DSP48A1

by Ahmed Khalaf Mohamed Ali

Under the guidance of **Eng. Kareem Waseem**

OVERVIEW

Description

This project centers on the design and implementation of a DSP48A1 slice within a Spartan-6 FPGA, aimed at supporting computation-heavy applications. The DSP48A1 slice is a specialized hardware unit designed to handle complex digital signal processing (DSP) functions such as multiplication, addition, and accumulation. It is particularly well-suited for high-performance tasks requiring efficient arithmetic operations.

Key Features

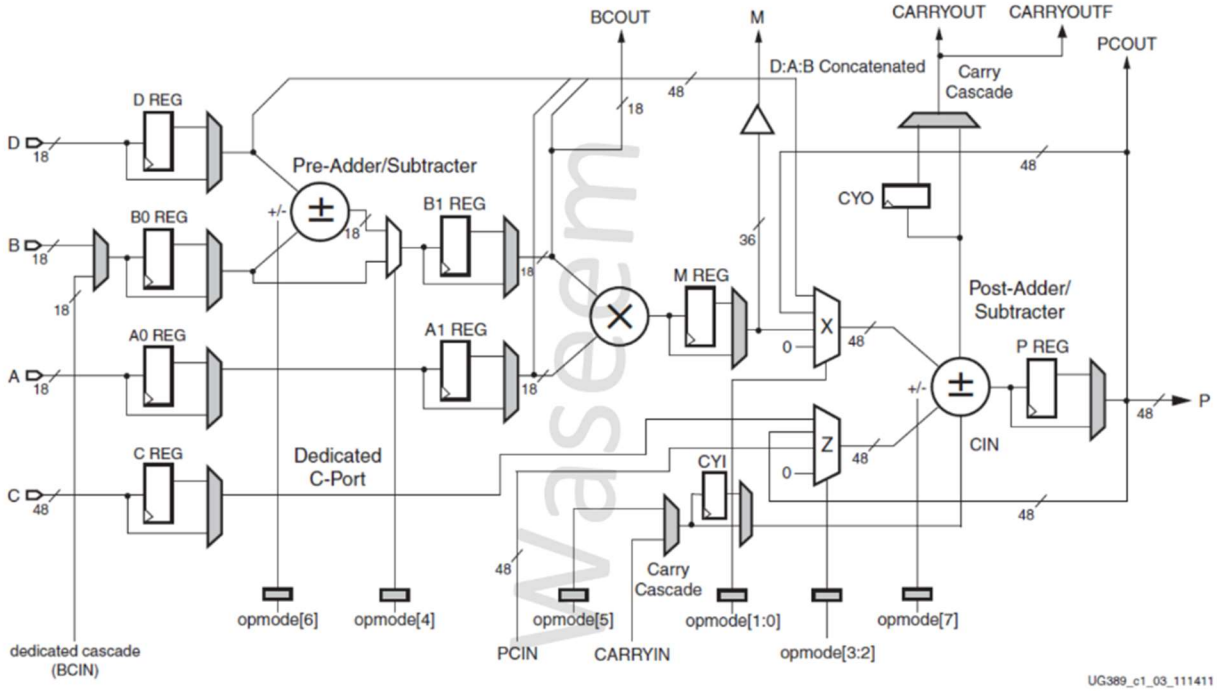
Optimized DSP48A1 Slice Implementation: The project delivers an optimized implementation of the DSP48A1 slice within the Spartan-6 FPGA. By incorporating advanced pipelining techniques, the design significantly enhances processing speed and efficiency for complex arithmetic computations, making it ideal for intensive digital signal processing tasks.

Advanced Vivado Workflow: Leveraging the Vivado design suite, the project follows a rigorous development workflow. This includes detailed elaboration, synthesis, and implementation phases, which ensure that the design meets high standards of reliability and performance, while also minimizing errors through exhaustive simulation and analysis.

Comprehensive Testbench Strategy: The project employs a robust testbench methodology, utilizing sophisticated test patterns to thoroughly verify the functionality of the DSP48A1 slice. This includes testing for accuracy in arithmetic operations, waveform integrity, and meeting timing constraints across various operational scenarios.

Highly Adaptive Configuration Options: The DSP48A1 slice design is engineered for flexibility, featuring a wide range of configurable parameters. Users can adjust pipeline stages and control registers, including AOREG, A1REG, BOREG, and B1REG, to tailor the design to meet specific computational needs. This adaptability makes the design versatile across different application domains, from basic arithmetic to complex DSP operations.

DSP BLOCK DIGRAM



DSP48A1 BLOCK

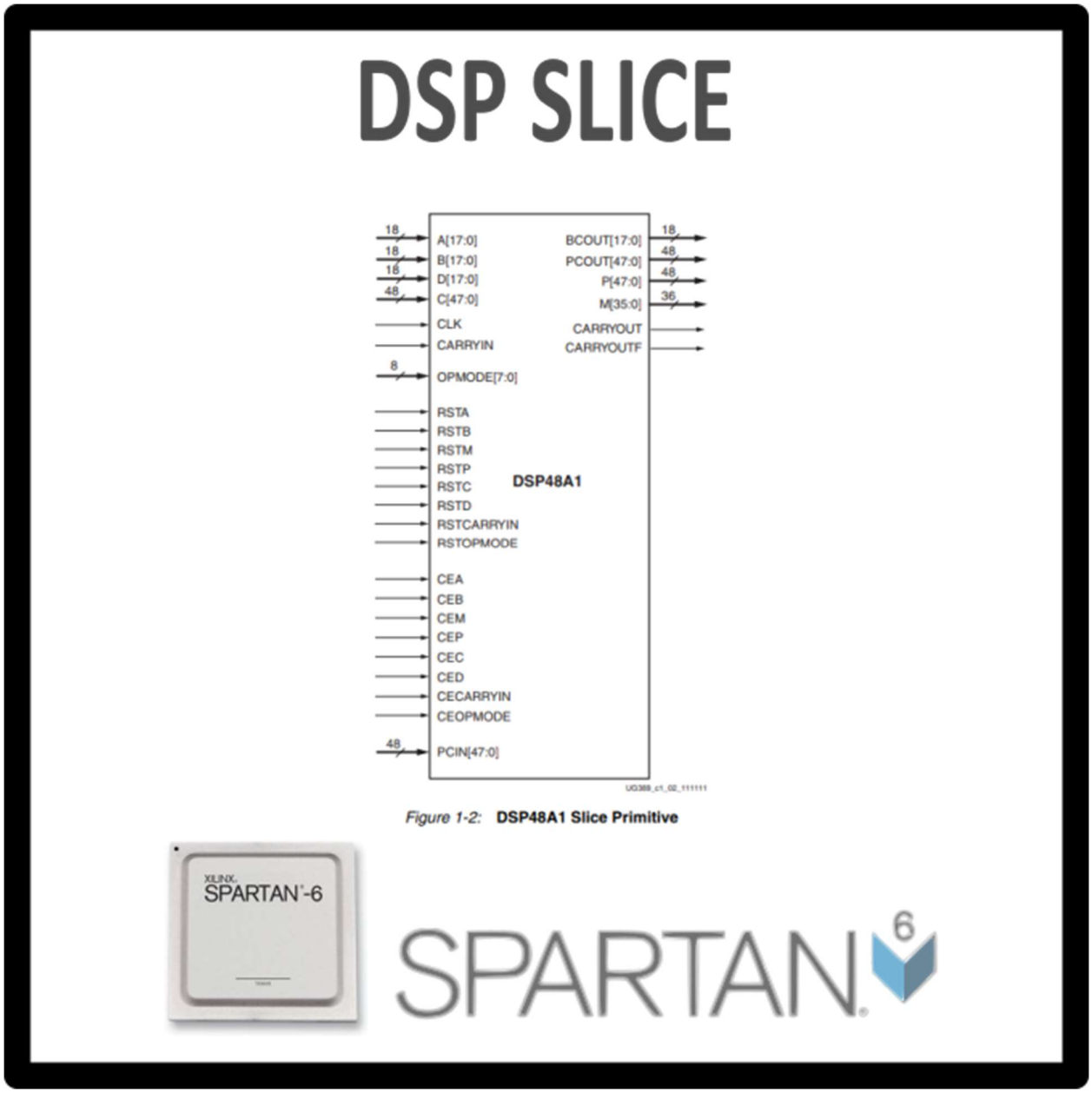


Figure 1-2: **DSP48A1 Slice Primitive**

DSP48A1:

I make this project with structural modeling Make instance to each block. i will insert all instance codes in end of report

```
module DSP_Spartn_six(CLK,OPMODE,A,B,BCIN,PCIN,C,D,CARRYIN,RSTA, RSTB, RSTM, RSTP, RSTC, RSTD, RSTCARRYIN, RSTOPMODE,CEA, CEB, CEM, CEP, CEC,
CED, CECARRYIN, CEOPMODE,BCOUT,PCOUT,P,M,CARRYOUT,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 = "OPMODE5";
parameter B_INPUT = "DIRECT";
parameter RSTTYPE = "SYNC";
    input CLK;
    input [7:0] OPMODE;
    input [17:0] A;
    input [17:0] B;
    input [17:0] BCIN;
    input [47:0] PCIN;
    input [47:0] C;
    input [17:0] D;
    input CARRYIN;
    input RSTA, RSTB, RSTM, RSTP, RSTC, RSTD, RSTCARRYIN, RSTOPMODE;
    input CEA, CEB, CEM, CEP, CEC, CED, CECARRYIN, CEOPMODE;
    output [17:0] BCOUT;
    output [47:0] PCOUT;
    output [47:0] P;
    output [35:0] M;
    output CARRYOUT;
    output CARRYOUTF;
```

```
wire [17:0] Q_A0REG,Q_A1REG;
wire [17:0] Q_DREG;
wire [47:0] Q_CREG;
wire [17:0] Q_B0REG,Q_B1REG;
wire [7:0] Q_OPMODEREG;
wire [17:0] pre_add_sub;
wire [17:0] pre_add_sub_mux;
wire [35:0] MUl_REG;
wire [35:0] M_REG;
wire [47:0] MUX_REGX;
wire [47:0] MUX_REGZ;
wire mux_carryin;
wire MUX_CARRYIN_REG;
wire [47:0]P_sum;
wire P_carry;
wire [47:0]P_sum_reg;
wire P_carry_reg;
```

```
    MUX_DFF #(.DFF_SELECT(A0REG),.RSTTYPE(RSTTYPE),.DFF_SIZE(18)) A0REG_DESIGN (.clk(CLK),.rst(RSTA),.enable(CEA),.PIN_IN(A),.PIN_OUT(Q_A0REG));
    MUX_DFF #(.DFF_SELECT(DREG),.RSTTYPE(RSTTYPE),.DFF_SIZE(18)) DREG_DESIGN (.clk(CLK),.rst(RSTD),.enable(CED),.PIN_IN(D),.PIN_OUT(Q_DREG));
    MUX_DFF #(.DFF_SELECT(CREG),.RSTTYPE(RSTTYPE),.DFF_SIZE(48)) CREG_DESIGN (.clk(CLK),.rst(RSTC),.enable(CEC),.PIN_IN(C),.PIN_OUT(Q_CREG));
    MUX_DFF_B #(.DFF_SELECT(B0REG),.RSTTYPE(RSTTYPE),.DFF_SIZE(18),.B_INPUT(B_INPUT)) B0REG_DESIGN (.clk(CLK),.rst(RSTB),.enable(CEB),.PIN_IN(B),.
    CASCADE(BCIN),.PIN_OUT(Q_B0REG));
    MUX_DFF #(.DFF_SELECT(OPMODEREG),.RSTTYPE(RSTTYPE),.DFF_SIZE(8)) OPCODEREG_DESIGN (.clk(CLK),.rst(RSTOPMODE),.enable(CEOPMODE),.PIN_IN
    (OPMODE),.PIN_OUT(Q_OPMODEREG));
    N_BIT_ADDER #(.N(18)) ADDER (.D(Q_DREG),.B(Q_B0REG),.SEL(Q_OPMODEREG[6]),.C(pre_add_sub));
    mux2X1  #(.N(18)) mux2x1_design(.D0(Q_B0REG),.D1(pre_add_sub),.SEL(Q_OPMODEREG[4]),.Y(pre_add_sub_mux));
    MUX_DFF #(.DFF_SELECT(A1REG),.RSTTYPE(RSTTYPE),.DFF_SIZE(18)) A1REG_DESIGN (.clk(CLK),.rst(RSTA),.enable(CEA),.PIN_IN(Q_A0REG),.PIN_OUT
    (Q_A1REG));
    MUX_DFF #(.DFF_SELECT(B1REG),.RSTTYPE(RSTTYPE),.DFF_SIZE(18)) B1REG_DESIGN (.clk(CLK),.rst(RSTB),.enable(CEB),.PIN_IN(pre_add_sub_mux),.
    PIN_OUT(Q_B1REG));
    N_BIT_Multiplier #(.N(18)) N_BIT_Multiplier_design (.D(Q_B1REG),.B(Q_A1REG),.Out(MUl_REG));
    MUX_DFF #(.DFF_SELECT(MREG),.RSTTYPE(RSTTYPE),.DFF_SIZE(36)) MREG_DESIGN (.clk(CLK),.rst(RSTM),.enable(CEM),.PIN_IN(MUl_REG),.PIN_OUT(M_REG));
    N_BIT_MUX4X1 #(.N(48)) N_BIT_MUX4X1_design(.D0(48'b0),.D1({12'b0,M_REG}),.D2(P),.D3({Q_DREG[11:0],Q_A1REG,Q_B1REG}),.S0(Q_OPMODEREG[0]),.S1
    (Q_OPMODEREG[1]),.Y(MUX_REGX));
    N_BIT_MUX4X1 #(.N(48)) N_BIT_MUX4X1_design2(.D0(48'b0),.D1(PCIN),.D2(P),.D3(Q_CREG),.S0(Q_OPMODEREG[2]),.S1(Q_OPMODEREG[3]),.Y(MUX_REGZ));
    MUX_CARRYIN #(.DFF_SIZE(1),.CARRY_INPUT(CARRYINSEL)) MUX_CARRYIN_DESIGN (.CASCADE(CARRYIN),.PIN_IN(Q_OPMODEREG[5]),.PIN_OUT(mux_carryin));
    MUX_DFF #(.DFF_SELECT(CARRYINREG),.RSTTYPE(RSTTYPE),.DFF_SIZE(1)) CARRYIN_DESIGN (.clk(CLK),.rst(RSTCARRYIN),.enable(CECARRYIN),.PIN_IN
    (mux_carryin),.PIN_OUT(MUX_CARRYIN_REG));
    FUll_adder_con #(.N(48)) ADDER2 (.A(MUX_REGZ),.B(MUX_REGX),.SEL(Q_OPMODEREG[7]),.CIN(MUX_CARRYIN_REG),.carry(P_carry),.sum(P_sum));
    MUX_DFF #(.DFF_SELECT(PREG),.RSTTYPE(RSTTYPE),.DFF_SIZE(48)) PREG_DESIGN (.clk(CLK),.rst(RSTP),.enable(CEP),.PIN_IN(P_sum),.PIN_OUT
    (P_sum_reg));
    MUX_DFF #(.DFF_SELECT(CARRYOUTREG),.RSTTYPE(RSTTYPE),.DFF_SIZE(1)) CARRYOUTREG_DESIGN (.clk(CLK),.rst(RSTCARRYIN),.enable(CECARRYIN),.PIN_IN
    (P_carry),.PIN_OUT(P_carry_reg));
assign BCOUT=Q_B1REG;
assign PCOUT=P_sum_reg;
assign P=P_sum_reg;
assign M=M_REG;
assign CARRYOUT=P_carry_reg;
assign CARRYOUTF=CARRYOUT;
endmodule

```

```
`timescale 1ns / 1ps
module DSP_Spartn_six_tb();
  reg CLK;
  reg [7:0] OPMODE;
  reg [17:0] A;
  reg [17:0] B;
  reg [17:0] BCIN;
  reg [47:0] PCIN;
  reg [47:0] C;
  reg [17:0] D;
  reg CARRYIN;
  reg RSTA, RSTB, RSTM, RSTP, RSTC, RSTD, RSTCARRYIN, RSTOPMODE;
  reg CEA, CEB, CEM, CEP, CEC, CED, CECARRYIN, CEOPMODE;
  wire [17:0] BCOUT;
  wire [47:0] PCOUT;
  wire [47:0] P;
  wire [35:0] M;
  wire CARRYOUT;
  wire CARRYOUTF;
```

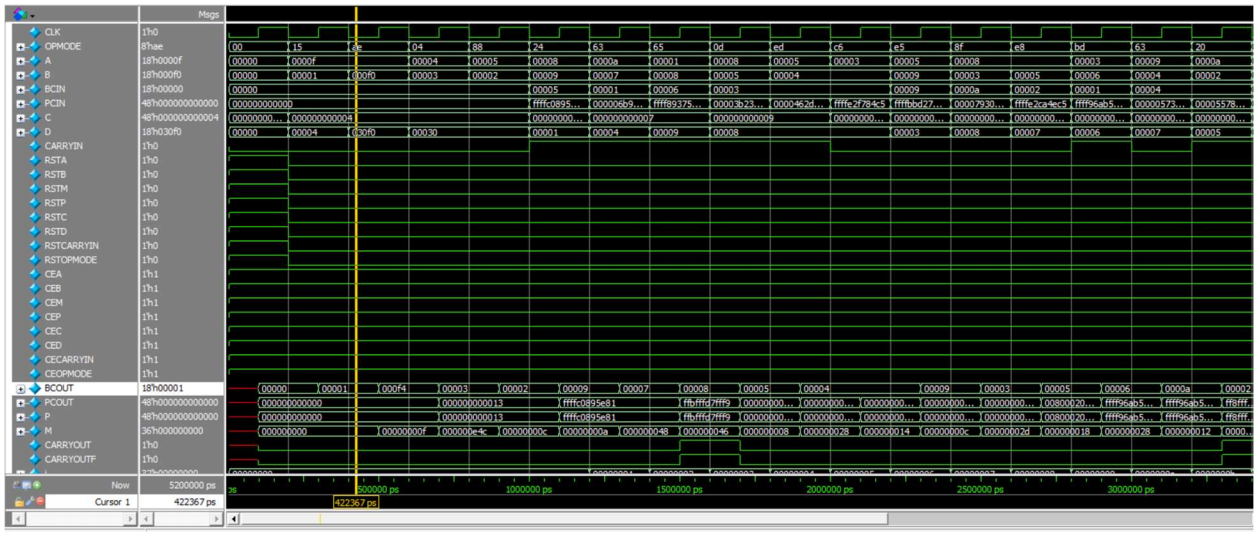
```
20  // Instantiate the DSP48A1 module
21  DSP_Spartn_six DSP_instance (
22    .CLK(CLK),
23    .OPMODE(OPMODE),
24    .A(A),
25    .B(B),
26    .BCIN(BCIN),
27    .PCIN(PCIN),
28    .C(C),
29    .D(D),
30    .CARRYIN(CARRYIN),
31    .RSTA(RSTA),
32    .RSTB(RSTB),
33    .RSTM(RSTM),
34    .RSTP(RSTP),
35    .RSTC(RSTC),
36    .RSTD(RSTD),
37    .RSTCARRYIN(RSTCARRYIN),
38    .RSTOPMODE(RSTOPMODE),
39    .CEA(CEA),
40    .CEB(CEB),
41    .CEM(CEM),
42    .CEP(CEP),
43    .CEC(CEC),
44    .CED(CED),
45    .CECARRYIN(CECARRYIN),
46    .CEOPMODE(CEOPMODE),
47    .BCOUT(BCOUT),
48    .PCOUT(PCOUT),
49    .P(P),
50    .M(M),
51    .CARRYOUT(CARRYOUT),
52    .CARRYOUTF(CARRYOUTF)
53  );
```

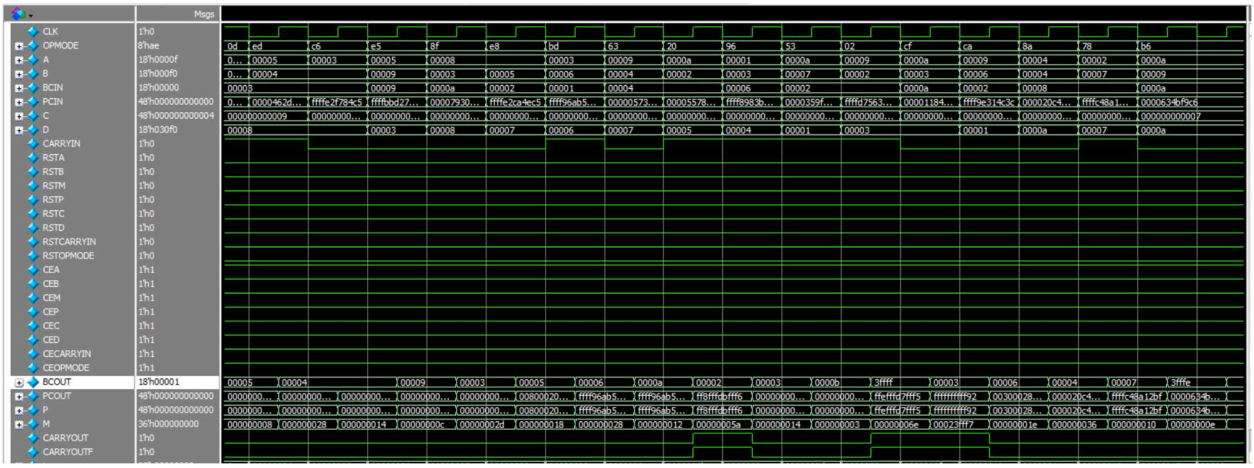
```
  integer i=0;
integer TC=200;
// Clock generation
 initial begin
  CLK = 0;
  forever #(TC/2) CLK = ~CLK;
 end
 initial begin
   OPMODE = 8'b00000000;
   A = 18'b000000000000000000;
   B = 18'b000000000000000000;
   BCIN = 18'b000000000000000000;
   PCIN = 48'b000000000000000000000000000000000000000000000000;
   C = 48'b000000000000000000000000000000000000000000000000;
   D = 18'b000000000000000000;
   CARRYIN = 0;
   RSTA = 1;
   RSTB = 1;
   RSTM = 1;
   RSTP = 1;
   RSTC = 1;
   RSTD = 1;
   RSTCARRYIN = 1;
   RSTOPMODE = 1;
   CEA = 1;
   CEB = 1;
   CEM = 1;
   CEP = 1;
   CEC = 1;
   CED = 1;
   CECARRYIN = 1;
   CEOPMODE = 1;

```

```
87      // Apply stimulus
88      @(negedge CLK)
89      RSTA = 0;
90      RSTB = 0;
91      RSTM = 0;
92      RSTP = 0;
93      RSTC = 0;
94      RSTD = 0;
95      RSTCARRYIN = 0;
96      RSTOPMODE = 0;
97      A = 18'b0000000000001111;
98       B = 18'b0000000000000001;
99       D = 18'b00_0000_0000_0000_0100;
100      C = 48'b000000000000000000000000000000000000000000000100;
101      OPMODE = 8'b00010101;
102
103
104     // Apply additional stimulus
105  @(negedge CLK) A = 18'b0000000000001111;
106   B = 18'b0000000011110000;
107   D = 18'b0011000011110000;
108   OPMODE = 8'b10101110;
109
110
111
112  @(negedge CLK)
113   A = 18'b0000000000000100;
114   B = 18'b0000000000000011;
115   D = 18'b0000000000110000;
116   OPMODE = 8'b00000100;
```

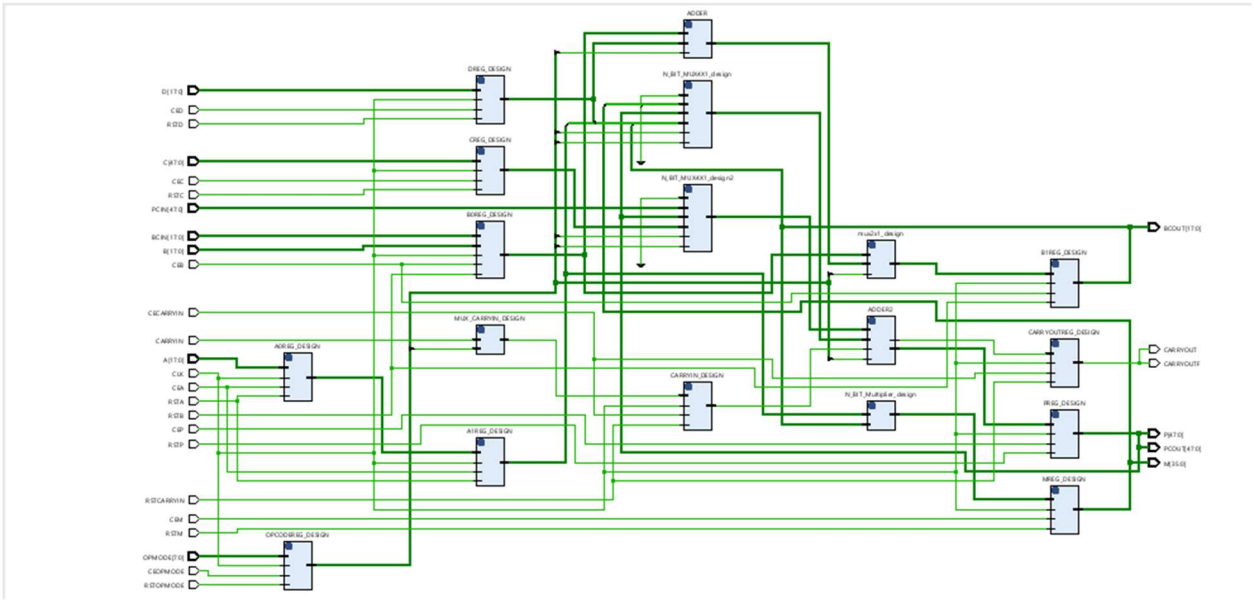
```
 OPMODE = 8'b00000100;
@(negedge CLK)
 A = 18'b000000000000101;
 B = 18'b000000000000010;
OPMODE = 8'b10001000;
 @(negedge CLK) ;
for (i=0; i<20 ; i=i+1) begin
    OPMODE = $random;
    A = $urandom_range(1,10);
    B = $urandom_range(1,10);
    BCIN = $urandom_range(1,10);
    PCIN = $random;
    C = $urandom_range(1,10);
    D = $urandom_range(1,10);
    CARRYIN = $random;
@(negedge CLK);
end
@(negedge CLK);
$stop;
end

endmodule
```

```
vlib work
vlog  DSP_Spartn_six.v ADDER_SUB.v  DSP_Spartn_six_tb.v N_BIT_Multiplier.v mux2X1.v N_BIT_MUX4X1.v MUX_CARRYIN.v MUX_DFF.v MUX_DFF_B.v FUll_adder_con.v
vsim -voptargs=+acc work.DSP_Spartn_six_tb
add wave *
run -all
#quit -sim
```

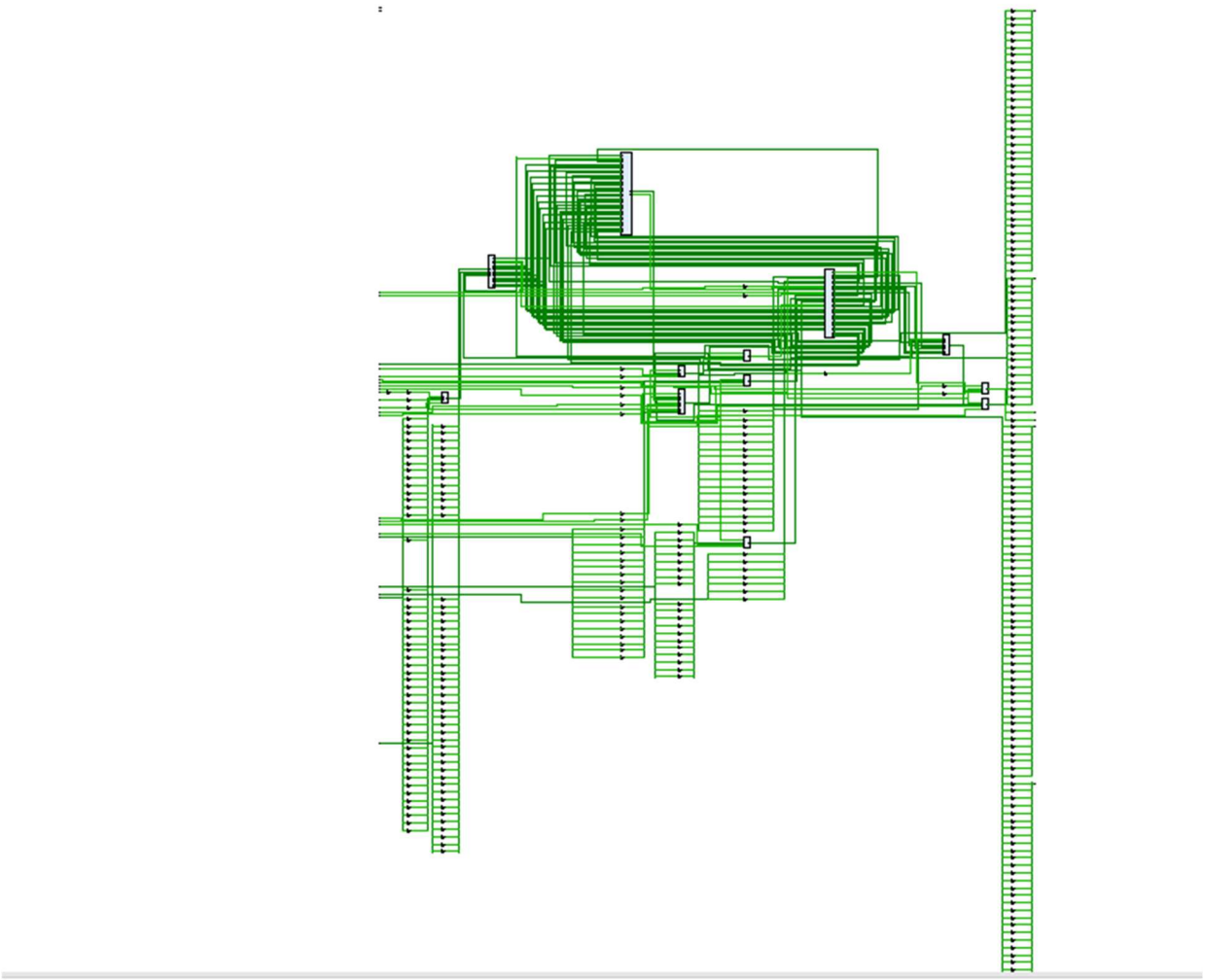

Msgs
CLK
OPMODE
A
B
BCIN
PCIN
C
D
CARRYIN
RSTA
RSTB
RSTM
RSTP
RSTC
RSTD
RSTCARRYIN
RSTOPMODE
CEA
CEB
CEM
CEP
CEC
CED
CECARRYIN
CEOPMODE
BCOUT
PCOUT
P
M
CARRYOUT
CARRYOUTF
Now
5200000 ps
Cursor 1
422367 ps
500000 ps
1000000 ps
1500000 ps
2000000 ps
2500000 ps
3000000 ps


Msgs
CLK
OPMODE
A
B
BCIN
PCIN
C
D
CARRYIN
RSTA
RSTB
RSTM
RSTP
RSTC
RSTD
RSTCARRYIN
RSTOPMODE
CEA
CEB
CEM
CEP
CEC
CED
CECARRYIN
CEOPMODE
BCOUT
PCOUT
P
M
CARRYOUT
CARRYOUTF

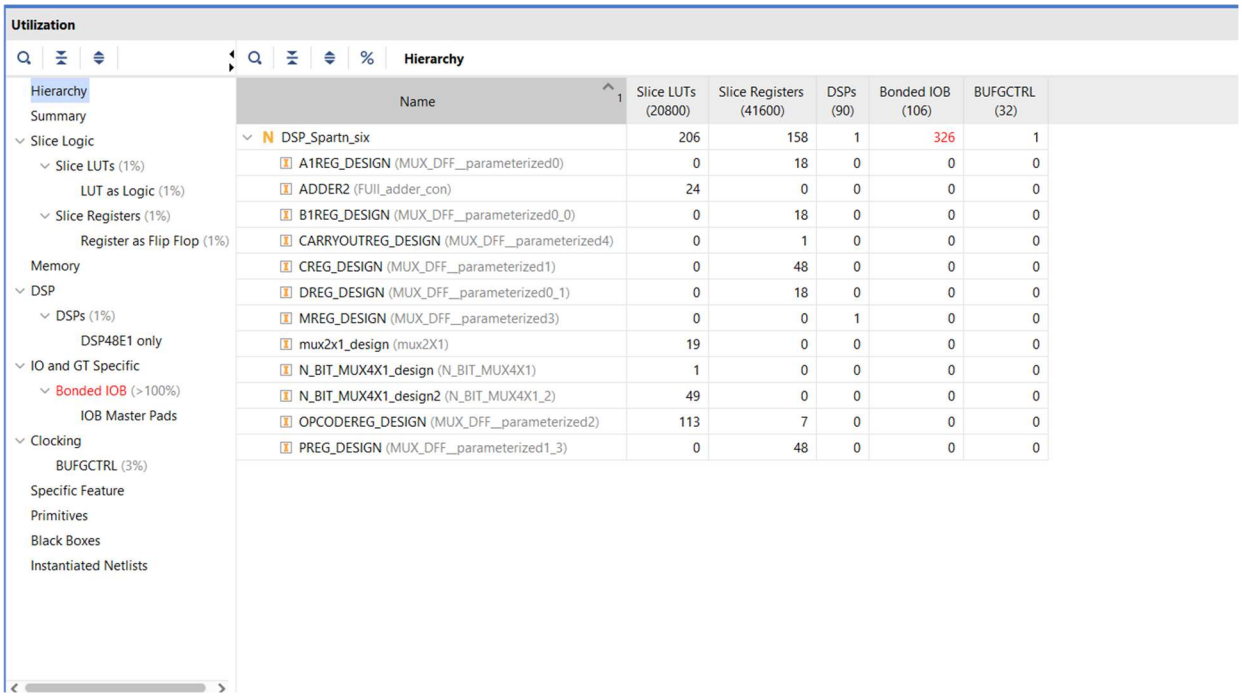
Schematic after elaboration:



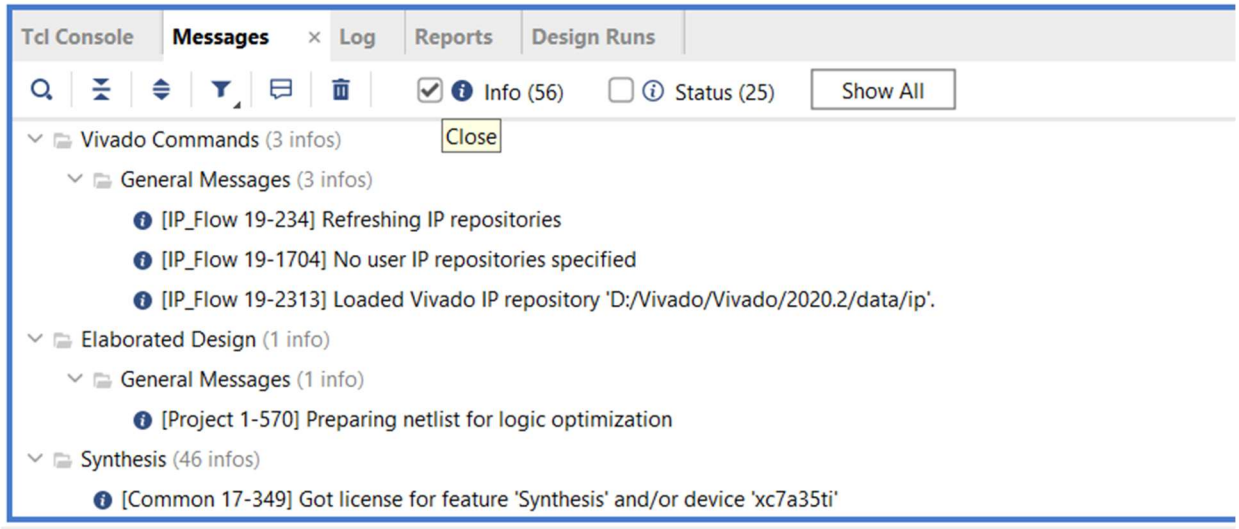
Schematic after synthesis:



Report utilization after synthesis:



Report messages after synthesis:



instance codes:

```
module N_BIT_MUX4X1(D0,D1,D2,D3,S0,S1,Y);
parameter N=48;
input [N-1:0]D0,D1,D2,D3;
input S0,S1;
output reg [N-1:0] Y;
always @(*) begin
case ({S0,S1})
2'b00 : Y=D0;
2'b01 : Y=D1;
2'b10 : Y=D2;
2'b11 : Y=D3;
endcase
end
endmodule
```

```
module mux2X1(D0,D1,SEL,Y);
parameter N=18;
input [N-1:0]D0,D1;
input SEL;
output reg [N-1:0] Y;
always @(*)
begin
if (SEL)
Y = D1;
else
Y = D0;
end
endmodule
```

```
module MUX_CARRYIN(CASCADE,PIN_IN,PIN_OUT);
parameter DFF_SIZE=1;
parameter CARRY_INPUT = "OPCODE5";
input [DFF_SIZE-1:0]CASCADE;
input [DFF_SIZE-1:0]PIN_IN;
output reg [DFF_SIZE-1 :0]PIN_OUT;
always @(*) begin
if(CARRY_INPUT == "OPCODE5")
PIN_OUT=PIN_IN;
else if(CARRY_INPUT == "CARRYIN")
PIN_OUT=CASCADE;
else
PIN_OUT=0;
end
endmodule
```

```
module MUX_DFF(clk,rst,enable,PIN_IN,PIN_OUT);
parameter DFF_SELECT=1;
parameter RSTTYPE="SYNC";
parameter DFF_SIZE=18;
input clk,rst,enable;
input [DFF_SIZE-1:0]PIN_IN;
output reg [DFF_SIZE-1 :0]PIN_OUT;
generate
case (DFF_SELECT)
0: begin
     always @(*) begin
        PIN_OUT=PIN_IN;
     end
end
1: begin
if(RSTTYPE=="SYNC") begin
   always @(posedge clk) begin
        if(rst)
        PIN_OUT<=0;
        else if (enable)
        PIN_OUT<=PIN_IN;
        end
        end
else if (RSTTYPE=="ASYNC") begin
always @(posedge clk or posedge rst ) begin
        if(rst)
        PIN_OUT<=0;
        else if (enable)
        PIN_OUT<=PIN_IN;
        end
end

end
endcase
endgenerate
endmodule
```

```
23  module MUX_DFF_B(clk,rst,enable,CASCADE,PIN_IN,PIN_OUT);
24  parameter DFF_SELECT=1;
25  parameter RSTTYPE="SYNC";
26  parameter DFF_SIZE=18;
27  parameter B_INPUT = "CASCADE";
28  input clk,rst,enable;
29  input [DFF_SIZE-1:0]CASCADE;
30  input [DFF_SIZE-1:0]PIN_IN;
31  output reg [DFF_SIZE-1 :0]PIN_OUT;
32  generate
33  case (DFF_SELECT)
34  0: begin
35       always @(*) begin
36          if ( B_INPUT == "DIRECT") PIN_OUT<=PIN_IN;
37          else if (B_INPUT == "CASCADE") PIN_OUT<=CASCADE;
38          end
39       end
40  1: begin
41  if(RSTTYPE=="SYNC") begin
42     always @(posedge clk) begin
43          if(rst) PIN_OUT<=0;
44          else if (enable) begin
45         if ( B_INPUT == "DIRECT") PIN_OUT<=PIN_IN;
46          else if (B_INPUT == "CASCADE")  PIN_OUT<=CASCADE;
47          end end end
48  else if (RSTTYPE=="ASYNC") begin
49  always @(posedge clk or posedge rst ) begin
50          if(rst)
51          PIN_OUT<=0;
52           else if (enable) begin
53        if ( B_INPUT == "DIRECT")
54          PIN_OUT<=PIN_IN;
55          else if (B_INPUT == "CASCADE")
56          PIN_OUT<=CASCADE;
57          end end endend
58  endcase endgenerate
59  endmodule
```

```
module FUll_adder_con( A,CIN,B,SEL,carry,sum);
parameter N=48;
input [N-1:0] A,B;
input CIN,SEL;
output reg carry;
output reg [N-1:0]sum;
always @(*) begin
if (SEL)
{carry,sum}=A+B+CIN;
else
{carry,sum}=A-(B+CIN);
end
endmodule
```

```
module N_BIT_Multiplier (B,D,Out);
parameter N=18;
localparam N_local=2*N;
input  [N-1:0] B;
input  [N-1:0] D;
output  [N_local-1:0] Out;
reg [N_local-1:0]out_reg;
always @(*) begin
out_reg= B * D;
end
assign Out=out_reg;
endmodule
```

```
module N_BIT_ADDER (D,B,SEL,C);
parameter N=18;
input [N-1:0] B;
input [N-1:0] D;
input SEL;
output reg  [N-1:0] C;
always @(*) begin
if (SEL)
C = D-B;
else
C = D+B;
end
endmodule
```