****

**EE4620L/EE6620L/CEG4324L/CEG6324L**

**DIGITAL INTEGRATED CIRCUIT DESIGN LAB**

**Project**

By

Current, Logan

UID: U01019510

Email: Current.22@wright.edu

Submission date: 04/18/2024

“I have neither given nor received aid on this assignment, nor have I observed any violation of the Honor code”

Signature :



Date : 04/18/2024

**Table of Contents**

**1. Aim/Objective (p.g. 3)**

**2. 12-bit pipeline Arch & WPD Calculation (p.g 3)**

**3. VHDL Code & TB Code (p.g. 4)**

**4. Simulation Waveform (p.g. 21)**

**5. Post-synthesis Schematic Diagram (p.g. 22)**

**6. Hardware Report (p.g. 24)**

**7. Compare Hardware & Clock Speed (p.g. 26)**

**8. Conclusion (p.g. 26)**

**List of Figures**

**2. 12-bit pipelining Arch & WPD Calculation: 2 figures showing how the pipelining will look for each case conceptually**

**4. Simulation Waveform: 2 images showing the waveform for each of the cases**

**5. Post-synthesis Schematic Diagram: 2 images showing the schematic diagram**

**6. Hardware report: 2 images portraying the hardware generated from the code and compiler**

1. AIM / OBJECTIVE:

The aim of this lab was to explore and design the concept of pipelining as well as implementing it in the Vivado environment.

1. 12 -bit pipeline Arch & WPD Calculation:

Case 1:

WPD = (6 \* 15ns) + (2 \* 5ns) = 100nsA piece of paper with math equations

Description automatically generated

Case 2:

WPD = (4 \* 15ns) + (2 \* 5ns) = 70ns

A piece of paper with writing on it

Description automatically generated

1. VHDL Code & TB Code:

VHDL Code:

Case 1 :

----------------------------------------------------------------------------------

---REGISTER

library ieee;

use ieee.std\_logic\_1164.all;

entity REG is

    port(D, CLK, RSTn: in std\_logic;

         Q: out std\_logic);

end entity;

architecture behaviour of REG is

begin

process(CLK)

    begin

        if (rising\_edge(CLK)) then

            if(RSTn = '1') then

                Q <= '0'

                  --pragma\_sythesis\_off

                  after 10ns

                  --pragma\_sythesis\_on

                  ;

               else

                Q <= D

                  --pragma\_sythesis\_off

                  after 10ns

                  --pragma\_sythesis\_on

                  ;

            end if;

        end if;

    end process;

end architecture;

----------------------------------------------------------------------------------

---FULL ADDER

library ieee;

use ieee.std\_logic\_1164.all;

entity FA is

    port (A, B, Ci: IN STD\_LOGIC ;

          S, Co: OUT STD\_LOGIC ) ;

end entity;

architecture behaviour of FA is

    begin

        S  <= A XOR B XOR Ci

        --pragma\_sythesis\_off

        after 10ns

        --pragma\_sythesis\_on

        ;

        Co <= (A AND B) OR (Ci AND A) OR (Ci AND B)

        --pragma\_sythesis\_off

        after 15ns

        --pragma\_sythesis\_on

        ;

end architecture;

----------------------------------------------------------------------------------

---PIPELINED ADDER

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity pipelined\_adder is

    generic(N :integer := 12); --4 bit pipelined adder is given here as an example

    Port   (A  : in STD\_LOGIC\_VECTOR (N-1 downto 0);

            B  : in STD\_LOGIC\_VECTOR (N-1 downto 0);

            Ci : in STD\_LOGIC;

            S  : out STD\_LOGIC\_VECTOR (N downto 0);

            CLK, RSTn : in STD\_LOGIC );

end pipelined\_adder;

architecture Behavioral of pipelined\_adder is

--- component declaration

component FA

    PORT (A, B, Ci: IN STD\_LOGIC ;

          S, Co: OUT STD\_LOGIC ) ;

end component;

component REG

    port(D, CLK, RSTn: in std\_logic;

         Q: out std\_logic);

end component;

--- signal declaration example is given below.

--- You'll need to modify signals and their vector sizes for your pipeline designs

signal Co       : std\_logic;    --use this signal as your last carry out signal

signal tmp\_cout : std\_logic\_vector(12 downto 0);

signal tmp\_S0   : std\_logic\_vector(1 downto 0);

signal tmp\_S1   : std\_logic\_vector(1 downto 0);

signal tmp\_S2   : std\_logic\_vector(1 downto 0);

signal tmp\_S3   : std\_logic\_vector(1 downto 0);

signal tmp\_S4   : std\_logic\_vector(1 downto 0);

signal tmp\_S5   : std\_logic\_vector(1 downto 0);

signal tmp\_S6   : std\_logic;

signal tmp\_S7   : std\_logic;

signal tmp\_S8   : std\_logic;

signal tmp\_S9   : std\_logic;

signal tmp\_S10   : std\_logic;

signal tmp\_S11   : std\_logic;

signal tmp\_A6 : std\_logic;

signal tmp\_B6 : std\_logic;

signal tmp\_A7 : std\_logic;

signal tmp\_B7 : std\_logic;

signal tmp\_A8 : std\_logic;

signal tmp\_B8 : std\_logic;

signal tmp\_A9 : std\_logic;

signal tmp\_B9 : std\_logic;

signal tmp\_A10 : std\_logic;

signal tmp\_B10 : std\_logic;

signal tmp\_A11 : std\_logic;

signal tmp\_B11 : std\_logic;

--- add necessary signals here

begin

--- 1st Full Adder

FA\_0    : FA port map(A(0), B(0), Ci, tmp\_S0(0), tmp\_cout(0));

reg\_s0\_0: REG port map(tmp\_S0(0), CLK, RSTn, tmp\_S0(1));

reg\_s0\_1: REG port map(tmp\_S0(1), CLK, RSTn, S(0));

--- 2nd Full Adder

FA\_1    : FA port map(A(1), B(1), tmp\_cout(0), tmp\_S1(0), tmp\_cout(1));

reg\_s1\_0: REG port map(tmp\_S1(0), CLK, RSTn, tmp\_S1(1));

reg\_s1\_1: REG port map(tmp\_S1(1), CLK, RSTn, S(1));

---3rd Full Adder

FA\_2: FA port map(A(2), B(2), tmp\_cout(1), tmp\_S2(0), tmp\_cout(2));

reg\_s2\_0: REG port map(tmp\_S2(0), CLK, RSTn, tmp\_S2(1));

reg\_s2\_1: REG port map(tmp\_S2(1), CLK, RSTn, S(2));

--- 4th Full Adder

FA\_3: FA port map(A(3), B(3), tmp\_cout(2), tmp\_S3(0), tmp\_cout(3));

reg\_s3\_0: REG port map(tmp\_S3(0), CLK, RSTn, tmp\_S3(1));

reg\_s3\_1: REG port map(tmp\_S3(1), CLK, RSTn, S(3));

--- 5th Full Adder

FA\_4: FA port map(A(4), B(4), tmp\_cout(3), tmp\_S4(0), tmp\_cout(4));

reg\_s4\_0: REG port map(tmp\_S4(0), CLK, RSTn, tmp\_S4(1));

reg\_s4\_1: REG port map(tmp\_S4(1), CLK, RSTn, S(4));

--- 6th Full Adder

FA\_5: FA port map(A(5), B(5), tmp\_cout(4), tmp\_S5(0), tmp\_cout(5));

reg\_s5\_0: REG port map(tmp\_S5(0), CLK, RSTn, tmp\_S5(1));

reg\_s5\_1: REG port map(tmp\_S5(1), CLK, RSTn, S(5));

cout\_reg\_0: REG port map(tmp\_cout(5), CLK, RSTn, tmp\_cout(6));

--- 7th Full Adder

reg\_tmp\_A6: REG port map(A(6), CLK, RSTn, tmp\_A6);

reg\_tmp\_B6: REG port map(B(6), CLK, RSTn, tmp\_B6);

FA\_6: FA port map(tmp\_A6, tmp\_B6, tmp\_cout(6), tmp\_S6, tmp\_cout(7));

reg\_s6: REG port map(tmp\_S6, CLK, RSTn, S(6));

--- 8th Full Adder

reg\_tmp\_A7: REG port map(A(7), CLK, RSTn, tmp\_A7);

reg\_tmp\_B7: REG port map(B(7), CLK, RSTn, tmp\_B7);

FA\_7: FA port map(tmp\_A7, tmp\_B7, tmp\_cout(7), tmp\_S7, tmp\_cout(8));

reg\_s7: REG port map(tmp\_S7, CLK, RSTn, S(7));

--- 9th Full Adder

reg\_tmp\_A8: REG port map(A(8), CLK, RSTn, tmp\_A8);

reg\_tmp\_B8: REG port map(B(8), CLK, RSTn, tmp\_B8);

FA\_8: FA port map(tmp\_A8, tmp\_B8, tmp\_cout(8), tmp\_S8, tmp\_cout(9));

reg\_s8: REG port map(tmp\_S8, CLK, RSTn, S(8));

--- 10th Full Adder

reg\_tmp\_A9: REG port map(A(9), CLK, RSTn, tmp\_A9);

reg\_tmp\_B9: REG port map(B(9), CLK, RSTn, tmp\_B9);

FA\_9: FA port map(tmp\_A9, tmp\_B9, tmp\_cout(9), tmp\_S9, tmp\_cout(10));

reg\_s9: REG port map(tmp\_S9, CLK, RSTn, S(9));

--- 11th Full Adder

reg\_tmp\_A10: REG port map(A(10), CLK, RSTn, tmp\_A10);

reg\_tmp\_B10: REG port map(B(10), CLK, RSTn, tmp\_B10);

FA\_10: FA port map(tmp\_A10, tmp\_B10, tmp\_cout(10), tmp\_S10, tmp\_cout(11));

reg\_s10: REG port map(tmp\_S10, CLK, RSTn, S(10));

--- 12th Full Adder

reg\_tmp\_A11: REG port map(A(11), CLK, RSTn, tmp\_A11);

reg\_tmp\_B11: REG port map(B(11), CLK, RSTn, tmp\_B11);

FA\_11: FA port map(tmp\_A11, tmp\_B11, tmp\_cout(11), tmp\_S11, tmp\_cout(12));

reg\_s11: REG port map(tmp\_S11, CLK, RSTn, S(11));

cout\_reg\_1: REG port map(tmp\_cout(12), CLK, RSTn, Co);

S(N)<=Co;

end Behavioral;

Case 2 :

----------------------------------------------------------------------------------

---REGISTER

library ieee;

use ieee.std\_logic\_1164.all;

entity REG is

    port(D, CLK, RSTn: in std\_logic;

         Q: out std\_logic);

end entity;

architecture behaviour of REG is

begin

process(CLK)

    begin

        if (rising\_edge(CLK)) then

            if(RSTn = '1') then

                Q <= '0'

                  --pragma\_sythesis\_off

                  after 10ns

                  --pragma\_sythesis\_on

                  ;

               else

                Q <= D

                  --pragma\_sythesis\_off

                  after 10ns

                  --pragma\_sythesis\_on

                  ;

            end if;

        end if;

    end process;

end architecture;

----------------------------------------------------------------------------------

---FULL ADDER

library ieee;

use ieee.std\_logic\_1164.all;

entity FA is

    port (A, B, Ci: IN STD\_LOGIC ;

          S, Co: OUT STD\_LOGIC ) ;

end entity;

architecture behaviour of FA is

    begin

        S  <= A XOR B XOR Ci

        --pragma\_sythesis\_off

        after 10ns

        --pragma\_sythesis\_on

        ;

        Co <= (A AND B) OR (Ci AND A) OR (Ci AND B)

        --pragma\_sythesis\_off

        after 15ns

        --pragma\_sythesis\_on

        ;

end architecture;

----------------------------------------------------------------------------------

---PIPELINED ADDER

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity pipelined\_adder is

    generic(N :integer := 12); --4 bit pipelined adder is given here as an example

    Port   (A  : in STD\_LOGIC\_VECTOR (N-1 downto 0);

            B  : in STD\_LOGIC\_VECTOR (N-1 downto 0);

            Ci : in STD\_LOGIC;

            S  : out STD\_LOGIC\_VECTOR (N downto 0);

            CLK, RSTn : in STD\_LOGIC );

end pipelined\_adder;

architecture Behavioral of pipelined\_adder is

--- component declaration

component FA

    PORT (A, B, Ci: IN STD\_LOGIC ;

          S, Co: OUT STD\_LOGIC ) ;

end component;

component REG

    port(D, CLK, RSTn: in std\_logic;

         Q: out std\_logic);

end component;

--- signal declaration example is given below.

--- You'll need to modify signals and their vector sizes for your pipeline designs

signal Co       : std\_logic;    --use this signal as your last carry out signal

signal tmp\_cout : std\_logic\_vector(13 downto 0);

signal tmp\_S0   : std\_logic\_vector(2 downto 0);

signal tmp\_S1   : std\_logic\_vector(2 downto 0);

signal tmp\_S2   : std\_logic\_vector(2 downto 0);

signal tmp\_S3   : std\_logic\_vector(2 downto 0);

signal tmp\_S4   : std\_logic\_vector(1 downto 0);

signal tmp\_S5   : std\_logic\_vector(1 downto 0);

signal tmp\_S6   : std\_logic\_vector(1 downto 0);

signal tmp\_S7   : std\_logic\_vector(1 downto 0);

signal tmp\_S8   : std\_logic;

signal tmp\_S9   : std\_logic;

signal tmp\_S10   : std\_logic;

signal tmp\_S11   : std\_logic;

signal tmp\_A4 : std\_logic;

signal tmp\_B4 : std\_logic;

signal tmp\_A5 : std\_logic;

signal tmp\_B5 : std\_logic;

signal tmp\_A6 : std\_logic;

signal tmp\_B6 : std\_logic;

signal tmp\_A7 : std\_logic;

signal tmp\_B7 : std\_logic;

signal tmp\_A8 : std\_logic\_vector(1 downto 0);

signal tmp\_B8 : std\_logic\_vector(1 downto 0);

signal tmp\_A9 : std\_logic\_vector(1 downto 0);

signal tmp\_B9 : std\_logic\_vector(1 downto 0);

signal tmp\_A10 : std\_logic\_vector(1 downto 0);

signal tmp\_B10 : std\_logic\_vector(1 downto 0);

signal tmp\_A11 : std\_logic\_vector(1 downto 0);

signal tmp\_B11 : std\_logic\_vector(1 downto 0);

--- add necessary signals here

begin

--- 1st Full Adder

FA\_0    : FA port map(A(0), B(0), Ci, tmp\_S0(0), tmp\_cout(0));

reg\_s0\_0: REG port map(tmp\_S0(0), CLK, RSTn, tmp\_S0(1));

reg\_s0\_1: REG port map(tmp\_S0(1), CLK, RSTn, tmp\_S0(2));

reg\_s0\_2: REG port map(tmp\_S0(2), CLK, RSTn, S(0));

--- 2nd Full Adder

FA\_1    : FA port map(A(1), B(1), tmp\_cout(0), tmp\_S1(0), tmp\_cout(1));

reg\_s1\_0: REG port map(tmp\_S1(0), CLK, RSTn, tmp\_S1(1));

reg\_s1\_1: REG port map(tmp\_S1(1), CLK, RSTn, tmp\_S1(2));

reg\_s1\_2: REG port map(tmp\_S1(2), CLK, RSTn, S(1));

---3rd Full Adder

FA\_2: FA port map(A(2), B(2), tmp\_cout(1), tmp\_S2(0), tmp\_cout(2));

reg\_s2\_0: REG port map(tmp\_S2(0), CLK, RSTn, tmp\_S2(1));

reg\_s2\_1: REG port map(tmp\_S2(1), CLK, RSTn, tmp\_S2(2));

reg\_s2\_2: REG port map(tmp\_S2(2), CLK, RSTn, S(2));

--- 4th Full Adder

FA\_3: FA port map(A(3), B(3), tmp\_cout(2), tmp\_S3(0), tmp\_cout(3));

reg\_s3\_0: REG port map(tmp\_S3(0), CLK, RSTn, tmp\_S3(1));

reg\_s3\_1: REG port map(tmp\_S3(1), CLK, RSTn, tmp\_S3(2));

reg\_s3\_2: REG port map(tmp\_S3(2), CLK, RSTn, S(3));

cout\_reg\_0: REG port map(tmp\_cout(3), CLK, RSTn, tmp\_cout(4));

--- 5th Full Adder

reg\_tmp\_A4: REG port map(A(4), CLK, RSTn, tmp\_A4);

reg\_tmp\_B4: REG port map(B(4), CLK, RSTn, tmp\_B4);

FA\_4: FA port map(tmp\_A4, tmp\_B4, tmp\_cout(4), tmp\_S4(0), tmp\_cout(5));

reg\_s4\_0: REG port map(tmp\_S4(0), CLK, RSTn, tmp\_S4(1));

reg\_s4\_1: REG port map(tmp\_S4(1), CLK, RSTn, S(4));

--- 6th Full Adder

reg\_tmp\_A5: REG port map(A(5), CLK, RSTn, tmp\_A5);

reg\_tmp\_B5: REG port map(B(5), CLK, RSTn, tmp\_B5);

FA\_5: FA port map(tmp\_A5, tmp\_B5, tmp\_cout(5), tmp\_S5(0), tmp\_cout(6));

reg\_s5\_0: REG port map(tmp\_S5(0), CLK, RSTn, tmp\_S5(1));

reg\_s5\_1: REG port map(tmp\_S5(1), CLK, RSTn, S(5));

--- 7th Full Adder

reg\_tmp\_A6: REG port map(A(6), CLK, RSTn, tmp\_A6);

reg\_tmp\_B6: REG port map(B(6), CLK, RSTn, tmp\_B6);

FA\_6: FA port map(tmp\_A6, tmp\_B6, tmp\_cout(6), tmp\_S6(0), tmp\_cout(7));

reg\_s6\_0: REG port map(tmp\_S6(0), CLK, RSTn, tmp\_S6(1));

reg\_s6\_1: REG port map(tmp\_S6(1), CLK, RSTn, S(6));

--- 8th Full Adder

reg\_tmp\_A7: REG port map(A(7), CLK, RSTn, tmp\_A7);

reg\_tmp\_B7: REG port map(B(7), CLK, RSTn, tmp\_B7);

FA\_7: FA port map(tmp\_A7, tmp\_B7, tmp\_cout(7), tmp\_S7(0), tmp\_cout(8));

reg\_s7\_0: REG port map(tmp\_S7(0), CLK, RSTn, tmp\_S7(1));

reg\_s7\_1: REG port map(tmp\_S7(1), CLK, RSTn, S(7));

cout\_reg\_1: REG port map(tmp\_cout(8), CLK, RSTn, tmp\_cout(9));

--- 9th Full Adder

reg\_tmp\_A8\_0: REG port map(A(8), CLK, RSTn, tmp\_A8(0));

reg\_tmp\_B8\_0: REG port map(B(8), CLK, RSTn, tmp\_B8(0));

reg\_tmp\_A8\_1: REG port map(tmp\_A8(0), CLK, RSTn, tmp\_A8(1));

reg\_tmp\_B8\_1: REG port map(tmp\_B8(0), CLK, RSTn, tmp\_B8(1));

FA\_8: FA port map(tmp\_A8(1), tmp\_B8(1), tmp\_cout(9), tmp\_S8, tmp\_cout(10));

reg\_s8: REG port map(tmp\_S8, CLK, RSTn, S(8));

--- 10th Full Adder

reg\_tmp\_A9\_0: REG port map(A(9), CLK, RSTn, tmp\_A9(0));

reg\_tmp\_B9\_0: REG port map(B(9), CLK, RSTn, tmp\_B9(0));

reg\_tmp\_A9\_1: REG port map(tmp\_A9(0), CLK, RSTn, tmp\_A9(1));

reg\_tmp\_B9\_1: REG port map(tmp\_B9(0), CLK, RSTn, tmp\_B9(1));

FA\_9: FA port map(tmp\_A9(1), tmp\_B9(1), tmp\_cout(10), tmp\_S9, tmp\_cout(11));

reg\_s9: REG port map(tmp\_S9, CLK, RSTn, S(9));

--- 11th Full Adder

reg\_tmp\_A10\_0: REG port map(A(10), CLK, RSTn, tmp\_A10(0));

reg\_tmp\_B10\_0: REG port map(B(10), CLK, RSTn, tmp\_B10(0));

reg\_tmp\_A10\_1: REG port map(tmp\_A10(0), CLK, RSTn, tmp\_A10(1));

reg\_tmp\_B10\_1: REG port map(tmp\_B10(0), CLK, RSTn, tmp\_B10(1));

FA\_10: FA port map(tmp\_A10(1), tmp\_B10(1), tmp\_cout(11), tmp\_S10, tmp\_cout(12));

reg\_s10: REG port map(tmp\_S10, CLK, RSTn, S(10));

--- 12th Full Adder

reg\_tmp\_A11\_0: REG port map(A(11), CLK, RSTn, tmp\_A11(0));

reg\_tmp\_B11\_0: REG port map(B(11), CLK, RSTn, tmp\_B11(0));

reg\_tmp\_A11\_1: REG port map(tmp\_A11(0), CLK, RSTn, tmp\_A11(1));

reg\_tmp\_B11\_1: REG port map(tmp\_B11(0), CLK, RSTn, tmp\_B11(1));

FA\_11: FA port map(tmp\_A11(1), tmp\_B11(1), tmp\_cout(12), tmp\_S11, tmp\_cout(13));

reg\_s11: REG port map(tmp\_S11, CLK, RSTn, S(11));

cout\_reg\_2: REG port map(tmp\_cout(13), CLK, RSTn, Co);

S(N)<=Co;

end Behavioral;

TB Code:

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date:

-- Design Name:

-- Module Name: tb\_pipelined\_adder - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity tb\_pipelined\_adder is

    generic(N :integer := 12); --4 bit pipelined adder is given here as an example

--  Port ( );

end tb\_pipelined\_adder;

architecture Behavioral of tb\_pipelined\_adder is

component pipelined\_adder is

    generic(N :integer := 12); --4 bit pipelined adder (change accordingly)

    Port ( A  : in STD\_LOGIC\_VECTOR (N-1 downto 0);

           B  : in STD\_LOGIC\_VECTOR (N-1 downto 0);

           Ci : in STD\_LOGIC;

           S  : out STD\_LOGIC\_VECTOR (N downto 0);

--           Co : out STD\_LOGIC;

           CLK, RSTn : in STD\_LOGIC );

end component;

   signal A,B : std\_logic\_vector (N-1 downto 0);

   signal Ci  : std\_logic := '0';

   signal clk : std\_logic := '0';

   signal rst : std\_logic := '1';

   signal Sum : std\_logic\_vector (N downto 0);

--   signal Co  : std\_logic;

constant clk\_period : time := 70 ns;     -- Clock Frequency

constant  P: integer:= 3;                -- number of pipeline stages

begin

UTT: pipelined\_adder generic map(N=>12) port map(A,B,Ci,Sum,clk,rst); --4 bit pipelined adder (change accordingly)

clk\_process :process

                  begin

                      clk <= '0';

                      wait for clk\_period/2;

                      clk <= '1';

                  wait for clk\_period/2;

              end process;

stim\_proc: process

              begin

-------------12-bit ADDER INPUTS are given here as an example

                    rst <= '1';

                    A <= "000000000000";

                    B <= "000000000000";

                    Ci<= '0';

                    wait for clk\_period;

                    rst <= '0';         --- 27+127

                    A <= "000000011011";

                    B <= "000001111111";

                    Ci<= '0';

                    wait for (P)\*clk\_period;

                    A <= "000000011011"; -- 27+127+1

                    B <= "000001111111";

                    Ci<= '1';

                    wait for (P)\*clk\_period;

                    A <= "000000011011"; -- 27+4095

                    B <= "111111111111";

                    Ci<= '0';

                    wait for (P)\*clk\_period;

                    A <= "000000011011"; -- 27+4095+1

                    B <= "111111111111";

                    Ci<= '1';

                    wait for (P)\*clk\_period;

                    A <= "000001110101"; --117+127

                    B <= "000001111111";

                    Ci<= '0';

                    wait for (P)\*clk\_period;

                    A <= "000001110101"; --117+127+1

                    B <= "000001111111";

                    Ci<= '1';

                    wait for (P)\*clk\_period;

                    A <= "000001110101"; --117+4095

                    B <= "111111111111";

                    Ci<= '0';

                    wait for (P)\*clk\_period;

                    A <= "000001110101"; --117+4095+1

                    B <= "111111111111";

                    Ci<= '1';

                    wait for (P)\*clk\_period;

                std.env.finish;

---------------------------4-bit ADDER INPUT-------------------------------

            end process;

end Behavioral;

1. Simulation Waveforms:

Case 1 :

A screenshot of a computer

Description automatically generated

Case 2 :

A screenshot of a computer

Description automatically generated

1. Post-synthesis Schematic Diagram:

Case 1:

A screenshot of a computer

Description automatically generated

Case 2:

A screenshot of a computer

Description automatically generated

1. Hardware Report:

Case 1:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Case 2:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

1. Compare Hardware & Clock Speed:

Hardware Comparison:

For both case 1 and case 2 they use over 320 FF, but in case 1 it uses 322 registers to make the 2-pipelining work and in case 2 it uses 333 registers in order to make it work. So, in these two cases, case 1 uses less hardware than case 2 uses in order to make the design.

Clock speed Comparison:

The speed than the time the first case takes is about 100ns to go through a full clock cycle, but in case 2 it uses less time at 70ns for a full clock cycle of the design.

1. Conclusion:

We learned in this project how to implement pipelining in the Vivado environment as well as comparing two cases of our pipelining. In the first case we designed a circuit using 2 pipelines for 12 1-bit full adders and in the second case we used 3 pipelines for the 12 1-bit full adders. Between the two cases here, case 1 ended up using less hardware than case 2 but it took more time to go though a full clock cycle than case 2 did. This showed us that 3 pipelines is faster than 2 pipelines for this particular project but it is more hardware intensive to design than using less pipelines as seen. Overall, this was a very fun and interesting project as it taught me how to implement pipelining in VHDL and in the Vivado environment as well as compare 2 different instances of pipelining and the effects of using more or less pipes in the same design.