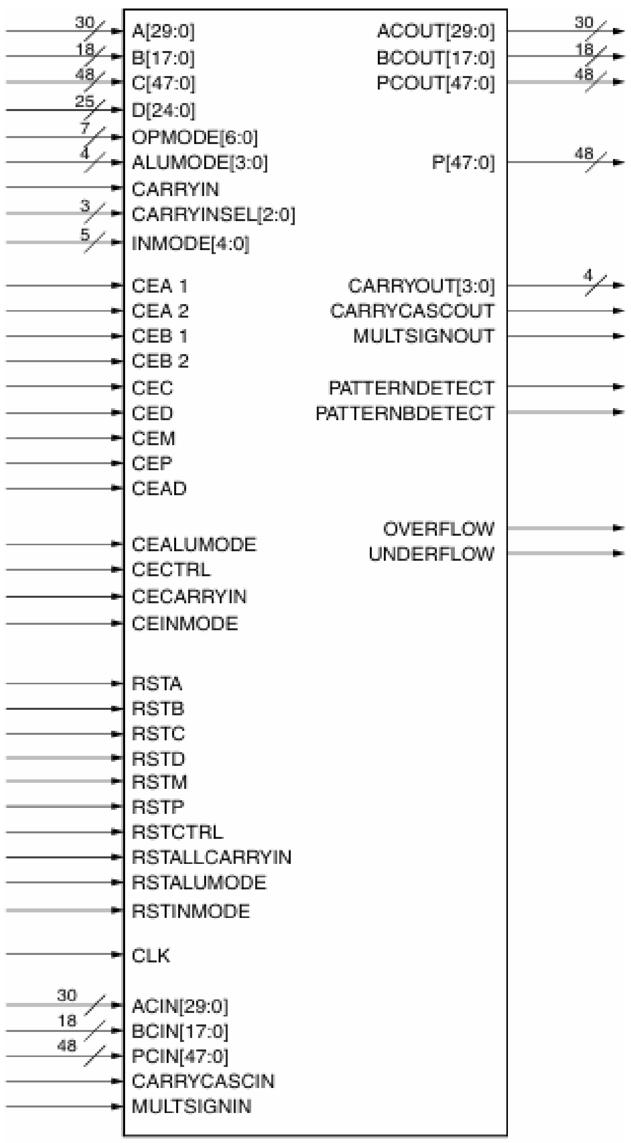
DSP48E1

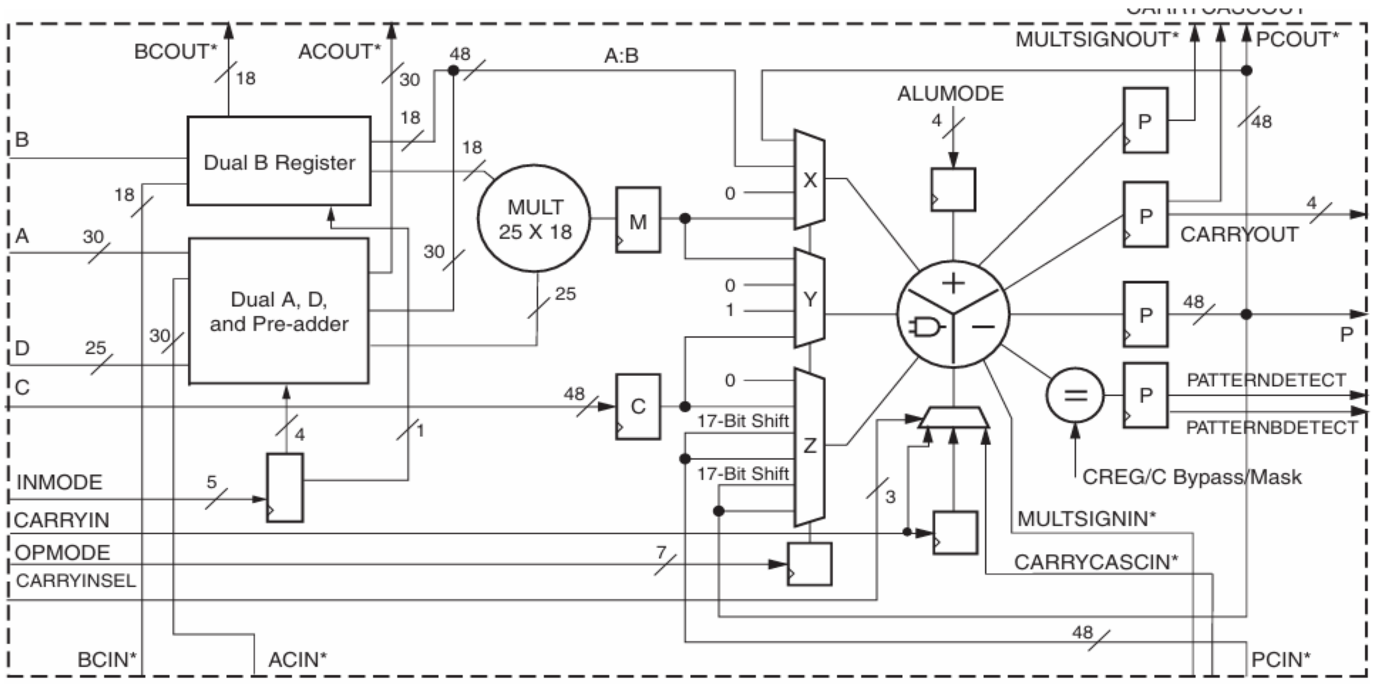


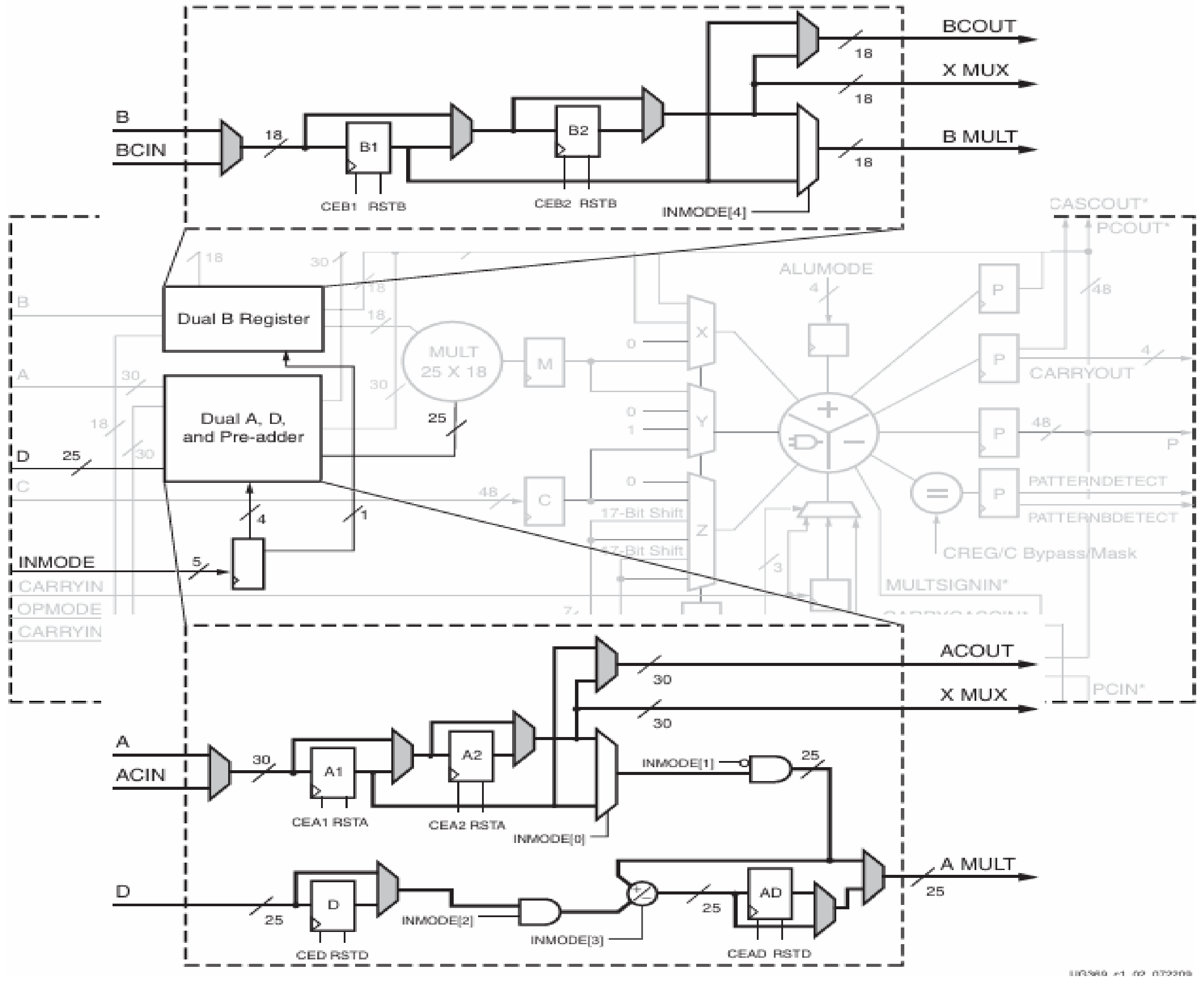
Prepared By: Mohamed Shaban Moussa

Email: mohamedmouse066@gmail.com

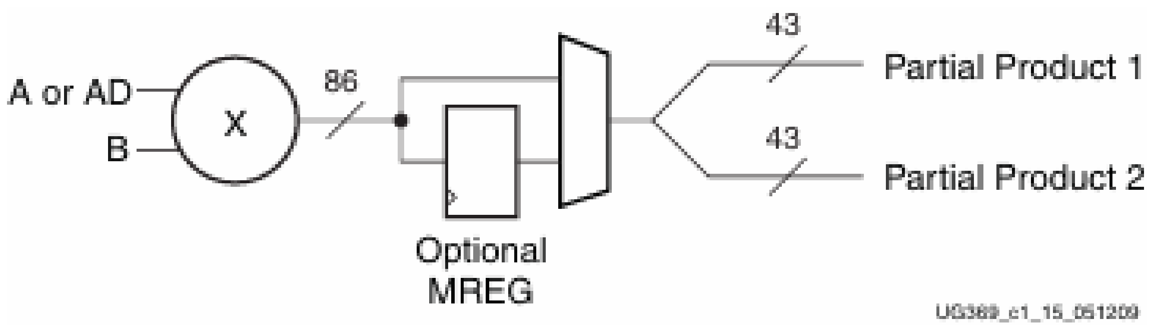
GitHub Repository: Press Here

Design Construction

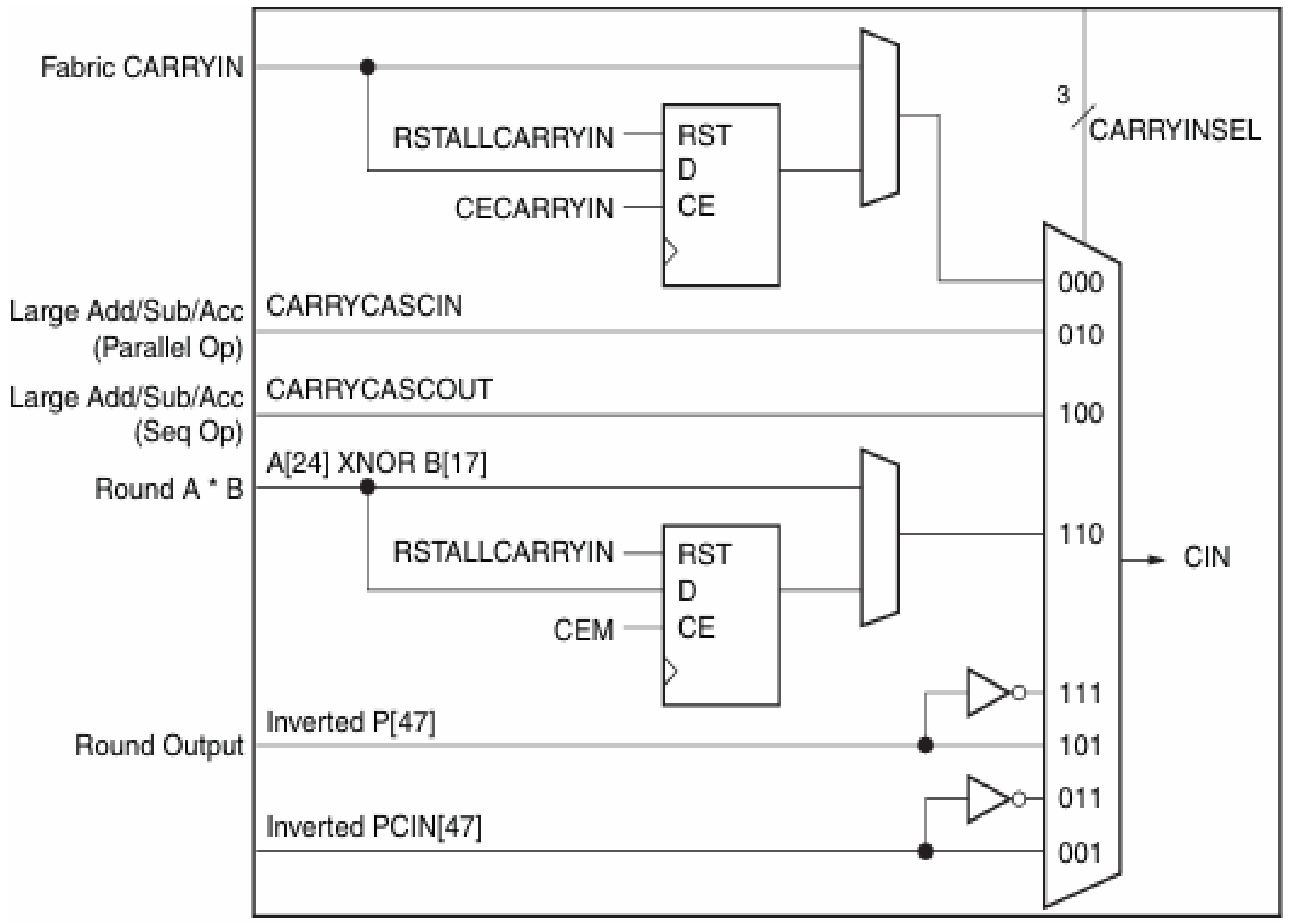




25*18 Patial MULT



CIN MUX



X, Y, Z Muxes

Table 2-7: **OPMODE Control Bits Select X Multiplexer Outputs**

Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	X Multiplexer Output	Notes
xxx	xx	00	0	Default
xxx	01	01	M	Must select with OPMODE[3:2] = 01
xxx	xx	10	P	Must select with PREG = 1
xxx	xx	11	A:B	48 bits wide

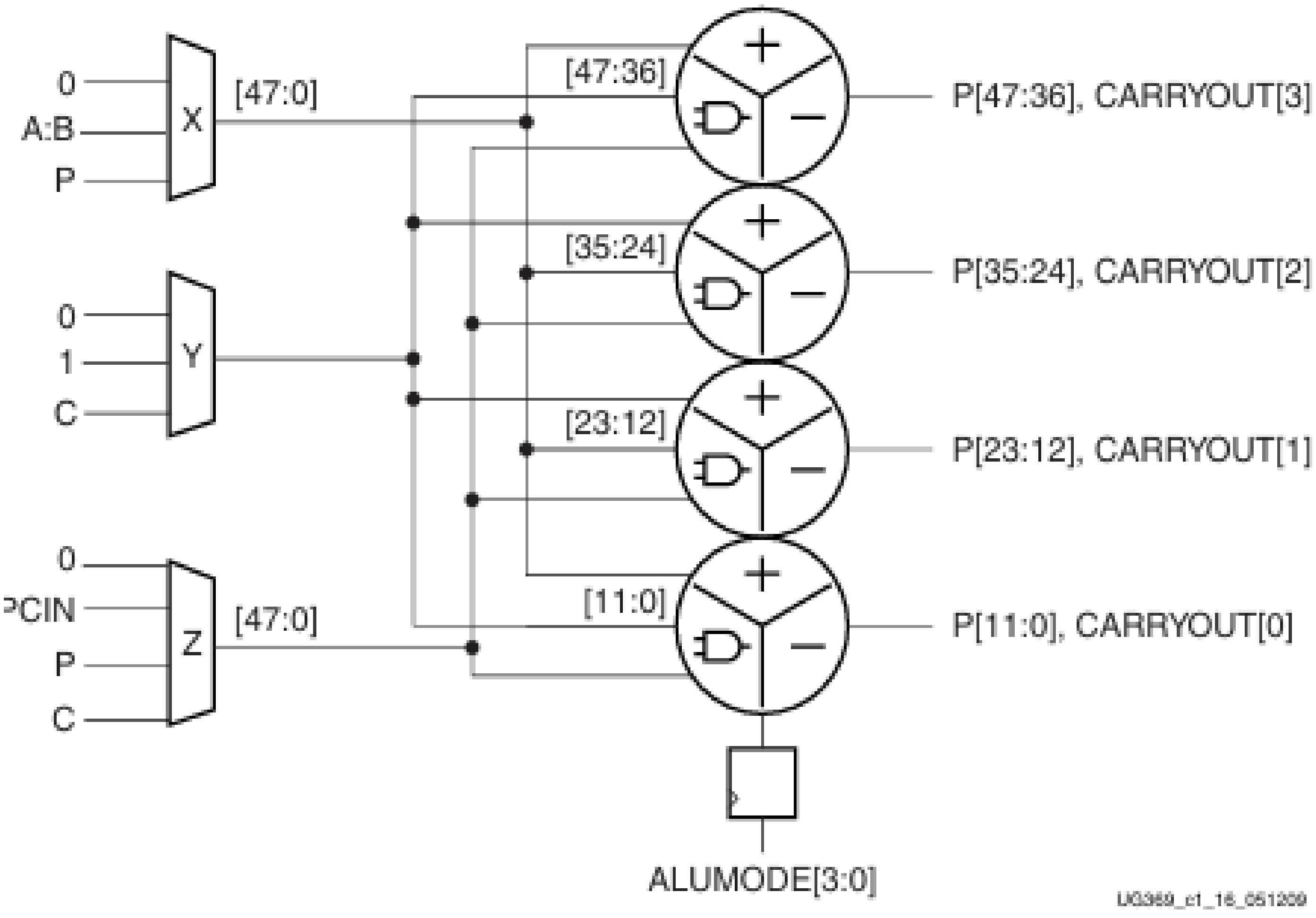
Table 2-8: **OPMODE Control Bits Select Y Multiplexer Outputs**

Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	Y Multiplexer Output	Notes
xxx	00	xx	0	Default
xxx	01	01	M	Must select with OPMODE[1:0] = 01
xxx	10	xx	48'FFFFFFFFFFFF	Used mainly for logic unit bitwise operations on the X and Z multiplexers
xxx	11	xx	C	

Table 2-9: **OPMODE Control Bits Select Z Multiplexer Outputs**

Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	Z Multiplexer Output	Notes
000	xx	xx	0	Default
001	xx	xx	PCIN	
010	xx	xx	P	Must select with PREG = 1
011	xx	xx	C	
100	10	00	P	Use for MACC extend only. Must select with PREG = 1
101	xx	xx	17-bit Shift (PCIN)	
110	xx	xx	17-bit Shift (P)	Must select with PREG = 1
111	xx	xx	xx	Illegal selection

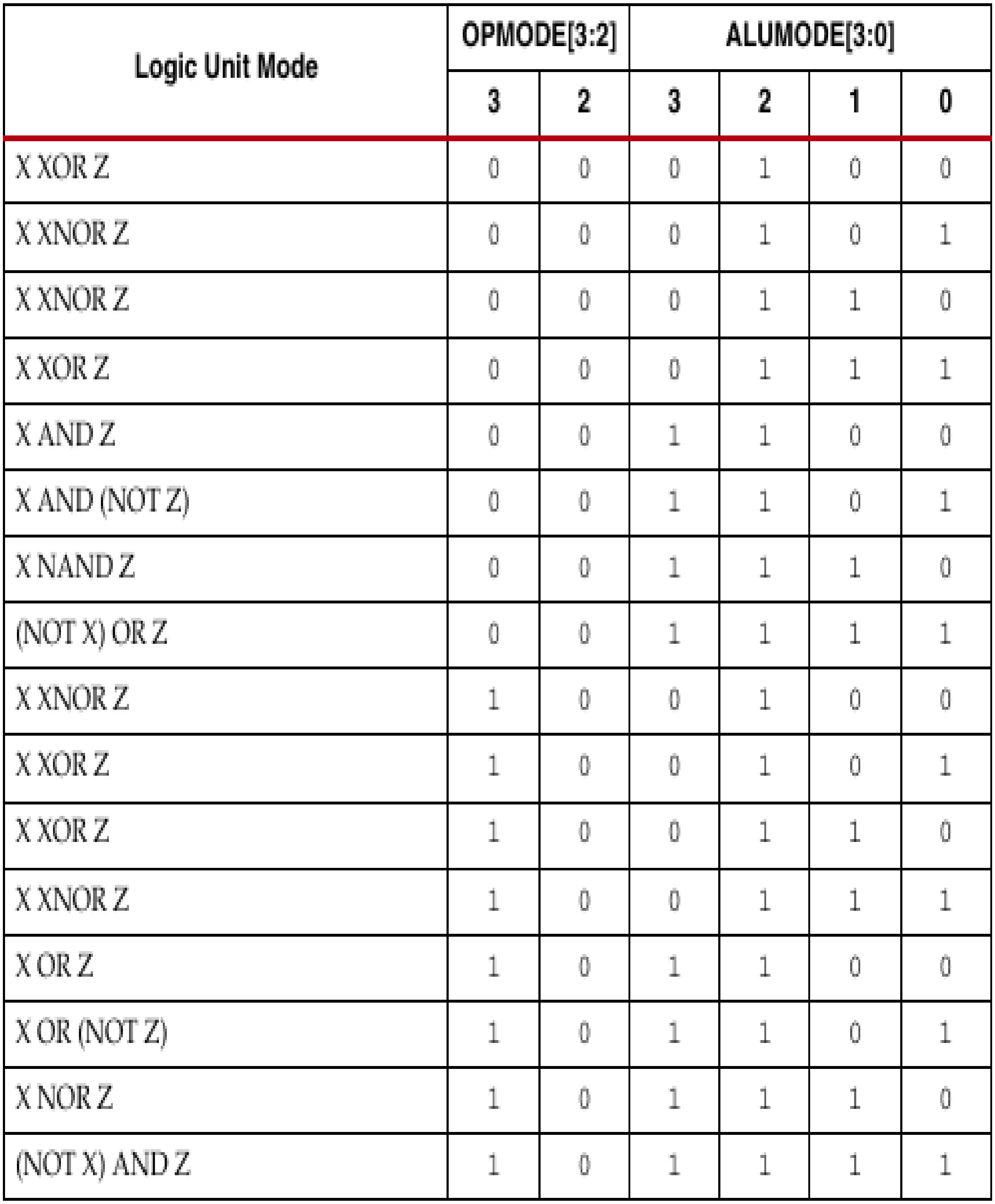
ALU With Maximum SIMD



Arithmetic Part of ALU

DSP Operation	OPMODE[6:0]	ALUMODE[3:0]			
		3	2	1	0
Z + X + Y + CIN	Any legal OPMODE	0	0	0	0
Z – (X + Y + CIN)	Any legal OPMODE	0	0	1	1
–Z + (X + Y + CIN) – 1 = not (Z) + X + Y + CIN	Any legal OPMODE	0	0	0	1
not (Z + X + Y + CIN) = –Z – X – Y – CIN - 1	Any legal OPMODE	0	0	1	0

Logical Part Of ALU

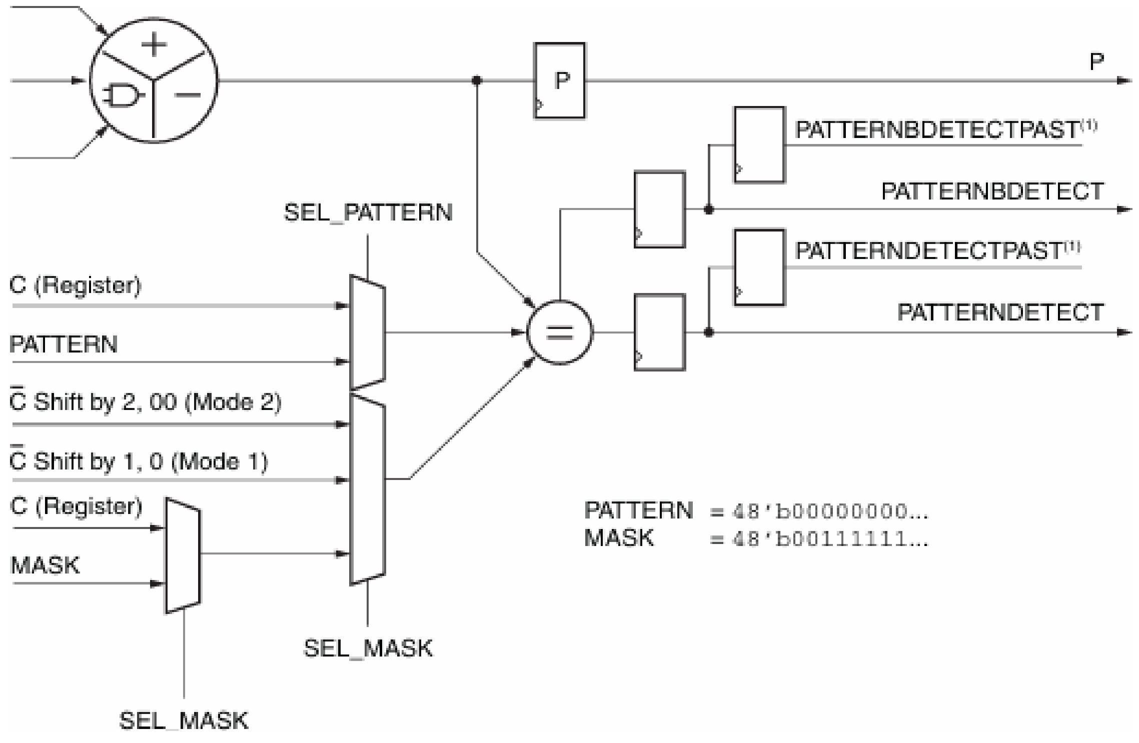


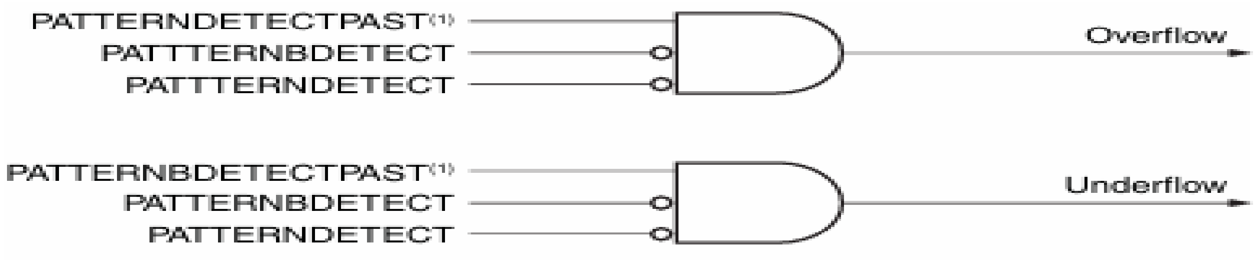
Logic Unit Mode	OPMODE[3:2]		ALUMODE[3:0]			
	3	2	3	2	1	0
X XOR Z	0	0	0	1	0	0
X XNOR Z	0	0	0	1	0	1
X XNOR Z	0	0	0	1	1	0
X XOR Z	0	0	0	1	1	1
X AND Z	0	0	1	1	0	0
X AND (NOT Z)	0	0	1	1	0	1
X NAND Z	0	0	1	1	1	0
(NOT X) OR Z	0	0	1	1	1	1
X XNOR Z	1	0	0	1	0	0
X XOR Z	1	0	0	1	0	1
X XOR Z	1	0	0	1	1	0
X XNOR Z	1	0	0	1	1	1
X OR Z	1	0	1	1	0	0
X OR (NOT Z)	1	0	1	1	0	1
X NOR Z	1	0	1	1	1	0
(NOT X) AND Z	1	0	1	1	1	1

Carry out Distribution

SIMD Mode	Adder Bit Width	Corresponding CARRYOUT
FOUR12	P[11:0]	CARRYOUT[0]
	P[23:12]	CARRYOUT[1]
	P[35:24]	CARRYOUT[2]
	P[47:36]	CARRYOUT[3]
TWO24	P[23:0]	CARRYOUT[1]
	P[47:24]	CARRYOUT[3]
ONE48	P[47:0]	CARRYOUT[3]

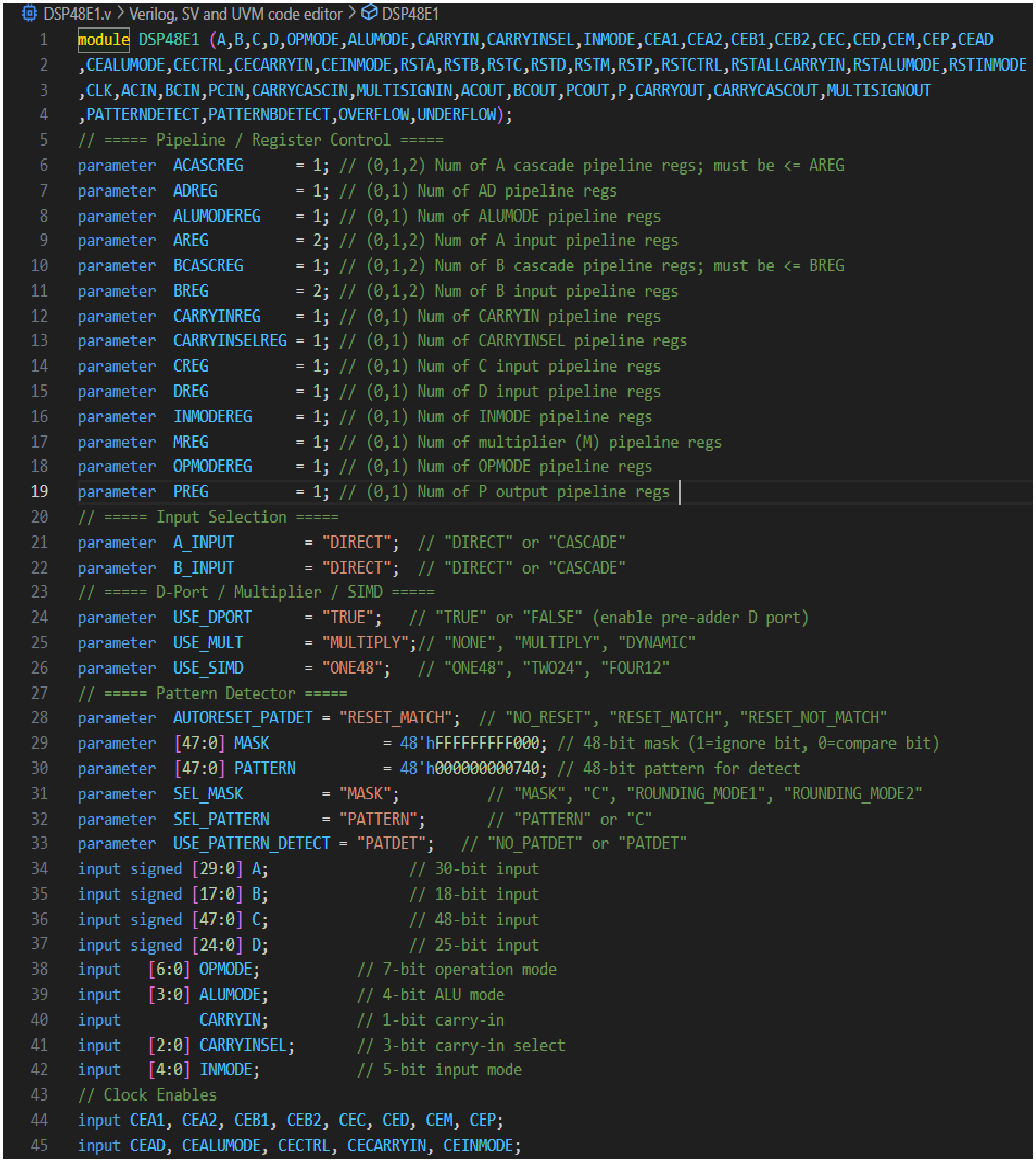
Masked Pattern Detection and Overflow



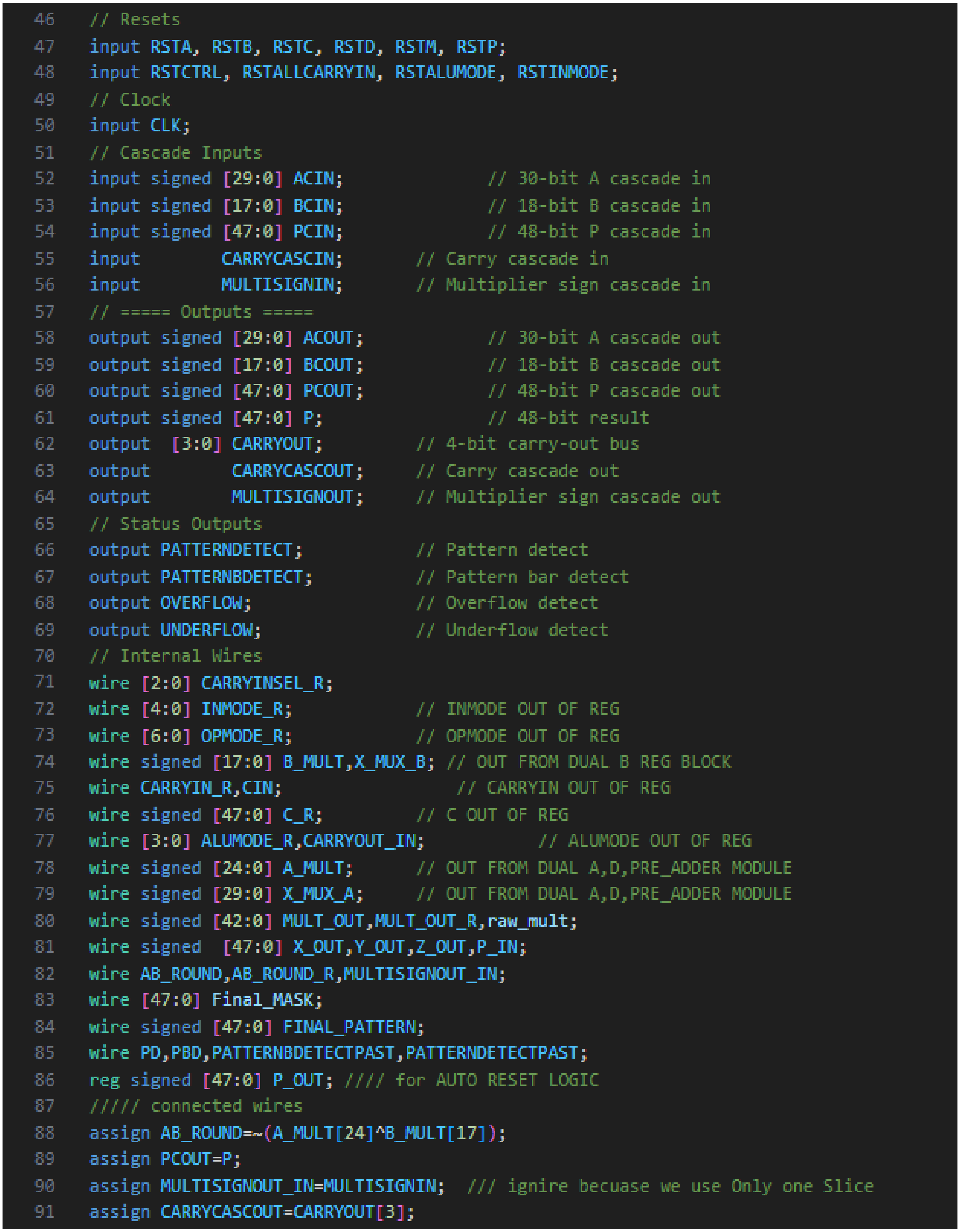


Design Code with Explaining

1- parameters and ports declaration

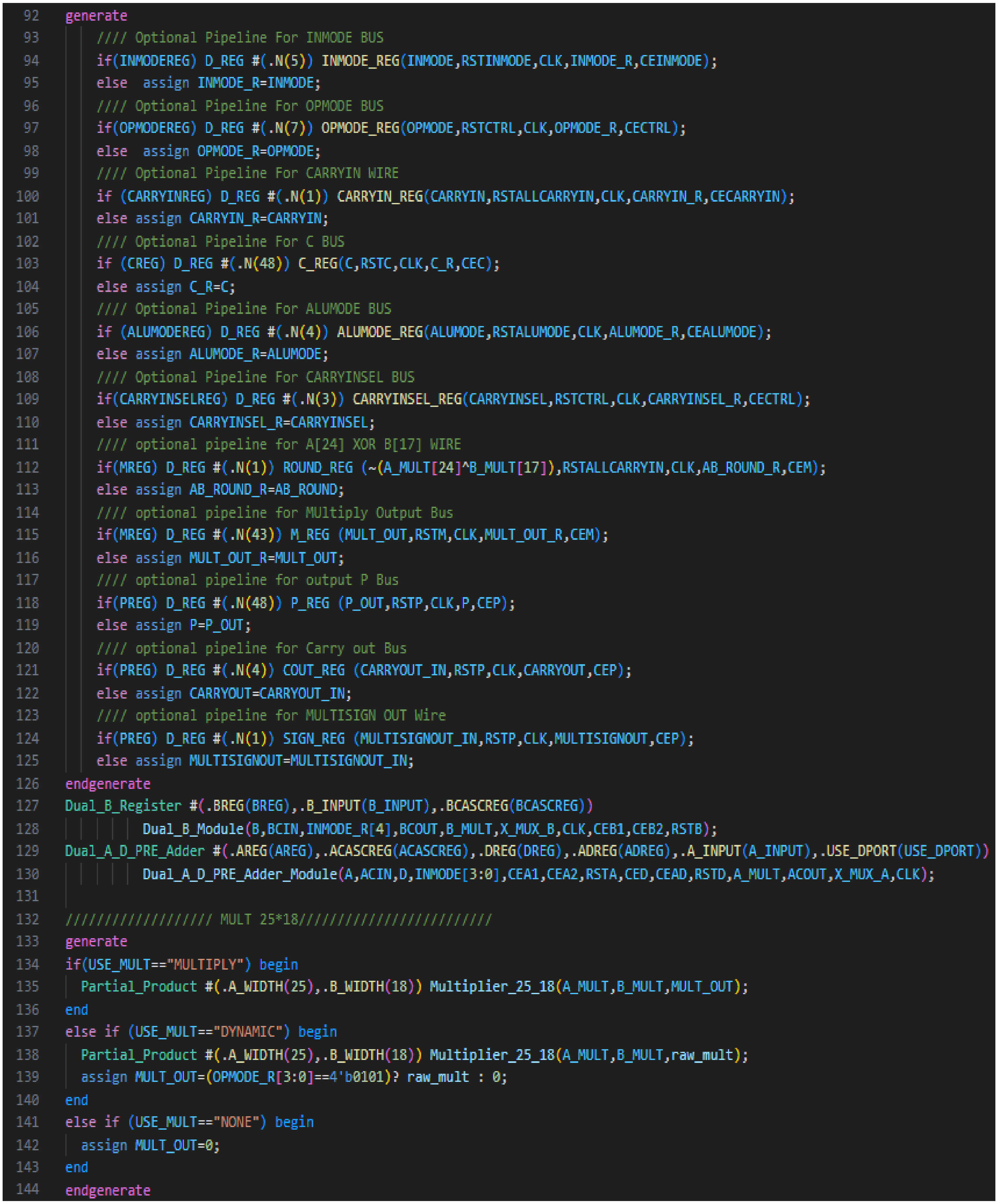


```verilog
module DSP48E1 (A,B,C,D,OPMODE,ALUMODE,CARRYIN,CARRYINSEL,INMODE,CEA1,CEA2,CEB1,CEB2,CEC,CED,CEM,CEP,CEAD
,CEALUMODE,CECTRL,CECARRYIN,CEINMODE,RSTA,RSTB,RSTC,RSTD,RSTM,RSTP,RSTCTRL,RSTALLCARRYIN,RSTALUMODE,RSTINMODE
,CLK,ACIN,BCIN,PCIN,CARRYCASCIN,MULTISIGNIN,ACOUT,BCOUT,PCOUT,P,CARRYOUT,CARRYCASCOUT,MULTISIGNOUT
,PATTERNDETECT,PATTERNBDETECT,OVERFLOW,UNDERFLOW);
// ===== Pipeline / Register Control =====
parameter  ACASCREG      = 1; // (0,1,2) Num of A cascade pipeline regs; must be <= AREG
parameter  ADREG         = 1; // (0,1) Num of AD pipeline regs
parameter  ALUMODEREG    = 1; // (0,1) Num of ALUMODE pipeline regs
parameter  AREG          = 2; // (0,1,2) Num of A input pipeline regs
parameter  BCASCREG      = 1; // (0,1,2) Num of B cascade pipeline regs; must be <= BREG
parameter  BREG          = 2; // (0,1,2) Num of B input pipeline regs
parameter  CARRYINREG    = 1; // (0,1) Num of CARRYIN pipeline regs
parameter  CARRYINSELREG = 1; // (0,1) Num of CARRYINSEL pipeline regs
parameter  CREG          = 1; // (0,1) Num of C input pipeline regs
parameter  DREG          = 1; // (0,1) Num of D input pipeline regs
parameter  INMODEREG     = 1; // (0,1) Num of INMODE pipeline regs
parameter  MREG          = 1; // (0,1) Num of multiplier (M) pipeline regs
parameter  OPMODEREG     = 1; // (0,1) Num of OPMODE pipeline regs
parameter  PREG          = 1; // (0,1) Num of P output pipeline regs
// ===== Input Selection =====
parameter  A_INPUT        = "DIRECT";   // "DIRECT" or "CASCADE"
parameter  B_INPUT        = "DIRECT";   // "DIRECT" or "CASCADE"
// ===== D-Port / Multiplier / SIMD =====
parameter  USE_DPORT      = "TRUE";   // "TRUE" or "FALSE" (enable pre-adder D port)
parameter  USE_MULT       = "MULTIPLY";// "NONE", "MULTIPLY", "DYNAMIC"
parameter  USE_SIMD       = "ONE48";    // "ONE48", "TWO24", "FOUR12"
// ===== Pattern Detector =====
parameter  AUTORESET_PATDET = "RESET_MATCH";  // "NO_RESET", "RESET_MATCH", "RESET_NOT_MATCH"
parameter  [47:0] MASK            = 48'hFFFFFFFFF000; // 48-bit mask (1=ignore bit, 0=compare bit)
parameter  [47:0] PATTERN         = 48'h000000000740; // 48-bit pattern for detect
parameter  SEL_MASK         = "MASK";           // "MASK", "C", "ROUNDING_MODE1", "ROUNDING_MODE2"
parameter  SEL_PATTERN      = "PATTERN";        // "PATTERN" or "C"
parameter  USE_PATTERN_DETECT = "PATDET";   // "NO_PATDET" or "PATDET"
input signed [29:0] A;                // 30-bit input
input signed [17:0] B;                // 18-bit input
input signed [47:0] C;                // 48-bit input
input signed [24:0] D;                // 25-bit input
input   [6:0] OPMODE;           // 7-bit operation mode
input   [3:0] ALUMODE;          // 4-bit ALU mode
input         CARRYIN;          // 1-bit carry-in
input   [2:0] CARRYINSEL;       // 3-bit carry-in select
input   [4:0] INMODE;           // 5-bit input mode
// Clock Enables
input CEA1, CEA2, CEB1, CEB2, CEC, CED, CEM, CEP;
input CEAD, CEALUMODE, CECTRL, CECARRYIN, CEINMODE;
```



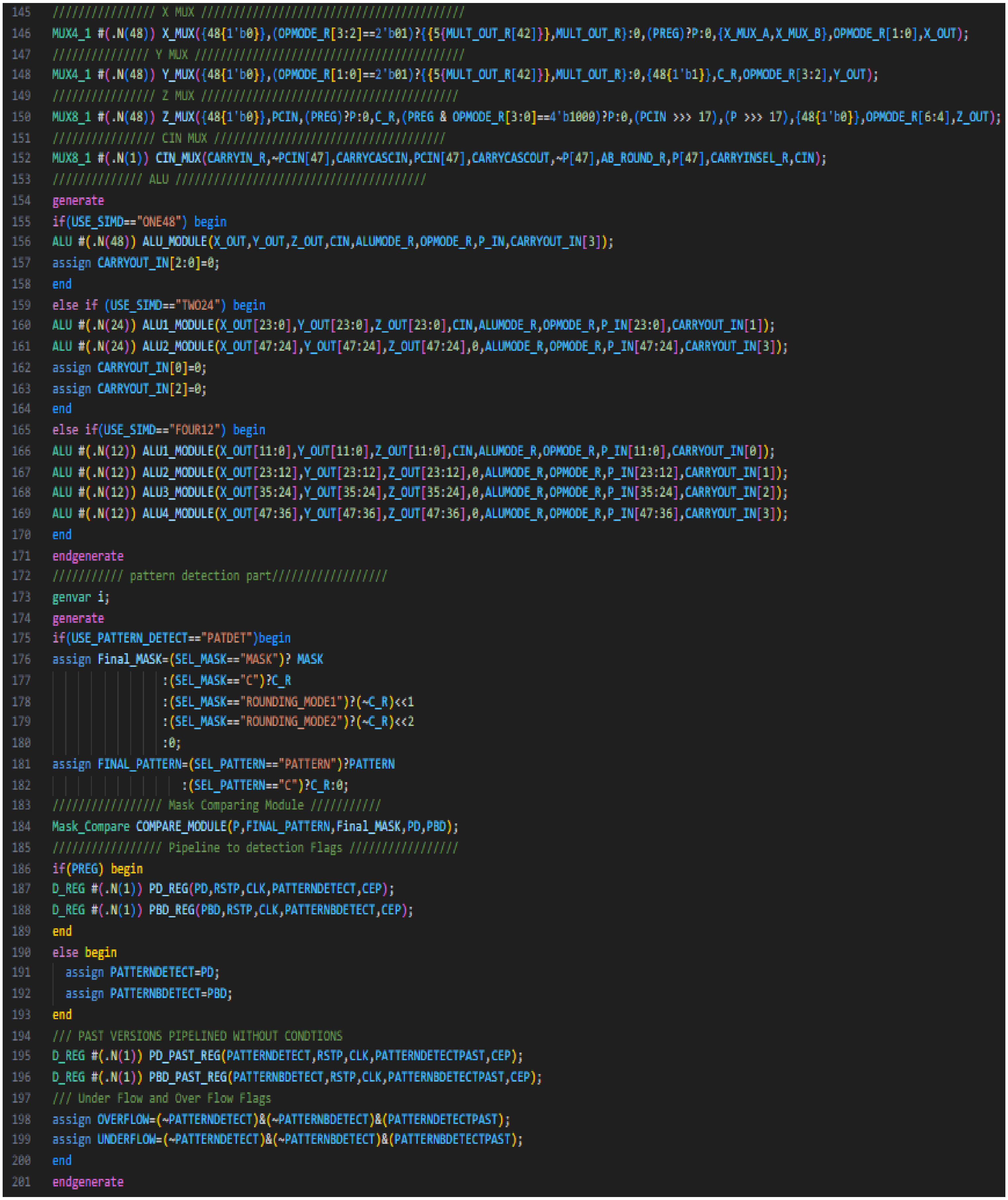
```verilog
// Resets
input RSTA, RSTB, RSTC, RSTD, RSTM, RSTP;
input RSTCTRL, RSTALLCARRYIN, RSTALUMODE, RSTINMODE;
// Clock
input CLK;
// Cascade Inputs
input signed [29:0] ACIN;              // 30-bit A cascade in
input signed [17:0] BCIN;              // 18-bit B cascade in
input signed [47:0] PCIN;              // 48-bit P cascade in
input        CARRYCASCIN;       // Carry cascade in
input        MULTISIGNIN;       // Multiplier sign cascade in
// ===== Outputs =====
output signed [29:0] ACOUT;            // 30-bit A cascade out
output signed [17:0] BCOUT;            // 18-bit B cascade out
output signed [47:0] PCOUT;            // 48-bit P cascade out
output signed [47:0] P;                // 48-bit result
output  [3:0] CARRYOUT;         // 4-bit carry-out bus
output        CARRYCASCOUT;     // Carry cascade out
output        MULTISIGNOUT;     // Multiplier sign cascade out
// Status Outputs
output PATTERNDETECT;           // Pattern detect
output PATTERNBDETECT;          // Pattern bar detect
output OVERFLOW;                // Overflow detect
output UNDERFLOW;               // Underflow detect
// Internal Wires
wire [2:0] CARRYINSEL_R;
wire [4:0] INMODE_R;            // INMODE OUT OF REG
wire [6:0] OPMODE_R;            // OPMODE OUT OF REG
wire signed [17:0] B_MULT,X_MUX_B; // OUT FROM DUAL B REG BLOCK
wire CARRYIN_R,CIN;                 // CARRYIN OUT OF REG
wire signed [47:0] C_R;         // C OUT OF REG
wire [3:0] ALUMODE_R,CARRYOUT_IN;           // ALUMODE OUT OF REG
wire signed [24:0] A_MULT;      // OUT FROM DUAL A,D,PRE_ADDER MODULE
wire signed [29:0] X_MUX_A;     // OUT FROM DUAL A,D,PRE_ADDER MODULE
wire signed [42:0] MULT_OUT,MULT_OUT_R,raw_mult;
wire signed  [47:0] X_OUT,Y_OUT,Z_OUT,P_IN;
wire AB_ROUND,AB_ROUND_R,MULTISIGNOUT_IN;
wire [47:0] Final_MASK;
wire signed [47:0] FINAL_PATTERN;
wire PD,PBD,PATTERNBDETECTPAST,PATTERNDETECTPAST;
reg signed [47:0] P_OUT; //// for AUTO RESET LOGIC
///// connected wires
assign AB_ROUND=~(A_MULT[24]^B_MULT[17]);
assign PCOUT=P;
assign MULTISIGNOUT_IN=MULTISIGNIN;  /// ignire becuase we use Only one Slice
assign CARRYCASCOUT=CARRYOUT[3];
```

2- Optional Pipeline and 2 Duals and Multiplier



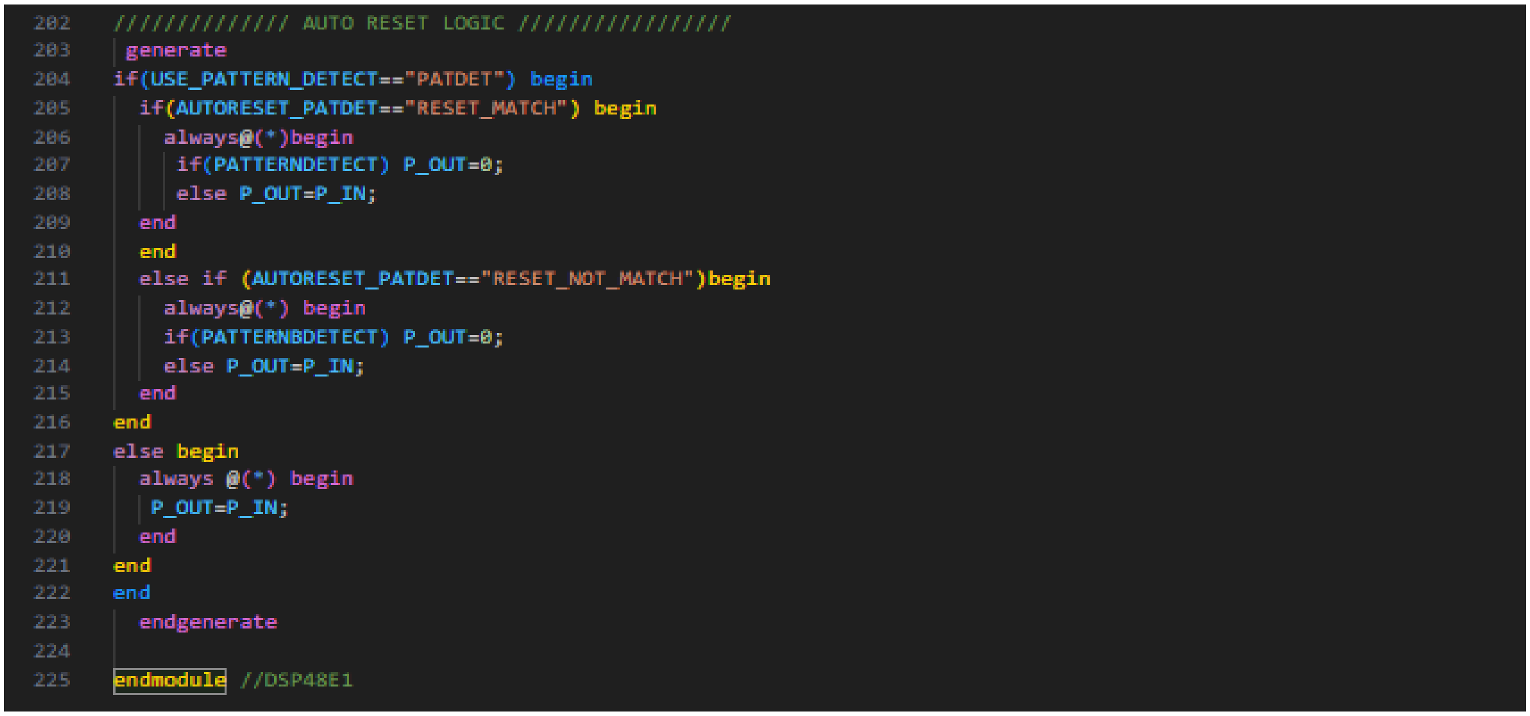
```verilog
generate
    //// Optional Pipeline For INMODE BUS
    if(INMODEREG) D_REG #(.N(5)) INMODE_REG(INMODE,RSTINMODE,CLK,INMODE_R,CEINMODE);
    else  assign INMODE_R=INMODE;
    //// Optional Pipeline For OPMODE BUS
    if(OPMODEREG) D_REG #(.N(7)) OPMODE_REG(OPMODE,RSTCTRL,CLK,OPMODE_R,CECTRL);
    else  assign OPMODE_R=OPMODE;
    //// Optional Pipeline For CARRYIN WIRE
    if (CARRYINREG) D_REG #(.N(1)) CARRYIN_REG(CARRYIN,RSTALLCARRYIN,CLK,CARRYIN_R,CECARRYIN);
    else assign CARRYIN_R=CARRYIN;
    //// Optional Pipeline For C BUS
    if (CREG) D_REG #(.N(48)) C_REG(C,RSTC,CLK,C_R,CEC);
    else assign C_R=C;
    //// Optional Pipeline For ALUMODE BUS
    if (ALUMODEREG) D_REG #(.N(4)) ALUMODE_REG(ALUMODE,RSTALUMODE,CLK,ALUMODE_R,CEALUMODE);
    else assign ALUMODE_R=ALUMODE;
    //// Optional Pipeline For CARRYINSEL BUS
    if(CARRYINSELREG) D_REG #(.N(3)) CARRYINSEL_REG(CARRYINSEL,RSTCTRL,CLK,CARRYINSEL_R,CECTRL);
    else assign CARRYINSEL_R=CARRYINSEL;
    //// optional pipeline for A[24] XOR B[17] WIRE
    if(MREG) D_REG #(.N(1)) ROUND_REG (~(A_MULT[24]^B_MULT[17]),RSTALLCARRYIN,CLK,AB_ROUND_R,CEM);
    else assign AB_ROUND_R=AB_ROUND;
    //// optional pipeline for MUltiply Output Bus
    if(MREG) D_REG #(.N(43)) M_REG (MULT_OUT,RSTM,CLK,MULT_OUT_R,CEM);
    else assign MULT_OUT_R=MULT_OUT;
    //// optional pipeline for output P Bus
    if(PREG) D_REG #(.N(48)) P_REG (P_OUT,RSTP,CLK,P,CEP);
    else assign P=P_OUT;
    //// optional pipeline for Carry out Bus
    if(PREG) D_REG #(.N(4)) COUT_REG (CARRYOUT_IN,RSTP,CLK,CARRYOUT,CEP);
    else assign CARRYOUT=CARRYOUT_IN;
    //// optional pipeline for MULTISIGN OUT Wire
    if(PREG) D_REG #(.N(1)) SIGN_REG (MULTISIGNOUT_IN,RSTP,CLK,MULTISIGNOUT,CEP);
    else assign MULTISIGNOUT=MULTISIGNOUT_IN;
endgenerate
Dual_B_Register #(.BREG(BREG),.B_INPUT(B_INPUT),.BCASCREG(BCASCREG))
                Dual_B_Module(B,BCIN,INMODE_R[4],BCOUT,B_MULT,X_MUX_B,CLK,CEB1,CEB2,RSTB);
Dual_A_D_PRE_Adder #(.AREG(AREG),.ACASCREG(ACASCREG),.DREG(DREG),.ADREG(ADREG),.A_INPUT(A_INPUT),.USE_DPORT(USE_DPORT))
                Dual_A_D_PRE_Adder_Module(A,ACIN,D,INMODE[3:0],CEA1,CEA2,RSTA,CED,CEAD,RSTD,A_MULT,ACOUT,X_MUX_A,CLK);

/////////////////// MULT 25*18////////////////////////
generate
if(USE_MULT=="MULTIPLY") begin
  Partial_Product #(.A_WIDTH(25),.B_WIDTH(18)) Multiplier_25_18(A_MULT,B_MULT,MULT_OUT);
end
else if (USE_MULT=="DYNAMIC") begin
  Partial_Product #(.A_WIDTH(25),.B_WIDTH(18)) Multiplier_25_18(A_MULT,B_MULT,raw_mult);
  assign MULT_OUT=(OPMODE_R[3:0]==4'b0101)? raw_mult : 0;
end
else if (USE_MULT=="NONE") begin
  assign MULT_OUT=0;
end
endgenerate
```

3-Muxes and Pattern Part



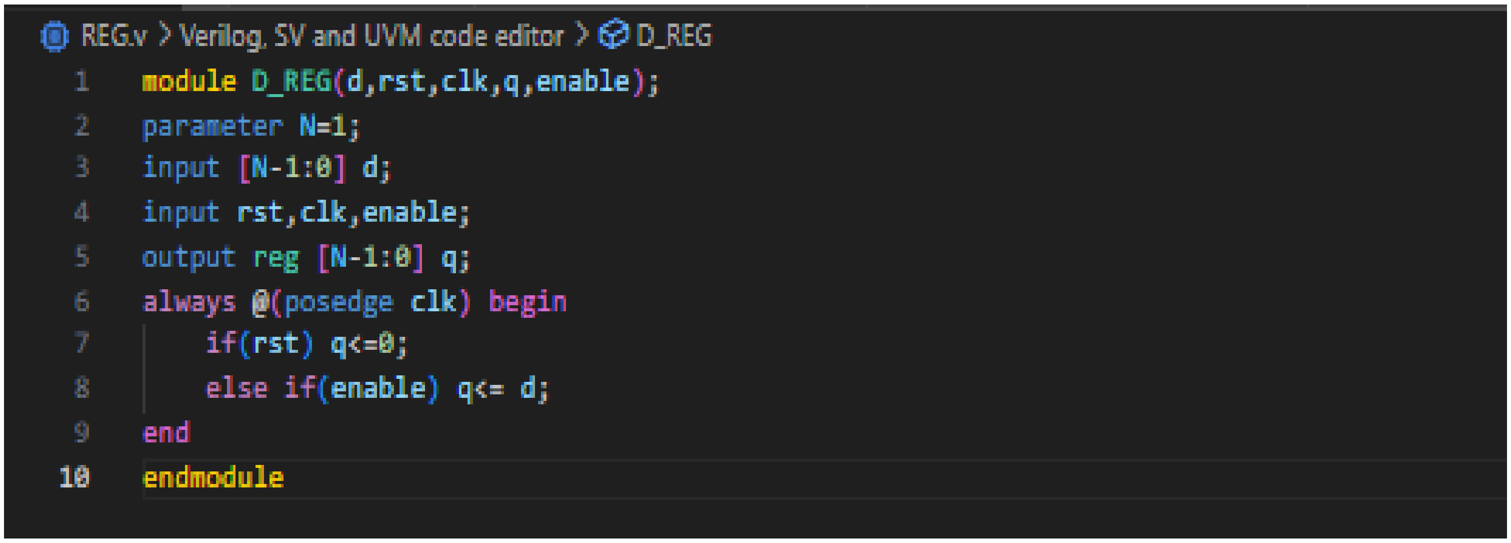
```verilog
//////////////// X MUX /////////////////////////////////////////
MUX4_1 #(.N(48)) X_MUX({48{1'b0}},(OPMODE_R[3:2]==2'b01)?{{5{MULT_OUT_R[42]}},MULT_OUT_R}:0,(PREG)?P:0,{X_MUX_A,X_MUX_B},OPMODE_R[1:0],X_OUT);
/////////////// Y MUX //////////////////////////////////////////
MUX4_1 #(.N(48)) Y_MUX({48{1'b0}},(OPMODE_R[1:0]==2'b01)?{{5{MULT_OUT_R[42]}},MULT_OUT_R}:0,{48{1'b1}},C_R,OPMODE_R[3:2],Y_OUT);
//////////////// Z MUX ////////////////////////////////////////
MUX8_1 #(.N(48)) Z_MUX({48{1'b0}},PCIN,(PREG)?P:0,C_R,(PREG & OPMODE_R[3:0]==4'b1000)?P:0,(PCIN >>> 17),(P >>> 17),{48{1'b0}},OPMODE_R[6:4],Z_OUT);
//////////////// CIN MUX ////////////////////////////////////
MUX8_1 #(.N(1)) CIN_MUX(CARRYIN_R,~PCIN[47],CARRYCASCIN,PCIN[47],CARRYCASCOUT,~P[47],AB_ROUND_R,P[47],CARRYINSEL_R,CIN);
////////////// ALU //////////////////////////////////////
generate
if(USE_SIMD=="ONE48") begin
ALU #(.N(48)) ALU_MODULE(X_OUT,Y_OUT,Z_OUT,CIN,ALUMODE_R,OPMODE_R,P_IN,CARRYOUT_IN[3]);
assign CARRYOUT_IN[2:0]=0;
end
else if (USE_SIMD=="TWO24") begin
ALU #(.N(24)) ALU1_MODULE(X_OUT[23:0],Y_OUT[23:0],Z_OUT[23:0],CIN,ALUMODE_R,OPMODE_R,P_IN[23:0],CARRYOUT_IN[1]);
ALU #(.N(24)) ALU2_MODULE(X_OUT[47:24],Y_OUT[47:24],Z_OUT[47:24],0,ALUMODE_R,OPMODE_R,P_IN[47:24],CARRYOUT_IN[3]);
assign CARRYOUT_IN[0]=0;
assign CARRYOUT_IN[2]=0;
end
else if(USE_SIMD=="FOUR12") begin
ALU #(.N(12)) ALU1_MODULE(X_OUT[11:0],Y_OUT[11:0],Z_OUT[11:0],CIN,ALUMODE_R,OPMODE_R,P_IN[11:0],CARRYOUT_IN[0]);
ALU #(.N(12)) ALU2_MODULE(X_OUT[23:12],Y_OUT[23:12],Z_OUT[23:12],0,ALUMODE_R,OPMODE_R,P_IN[23:12],CARRYOUT_IN[1]);
ALU #(.N(12)) ALU3_MODULE(X_OUT[35:24],Y_OUT[35:24],Z_OUT[35:24],0,ALUMODE_R,OPMODE_R,P_IN[35:24],CARRYOUT_IN[2]);
ALU #(.N(12)) ALU4_MODULE(X_OUT[47:36],Y_OUT[47:36],Z_OUT[47:36],0,ALUMODE_R,OPMODE_R,P_IN[47:36],CARRYOUT_IN[3]);
end
endgenerate
////////// pattern detection part////////////////
genvar i;
generate
if(USE_PATTERN_DETECT=="PATDET")begin
assign Final_MASK=(SEL_MASK=="MASK")? MASK
                                   :(SEL_MASK=="C")?C_R
                                   :(SEL_MASK=="ROUNDING_MODE1")?(~C_R)<<1
                                   :(SEL_MASK=="ROUNDING_MODE2")?(~C_R)<<2
                                   :0;
assign FINAL_PATTERN=(SEL_PATTERN=="PATTERN")?PATTERN
                                        :(SEL_PATTERN=="C")?C_R:0;
//////////////// Mask Comparing Module //////////
Mask_Compare COMPARE_MODULE(P,FINAL_PATTERN,Final_MASK,PD,PBD);
//////////////// Pipeline to detection Flags ///////////////
if(PREG) begin
D_REG #(.N(1)) PD_REG(PD,RSTP,CLK,PATTERNDETECT,CEP);
D_REG #(.N(1)) PBD_REG(PBD,RSTP,CLK,PATTERNBDETECT,CEP);
end
else begin
  assign PATTERNDETECT=PD;
  assign PATTERNBDETECT=PBD;
end
/// PAST VERSIONS PIPELINED WITHOUT CONDTIONS
D_REG #(.N(1)) PD_PAST_REG(PATTERNDETECT,RSTP,CLK,PATTERNDETECTPAST,CEP);
D_REG #(.N(1)) PBD_PAST_REG(PATTERNBDETECT,RSTP,CLK,PATTERNBDETECTPAST,CEP);
/// Under Flow and Over Flow Flags
assign OVERFLOW=(~PATTERNDETECT)&(~PATTERNBDETECT)&(PATTERNDETECTPAST);
assign UNDERFLOW=(~PATTERNDETECT)&(~PATTERNBDETECT)&(PATTERNBDETECTPAST);
end
endgenerate
```

4-Auto Detection Reset Part



```
////////////// AUTO RESET LOGIC /////////////////
 generate
if(USE_PATTERN_DETECT=="PATDET") begin
  if(AUTORESET_PATDET=="RESET_MATCH") begin
    always@(*)begin
     if(PATTERNDETECT) P_OUT=0;
     else P_OUT=P_IN;
  end
  end
  else if (AUTORESET_PATDET=="RESET_NOT_MATCH")begin
    always@(*) begin
    if(PATTERNBDETECT) P_OUT=0;
    else P_OUT=P_IN;
  end
end
else begin
  always @(*) begin
   P_OUT=P_IN;
  end
end
end
  endgenerate

endmodule //DSP48E1
```

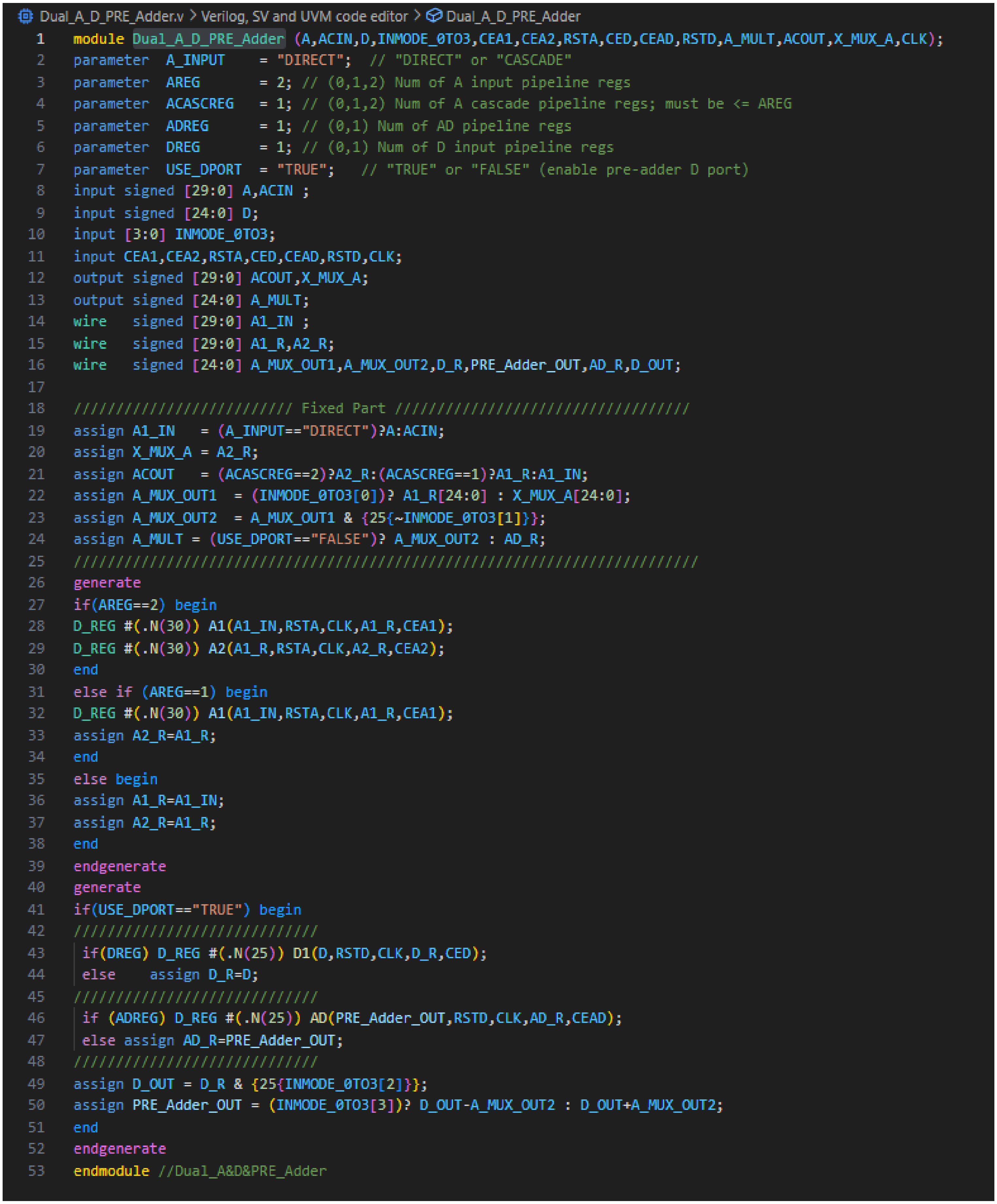
Reg Module



REG.v > Verilog, SV and UVM code editor > D_REG

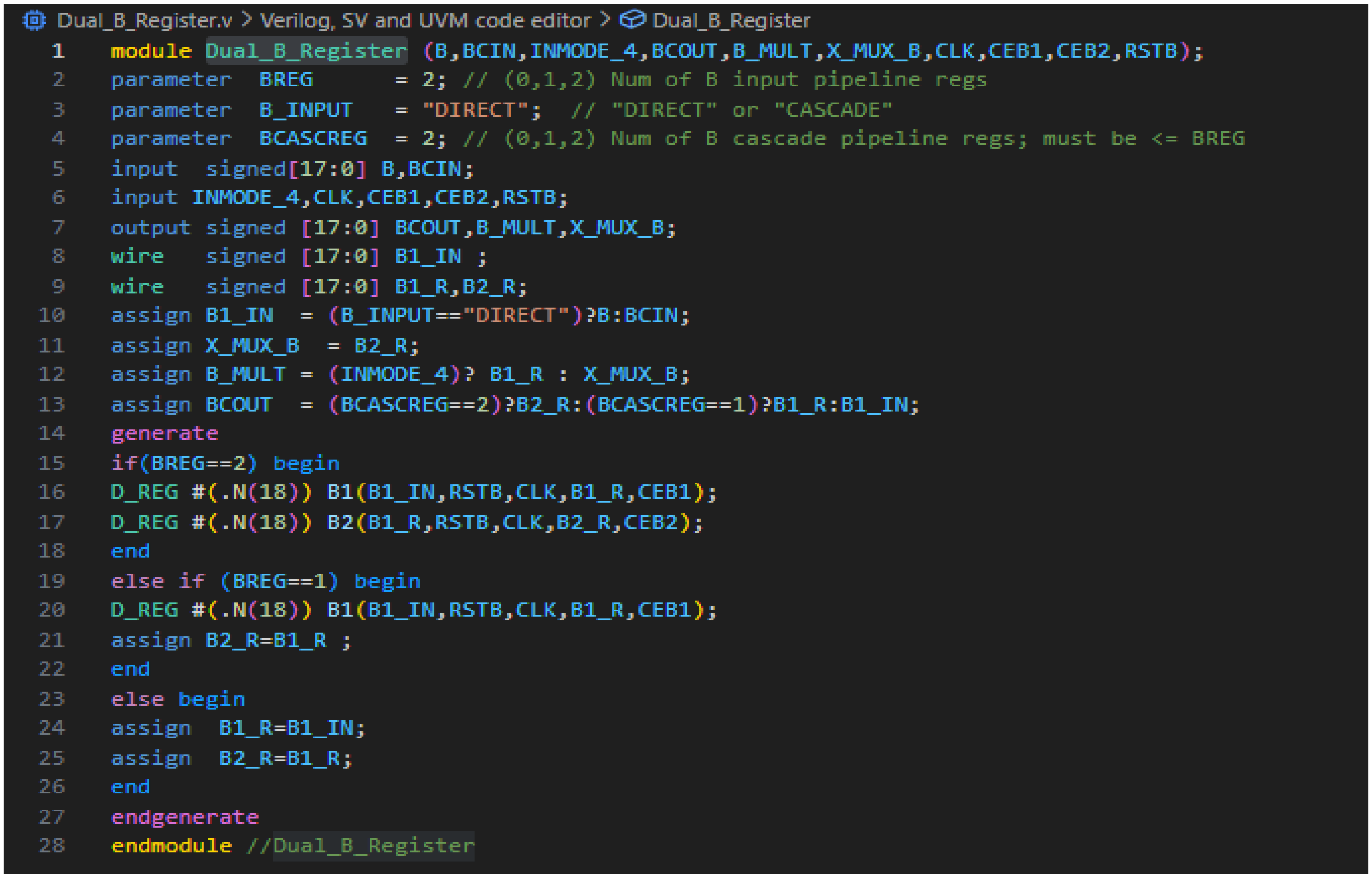
```
module D_REG(d,rst,clk,q,enable);
parameter N=1;
input [N-1:0] d;
input rst,clk,enable;
output reg [N-1:0] q;
always @(posedge clk) begin
     if(rst) q<=0;
     else if(enable) q<= d;
end
endmodule
```

Dual A D and pre–Adder Module


Dual_A_D_PRE_Adder.v > Verilog, SV and UVM code editor > Dual_A_D_PRE_Adder

```verilog
module Dual_A_D_PRE_Adder (A,ACIN,D,INMODE_0TO3,CEA1,CEA2,RSTA,CED,CEAD,RSTD,A_MULT,ACOUT,X_MUX_A,CLK);
parameter  A_INPUT    = "DIRECT";  // "DIRECT" or "CASCADE"
parameter  AREG       = 2; // (0,1,2) Num of A input pipeline regs
parameter  ACASCREG   = 1; // (0,1,2) Num of A cascade pipeline regs; must be <= AREG
parameter  ADREG      = 1; // (0,1) Num of AD pipeline regs
parameter  DREG       = 1; // (0,1) Num of D input pipeline regs
parameter  USE_DPORT  = "TRUE";   // "TRUE" or "FALSE" (enable pre-adder D port)
input signed [29:0] A,ACIN ;
input signed [24:0] D;
input [3:0] INMODE_0TO3;
input CEA1,CEA2,RSTA,CED,CEAD,RSTD,CLK;
output signed [29:0] ACOUT,X_MUX_A;
output signed [24:0] A_MULT;
wire   signed [29:0] A1_IN ;
wire   signed [29:0] A1_R,A2_R;
wire   signed [24:0] A_MUX_OUT1,A_MUX_OUT2,D_R,PRE_Adder_OUT,AD_R,D_OUT;

////////////////////////// Fixed Part ///////////////////////////////////
assign A1_IN   = (A_INPUT=="DIRECT")?A:ACIN;
assign X_MUX_A = A2_R;
assign ACOUT   = (ACASCREG==2)?A2_R:(ACASCREG==1)?A1_R:A1_IN;
assign A_MUX_OUT1  = (INMODE_0TO3[0])? A1_R[24:0] : X_MUX_A[24:0];
assign A_MUX_OUT2  = A_MUX_OUT1 & {25{~INMODE_0TO3[1]}};
assign A_MULT = (USE_DPORT=="FALSE")? A_MUX_OUT2 : AD_R;
//////////////////////////////////////////////////////////////////////////
generate
if(AREG==2) begin
D_REG #(.N(30)) A1(A1_IN,RSTA,CLK,A1_R,CEA1);
D_REG #(.N(30)) A2(A1_R,RSTA,CLK,A2_R,CEA2);
end
else if (AREG==1) begin
D_REG #(.N(30)) A1(A1_IN,RSTA,CLK,A1_R,CEA1);
assign A2_R=A1_R;
end
else begin
assign A1_R=A1_IN;
assign A2_R=A1_R;
end
endgenerate
generate
if(USE_DPORT=="TRUE") begin
/////////////////////////////
 if(DREG) D_REG #(.N(25)) D1(D,RSTD,CLK,D_R,CED);
 else    assign D_R=D;
/////////////////////////////
 if (ADREG) D_REG #(.N(25)) AD(PRE_Adder_OUT,RSTD,CLK,AD_R,CEAD);
 else assign AD_R=PRE_Adder_OUT;
/////////////////////////////
assign D_OUT = D_R & {25{INMODE_0TO3[2]}};
assign PRE_Adder_OUT = (INMODE_0TO3[3])? D_OUT-A_MUX_OUT2 : D_OUT+A_MUX_OUT2;
end
endgenerate
endmodule //Dual_A&D&PRE_Adder
```

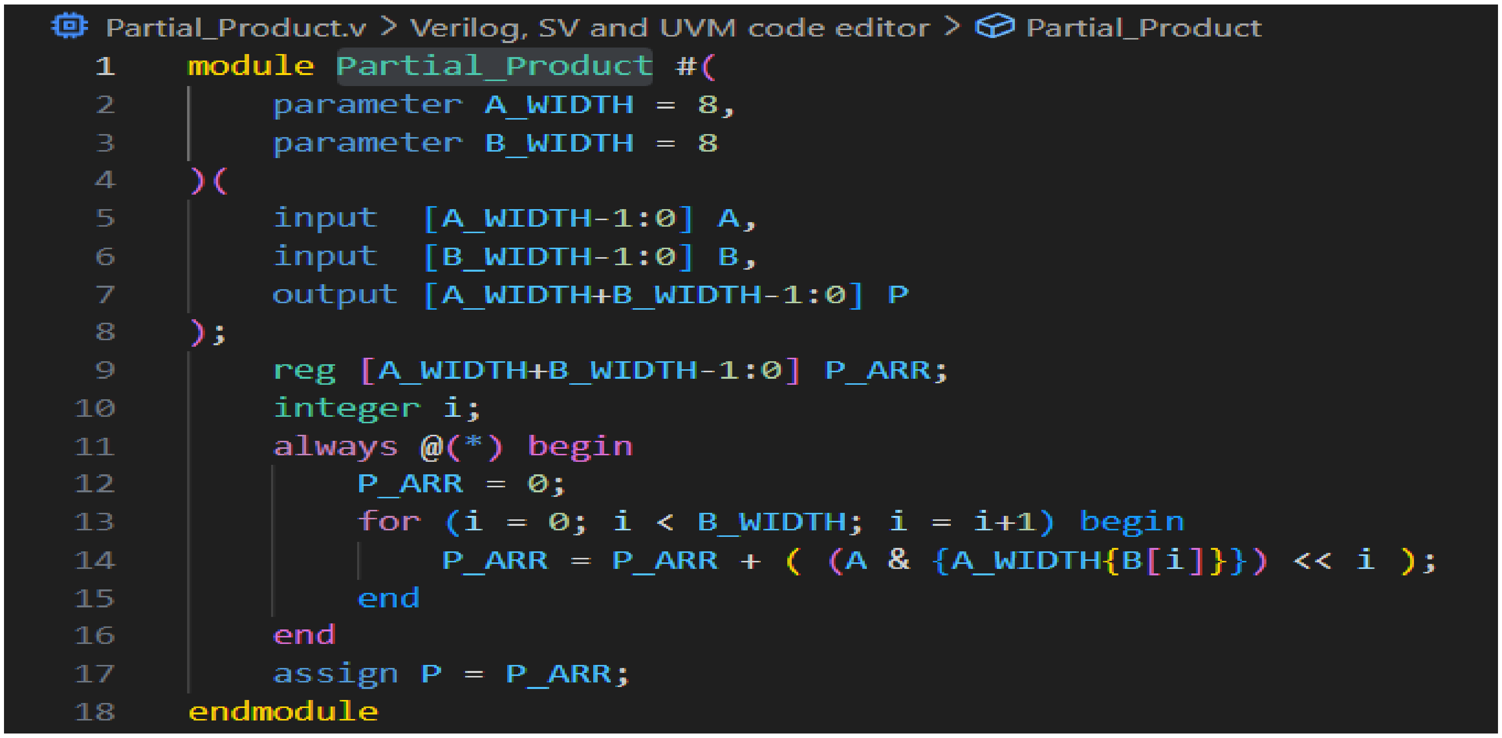
Dual B Module



Dual_B_Register.v > Verilog, SV and UVM code editor > Dual_B_Register

```verilog
module Dual_B_Register (B,BCIN,INMODE_4,BCOUT,B_MULT,X_MUX_B,CLK,CEB1,CEB2,RSTB);
parameter  BREG      = 2; // (0,1,2) Num of B input pipeline regs
parameter  B_INPUT   = "DIRECT";  // "DIRECT" or "CASCADE"
parameter  BCASCREG  = 2; // (0,1,2) Num of B cascade pipeline regs; must be <= BREG
input  signed[17:0] B,BCIN;
input INMODE_4,CLK,CEB1,CEB2,RSTB;
output signed [17:0] BCOUT,B_MULT,X_MUX_B;
wire   signed [17:0] B1_IN ;
wire   signed [17:0] B1_R,B2_R;
assign B1_IN  = (B_INPUT=="DIRECT")?B:BCIN;
assign X_MUX_B  = B2_R;
assign B_MULT = (INMODE_4)? B1_R : X_MUX_B;
assign BCOUT  = (BCASCREG==2)?B2_R:(BCASCREG==1)?B1_R:B1_IN;
generate
if(BREG==2) begin
D_REG #(.N(18)) B1(B1_IN,RSTB,CLK,B1_R,CEB1);
D_REG #(.N(18)) B2(B1_R,RSTB,CLK,B2_R,CEB2);
end
else if (BREG==1) begin
D_REG #(.N(18)) B1(B1_IN,RSTB,CLK,B1_R,CEB1);
assign B2_R=B1_R ;
end
else begin
assign  B1_R=B1_IN;
assign  B2_R=B1_R;
end
endgenerate
endmodule //Dual_B_Register
```

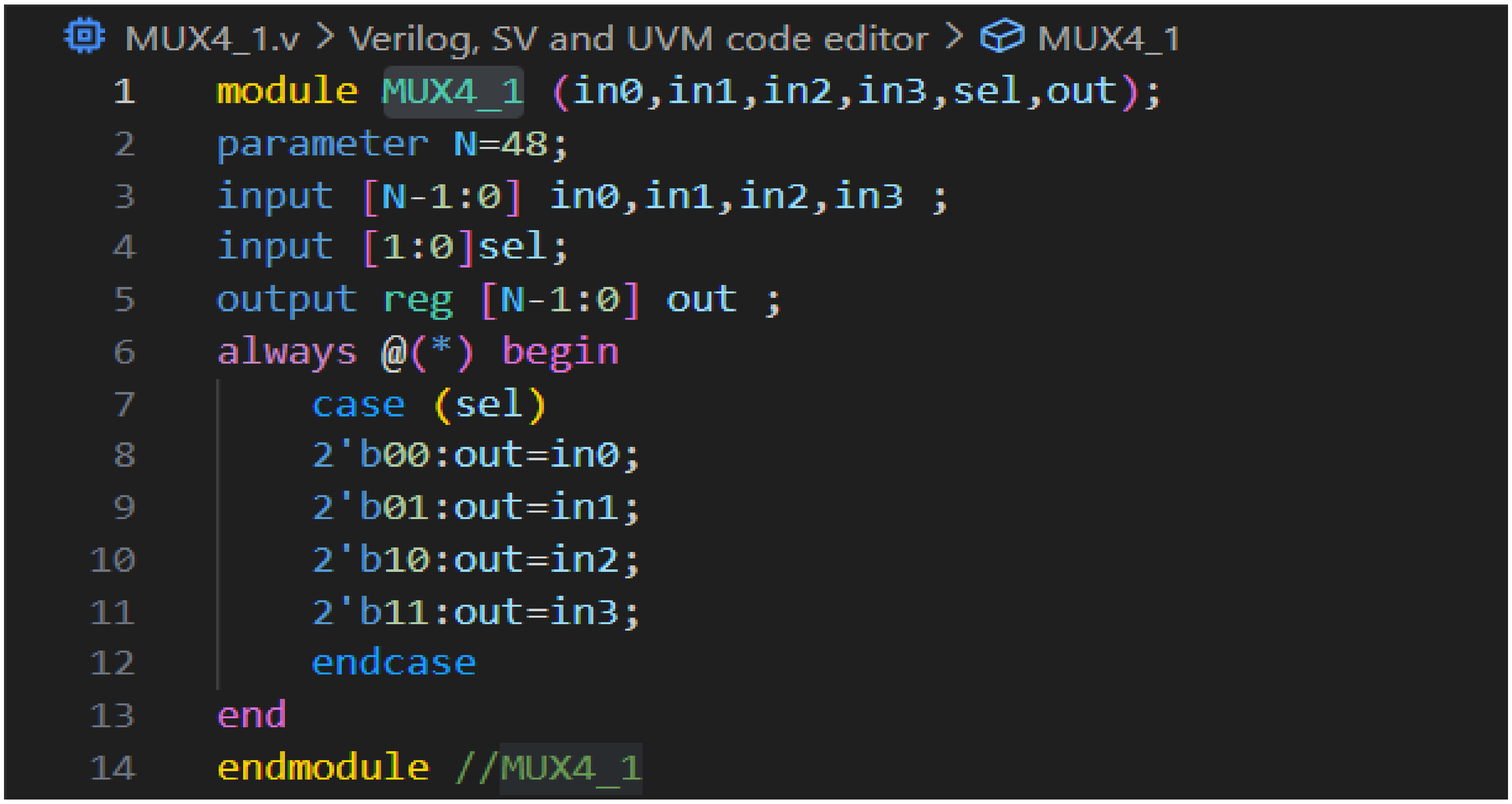
Partial Product Module



Partial_Product.v > Verilog, SV and UVM code editor > Partial_Product

```verilog
module Partial_Product #(
    parameter A_WIDTH = 8,
    parameter B_WIDTH = 8
)(
    input  [A_WIDTH-1:0] A,
    input  [B_WIDTH-1:0] B,
    output [A_WIDTH+B_WIDTH-1:0] P
);
    reg [A_WIDTH+B_WIDTH-1:0] P_ARR;
    integer i;
    always @(*) begin
        P_ARR = 0;
        for (i = 0; i < B_WIDTH; i = i+1) begin
            P_ARR = P_ARR + ( (A & {A_WIDTH{B[i]}}) << i );
        end
    end
    assign P = P_ARR;
endmodule
```

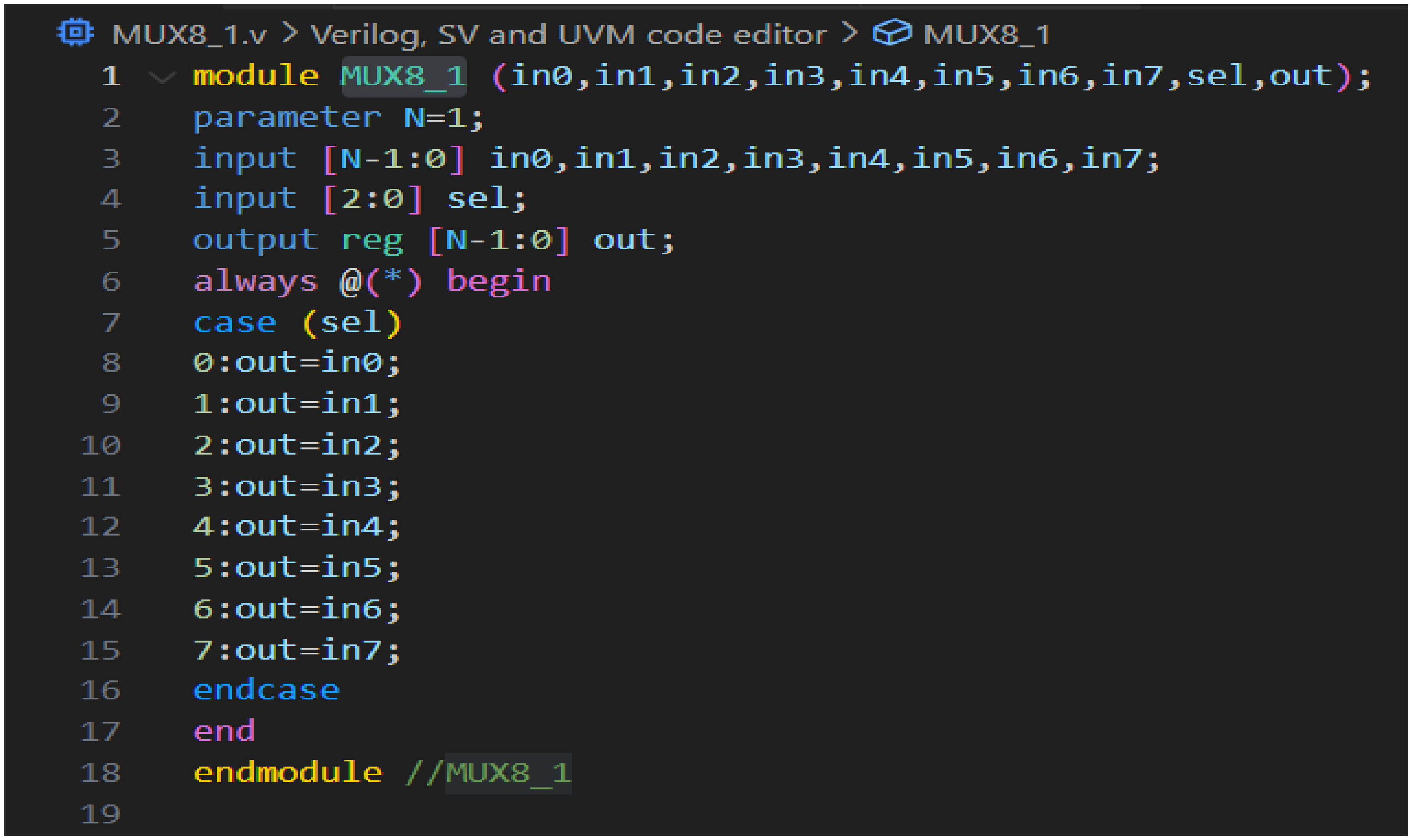
Mux4_1 Module



MUX4_1.v > Verilog, SV and UVM code editor > MUX4_1

```verilog
module MUX4_1 (in0,in1,in2,in3,sel,out);
parameter N=48;
input [N-1:0] in0,in1,in2,in3 ;
input [1:0]sel;
output reg [N-1:0] out ;
always @(*) begin
    case (sel)
    2'b00:out=in0;
    2'b01:out=in1;
    2'b10:out=in2;
    2'b11:out=in3;
    endcase
end
endmodule //MUX4_1
```

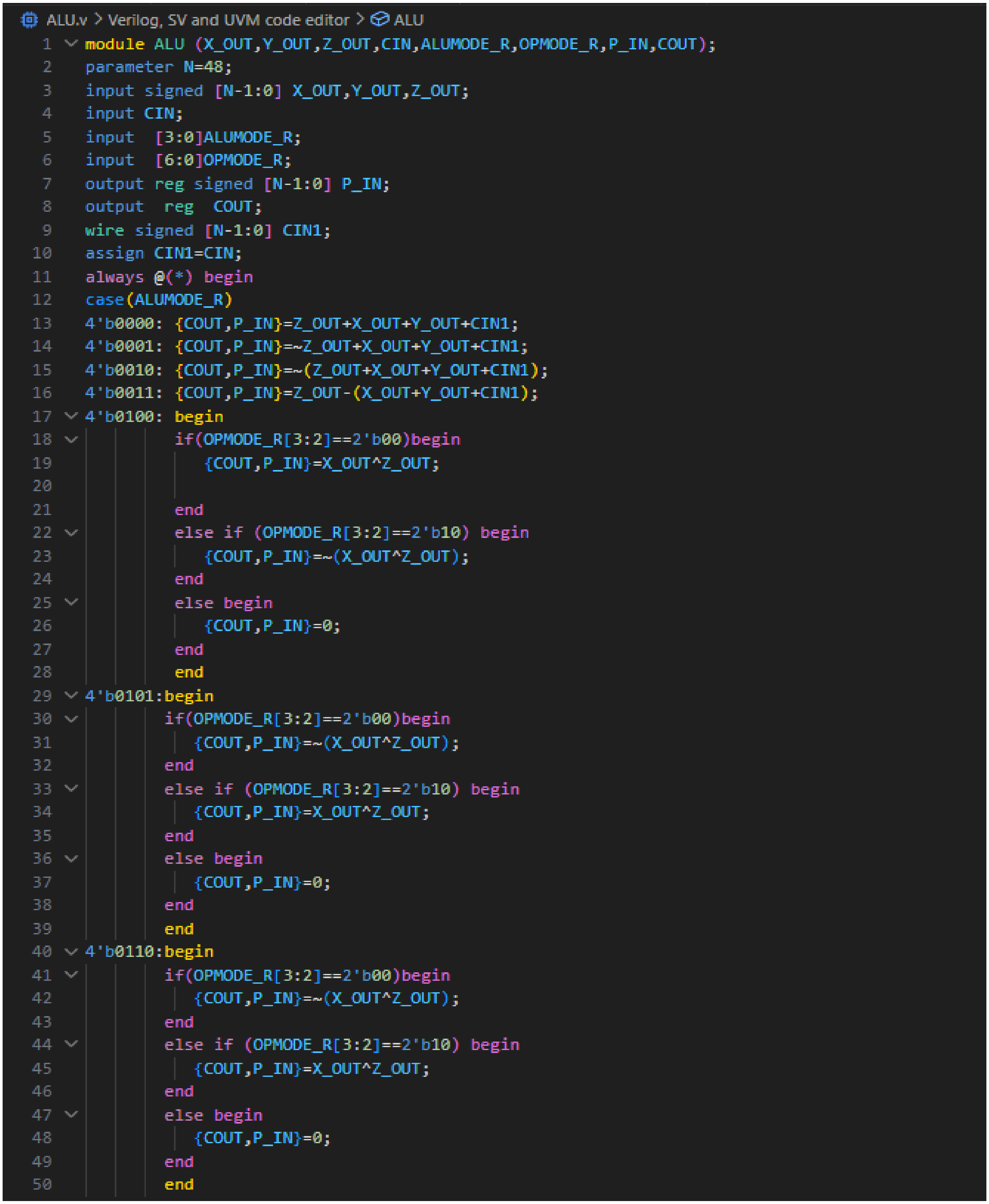
Mux8_1 Module



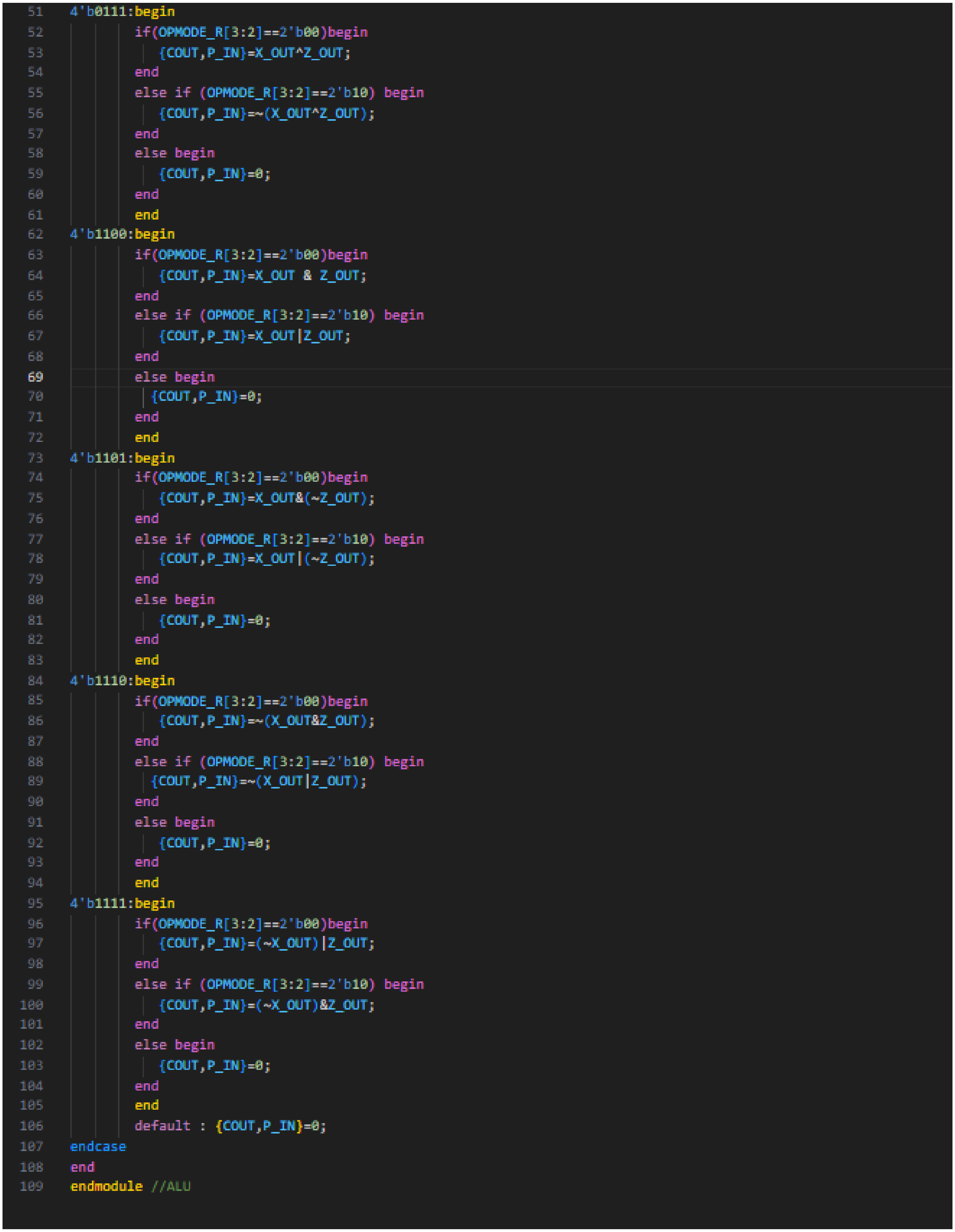
MUX8_1.v > Verilog, SV and UVM code editor > MUX8_1

```verilog
module MUX8_1 (in0,in1,in2,in3,in4,in5,in6,in7,sel,out);
parameter N=1;
input [N-1:0] in0,in1,in2,in3,in4,in5,in6,in7;
input [2:0] sel;
output reg [N-1:0] out;
always @(*) begin
case (sel)
0:out=in0;
1:out=in1;
2:out=in2;
3:out=in3;
4:out=in4;
5:out=in5;
6:out=in6;
7:out=in7;
endcase
end
endmodule //MUX8_1

```

ALU Module

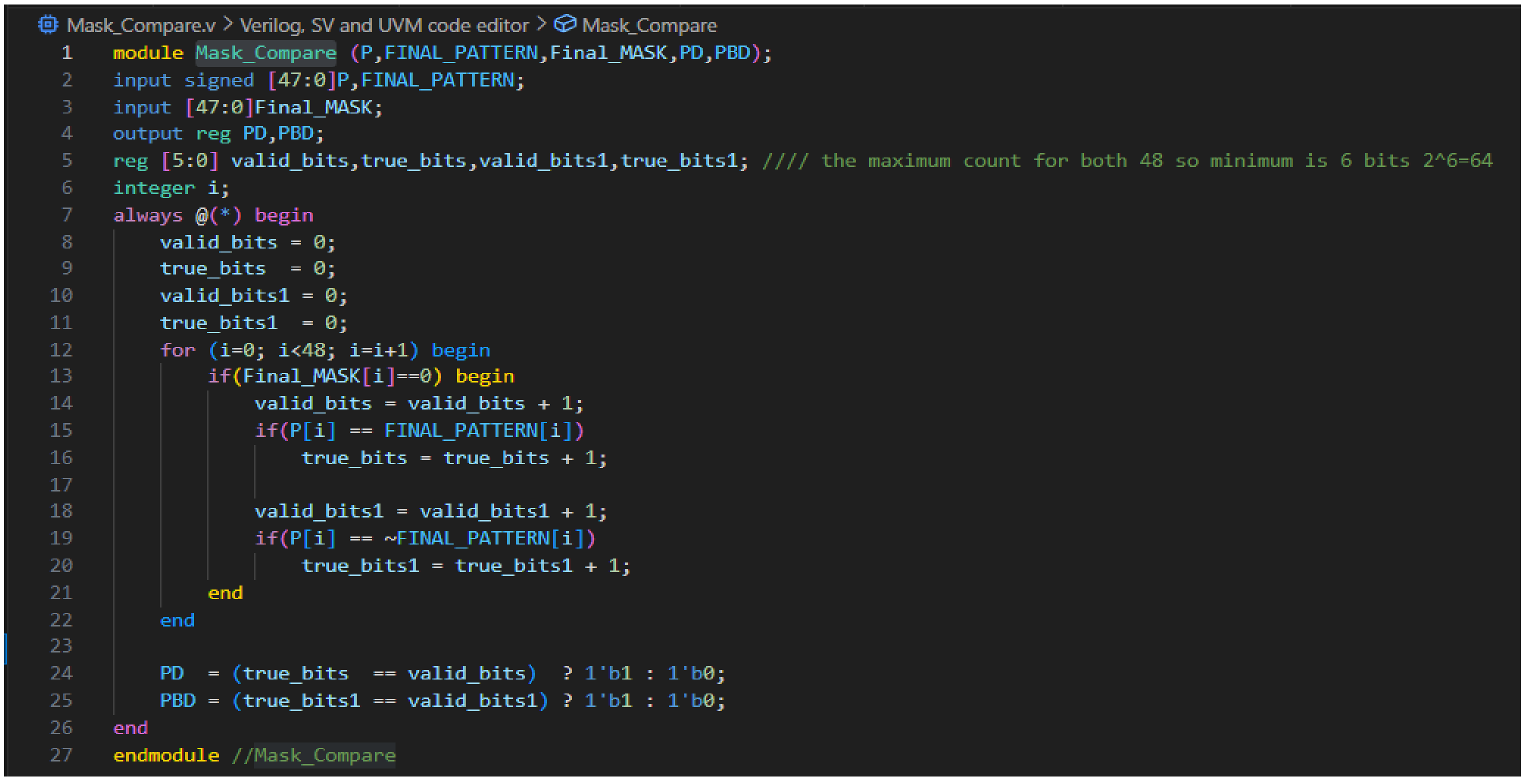


```verilog
module ALU (X_OUT,Y_OUT,Z_OUT,CIN,ALUMODE_R,OPMODE_R,P_IN,COUT);
parameter N=48;
input signed [N-1:0] X_OUT,Y_OUT,Z_OUT;
input CIN;
input  [3:0]ALUMODE_R;
input  [6:0]OPMODE_R;
output reg signed [N-1:0] P_IN;
output  reg  COUT;
wire signed [N-1:0] CIN1;
assign CIN1=CIN;
always @(*) begin
case(ALUMODE_R)
4'b0000: {COUT,P_IN}=Z_OUT+X_OUT+Y_OUT+CIN1;
4'b0001: {COUT,P_IN}=~Z_OUT+X_OUT+Y_OUT+CIN1;
4'b0010: {COUT,P_IN}=~(Z_OUT+X_OUT+Y_OUT+CIN1);
4'b0011: {COUT,P_IN}=Z_OUT-(X_OUT+Y_OUT+CIN1);
4'b0100: begin
          if(OPMODE_R[3:2]==2'b00)begin
             {COUT,P_IN}=X_OUT^Z_OUT;

          end
          else if (OPMODE_R[3:2]==2'b10) begin
             {COUT,P_IN}=~(X_OUT^Z_OUT);
          end
          else begin
             {COUT,P_IN}=0;
          end
          end
4'b0101:begin
         if(OPMODE_R[3:2]==2'b00)begin
            {COUT,P_IN}=~(X_OUT^Z_OUT);
         end
         else if (OPMODE_R[3:2]==2'b10) begin
            {COUT,P_IN}=X_OUT^Z_OUT;
         end
         else begin
            {COUT,P_IN}=0;
         end
         end
4'b0110:begin
         if(OPMODE_R[3:2]==2'b00)begin
            {COUT,P_IN}=~(X_OUT^Z_OUT);
         end
         else if (OPMODE_R[3:2]==2'b10) begin
            {COUT,P_IN}=X_OUT^Z_OUT;
         end
         else begin
            {COUT,P_IN}=0;
         end
         end
```



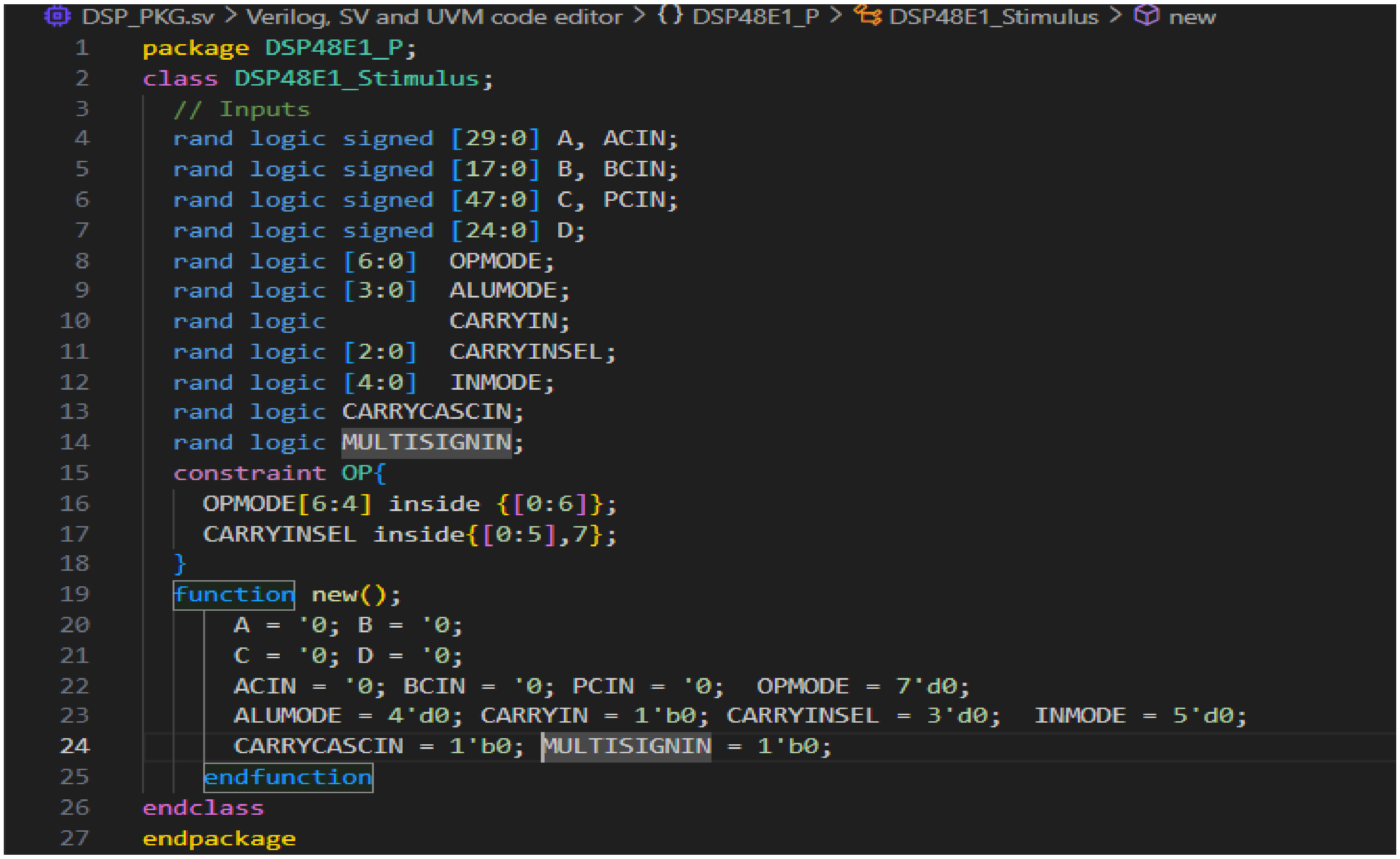
```
4'b0111:begin
        if(OPMODE_R[3:2]==2'b00)begin
           {COUT,P_IN}=X_OUT^Z_OUT;
        end
        else if (OPMODE_R[3:2]==2'b10) begin
           {COUT,P_IN}=~(X_OUT^Z_OUT);
        end
        else begin
           {COUT,P_IN}=0;
        end
        end
4'b1100:begin
        if(OPMODE_R[3:2]==2'b00)begin
           {COUT,P_IN}=X_OUT & Z_OUT;
        end
        else if (OPMODE_R[3:2]==2'b10) begin
           {COUT,P_IN}=X_OUT|Z_OUT;
        end
        else begin
          {COUT,P_IN}=0;
        end
        end
4'b1101:begin
        if(OPMODE_R[3:2]==2'b00)begin
           {COUT,P_IN}=X_OUT&(~Z_OUT);
        end
        else if (OPMODE_R[3:2]==2'b10) begin
           {COUT,P_IN}=X_OUT|(~Z_OUT);
        end
        else begin
           {COUT,P_IN}=0;
        end
        end
4'b1110:begin
        if(OPMODE_R[3:2]==2'b00)begin
           {COUT,P_IN}=~(X_OUT&Z_OUT);
        end
        else if (OPMODE_R[3:2]==2'b10) begin
          {COUT,P_IN}=~(X_OUT|Z_OUT);
        end
        else begin
           {COUT,P_IN}=0;
        end
        end
4'b1111:begin
        if(OPMODE_R[3:2]==2'b00)begin
           {COUT,P_IN}=(~X_OUT)|Z_OUT;
        end
        else if (OPMODE_R[3:2]==2'b10) begin
           {COUT,P_IN}=(~X_OUT)&Z_OUT;
        end
        else begin
           {COUT,P_IN}=0;
        end
        end
        default : {COUT,P_IN}=0;
endcase
end
endmodule //ALU
```

Mask Comparing Module


Mask_Compare.v > Verilog, SV and UVM code editor > Mask_Compare

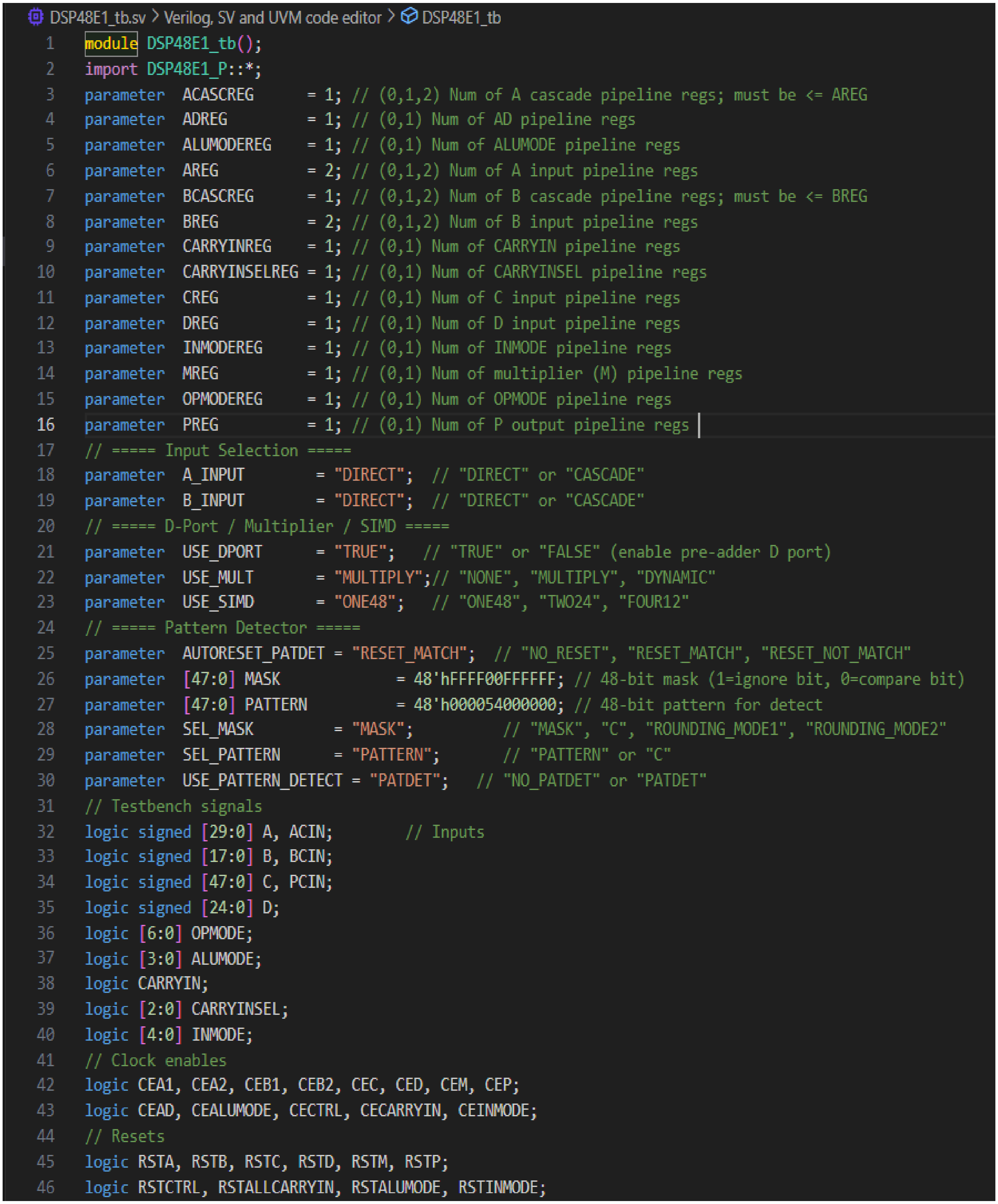
```verilog
module Mask_Compare (P,FINAL_PATTERN,Final_MASK,PD,PBD);
input signed [47:0]P,FINAL_PATTERN;
input [47:0]Final_MASK;
output reg PD,PBD;
reg [5:0] valid_bits,true_bits,valid_bits1,true_bits1; //// the maximum count for both 48 so minimum is 6 bits 2^6=64
integer i;
always @(*) begin
    valid_bits = 0;
    true_bits  = 0;
    valid_bits1 = 0;
    true_bits1  = 0;
    for (i=0; i<48; i=i+1) begin
        if(Final_MASK[i]==0) begin
            valid_bits = valid_bits + 1;
            if(P[i] == FINAL_PATTERN[i])
                true_bits = true_bits + 1;

            valid_bits1 = valid_bits1 + 1;
            if(P[i] == ~FINAL_PATTERN[i])
                true_bits1 = true_bits1 + 1;
        end
    end

    PD  = (true_bits  == valid_bits)  ? 1'b1 : 1'b0;
    PBD = (true_bits1 == valid_bits1) ? 1'b1 : 1'b0;
end
endmodule //Mask_Compare
```

Package For Testbench

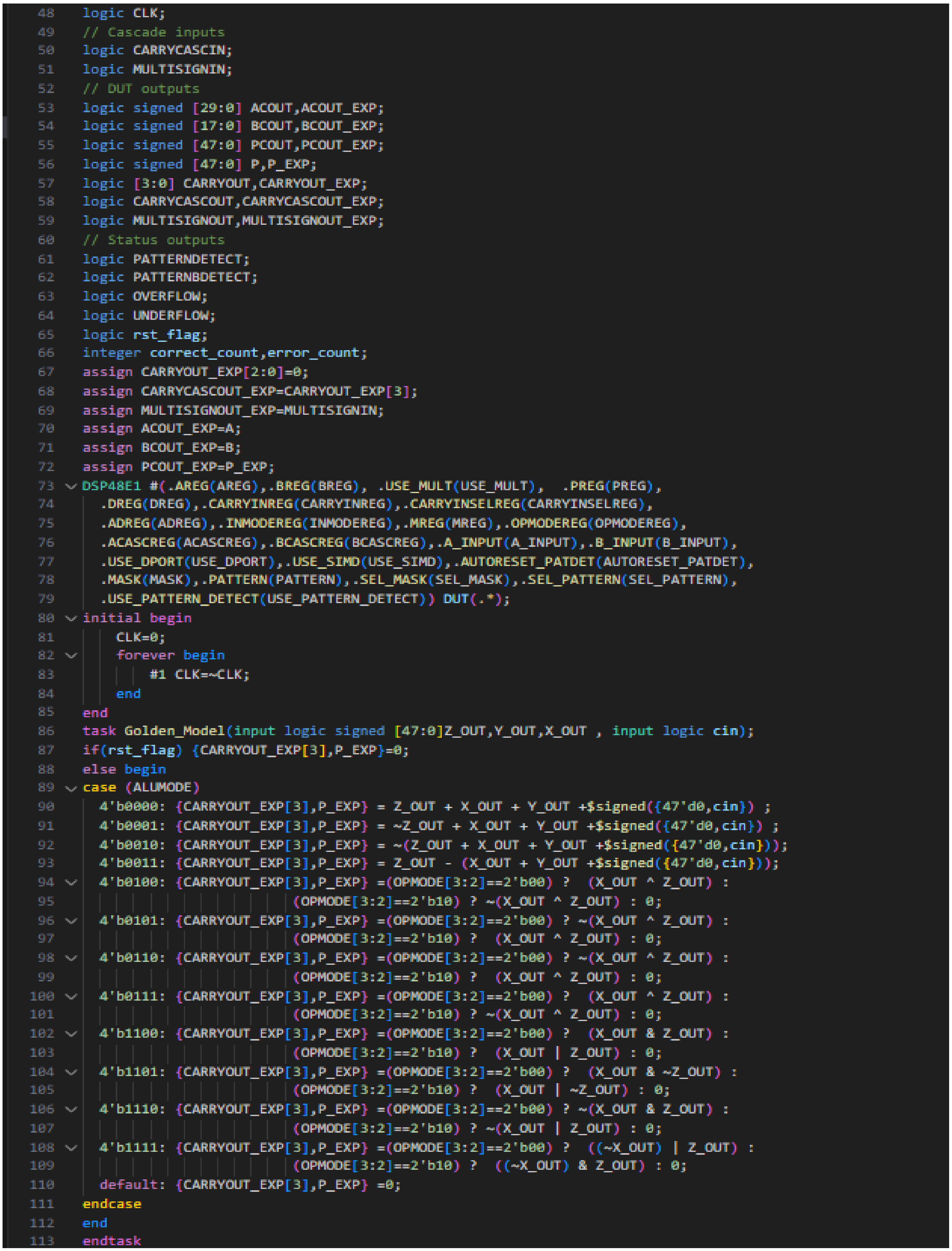

DSP_PKG.sv > Verilog, SV and UVM code editor > {} DSP48E1_P > DSP48E1_Stimulus > new

```systemverilog
package DSP48E1_P;
class DSP48E1_Stimulus;
  // Inputs
  rand logic signed [29:0] A, ACIN;
  rand logic signed [17:0] B, BCIN;
  rand logic signed [47:0] C, PCIN;
  rand logic signed [24:0] D;
  rand logic [6:0]  OPMODE;
  rand logic [3:0]  ALUMODE;
  rand logic        CARRYIN;
  rand logic [2:0]  CARRYINSEL;
  rand logic [4:0]  INMODE;
  rand logic CARRYCASCIN;
  rand logic MULTISIGNIN;
  constraint OP{
    OPMODE[6:4] inside {[0:6]};
    CARRYINSEL inside{[0:5],7};
  }
  function new();
      A = '0; B = '0;
      C = '0; D = '0;
      ACIN = '0; BCIN = '0; PCIN = '0;  OPMODE = 7'd0;
      ALUMODE = 4'd0; CARRYIN = 1'b0; CARRYINSEL = 3'd0;  INMODE = 5'd0;
      CARRYCASCIN = 1'b0; MULTISIGNIN = 1'b0;
    endfunction
endclass
endpackage
```

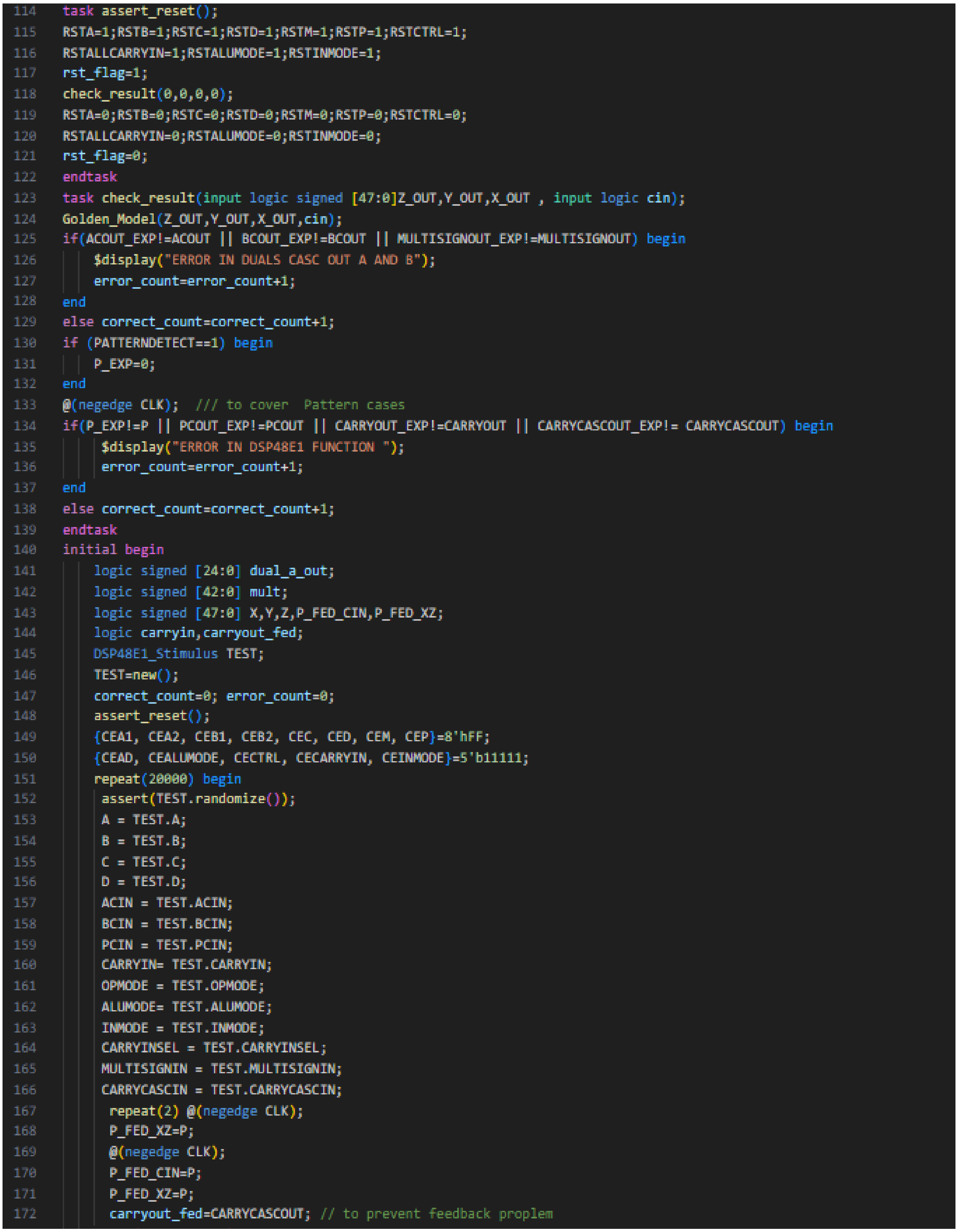
Testbench (Fully Verified)


DSP48E1_tb.sv > Verilog, SV and UVM code editor > DSP48E1_tb

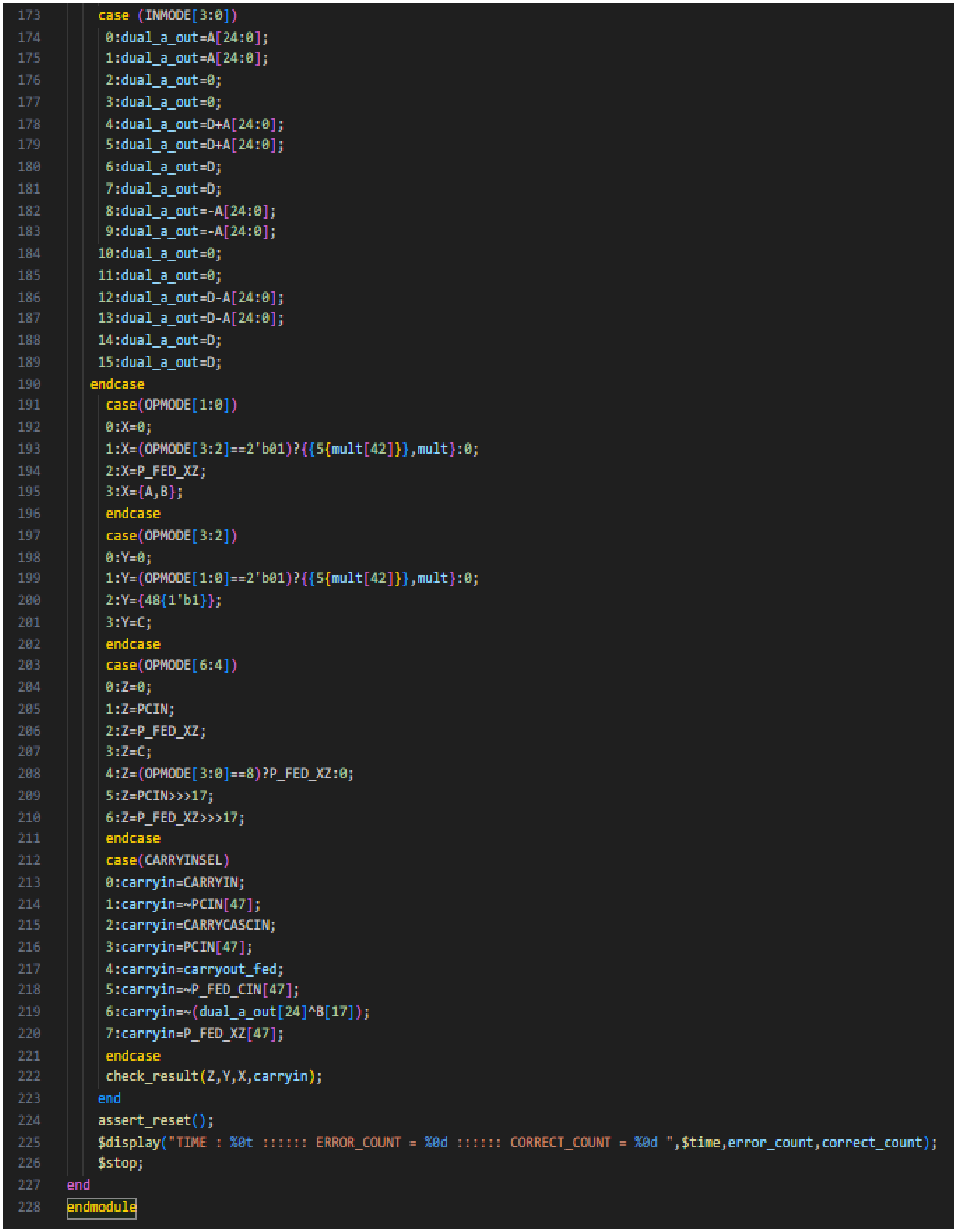
```systemverilog
module DSP48E1_tb();
import DSP48E1_P::*;
parameter  ACASCREG     = 1; // (0,1,2) Num of A cascade pipeline regs; must be <= AREG
parameter  ADREG        = 1; // (0,1) Num of AD pipeline regs
parameter  ALUMODEREG   = 1; // (0,1) Num of ALUMODE pipeline regs
parameter  AREG         = 2; // (0,1,2) Num of A input pipeline regs
parameter  BCASCREG     = 1; // (0,1,2) Num of B cascade pipeline regs; must be <= BREG
parameter  BREG         = 2; // (0,1,2) Num of B input pipeline regs
parameter  CARRYINREG   = 1; // (0,1) Num of CARRYIN pipeline regs
parameter  CARRYINSELREG = 1; // (0,1) Num of CARRYINSEL pipeline regs
parameter  CREG         = 1; // (0,1) Num of C input pipeline regs
parameter  DREG         = 1; // (0,1) Num of D input pipeline regs
parameter  INMODEREG    = 1; // (0,1) Num of INMODE pipeline regs
parameter  MREG         = 1; // (0,1) Num of multiplier (M) pipeline regs
parameter  OPMODEREG    = 1; // (0,1) Num of OPMODE pipeline regs
parameter  PREG         = 1; // (0,1) Num of P output pipeline regs
// ===== Input Selection =====
parameter  A_INPUT       = "DIRECT";  // "DIRECT" or "CASCADE"
parameter  B_INPUT       = "DIRECT";  // "DIRECT" or "CASCADE"
// ===== D-Port / Multiplier / SIMD =====
parameter  USE_DPORT     = "TRUE";   // "TRUE" or "FALSE" (enable pre-adder D port)
parameter  USE_MULT      = "MULTIPLY";// "NONE", "MULTIPLY", "DYNAMIC"
parameter  USE_SIMD      = "ONE48";   // "ONE48", "TWO24", "FOUR12"
// ===== Pattern Detector =====
parameter  AUTORESET_PATDET = "RESET_MATCH";  // "NO_RESET", "RESET_MATCH", "RESET_NOT_MATCH"
parameter  [47:0] MASK           = 48'hFFFF00FFFFFF; // 48-bit mask (1=ignore bit, 0=compare bit)
parameter  [47:0] PATTERN        = 48'h000054000000; // 48-bit pattern for detect
parameter  SEL_MASK         = "MASK";          // "MASK", "C", "ROUNDING_MODE1", "ROUNDING_MODE2"
parameter  SEL_PATTERN      = "PATTERN";       // "PATTERN" or "C"
parameter  USE_PATTERN_DETECT = "PATDET";   // "NO_PATDET" or "PATDET"
// Testbench signals
logic signed [29:0] A, ACIN;        // Inputs
logic signed [17:0] B, BCIN;
logic signed [47:0] C, PCIN;
logic signed [24:0] D;
logic [6:0] OPMODE;
logic [3:0] ALUMODE;
logic CARRYIN;
logic [2:0] CARRYINSEL;
logic [4:0] INMODE;
// Clock enables
logic CEA1, CEA2, CEB1, CEB2, CEC, CED, CEM, CEP;
logic CEAD, CEALUMODE, CECTRL, CECARRYIN, CEINMODE;
// Resets
logic RSTA, RSTB, RSTC, RSTD, RSTM, RSTP;
logic RSTCTRL, RSTALLCARRYIN, RSTALUMODE, RSTINMODE;
```



```
logic CLK;
// Cascade inputs
logic CARRYCASCIN;
logic MULTISIGNIN;
// DUT outputs
logic signed [29:0] ACOUT,ACOUT_EXP;
logic signed [17:0] BCOUT,BCOUT_EXP;
logic signed [47:0] PCOUT,PCOUT_EXP;
logic signed [47:0] P,P_EXP;
logic [3:0] CARRYOUT,CARRYOUT_EXP;
logic CARRYCASCOUT,CARRYCASCOUT_EXP;
logic MULTISIGNOUT,MULTISIGNOUT_EXP;
// Status outputs
logic PATTERNDETECT;
logic PATTERNBDETECT;
logic OVERFLOW;
logic UNDERFLOW;
logic rst_flag;
integer correct_count,error_count;
assign CARRYOUT_EXP[2:0]=0;
assign CARRYCASCOUT_EXP=CARRYOUT_EXP[3];
assign MULTISIGNOUT_EXP=MULTISIGNIN;
assign ACOUT_EXP=A;
assign BCOUT_EXP=B;
assign PCOUT_EXP=P_EXP;
DSP48E1 #(.AREG(AREG),.BREG(BREG), .USE_MULT(USE_MULT),  .PREG(PREG),
  .DREG(DREG),.CARRYINREG(CARRYINREG),.CARRYINSELREG(CARRYINSELREG),
  .ADREG(ADREG),.INMODEREG(INMODEREG),.MREG(MREG),.OPMODEREG(OPMODEREG),
  .ACASCREG(ACASCREG),.BCASCREG(BCASCREG),.A_INPUT(A_INPUT),.B_INPUT(B_INPUT),
  .USE_DPORT(USE_DPORT),.USE_SIMD(USE_SIMD),.AUTORESET_PATDET(AUTORESET_PATDET),
  .MASK(MASK),.PATTERN(PATTERN),.SEL_MASK(SEL_MASK),.SEL_PATTERN(SEL_PATTERN),
  .USE_PATTERN_DETECT(USE_PATTERN_DETECT)) DUT(.*);
initial begin
    CLK=0;
    forever begin
        #1 CLK=~CLK;
    end
end
task Golden_Model(input logic signed [47:0]Z_OUT,Y_OUT,X_OUT , input logic cin);
if(rst_flag) {CARRYOUT_EXP[3],P_EXP}=0;
else begin
case (ALUMODE)
  4'b0000: {CARRYOUT_EXP[3],P_EXP} = Z_OUT + X_OUT + Y_OUT +$signed({47'd0,cin}) ;
  4'b0001: {CARRYOUT_EXP[3],P_EXP} = ~Z_OUT + X_OUT + Y_OUT +$signed({47'd0,cin}) ;
  4'b0010: {CARRYOUT_EXP[3],P_EXP} = ~(Z_OUT + X_OUT + Y_OUT +$signed({47'd0,cin}));
  4'b0011: {CARRYOUT_EXP[3],P_EXP} = Z_OUT - (X_OUT + Y_OUT +$signed({47'd0,cin}));
  4'b0100: {CARRYOUT_EXP[3],P_EXP} =(OPMODE[3:2]==2'b00) ?  (X_OUT ^ Z_OUT) :
                         (OPMODE[3:2]==2'b10) ? ~(X_OUT ^ Z_OUT) : 0;
  4'b0101: {CARRYOUT_EXP[3],P_EXP} =(OPMODE[3:2]==2'b00) ? ~(X_OUT ^ Z_OUT) :
                         (OPMODE[3:2]==2'b10) ?  (X_OUT ^ Z_OUT) : 0;
  4'b0110: {CARRYOUT_EXP[3],P_EXP} =(OPMODE[3:2]==2'b00) ? ~(X_OUT ^ Z_OUT) :
                         (OPMODE[3:2]==2'b10) ?  (X_OUT ^ Z_OUT) : 0;
  4'b0111: {CARRYOUT_EXP[3],P_EXP} =(OPMODE[3:2]==2'b00) ?  (X_OUT ^ Z_OUT) :
                         (OPMODE[3:2]==2'b10) ? ~(X_OUT ^ Z_OUT) : 0;
  4'b1100: {CARRYOUT_EXP[3],P_EXP} =(OPMODE[3:2]==2'b00) ?  (X_OUT & Z_OUT) :
                         (OPMODE[3:2]==2'b10) ?  (X_OUT | Z_OUT) : 0;
  4'b1101: {CARRYOUT_EXP[3],P_EXP} =(OPMODE[3:2]==2'b00) ?  (X_OUT & ~Z_OUT) :
                         (OPMODE[3:2]==2'b10) ?  (X_OUT | ~Z_OUT) : 0;
  4'b1110: {CARRYOUT_EXP[3],P_EXP} =(OPMODE[3:2]==2'b00) ? ~(X_OUT & Z_OUT) :
                         (OPMODE[3:2]==2'b10) ? ~(X_OUT | Z_OUT) : 0;
  4'b1111: {CARRYOUT_EXP[3],P_EXP} =(OPMODE[3:2]==2'b00) ?  ((~X_OUT) | Z_OUT) :
                         (OPMODE[3:2]==2'b10) ?  ((~X_OUT) & Z_OUT) : 0;
  default: {CARRYOUT_EXP[3],P_EXP} =0;
endcase
end
endtask
```

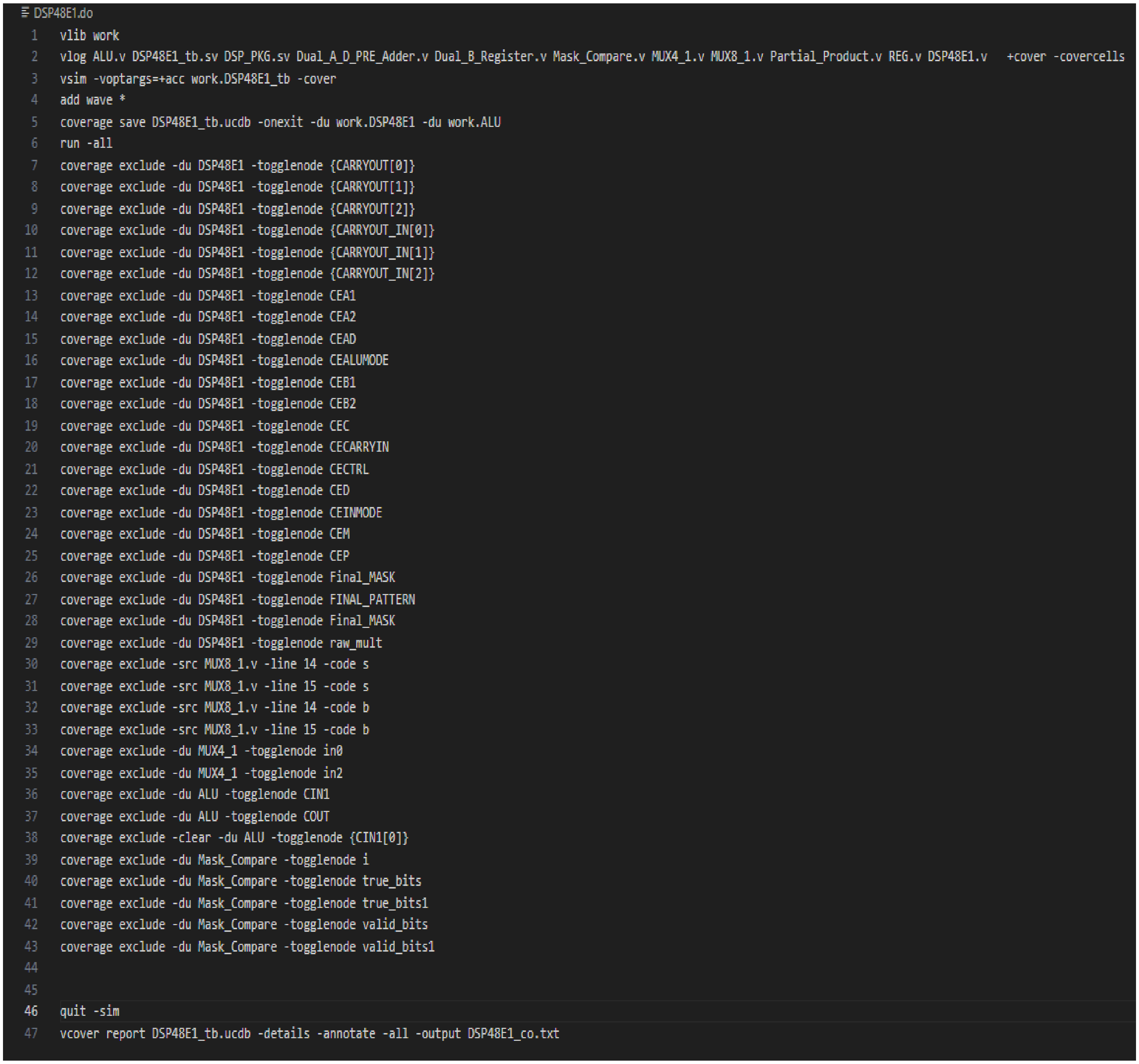


```
task assert_reset();
RSTA=1;RSTB=1;RSTC=1;RSTD=1;RSTM=1;RSTP=1;RSTCTRL=1;
RSTALLCARRYIN=1;RSTALUMODE=1;RSTINMODE=1;
rst_flag=1;
check_result(0,0,0,0);
RSTA=0;RSTB=0;RSTC=0;RSTD=0;RSTM=0;RSTP=0;RSTCTRL=0;
RSTALLCARRYIN=0;RSTALUMODE=0;RSTINMODE=0;
rst_flag=0;
endtask
task check_result(input logic signed [47:0]Z_OUT,Y_OUT,X_OUT , input logic cin);
Golden_Model(Z_OUT,Y_OUT,X_OUT,cin);
if(ACOUT_EXP!=ACOUT || BCOUT_EXP!=BCOUT || MULTISIGNOUT_EXP!=MULTISIGNOUT) begin
    $display("ERROR IN DUALS CASC OUT A AND B");
    error_count=error_count+1;
end
else correct_count=correct_count+1;
if (PATTERNDETECT==1) begin
    P_EXP=0;
end
@(negedge CLK);  /// to cover  Pattern cases
if(P_EXP!=P || PCOUT_EXP!=PCOUT || CARRYOUT_EXP!=CARRYOUT || CARRYCASCOUT_EXP!= CARRYCASCOUT) begin
     $display("ERROR IN DSP48E1 FUNCTION ");
     error_count=error_count+1;
end
else correct_count=correct_count+1;
endtask
initial begin
    logic signed [24:0] dual_a_out;
    logic signed [42:0] mult;
    logic signed [47:0] X,Y,Z,P_FED_CIN,P_FED_XZ;
    logic carryin,carryout_fed;
    DSP48E1_Stimulus TEST;
    TEST=new();
    correct_count=0; error_count=0;
    assert_reset();
    {CEA1, CEA2, CEB1, CEB2, CEC, CED, CEM, CEP}=8'hFF;
    {CEAD, CEALUMODE, CECTRL, CECARRYIN, CEINMODE}=5'b11111;
    repeat(20000) begin
     assert(TEST.randomize());
     A = TEST.A;
     B = TEST.B;
     C = TEST.C;
     D = TEST.D;
     ACIN = TEST.ACIN;
     BCIN = TEST.BCIN;
     PCIN = TEST.PCIN;
     CARRYIN= TEST.CARRYIN;
     OPMODE = TEST.OPMODE;
     ALUMODE= TEST.ALUMODE;
     INMODE = TEST.INMODE;
     CARRYINSEL = TEST.CARRYINSEL;
     MULTISIGNIN = TEST.MULTISIGNIN;
     CARRYCASCIN = TEST.CARRYCASCIN;
      repeat(2) @(negedge CLK);
      P_FED_XZ=P;
      @(negedge CLK);
      P_FED_CIN=P;
      P_FED_XZ=P;
      carryout_fed=CARRYCASCOUT; // to prevent feedback proplem
```



```
    case (INMODE[3:0])
     0:dual_a_out=A[24:0];
     1:dual_a_out=A[24:0];
     2:dual_a_out=0;
     3:dual_a_out=0;
     4:dual_a_out=D+A[24:0];
     5:dual_a_out=D+A[24:0];
     6:dual_a_out=D;
     7:dual_a_out=D;
     8:dual_a_out=-A[24:0];
     9:dual_a_out=-A[24:0];
    10:dual_a_out=0;
    11:dual_a_out=0;
    12:dual_a_out=D-A[24:0];
    13:dual_a_out=D-A[24:0];
    14:dual_a_out=D;
    15:dual_a_out=D;
   endcase
     case(OPMODE[1:0])
     0:X=0;
     1:X=(OPMODE[3:2]==2'b01)?{{5{mult[42]}},mult}:0;
     2:X=P_FED_XZ;
     3:X={A,B};
     endcase
     case(OPMODE[3:2])
     0:Y=0;
     1:Y=(OPMODE[1:0]==2'b01)?{{5{mult[42]}},mult}:0;
     2:Y={48{1'b1}};
     3:Y=C;
     endcase
     case(OPMODE[6:4])
     0:Z=0;
     1:Z=PCIN;
     2:Z=P_FED_XZ;
     3:Z=C;
     4:Z=(OPMODE[3:0]==8)?P_FED_XZ:0;
     5:Z=PCIN>>>17;
     6:Z=P_FED_XZ>>>17;
     endcase
     case(CARRYINSEL)
     0:carryin=CARRYIN;
     1:carryin=~PCIN[47];
     2:carryin=CARRYCASCIN;
     3:carryin=PCIN[47];
     4:carryin=carryout_fed;
     5:carryin=~P_FED_CIN[47];
     6:carryin=~(dual_a_out[24]^B[17]);
     7:carryin=P_FED_XZ[47];
     endcase
     check_result(Z,Y,X,carryin);
    end
    assert_reset();
    $display("TIME : %0t ::::::: ERROR_COUNT = %0d ::::::: CORRECT_COUNT = %0d ",$time,error_count,correct_count);
    $stop;
end
endmodule
```

Do File

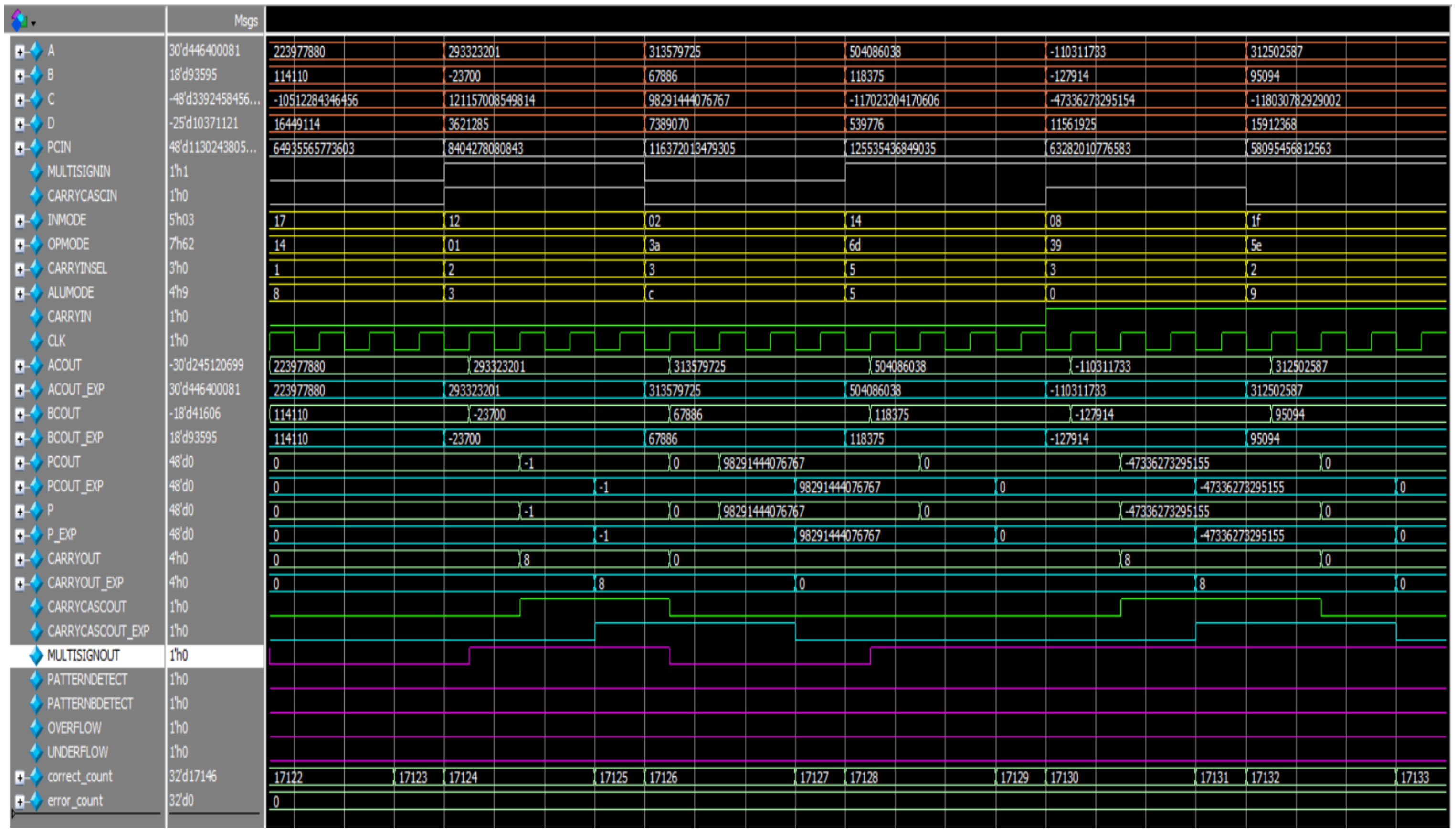


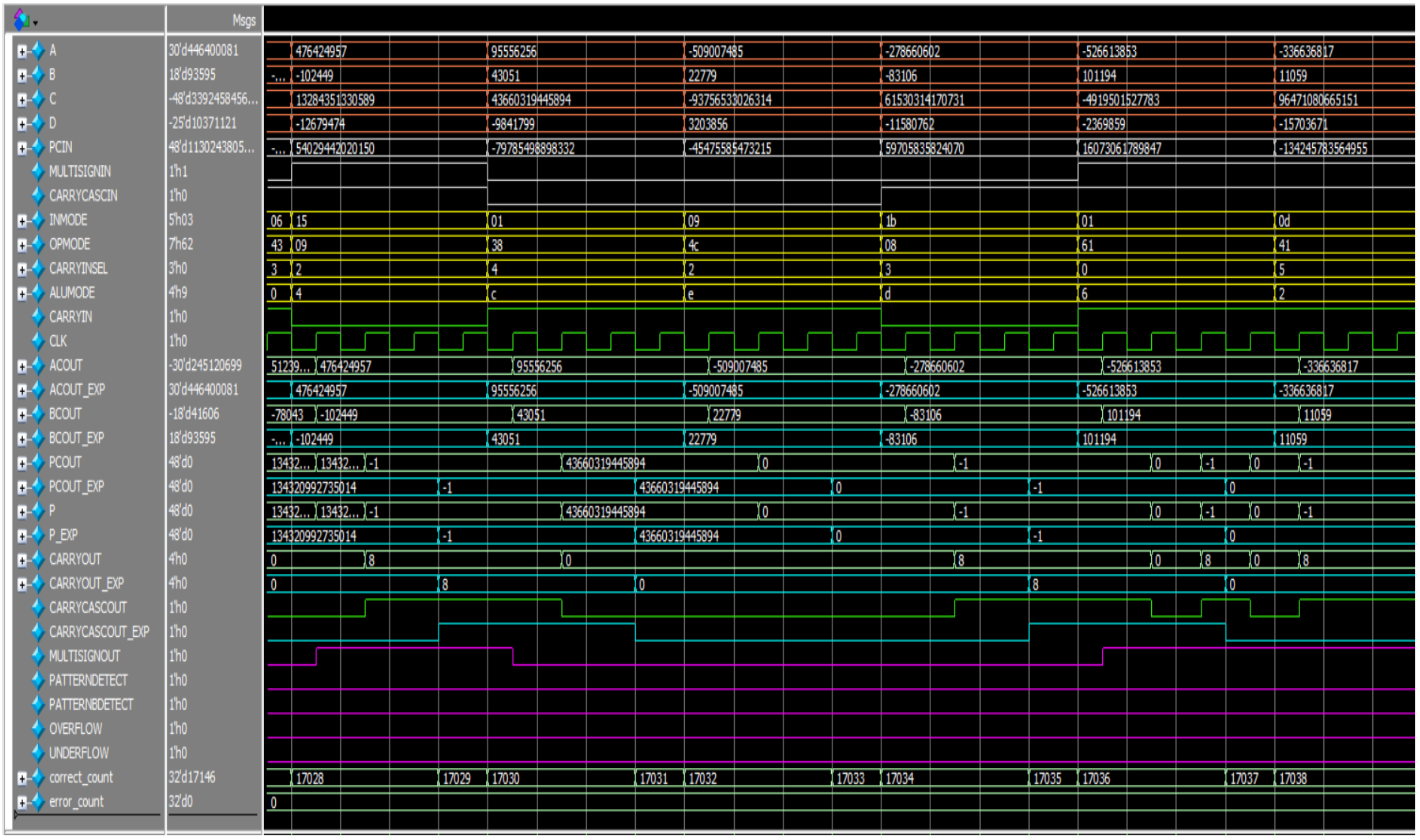
DSP48E1.do

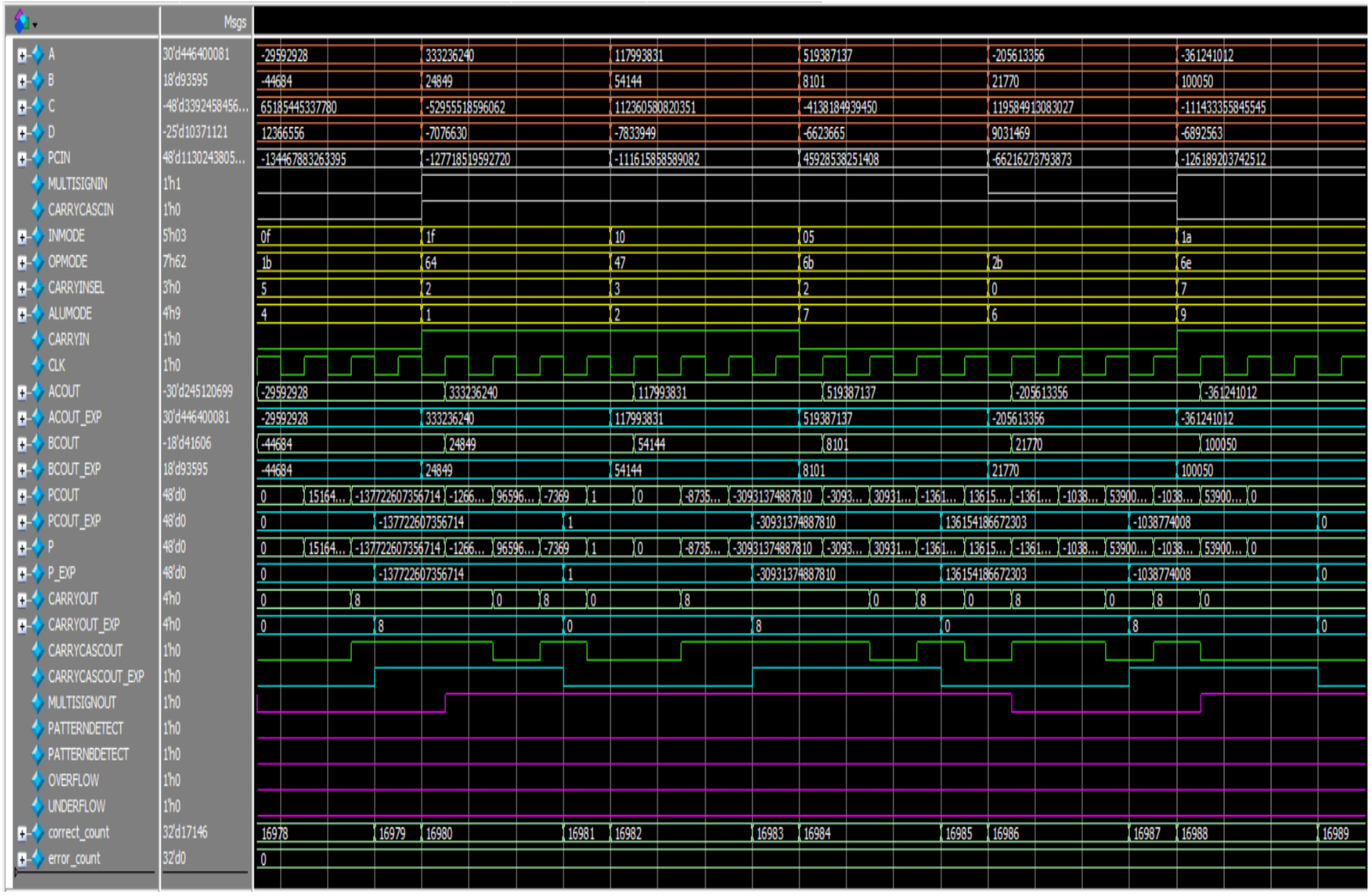
```
vlib work
vlog ALU.v DSP48E1_tb.sv DSP_PKG.sv Dual_A_D_PRE_Adder.v Dual_B_Register.v Mask_Compare.v MUX4_1.v MUX8_1.v Partial_Product.v REG.v DSP48E1.v   +cover -covercells
vsim -voptargs=+acc work.DSP48E1_tb -cover
add wave *
coverage save DSP48E1_tb.ucdb -onexit -du work.DSP48E1 -du work.ALU
run -all
coverage exclude -du DSP48E1 -togglenode {CARRYOUT[0]}
coverage exclude -du DSP48E1 -togglenode {CARRYOUT[1]}
coverage exclude -du DSP48E1 -togglenode {CARRYOUT[2]}
coverage exclude -du DSP48E1 -togglenode {CARRYOUT_IN[0]}
coverage exclude -du DSP48E1 -togglenode {CARRYOUT_IN[1]}
coverage exclude -du DSP48E1 -togglenode {CARRYOUT_IN[2]}
coverage exclude -du DSP48E1 -togglenode CEA1
coverage exclude -du DSP48E1 -togglenode CEA2
coverage exclude -du DSP48E1 -togglenode CEAD
coverage exclude -du DSP48E1 -togglenode CEALUMODE
coverage exclude -du DSP48E1 -togglenode CEB1
coverage exclude -du DSP48E1 -togglenode CEB2
coverage exclude -du DSP48E1 -togglenode CEC
coverage exclude -du DSP48E1 -togglenode CECARRYIN
coverage exclude -du DSP48E1 -togglenode CECTRL
coverage exclude -du DSP48E1 -togglenode CED
coverage exclude -du DSP48E1 -togglenode CEINMODE
coverage exclude -du DSP48E1 -togglenode CEM
coverage exclude -du DSP48E1 -togglenode CEP
coverage exclude -du DSP48E1 -togglenode Final_MASK
coverage exclude -du DSP48E1 -togglenode FINAL_PATTERN
coverage exclude -du DSP48E1 -togglenode Final_MASK
coverage exclude -du DSP48E1 -togglenode raw_mult
coverage exclude -src MUX8_1.v -line 14 -code s
coverage exclude -src MUX8_1.v -line 15 -code s
coverage exclude -src MUX8_1.v -line 14 -code b
coverage exclude -src MUX8_1.v -line 15 -code b
coverage exclude -du MUX4_1 -togglenode in0
coverage exclude -du MUX4_1 -togglenode in2
coverage exclude -du ALU -togglenode CIN1
coverage exclude -du ALU -togglenode COUT
coverage exclude -clear -du ALU -togglenode {CIN1[0]}
coverage exclude -du Mask_Compare -togglenode i
coverage exclude -du Mask_Compare -togglenode true_bits
coverage exclude -du Mask_Compare -togglenode true_bits1
coverage exclude -du Mask_Compare -togglenode valid_bits
coverage exclude -du Mask_Compare -togglenode valid_bits1


quit -sim
vcover report DSP48E1_tb.ucdb -details -annotate -all -output DSP48E1_co.txt
```

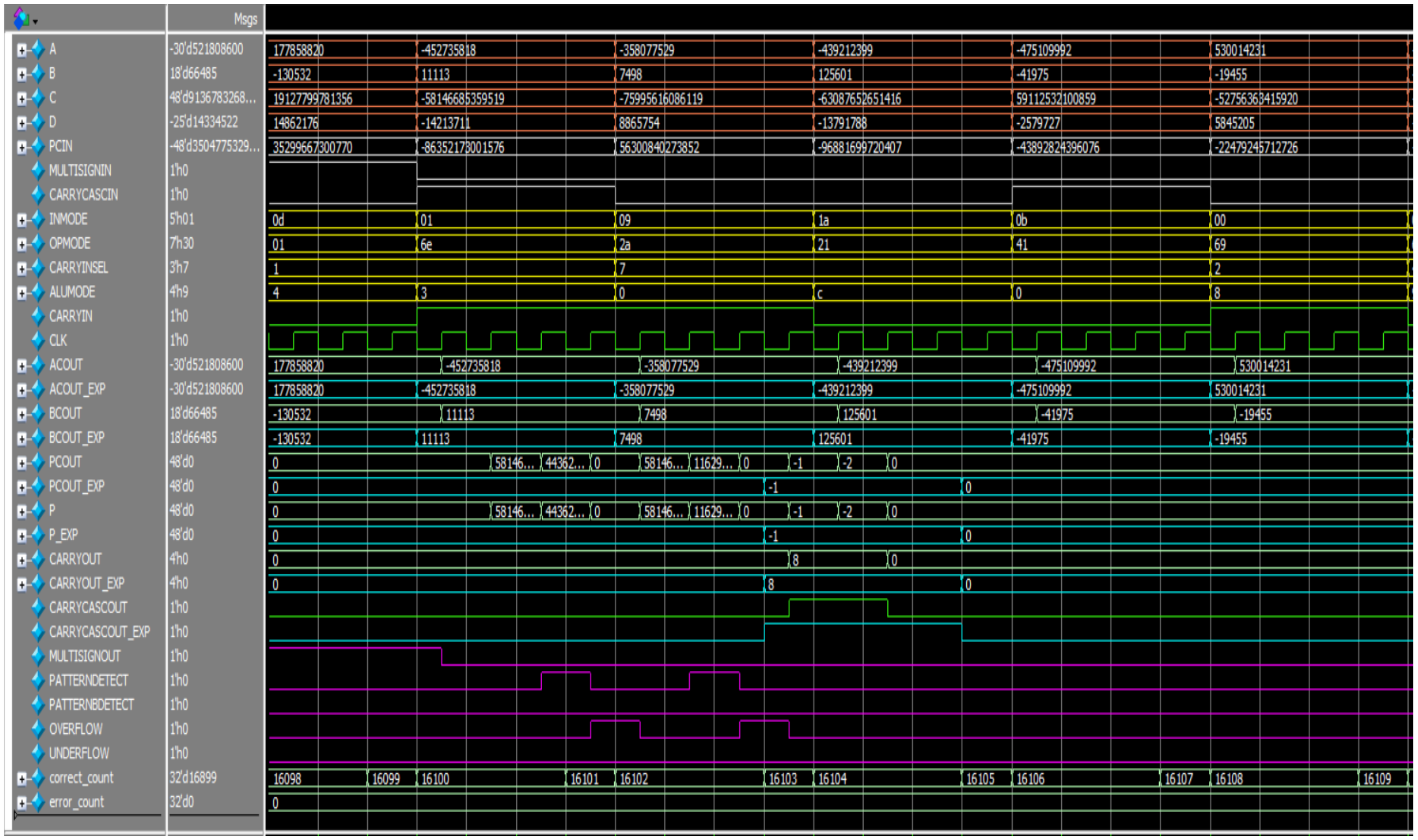
- **I Excluded the Wires that Takes Value from Parameter’s to Achieve Toggle Coverage 100%**
- **Also, I exclude Clock Enables because I forced them to 1**

Wave Forms Showing true operations

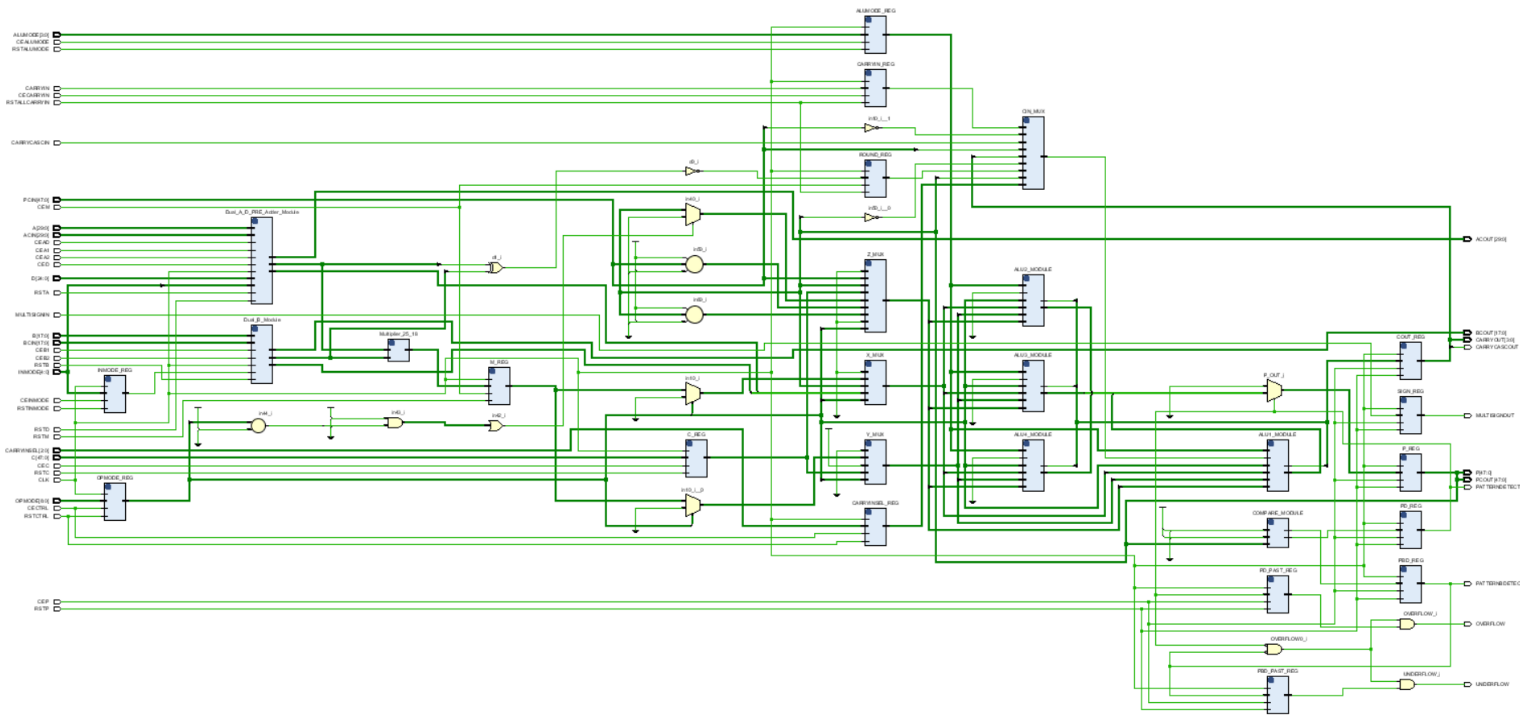

Msgs
A 30'd446400081 223977880 293323201 313579725 504086038 -110311783 312502587
B 18'd93595 114110 -23700 67886 118375 -127914 95094
C -48'd3392458456... -10512284346456 121157008549814 98291444076767 -117023204170606 -47336273295154 -118030782929002
D -25'd10371121 16449114 3621285 7389070 539776 11561925 15912368
PCIN 48'd1130243805... 64935565773603 84042780080843 116372013479305 125535436849035 63282010776583 58095456812563
MULTISIGNIN 1'h1
CARRYCASCIN 1'h0
INMODE 5'h03 17 12 02 14 08 1f
OPMODE 7'h62 14 01 3a 6d 39 5e
CARRYINSEL 3'h0 1 2 3 5 3 2
ALUMODE 4'h9 8 3 c 5 0 9
CARRYIN 1'h0
CLK 1'h0
ACOUT -30'd245120699 223977880 293323201 313579725 504086038 -110311783 312502587
ACOUT_EXP 30'd446400081 223977880 293323201 313579725 504086038 -110311783 312502587
BCOUT -18'd41606 114110 -23700 67886 118375 -127914 95094
BCOUT_EXP 18'd93595 114110 -23700 67886 118375 -127914 95094
PCOUT 48'd0 0 -1 0 98291444076767 0 -47336273295155 0
PCOUT_EXP 48'd0 0 -1 98291444076767 0 -47336273295155 0
P 48'd0 0 -1 0 98291444076767 0 -47336273295155 0
P_EXP 48'd0 0 -1 98291444076767 0 -47336273295155 0
CARRYOUT 4'h0 0 8 0 8 0
CARRYOUT_EXP 4'h0 0 8 0 8 0
CARRYCASCOUT 1'h0
CARRYCASCOUT_EXP 1'h0
MULTISIGNOUT 1'h0
PATTERNDETECT 1'h0
PATTERNBDETECT 1'h0
OVERFLOW 1'h0
UNDERFLOW 1'h0
correct_count 32'd17146 17122 17123 17124 17125 17126 17127 17128 17129 17130 17131 17132 17133
error_count 32'd0 0


Msgs
A 30'd446400081 476424957 95556256 -509007485 -278660602 -526613853 -336636817
B 18'd93595 -... -102449 43051 22779 -83106 101194 11059
C -48'd3392458456... 13284351330589 43660319445894 -93756533026314 61530314170731 -4919501527783 96471080665151
D -25'd10371121 -12679474 -9841799 3203856 -11580762 -2369859 -15703671
PCIN 48'd1130243805... -... 54029442020150 -79785498898332 -45475585473215 59705835824070 16073061789847 -134245783564955
MULTISIGNIN 1'h1
CARRYCASCIN 1'h0
INMODE 5'h03 06 15 01 09 1b 01 0d
OPMODE 7'h62 43 09 38 4c 08 61 41
CARRYINSEL 3'h0 3 2 4 2 3 0 5
ALUMODE 4'h9 0 4 c e d 6 2
CARRYIN 1'h0
CLK 1'h0
ACOUT -30'd245120699 51239... 476424957 95556256 -509007485 -278660602 -526613853 -336636817
ACOUT_EXP 30'd446400081 476424957 95556256 -509007485 -278660602 -526613853 -336636817
BCOUT -18'd41606 -78043 -102449 43051 22779 -83106 101194 11059
BCOUT_EXP 18'd93595 -... -102449 43051 22779 -83106 101194 11059
PCOUT 48'd0 13432... 13432... -1 43660319445894 0 -1 0 -1 0 -1
PCOUT_EXP 48'd0 134320992735014 -1 43660319445894 0 -1 0
P 48'd0 13432... 13432... -1 43660319445894 0 -1 0 -1 0 -1
P_EXP 48'd0 134320992735014 -1 43660319445894 0 -1 0
CARRYOUT 4'h0 0 8 0 8 0 8 0 8
CARRYOUT_EXP 4'h0 0 8 0 8 0
CARRYCASCOUT 1'h0
CARRYCASCOUT_EXP 1'h0
MULTISIGNOUT 1'h0
PATTERNDETECT 1'h0
PATTERNBDETECT 1'h0
OVERFLOW 1'h0
UNDERFLOW 1'h0
correct_count 32'd17146 17028 17029 17030 17031 17032 17033 17034 17035 17036 17037 17038
error_count 32'd0 0

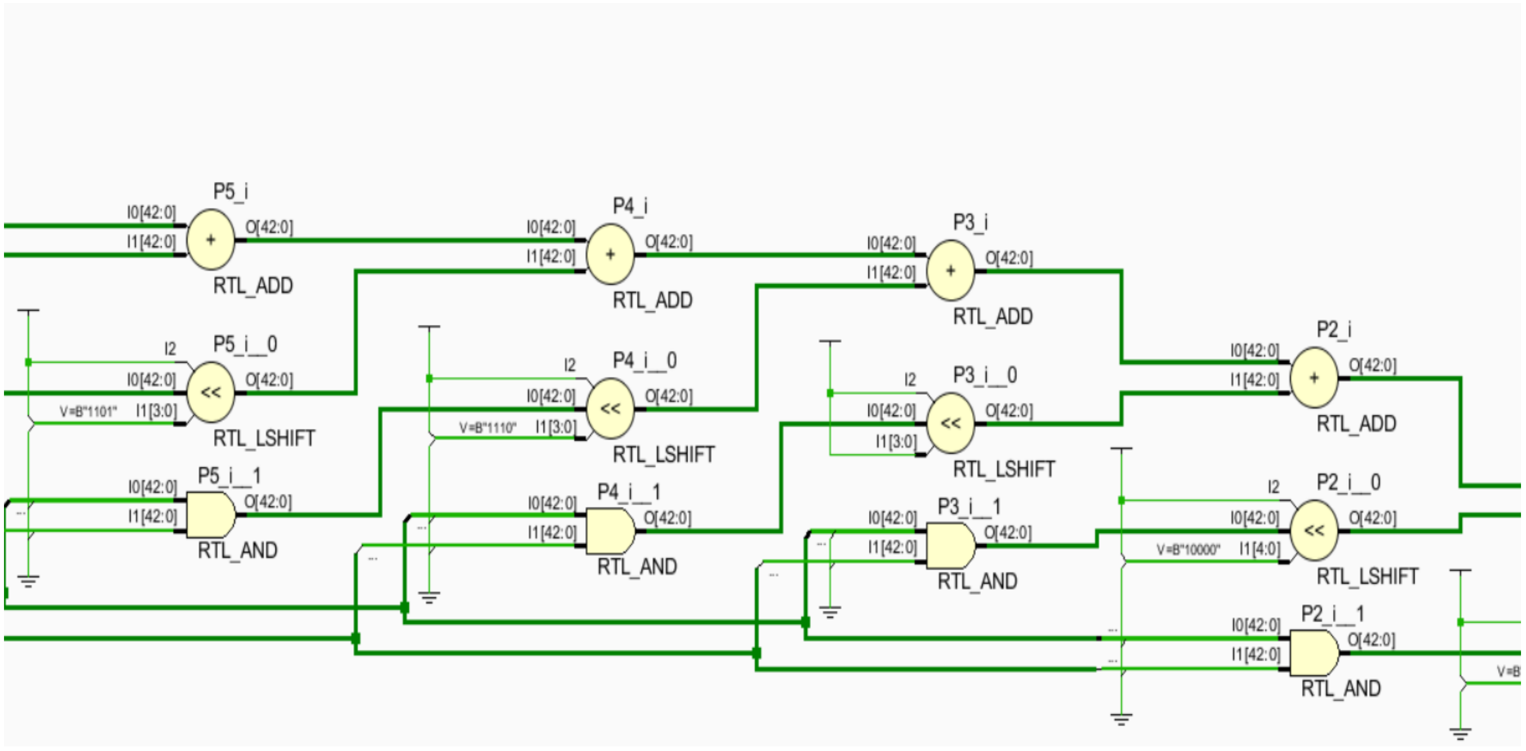

Msgs
A 30'd446400081
B 18'd93595
C -48'd3392458456...
D -25'd10371121
PCIN 48'd1130243805...
MULTISIGNIN 1'h1
CARRYCASCIN 1'h0
INMODE 5'h03
OPMODE 7'h62
CARRYINSEL 3'h0
ALUMODE 4'h9
CARRYIN 1'h0
CLK 1'h0
ACOUT -30'd245120699
ACOUT_EXP 30'd446400081
BCOUT -18'd41606
BCOUT_EXP 18'd93595
PCOUT 48'd0
PCOUT_EXP 48'd0
P 48'd0
P_EXP 48'd0
CARRYOUT 4'h0
CARRYOUT_EXP 4'h0
CARRYCASCOUT 1'h0
CARRYCASCOUT_EXP 1'h0
MULTSIGNOUT 1'h0
PATTERNDETECT 1'h0
PATTERNBDETECT 1'h0
OVERFLOW 1'h0
UNDERFLOW 1'h0
correct_count 32'd17146
error_count 32'd0


Msgs
A -30'd521808600
B 18'd66485
C 48'd9136783268...
D -25'd14334522
PCIN -48'd3504775329...
MULTISIGNIN 1'h0
CARRYCASCIN 1'h0
INMODE 5'h01
OPMODE 7'h30
CARRYINSEL 3'h7
ALUMODE 4'h9
CARRYIN 1'h0
CLK 1'h0
ACOUT -30'd521808600
ACOUT_EXP -30'd521808600
BCOUT 18'd66485
BCOUT_EXP 18'd66485
PCOUT 48'd0
PCOUT_EXP 48'd0
P 48'd0
P_EXP 48'd0
CARRYOUT 4'h0
CARRYOUT_EXP 4'h0
CARRYCASCOUT 1'h0
CARRYCASCOUT_EXP 1'h0
MULTISIGNOUT 1'h0
PATTERNDETECT 1'h0
PATTERNBDETECT 1'h0
OVERFLOW 1'h0
UNDERFLOW 1'h0
correct_count 32'd16899
error_count 32'd0

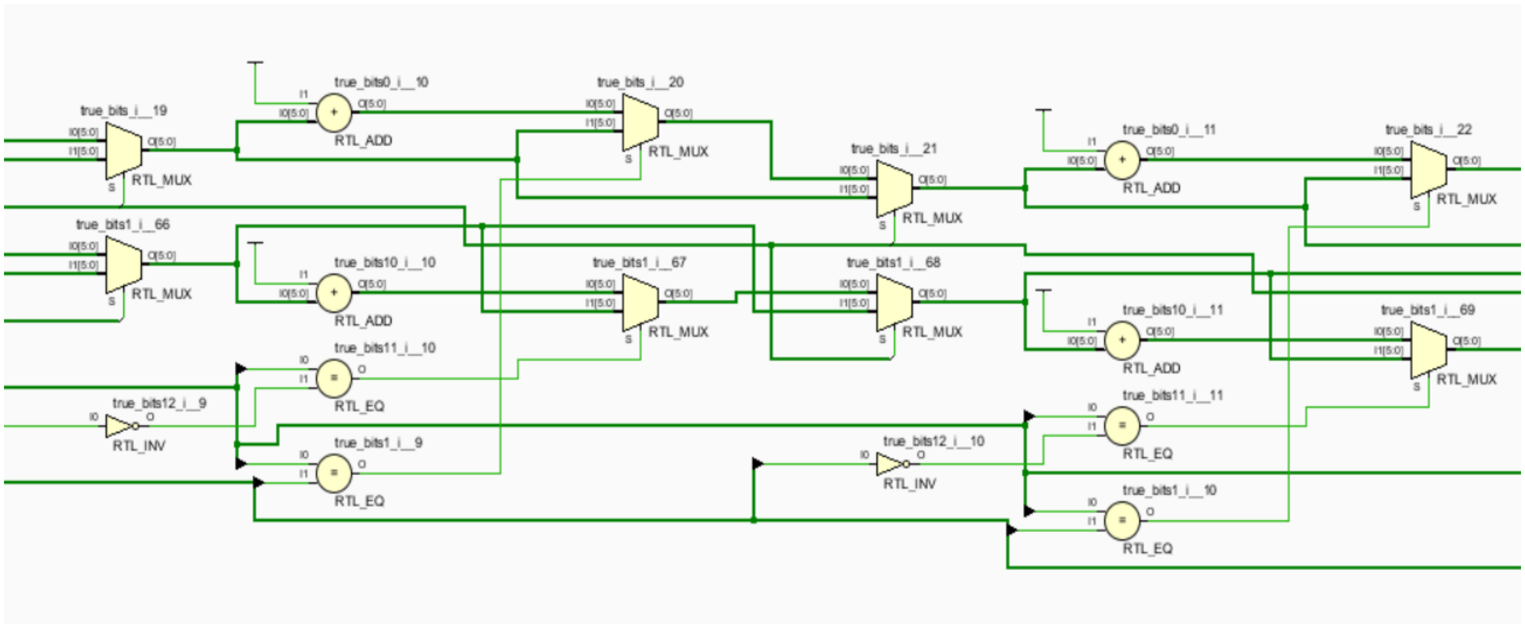
RTL schematic



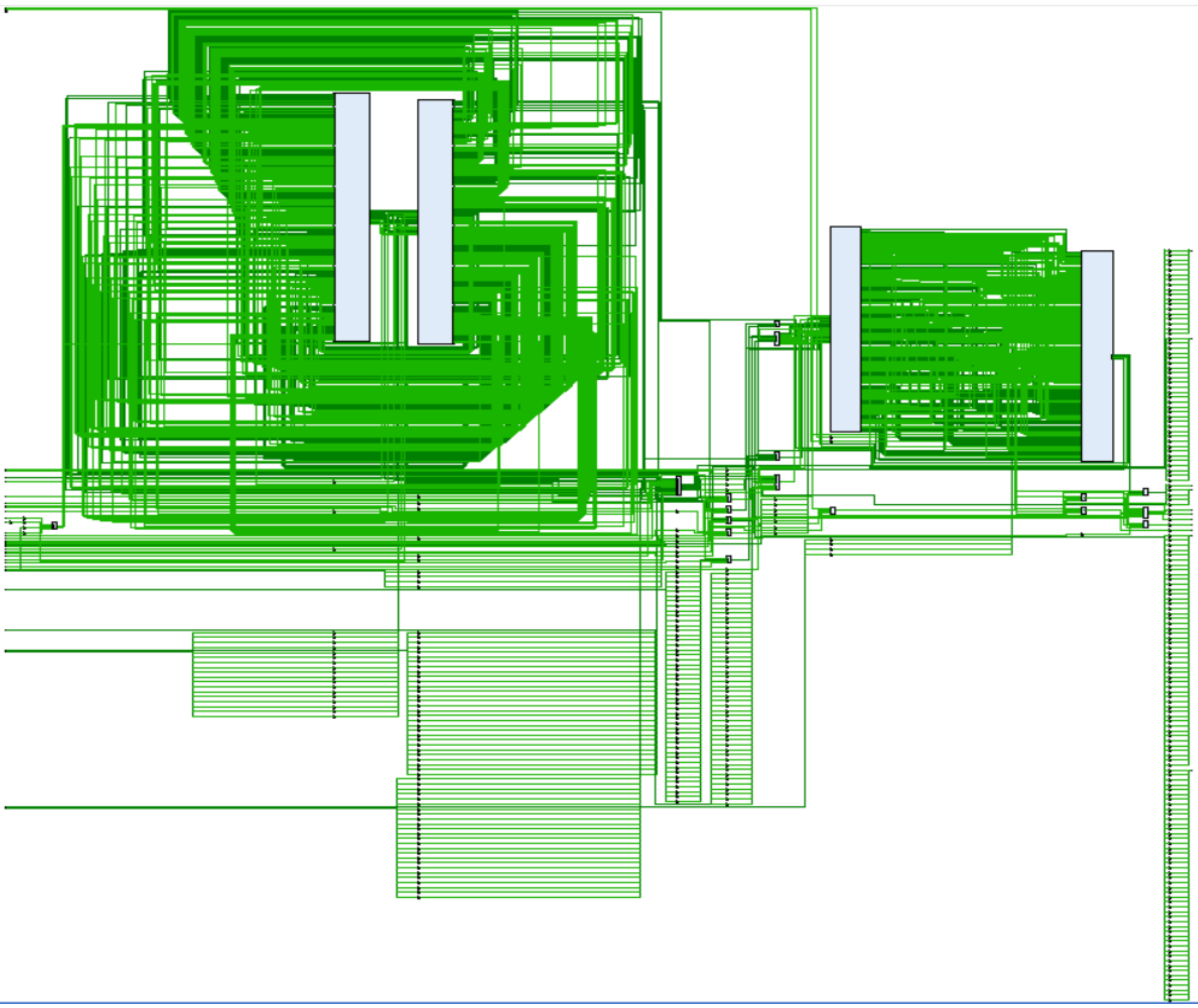
Partial Multiply Module Stages

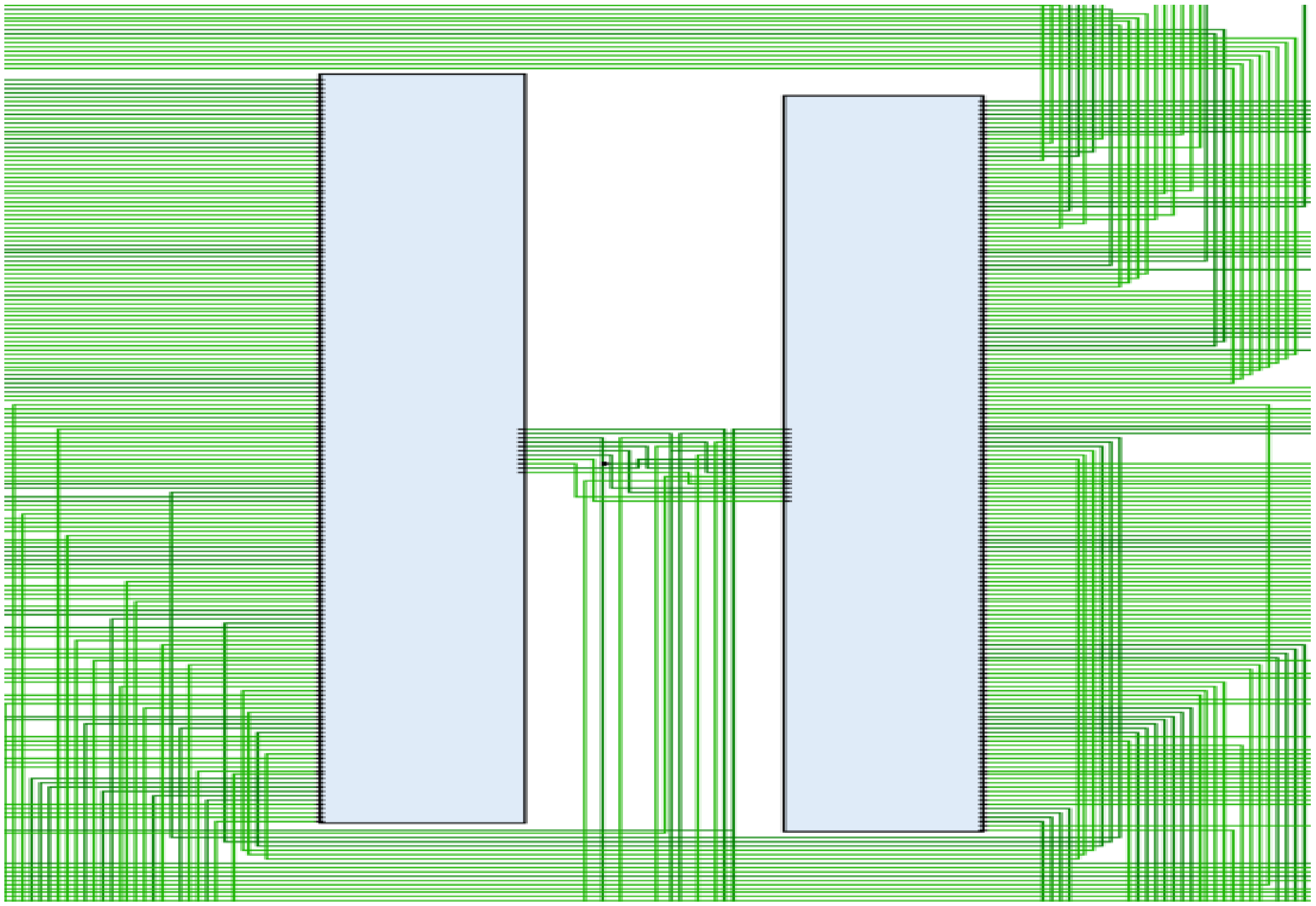


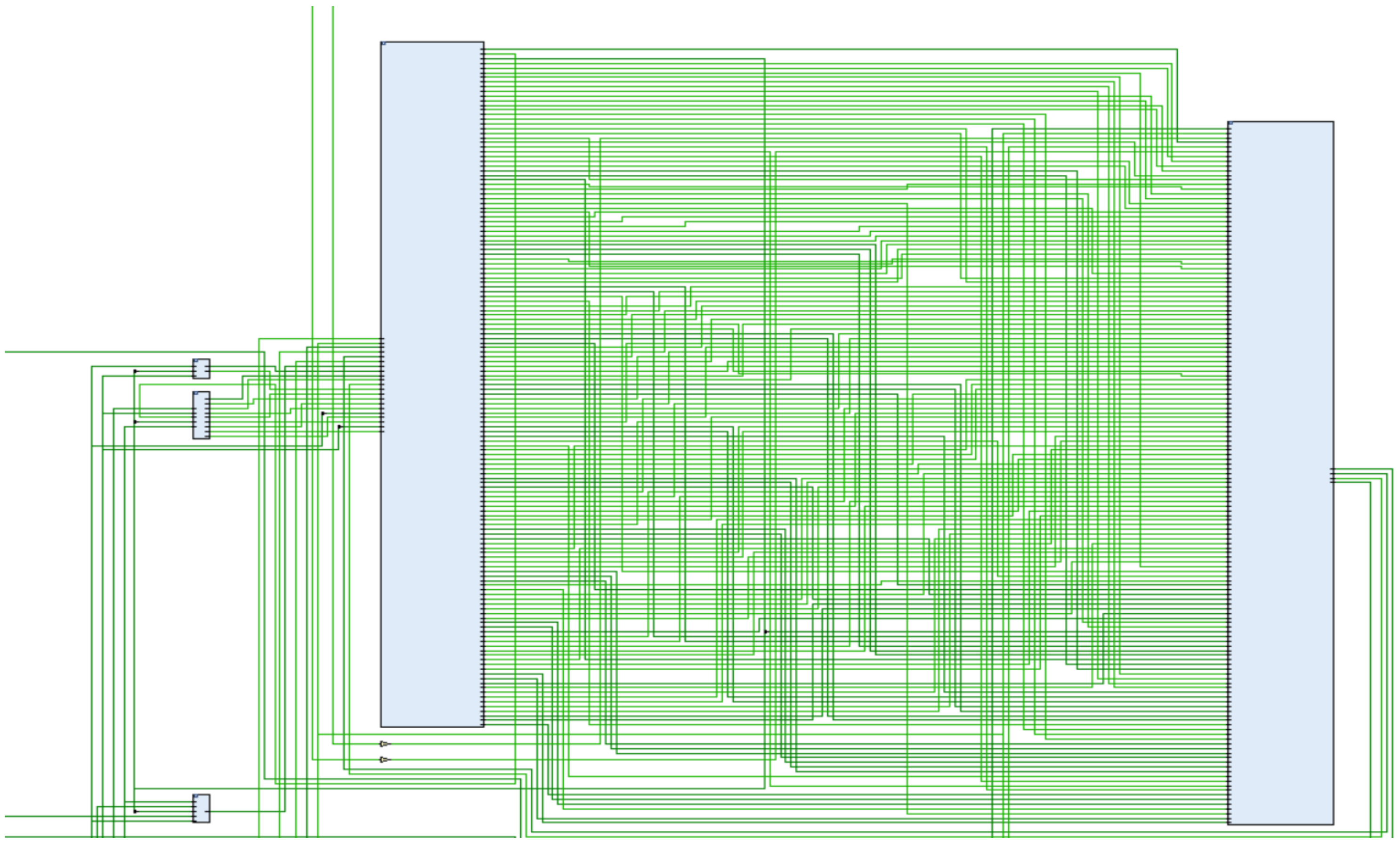
Masked Compare Stages



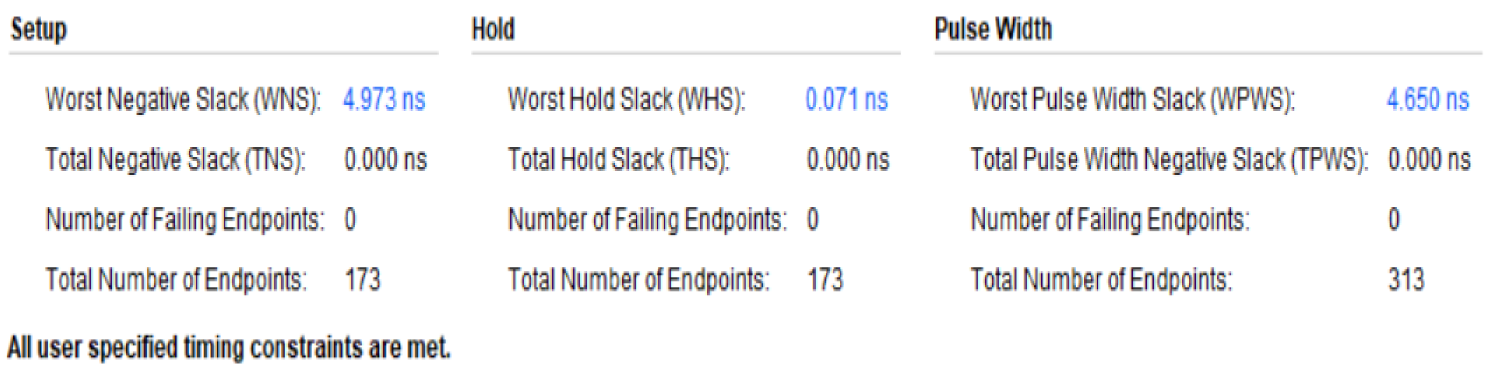
Synthesis Schematic







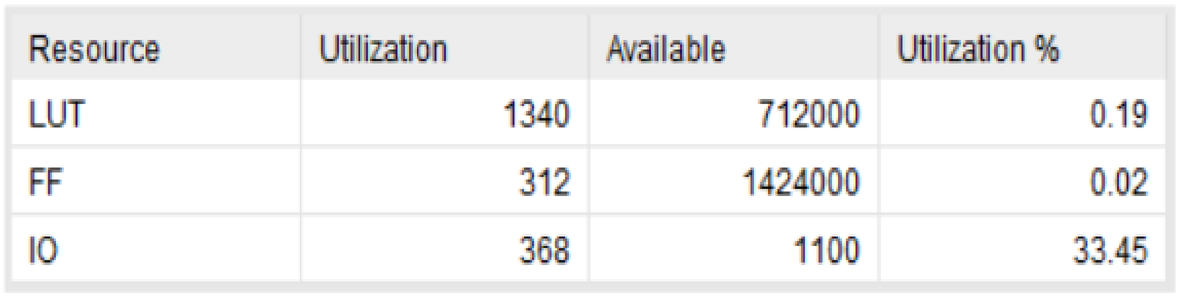
Time Report



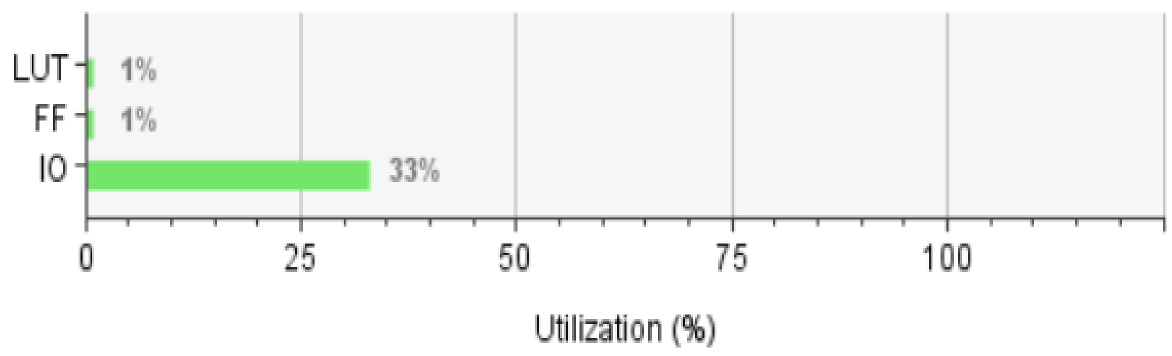
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.973 ns	Worst Hold Slack (WHS):	0.071 ns	Worst Pulse Width Slack (WPWS):	4.650 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	173	Total Number of Endpoints:	173	Total Number of Endpoints:	313

All user specified timing constraints are met.

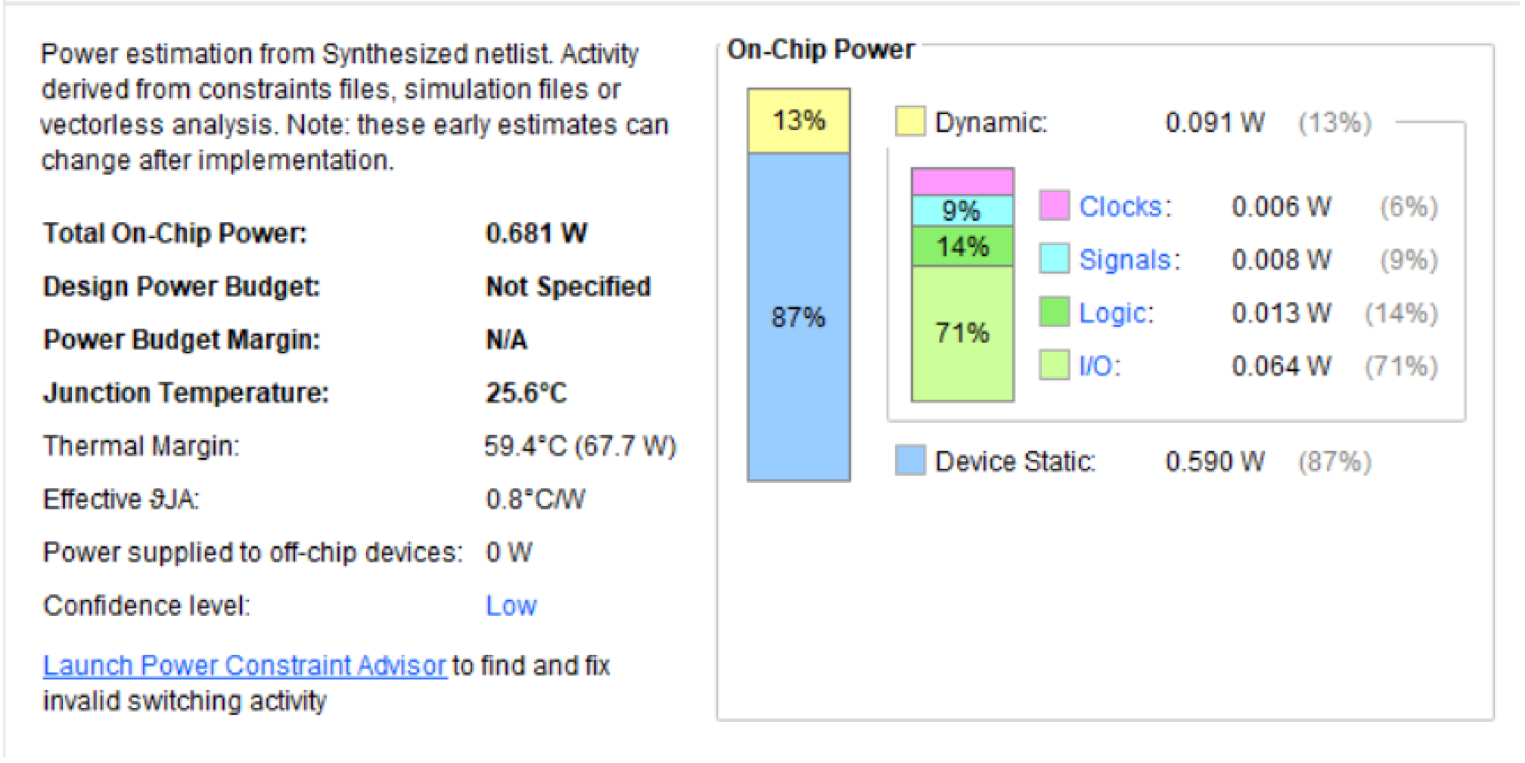
Utilization Report



Resource	Utilization	Available	Utilization %
LUT	1340	712000	0.19
FF	312	1424000	0.02
IO	368	1100	33.45



Power Report



Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

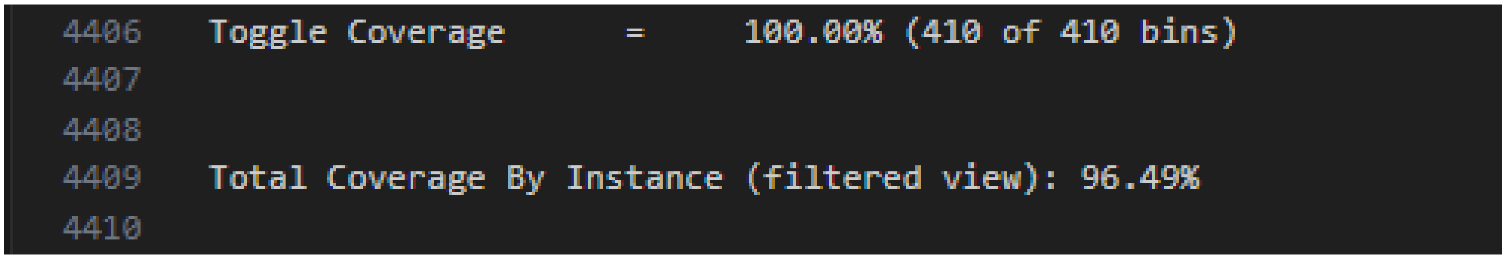
Total On-Chip Power:	**0.681 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.6°C**
Thermal Margin:	59.4°C (67.7 W)
Effective ϑJA:	0.8°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

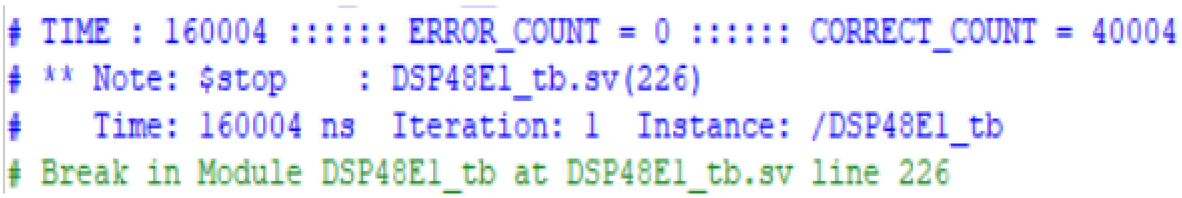
Dynamic:	0.091 W	(13%)
Clocks:	0.006 W	(6%)
Signals:	0.008 W	(9%)
Logic:	0.013 W	(14%)
I/O:	0.064 W	(71%)
Device Static:	0.590 W	(87%)

Total Coverage Percentage



```
Toggle Coverage      =     100.00% (410 of 410 bins)


Total Coverage By Instance (filtered view): 96.49%

```

Error Count and True Count



```
# TIME : 160004 :::::: ERROR_COUNT = 0 :::::: CORRECT_COUNT = 40004
# ** Note: $stop    : DSP48E1_tb.sv(226)
#    Time: 160004 ns  Iteration: 1  Instance: /DSP48E1_tb
# Break in Module DSP48E1_tb at DSP48E1_tb.sv line 226
```