

Project_1

Spartan6 - DSP48A1

Supervisor: Eng. Kareem Waseem

Created by: Abdelrahman Mahmoud Abdelhamed

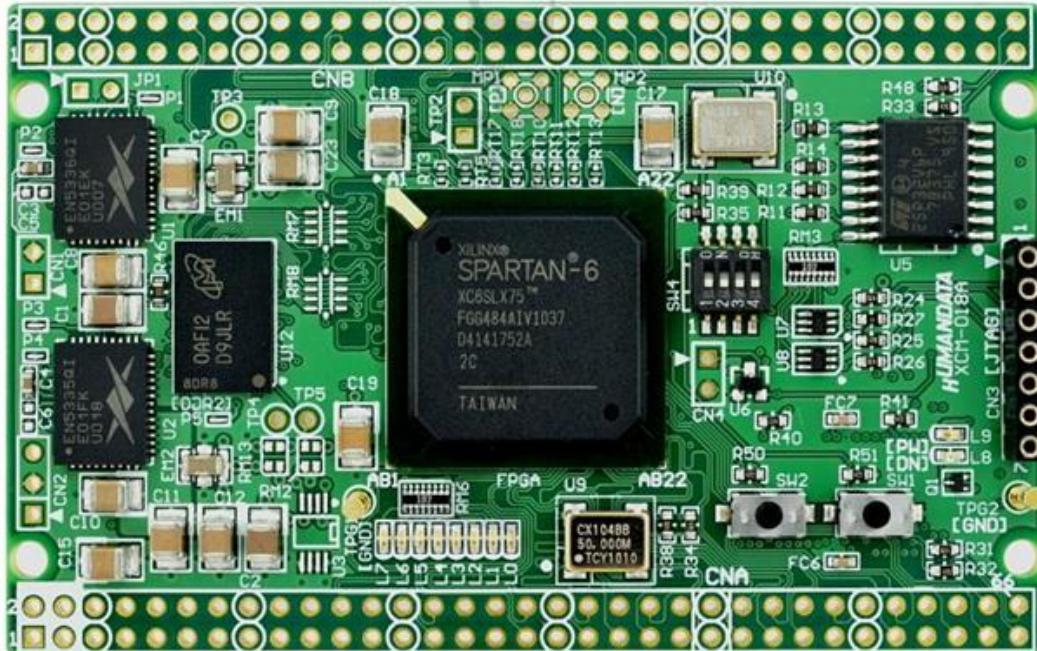


Table of Contents

RTL CODE:	3
Register pipeline code:	3
Code.....	4
Test bench code:	6
2.1. Verify Reset Operation:	6
Code:.....	6
RESULTS:	7
2.2. Verify DSP Path 1:	8
Code:	8
Results:	9
2.3. Verify DSP Path 2.....	10
Code:.....	10
Results:	11
2.4. Verify DSP Path 3.....	11
Code:.....	12
Results:	13
2.5. Verify DSP Path 4.....	13
Code:.....	14
Results:	14
Do file:.....	15
Constraints file:	15
Synthesis schematic:	16
Synthesis Messages:	17
Synthesis Timing summary:	18
Synthesis Utilization report:	18

Device snippets:	19
Implementation schematic:	19
Implementation timing summary	20
Implementation Utilization report.....	20

RTL CODE:

Register pipeline code:

```
 1 module register( clk, rst, cenable, reg_in, reg_out);
 2
 3   parameter WIDTH = 18 ;
 4   parameter A0REG = 1 ;
 5   parameter RSTTYPE = 1 ;                                // 1 FOR SYNC & 0 FOR ASYNC
 6
 7   input clk, rst, cenable ;
 8   input [WIDTH-1:0] reg_in ;
 9   output reg [WIDTH-1:0] reg_out ;
10
11   reg [WIDTH-1:0] dff ;
12
13   generate
14     if (RSTTYPE == 1'b0) begin                         // ASYNC RS
15       always @(posedge clk or posedge rst) begin
16         if (rst) begin
17           dff <= 0 ;
18         end
19         else if (cenable) begin
20           dff <= reg_in ;
21         end
22       end
23     end
24     else if (RSTTYPE == 1'b1) begin                     // SYNC RST
25       always @(posedge clk ) begin
26         if (rst) begin
27           dff <= 0 ;
28         end
29         else if (cenable) begin
30           dff <= reg_in ;
31         end
32       end
33     end
34   endgenerate
35   always @(*)begin
36     if (A0REG) begin
37       reg_out = dff ;
38     end
39     else begin
40       reg_out = reg_in ;
41     end
42   end
43 endmodule
```

Code:

```
1 module DSP48 (
2     input [17:0] A ,
3     input [17:0] B ,
4     input [17:0] BCIN ,
5     input [17:0] D ,
6     input [47:0] C ,
7     input [47:0] PCIN ,
8     input [7:0] opmode ,
9     input CARRYIN ,
10    input CLK ,
11    input CEA ,
12    input CEB ,
13    input CEC ,
14    input CED ,
15    input CEM ,
16    input CEP ,
17    input CEOPMODE ,
18    input CECARRYIN ,
19    input RSTA ,
20    input RSTB ,
21    input RSTC ,
22    input RSTD ,
23    input RSTM ,
24    input RSTP ,
25    input RSTOPMODE ,
26    input RSTCARRYIN ,
27
28    output [17:0] BCOUT ,
29    output [35:0] M ,
30    output [47:0] P ,
31    output [47:0] PCOUT ,
32    output CARRYOUT ,
33    output CARRYOUTF
34 );
35
36 parameter A0REG = 0 ;
37 parameter A1REG = 1 ;
38 parameter B0REG = 0 ;
39 parameter B1REG = 1 ;
40 parameter CREG = 1 ;
41 parameter DREG = 1 ;
42 parameter MREG = 1 ;
43 parameter PREG = 1 ;
44 parameter CARRYINREG = 1 ;
45 parameter CARRYOUTREG = 1 ;
46 parameter OPMODEREG = 1 ;
47 parameter CARRYINSEL = 1 ; // 1 FOR OPMODE[5] & 0 FOR CARRYIN
48 parameter B_INPUT = 1 ; // 1 FOR DIRECT & 0 FOR CASCADED
49 parameter RSTTYPE = 1 ; // 1 FOR SYNC & 0 FOR ASYNC
50
51
52 // A pipeline 1st stage
53 wire [17:0] A_stage1 ;
54 register #(.WIDTH(18), .A0REG(A0REG), .RSTTYPE(RSTTYPE)) REG_A1
55 (
56     .clk(CLK), .rst(RSTA), .cenable(CEA), .reg_in(A), .reg_out(A_stage1)
57 );
58
59
60 // B pipeline 1st stage
61 wire [17:0] B_stage1 ;
62 wire [17:0] B_mux ;
63 assign B_mux = ((B_INPUT == 1'b1) ? B : BCIN) ;
64 register #(.WIDTH(18), .A0REG(B0REG), .RSTTYPE(RSTTYPE)) REG_B1
65 (
66     .clk(CLK), .rst(RSTB), .cenable(CEB), .reg_in(B_mux), .reg_out(B_stage1)
67 );
68
69
70 // C pipeline 1st stage
71 wire [47:0] C_stage1 ;
72 register #(.WIDTH(48), .A0REG(CREG), .RSTTYPE(RSTTYPE)) REG_C1
73 (
74     .clk(CLK), .rst(RSTC), .cenable(CEC), .reg_in(C), .reg_out(C_stage1)
75 );
76
77
78 // D pipeline 1st stage
79 wire [17:0] D_stage1 ;
80 register #(.WIDTH(18), .A0REG(DREG), .RSTTYPE(RSTTYPE)) REG_D1
81 (
82     .clk(CLK), .rst(RSTD), .cenable(CED), .reg_in(D), .reg_out(D_stage1)
83 );
84
85
86 // OPMODE pipeline stage
87 wire [7:0] opmode_stage ;
88 register #(.WIDTH(8), .A0REG(OPMODEREG), .RSTTYPE(RSTTYPE)) REG_OPMODE
89 (
90     .clk(CLK), .rst(RSTOPMODE), .cenable(CEOPMODE), .reg_in(opmode), .reg_out(opmode_stage)
91 );
92
93
94 // PRE_ADD/SUB
95 wire [17:0] pre_addsub_result ;
96 assign pre_addsub_result = ((opmode[6] == 0) ? (D_stage1 + B_stage1) : (D_stage1 - B_stage1)) ;
97
```

```

● ○ ●
1 // A pipeline 2nd stage
2   wire [17:0] A_stage2 ;
3   register #( .WIDTH(18), .A0REG(A1REG), .RSTTYPE(RSTTYPE)) REG_A2
4   (
5     .clk(CLK), .rst(RSTA), .cenable(CEA), .reg_in(A_stage1), .reg_out(A_stage2)
6   );
7
8
9 // B pipeline 2nd stage
10  wire [17:0] B_stage2 ;
11  wire [17:0] B_mux2 ;
12  assign B_mux2 = ((opmode_stage[4] == 1'b1) ? pre_addsub_result : B_stage1 ) ;
13  register #( .WIDTH(18), .A0REG(B1REG), .RSTTYPE(RSTTYPE)) REG_B2
14  (
15    .clk(CLK), .rst(RSTB), .cenable(CEB), .reg_in(B_mux2), .reg_out(B_stage2)
16  );
17
18
19 //X MULTIPLIXER & BCOUT
20  wire [35:0] mult_result ;
21  assign mult_result = (A_stage2 * B_stage2);
22  assign BCOUT = B_stage2 ;
23
24
25 // M pipeline stage
26  wire [35:0] m_stage ;
27  register #( .WIDTH(36), .A0REG(MREG), .RSTTYPE(RSTTYPE)) REG_M
28  (
29    .clk(CLK), .rst(RSTM), .cenable(CEM), .reg_in(mult_result), .reg_out(m_stage)
30  );
31
32 // M assignment
33  assign M = m_stage ;
34
35
36 // carryin pipeline stage
37  wire cin_stage ;
38  wire carryin ;
39  assign carryin = (CARRYINSEL == 1'b1) ? opmode_stage[5] : CARRYIN ;
40  register #( .WIDTH(1), .A0REG(CARRYINREG), .RSTTYPE(RSTTYPE)) REG_CARRYIN
41  (
42    .clk(CLK), .rst(RSTCARRYIN), .cenable(CECARRYIN), .reg_in(carryin), .reg_out(cin_stage)
43  );
44
45
46  wire [47:0] X ;
47  wire [47:0] Z ;
48
49  assign X = (opmode[1:0] == 2'b00) ? 48'b0 :
50    (opmode[1:0] == 2'b01) ? {12'b0,m_stage} :
51    (opmode[1:0] == 2'b10) ? PCOUT : { D_stage1[11:0], A_stage2[17:0], B_stage2[17:0] } ;
52
53  assign Z = (opmode[3:2] == 2'b00) ? 48'b0 :
54    (opmode[3:2] == 2'b01) ? PCIN :
55    (opmode[3:2] == 2'b10) ? PCOUT : C_stage1 ;
56
57 // post_add/sub
58  wire [47:0] post_addsub_result;
59  wire cout ;
60  assign [cout, post_addsub_result] = ((opmode_stage[7] == 0) ? (X + Z + cin_stage) : (Z - (X + cin_stage))) ;
61
62
63 // CARRYOUT pipeline stage
64  wire cout_stage ;
65  register #( .WIDTH(1), .A0REG(CARRYOUTREG), .RSTTYPE(RSTTYPE)) REG_COUT
66  (
67    .clk(CLK), .rst(RSTCARRYIN), .cenable(CECARRYIN), .reg_in(cout), .reg_out(cout_stage)
68  );
69
70  assign CARRYOUT = cout_stage ;
71  assign CARRYOUTF = cout_stage ;
72
73
74 // P pipeline stage
75  wire [47:0] P_stage ;
76  register #( .WIDTH(48), .A0REG(PREG), .RSTTYPE(RSTTYPE)) REG_P
77  (
78    .clk(CLK), .rst(RSTP), .cenable(CEP), .reg_in(post_addsub_result), .reg_out(P_stage)
79  );
80
81
82  assign P = P_stage ;
83  assign PCOUT = P ;
84
85
86 endmodule

```

Test bench code:

2.1. Verify Reset Operation:

Code:

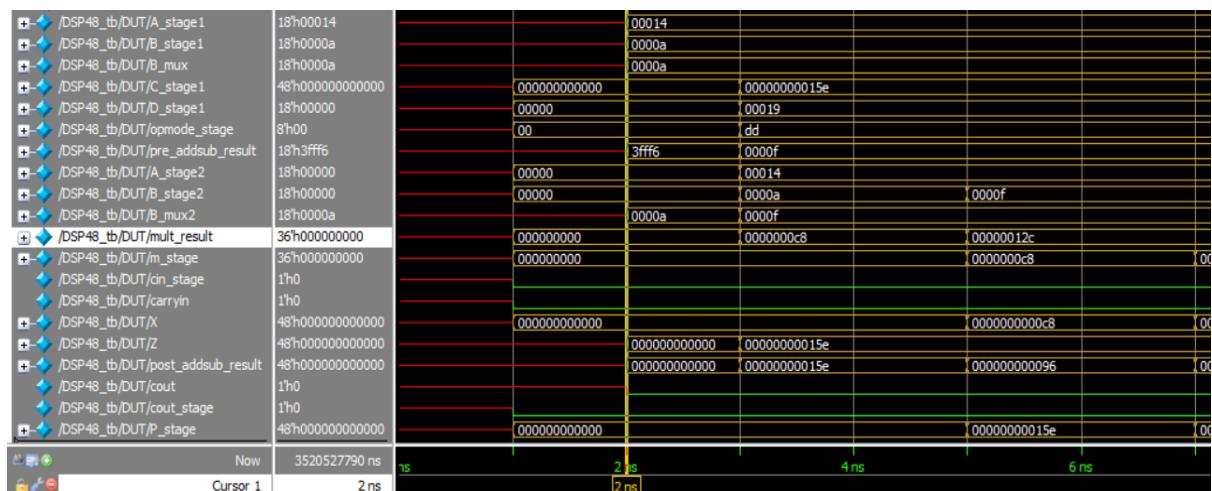
```
● ● ●
1 module DSP48_tb();
2
3 parameter A0REG = 0 ;
4 parameter A1REG = 1 ;
5 parameter B0REG = 0 ;
6 parameter B1REG = 1 ;
7 parameter CREG = 1 ;
8 parameter DREG = 1 ;
9 parameter MREG = 1 ;
10 parameter PREG = 1 ;
11 parameter CARRYINREG = 1 ;
12 parameter CARRYOUTREG = 1 ;
13 parameter OPMODEREG = 1 ;
14 parameter CARRYINSEL = 1 ; // 1 FOR OPMODE[5] & 0 FOR CARRYIN
15 parameter B_INPUT = 1 ; // 1 FOR DIRECT & 0 FOR CASCADED
16 parameter RSTTYPE = 1 ; // 1 FOR SYNC & 0 FOR ASYNC
17
18 reg [17:0] A ;
19 reg [17:0] B ;
20 reg [17:0] BCIN ;
21 reg [17:0] D ;
22 reg [47:0] C ;
23 reg [47:0] PCIN ;
24 reg [7:0] opmode ;
25 reg CARRYIN ;
26 reg CLK ;
27 reg CEA ;
28 reg CEB ;
29 reg CEC ;
30 reg CED ;
31 reg CEM ;
32 reg CEP ;
33 reg CEOPMODE ;
34 reg CECARRYIN ;
35 reg RSTA ;
36 reg RSTB ;
37 reg RSTC ;
38 reg RSTD ;
39 reg RSTM ;
40 reg RSTP ;
41 reg RSTCARRYIN ;
42 reg RSTOPMODE ;
43 wire [47:0] PCOUT ;
44 wire CARRYOUTF ;
45
46 wire [47:0] P_dut ;
47 wire [17:0] BCOUT_dut ;
48 wire [35:0] M_dut ;
49 wire CARRYOUT_dut ;
```

```

1  initial begin
2      CLK = 0;
3      forever begin
4          #1 CLK = ~CLK;
5      end
6  end
7
8
9  initial begin
10 // Verify Reset Operation
11     RSTA = 1 ;
12     RSTB = 1 ;
13     RSTC = 1 ;
14     RSTD = 1 ;
15     RSTM = 1 ;
16     RSTP = 1 ;
17     RSTCARRYIN = 1 ;
18     RSTOPMODE = 1 ;
19     repeat(1)begin
20         @(negedge CLK);
21         if (M_dut != 0 || P_dut != 0 || BCOUT_dut != 0 || CARRYOUT_dut !=0) begin
22             $display ("RESET FAILED");
23             $stop;
24         end
25         else begin
26             $display ("reset passed");
27         end
28     end
29

```

RESULTS:



```

# M_dut=xxxxxxxxxx, BCOUT_dut=xxxxxx, CARRYOUT_dut=x, P_dut=xxxxxxxxxxxx
# M_dut=0000000000, BCOUT_dut=00000, CARRYOUT_dut=0, P_dut=000000000000
# reset passed

```

2.2. Verify DSP Path 1:

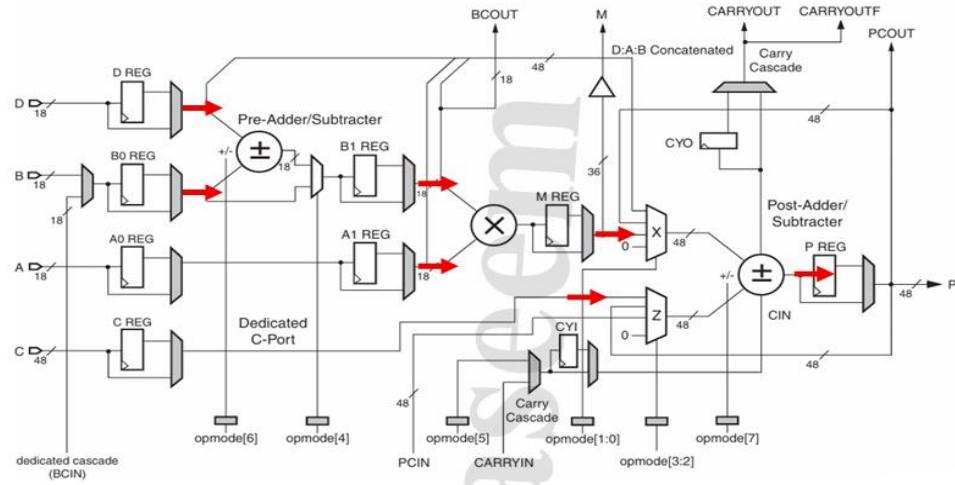


Figure 1: Path 1 Data Flow

Code:

```

1 //Deassert all reset signals
2     RSTA = 0 ;
3     RSTB = 0 ;
4     RSTC = 0 ;
5     RSTD = 0 ;
6     RSTM = 0 ;
7     RSTP = 0 ;
8     RSTCARRYIN = 0 ;
9     RSTOPMODE = 0 ;
10    // assert all clock enable signals
11    CEA = 1 ;
12    CEB = 1 ;
13    CEC = 1 ;
14    CED = 1 ;
15    CEM = 1 ;
16    CEP = 1 ;
17    CEOPMODE = 1 ;
18    CECARRYIN = 1 ;
19
20    //Verify DSP Path 1
21    opmode = 8'b11011101 ;
22    A = 20 ;
23    B = 10 ;
24    C = 350 ;
25    D = 25 ;

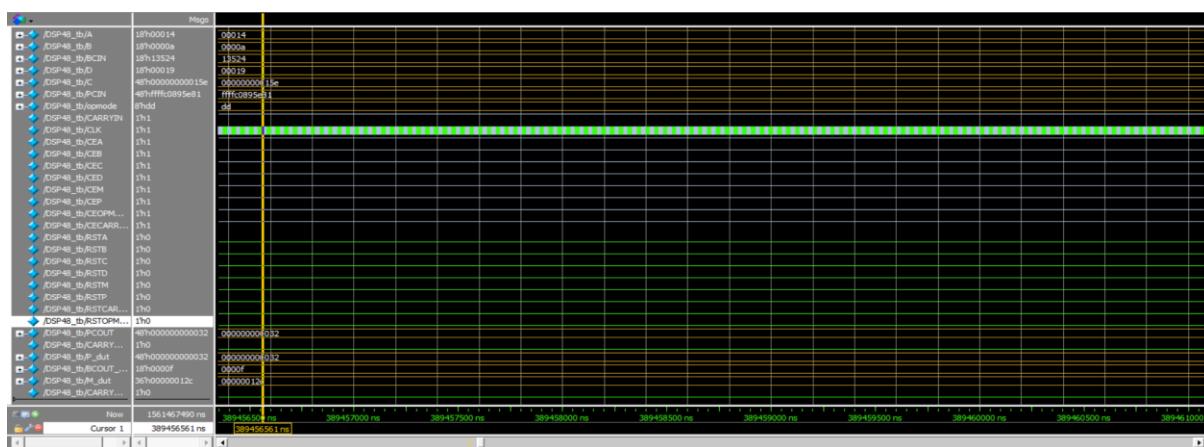
```

```

1 BCIN = $random ;
2          PCIN = $random ;
3          CARRYIN = $random ;
4          repeat(4) @(posedge CLK);
5          if ( P_dut != 'h32) begin
6              $display ("PATH_1 'P' FAILED");
7              $stop;
8          end
9          else if (BCOUT_dut != 'hf) begin
10             $display ("PATH_1 'BCOUT' FAILED");
11             $stop;
12         end
13         else if (CARRYOUT_dut !=0) begin
14             $display ("PATH_1 'CARRYOUT' FAILED");
15             $stop;
16         end
17         else if (M_dut != 'h12c ) begin
18             $display ("PATH_1 'M' FAILED");
19             $stop;
20         end
21         else begin
22             $display ("PATH_1 passed");
23         end
24     end

```

Results:



```
+ M_dut=0000000000, BCOUT_dut=0000a, CARRYOUT_dut=0, P_dut=00000000000000  
+ M_dut=00000000c8, BCOUT_dut=0000f, CARRYOUT_dut=0, P_dut=000000000015e  
+ M_dut=00000012c, BCOUT_dut=0000f, CARRYOUT_dut=0, P_dut=0000000000096  
+ M_dut=00000012c, BCOUT_dut=0000f, CARRYOUT_dut=0, P_dut=0000000000032  
+ PATH 1 passed
```

2.3. Verify DSP Path 2

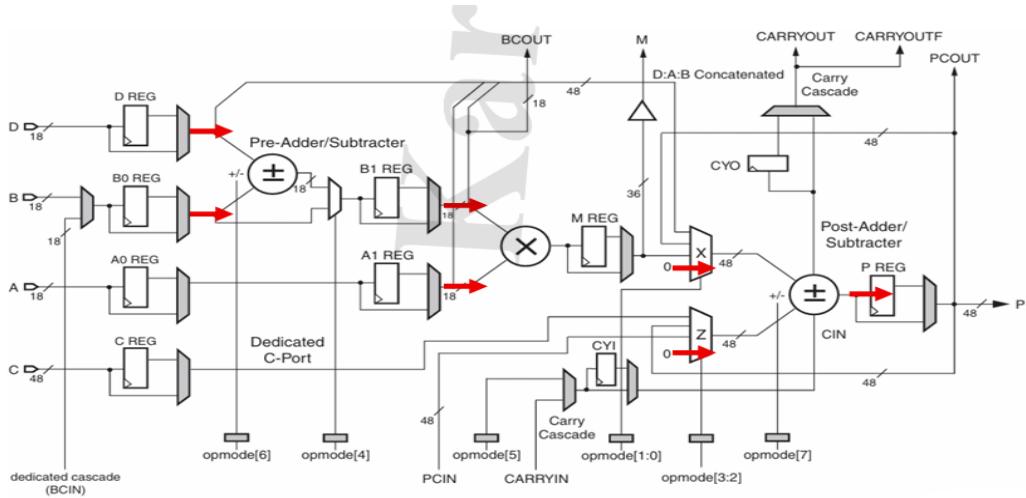


Figure 2: Path 2 Data Flow

Code:

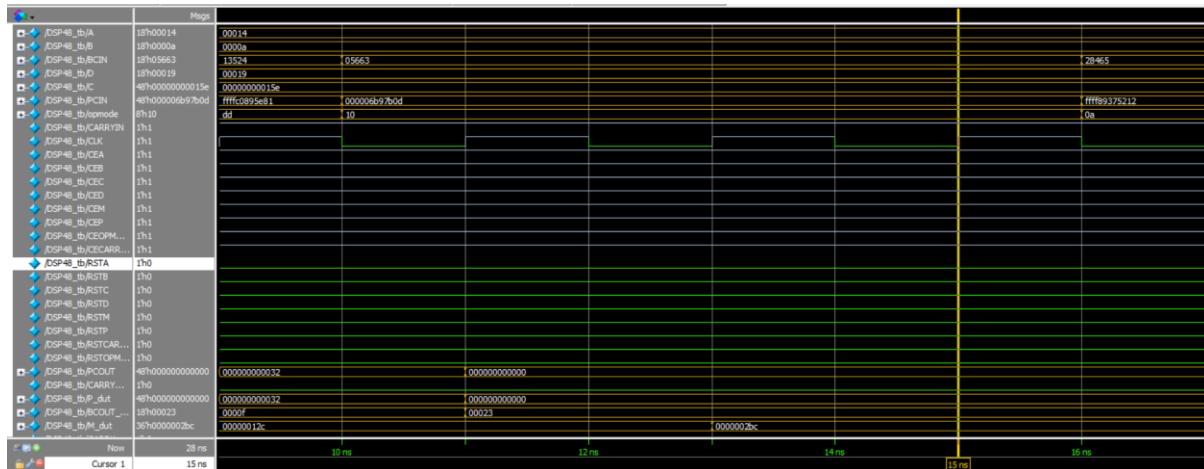
```

1 // 2.3. Verify DSP Path 2
2     opmode = 8'b00010000 ;
3     A = 20 ;
4     B = 10 ;
5     C = 350 ;
6     D = 25 ;
7
8     BCIN = $random ;
9     PCIN = $random ;
10    CARRYIN = $random ;
11    repeat(3) @(`negedge CLK);
12    if ( P_dut != 'h0) begin
13        $display ("PATH_2 'P' FAILED");
14        $stop;
15    end
16    else if (BCOUT_dut != 'h23) begin
17        $display ("PATH_2 'BCOUT' FAILED");
18        $stop;
19    end
20    else if (CARRYOUT_dut !=0) begin
21        $display ("PATH_2 'CARRYOUT' FAILED");
22        $stop;
23    end
24    else if (M_dut != 'h2bc ) begin
25        $display ("PATH_2 'M' FAILED");
26        $stop;
27    end
28    else begin
29        $display ("PATH_2 passed");
30    end

```

Results:

```
# M_dut=00000012c, BCOUT_dut=00023, CARRYOUT_dut=0, P_dut=00000000000000
# M_dut=0000002bc, BCOUT_dut=00023, CARRYOUT_dut=0, P_dut=00000000000000
# PATH_2 passed
```



2.4. Verify DSP Path 3

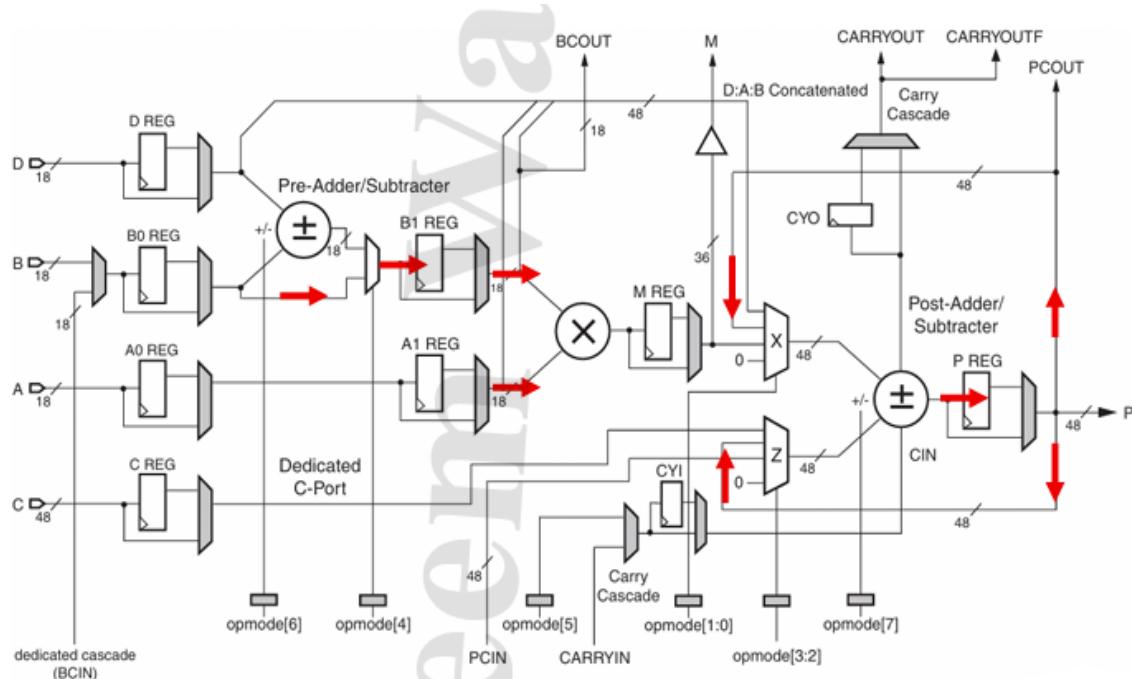


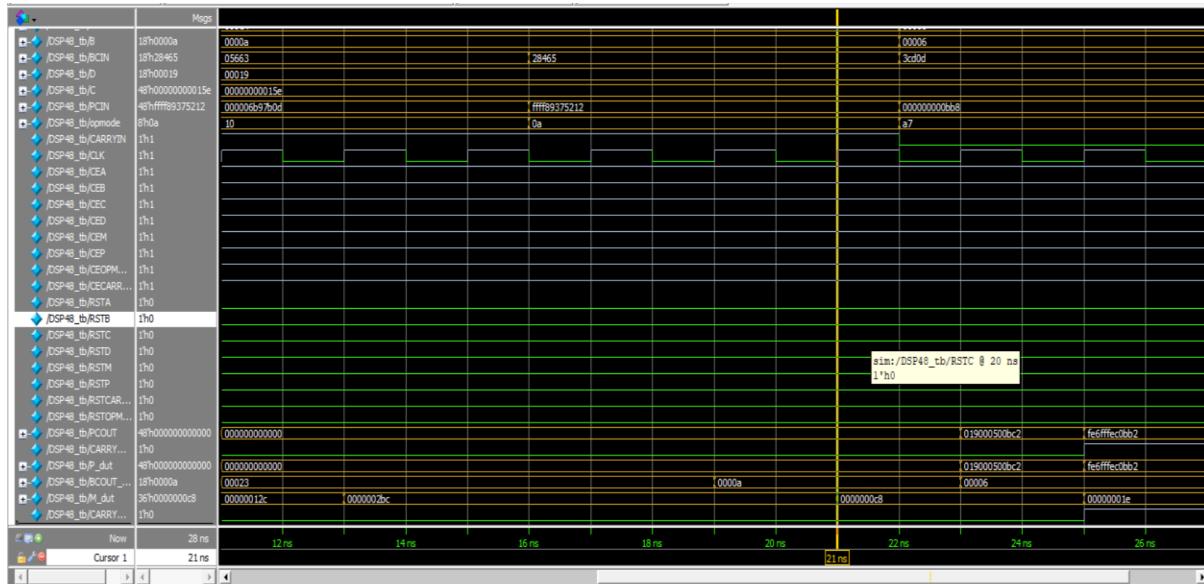
Figure 3: Path 3 Data Flow

Code:

```
1 // 2.4. Verify DSP Path 3
2         opmode = 8'b00001010 ;
3         A = 20 ;
4         B = 10 ;
5         C = 350 ;
6         D = 25 ;
7
8         BCIN = $random ;
9         PCIN = $random ;
10        CARRYIN = $random ;
11        repeat(3) @(negedge CLK);
12        if ( P_dut != PCOUT) begin
13            $display ("PATH_3 'P' FAILED");
14            $stop;
15        end
16        else if (BCOUT_dut != 'ha) begin
17            $display ("PATH_3 'BCOUT' FAILED");
18            $stop;
19        end
20        else if (CARRYOUT_dut != CARRYOUTF) begin
21            $display ("PATH_3 'CARRYOUT' FAILED");
22            $stop;
23        end
24        else if (M_dut != 'hc8 ) begin
25            $display ("PATH_3 'M' FAILED");
26            $stop;
27        end
28        else begin
29            $display ("PATH_3 passed");
30        end
```

Results:

```
# M_dut=0000002bc, BCOUT_dut=0000a, CARRYOUT_dut=0, P_dut=00000000000000
# M_dut=0000000c8, BCOUT_dut=0000a, CARRYOUT_dut=0, P_dut=00000000000000
# PATH_3 passed
```



2.5. Verify DSP Path 4

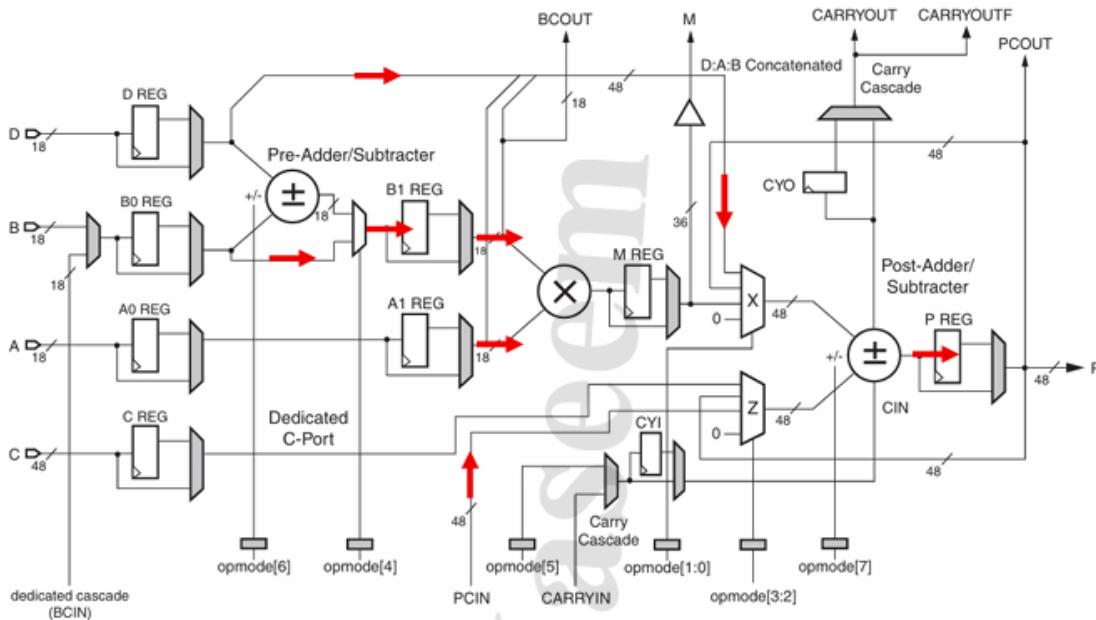


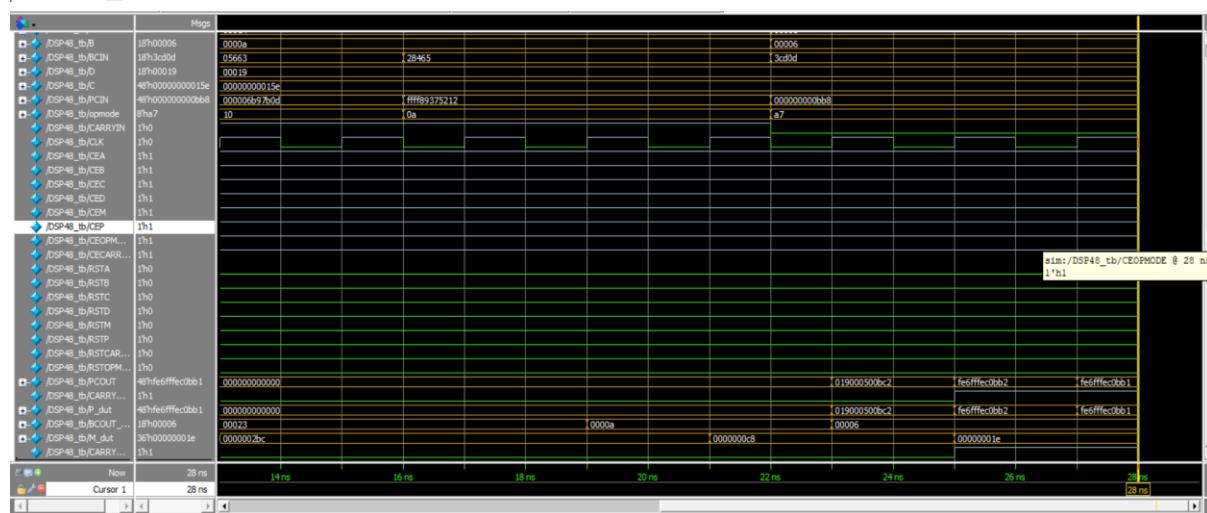
Figure 4: Path 4 Data Flow

Code:

```
1 // 2.5. Verify DSP Path 4
2     opmode = 8'b10100111 ;
3     A = 5 ;
4     B = 6 ;
5     C = 350 ;
6     D = 25 ;
7     PCIN = 3000 ;
8
9     BCIN = $random ;
10    CARRYIN = $random ;
11    repeat(3) @ (negedge CLK);
12        if ( P_dut != 'hfe6ffffec0bb1) begin
13            $display ("PATH_4 'P' FAILED");
14            $stop;
15        end
16        else if (BCOUT_dut != 'h6) begin
17            $display ("PATH_4 'BCOUT' FAILED");
18            $stop;
19        end
20        else if (CARRYOUT_dut != 1) begin
21            $display ("PATH_4 'CARRYOUT' FAILED");
22            $stop;
23        end
24        else if (M_dut != 'h1e ) begin
25            $display ("PATH_4 'M' FAILED");
26            $stop;
27        end
28        else begin
29            $display ("PATH_4 passed");
30        end
31    end
32
33    initial begin
34        $monitor ("M_dut=%h, BCOUT_dut=%h, CARRYOUT_dut=%h, P_dut=%h", M_dut, BCOUT_dut, CARRYOUT_dut, P_dut);
35    end
36
37 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=fe6ffffec0bb2
# M_dut=00000001e, BCOUT_dut=00006, CARRYOUT_dut=1, P_dut=fe6ffffec0bb1
# PATH_4 passed
```



Do file:

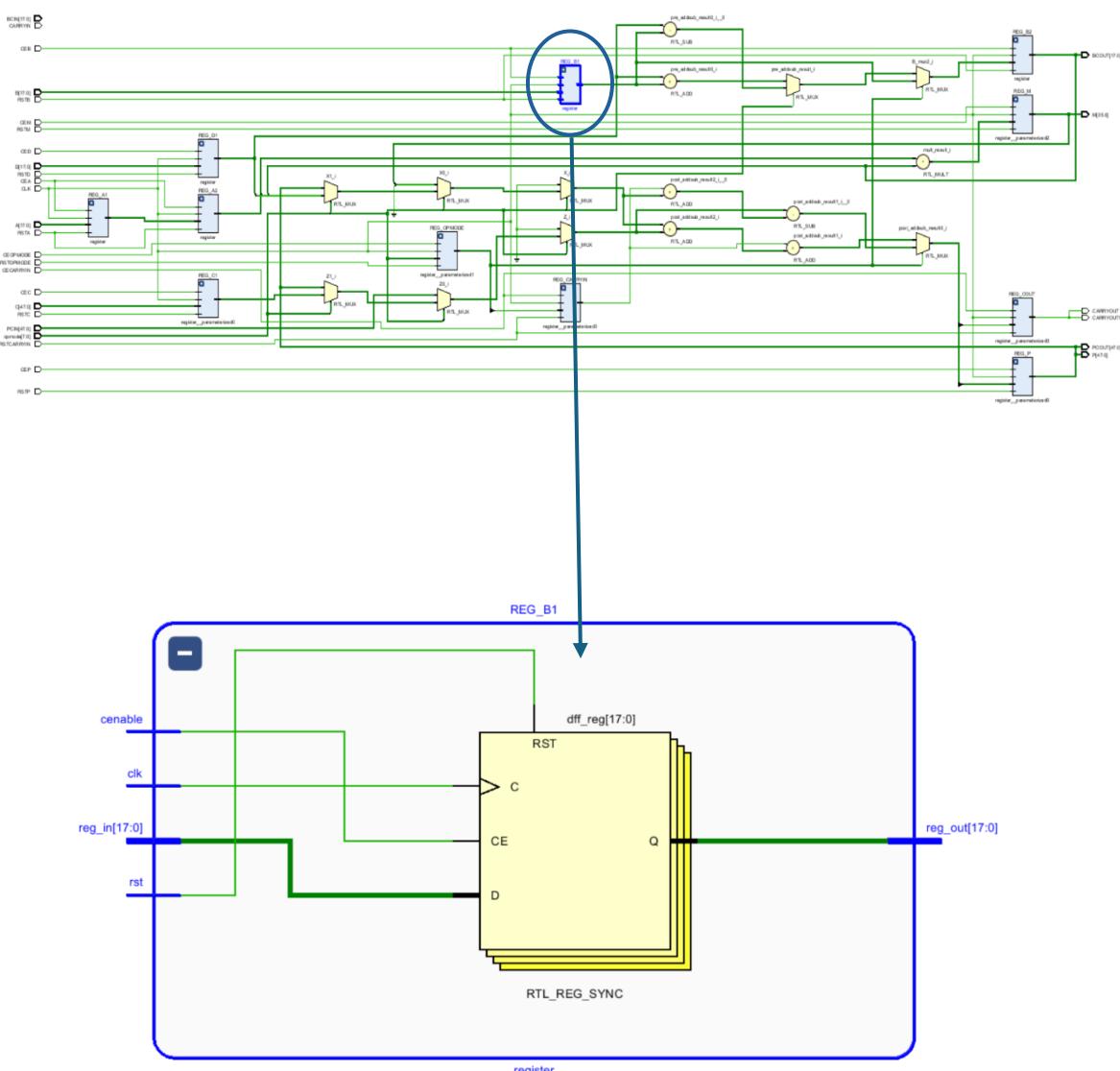
```
1 vlib work
2 vlog DSP48.v DSP48_tb.v
3 vsim -voptargs+=acc work.DSP48_tb
4 add wave *
5 run -all
6 #quit -sim
```

Constraints file:

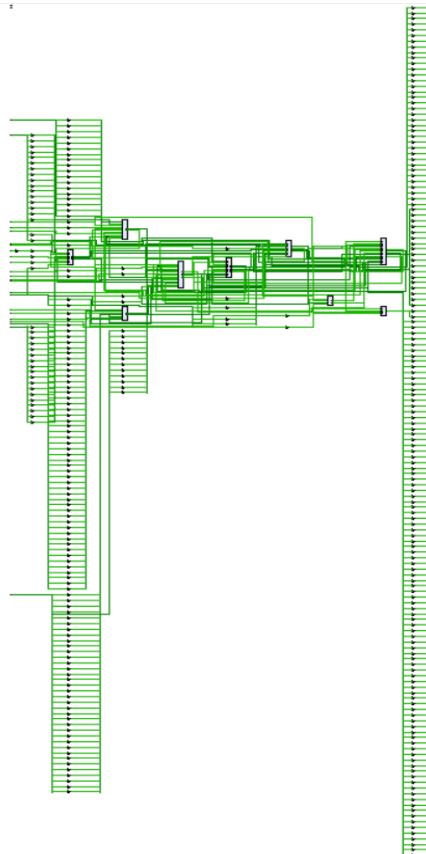
```
1 ## Clock signal
2 set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVC MOS33} [get_ports CLK]
3 create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports CLK]
4
5
```

Elaboration:

Schematic snippets:



Synthesis schematic:

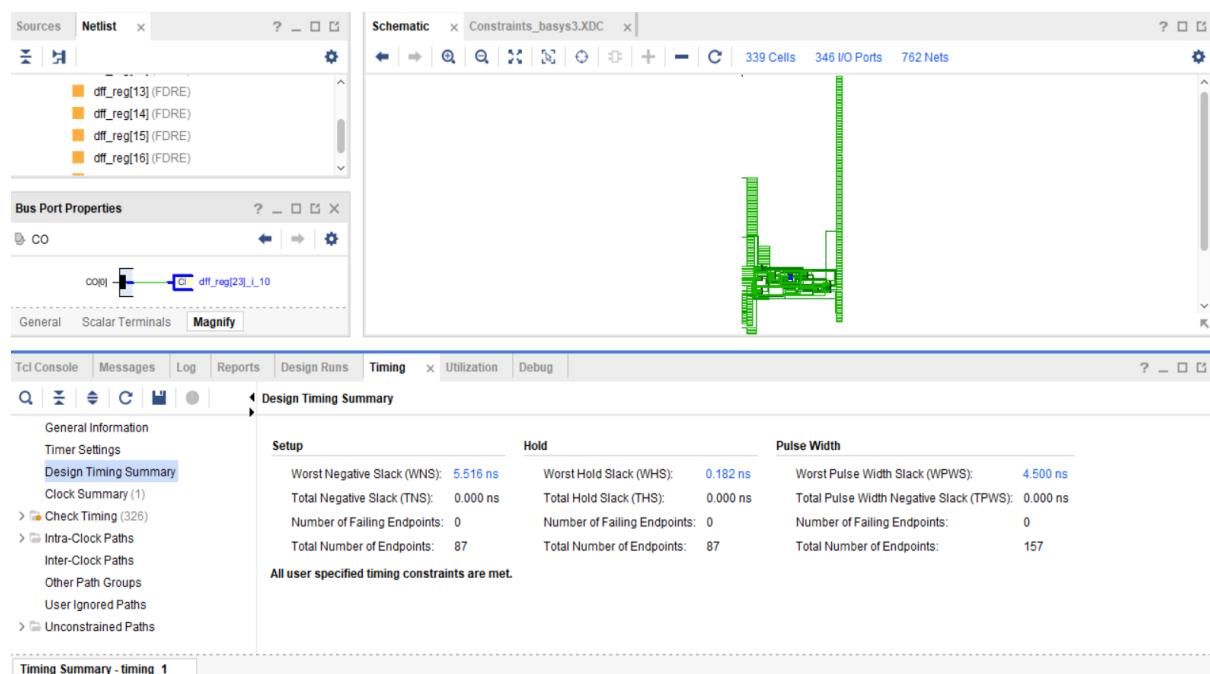


Synthesis Messages:

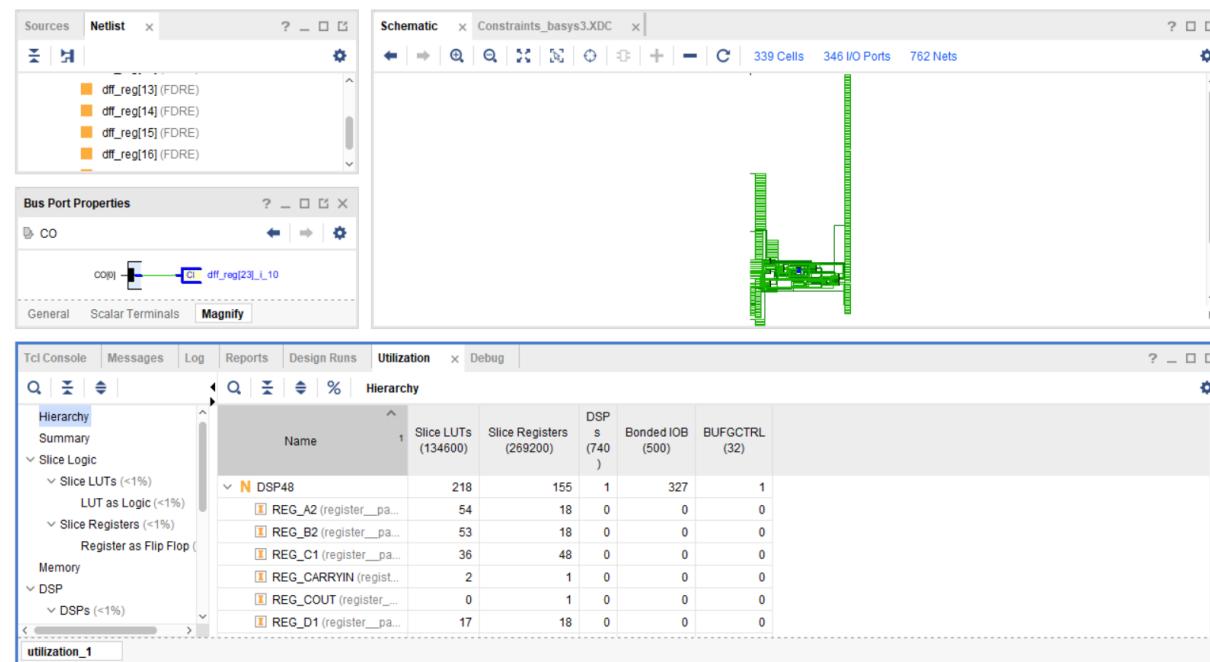
The screenshot shows the Synthesis tool interface with three main tabs:

- Schematic**: Displays the synthesized logic circuit.
- Netlist**: Shows a list of components used in the design, such as `dff_reg[13]`, `dff_reg[14]`, `dff_reg[15]`, and `dff_reg[16]`.
- Messages**: Displays synthesis warnings, infos, and status messages. Key messages include:
 - Command: `synth_design -top DSP48 -part xc7a200tfg1156-3`
 - [Common 17-349] Got license for feature 'Synthesis' and/or device 'xc7a200t'
 - [Synth 8-2490] overwriting previous definition of module `DSP48 [DSP48.v48]`
 - [Synth 8-6157] synthesizing module 'DSP48' [`DSP48.v48`] (6 more like this)
 - [Synth 8-6155] done synthesizing module 'register' (#1) [`DSP48.v1`] (6 more like this)
 - [Synth 8-3331] design register has unconnected port `clk` (40 more like this)
 - [Device 21-403] Loading part `xc7a200tfg1156-3`
 - [Project 1-236] Implementation specific constraints were found while reading constraint file [D:/digital diploma/project 1/mvado/Constraints_basys3.XDC]. These constraints will be ignored for synthesis but will be used in implementation. Impacted constraints are listed in the file [XIL/DSP48_proplmpl.xdc]. Resolution: To avoid this warning, move constraints listed in [Undefined] to another XDC file and exclude this new file from synthesis with the `used_in_synthesis` property (File Properties dialog in GUI) and re-run elaboration/synthesis.
 - [Synth 8-5818] HDL ADVISOR - The operator resource <operator> is shared. To prevent sharing consider applying a `KEEP` on the output of the operator [`DSP48.v:143`] (1 more like this)

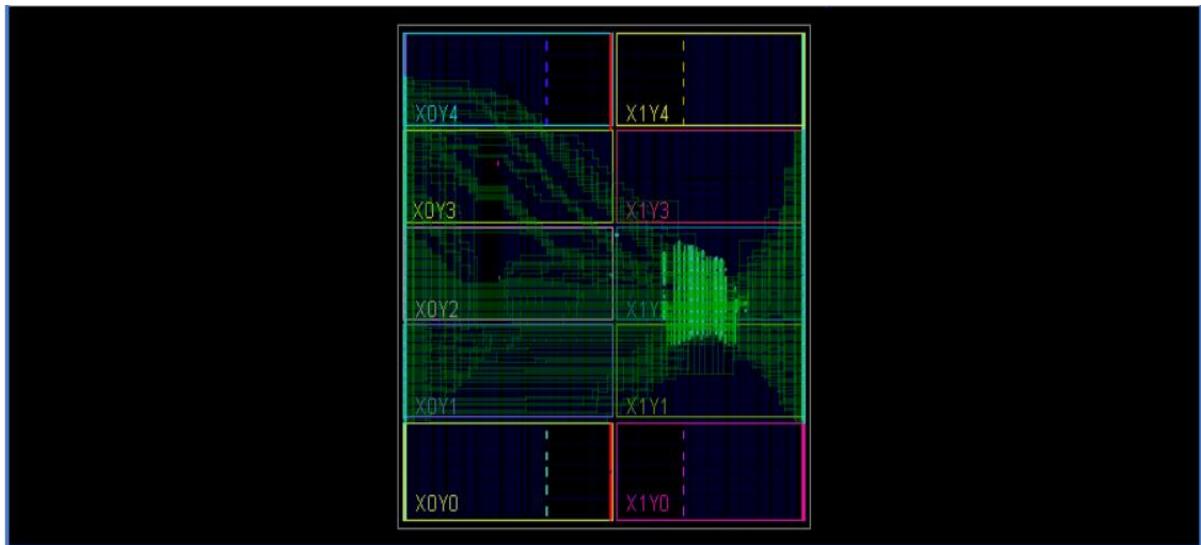
Synthesis Timing summary:



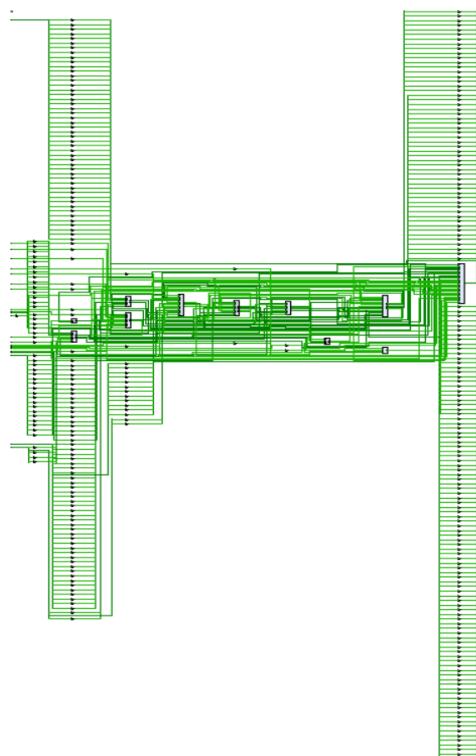
Synthesis Utilization report:



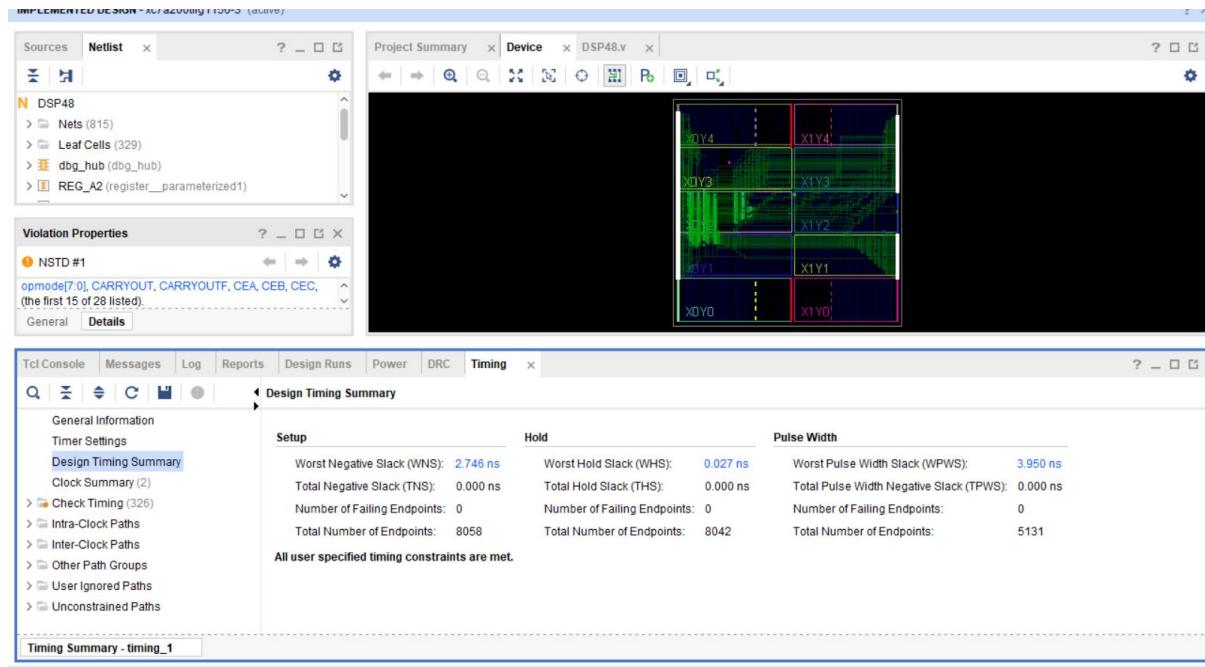
Device snippets:



Implementation schematic:



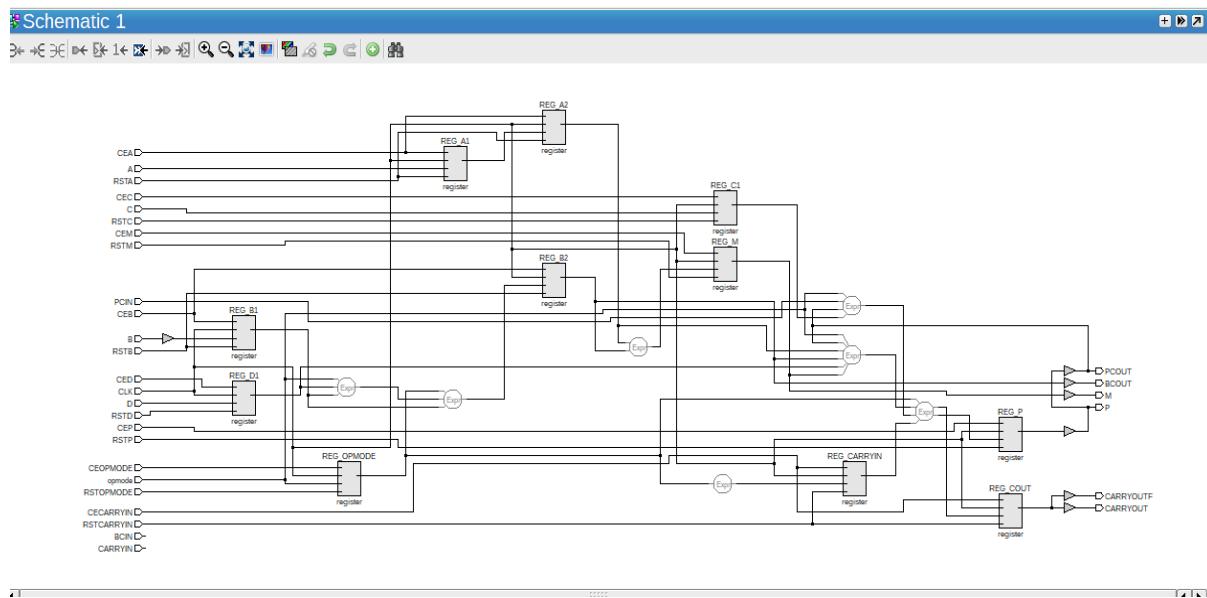
Implementation timing summary:



Implementation Utilization report:

Name	Slice LUTs (133800)	Slice Registers (267600)	F7 Muxes (66900)	F8 Muxes (33450)	Slice (33450)	LUT as Logic (133800)	LUT as Memory (46200)	LUT Flip Flop Pairs (133800)	Block RAM Tile (365)	DSPs (740)
N DSP48	2649	4220	102	7	1332	2175	474	1598	8	1
> dbg_hub (dbg_hub)	475	727	0	0	249	451	24	319	0	0
REG_A2 (register_parameterized1)	52	18	0	0	20	52	0	0	0	0
REG_B2 (register_parameterized1)	55	18	0	0	26	55	0	0	0	0
REG_C1 (register_parameterized1)	36	48	0	0	29	36	0	0	0	0
REG_CARRYIN (register_parameterized1)	2	1	0	0	2	2	0	1	0	0
REG_COUT (register_parameterized1)	0	1	0	0	1	0	0	0	0	0
REG_D1 (register_parameterized1)	33	18	0	0	16	33	0	0	0	0

LINT Schematic:



Snippets showing no errors:

```
D:/lint/DSP48.v [DSP48]
104 );
105
106
107 // B pipeline 1st stage
108 wire [17:0] B_stage1 ;
109 wire [17:0] B_mux ;
110 assign B_mux = ((B_INPUT == 1'b1) ? B : BCIN) ;
111 register #( .WIDTH(18), .A0REG(B0REG), .RSTTYPE(RSTTYPE))
112 (
113     .clk(CLK), .rst(RSTB), .cenable(CEB), .reg_in(B_mux),
114 );
115
116
117 // C pipeline 1st stage
118 wire [47:0] C_stage1 ;
119 register #( .WIDTH(48), .A0REG(CREG), .RSTTYPE(RSTTYPE))
120 (
121     .clk(CLK), .rst(RSTC), .cenable(CEC), .reg_in(C),
122 );
123
124
125 // D pipeline 1st stage
126 wire [17:0] D_stage1 ;
127 register #( .WIDTH(18), .A0REG(DREG), .RSTTYPE(RSTTYPE))
128 (
```

Name	Count
Open(uninspected, pending)	6
Info	6