****

**EE4620L/EE6620L/CEG4324L/CEG6324L**

**DIGITAL INTEGRATED CIRCUIT DESIGN LAB**

**Lab 4**

By

Yeoh\_\_\_\_\_\_\_\_\_ Alex\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

UID: U00931845

Email: yeoh.3@wright.edu

Submission date: 10/06/2024

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

Signature : Alex Yeoh

Date : 10/06/2024

**Table of Contents (include page #)**

I. Introduction (page 4)

II. Process (page 4)

III. Results (page 4)

IV. Conclusion (page 9)

V. Code (page 9)

**List of Figures**

Figure 1: Design for an 8-bit full adder pipelined between every adder (page 4)

Figure 2: Decimal waveform for an 8-bit full adder pipelined between every adder (page 4)

Figure 3: Hex waveform for an 8-bit full adder pipelined between every adder (page 5)

Figure 4: Binary waveform for an 8-bit full adder pipelined between every adder (page 5)

Figure 5: Schematic for an 8-bit full adder pipelined between every adder (page 5)

Figure 6: Hardware report for an 8-bit full adder pipelined between every adder (page 6)

Figure 7: 8-bit full adder pipelined between every adder on Zedboard (page 6)

Figure 8: Design for an 8-bit full adder pipelined between every other adder (page 7)

Figure 9: Decimal waveform for an 8-bit full adder pipelined between every other adder (page 7)

Figure 10: Hex waveform for an 8-bit full adder pipelined between every other adder (page 7)

Figure 11: Binary waveform for an 8-bit full adder pipelined between every other adder (page 7)

Figure 12: Schematic for an 8-bit full adder pipelined between every other adder (page 8)

Figure 13: Hardware report for an 8-bit full adder pipelined between every other adder (page 8)

Figure 14: 8-bit full adder pipelined between every other adder on Zedboard (page 9)

I. Introduction

The objective of the lab is to become more familiar with Vivado by building pipelined circuits and simulating them.

II. Process

We designed the layout for an 8-bit pipelined adder with every adder pipelined and an 8-bit pipelined adder with every other adder pipelined. We then modified the given 4-bit pipelined adder to become an 8-bit pipelined adder with every adder pipelined, and we then modified the 8-bit pipelined adder to an 8-bit pipelined adder with every other adder pipelined. We then ran the behavioral simulations for both pipelined adders. We finally loaded the program into the Zedboard and demonstrated it functioning on the Zedboard.

III. Results

1. 8-bit full adder pipelined between every adder

A diagram of a diagram

Description automatically generated with medium confidence

Figure 1: Design for an 8-bit full adder pipelined between every adder

From this design, the minimum clock period was determined to be 35ns.

A screenshot of a computer

Description automatically generated

Figure 2: Decimal waveform for an 8-bit full adder pipelined between every adder

A screenshot of a computer

Description automatically generated

Figure 3: Hex waveform for an 8-bit full adder pipelined between every adder

A screenshot of a computer

Description automatically generated

Figure 4: Binary waveform for an 8-bit full adder pipelined between every adder

A computer screen shot of a computer program

Description automatically generated

Figure 5: Schematic for an 8-bit full adder pipelined between every adder

A screenshot of a computer

Description automatically generated

Figure 6: Hardware report for an 8-bit full adder pipelined between every adder

A green circuit board with red and blue lights

Description automatically generated

Figure 7: 8-bit full adder pipelined between every adder on Zedboard

1. 8-bit full adder pipelined between every other adder

A diagram of a block diagram

Description automatically generated

Figure 8: Design for an 8-bit full adder pipelined between every other adder

From this design, the minimum clock period was determined to be 60ns.

A screenshot of a computer

Description automatically generated

Figure 9: Decimal waveform for an 8-bit full adder pipelined between every other adder

A screenshot of a computer

Description automatically generated

Figure 10: Hex waveform for an 8-bit full adder pipelined between every other adder

A screenshot of a computer

Description automatically generated

Figure 11: Binary waveform for an 8-bit full adder pipelined between every other adder

A computer screen shot of a computer program

Description automatically generated

Figure 12: Schematic for an 8-bit full adder pipelined between every other adder

A screenshot of a computer

Description automatically generated

Figure 13: Hardware report for an 8-bit full adder pipelined between every other adder

A green electronic board with many small ports

Description automatically generated with medium confidence

Figure 14: 8-bit full adder pipelined between every other adder on Zedboard

IV. Conclusion

From this lab, I have learned how to implement pipelining in VHDL code.

V. Code

|  |
| --- |
| ----------------------------------------------------------------------------------  ---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 20ns  --pragma\_sythesis\_on  ;  Co <= (A AND B) OR (Ci AND A) OR (Ci AND B)  --pragma\_sythesis\_off  AFTER 25ns  --pragma\_sythesis\_on  ;  END ARCHITECTURE;  ----------------------------------------------------------------------------------  ---PIPELINED ADDER  LIBRARY IEEE;  USE IEEE.STD\_LOGIC\_1164.ALL;  ENTITY pipelined\_adder IS  GENERIC (N : INTEGER := 4); --4 bit pipelined adder is given here (change accordingly for 8 bit)  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  SIGNAL Co : STD\_LOGIC; --use this signal as your last carry out signal  SIGNAL tmp\_cout : STD\_LOGIC\_VECTOR(7 DOWNTO 0);  SIGNAL tmp\_cin : STD\_LOGIC\_VECTOR(7 DOWNTO 1);  SIGNAL tmp\_S0 : STD\_LOGIC\_VECTOR(7 DOWNTO 0);  SIGNAL tmp\_S1 : STD\_LOGIC\_VECTOR(6 DOWNTO 0);  SIGNAL tmp\_S2 : STD\_LOGIC\_VECTOR(5 DOWNTO 0);  SIGNAL tmp\_S3 : STD\_LOGIC\_VECTOR(4 DOWNTO 0);  SIGNAL tmp\_S4 : STD\_LOGIC\_VECTOR(3 DOWNTO 0);  SIGNAL tmp\_S5 : STD\_LOGIC\_VECTOR(2 DOWNTO 0);  SIGNAL tmp\_S6 : STD\_LOGIC\_VECTOR(1 DOWNTO 0);  SIGNAL tmp\_S7 : STD\_LOGIC;  SIGNAL tmp\_A1 : STD\_LOGIC;  SIGNAL tmp\_B1 : STD\_LOGIC;  SIGNAL tmp\_A2 : STD\_LOGIC\_VECTOR(1 DOWNTO 0);  SIGNAL tmp\_B2 : STD\_LOGIC\_VECTOR(1 DOWNTO 0);  SIGNAL tmp\_A3 : STD\_LOGIC\_VECTOR(2 DOWNTO 0);  SIGNAL tmp\_B3 : STD\_LOGIC\_VECTOR(2 DOWNTO 0);  SIGNAL tmp\_A4 : STD\_LOGIC\_VECTOR(3 DOWNTO 0);  SIGNAL tmp\_B4 : STD\_LOGIC\_VECTOR(3 DOWNTO 0);  SIGNAL tmp\_A5 : STD\_LOGIC\_VECTOR(4 DOWNTO 0);  SIGNAL tmp\_B5 : STD\_LOGIC\_VECTOR(4 DOWNTO 0);  SIGNAL tmp\_A6 : STD\_LOGIC\_VECTOR(5 DOWNTO 0);  SIGNAL tmp\_B6 : STD\_LOGIC\_VECTOR(5 DOWNTO 0);  SIGNAL tmp\_A7 : STD\_LOGIC\_VECTOR(6 DOWNTO 0);  SIGNAL tmp\_B7 : STD\_LOGIC\_VECTOR(6 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, tmp\_S0(3));  reg\_s0\_3 : REG PORT MAP(tmp\_S0(3), CLK, RSTn, tmp\_S0(4));  reg\_s0\_4 : REG PORT MAP(tmp\_S0(4), CLK, RSTn, tmp\_S0(5));  reg\_s0\_5 : REG PORT MAP(tmp\_S0(5), CLK, RSTn, tmp\_S0(6));  reg\_s0\_6 : REG PORT MAP(tmp\_S0(6), CLK, RSTn, tmp\_S0(7));  reg\_s0\_7 : REG PORT MAP(tmp\_S0(7), CLK, RSTn, S(0));  reg\_cou0 : REG PORT MAP(tmp\_cout(0), CLK, RSTn, tmp\_cin(1));  --- 2nd Full Adder  reg\_A1 : REG PORT MAP(A(1), CLK, RSTn, tmp\_A1);  reg\_B1 : REG PORT MAP(B(1), CLK, RSTn, tmp\_B1);  FA\_1 : FA PORT MAP(tmp\_A1, tmp\_B1, tmp\_cin(1), 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, tmp\_S1(3));  reg\_s1\_3 : REG PORT MAP(tmp\_S1(3), CLK, RSTn, tmp\_S1(4));  reg\_s1\_4 : REG PORT MAP(tmp\_S1(4), CLK, RSTn, tmp\_S1(5));  reg\_s1\_5 : REG PORT MAP(tmp\_S1(5), CLK, RSTn, tmp\_S1(6));  reg\_s1\_6 : REG PORT MAP(tmp\_S1(6), CLK, RSTn, S(1));  reg\_cou1 : REG PORT MAP(tmp\_cout(1), CLK, RSTn, tmp\_cin(2));  ---3rd Full Adder  reg\_A2\_0 : REG PORT MAP(A(2), CLK, RSTn, tmp\_A2(0));  reg\_B2\_0 : REG PORT MAP(B(2), CLK, RSTn, tmp\_B2(0));  reg\_A2\_1 : REG PORT MAP(tmp\_A2(0), CLK, RSTn, tmp\_A2(1));  reg\_B2\_1 : REG PORT MAP(tmp\_B2(0), CLK, RSTn, tmp\_B2(1));  FA\_2 : FA PORT MAP(tmp\_A2(1), tmp\_B2(1), tmp\_cin(2), 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, tmp\_S2(3));  reg\_s2\_3 : REG PORT MAP(tmp\_S2(3), CLK, RSTn, tmp\_S2(4));  reg\_s2\_4 : REG PORT MAP(tmp\_S2(4), CLK, RSTn, tmp\_S2(5));  reg\_s2\_5 : REG PORT MAP(tmp\_S2(5), CLK, RSTn, S(2));  reg\_cou2 : REG PORT MAP(tmp\_cout(2), CLK, RSTn, tmp\_cin(3));  --- 4th Full Adder  reg\_A3\_0 : REG PORT MAP(A(3), CLK, RSTn, tmp\_A3(0));  reg\_B3\_0 : REG PORT MAP(B(3), CLK, RSTn, tmp\_B3(0));  reg\_A3\_1 : REG PORT MAP(tmp\_A3(0), CLK, RSTn, tmp\_A3(1));  reg\_B3\_1 : REG PORT MAP(tmp\_B3(0), CLK, RSTn, tmp\_B3(1));  reg\_A3\_2 : REG PORT MAP(tmp\_A3(1), CLK, RSTn, tmp\_A3(2));  reg\_B3\_2 : REG PORT MAP(tmp\_B3(1), CLK, RSTn, tmp\_B3(2));  FA\_3 : FA PORT MAP(tmp\_A3(2), tmp\_B3(2), tmp\_cin(3), 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, tmp\_S3(3));  reg\_s3\_3 : REG PORT MAP(tmp\_S3(3), CLK, RSTn, tmp\_S3(4));  reg\_s3\_4 : REG PORT MAP(tmp\_S3(4), CLK, RSTn, S(3));  reg\_cou3 : REG PORT MAP(tmp\_cout(3), CLK, RSTn, tmp\_cin(4));  --- 5th Full Adder  reg\_A4\_0 : REG PORT MAP(A(4), CLK, RSTn, tmp\_A4(0));  reg\_B4\_0 : REG PORT MAP(B(4), CLK, RSTn, tmp\_B4(0));  reg\_A4\_1 : REG PORT MAP(tmp\_A4(0), CLK, RSTn, tmp\_A4(1));  reg\_B4\_1 : REG PORT MAP(tmp\_B4(0), CLK, RSTn, tmp\_B4(1));  reg\_A4\_2 : REG PORT MAP(tmp\_A4(1), CLK, RSTn, tmp\_A4(2));  reg\_B4\_2 : REG PORT MAP(tmp\_B4(1), CLK, RSTn, tmp\_B4(2));  reg\_A4\_3 : REG PORT MAP(tmp\_A4(2), CLK, RSTn, tmp\_A4(3));  reg\_B4\_3 : REG PORT MAP(tmp\_B4(2), CLK, RSTn, tmp\_B4(3));  FA\_4 : FA PORT MAP(tmp\_A4(3), tmp\_B4(3), tmp\_cin(4), 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, tmp\_S4(2));  reg\_s4\_2 : REG PORT MAP(tmp\_S4(2), CLK, RSTn, tmp\_S4(3));  reg\_s4\_3 : REG PORT MAP(tmp\_S4(3), CLK, RSTn, S(4));  reg\_cou4 : REG PORT MAP(tmp\_cout(4), CLK, RSTn, tmp\_cin(5));  --- 6th Full Adder  reg\_A5\_0 : REG PORT MAP(A(5), CLK, RSTn, tmp\_A5(0));  reg\_B5\_0 : REG PORT MAP(B(5), CLK, RSTn, tmp\_B5(0));  reg\_A5\_1 : REG PORT MAP(tmp\_A5(0), CLK, RSTn, tmp\_A5(1));  reg\_B5\_1 : REG PORT MAP(tmp\_B5(0), CLK, RSTn, tmp\_B5(1));  reg\_A5\_2 : REG PORT MAP(tmp\_A5(1), CLK, RSTn, tmp\_A5(2));  reg\_B5\_2 : REG PORT MAP(tmp\_B5(1), CLK, RSTn, tmp\_B5(2));  reg\_A5\_3 : REG PORT MAP(tmp\_A5(2), CLK, RSTn, tmp\_A5(3));  reg\_B5\_3 : REG PORT MAP(tmp\_B5(2), CLK, RSTn, tmp\_B5(3));  reg\_A5\_4 : REG PORT MAP(tmp\_A5(3), CLK, RSTn, tmp\_A5(4));  reg\_B5\_4 : REG PORT MAP(tmp\_B5(3), CLK, RSTn, tmp\_B5(4));  FA\_5 : FA PORT MAP(tmp\_A5(4), tmp\_B5(4), tmp\_cin(5), 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, tmp\_S5(2));  reg\_s5\_2 : REG PORT MAP(tmp\_S5(2), CLK, RSTn, S(5));  reg\_cou5 : REG PORT MAP(tmp\_cout(5), CLK, RSTn, tmp\_cin(6));  --- 7th Full Adder  reg\_A6\_0 : REG PORT MAP(A(6), CLK, RSTn, tmp\_A6(0));  reg\_B6\_0 : REG PORT MAP(B(6), CLK, RSTn, tmp\_B6(0));  reg\_A6\_1 : REG PORT MAP(tmp\_A6(0), CLK, RSTn, tmp\_A6(1));  reg\_B6\_1 : REG PORT MAP(tmp\_B6(0), CLK, RSTn, tmp\_B6(1));  reg\_A6\_2 : REG PORT MAP(tmp\_A6(1), CLK, RSTn, tmp\_A6(2));  reg\_B6\_2 : REG PORT MAP(tmp\_B6(1), CLK, RSTn, tmp\_B6(2));  reg\_A6\_3 : REG PORT MAP(tmp\_A6(2), CLK, RSTn, tmp\_A6(3));  reg\_B6\_3 : REG PORT MAP(tmp\_B6(2), CLK, RSTn, tmp\_B6(3));  reg\_A6\_6 : REG PORT MAP(tmp\_A6(3), CLK, RSTn, tmp\_A6(4));  reg\_B6\_6 : REG PORT MAP(tmp\_B6(3), CLK, RSTn, tmp\_B6(4));  reg\_A6\_5 : REG PORT MAP(tmp\_A6(4), CLK, RSTn, tmp\_A6(5));  reg\_B6\_5 : REG PORT MAP(tmp\_B6(4), CLK, RSTn, tmp\_B6(5));  FA\_6 : FA PORT MAP(tmp\_A6(5), tmp\_B6(5), tmp\_cin(6), tmp\_S6(0), tmp\_cout(6));  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));  reg\_cou6 : REG PORT MAP(tmp\_cout(6), CLK, RSTn, tmp\_cin(7));  --- 8th Full Adder  reg\_A7\_0 : REG PORT MAP(A(7), CLK, RSTn, tmp\_A7(0));  reg\_B7\_0 : REG PORT MAP(B(7), CLK, RSTn, tmp\_B7(0));  reg\_A7\_1 : REG PORT MAP(tmp\_A7(0), CLK, RSTn, tmp\_A7(1));  reg\_B7\_1 : REG PORT MAP(tmp\_B7(0), CLK, RSTn, tmp\_B7(1));  reg\_A7\_2 : REG PORT MAP(tmp\_A7(1), CLK, RSTn, tmp\_A7(2));  reg\_B7\_2 : REG PORT MAP(tmp\_B7(1), CLK, RSTn, tmp\_B7(2));  reg\_A7\_3 : REG PORT MAP(tmp\_A7(2), CLK, RSTn, tmp\_A7(3));  reg\_B7\_3 : REG PORT MAP(tmp\_B7(2), CLK, RSTn, tmp\_B7(3));  reg\_A7\_4 : REG PORT MAP(tmp\_A7(3), CLK, RSTn, tmp\_A7(4));  reg\_B7\_4 : REG PORT MAP(tmp\_B7(3), CLK, RSTn, tmp\_B7(4));  reg\_A7\_5 : REG PORT MAP(tmp\_A7(4), CLK, RSTn, tmp\_A7(5));  reg\_B7\_5 : REG PORT MAP(tmp\_B7(4), CLK, RSTn, tmp\_B7(5));  reg\_A7\_6 : REG PORT MAP(tmp\_A7(5), CLK, RSTn, tmp\_A7(6));  reg\_B7\_6 : REG PORT MAP(tmp\_B7(5), CLK, RSTn, tmp\_B7(6));  FA\_7 : FA PORT MAP(tmp\_A7(6), tmp\_B7(6), tmp\_cin(7), tmp\_S7, tmp\_cout(7));  reg\_s7\_0 : REG PORT MAP(tmp\_S7, CLK, RSTn, S(7));  reg\_cou7 : REG PORT MAP(tmp\_cout(7), CLK, RSTn, Co);  S(N) <= Co;  END Behavioral; |

Code for pipelined\_adder.vhd

|  |
| --- |
| ----------------------------------------------------------------------------------  ---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 20ns  --pragma\_sythesis\_on  ;  Co <= (A AND B) OR (Ci AND A) OR (Ci AND B)  --pragma\_sythesis\_off  AFTER 25ns  --pragma\_sythesis\_on  ;  END ARCHITECTURE;  ----------------------------------------------------------------------------------  ---PIPELINED ADDER  LIBRARY IEEE;  USE IEEE.STD\_LOGIC\_1164.ALL;  ENTITY pipelined\_adder IS  GENERIC (N : INTEGER := 4); --4 bit pipelined adder is given here (change accordingly for 8 bit)  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  SIGNAL Co : STD\_LOGIC; --use this signal as your last carry out signal  SIGNAL tmp\_cout : STD\_LOGIC\_VECTOR(7 DOWNTO 0);  SIGNAL tmp\_cin : STD\_LOGIC\_VECTOR(7 DOWNTO 1);  SIGNAL tmp\_S0 : STD\_LOGIC\_VECTOR(7 DOWNTO 0);  SIGNAL tmp\_S1 : STD\_LOGIC\_VECTOR(6 DOWNTO 0);  SIGNAL tmp\_S2 : STD\_LOGIC\_VECTOR(5 DOWNTO 0);  SIGNAL tmp\_S3 : STD\_LOGIC\_VECTOR(4 DOWNTO 0);  SIGNAL tmp\_S4 : STD\_LOGIC\_VECTOR(3 DOWNTO 0);  SIGNAL tmp\_S5 : STD\_LOGIC\_VECTOR(2 DOWNTO 0);  SIGNAL tmp\_S6 : STD\_LOGIC\_VECTOR(1 DOWNTO 0);  SIGNAL tmp\_S7 : STD\_LOGIC;  SIGNAL tmp\_A1 : STD\_LOGIC;  SIGNAL tmp\_B1 : STD\_LOGIC;  SIGNAL tmp\_A2 : STD\_LOGIC\_VECTOR(1 DOWNTO 0);  SIGNAL tmp\_B2 : STD\_LOGIC\_VECTOR(1 DOWNTO 0);  SIGNAL tmp\_A3 : STD\_LOGIC\_VECTOR(2 DOWNTO 0);  SIGNAL tmp\_B3 : STD\_LOGIC\_VECTOR(2 DOWNTO 0);  SIGNAL tmp\_A4 : STD\_LOGIC\_VECTOR(3 DOWNTO 0);  SIGNAL tmp\_B4 : STD\_LOGIC\_VECTOR(3 DOWNTO 0);  SIGNAL tmp\_A5 : STD\_LOGIC\_VECTOR(4 DOWNTO 0);  SIGNAL tmp\_B5 : STD\_LOGIC\_VECTOR(4 DOWNTO 0);  SIGNAL tmp\_A6 : STD\_LOGIC\_VECTOR(5 DOWNTO 0);  SIGNAL tmp\_B6 : STD\_LOGIC\_VECTOR(5 DOWNTO 0);  SIGNAL tmp\_A7 : STD\_LOGIC\_VECTOR(6 DOWNTO 0);  SIGNAL tmp\_B7 : STD\_LOGIC\_VECTOR(6 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, tmp\_S0(3));  reg\_s0\_3 : REG PORT MAP(tmp\_S0(3), CLK, RSTn, S(0));  tmp\_cin(1) <= tmp\_cout(0);  --- 2nd Full Adder  FA\_1 : FA PORT MAP(A(1), B(1), tmp\_cin(1), 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, tmp\_S1(3));  reg\_s1\_3 : REG PORT MAP(tmp\_S1(3), CLK, RSTn, S(1));  reg\_cou1 : REG PORT MAP(tmp\_cout(1), CLK, RSTn, tmp\_cin(2));  ---3rd Full Adder  reg\_A2\_0 : REG PORT MAP(A(2), CLK, RSTn, tmp\_A2(0));  reg\_B2\_0 : REG PORT MAP(B(2), CLK, RSTn, tmp\_B2(0));  FA\_2 : FA PORT MAP(tmp\_A2(0), tmp\_B2(0), tmp\_cin(2), 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));  tmp\_cin(3) <= tmp\_cout(2);  --- 4th Full Adder  reg\_A3\_0 : REG PORT MAP(A(3), CLK, RSTn, tmp\_A3(0));  reg\_B3\_0 : REG PORT MAP(B(3), CLK, RSTn, tmp\_B3(0));  FA\_3 : FA PORT MAP(tmp\_A3(0), tmp\_B3(0), tmp\_cin(3), 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));  reg\_cou3 : REG PORT MAP(tmp\_cout(3), CLK, RSTn, tmp\_cin(4));  --- 5th Full Adder  reg\_A4\_0 : REG PORT MAP(A(4), CLK, RSTn, tmp\_A4(0));  reg\_B4\_0 : REG PORT MAP(B(4), CLK, RSTn, tmp\_B4(0));  reg\_A4\_1 : REG PORT MAP(tmp\_A4(0), CLK, RSTn, tmp\_A4(1));  reg\_B4\_1 : REG PORT MAP(tmp\_B4(0), CLK, RSTn, tmp\_B4(1));  FA\_4 : FA PORT MAP(tmp\_A4(1), tmp\_B4(1), tmp\_cin(4), 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));  tmp\_cin(5) <= tmp\_cout(4);  --- 6th Full Adder  reg\_A5\_0 : REG PORT MAP(A(5), CLK, RSTn, tmp\_A5(0));  reg\_B5\_0 : REG PORT MAP(B(5), CLK, RSTn, tmp\_B5(0));  reg\_A5\_1 : REG PORT MAP(tmp\_A5(0), CLK, RSTn, tmp\_A5(1));  reg\_B5\_1 : REG PORT MAP(tmp\_B5(0), CLK, RSTn, tmp\_B5(1));  FA\_5 : FA PORT MAP(tmp\_A5(1), tmp\_B5(1), tmp\_cin(5), 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));  reg\_cou5 : REG PORT MAP(tmp\_cout(5), CLK, RSTn, tmp\_cin(6));  --- 7th Full Adder  reg\_A6\_0 : REG PORT MAP(A(6), CLK, RSTn, tmp\_A6(0));  reg\_B6\_0 : REG PORT MAP(B(6), CLK, RSTn, tmp\_B6(0));  reg\_A6\_1 : REG PORT MAP(tmp\_A6(0), CLK, RSTn, tmp\_A6(1));  reg\_B6\_1 : REG PORT MAP(tmp\_B6(0), CLK, RSTn, tmp\_B6(1));  reg\_A6\_2 : REG PORT MAP(tmp\_A6(1), CLK, RSTn, tmp\_A6(2));  reg\_B6\_2 : REG PORT MAP(tmp\_B6(1), CLK, RSTn, tmp\_B6(2));  FA\_6 : FA PORT MAP(tmp\_A6(2), tmp\_B6(2), tmp\_cin(6), tmp\_S6(0), tmp\_cout(6));  reg\_s6\_0 : REG PORT MAP(tmp\_S6(0), CLK, RSTn, S(6));  tmp\_cin(7) <= tmp\_cout(6);  --- 8th Full Adder  reg\_A7\_0 : REG PORT MAP(A(7), CLK, RSTn, tmp\_A7(0));  reg\_B7\_0 : REG PORT MAP(B(7), CLK, RSTn, tmp\_B7(0));  reg\_A7\_1 : REG PORT MAP(tmp\_A7(0), CLK, RSTn, tmp\_A7(1));  reg\_B7\_1 : REG PORT MAP(tmp\_B7(0), CLK, RSTn, tmp\_B7(1));  reg\_A7\_2 : REG PORT MAP(tmp\_A7(1), CLK, RSTn, tmp\_A7(2));  reg\_B7\_2 : REG PORT MAP(tmp\_B7(1), CLK, RSTn, tmp\_B7(2));  FA\_7 : FA PORT MAP(tmp\_A7(2), tmp\_B7(2), tmp\_cin(7), tmp\_S7, tmp\_cout(7));  reg\_s7\_0 : REG PORT MAP(tmp\_S7, CLK, RSTn, S(7));  reg\_cou7 : REG PORT MAP(tmp\_cout(7), CLK, RSTn, Co);  S(N) <= Co;  END Behavioral; |

Code for pipelined\_adder\_half\_pipe.vhd

|  |
| --- |
| ----------------------------------------------------------------------------------  -- 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 := 8); --4 bit pipelined adder (change accordingly)  -- Port ( );  end tb\_pipelined\_adder;  architecture Behavioral of tb\_pipelined\_adder is  component pipelined\_adder is  generic(N :integer := 8); --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 := 61 ns; -- Clock Frequency  constant P: integer:= 8; -- number of pipeline stages  begin  UTT: pipelined\_adder generic map(N=>8) 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  ---------------------------4-bit ADDER INPUT Example-------------------------------  -- rst <= '1';  -- A <= "0000";  -- B <= "0000";  -- Ci<= '0';  -- wait for clk\_period;  -- rst <= '0';  -- A <= "1111";  -- B <= "1101";  -- Ci<= '0';  -- wait for (P)\*clk\_period;  -- A <= "1011";  -- B <= "1111";  -- Ci<= '1';  -- wait for (P)\*clk\_period;  -- A <= "1001";  -- B <= "1000";  -- Ci<= '0';  -- wait for (P)\*clk\_period;  -- A <= "0000";  -- B <= "1111";  -- Ci<= '1';  -- wait for (P)\*2\*clk\_period;  -- std.env.finish;  ---------------------------4-bit ADDER INPUT Example-------------------------------  ---------------------------8-bit ADDER INPUT Example-------------------------------  rst <= '1';  A <= "00000000"; --00  B <= "00000000"; --00  Ci<= '0';  wait for 1.5\*clk\_period;  rst <= '0';  A <= "00101001"; --41  B <= "01111111"; --127  Ci<= '0';  wait for (P)\*clk\_period;    rst <= '0';  A <= "00101001"; --41  B <= "01111111"; --127  Ci<= '1';  wait for (P)\*clk\_period;    rst <= '0';  A <= "00101001"; --41  B <= "11111101"; --253  Ci<= '0';  wait for (P)\*clk\_period;    rst <= '0';  A <= "00101001"; --41  B <= "11111101"; --253  Ci<= '1';  wait for (P)\*clk\_period;    rst <= '0';  A <= "11010101"; --213  B <= "11111101"; --253  Ci<= '0';  wait for (P)\*clk\_period;    rst <= '0';  A <= "11010101"; --213  B <= "11111101"; --253  Ci<= '1';  wait for (P)\*clk\_period;    rst <= '0';  A <= "11010101"; --213  B <= "01111111"; --127  Ci<= '0';  wait for (P)\*clk\_period;    rst <= '0';  A <= "11010101"; --213  B <= "01111111"; --127  Ci<= '1';  wait for (P)\*2\*clk\_period;  std.env.finish;  ---------------------------8-bit ADDER INPUT Example-------------------------------  end process;  end Behavioral; |

Code for tb\_pipelined\_adder.vhd