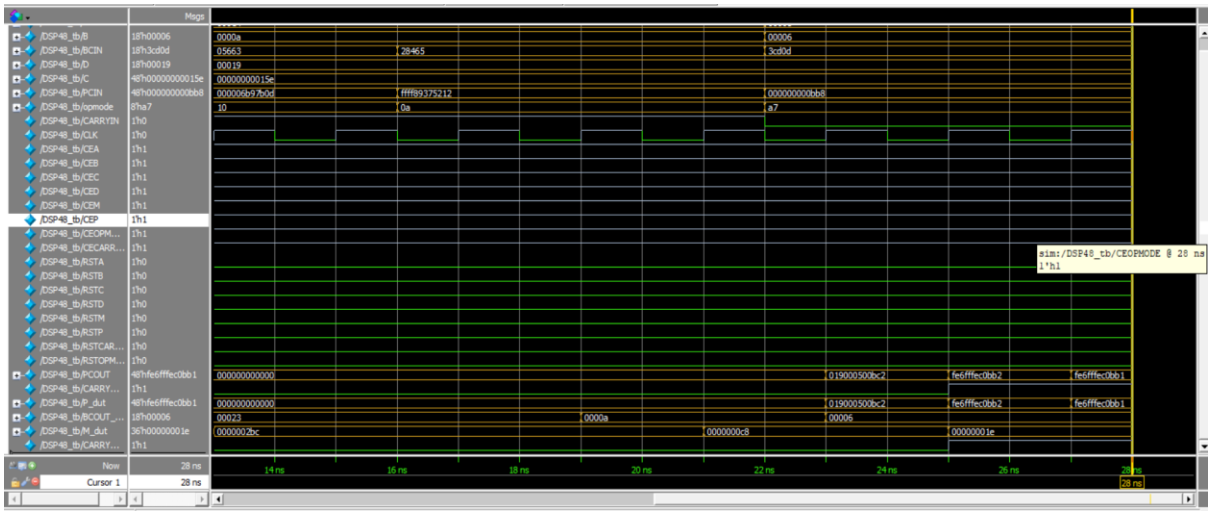
Figure 4: Path 4 Data Flow

Code:

```
// 2.5. Verify DSP Path 4
        opmode =  8'b10100111 ;
        A = 5 ;
        B = 6 ;
        C = 350 ;
        D = 25 ;
        PCIN = 3000 ;

        BCIN = $random ;
        CARRYIN = $random ;
        repeat(3) @(negedge CLK);
        if ( P_dut != 'hfe6fffec0bb1) begin
            $display ("PATH_4 'P' FAILED");
            $stop;
        end
        else if (BCOUT_dut != 'h6) begin
            $display ("PATH_4 'BCOUT' FAILED");
            $stop;
        end
        else if (CARRYOUT_dut != 1) begin
            $display ("PATH_4 'CARRYOUT' FAILED");
            $stop;
        end
        else if (M_dut != 'h1e ) begin
            $display ("PATH_4 'M' FAILED");
            $stop;
        end
        else begin
            $display ("PATH_4 passed");
        end
    end

    initial begin
        $monitor ("M_dut=%h, BCOUT_dut=%h, CARRYOUT_dut=%h, P_dut=%h", M_dut, BCOUT_dut, CARRYOUT_dut, P_dut);
    end

endmodule
```

Results:

```
# M_dut=0000000c8, BCOUT_dut=00006, CARRYOUT_dut=0, P_dut=019000500bc2
# M_dut=00000001e, BCOUT_dut=00006, CARRYOUT_dut=1, P_dut=fe6fffec0bb2
# M_dut=00000001e, BCOUT_dut=00006, CARRYOUT_dut=1, P_dut=fe6fffec0bb1
# PATH_4 passed
```



Do file:

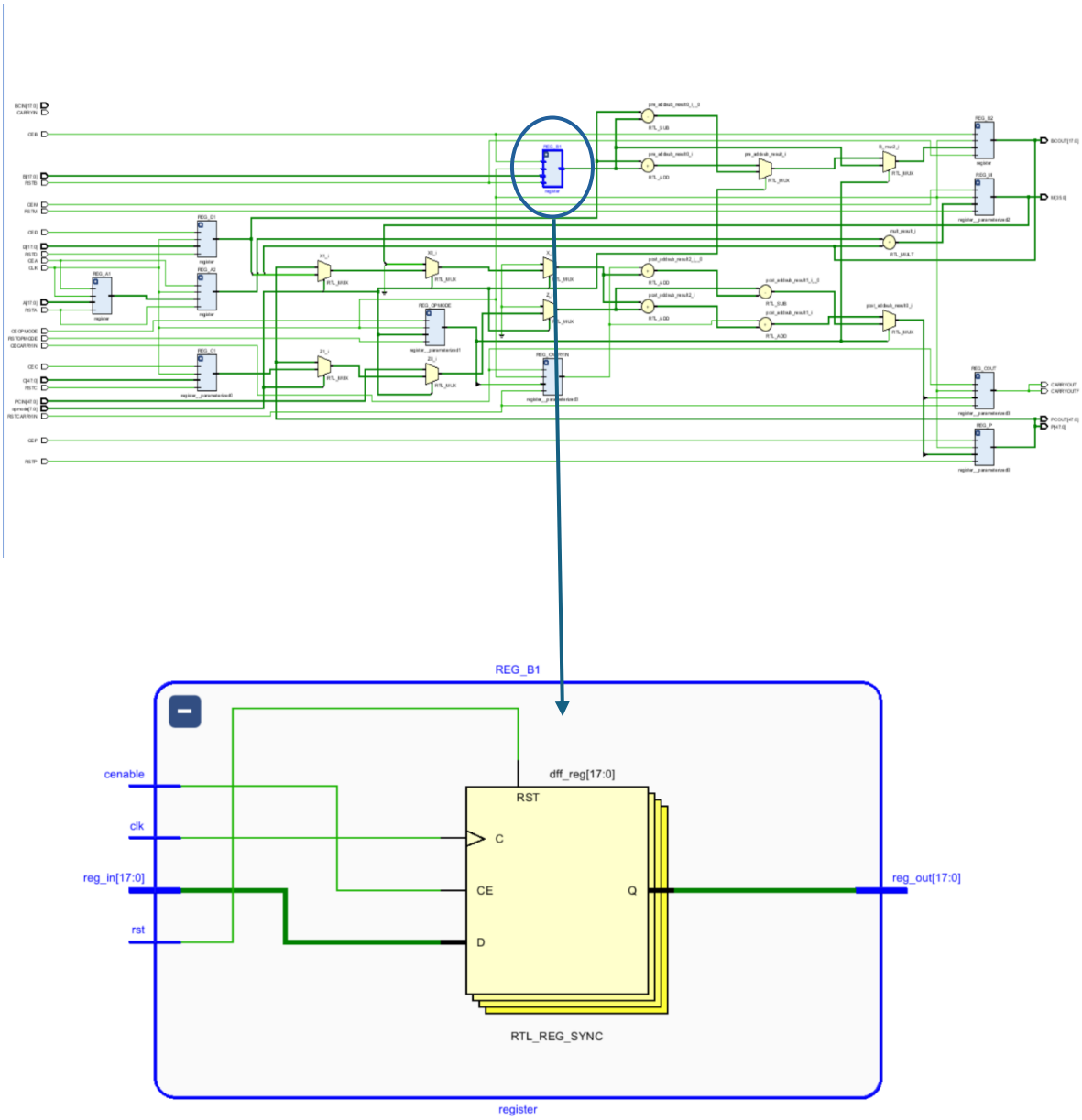
```
1   vlib work
2   vlog DSP48.v DSP48_tb.v
3   vsim -voptargs=+acc work.DSP48_tb
4   add wave *
5   run -all
6   #quit -sim
```

Constraints file:

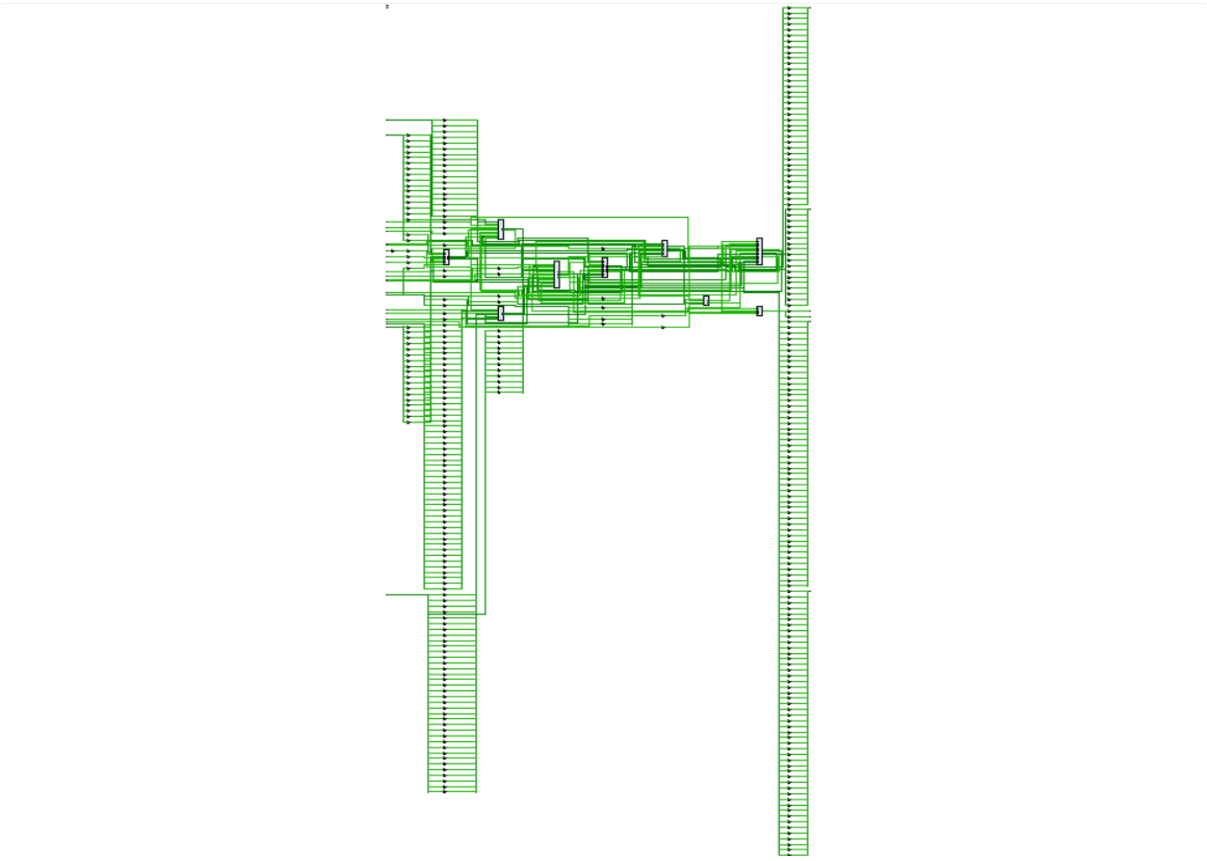
```
1  ## Clock signal
2  set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33} [get_ports CLK]
3  create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports CLK]
4
5
```

Elaboration:

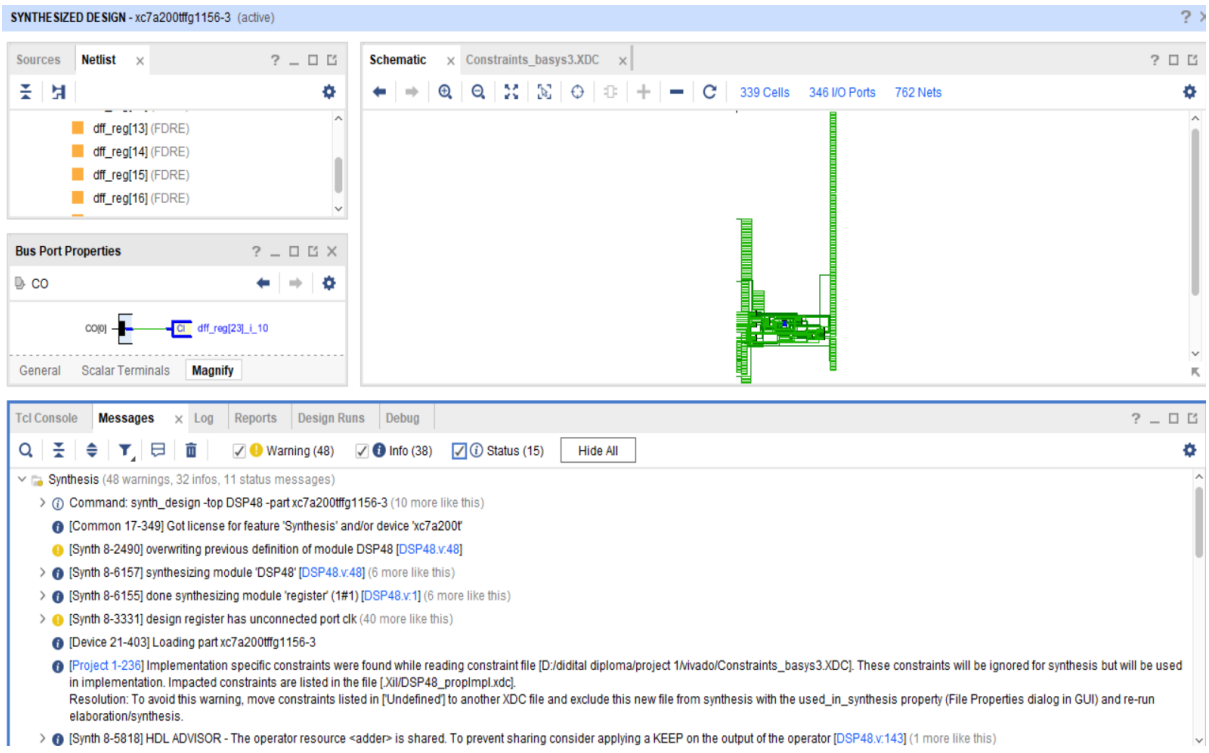
Schematic snippets:



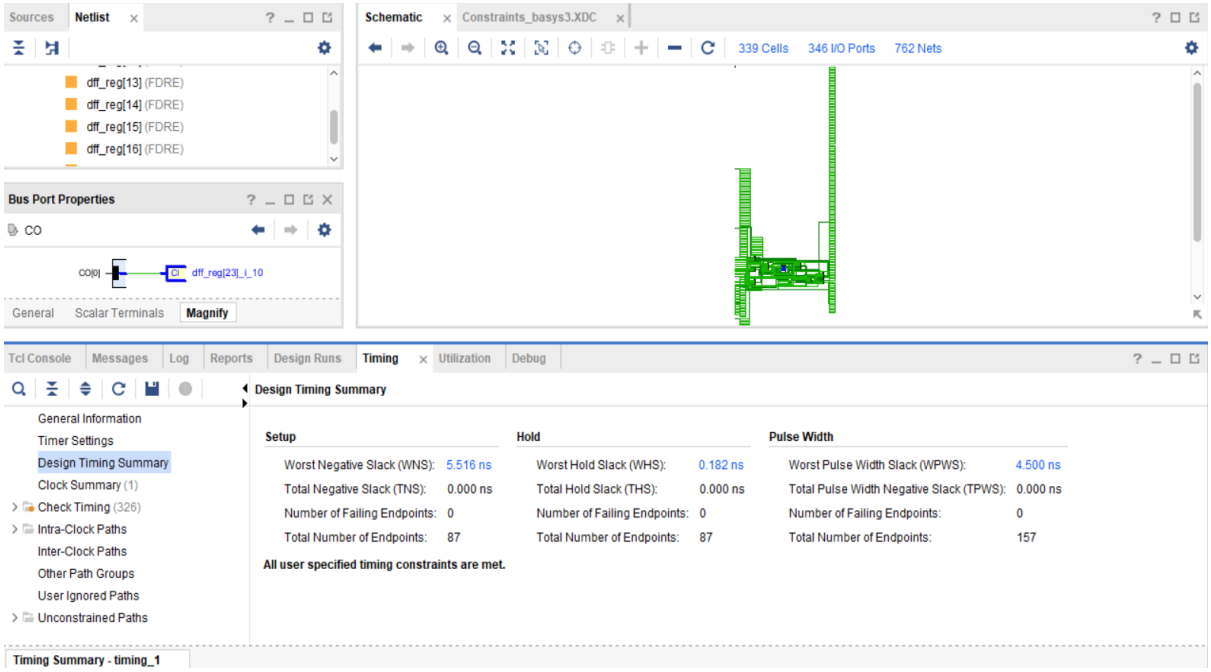
Synthesis schematic:



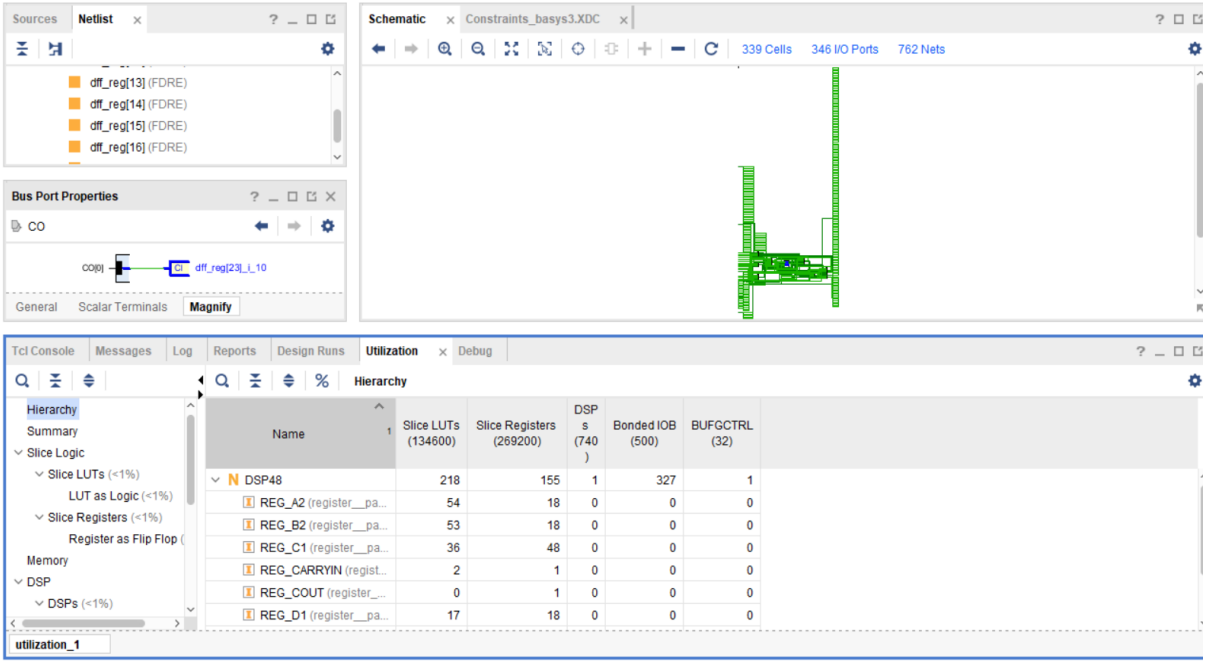
Synthesis Messages:



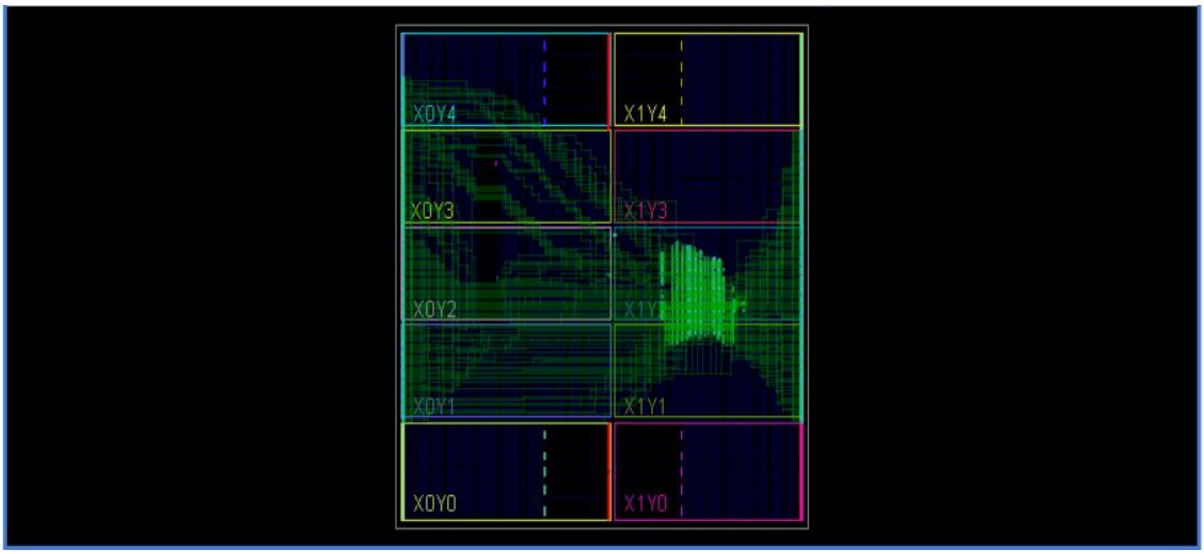
Synthesis Timing summary:



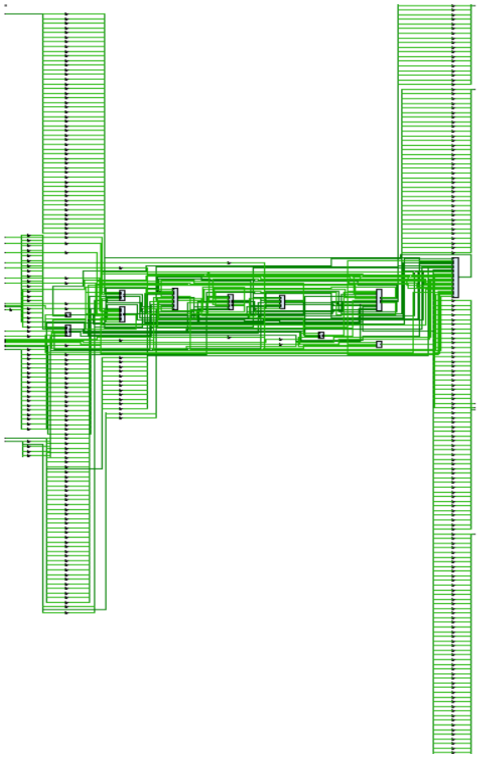
Synthesis Utilization report:



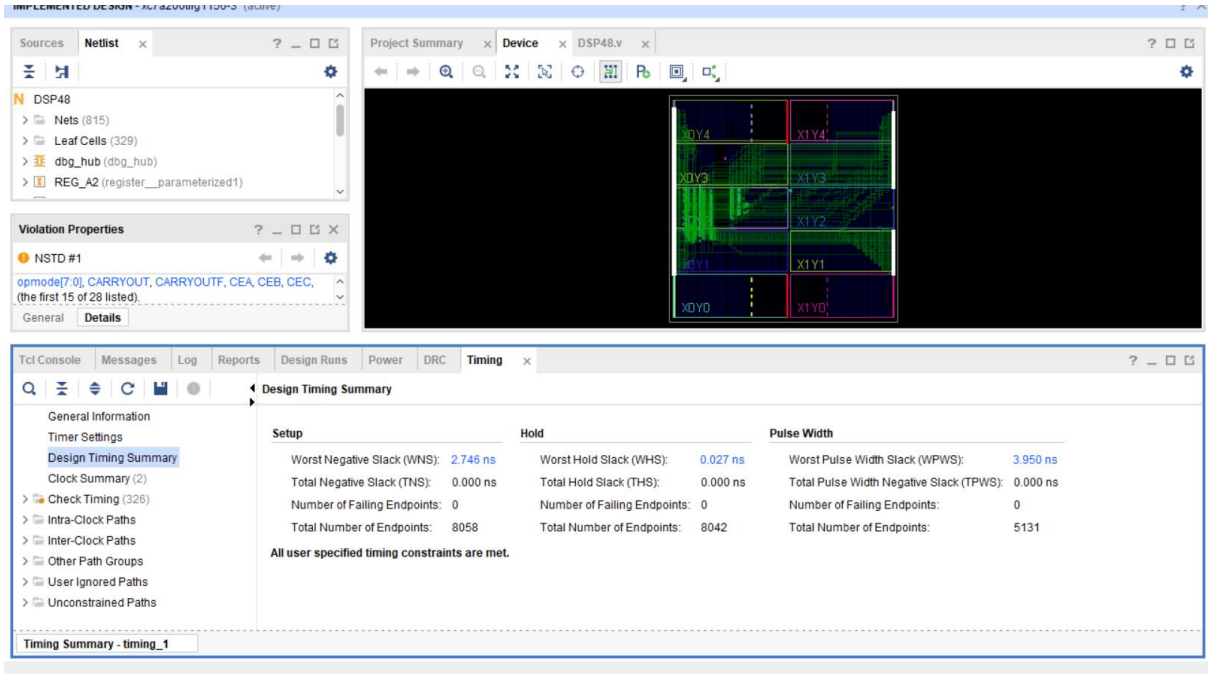
Device snippets:



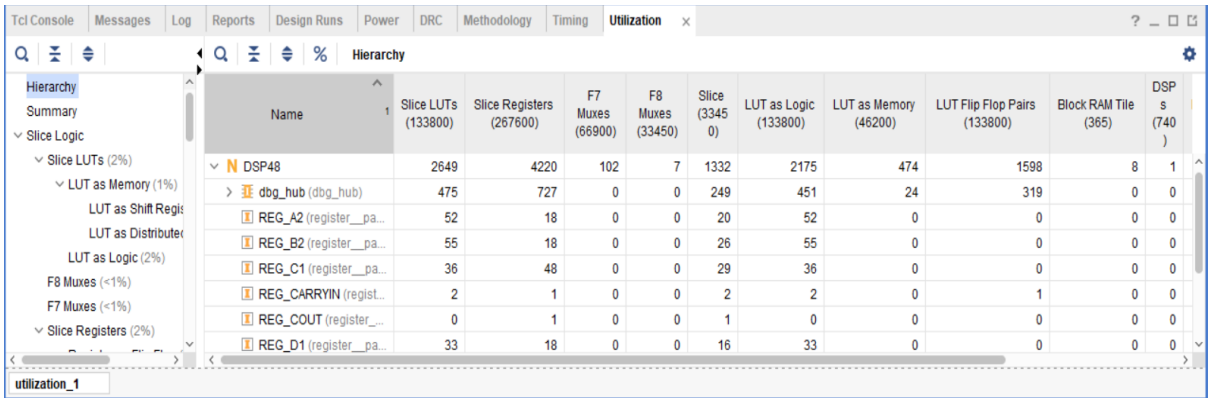
Implementation schematic:



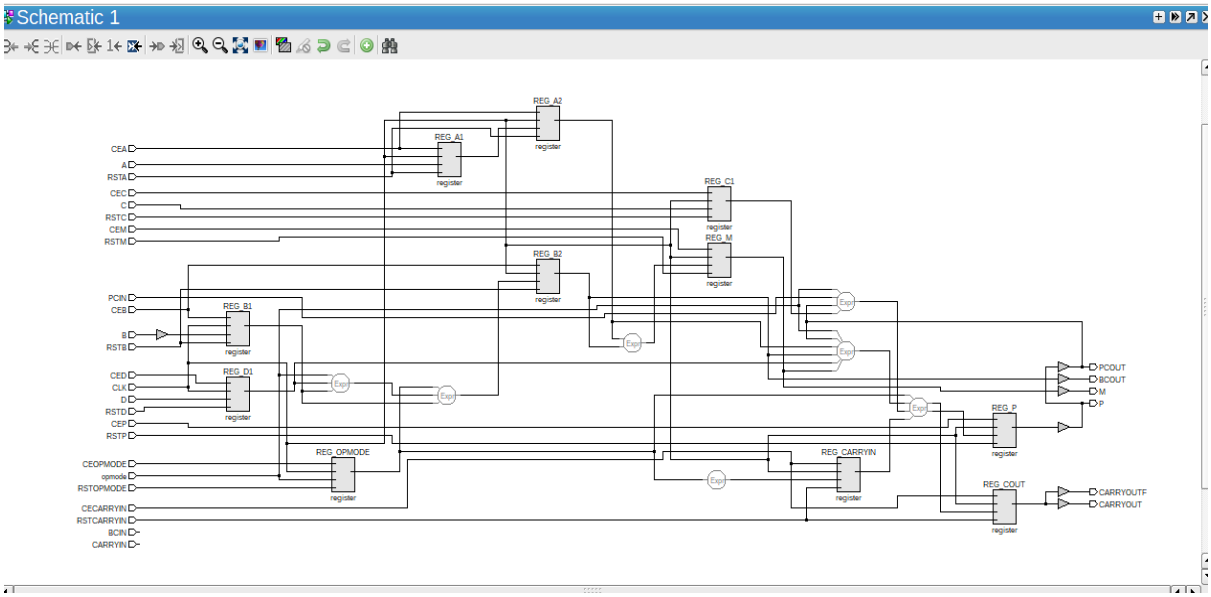
Implementation timing summary:



Implementation Utilization report:



LINT Schematic:



Snippets showing no errors: