2/28/2025

# Spartan-6 DSP48A1

Salah-Eldin Hassen Hassen.

# Table of Contents

# RTL Code:

Projects - DFF.v

```
1  module DFF
2      #(
3           parameter WIDTH = 1,
4           parameter RSTTYPE = "ASYNC",
5           parameter REGEN = 1
6       )
7       (
8           input wire clk,
9           input wire rst,
10          input wire en,
11          input wire [WIDTH-1:0] d,
12          output reg [WIDTH-1:0] q
13      );
14
15      generate
16          if (REGEN == 1) begin
17              if (RSTTYPE == "ASYNC") begin
18                  always @(posedge clk or posedge rst) begin
19                      if (rst) begin
20                          q <= 0;
21                      end else if (en) begin
22                          q <= d;
23                      end
24                  end
25              end else begin
26                  always @(posedge clk) begin
27                      if (rst) begin
28                          q <= 0;
29                      end else if (en) begin
30                          q <= d;
31                      end
32                  end
33              end
34          end else begin
35              always @(*) begin
36                  q = d;
37              end
38          end
39      endgenerate
40
41 endmodule
42
```

Projects - DSP48A1.v

```v
// % The  Spartan-6  family  offers  a  high  ratio  of  DSP48A1  slices  to  logic,
// % making  it  ideal  for  math-intensive applications. Design DSP48A1 slice of the spartan6 FPGAs.

module DSP48A1 #(
        //// ! Parameter (Attributes):
        parameter WIDTH_1 = 8,  // * Width for OPMODE
        parameter WIDTH_2 = 18, // * Width for A, B, D, BCIN, BCOUT
        parameter WIDTH_3 = 36, // * Width for M
        parameter WIDTH_4 = 48, // * Width for C, PCIN, PCOUT, P

        /**
         * * The A0REG, A1REG, B0REG, and B1REG attributes can take values of 0 or 1.
         * * These values define the number of pipeline registers in the A and B input paths.
         * # A0 and B0 are the first stages of the pipelines.
         * # A1 and B1 are the second stages of the pipelines.
         * ? A0REG defaults to 0 (no register). A1REG defaults to 1 (register).
         * ? B0REG defaults to 0 (no register) B1REG defaults to 1 (register).
         */
        parameter A0REG = 0,
        parameter A1REG = 1,
        parameter B0REG = 0,
        parameter B1REG = 1,

        /**
         * * These attributes can take a value of 0 or 1.
         * * The number defines the number of pipeline stages.
         * ? Default: 1 (registered)
         */
        parameter CREG = 1,
        parameter DREG = 1,
        parameter MREG = 1,
        parameter PREG = 1,
        parameter CARRYINREG = 1,
        parameter CARRYOUTREG = 1,
        parameter OPMODEREG = 1,

        /**
         * * This attribute can be set to the string CARRYIN or OPMODE5.
         * * The CARRYINSEL attribute is used in the carry cascade input,
         * * either the CARRYIN input will be considered or the value of opcode[5].
         * ? Default: OPMODE5. Tie the output of the mux to 0 if none of these string values exist.
         */
        parameter CARRYINSEL = "OPMODE5",

        /**
         * * The B_INPUT attribute defines whether:
         * * the input to the B port is routed from the B input (attribute = DIRECT)
         * * or the cascaded input (BCIN) from the previous DSP48A1 slice (attribute = CASCADE).
         * ? Default: DIRECT. Tie the output of the mux to 0 if none of these string values exist.
         */
        parameter B_INPUT = "DIRECT",

        /**
         * * This attribute can be set to ASYNC or SYNC.
         * * The RSTTYPE attribute selects whether all resets for the DSP48A1 slice should have
         * * a synchronous or asynchronous reset capability.
         * ? Default: SYNC.
         */
        parameter RSTTYPE = "SYNC"
    )(
        //// ! Data Ports:
        /**
         * * 18-bit data input to multiplier, and optionally to post-
         * * adder/subtracter depending on the value of OPMODE[1:0].
         */
        input [WIDTH_2-1:0] A,

        /**
         * * 18-bit data input to pre-adder/subtracter, to multiplier depending on
```

```
70          * * OPMODE[4], or to post-adder/subtracter depending on OPMODE[1:0].
71          */
72         input [WIDTH_2-1:0] B,
73
74         /**
75          * * 48-bit data input to post-adder/subtracter.
76          */
77         input [WIDTH_4-1:0] C,
78
79         /**
80          * * 18-bit data input to pre-adder/subtracter. D[11:0] are concatenated
81          * * with A and B and optionally sent to post-adder/subtracter depending on the value of OPMODE[1:0].
82          */
83         input [WIDTH_2-1:0] D,
84
85         /**
86          * * carry input to the post-adder/subtracter.
87          */
88         input CARRYIN,
89
90         /**
91          * * 36-bit buffered multiplier data output, routable to the FPGA logic.
92          * * It is either the output of the M register (MREG = 1) or the direct output of the multiplier (MREG = 0).
93          */
94         output [WIDTH_3-1:0] M,
95
96         /**
97          * * Primary data output from the post-adder/subtracter.
98          * * It is either the output of the P register (PREG = 1) or the direct output of the post-adder/subtracter (PREG = 0).
99          */
100        output [WIDTH_4-1:0] P,
101
102        /**
103         * * Cascade carry out signal from post-adder/subtracter.
104         * * It can be registered in (CARRYOUTREG = 1) or unregistered (CARRYOUTREG = 0).
105         * * This output is to be connected only to CARRYIN of adjacent DSP48A1 if multiple DSP blocks are used.
106         */
107        output CARRYOUT,
108
109        /**
110         * * Carry out signal from post-adder/subtracter for use in the FPGA logic.
111         * * It is a copy of the CARRYOUT signal that can be routed to the user logic.
112         */
113        output CARRYOUTF,
114
115        //// ! Control Input Ports:
116        input CLK,                  // # DSP clock
117        input [WIDTH_1-1:0] OPMODE, // # Control input to select the arithmetic operations of the DSP48A1 slice.
118
119        //// ! OPMODE Pin Descriptions:
120        /**
121         * # OPMODE[1:0]:
122         *      ? Specifies the source of the X input to the post-adder/subtracter
123         *      * 0 - Specifies to place all zeros (disable the post-adder/subtracter
124         *      *      and propagate the Z result to P)
125         *      * 1 - Use the multiplier product
126         *      * 2 - Use the P output signal (accumulator)
127         *      * 3 - Use the concatenated D:A:B input signals
128         * # OPMODE[3:2]:
129         *      ? Specifies the source of the Z input to the post-adder/subtracter
130         *      * 0 - Specifies to place all zeros (disable the post-adder/subtracter
131         *      *     and propagate the multiplier product or other X result to P)
132         *      * 1 - Use the PCIN
133         *      * 2 - Use the P output signal (accumulator)
134         *      * 3 - Use the C port
135         * # OPMODE[4]:
136         *      ? Specifies the use of the pre-adder/subtracter
137         *      * 0 - Bypass the pre-adder supplying the data on port B directly to the multiplier
138         *      * 1 - Selects to use the pre-adder adding or subtracting the values on the B and D ports
139         *      *     prior to the multiplier.
140         * # OPMODE[5]:
141         *      ? Forces a value on the carry input of the carry-in register (CYI) or direct to the CIN
```

```
         *      ? to the post-adder. Only applicable when CARRYINSEL = OPMODE5
         * # OPMODE[6]:
         *      ? Specifies whether the pre-adder/subtracter is an adder or subtracter
         *      * 0 - Specifies pre-adder/subtracter to perform an addition operation
         *      * 1 - Specifies pre-adder/subtracter to perform a subtraction operation (D-B)
         * # OPMODE[7]:
         *      ? Specifies whether the post-adder/subtracter is an adder or subtracter
         *      * 0 - Specifies post-adder/subtracter to perform an addition operation
         *      * 1 - Specifies post-adder/subtracter to perform a subtraction operation (Z-(X+CIN))
         */

        //// ! Clock Enable Input Ports:
        input CEA,          // * Clock enable for the A port registers: (A0REG & A1REG).
        input CEB,          // * Clock enable for the B port registers: (B0REG & B1REG).
        input CEC,          // * Clock enable for the C port registers (CREG).
        input CECARRYIN,    // * Clock enable for the carry-in register (CYI) and the carry-out register (CYO).
        input CED,          // * Clock enable for the D port register (DREG).
        input CEM,          // * Clock enable for the multiplier register (MREG).
        input CEOPMODE,     // * Clock enable for the opmode register (OPMODEREG).
        input CEP,          // * Clock enable for the P output port registers (PREG = 1).

        //// ! Reset Input Ports:
        // # All the resets are active high. They are either sync or async depending on the parameter RSTTYPE.
        input RSTA,         // * Reset  for the A registers: (A0REG & A1REG).
        input RSTB,         // * Reset for the B registers: (B0REG & B1REG).
        input RSTC,         // * Reset for the C registers (CREG).
        input RSTCARRYIN,   // * Reset for the carry-in register (CYI) and the carry-out register (CYO).
        input RSTD,         // * Reset for the D register (DREG).
        input RSTM,         // * Reset for the multiplier register (MREG).
        input RSTOPMODE,    // * Reset for the opmode register (OPMODEREG).
        input RSTP,         // * Reset for the P output registers (PREG = 1).

        //// ! Cascade Ports:
        input [WIDTH_2-1:0] BCIN,   // * Cascade input for Port B.
        output [WIDTH_2-1:0] BCOUT, // * Cascade output for Port B.
        input [WIDTH_4-1:0] PCIN,   // * Cascade input for Port P.
        output [WIDTH_4-1:0] PCOUT  // * Cascade output for Port P.
    );

    //// ! Internal Signals:
    // * 01: Input Stage
    wire [WIDTH_2-1:0] B_0;
    wire [WIDTH_1-1:0] OPMODE_O;

    // * 02: Second Stage
    wire [WIDTH_2-1:0] D_1;
    wire [WIDTH_2-1:0] B_0_O;
    wire [WIDTH_2-1:0] A_1;
    wire [WIDTH_4-1:0] C_1;

    // * 03: Pre-Adder/Subtracter
    wire [WIDTH_2-1:0] D_PAS_B; // ? D + B or D - B (OPMODE[6])

    // * 04: Fourth Stage
    wire [WIDTH_2-1:0] B_1;

    // * 05: Fifth Stage
    wire [WIDTH_4-1:0] D_A_B; // ? {D_1[11:0], A_1_O, B_1_O}
    wire [WIDTH_2-1:0] B_1_O;
    wire [WIDTH_2-1:0] A_1_O;

    // * 06: Multiplier
    wire [WIDTH_3-1:0] B1_Mul_A1;
    wire CYI;

    // * 07: Seventh Stage
    wire [WIDTH_3-1:0] M_B;
    wire CYI_O;

    // * 08: Eighth Stage
    wire [WIDTH_4-1:0] X;
    wire [WIDTH_4-1:0] Z;
```

```
214
215     // * 09: Post-Adder/Subtracter
216     wire [WIDTH_4-1:0] X_Z_CIN_OP; // ? X + Z + CIN or Z - (X + CIN) (OPMODE[7])
217     wire CYO;
218
219     // * 10: Output Stage
220     // ? No Signals Required
221
222     //// ! Implementation
223     // * 01: Input Stage
224     assign B_0 = (B_INPUT == "DIRECT") ? B : (B_INPUT == "CASCADE") ? BCIN : 0;
225
226     DFF #(.WIDTH(WIDTH_1), .RSTTYPE(RSTTYPE), .REGEN(OPMODEREG)) DFF_OPMODE
    (.d(OPMODE), .clk(CLK), .rst(RSTOPMODE), .en(CEOPMODE), .q(OPMODE_O));
227
228     // * 02: Second Stage
229     DFF #(.WIDTH(WIDTH_2), .RSTTYPE(RSTTYPE), .REGEN(DREG)) DFF_D (.d(D), .clk(CLK), .rst(RSTD), .en(CED), .q(D_1));
230     DFF #(.WIDTH(WIDTH_2), .RSTTYPE(RSTTYPE), .REGEN(B0REG)) DFF_B_0 (.d(B_0), .clk(CLK), .rst(RSTB), .en(CEB), .q(B_0_O));
231     DFF #(.WIDTH(WIDTH_2), .RSTTYPE(RSTTYPE), .REGEN(A0REG)) DFF_A_0 (.d(A), .clk(CLK), .rst(RSTA), .en(CEA), .q(A_1));
232     DFF #(.WIDTH(WIDTH_4), .RSTTYPE(RSTTYPE), .REGEN(CREG)) DFF_C (.d(C), .clk(CLK), .rst(RSTC), .en(CEC), .q(C_1));
233
234     // * 03: Pre-Adder/Subtracter
235     assign D_PAS_B = (OPMODE_O[6]) ? D_1 - B_0_O : D_1 + B_0_O;
236
237     // * 04: Fourth Stage
238     assign B_1 = (OPMODE_O[4]) ? D_PAS_B : B_0_O;
239
240     // * 05: Fifth Stage
241     DFF #(.WIDTH(WIDTH_2), .RSTTYPE(RSTTYPE), .REGEN(B1REG)) DFF_B_1 (.d(B_1), .clk(CLK), .rst(RSTB), .en(CEB), .q(B_1_O));
242     DFF #(.WIDTH(WIDTH_2), .RSTTYPE(RSTTYPE), .REGEN(A1REG)) DFF_A_1 (.d(A_1), .clk(CLK), .rst(RSTA), .en(CEA), .q(A_1_O));
243
244     assign D_A_B = {D_1[11:0], A_1_O, B_1_O};
245     assign BCOUT = B_1_O;
246
247     // * 06: Multiplier
248     assign B1_Mul_A1 = B_1_O * A_1_O;
249
250     assign CYI = (CARRYINSEL == "OPMODE5") ? OPMODE[5] : (CARRYINSEL == "CARRYIN") ? CARRYIN: 1'b0;
251
252     // * 07: Seventh Stage
253     DFF #(.WIDTH(WIDTH_3), .RSTTYPE(RSTTYPE), .REGEN(MREG)) DFF_M_O (.d(B1_Mul_A1), .clk(CLK), .rst(RSTM), .en(CEM), .q(M_B));
254     DFF #(.WIDTH(1), .RSTTYPE(RSTTYPE), .REGEN(CARRYINREG)) DFF_CYI_O (.d(CYI), .clk(CLK), .rst(RSTCARRYIN), .en(CECARRYIN), .q(CYI_O));
255
256     (* keep *) assign M = M_B;
257
258     // * 08: Eighth Stage
259     assign X = (OPMODE_O[1:0] == 0) ? {WIDTH_4{1'b0}} :
260            (OPMODE_O[1:0] == 1) ? {{(WIDTH_4-WIDTH_3){1'b0}}, M_B} :
261            (OPMODE_O[1:0] == 2) ? P :
262            D_A_B;
263
264     assign Z = (OPMODE_O[3:2] == 0) ? {WIDTH_4{1'b0}} :
265            (OPMODE_O[3:2] == 1) ? PCIN :
266            (OPMODE_O[3:2] == 2) ? P :
267            C_1;
268
269     // * 09: Post-Adder/Subtracter
270     assign {CYO, X_Z_CIN_OP} = (OPMODE_O[1:0] == 0) ? {1'b0, Z} :
271            (OPMODE_O[3:2] == 0) ? {1'b0, X} :
272            (OPMODE_O[7]) ? Z - X - CYI_O : Z + X + CYI_O;
273
274     // * 10: Output Stage
275     DFF #(.WIDTH(1), .RSTTYPE(RSTTYPE), .REGEN(CARRYOUTREG)) DFF_CYO (.d(CYO), .clk(CLK), .rst(RSTCARRYIN), .en(CECARRYIN), .q(CARRYOUT));
276     DFF #(.WIDTH(WIDTH_4), .RSTTYPE(RSTTYPE), .REGEN(PREG)) DFF_P (.d(X_Z_CIN_OP), .clk(CLK), .rst(RSTP), .en(CEP), .q(P));
277
278     assign CARRYOUTF = CARRYOUT;
279     assign PCOUT = P;
280
281 endmodule
282
283
```

# Test Bench Code:

Projects - DSP48A1_tb.sv

```
`timescale 1ns / 1ps

module DSP48A1_tb;

    // Parameters
    parameter WIDTH_1 = 8; parameter WIDTH_2 = 18; parameter WIDTH_3 = 36; parameter WIDTH_4 = 48;
    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";

    // Inputs
    reg [WIDTH_2-1:0] A, B, D, BCIN;
    reg [WIDTH_4-1:0] C, PCIN;
    reg [WIDTH_1-1:0] OPMODE;
    reg CARRYIN, CLK;
    reg CEA, CEB, CEC, CECARRYIN, CED, CEM, CEOPMODE, CEP;
    reg RSTA, RSTB, RSTC, RSTCARRYIN, RSTD, RSTM, RSTOPMODE, RSTP;

    // Outputs
    wire CARRYOUT, CARRYOUTF;
    wire [WIDTH_4-1:0] P, PCOUT;
    wire [WIDTH_2-1:0] BCOUT;
    wire [WIDTH_3-1:0] M;

    reg [WIDTH_2-1:0] golden_model_BCOUT;
    reg [WIDTH_3-1:0] golden_model_M;
    reg golden_model_CARRYOUT;
    reg [WIDTH_4-1:0] golden_model_P;

    // Instantiate the Unit Under Test (UUT)
    DSP48A1 #(
                .WIDTH_1(WIDTH_1), .WIDTH_2(WIDTH_2), .WIDTH_3(WIDTH_3), .WIDTH_4(WIDTH_4),
                .A0REG(A0REG), .A1REG(A1REG), .B0REG(B0REG), .B1REG(B1REG),
                .CREG(CREG), .DREG(DREG), .MREG(MREG), .PREG(PREG),
                .CARRYINREG(CARRYINREG), .CARRYOUTREG(CARRYOUTREG), .OPMODEREG(OPMODEREG),
                .CARRYINSEL(CARRYINSEL), .B_INPUT(B_INPUT), .RSTTYPE(RSTTYPE)
            ) uut (
                .A(A), .B(B), .C(C), .D(D), .OPMODE(OPMODE), .BCIN(BCIN), .PCIN(PCIN),
                .CARRYIN(CARRYIN), .CLK(CLK), .CEA(CEA), .CEB(CEB), .CEC(CEC),
                .CECARRYIN(CECARRYIN), .CED(CED), .CEM(CEM), .CEOPMODE(CEOPMODE),
                .CEP(CEP), .RSTA(RSTA), .RSTB(RSTB), .RSTC(RSTC), .RSTCARRYIN(RSTCARRYIN),
                .RSTD(RSTD), .RSTM(RSTM), .RSTOPMODE(RSTOPMODE), .RSTP(RSTP),
                .CARRYOUT(CARRYOUT), .P(P), .BCOUT(BCOUT), .PCOUT(PCOUT), .M(M),
                .CARRYOUTF(CARRYOUTF)
            );

    integer file;
    integer status;
    integer i;
    string filelist[100];
    integer filecount;
    integer j;

    // Clock generation
    initial begin
        CLK = 0;
        forever #5 CLK = ~CLK;
    end

    initial begin
        // Initialize Inputs
        {B, BCIN, OPMODE, D, A, C, PCIN, CARRYIN} = 0;
        {CEA, CEB, CEC, CECARRYIN, CED, CEM, CEOPMODE, CEP} = 8'b11111111;
        {RSTA, RSTB, RSTC, RSTCARRYIN, RSTD, RSTM, RSTOPMODE, RSTP} = 0;

```

```
        // Apply reset
        {RSTA, RSTB, RSTC, RSTCARRYIN, RSTD, RSTM, RSTOPMODE, RSTP} = 8'b11111111;
        @(negedge CLK);
        {RSTA, RSTB, RSTC, RSTCARRYIN, RSTD, RSTM, RSTOPMODE, RSTP} = 8'b00000000;

        i = 0;
        filecount = 0;

        // Open the directory and read file names
        file = $fopen("../Golden/filelist.txt", "r");
        if (file == 0) begin
            $display("Failed to open file list.");
            $finish;
        end

        while (!$feof(file)) begin
            status = $fscanf(file, "%s\n", filelist[filecount]);
            if (status == 1) begin
                filecount = filecount + 1;
            end
        end
        $fclose(file);

        // Loop through each file
        for (j = 0; j < filecount; j = j + 1) begin
            $display("Testing with file: %s", filelist[j]);
            #0;
            file = $fopen({"../Golden/", filelist[j]}, "r");
            if (file == 0) begin
                $display("Failed to open golden model file: %s", filelist[j]);
                $finish;
            end

            // Read and apply test vectors
            while (!$feof(file)) begin
                status = $fscanf(file, "%b %b %b %b %b %b %b %b -> %b %b %b %b\n",
                                 A, B, C, D, BCIN, OPMODE, PCIN, CARRYIN, golden_model_BCOUT, golden_model_M, golden_model_CARRYOUT,
golden_model_P);

                if (status != 12) begin
                    $display("Failed: ERROR fscanf failed or incomplete data read.");
                    $finish;
                end

                i = i + 1;
                if(i>4)
                    i = 1;

                // Compare expected and actual results
                if (golden_model_BCOUT !== BCOUT || golden_model_M !== M || golden_model_CARRYOUT !== CARRYOUT || golden_model_P !== P) begin
                    $display("%d\n", i);
                    $display("Failed at A=%h B=%h C=%h D=%h BCIN=%h OPMODE=%b PCIN=%h CARRYIN=%h: Expected BCOUT= %h, Got %h | Expected M=%h
, Got %h | Expected CARRYOUT=%h, Got %h | Expected P=%h, Got %h",
                             A, B, C, D, BCIN, OPMODE, PCIN, CARRYIN, golden_model_BCOUT, BCOUT, golden_model_M, M, golden_model_CARRYOUT
, CARRYOUT, golden_model_P, P);
                end

                repeat(1) @(negedge CLK);
            end

            $fclose(file);
        end

        $finish;
    end

endmodule
```

# Golden model (C/C++):

Projects - module.h

```c
#ifndef MODULE_H
#define MODULE_H

#include <stdint.h>
#include <stdbool.h>
#include <string.h>

#ifdef __cplusplus
extern "C"
{
#endif

#define B_INPUT 1    // 0 for CASCADE, 1 for DIRECT
#define CARRYINSEL 0 // 0 for OPMODE5, 1 for CARRYIN

// Define constants for bit masks
#define MASK_8 0x0FF
#define MASK_18 0x03FFF
#define MASK_36 0x00000FFFFFFFF
#define MASK_48 0x00FFFFFFFFFFF
#define MASK_49 0x01FFFFFFFFFFF

#define WIDTH_1 8
#define WIDTH_2 18
#define WIDTH_3 36
#define WIDTH_4 48
#define WIDTH_5 49

    // Define a structure to hold the state and inputs of the DSP48A1
    typedef struct
    {
        // Inputs
        uint32_t A : WIDTH_2;
        uint32_t B : WIDTH_2;
        uint64_t C : WIDTH_4;
        uint32_t D : WIDTH_2;
        uint32_t BCIN : WIDTH_2;
        uint64_t PCIN : WIDTH_4;
        bool CARRYIN;
        uint8_t OPMODE;

        // Outputs
        uint32_t BCOUT : WIDTH_2;
        uint64_t M : WIDTH_3;
        bool CARRYOUT;
        uint64_t P : WIDTH_4;

        // Internal signals
        uint32_t B_0_wire : WIDTH_2;

        uint32_t D_reg : WIDTH_2;
        uint32_t B_reg : WIDTH_2;
        uint64_t C_reg : WIDTH_4;
        uint32_t A_reg : WIDTH_2;
        uint8_t OPMODE_reg;
        bool CARRYIN_reg;

        uint64_t X : WIDTH_4;
        uint64_t Z : WIDTH_4;

        uint64_t post_adder : WIDTH_5;

        uint64_t P_b_wire : WIDTH_4;
        bool CARRYOUT_b_wire;
    } DSP48A1_State;

    // Function prototypes
    void DSP48A1_clock_cycle(DSP48A1_State *state, DSP48A1_State *prev);

#ifdef __cplusplus
}
#endif

#endif // MODULE_H
```

```c
#include "module.h"
#include <stdio.h>

// Simulate a clock cycle for the DSP48A1
void DSP48A1_clock_cycle(DSP48A1_State *state, DSP48A1_State *prev)
{
// Select B input based on B_INPUT parameter
#if B_INPUT == 1
    state->B_0_wire = state->B;
#elif B_INPUT == 0
    state->B_0 = state->BCIN;
#else
    state->B_0 = 0;
#endif

    state->D_reg = prev->D;
    state->C_reg = prev->C;
    state->OPMODE_reg = prev->OPMODE;

    state->B_reg = (prev->OPMODE_reg & (1 << 4)) ? ((prev->OPMODE_reg & (1 << 6)) ? prev->D_reg - prev->B_0_wire : prev->D_reg + prev->B_0_wire) : prev->B_0_wire;


    state->A_reg = prev->A;

    state->BCOUT = state->B_reg;

    // state->M = (uint64_t)prev->B_1 * (uint64_t)prev->A_1;
    // Ensure B_1 and A_1 are correctly sign-extended
    uint64_t b_val = prev->B_reg; // Assuming B_1 is already sign-extended
    uint64_t a_val = prev->A_reg; // Assuming A_1 is already sign-extended

    // Perform a signed multiplication
    uint64_t product = b_val * a_val;

    // Mask the result to fit into a 36-bit signed field
    state->M = product & MASK_36;

    // Select carry-in based on CARRYINSEL parameter
#if CARRYINSEL == 1
    state->CARRYIN_1 = prev->CARRYIN;
#elif CARRYINSEL == 0
    prev->CARRYIN_reg = (prev->OPMODE_reg & (1 << 5)) ? 1 : 0;
#else
    state->CARRYIN_1 = 0;
#endif

    // X input selection based on OPMODE[1:0]
    switch (prev->OPMODE_reg & 0x03)
    {
    case 0:
        prev->X = 0;
        break;
    case 1:
        prev->X = (uint64_t)prev->M;
        break;
    case 2:
        prev->X = prev->P;
        break;
    case 3:
        prev->X = (((uint64_t)prev->D_reg << 36) & 0x0FFF000000000) | (((uint64_t)prev->A_reg << 18) & 0x00FFFFC0000) | ((uint64_t)prev->B_reg & 0x03FFFF);
        break;
    }

    // Z input selection based on OPMODE[3:2]
    switch ((prev->OPMODE_reg >> 2) & 0x03)
    {
    case 0:
        prev->Z = 0;
        break;
    case 1:
        prev->Z = prev->PCIN;
        break;
    case 2:
        prev->Z = prev->P;
        break;
    case 3:
        prev->Z = prev->C_reg;
        break;
    }

    // Post-adder/subtracter operation
    if ((prev->OPMODE_reg & 0x03) == 0)
    {
        prev->P_b_wire = prev->Z;
        prev->CARRYOUT_b_wire = 0;
    }
    else if (((prev->OPMODE_reg >> 2) & 0x03) == 0)
    {
        prev->P_b_wire = prev->X;
        prev->CARRYOUT_b_wire = 0;
    }
    else
    {
        prev->post_adder = (prev->OPMODE_reg & (1 << 7)) ? (uint64_t)prev->Z - (uint64_t)prev->X - prev->CARRYIN_reg : (uint64_t)prev->X + (uint64_t)prev->Z + prev->CARRYIN_reg;

        // Update P register
        prev->P_b_wire = prev->post_adder & MASK_48;
        prev->CARRYOUT_b_wire = (prev->post_adder >> 48) & 1; // Extract carry out
    }

    state->P = prev->P_b_wire;
    state->CARRYOUT = prev->CARRYOUT_b_wire;
}
```

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <cstring>
#include <map>
#include <string>

#include "module.h"

DSP48A1_State state, prev;
int test_number = 1;

// Function to convert a number to a binary string
void to_binary_string(uint64_t num, int bits, char *str)
{
    for (int i = bits - 1; i >= 0; i--)
    {
        str[bits - 1 - i] = (num & (1ULL << i)) ? '1' : '0';
    }
    str[bits] = '\0';
}

// Function to generate a filename based on the signals to be randomized
std::string generate_filename(const std::map<std::string, bool> &randomize_signals)
{
    std::string filename = std::to_string(test_number++) + "_golden_model_";
    for (const auto &signal : randomize_signals)
    {
        if (!signal.second)
        {
            filename += signal.first + "_";
        }
    }
    filename += ".txt";
    return filename;
}

// Function to generate test vectors
void generate_test_vectors(const std::map<std::string, bool> &randomize_signals, int test_cases)
{
    std::string filename = generate_filename(randomize_signals);
    std::ofstream fp(filename);
    if (!fp.is_open())
    {
        std::cerr << "Error opening file: " << filename << std::endl;
        return;
    }

    srand(time(NULL));

    char bin_str_A[19], bin_str_B[19], bin_str_C[49], bin_str_D[19];
    char bin_str_BCIN[19], bin_str_PCIN[49], bin_str_OPMODE[9], bin_str_CARRYIN[2];
    char bin_str_BCOUT[19], bin_str_M[49], bin_str_CARRYOUT[2], bin_str_P[49];

    for (int i = 0; i < test_cases; i++)
    {
        if (randomize_signals.at("A"))
            state.A = rand() & MASK_18;
        if (randomize_signals.at("B"))
            state.B = rand() & MASK_18;
        if (randomize_signals.at("D"))
            state.D = rand() & MASK_18;
        if (randomize_signals.at("C"))
            state.C = (((int64_t)rand() << 16) | (int64_t)rand()) & MASK_48;
        if (randomize_signals.at("BCIN"))
            state.BCIN = rand() & MASK_18;
        if (randomize_signals.at("PCIN"))
            state.PCIN = (((int64_t)rand() << 16) | (int64_t)rand()) & MASK_48;
        if (randomize_signals.at("OPMODE"))
            state.OPMODE = (rand() & 0xFF);
        if (randomize_signals.at("CARRYIN"))
            state.CARRYIN = (bool)rand();

        // Simulate 4 clock cycles
        for (int j = 0; j < 4; j++)
        {
            // Copy current state to previous state
            DSP48A1_clock_cycle(&state, &prev);
            prev = state;

            // Convert numbers to binary strings
            to_binary_string(state.A, 18, bin_str_A);
            to_binary_string(state.B, 18, bin_str_B);
            to_binary_string(state.C, 48, bin_str_C);
            to_binary_string(state.D, 18, bin_str_D);
            to_binary_string(state.BCIN, 18, bin_str_BCIN);
            to_binary_string(state.PCIN, 48, bin_str_PCIN);
            to_binary_string(state.OPMODE, 8, bin_str_OPMODE);
            to_binary_string(state.CARRYIN, 1, bin_str_CARRYIN);
            to_binary_string(state.BCOUT, 18, bin_str_BCOUT);
            to_binary_string(state.M, 48, bin_str_M);
            to_binary_string(state.CARRYOUT, 1, bin_str_CARRYOUT);
            to_binary_string(state.P, 48, bin_str_P);

```

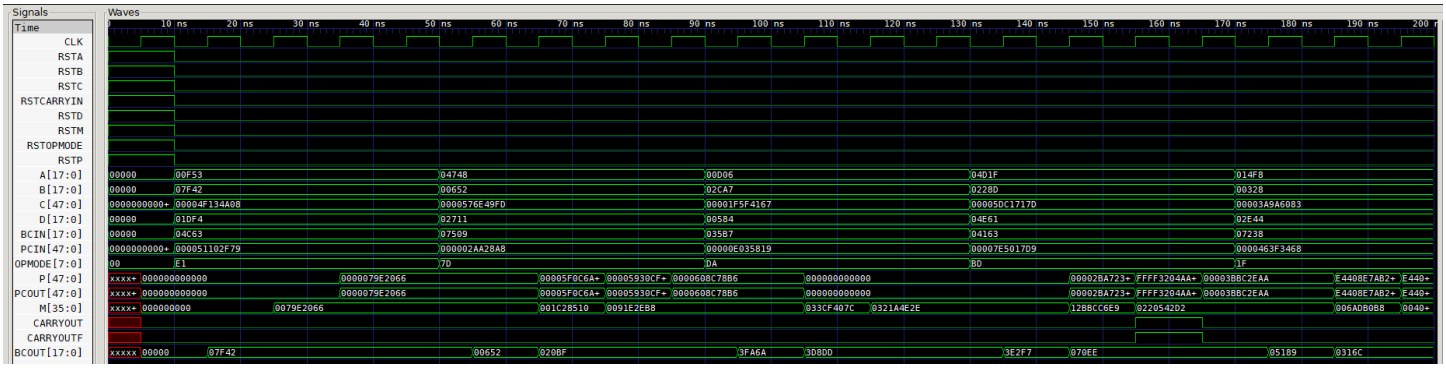
```
            // Write results to file
            fp << bin_str_A << " " << bin_str_B << " " << bin_str_C << " " << bin_str_D << " "
               << bin_str_BCIN << " " << bin_str_OPMODE << " " << bin_str_PCIN << " " << bin_str_CARRYIN
               << " -> " << bin_str_BCOUT << " " << bin_str_M << " " << bin_str_CARRYOUT << " " << bin_str_P << "\n";
        }
    }

    fp.close();
    std::cout << "Test vectors generated in " << filename << std::endl;

    // Add filename to filelist.txt
    std::ofstream filelist("filelist.txt", std::ios::app);
    if (!filelist.is_open())
    {
        std::cerr << "Error opening filelist.txt" << std::endl;
        return;
    }
    filelist << filename << std::endl;
    filelist.close();
}

int main()
{
    // Remove filelist.txt at the start
    std::remove("filelist.txt");

    memset(&state, 0, sizeof(DSP48A1_State));
    memset(&prev, 0, sizeof(DSP48A1_State));

    std::map<std::string, bool> randomize_signals;

    // --------------------------
    // Test Case 1: Fully randomized signals including OPMODE.
    // --------------------------
    randomize_signals = {
        {"A", true},
        {"B", true},
        {"C", true},
        {"D", true},
        {"BCIN", true},
        {"PCIN", true},
        {"OPMODE", true},
        {"CARRYIN", true}};
    generate_test_vectors(randomize_signals, 100);

    // --------------------------
    // Test Case 2: Fixed multiplication-only: OPMODE = 0x01 (00000001)
    // --------------------------
    state.OPMODE = 0x01;
    randomize_signals["OPMODE"] = false;
    generate_test_vectors(randomize_signals, 100);

    // --------------------------
    // Test Case 3: Fixed multiply-accumulate: OPMODE = 0x0B (00001011)
    // --------------------------
    state.OPMODE = 0x0B;
    generate_test_vectors(randomize_signals, 50);

    // --------------------------
    // Test Case 4: Fixed multiply-subtract: OPMODE = 0x1B (00011011)
    // --------------------------
    state.OPMODE = 0x1B;
    generate_test_vectors(randomize_signals, 50);

    // --------------------------
    // Test Case 5: Fixed addition: OPMODE = 0x26 (00100110)
    // --------------------------
    state.OPMODE = 0x26;
    generate_test_vectors(randomize_signals, 50);

    // --------------------------
    // Test Case 6: Fixed subtraction: OPMODE = 0x2E (00101110)
    // --------------------------
    state.OPMODE = 0x2E;
    generate_test_vectors(randomize_signals, 50);

    // --------------------------
    // Test Case 7: Fixed three-input addition: OPMODE = 0x66 (01100110)
    // --------------------------
    state.OPMODE = 0x66;
    generate_test_vectors(randomize_signals, 50);

    // --------------------------
    // Test Case 8: Fixed three-input add/subtract: OPMODE = 0x6E (01101110)
    // --------------------------
    state.OPMODE = 0x6E;
    generate_test_vectors(randomize_signals, 50);

    // --------------------------
    // Test Case 9: Fixed clear output: OPMODE = 0x00 (00000000)
    // --------------------------
    state.OPMODE = 0x00;
    generate_test_vectors(randomize_signals, 50);

    return 0;
}
```

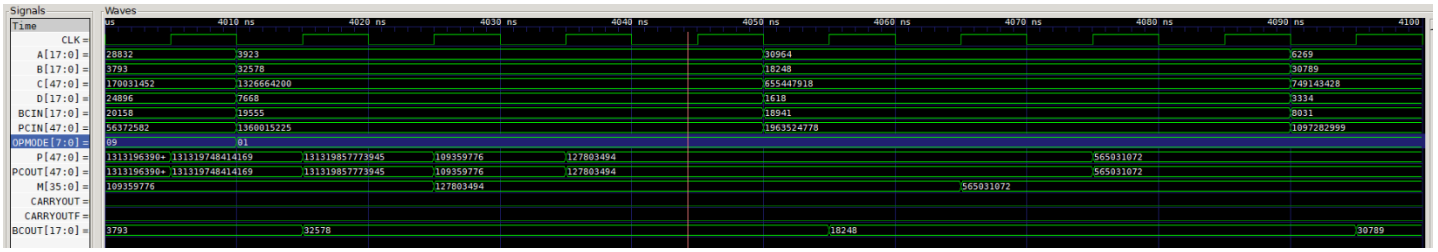
## Do file:

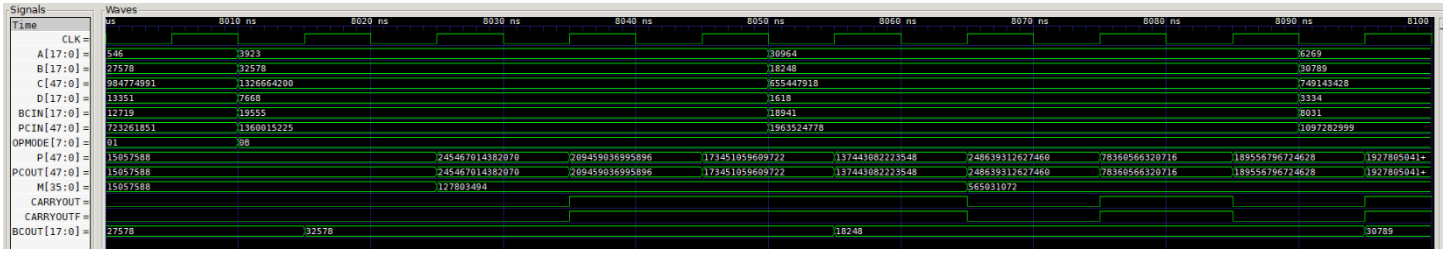
```
Projects - DSP48A1_tb.do

vlib work

vlog ../*.v ../*.sv

vsim -voptargs=+acc work.DSP48A1_tb

add wave *

run -all
```

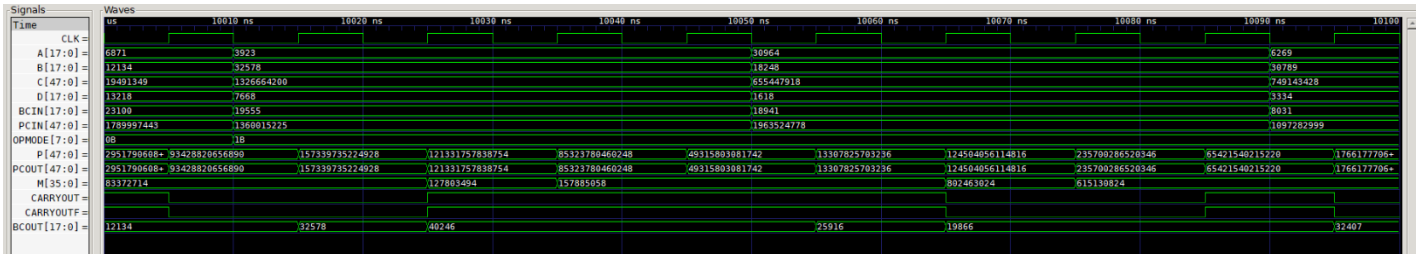
## Waveform snippets:

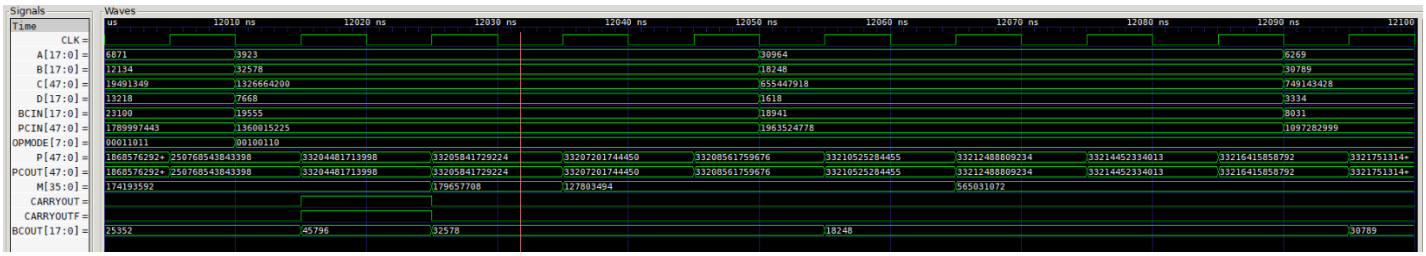
```
# do {D:\Learning\DigitalKareem\Projects\DSP48A1\sim\simulate.do}
# Testing with file: 1_golden_model_.txt
# Testing with file: 2_golden_model_OPMODE_.txt
# Testing with file: 3_golden_model_OPMODE_.txt
# Testing with file: 4_golden_model_OPMODE_.txt
# Testing with file: 5_golden_model_OPMODE_.txt
# Testing with file: 6_golden_model_OPMODE_.txt
# Testing with file: 7_golden_model_OPMODE_.txt
# Testing with file: 8_golden_model_OPMODE_.txt
# Testing with file: 9_golden_model_OPMODE_.txt
# ** Note: $finish    : ../DSP48A1_tb.sv(127)
#    Time: 22010 ns  Iteration: 1  Instance: /DSP48A1_tb
# End time: 17:40:00 on Feb 28,2025, Elapsed time: 0:00:07
# Errors: 0, Warnings: 0
Simulation Errors: 0
Simulation PASSED. Proceeding to waveform display...
```

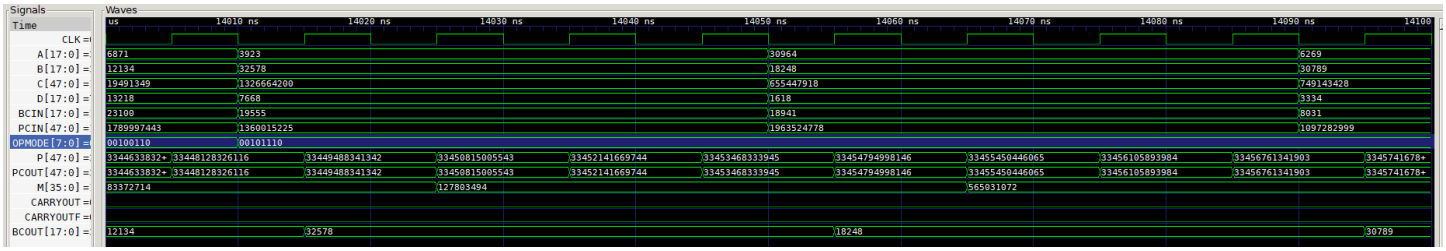


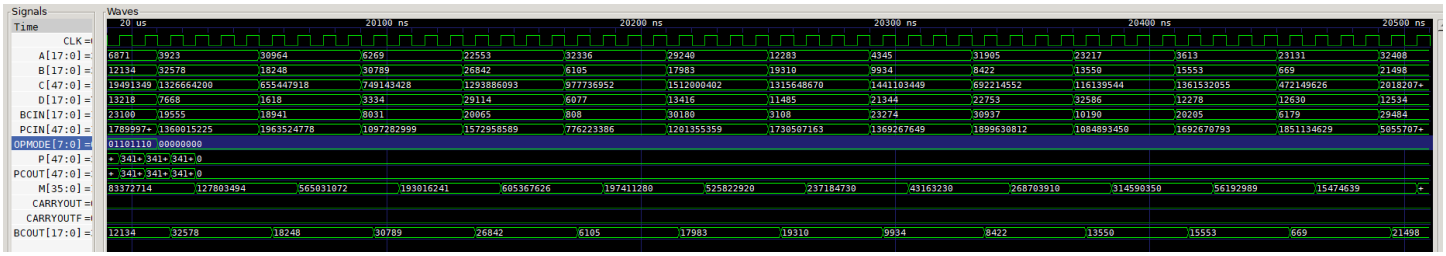

Signals
Waves
Time
CLK
A[17:0]
B[17:0]
C[47:0]
D[17:0]
BCIN[17:0]
PCIN[47:0]
OPMODE[7:0]
P[47:0]
PCOUT[47:0]
M[35:0]
CARRYOUT
CARRYOUTF
BCOUT[17:0]


Signals
Waves
Time
CLK
A[17:0]
B[17:0]
C[47:0]
D[17:0]
BCIN[17:0]
PCIN[47:0]
OPMODE[7:0]
P[47:0]
PCOUT[47:0]
M[35:0]
CARRYOUT
CARRYOUTF
BCOUT[17:0]


Signals
Waves
Time
CLK
A[17:0]
B[17:0]
C[47:0]
D[17:0]
BCIN[17:0]
PCIN[47:0]
OPMODE[7:0]
P[47:0]
PCOUT[47:0]
M[35:0]
CARRYOUT
CARRYOUTF
BCOUT[17:0]

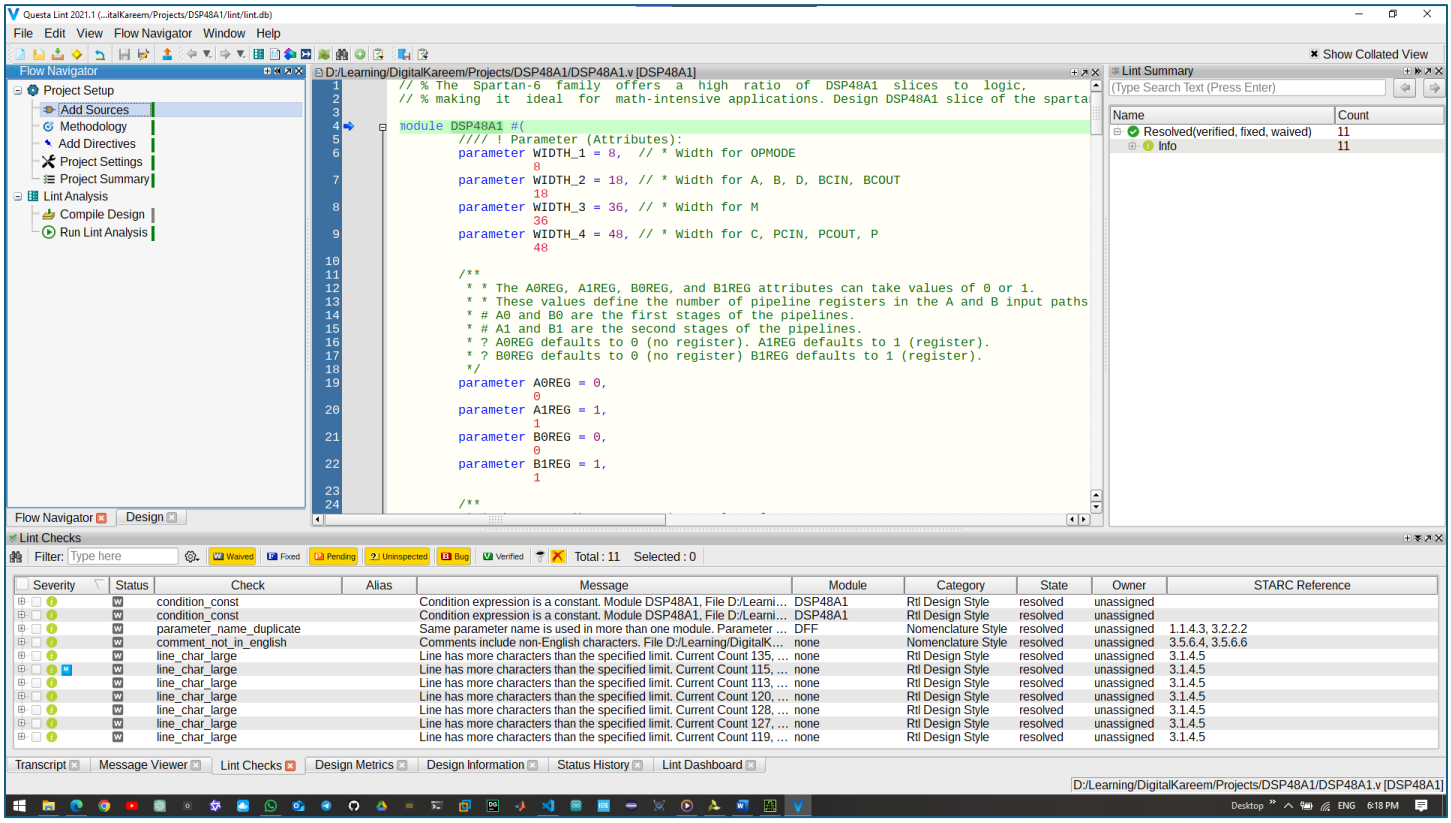

Signals
Waves
Time
CLK
A[17:0]
B[17:0]
C[47:0]
D[17:0]
BCIN[17:0]
PCIN[47:0]
OPMODE[7:0]
P[47:0]
PCOUT[47:0]
M[35:0]
CARRYOUT
CARRYOUTF
BCOUT[17:0]


Signals
Waves
Time
CLK
A[17:0]
B[17:0]
C[47:0]
D[17:0]
BCIN[17:0]
PCIN[47:0]
OPMODE[7:0]
P[47:0]
PCOUT[47:0]
M[35:0]
CARRYOUT
CARRYOUTF
BCOUT[17:0]


Signals
Waves
Time
CLK
A[17:0]
B[17:0]
C[47:0]
D[17:0]
BCIN[17:0]
PCIN[47:0]
OPMODE[7:0]
P[47:0]
PCOUT[47:0]
M[35:0]
CARRYOUT
CARRYOUTF
BCOUT[17:0]

# Lint snippets:

```
# Result Summary
# ------------------------------------------------------------
# Error (0)
# ------------------------------------------------------------
#
# ------------------------------------------------------------
# Warning (0)
# ------------------------------------------------------------
#
# ------------------------------------------------------------
# Info (11)
# ------------------------------------------------------------
#   condition_const                          :2
#   parameter_name_duplicate                 :1
#   comment_not_in_english                   :1
#   line_char_large                          :7
#
```

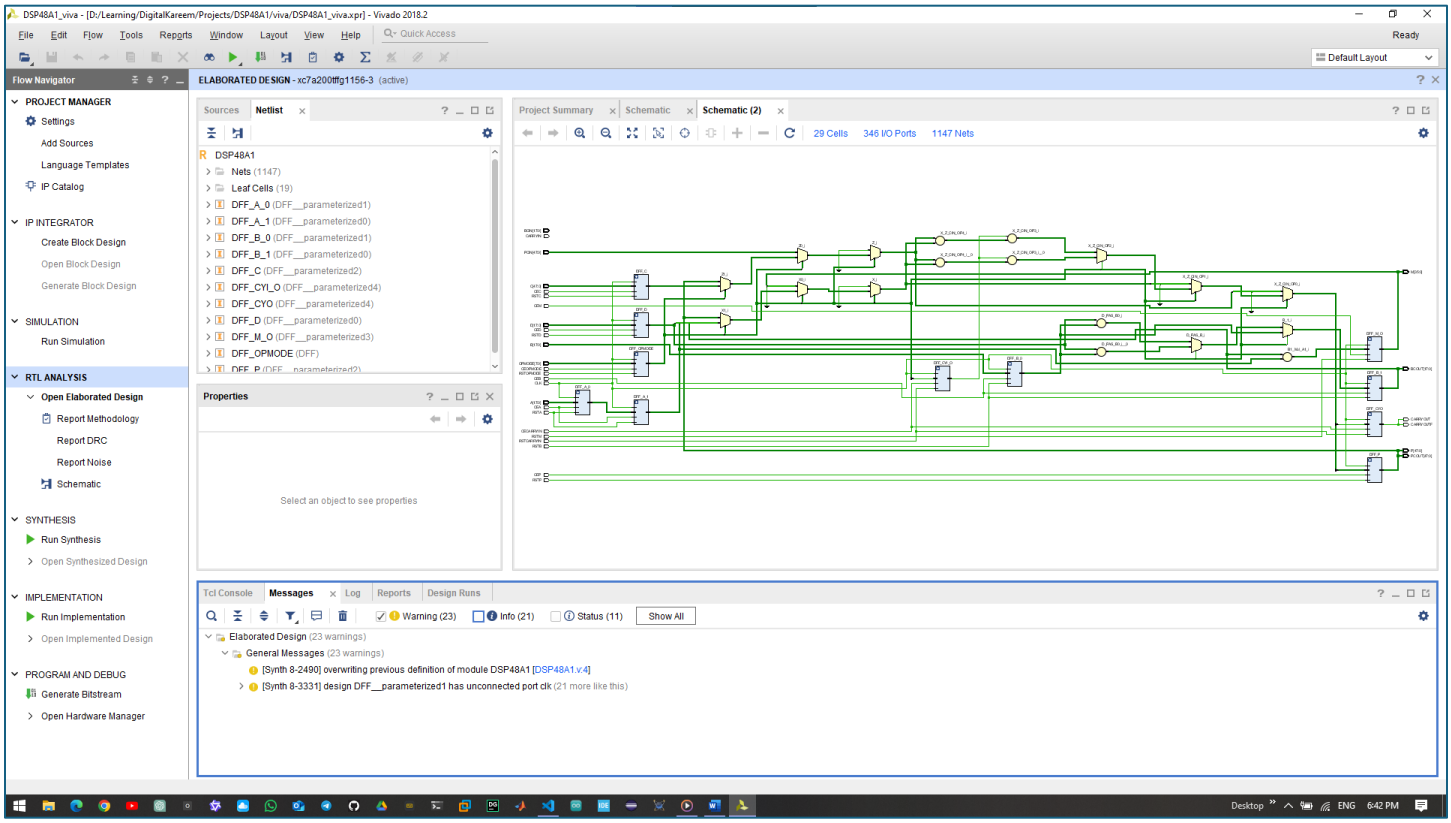


# Constraints file:

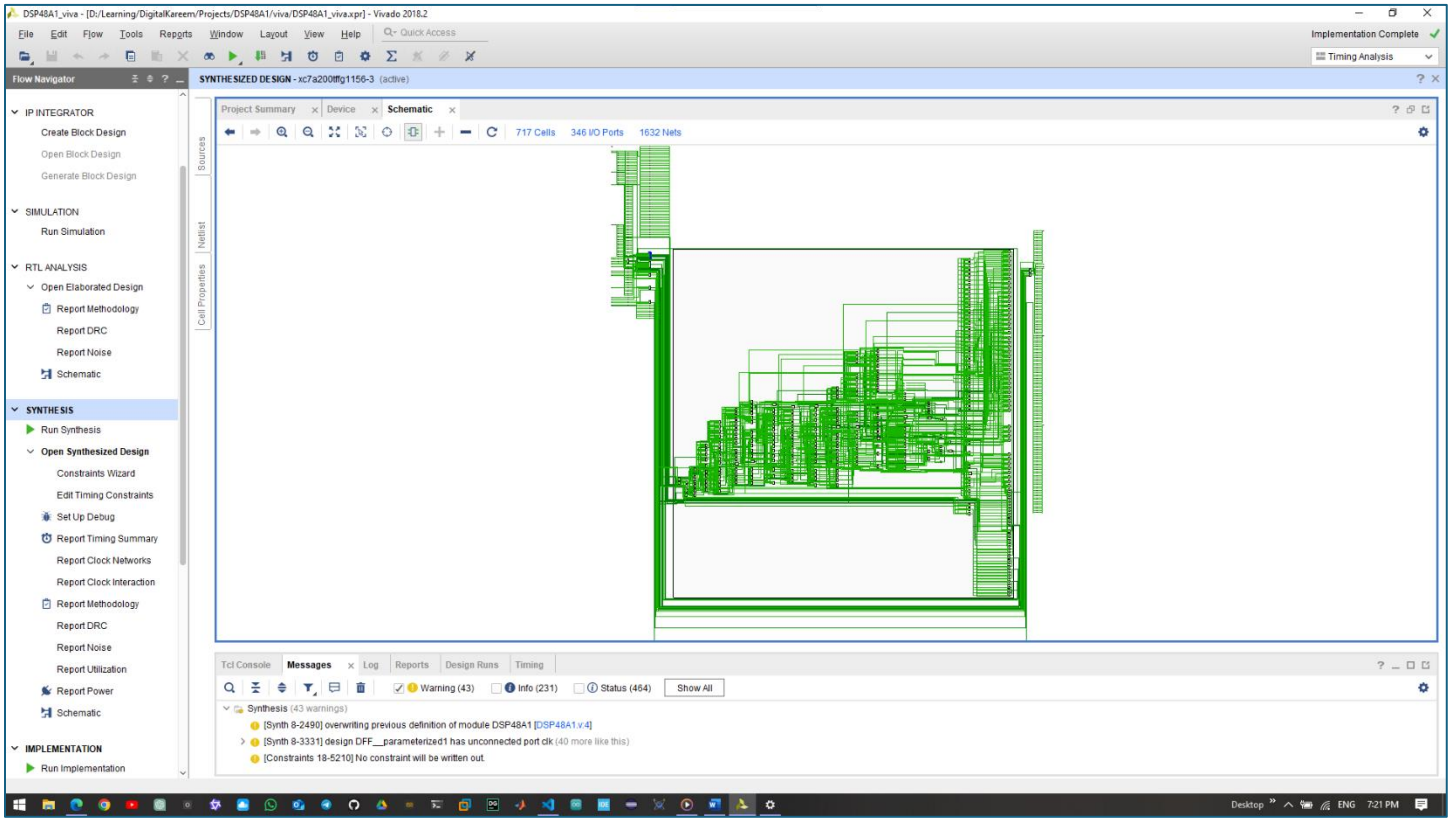
Projects - DSP48A1.xdc

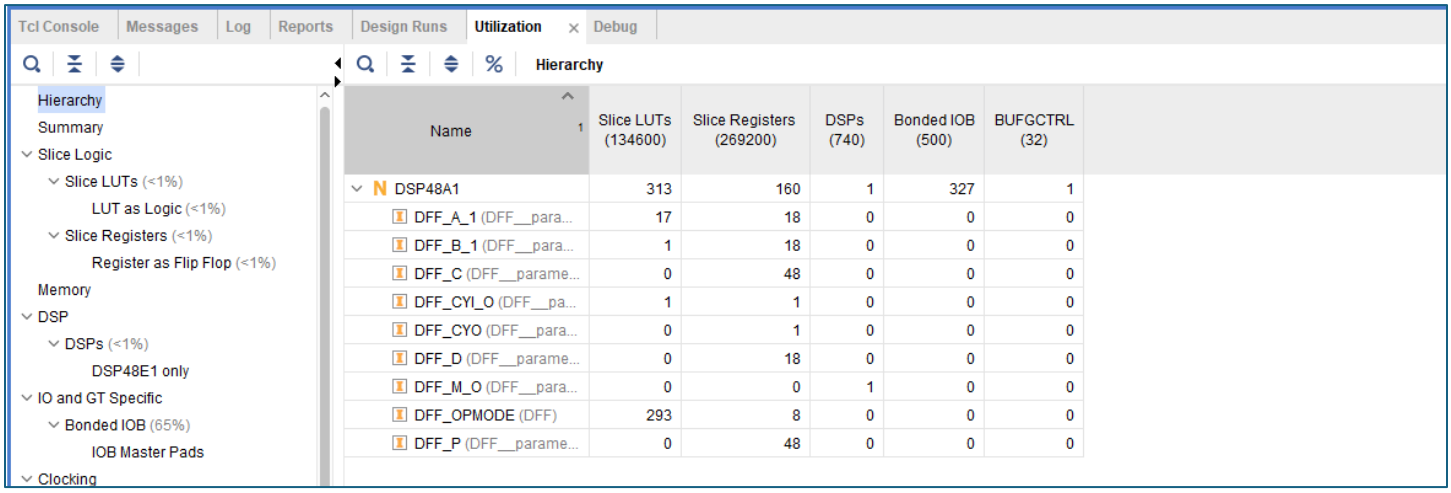
```
## FPGA part: xc7a200tffg1156-3

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


```

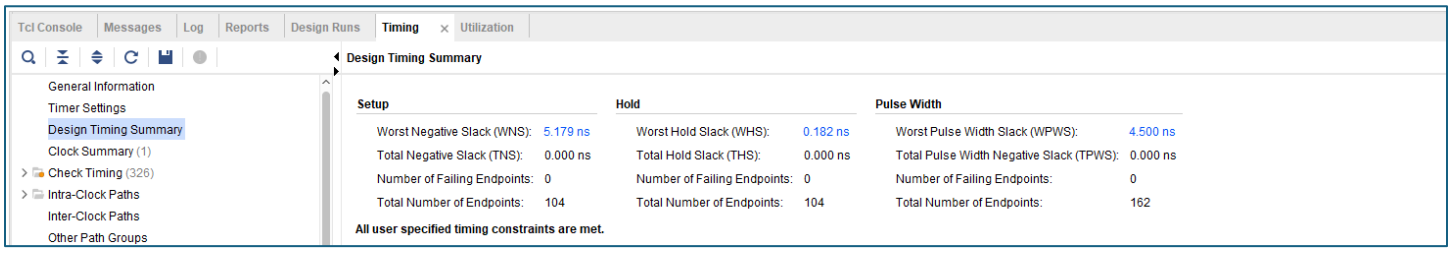
# Elaboration:



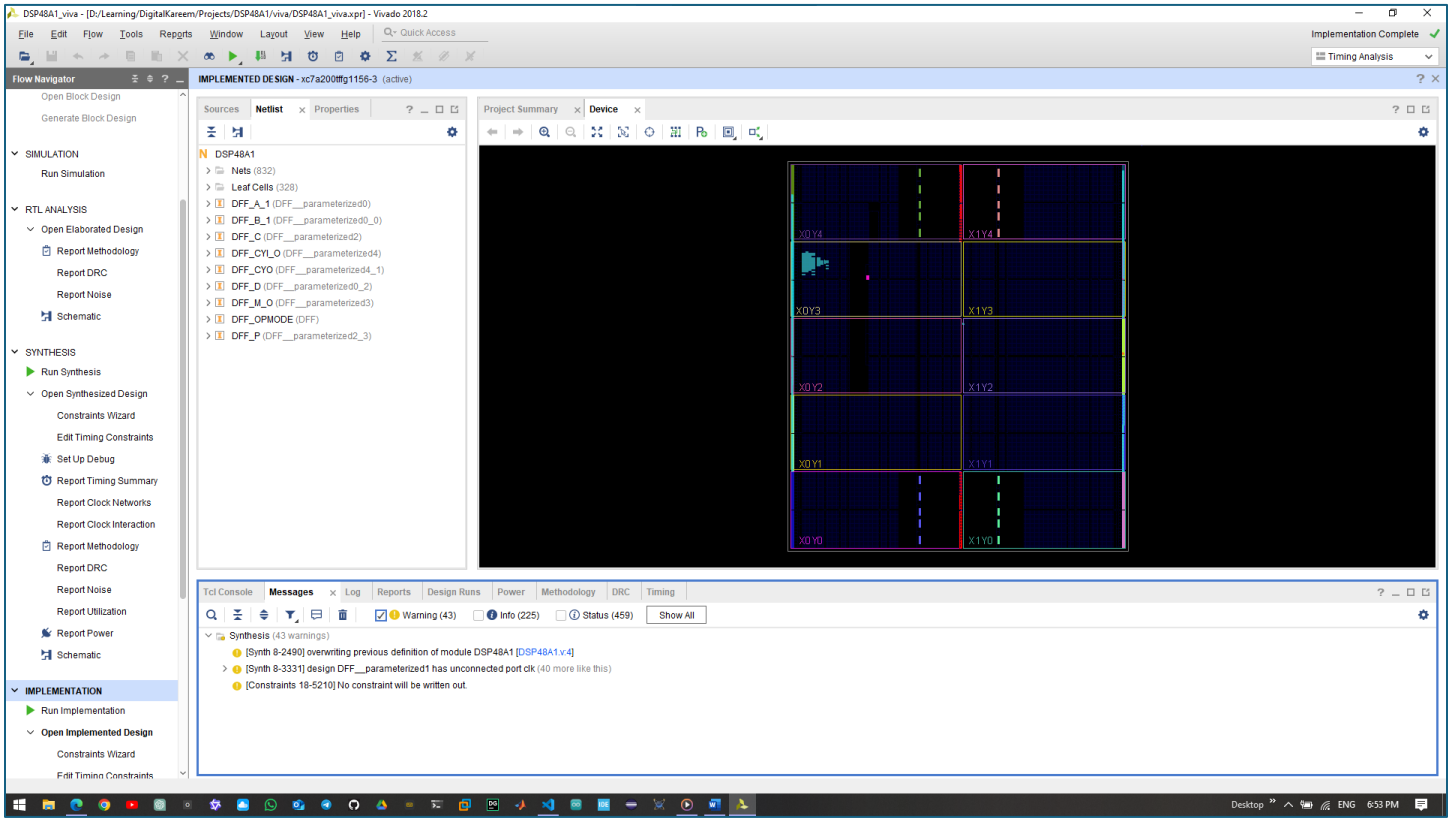
# Synthesis:

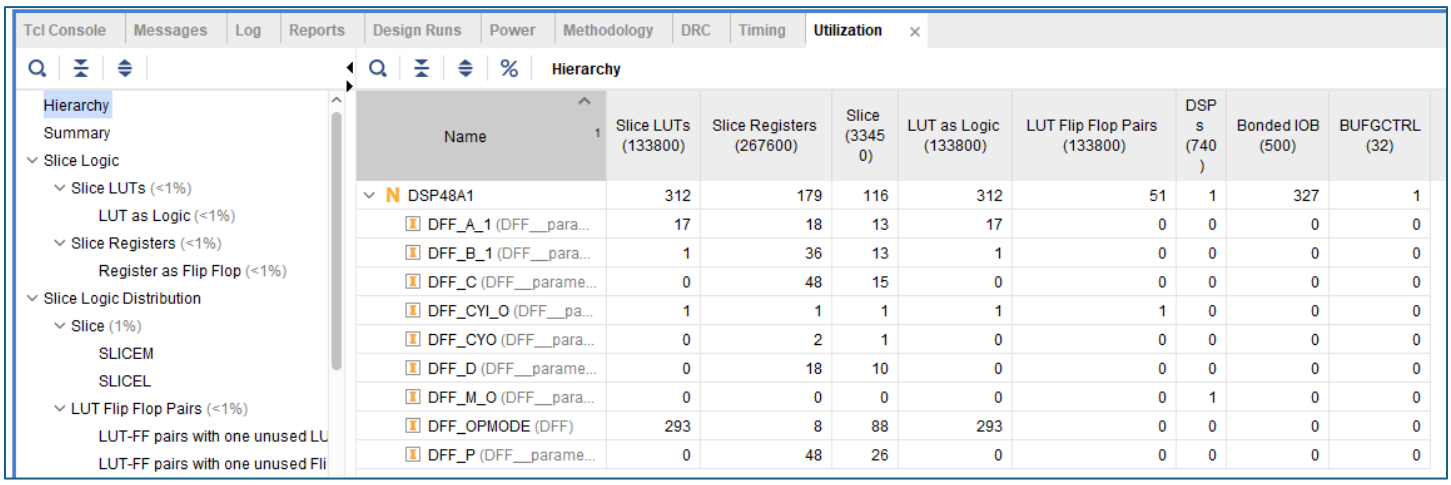

DSP48A1_viva - [D:/Learning/DigitalKareem/Projects/DSP48A1/viva/DSP48A1_viva.xpr] - Vivado 2018.2
File Edit Flow Tools Reports Window Layout View Help
Implementation Complete
Timing Analysis
Flow Navigator
SYNTHESIZED DESIGN - xc7a200tffg1156-3 (active)
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
Report Methodology
Report DRC
Report Noise
Schematic
SYNTHESIS
Run Synthesis
Open Synthesized Design
Constraints Wizard
Edit Timing Constraints
Set Up Debug
Report Timing Summary
Report Clock Networks
Report Clock Interaction
Report Methodology
Report DRC
Report Noise
Report Utilization
Report Power
Schematic
IMPLEMENTATION
Run Implementation
Project Summary
Device
Schematic
717 Cells
346 I/O Ports
1632 Nets
Tcl Console
Messages
Log
Reports
Design Runs
Timing
Warning (43)
Info (231)
Status (464)
Show All
Synthesis (43 warnings)
[Synth 8-2490] overwriting previous definition of module DSP48A1 [DSP48A1.v:4]
[Synth 8-3331] design DFF__parameterized1 has unconnected port clk (40 more like this)
[Constraints 18-5210] No constraint will be written out.

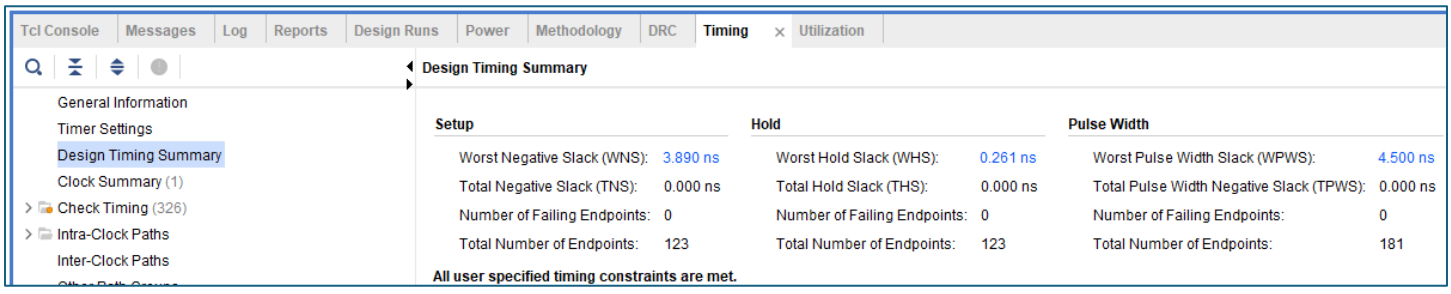

Tcl Console
Messages
Log
Reports
Design Runs
Utilization
Debug
Hierarchy
Summary
Slice Logic
Slice LUTs (<1%)
LUT as Logic (<1%)
Slice Registers (<1%)
Register as Flip Flop (<1%)
Memory
DSP
DSPs (<1%)
DSP48E1 only
IO and GT Specific
Bonded IOB (65%)
IOB Master Pads
Clocking
Hierarchy
Name
Slice LUTs (134600)
Slice Registers (269200)
DSPs (740)
Bonded IOB (500)
BUFGCTRL (32)
DSP48A1 313 160 1 327 1
DFF_A_1 (DFF__para... 17 18 0 0 0
DFF_B_1 (DFF__para... 1 18 0 0 0
DFF_C (DFF__parame... 0 48 0 0 0
DFF_CYI_O (DFF__pa... 1 1 0 0 0
DFF_CYO (DFF__para... 0 1 0 0 0
DFF_D (DFF__parame... 0 18 0 0 0
DFF_M_O (DFF__para... 0 0 1 0 0
DFF_OPMODE (DFF) 293 8 0 0 0
DFF_P (DFF__parame... 0 48 0 0 0


Tcl Console
Messages
Log
Reports
Design Runs
Timing
Utilization
General Information
Timer Settings
Design Timing Summary
Clock Summary (1)
Check Timing (326)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
Design Timing Summary
Setup
Worst Negative Slack (WNS): 5.179 ns
Total Negative Slack (TNS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 104
Hold
Worst Hold Slack (WHS): 0.182 ns
Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 104
Pulse Width
Worst Pulse Width Slack (WPWS): 4.500 ns
Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 162
All user specified timing constraints are met.

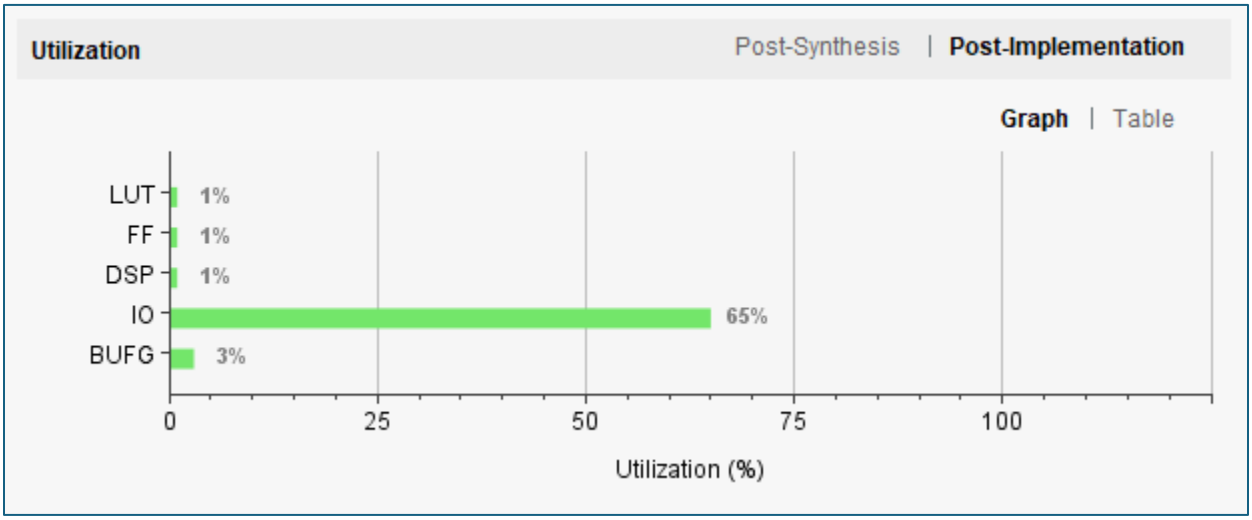
# Implementation:



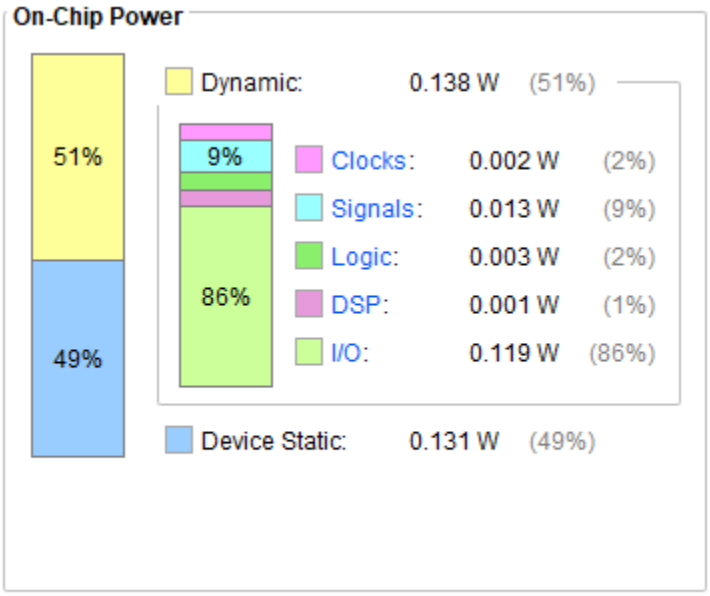
# Project Summary:




Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

| | |
|---|---|
| **Total On-Chip Power:** | **0.269 W** |
| **Design Power Budget:** | **Not Specified** |
| **Power Budget Margin:** | **N/A** |
| **Junction Temperature:** | **25.4°C** |
| Thermal Margin: | 74.6°C (50.8 W) |
| Effective ϑJA: | 1.5°C/W |
| Power supplied to off-chip devices: | 0 W |
| Confidence level: | Low |

Launch Power Constraint Advisor to find and fix invalid switching activity



# Notes:

- There were one minor mistake about opmode [5] where is used the direct one, I fixed the code in both Verilog and golden but it would take forever to recreate pictures and pdf but I assure you everything works perfectly.
- Testbench is in System Verilog because I needed an array of strings to hold the files names only this part to be easier than Verilog.
- I know that my code doesn't make Vivado map the module to the hard module on the FPGA that due to little differences that I find my code matches the datasheet more. (I could make them match but I am ok with the results if I wan to use the hard ones, simply go to Vivado and extract the exact module to use.
- Everything could be found in this repo: salah0eldin/Digital-Projects