Logo

Description automatically generated**EEDG/CE 6370**

**Due Date**: Sunday October 1, 11:59pm

16

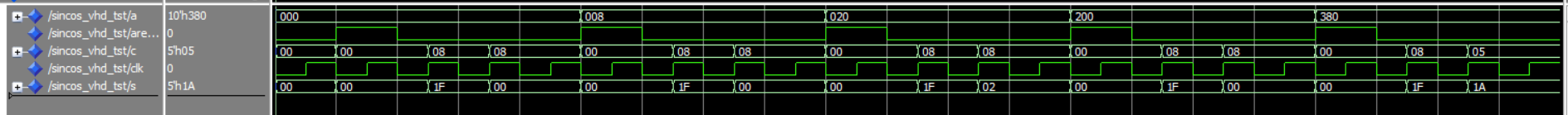
**Design and Analysis of Reconfigurable Systems**

**Homework 3 – IP generator**

**Students Name: Khoa Diep, Dismas Ezechukwu**

**Part I – CORDIC IP**

a.) Follow the instructions in the lab sheet. Show the simulation result of the CORDIC IP clearly indicating that the circuit works



|  |
| --- |
| Marks |
| 4 |
|  |

b.) Report the number of ALMs and critical path of the design from the synthesis report. Compute the maximum frequency.

Fmax: 160.9 MHz

Critical Path: 6.22 ns

ALM: 150

|  |
| --- |
| Marks |
| 2 |
|  |

c.) Create a short video showing that the system is successfully prototyped on the DE1-SoC Board

|  |
| --- |
| Marks |
| 2 |
|  |

[**https://youtu.be/ODb07HkTAW4**](https://youtu.be/ODb07HkTAW4)

**Part II – CORDIC IP Output Modification (7-segment display)**

1. Modify the program such that the value of the sin and cosine are output on the 7-segment display instead of the LEDs. As shown below the first three 7-segment displays should show the value of the cosine, while the last three the value of the sine, where the first 7-segment display should show the sign, the second the integer value and the last the fractional part.

<https://youtu.be/JvS99K9nJ_I>

sine

cosine



sign integer frac

sign integer frac

Include here your code and a short video showing that it works. Report also the number of resources used and the critical path

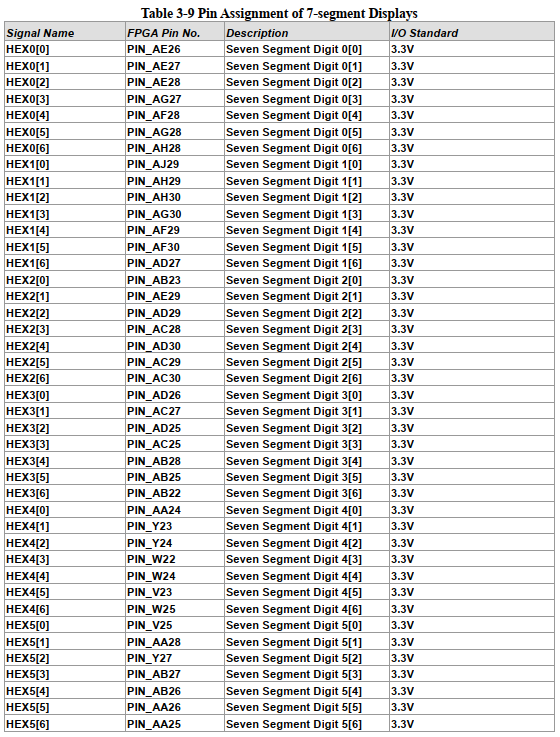
Fmax: 153.12 Mhz

Critical path: 6.53 ns

ALMs: 157



|  |
| --- |
| Marks |
| 8 |
|  |



Sincos.vhd (added on and edited the original file)

-- sincos.vhd

-- Generated using ACDS version 22.1 922

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

use ieee.std\_logic\_unsigned.all;

entity sincos is

port (

a : in std\_logic\_vector(9 downto 0) := (others => '0'); -- a.a

areset : in std\_logic := '0'; -- areset.reset

c : out std\_logic\_vector(4 downto 0); -- c.c

clk : in std\_logic := '0'; -- clk.clk

s : out std\_logic\_vector(4 downto 0) ; -- s.s

c\_frac : out std\_logic\_vector(6 downto 0) := "0000000"; -- cosine fractional part

c\_int : out std\_logic\_vector(6 downto 0) := "0000000"; -- cosine integer part

c\_sign : out std\_logic\_vector(6 downto 0) := "0000000"; -- cosine signed part

s\_frac : out std\_logic\_vector(6 downto 0) := "0000000"; -- sine fractional part

s\_int : out std\_logic\_vector(6 downto 0) := "0000000"; -- sine integer part

s\_sign : out std\_logic\_vector(6 downto 0) := "0000000"-- sine signed part

);

end entity sincos;

architecture rtl of sincos is

component sincos\_CORDIC\_0 is

port (

clk : in std\_logic := 'X'; -- clk

areset : in std\_logic := 'X'; -- reset

a : in std\_logic\_vector(9 downto 0) := (others => 'X'); -- a

c : out std\_logic\_vector(4 downto 0); -- c

s : out std\_logic\_vector(4 downto 0) -- s

);

end component sincos\_CORDIC\_0;

-- Holds absolute value of the binary bits

signal c\_2, s\_2 : std\_logic\_vector(4 downto 0) := "00000";

-- Holds signed part of sin and cos

signal c\_s, s\_s : std\_logic;

-- Holds integer value of sin and cos

signal c\_i, s\_i : std\_logic;

-- Holds fractional value of sin and cos

signal c\_f, s\_f : std\_logic\_vector(2 downto 0) := "000";

begin

cordic\_0 : component sincos\_CORDIC\_0

port map (

clk => clk, -- clk.clk

areset => areset, -- areset.reset

a => a, -- a.a

c => c, -- c.c

s => s -- s.s

);

process(clk, areset, a)

BEGIN

--Signs

c\_s <= c(4);

s\_s <= s(4);

-- Signed Logic

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

--Checking sign of cosine

if (c\_s = '1') then

c\_sign <= "0111111"; -- make minus(-) sign

-- two's complement ONLY if sign is negative (msb is equal to one), otherwise use original value

c\_2 <= not (c) + "00001";

else

c\_sign <= "1111111"; -- (all off) represent positive

c\_2 <= c;

end if;

--Checking sign of sine

if (s\_s = '1') then

s\_sign <= "0111111"; -- make minus(-) sign

s\_2 <= not (s) + "00001";

else

s\_sign <= "1111111"; -- (all off) represent positive

s\_2 <= s;

end if;

-- Integer Logic

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

c\_i <= c\_2(3);

s\_i <= s\_2(3);

-- Int of cosine

if (c\_i = '1') then

c\_int <= "1111001"; -- make 1

else

c\_int <= "1000000"; -- make 0

end if;

-- Int of sine

if (s\_i = '1') then

s\_int <= "1111001"; -- make 1

else

s\_int <= "1000000"; -- make 0

end if;

-- Fraction Logic

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

c\_f <= c\_2(2 downto 0);

s\_f <= s\_2(2 downto 0);

-- Hardcode these value

case c\_f is

when "000" => c\_frac <= "1111111"; -- .000 rounds to .0 (all off)

when "001" => c\_frac <= "1111001"; -- .125 rounds to .1 (1)

when "010" => c\_frac <= "0110000"; -- .25 rounds to .3 (3)

when "011" => c\_frac <= "0011001"; -- .375 rounds to .4

when "100" => c\_frac <= "0010010"; -- .5 rounds to .5

when "101" => c\_frac <= "1111000"; -- .675 rounds to .7

when "110" => c\_frac <= "0000000"; -- .75 rounds to .8

when "111" => c\_frac <= "0010000"; -- .875 rounds to .9

when others => c\_frac <="1111111"; -- all off

end case;

case s\_f is

when "000" => s\_frac <= "1111111"; -- .000 rounds to .0 (all off)

when "001" => s\_frac <= "1111001"; -- .125 rounds to .1 (1)

when "010" => s\_frac <= "0110000"; -- .25 rounds to .3 (3)

when "011" => s\_frac <= "0011001"; -- .375 rounds to .4

when "100" => s\_frac <= "0010010"; -- .5 rounds to .5

when "101" => s\_frac <= "1111000"; -- .675 rounds to .7

when "110" => s\_frac <= "0000000"; -- .75 rounds to .8

when "111" => s\_frac <= "0010000"; -- .875 rounds to .9

when others => s\_frac <="1111111";

end case;

end process;

end architecture rtl; -- of sincos