****

**EE4620L/EE6620L/CEG4324L/CEG6324L**

**DIGITAL INTEGRATED CIRCUIT DESIGN LAB**

**Lab 3**

By

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

UID: U00931845

Email: yeoh.3@wright.edu

Submission date: 01/06/2024

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

Signature : Alex Yeoh

Date : 31/05/2024

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

I. Introduction (page 4)

II. Process (page 4)

III. Results (page 4)

IV. Conclusion (page 8)

V. Code (page 9)

**List of Figures**

Figure 1: booth multiplier stage design in Logisim. (page 4)

Figure 2: booth multiplier simulation with the inputs and outputs in signed decimal. (page 7)

Figure 3: booth multiplier simulation with the inputs and outputs in hexadecimal. (page 7)

Figure 4: booth multiplier simulation with the inputs and outputs in binary. (page 7)

Figure 5: area, power, and delay analysis for the booth multiplier (page 7)

Figure 6: schematic for the booth multiplier (page 8)

Figure 7: Zedboard with the booth multiplier performing 7 \* 21. (page 8)

I. Introduction

The objective of the lab is to become more familiar with Vivado by building and simulating a booth multiplier.

II. Process

I first designed a booth multiplier in logisim to determine the required logical components with the design shown below in figure 1.

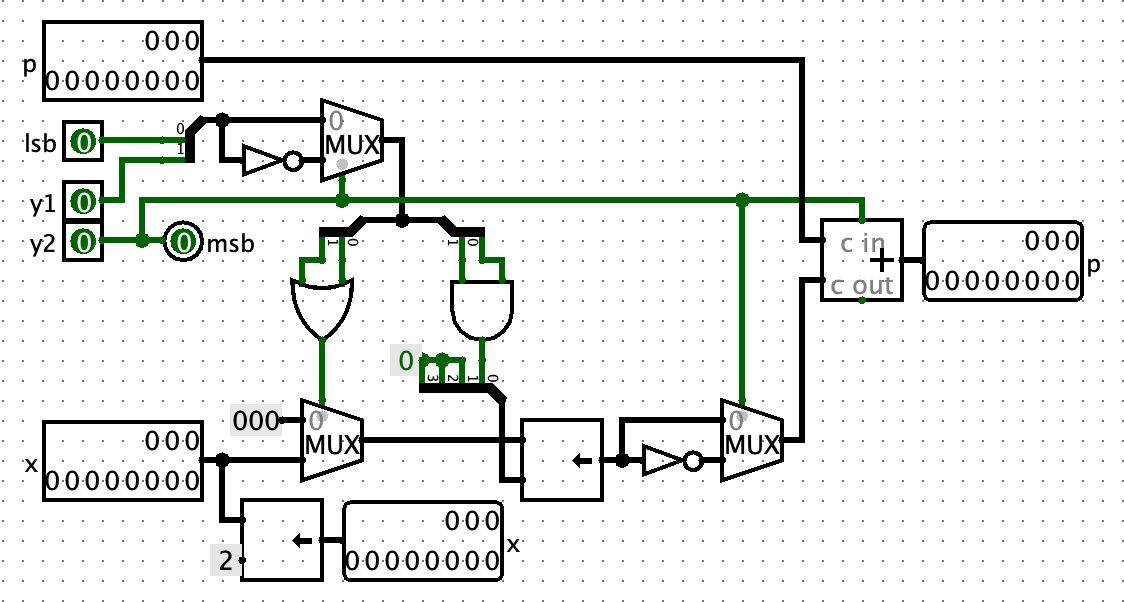


Figure 1: booth multiplier stage design in Logisim.

I then made the Vivado project and imported all the given files. I then replicated the booth multiplier design in booth.vhd. I then fixed tb\_booth.vhd to test the four test cases. Finally, I demonstrated the booth multiplier design working on the Zedboard.

III. Results

The following are the results for the booth multiplier.

The hand computed results for each of the four cases.

|  |
| --- |
| 7 \* 21  a: 00000111  b: 00010101  x: 000000000010101  p: 000000000000000  append 0 to a  a: 000001110  First operation 110: p = p + -x  000000000000000  + 111111111101011  p: 111111111101011  x: 000000001010100  Second operation 011: p = p + 2x  111111111101011  + 000000010101000  p: 000000010010011  x: 000000101010000  Third operation 000: p = p + 0  000000010010011  + 000000000000000  p: 000000010010011  x: 000010101000000  Forth operation 000: p = p + 0  000000010010011  + 000000000000000  p: 000000010010011  x: 001010100000000  Output: 000000010010011 = 147 |
| 7 \* -31  a: 00000111  b: 11100001  x: 111111111100001  p: 000000000000000  append 0 to a  a: 000001110  First operation 110: p = p + -x  000000000000000  + 000000000011111  p: 000000000011111  x: 111111110000100  Second operation 011: p = p + 2x  000000000011111  + 111111100001000  p: 111111100100111  x: 111111000010000  Third operation 000: p = p + 0  111111100100111  + 000000000000000  p: 111111100100111  x: 111100001000000  Forth operation 000: p = p + 0  111111100100111  + 000000000000000  p: 111111100100111  x: 110000100000000  Output: 111111100100111 = -217 |
| -31 \* -31  a: 11100001  b: 11100001  x: 111111111100001  p: 000000000000000  append 0 to a  a: 111000010  First operation 010: p = p + x  000000000000000  + 111111111100001  p: 111111111100001  x: 111111110000100  Second operation 000: p = p + 0  111111111100001  + 000000000000000  p: 111111111100001  x: 111111000010000  Third operation 100: p = p + -2x  111111111100001  + 000001111100000  p: 000001111000001  x: 111100001000000  Forth operation 111: p = p + 0  000001111000001  + 000000000000000  p: 000001111000001  x: 110000100000000  Output: 000001111000001 = 961 |
| -31 \* 21  a: 11100001  b: 00010101  x: 000000000010101  p: 000000000000000  append 0 to a  a: 111000010  First operation 010: p = p + x  000000000000000  + 000000000010101  p: 000000000010101  x: 000000001010100  Second operation 000: p = p + 0  000000000010101  + 000000000000000  p: 000000000010101  x: 000000101010000  Third operation 100: p = p + -2x  000000000010101  + 111110101100000  p: 111110101110101  x: 000010101000000  Forth operation 111: p = p + 0  111110101110101  + 000000000000000  p: 111110101110101  x: 0010101000000  Output: 111110101110101 = -651 |

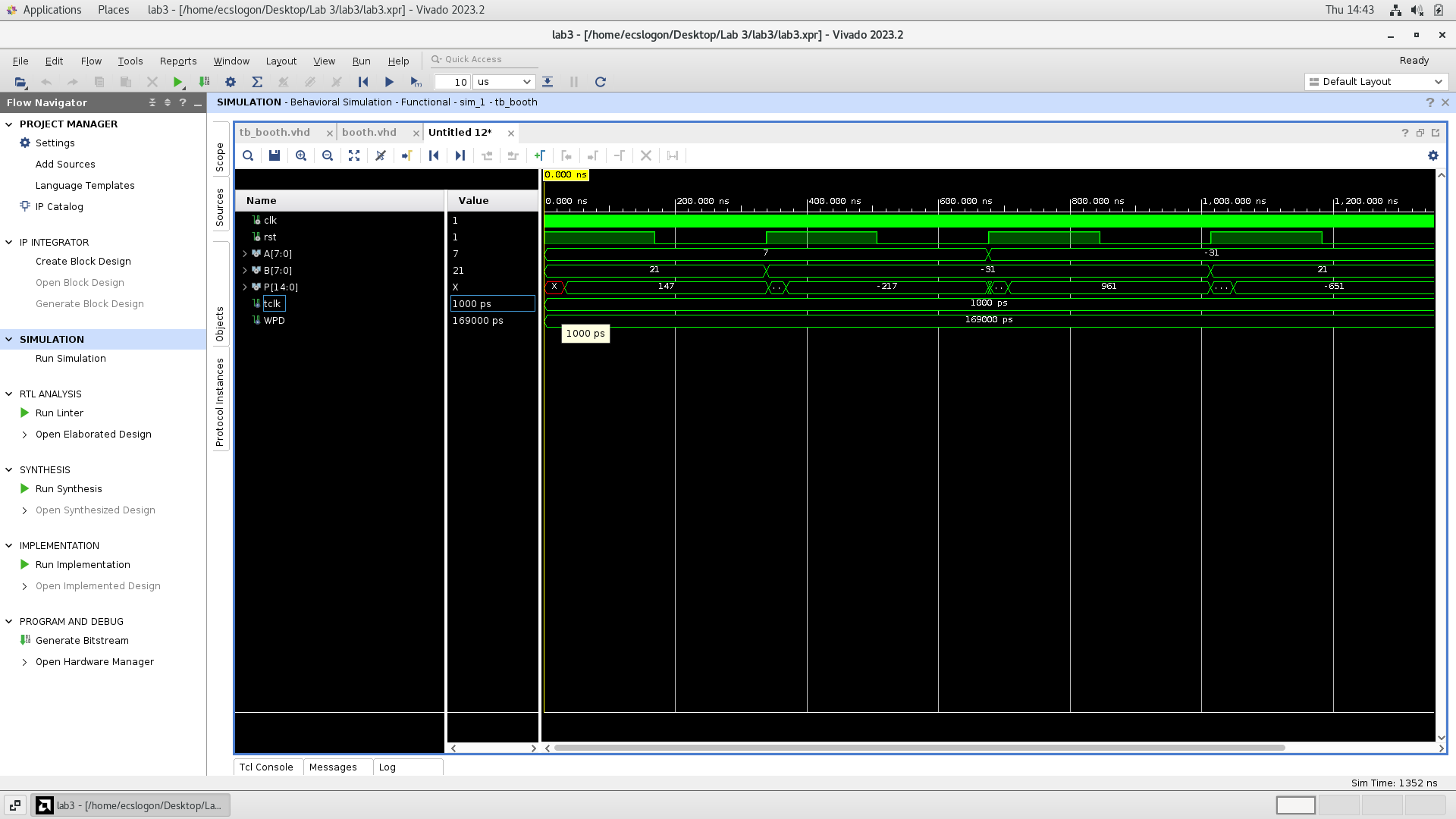


Figure 2: booth multiplier simulation with the inputs and outputs in signed decimal.

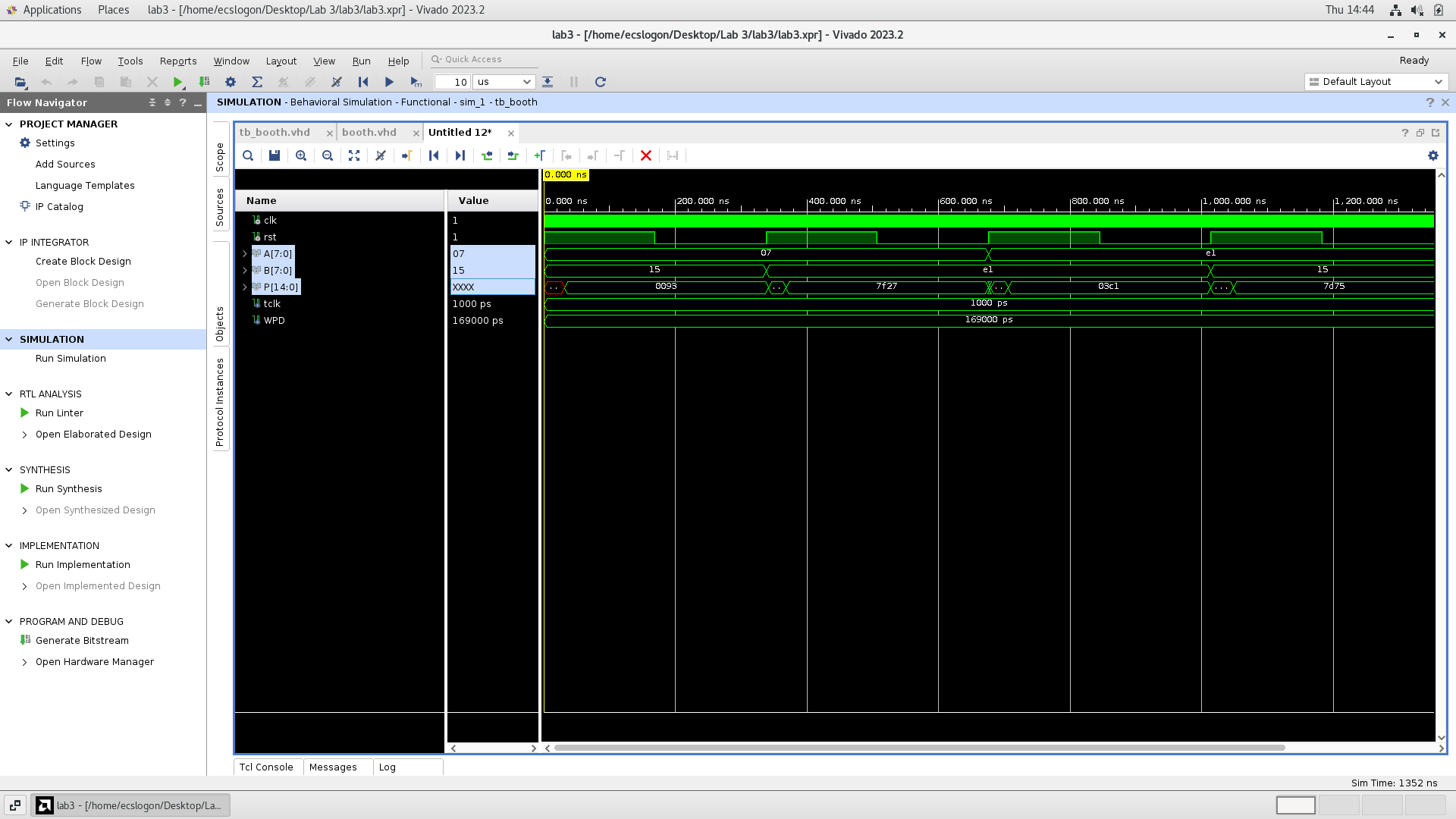


Figure 3: booth multiplier simulation with the inputs and outputs in hexadecimal.

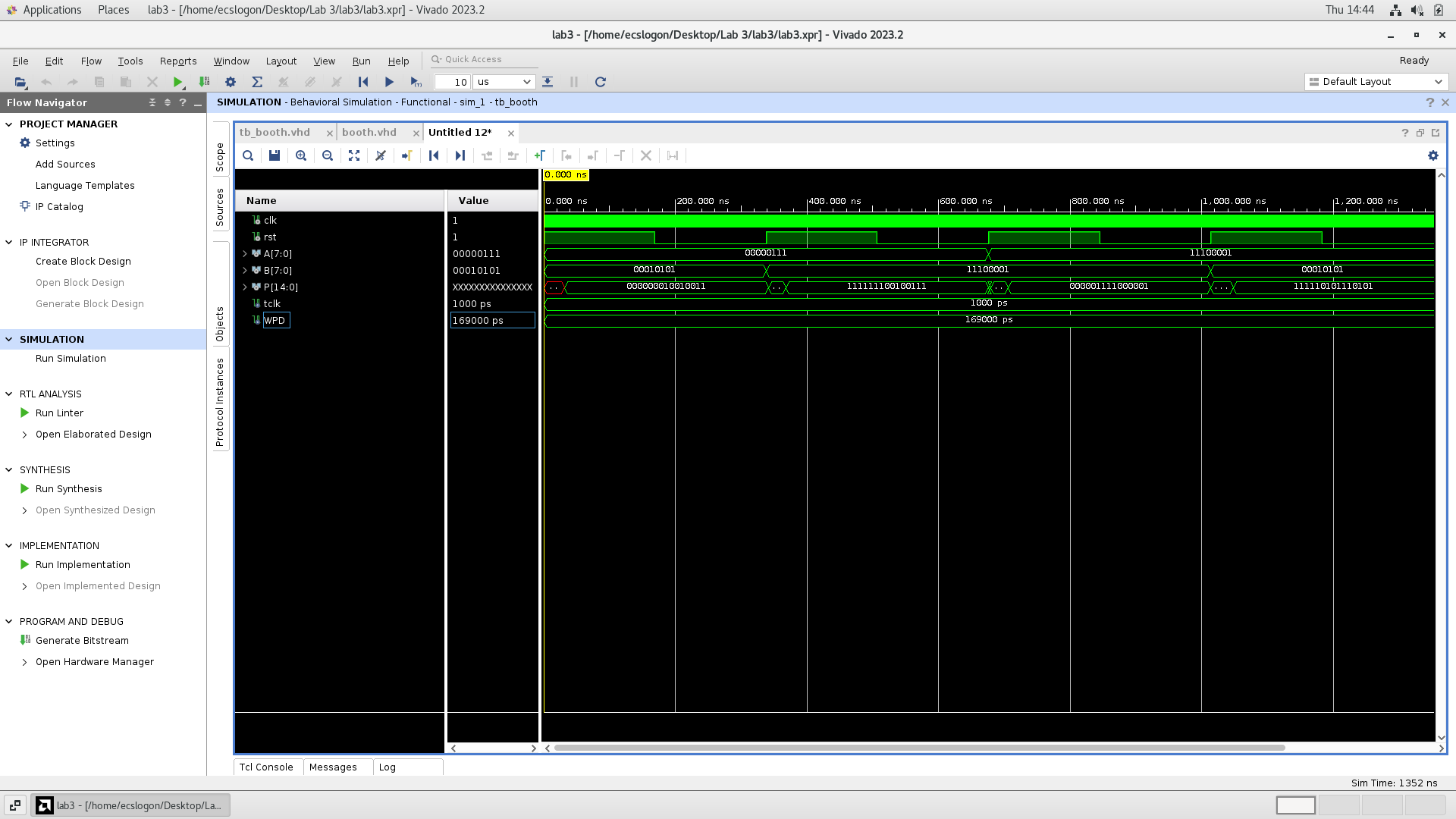


Figure 4: booth multiplier simulation with the inputs and outputs in binary.

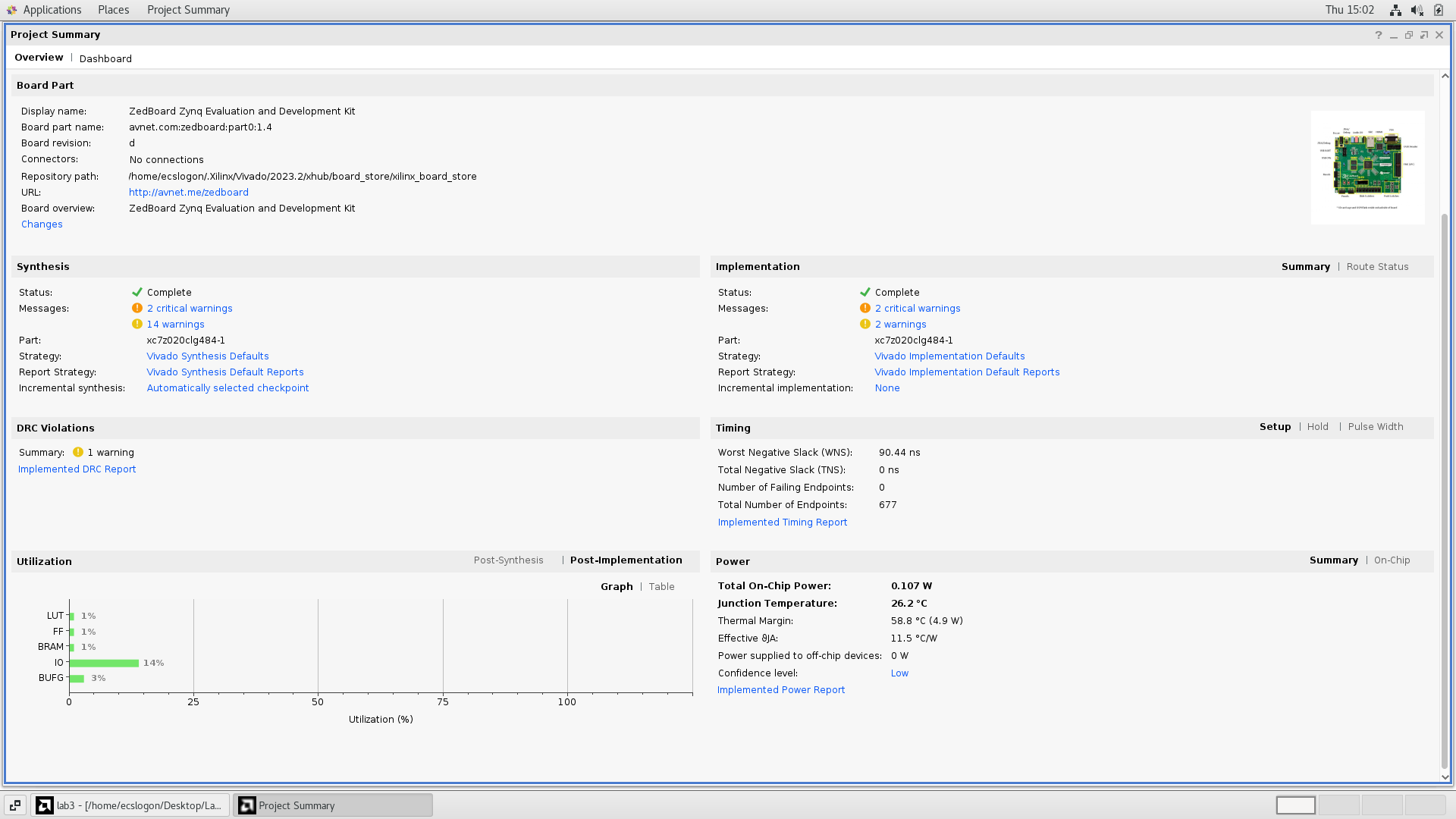


Figure 5: area, power, and delay analysis for the booth multiplier

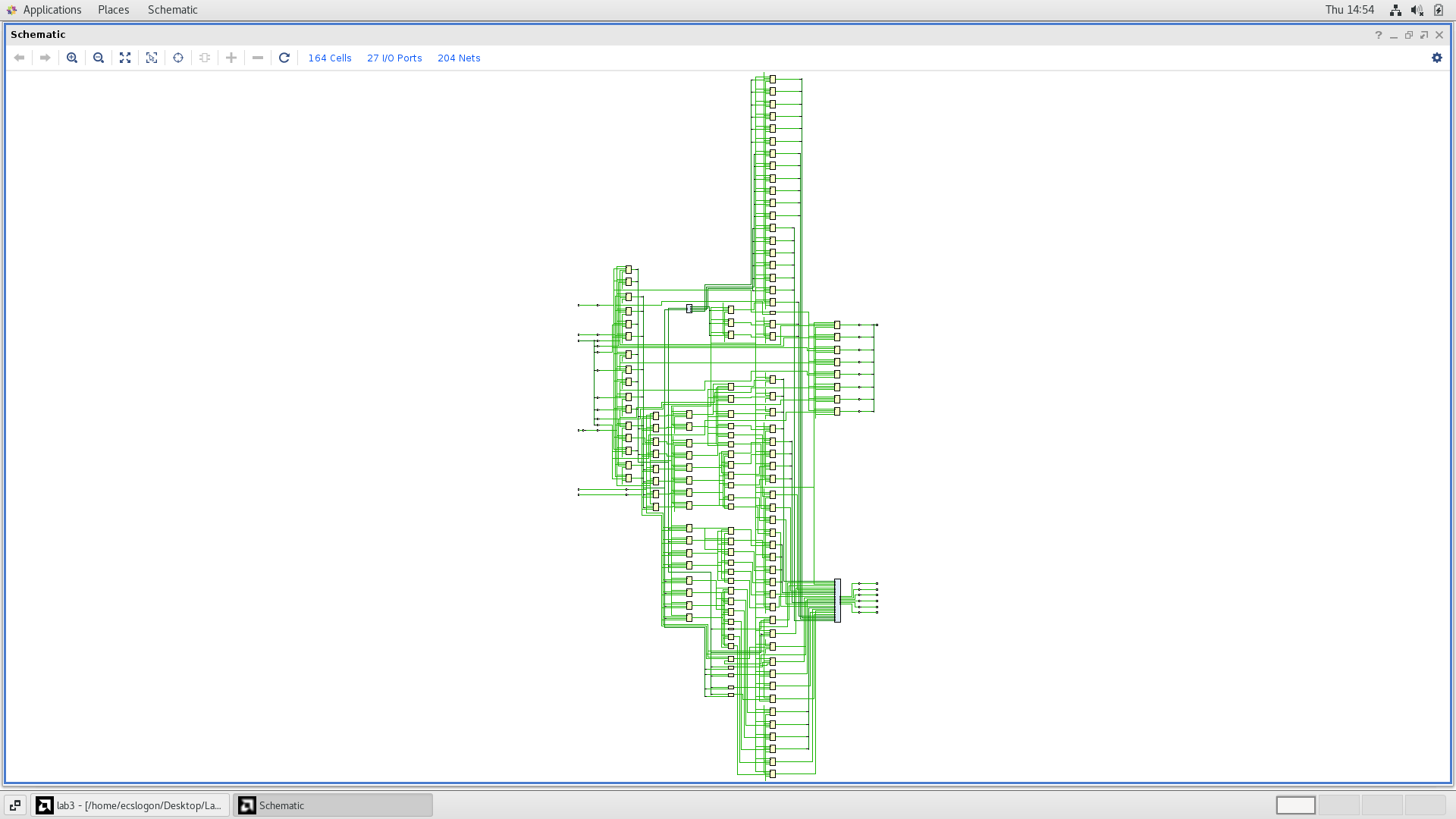


Figure 6: schematic for the booth multiplier

A green electronic board with many small buttons and a black wire

Description automatically generated with medium confidence

Figure 7: Zedboard with the booth multiplier performing 7 \* 15.

IV. Conclusion

From this lab, I have learned how to write more complicated VHDL code such as a booth multiplier.

V. Code

|  |
| --- |
| -- Booth Multiplier  -- Define libraries  library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity booth is  port (operand\_a : in std\_logic\_vector (7 downto 0);  operand\_b : in std\_logic\_vector (7 downto 0);  rst : in std\_logic;  clk : in std\_logic;  z : out std\_logic\_vector (14 downto 0));  end booth;  library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity stage is  port (y : in std\_logic\_vector (2 downto 0);  xin, pin: in std\_logic\_vector (14 downto 0);  overlap: out std\_logic;  xout, pout: out std\_logic\_vector (14 downto 0));  end stage;  architecture name of stage is  signal op:std\_logic\_vector(1 downto 0);  signal localx,localx1,localx2:std\_logic\_vector(14 downto 0);    begin  -- overlap bit so booth doesn't need to take from the overall instruction  overlap <= y(2);    -- left shift 2 for next stage  xout <= xin(12 downto 0) & "00";    -- making signal for choosing 0, x, or 2x  op(0) <= y(0) when (y(2)='0')else not y(0) after 2 ns; --1 inverter  op(1) <= y(1) when (y(2)='0')else not y(1) after 2 ns; --1 inverter    -- deciding between 0 or x  localx <= xin when ((op(0) or op(1))='1') else "000000000000000" after 4 ns; --2in or gate    -- multiplying by 2 by bit shifting, only relevant when previous step decided x  localx1 <= localx when ((op(0) and op(1))='0') else localx(13 downto 0) & '0' after 4 ns; --2in and gate    -- effectively xor localx to make adder into an adder/subtractor  localx2 <= localx1 when (y(2)='0')else not localx1 after 30 ns; --15 inverters    -- adder  pout <= pin + localx2 + y(2);  end;  architecture rtl of booth is  --Declare Signals  signal x\_ext, xout0, xout1, xout2, pout0, pout1, pout2: std\_logic\_vector (14 downto 0);  signal overlap: std\_logic\_vector (2 downto 0);  --Declare Components  component stage is  port (y : in std\_logic\_vector (2 downto 0);  xin, pin: in std\_logic\_vector (14 downto 0);  overlap: out std\_logic;  xout, pout: out std\_logic\_vector (14 downto 0));  end component;  begin  x\_ext <= operand\_b(7) & operand\_b(7) & operand\_b(7) & operand\_b(7) & operand\_b(7) & operand\_b(7) & operand\_b(7) & operand\_b;  stg1: stage port map (y(0) => '0', y(1) => operand\_a(0), y(2) => operand\_a(1), xin => x\_ext, pin => "000000000000000",  overlap => overlap(0), xout => xout0, pout => pout0);    stg2: stage port map (y(0) => overlap(0), y(1) => operand\_a(2), y(2) => operand\_a(3), xin => xout0, pin => pout0,  overlap => overlap(1), xout => xout1, pout => pout1);    stg3: stage port map (y(0) => overlap(1), y(1) => operand\_a(4), y(2) => operand\_a(5), xin => xout1, pin => pout1,  overlap => overlap(2), xout => xout2, pout => pout2);    stg4: stage port map (y(0) => overlap(2), y(1) => operand\_a(5), y(2) => operand\_a(7), xin => xout2, pin => pout2,  pout => z);  end; |

booth.vhd

|  |
| --- |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use std.env.stop;  entity tb\_booth is  generic(tclk:time := 1 ns; -- clock speed  WPD: time := 169 ns -- rate at which the input changes  );    end tb\_booth;  architecture Behavioral of tb\_booth is  signal clk : std\_logic;  signal rst : std\_logic;  signal A : std\_logic\_vector (7 downto 0);  signal B : std\_logic\_vector (7 downto 0);  signal P : std\_logic\_vector (14 downto 0);  component booth is  port (operand\_a : in std\_logic\_vector (7 downto 0);  operand\_b : in std\_logic\_vector (7 downto 0);  rst : in std\_logic;  clk : in std\_logic;  z : out std\_logic\_vector (14 downto 0));  end component;  begin  clock:process  begin  clk <= '1'; wait for tclk / 2;  clk <= '0'; wait for tclk / 2;  end process;    DUT: booth port map(clk=>clk,rst=>rst,operand\_a=>A,operand\_b=>B,z=>P);  test\_vectors:process  begin  A<="00000111"; --(7)  B<="00010101"; --(21)  rst<='1';  wait for WPD;  rst<='0';  wait for WPD;    A<="00000111"; --(7)  B<="11100001"; --(-31)  rst<='1';  wait for WPD;  rst<='0';  wait for WPD;    A<="11100001"; --(-31)  B<="11100001"; --(-31)  rst<='1';  wait for WPD;  rst<='0';  wait for WPD;    A<="11100001"; --(-31)  B<="00010101"; --(21)  rst<='1';  wait for WPD;  rst<='0';  wait for WPD;    stop;  end process;    end Behavioral; |

tb\_booth.vhd