****

**EE4620L/EE6620L/CEG4324L/CEG6324L**

**DIGITAL INTEGRATED CIRCUIT DESIGN LAB**

**Lab 0**

By

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

UID: U00931845

Email: yeoh.3@wright.edu

Submission date: 20/05/2024

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

Signature : Alex Yeoh

Date : 19/05/2024

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

I. Introduction (page 4)

II. Process (page 4)

III. Results (page 7)

IV. Conclusion (page 7)

V. Code (page 7)

**List of Figures**

Figure 1: Logisim truth table before filling values. (Page 4)

Figure 2: Logisim truth table after filling. (Page 5)

Figure 3: Logisim auto-generated circuit for 2-bit full adder. (Page 6)

Figure 4: Logisim 2-bit full adder, with input limits. (Page 6)

Figure 5: Simulation for case 1. (Page 7)

Figure 6: Simulation for case 2. (Page 7)

Figure 7: Simulation for case 3. (Page 7)

I. Introduction

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

II. Process

We first determined the delays for the outputs of the 1-bit full adder by reading the behavioral VHDL for each of those expressions to determine what their critical paths are based on the delays for each gate as determined by the instructions in the lab instructions. We then determined the behavioral VHDL for a 2-bit full adder following the truth table given in the instructions of the lab instructions. To do this, I manually filled in the truth table in Logisim as seen below in figures 1 and 2 by changing all the “don’t care” values in the Logisim table to their respective output values.

A screenshot of a computer

Description automatically generated

Figure 1: Logisim truth table before filling values.

A screenshot of a computer

Description automatically generated

Figure 2: Logisim truth table after filling.

After filling the truth table, the k-map simplified sum of products Boolean equations are automatically generated in the expression tab. These equations can be visualized below in figure 3 by pressing the “build circuit” button.

A diagram of a circuit

Description automatically generated

Figure 3: Logisim auto-generated circuit for 2-bit full adder.

I then modified the autogenerated Boolean equations to follow the input limits for each gate. The changes can be seen reflected in the circuit seen below in figure 4.

A diagram of a circuit

Description automatically generated

Figure 4: Logisim 2-bit full adder, with input limits.

Following a similar process to determining the critical paths for the 1-bit full adder’s outputs, the critical paths for the 2-bit full adder’s outputs are determined. With the Boolean equations and critical path delays, the relevant lines in rca.vhd, rca\_\_w\_add2.vhd, and rca\_\_w\_add3.vhd are corrected with that information. The relevant ports in rca\_\_w\_add2.vhd and rca\_\_w\_add3.vhd for the 1-bit and 2-bit full adders are then mapped by name. With these VHDL files completed, we determined the WPD for each of the cases, ran the simulations and the 8 specific test cases as seen below in figure 5, 6, 7 for case 1, 2, and 3 respectively.

A screenshot of a computer

Description automatically generated

Figure 5: Simulation for case 1.

A screenshot of a computer

Description automatically generated

Figure 6: Simulation for case 2.

A screenshot of a computer

Description automatically generated

Figure 7: Simulation for case 3.

Finally, we demonstrated the code functioning on the ZedBoard.

III. Results

The process for determining the Boolean expressions for the 1- and 2-bit adders are stated in section 2, Process, above. The VHDL for the three cases can be seen below in section 5, Code, below. The waveforms that demonstrate the VHDL code working are shown in figures 5, 6, and 7 in section 2, Process, above. These results meet the requirements for the report in the lab instructions.

IV. Conclusion

From this lab, I have learned how to write combinational VHDL code.

V. Code

|  |
| --- |
| entity TB\_RCA is  generic (  N:integer:= 16; -- number of input bits for adder  WPD: time:= 210 ns -- Make necessary changes  );  end ; |

The portion of tb\_rca.vhd changed to modify WPD

|  |
| --- |
| -- rca.vhd  -- Ripple Carry Adder with full adder subcomponent  --------------------------------------------------------------------------  --------------------------------------------------------------------------  --------------------------------------------------------------------------  library IEEE,WORK;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  entity FA is  port( A,B,Ci: in std\_logic;  Co,S: out std\_logic);  end;  architecture FA\_BEHAV of FA is  begin  Co <= ((A and B) or (A and Ci)) or (B and Ci)  -- synthesis\_off  after 9 ns  -- synthesis\_on  ;  S <= (A and B and Ci) or (A and not B and not Ci) or (not A and B and not Ci) or (not A and not B and Ci)  -- synthesis\_off  after 13 ns  -- synthesis\_on  ;  end;  --------------------------------------------------------------------------  --------------------------------------------------------------------------  --------------------------------------------------------------------------  library IEEE,WORK;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  entity RCA is  generic(N:integer:=16);  port( A,B: in std\_logic\_vector(N-1 downto 0);  Ci: in std\_logic;  Co: out std\_logic;  S: out std\_logic\_vector(N-1 downto 0));  end;  architecture RCA\_STRUCT of RCA is  -- declarative area  component FA  port(A,B,Ci:in std\_logic;Co,S:out std\_logic);  end component;  signal C:std\_logic\_vector(N downto 0);  begin  -- instantiation area  C(0) <= Ci;  GI: for I in 0 to N-1 generate  GI:FA port map(A => A(I),B => B(I), Ci => C(I),Co => C(I+1),S => S(I));  end generate;  Co <= C(N);  end; |

rca.vhd

|  |
| --- |
| -- rca\_\_w\_add2.vhd  -- ripple carry adder with 1-bit full adder and 2-bit full adder subcomponent  -- case 2  --------------------------------------------------------------------------  --------------------------------------------------------------------------  --------------------------------------------------------------------------  library IEEE,WORK;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  entity FA is  port( A,B,Ci: in std\_logic;  Co,S: out std\_logic);  end;  architecture FA\_BEHAV of FA is  begin  Co <= ((A and B) or (A and Ci)) or (B and Ci)  -- synthesis\_off  after 9 ns  -- synthesis\_on  ;  S <= (A and B and Ci) or (A and not B and not Ci) or (not A and B and not Ci) or (not A and not B and Ci)  -- synthesis\_off  after 13 ns  -- synthesis\_on  ;  end;  --------------------------------------------------------------------------  --------------------------------------------------------------------------  --------------------------------------------------------------------------  library IEEE,WORK;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  -- ADD2 is a two-level logic (ignoring inverters) circuit that adds two, 2-bit numbers  -- you must design this block  entity ADD2 is  generic(N:integer:=2);  port( A,B: in std\_logic\_vector(N-1 downto 0);  Ci: in std\_logic;  Co: out std\_logic;  S: out std\_logic\_vector(N-1 downto 0));  end;  architecture ADD2\_BEHAV of ADD2 is  begin  Co <= (A(0) and B(1) and B(0)) or (A(1) and B(1)) or (A(1) and A(0) and B(0)) or (Ci and B(1) and B(0)) or (Ci and A(0) and B(1)) or (Ci and A(1) and B(0)) or (Ci and A(1) and A(0))  -- pragma synthesis\_off  after 15 ns  -- pragma synthesis\_on  ;  S(0) <= (not Ci and not A(0) and B(0)) or (not Ci and A(0) and not B(0)) or (Ci and not A(0) and not B(0)) or (Ci and A(0) and B(0))  -- pragma synthesis\_off  after 13 ns  -- pragma synthesis\_on  ;  S(1) <= ((not Ci and not A(1) and not A(0) and B(1)) or (not Ci and not A(1) and B(1) and not B(0)) or (not A(1) and not A(0) and B(1) and not B(0)) or (not A(1) and A(0) and not B(1) and B(0))) or ((not Ci and A(1) and not A(0) and not B(1)) or (not Ci and A(1) and not B(1) and not B(0)) or (A(1) and not A(0) and not B(1) and not B(0)) or (A(1) and A(0) and B(1) and B(0))) or ((Ci and not A(1) and not B(1) and B(0)) or (Ci and not A(1) and A(0) and not B(1)) or (Ci and A(1) and B(1) and B(0)) or (Ci and A(1) and A(0) and B(1)))  -- pragma synthesis\_off  after 19 ns  -- pragma synthesis\_on  ;  end;  --------------------------------------------------------------------------  --------------------------------------------------------------------------  --------------------------------------------------------------------------  library IEEE,WORK;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  entity RCA is  generic(N:integer:=16);  port( A,B: in std\_logic\_vector(N-1 downto 0);  Ci: in std\_logic;  Co: out std\_logic;  S: out std\_logic\_vector(N-1 downto 0));  end;  architecture RCA\_STRUCT of RCA is  -- declarative area  component FA  port(A,B,Ci:in std\_logic;Co,S:out std\_logic);  end component;  component ADD2  generic(N:integer:=2);  port(A,B:in std\_logic\_vector(N-1 downto 0);Ci:in std\_logic;Co:out std\_logic;S:out std\_logic\_vector(N-1 downto 0));  end component;  signal C:std\_logic\_vector(N-4 downto 0);  begin  -- it helps to draw this out and label the signal lines  -- instantiation area  C(0) <= Ci;  GI: for I in 0 to N/2-1 generate  GI:FA port map(A => A(I),B => B(I), Ci => C(I),Co => C(I+1),S => S(I));  end generate;  GJ: for I in N/4 to N/2-1 generate  GI:ADD2  generic map(N=>2)  port map(A(1)=>A(I\*2+1)  ,A(0)=>A(I\*2)  ,B(1)=>B(I\*2+1)  ,B(0)=>B(I\*2)  ,Ci=>C(N/4+I)  ,Co=>C(N/4+I+1)  ,S(1)=>S(I\*2+1),  S(0)=>S(I\*2));  end generate;  Co <= C(N/2+N/4);  end; |

rca\_\_w\_add2.vhd

|  |
| --- |
| -- rca\_\_w\_add3.vhd  -- ripple carry adder with 2-bit adder (ADD2) subcomponent  -- case3  --------------------------------------------------------------------------  --------------------------------------------------------------------------  --------------------------------------------------------------------------  library IEEE,WORK;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  -- ADD2 is a two-level logic (ignoring inverters) circuit that adds two, 2-bit numbers  -- you must design this block  entity ADD2 is  generic(N:integer:=2);  port( A,B: in std\_logic\_vector(N-1 downto 0);  Ci: in std\_logic;  Co: out std\_logic;  S: out std\_logic\_vector(N-1 downto 0));  end;  architecture ADD2\_BEHAV of ADD2 is  begin  Co <= (A(0) and B(1) and B(0)) or (A(1) and B(1)) or (A(1) and A(0) and B(0)) or (Ci and B(1) and B(0)) or (Ci and A(0) and B(1)) or (Ci and A(1) and B(0)) or (Ci and A(1) and A(0))  -- pragma synthesis\_off  after 15 ns  -- pragma synthesis\_on  ;  S(0) <= (not Ci and not A(0) and B(0)) or (not Ci and A(0) and not B(0)) or (Ci and not A(0) and not B(0)) or (Ci and A(0) and B(0))  -- pragma synthesis\_off  after 13 ns  -- pragma synthesis\_on  ;  S(1) <= ((not Ci and not A(1) and not A(0) and B(1)) or (not Ci and not A(1) and B(1) and not B(0)) or (not A(1) and not A(0) and B(1) and not B(0)) or (not A(1) and A(0) and not B(1) and B(0))) or ((not Ci and A(1) and not A(0) and not B(1)) or (not Ci and A(1) and not B(1) and not B(0)) or (A(1) and not A(0) and not B(1) and not B(0)) or (A(1) and A(0) and B(1) and B(0))) or ((Ci and not A(1) and not B(1) and B(0)) or (Ci and not A(1) and A(0) and not B(1)) or (Ci and A(1) and B(1) and B(0)) or (Ci and A(1) and A(0) and B(1)))  -- pragma synthesis\_off  after 19 ns  -- pragma synthesis\_on  ;  end;  --------------------------------------------------------------------------  --------------------------------------------------------------------------  --------------------------------------------------------------------------  library IEEE,WORK;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  entity RCA is  generic(N:integer:=16);  port( A,B: in std\_logic\_vector(N-1 downto 0);  Ci: in std\_logic;  Co: out std\_logic;  S: out std\_logic\_vector(N-1 downto 0));  end;  architecture RCA\_STRUCT of RCA is  -- declarative area  component ADD2  generic(N:integer:=2);  port(A,B:in std\_logic\_vector(N-1 downto 0);Ci:in std\_logic;Co:out std\_logic;S:out std\_logic\_vector(N-1 downto 0));  end component;  signal C:std\_logic\_vector(N/2 downto 0);  begin  -- it helps to draw this out and label the signal lines  -- instantiation area  C(0) <= Ci;  GI: for I in 0 to N/2-1 generate  GI:ADD2  generic map(N=>2)  port map(A(1)=>A(I\*2+1)  ,A(0)=>A(I\*2)  ,B(1)=>B(I\*2+1)  ,B(0)=>B(I\*2)  ,Ci=>C(I)  ,Co=>C(I+1)  ,S(1)=>S(I\*2+1),  S(0)=>S(I\*2));  end generate;  Co <= C(N/2);  end; |

rca\_\_w\_add3.vhd