**Lab Report 3**

**HLS and Dynamic Partial Reconfiguration**

**Mina Ibrahim Henien 49-1743**

**This report includes:**

* **C++ code for an arithmetic logic unit “Addition, Multiplication and Division”**
* **Testbench for the ALU**
* **Synthesize the C++ code to generate VHDL modules**
* **Use dynamic partial reconfiguration to switch dynamically between multiplier and divider while the adder remains functional**

**ALU Header file:-**

//if the header file is not defined

#ifndef Adder\_H

//define the header file

#define Adder\_H

#include <ap\_int.h>

void adder(ap\_uint<5> &out,ap\_uint<4> in1,ap\_uint<4> in2);

void multiplier(ap\_uint<8> &out,ap\_uint<4> in1,ap\_uint<4> in2);

//for partial fpga reconfiguration

void divider(ap\_uint<8> &out,ap\_uint<4> in1,ap\_uint<4> in2);

//void divider(ap\_ufixed<4,3> &out,ap\_uint<4> in1,ap\_uint<4> in2);

#endif

**C++ Code for ALU:-**

#include "Adder.h"

#include <ap\_int.h>

void adder(ap\_uint<5> &out,ap\_uint<4> in1,ap\_uint<4> in2){

out = in1 + in2;

}

void multiplier(ap\_uint<8> &out,ap\_uint<4> in1,ap\_uint<4> in2){

out = in1 \* in2;

}

//for partial fpga reconfiguration

void divider(ap\_uint<8> &out,ap\_uint<4> in1,ap\_uint<4> in2){

out = in1 / in2;

}

**Testbench for ALU:-**

#include <iostream>

#include "Adder.h"

int main(){

ap\_uint<5> out\_adder;

ap\_uint<8> out\_multi;

ap\_uint<8> out\_div;

ap\_uint<4> in1;

ap\_uint<4> in2;

//Test 1: in1 = 10, in2 = 5

in1 = 10;

in2 = 5;

adder(out\_adder,in1,in2);

multiplier(out\_multi,in1,in2);

divider(out\_div,in1,in2);

std::cout << "the value of the output addition is " << out\_adder << std::endl;

std::cout << "the value of the output multiplication is " << out\_multi << std::endl;

std::cout << "the value of the output division is " << out\_div << std::endl;

if(out\_adder == 15) std::cout << "True!" << std::endl;

else std::cout << "False!" << std::endl;

if(out\_multi == 50) std::cout << "True!" << std::endl;

else std::cout << "False!" << std::endl;

if(out\_div == 2) std::cout << "True!" << std::endl;

else std::cout << "False!" << std::endl;

//Test 2: in1 = 15, in2 = 3

in1 = 15;

in2 = 3;

adder(out\_adder,in1,in2);

multiplier(out\_multi,in1,in2);

divider(out\_div,in1,in2);

std::cout << "the value of the output addition is " << out\_adder << std::endl;

std::cout << "the value of the output multiplication is " << out\_multi << std::endl;

std::cout << "the value of the output division is " << out\_div << std::endl;

if(out\_adder == 18) std::cout << "True!" << std::endl;

else std::cout << "False!" << std::endl;

if(out\_multi == 45) std::cout << "True!" << std::endl;

else std::cout << "False!" << std::endl;

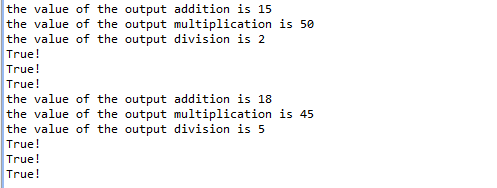
if(out\_div == 5) std::cout << "True!" << std::endl;

else std::cout << "False!" << std::endl;

return 0;

}

**ALU testbench results:-**



**VHDL code for adder:-**

-- ==============================================================

-- RTL generated by Vivado(TM) HLS - High-Level Synthesis from C, C++ and OpenCL

-- Version: 2020.1

-- Copyright (C) 1986-2020 Xilinx, Inc. All Rights Reserved.

--

-- ===========================================================

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity adder is

port (

ap\_start : IN STD\_LOGIC;

ap\_done : OUT STD\_LOGIC;

ap\_idle : OUT STD\_LOGIC;

ap\_ready : OUT STD\_LOGIC;

out\_V : OUT STD\_LOGIC\_VECTOR (4 downto 0);

out\_V\_ap\_vld : OUT STD\_LOGIC;

in1\_V : IN STD\_LOGIC\_VECTOR (3 downto 0);

in2\_V : IN STD\_LOGIC\_VECTOR (3 downto 0) );

end;

architecture behav of adder is

attribute CORE\_GENERATION\_INFO : STRING;

attribute CORE\_GENERATION\_INFO of behav : architecture is

"adder,hls\_ip\_2020\_1,{HLS\_INPUT\_TYPE=cxx,HLS\_INPUT\_FLOAT=0,HLS\_INPUT\_FIXED=1,HLS\_INPUT\_PART=xc7a35t-cpg236-1,HLS\_INPUT\_CLOCK=10.000000,HLS\_INPUT\_ARCH=others,HLS\_SYN\_CLOCK=1.777000,HLS\_SYN\_LAT=0,HLS\_SYN\_TPT=none,HLS\_SYN\_MEM=0,HLS\_SYN\_DSP=0,HLS\_SYN\_FF=0,HLS\_SYN\_LUT=15,HLS\_VERSION=2020\_1}";

constant ap\_const\_logic\_1 : STD\_LOGIC := '1';

constant ap\_const\_logic\_0 : STD\_LOGIC := '0';

constant ap\_const\_boolean\_1 : BOOLEAN := true;

signal rhs\_V\_fu\_39\_p1 : STD\_LOGIC\_VECTOR (4 downto 0);

signal lhs\_V\_fu\_35\_p1 : STD\_LOGIC\_VECTOR (4 downto 0);

begin

ap\_done <= ap\_start;

ap\_idle <= ap\_const\_logic\_1;

ap\_ready <= ap\_start;

lhs\_V\_fu\_35\_p1 <= std\_logic\_vector(IEEE.numeric\_std.resize(unsigned(in1\_V),5));

out\_V <= std\_logic\_vector(unsigned(rhs\_V\_fu\_39\_p1) + unsigned(lhs\_V\_fu\_35\_p1));

out\_V\_ap\_vld\_assign\_proc : process(ap\_start)

begin

if ((ap\_start = ap\_const\_logic\_1)) then

out\_V\_ap\_vld <= ap\_const\_logic\_1;

else

out\_V\_ap\_vld <= ap\_const\_logic\_0;

end if;

end process;

rhs\_V\_fu\_39\_p1 <= std\_logic\_vector(IEEE.numeric\_std.resize(unsigned(in2\_V),5));

end behav;

**VHDL code for multiplier:-**

-- ==============================================================

-- RTL generated by Vivado(TM) HLS - High-Level Synthesis from C, C++ and OpenCL

-- Version: 2020.1

-- Copyright (C) 1986-2020 Xilinx, Inc. All Rights Reserved.

--

-- ==========================================================

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity multiplier is

port (

ap\_clk : IN STD\_LOGIC;

ap\_rst : IN STD\_LOGIC;

ap\_start : IN STD\_LOGIC;

ap\_done : OUT STD\_LOGIC;

ap\_idle : OUT STD\_LOGIC;

ap\_ready : OUT STD\_LOGIC;

out\_V : OUT STD\_LOGIC\_VECTOR (7 downto 0);

out\_V\_ap\_vld : OUT STD\_LOGIC;

in1\_V : IN STD\_LOGIC\_VECTOR (3 downto 0);

in2\_V : IN STD\_LOGIC\_VECTOR (3 downto 0);

mult\_or\_div: out std\_logic);

end;

architecture behav of multiplier is

attribute CORE\_GENERATION\_INFO : STRING;

attribute CORE\_GENERATION\_INFO of behav : architecture is

"multiplier,hls\_ip\_2020\_1,{HLS\_INPUT\_TYPE=cxx,HLS\_INPUT\_FLOAT=0,HLS\_INPUT\_FIXED=1,HLS\_INPUT\_PART=xc7a35t-cpg236-1,HLS\_INPUT\_CLOCK=10.000000,HLS\_INPUT\_ARCH=others,HLS\_SYN\_CLOCK=2.050000,HLS\_SYN\_LAT=0,HLS\_SYN\_TPT=none,HLS\_SYN\_MEM=0,HLS\_SYN\_DSP=0,HLS\_SYN\_FF=0,HLS\_SYN\_LUT=13,HLS\_VERSION=2020\_1}";

constant ap\_const\_logic\_1 : STD\_LOGIC := '1';

constant ap\_const\_boolean\_1 : BOOLEAN := true;

constant ap\_const\_logic\_0 : STD\_LOGIC := '0';

signal ret\_V\_fu\_43\_p0 : STD\_LOGIC\_VECTOR (3 downto 0);

signal ret\_V\_fu\_43\_p1 : STD\_LOGIC\_VECTOR (3 downto 0);

signal ret\_V\_fu\_43\_p00 : STD\_LOGIC\_VECTOR (7 downto 0);

signal ret\_V\_fu\_43\_p10 : STD\_LOGIC\_VECTOR (7 downto 0);

begin

ap\_done <= ap\_start;

ap\_idle <= ap\_const\_logic\_1;

ap\_ready <= ap\_start;

out\_V <= std\_logic\_vector(IEEE.numeric\_std.resize(unsigned(ret\_V\_fu\_43\_p0) \* unsigned(ret\_V\_fu\_43\_p1), 8));

out\_V\_ap\_vld\_assign\_proc : process(ap\_start)

begin

if ((ap\_start = ap\_const\_logic\_1)) then

out\_V\_ap\_vld <= ap\_const\_logic\_1;

else

out\_V\_ap\_vld <= ap\_const\_logic\_0;

end if;

end process;

ret\_V\_fu\_43\_p0 <= ret\_V\_fu\_43\_p00(4 - 1 downto 0);

ret\_V\_fu\_43\_p00 <= std\_logic\_vector(IEEE.numeric\_std.resize(unsigned(in2\_V),8));

ret\_V\_fu\_43\_p1 <= ret\_V\_fu\_43\_p10(4 - 1 downto 0);

ret\_V\_fu\_43\_p10 <= std\_logic\_vector(IEEE.numeric\_std.resize(unsigned(in1\_V),8));

mult\_or\_div <= '0';

end behav;

**VHDL code for divider:-**

-- ==============================================================

-- RTL generated by Vivado(TM) HLS - High-Level Synthesis from C, C++ and OpenCL

-- Version: 2020.1

-- Copyright (C) 1986-2020 Xilinx, Inc. All Rights Reserved.

--

-- ===========================================================

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity divider is

port (

ap\_clk : IN STD\_LOGIC;

ap\_rst : IN STD\_LOGIC;

ap\_start : IN STD\_LOGIC;

ap\_done : OUT STD\_LOGIC;

ap\_idle : OUT STD\_LOGIC;

ap\_ready : OUT STD\_LOGIC;

out\_V : OUT STD\_LOGIC\_VECTOR (7 downto 0);

out\_V\_ap\_vld : OUT STD\_LOGIC;

in1\_V : IN STD\_LOGIC\_VECTOR (3 downto 0);

in2\_V : IN STD\_LOGIC\_VECTOR (3 downto 0);

mult\_or\_div: out std\_logic);

end;

architecture behav of divider is

attribute CORE\_GENERATION\_INFO : STRING;

attribute CORE\_GENERATION\_INFO of behav : architecture is

"divider,hls\_ip\_2020\_1,{HLS\_INPUT\_TYPE=cxx,HLS\_INPUT\_FLOAT=0,HLS\_INPUT\_FIXED=1,HLS\_INPUT\_PART=xc7a35t-cpg236-1,HLS\_INPUT\_CLOCK=10.000000,HLS\_INPUT\_ARCH=others,HLS\_SYN\_CLOCK=2.941000,HLS\_SYN\_LAT=7,HLS\_SYN\_TPT=none,HLS\_SYN\_MEM=0,HLS\_SYN\_DSP=0,HLS\_SYN\_FF=66,HLS\_SYN\_LUT=67,HLS\_VERSION=2020\_1}";

constant ap\_const\_logic\_1 : STD\_LOGIC := '1';

constant ap\_const\_logic\_0 : STD\_LOGIC := '0';

constant ap\_ST\_fsm\_state1 : STD\_LOGIC\_VECTOR (7 downto 0) := "00000001";

constant ap\_ST\_fsm\_state2 : STD\_LOGIC\_VECTOR (7 downto 0) := "00000010";

constant ap\_ST\_fsm\_state3 : STD\_LOGIC\_VECTOR (7 downto 0) := "00000100";

constant ap\_ST\_fsm\_state4 : STD\_LOGIC\_VECTOR (7 downto 0) := "00001000";

constant ap\_ST\_fsm\_state5 : STD\_LOGIC\_VECTOR (7 downto 0) := "00010000";

constant ap\_ST\_fsm\_state6 : STD\_LOGIC\_VECTOR (7 downto 0) := "00100000";

constant ap\_ST\_fsm\_state7 : STD\_LOGIC\_VECTOR (7 downto 0) := "01000000";

constant ap\_ST\_fsm\_state8 : STD\_LOGIC\_VECTOR (7 downto 0) := "10000000";

constant ap\_const\_lv32\_0 : STD\_LOGIC\_VECTOR (31 downto 0) := "00000000000000000000000000000000";

constant ap\_const\_lv32\_7 : STD\_LOGIC\_VECTOR (31 downto 0) := "00000000000000000000000000000111";

constant ap\_const\_boolean\_1 : BOOLEAN := true;

signal ap\_CS\_fsm : STD\_LOGIC\_VECTOR (7 downto 0) := "00000001";

attribute fsm\_encoding : string;

attribute fsm\_encoding of ap\_CS\_fsm : signal is "none";

signal ap\_CS\_fsm\_state1 : STD\_LOGIC;

attribute fsm\_encoding of ap\_CS\_fsm\_state1 : signal is "none";

signal ap\_CS\_fsm\_state8 : STD\_LOGIC;

attribute fsm\_encoding of ap\_CS\_fsm\_state8 : signal is "none";

signal grp\_fu\_35\_p2 : STD\_LOGIC\_VECTOR (3 downto 0);

signal grp\_fu\_35\_ap\_start : STD\_LOGIC;

signal grp\_fu\_35\_ap\_done : STD\_LOGIC;

signal ap\_NS\_fsm : STD\_LOGIC\_VECTOR (7 downto 0);

component divider\_udiv\_4ns\_bkb IS

generic (

ID : INTEGER;

NUM\_STAGE : INTEGER;

din0\_WIDTH : INTEGER;

din1\_WIDTH : INTEGER;

dout\_WIDTH : INTEGER );

port (

clk : IN STD\_LOGIC;

reset : IN STD\_LOGIC;

start : IN STD\_LOGIC;

done : OUT STD\_LOGIC;

din0 : IN STD\_LOGIC\_VECTOR (3 downto 0);

din1 : IN STD\_LOGIC\_VECTOR (3 downto 0);

ce : IN STD\_LOGIC;

dout : OUT STD\_LOGIC\_VECTOR (3 downto 0) );

end component;

begin

divider\_udiv\_4ns\_bkb\_U1 : component divider\_udiv\_4ns\_bkb

generic map (

ID => 1,

NUM\_STAGE => 8,

din0\_WIDTH => 4,

din1\_WIDTH => 4,

dout\_WIDTH => 4)

port map (

clk => ap\_clk,

reset => ap\_rst,

start => grp\_fu\_35\_ap\_start,

done => grp\_fu\_35\_ap\_done,

din0 => in1\_V,

din1 => in2\_V,

ce => ap\_const\_logic\_1,

dout => grp\_fu\_35\_p2);

ap\_CS\_fsm\_assign\_proc : process(ap\_clk)

begin

if (ap\_clk'event and ap\_clk = '1') then

if (ap\_rst = '1') then

ap\_CS\_fsm <= ap\_ST\_fsm\_state1;

else

ap\_CS\_fsm <= ap\_NS\_fsm;

end if;

end if;

end process;

ap\_NS\_fsm\_assign\_proc : process (ap\_start, ap\_CS\_fsm, ap\_CS\_fsm\_state1)

begin

case ap\_CS\_fsm is

when ap\_ST\_fsm\_state1 =>

if (((ap\_const\_logic\_1 = ap\_CS\_fsm\_state1) and (ap\_start = ap\_const\_logic\_1))) then

ap\_NS\_fsm <= ap\_ST\_fsm\_state2;

else

ap\_NS\_fsm <= ap\_ST\_fsm\_state1;

end if;

when ap\_ST\_fsm\_state2 =>

ap\_NS\_fsm <= ap\_ST\_fsm\_state3;

when ap\_ST\_fsm\_state3 =>

ap\_NS\_fsm <= ap\_ST\_fsm\_state4;

when ap\_ST\_fsm\_state4 =>

ap\_NS\_fsm <= ap\_ST\_fsm\_state5;

when ap\_ST\_fsm\_state5 =>

ap\_NS\_fsm <= ap\_ST\_fsm\_state6;

when ap\_ST\_fsm\_state6 =>

ap\_NS\_fsm <= ap\_ST\_fsm\_state7;

when ap\_ST\_fsm\_state7 =>

ap\_NS\_fsm <= ap\_ST\_fsm\_state8;

when ap\_ST\_fsm\_state8 =>

ap\_NS\_fsm <= ap\_ST\_fsm\_state1;

when others =>

ap\_NS\_fsm <= "XXXXXXXX";

end case;

end process;

ap\_CS\_fsm\_state1 <= ap\_CS\_fsm(0);

ap\_CS\_fsm\_state8 <= ap\_CS\_fsm(7);

ap\_done\_assign\_proc : process(ap\_CS\_fsm\_state8)

begin

if ((ap\_const\_logic\_1 = ap\_CS\_fsm\_state8)) then

ap\_done <= ap\_const\_logic\_1;

else

ap\_done <= ap\_const\_logic\_0;

end if;

end process;

ap\_idle\_assign\_proc : process(ap\_start, ap\_CS\_fsm\_state1)

begin

if (((ap\_start = ap\_const\_logic\_0) and (ap\_const\_logic\_1 = ap\_CS\_fsm\_state1))) then

ap\_idle <= ap\_const\_logic\_1;

else

ap\_idle <= ap\_const\_logic\_0;

end if;

end process;

ap\_ready\_assign\_proc : process(ap\_CS\_fsm\_state8)

begin

if ((ap\_const\_logic\_1 = ap\_CS\_fsm\_state8)) then

ap\_ready <= ap\_const\_logic\_1;

else

ap\_ready <= ap\_const\_logic\_0;

end if;

end process;

grp\_fu\_35\_ap\_start\_assign\_proc : process(ap\_start, ap\_CS\_fsm\_state1)

begin

if (((ap\_const\_logic\_1 = ap\_CS\_fsm\_state1) and (ap\_start = ap\_const\_logic\_1))) then

grp\_fu\_35\_ap\_start <= ap\_const\_logic\_1;

else

grp\_fu\_35\_ap\_start <= ap\_const\_logic\_0;

end if;

end process;

out\_V <= std\_logic\_vector(IEEE.numeric\_std.resize(unsigned(grp\_fu\_35\_p2),8));

out\_V\_ap\_vld\_assign\_proc : process(ap\_CS\_fsm\_state8)

begin

if ((ap\_const\_logic\_1 = ap\_CS\_fsm\_state8)) then

out\_V\_ap\_vld <= ap\_const\_logic\_1;

else

out\_V\_ap\_vld <= ap\_const\_logic\_0;

end if;

end process;

mult\_or\_div <= '1';

end behav;

**VHDL code for Top\_Module:-**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Top\_Module is

port (

ap\_clk : IN STD\_LOGIC;

ap\_rst : IN STD\_LOGIC;

ap\_start : IN STD\_LOGIC;

out\_seven\_segment : OUT STD\_LOGIC\_VECTOR (6 downto 0);

out\_V : OUT STD\_LOGIC\_VECTOR (7 downto 0);

in1 : IN STD\_LOGIC\_VECTOR (3 downto 0);

in2 : IN STD\_LOGIC\_VECTOR (3 downto 0);

mult\_or\_div: out std\_logic);

end;

architecture Behavioral of Top\_Module is

signal out\_add: std\_logic\_vector(4 downto 0);

signal ap\_done\_add :STD\_LOGIC;

signal ap\_idle\_add :STD\_LOGIC;

signal ap\_ready\_add : STD\_LOGIC;

signal out\_V\_ap\_vld\_add : STD\_LOGIC;

signal ap\_done\_mult :STD\_LOGIC;

signal ap\_idle\_mult :STD\_LOGIC;

signal ap\_ready\_mult : STD\_LOGIC;

signal out\_V\_ap\_vld\_mult : STD\_LOGIC;

component adder is

port (

ap\_start : IN STD\_LOGIC;

ap\_done : OUT STD\_LOGIC;

ap\_idle : OUT STD\_LOGIC;

ap\_ready : OUT STD\_LOGIC;

out\_V : OUT STD\_LOGIC\_VECTOR (4 downto 0);

out\_V\_ap\_vld : OUT STD\_LOGIC;

in1\_V : IN STD\_LOGIC\_VECTOR (3 downto 0);

in2\_V : IN STD\_LOGIC\_VECTOR (3 downto 0));

end component;

component multiplier is

port (

ap\_clk : IN STD\_LOGIC;

ap\_rst : IN STD\_LOGIC;

ap\_start : IN STD\_LOGIC;

ap\_done : OUT STD\_LOGIC;

ap\_idle : OUT STD\_LOGIC;

ap\_ready : OUT STD\_LOGIC;

out\_V : OUT STD\_LOGIC\_VECTOR (7 downto 0);

out\_V\_ap\_vld : OUT STD\_LOGIC;

in1\_V : IN STD\_LOGIC\_VECTOR (3 downto 0);

in2\_V : IN STD\_LOGIC\_VECTOR (3 downto 0);

mult\_or\_div: out std\_logic);

end component;

component seven\_segment\_decoder is

Port (data\_in: in std\_logic\_vector(3 downto 0); display\_out\_0: out std\_logic\_vector(6 downto 0));

end component;

begin

Add: adder port map(ap\_start,ap\_done\_add,ap\_idle\_add,ap\_ready\_add,out\_add,out\_V\_ap\_vld\_add,in1,in2);

seven\_segment: seven\_segment\_decoder port map(out\_add(3 downto 0),out\_seven\_segment);

Mult: multiplier port map(ap\_clk,ap\_rst,ap\_start,ap\_done\_mult,ap\_idle\_mult,ap\_ready\_mult,out\_V,out\_V\_ap\_vld\_mult,in1,in2,mult\_or\_div);

end Behavioral;

**Seven segment decoder to display adder results:-**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity seven\_segment\_decoder is

Port (data\_in: in std\_logic\_vector(3 downto 0); display\_out\_0: out std\_logic\_vector(6 downto 0));

end seven\_segment\_decoder;

architecture Behavioral of seven\_segment\_decoder is

begin

process(data\_in) begin

case (data\_in) is

when "0000" => display\_out\_0 <= "0000001"; --display 0

when "0001" => display\_out\_0 <= "1001111"; --display 1

when "0010" => display\_out\_0 <= "0010010"; --display 2

when "0011" => display\_out\_0 <= "0000110"; --display 3

when "0100" => display\_out\_0 <= "1001100"; --display 4

when "0101" => display\_out\_0 <= "0100100"; --display 5

when "0110" => display\_out\_0 <= "0100000"; --display 6

when "0111" => display\_out\_0 <= "0001111"; --display 7

when "1000" => display\_out\_0 <= "0000000"; --display 8

when "1001" => display\_out\_0 <= "0000100"; --display 9

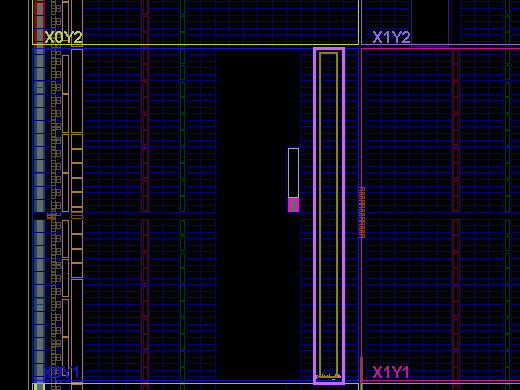
when others => display\_out\_0 <= "1111111"; --NULL

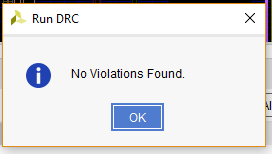
end case;

end process;

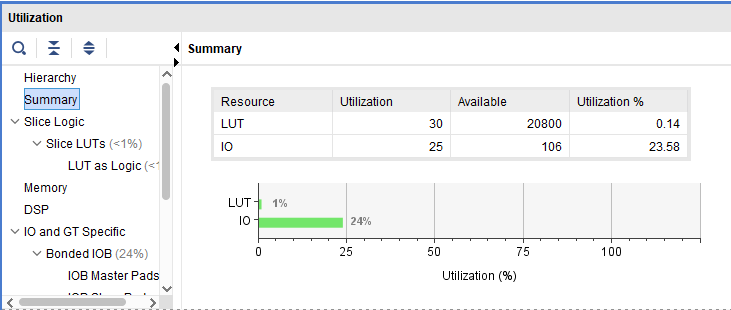
end Behavioral;

**Floorplanning:-**





**Utilization:-**



**Partial Bitstream:-**

