* Lab 1
  + Embedded Systems
  + Lab Report 1
  + Sahana Asokan
  + 2-28-2019
* Purpose
  + The purpose of this lab is to
* Clock
  + Theory of Operation
    - The circuit should have a counter that increments for every rising edge of the clock. When the signal is equivalent to the division ratio then the output single will be 1. For this specific circuit, clock enable will the the output signal. In this circuit we need to divide our input by a value that gets us us 2 Hz.
    - If designed correctly, I expect the circuit to have an output of 62.5, because 125/2=62.5.
  + Design
    - VHDL Code for the clock\_div

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity counters\_1 is

port(CLK :in std\_logic;

output : out std\_logic);

end counters\_1;

architecture CNT of counters\_1 is

signal count: std\_logic\_vector(25 downto 0):=(others => '0');

begin

process (CLK)

begin

if rising\_edge(CLK) then count <= std\_logic\_vector(unsigned(count) +1);

if(count ="11101110011010110010100000") then

output <= '1';

count <= (others => '0');

else

output <= '0';

end if;

end if;

end process;

end CNT;

* + - VHvVHDL code for Divider\_Top

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

--entity declaration reg1

entity reg1 is

port(C,CE :in std\_logic;

Q : out std\_logic);

end reg1;

--architecture declaration reg1

architecture register\_arc of reg1 is

signal D : std\_logic;

begin

reg\_prc: process(C)

begin

if (rising\_edge(C)) then

if (D ='1') then

Q <= CE;

end if;

end if;

end process reg\_prc;

end register\_arc;

--entity clock div

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity counters\_1 is

port(CLK :in std\_logic;

output : out std\_logic);

end counters\_1;

--arch clock div

architecture CNT of counters\_1 is

signal count: std\_logic\_vector(25 downto 0):=(others => '0');

begin

process (CLK)

begin

if rising\_edge(CLK) then count <= std\_logic\_vector(unsigned(count) +1);

if(count ="11101110011010110010100000") then

output <= '1';

count <= (others => '0');

else

output <= '0';

end if;

end if;

end process;

end CNT;

--entity declaration for re circuit

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity entire is

port( C : in std\_logic;

Q: out std\_logic);

end entire;

architecture entire\_circuit of entire is

-- component declaration for seperate parts

component reg1 is

port( C,CE :in std\_logic;

Q : out std\_logic);

end component;

component counters\_1 is

port(CLK :in std\_logic;

output : out std\_logic);

end component;

signal d: std\_logic;

signal CE: std\_logic;

begin

U1: counters\_1

port map (output => CE,

CLK => C);

led\_reg: process(C)

begin

if(rising\_edge(C)) then

if (CE ='1') then

Q <= not(d);

end if;

end if;

end process;

Q <= d;

end entire\_circuit;

* Test Bench for Clock

library IEEE;

use IEEE.std\_logic\_1164.all;

entity counter\_tb is

end counter\_tb;

architecture tb of counter\_tb is

component counter

port (

CLK : in std\_logic;

output : out std\_logic);

end component;

signal tb\_clk : std\_logic := '0';

signal tb\_cnt : std\_logic;

begin

dut: counter port map (clk => tb\_clk, output => tb\_cnt);

process begin

for iter in 0 to 15 loop

tb\_clk <= '0';

wait for 4 ns;

tb\_clk <= '1';

wait for 4 ns;

end loop;

end process;

end tb;

* Schematic for Divider Top

../Downloads/schematic.pdf

* Synthesis

dividertopsynth.pdf

* Debouncer
  + Theory of Operation
    - This circuit is implementing a button that will be enabled by the behavior of a clock. The button will be working in the same regard to our system clock from the previous part. If the register counter value is not given a value of 1 then the counter will reset. Once the max value is hit the counter will not increment anymore and the output of the debouncer entirely will be 0. So in real life oscillation for a button can cause issues so we need an algorithm or a logic statement that takes care of those outside factors.
    - If designed correctly, I expect this circuit to implement a logic vector where each clock bit of 0 gets a new value and the bit 1 gets a value at bit 0. If the tested value is not 1 then the counter will be reset, and once the max value is hit the output will display a 0 and the counter will stop incrementing as stating earlier in the theory.
  + Design
    - Code for Debouncer

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity button is

port(btn,CLK :in std\_logic;

output : out std\_logic);

end button;

architecture CNT of button is

signal count: std\_logic\_vector(25 downto 0):=(others => '0');

begin

process (CLK)

begin

if rising\_edge(CLK) then

if(btn = '1') then

if(count ="1001100010010110100000") then

output <= '1';

count <= (others => '0');

else

output <= '0';

end if;

end if;

end if;

end process;

end CNT;

* + - Test Bench for Debouncer

library IEEE;

use IEEE.std\_logic\_1164.all;

entity counter\_tb is

end counter\_tb;

architecture tb of counter\_tb is

component counter

port (

CLK : in std\_logic;

output : out std\_logic);

end component;

signal tb\_clk : std\_logic := '0';

signal tb\_cnt : std\_logic;

begin

dut: counter port map (clk => tb\_clk, output => tb\_cnt);

process begin

for iter in 0 to 15 loop

tb\_clk <= '0';

wait for 20 ns;

tb\_clk <= '1';

wait for 20 ns;

end loop;

end process;

end tb;

* Fancy Counter
  + Theory of Operation
    - This circuit is going to model a counter with extra signals. The only signal that can change the logic of the circuit is en=1. If en =1 and clk\_en =0 then nothing will change the circuit. When rst is implemented then the count value will become 0. When the direction affects the counter then when the clock rising edge is 1 in regards to the direction updn will also be 1. If ld is 1 the current value at val will be inputted into the value register. We can count up or down.
    - If designed correctly I expect the fancy counter to operate a logical system that was in the directions. All the signals should be in sync and the counter should be ale to count both up or down and act accordingly depending on the value of the direction register which is affected by the clock’s rising edge.
  + Design
    - VHDL Code for fancy counter

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity fancy is

port(clk, clk\_en,dir,en,ld,rst,updn : in std\_logic;

val: in std\_logic\_vector(3 downto 0);

CNT : inout std\_logic\_vector(3 downto 0));

end fancy;

architecture archi of fancy is

signal tmp: std\_logic;

begin

process(clk)

begin

if (en = '1') then

if(rising\_edge(clk)) then

if (rst = '0') then

if (clk\_en = '1') then

if (updn ='1') then

tmp <= dir;

end if;

if (ld = '1') then

CNT <= val;

else if (tmp = '1') then

if (CNT = val) then

CNT <= "0000";

else

CNT <= std\_logic\_vector (unsigned(cnt) + 1);

end if;

else if( tmp = '0') then

if (CNT = "0000") then

CNT <= val;

else

CNT <= std\_logic\_vector (unsigned(cnt) - 1);

end if;

end if;

end if;

end if;

end if;

else if (rst = '1') then

cnt <= "0000";

if (clk\_en = '1') then

if (updn = '1') then

tmp <= dir;

end if;

end if;

end if;

end if;

end if;

end if;

end process;

end archi;

* + - Test bench for Fancy Counter

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity fancy\_counter is

end fancy\_counter;

architecture fancy\_tb of fancy\_counter is

component fancy\_counter

port(

clk, clk\_en, dir, en, ld, rst, updn: in std\_logic;

val : in std\_logic\_vector(3 downto 0);

CNT: out std\_logic\_vector(3 downto 0));

end component;

signal tb\_clk: std\_logic := '0';

signal tb\_en, dir, en, ld, rst, updn : std\_logic := '0';

signal val, CNT : std\_logic\_vector(3 downto 0);

begin

clk: process

begin

wait for 4 ns;

tb\_clk <= '1';

wait for 4 ns;

tb\_clk <= '0';

end process clk;

dut: fancy\_counter port map ( clk => tb\_clk, clk\_en => tb\_en, dir => dir, en => en, ld => ld, rst => rst, updn => updn, val=> val, CNT => CNT);

end fancy\_tb;

* Entire Circuit
  + Theory of Operation
    - The theory behind this part of the lab is to assemble all the different parts that we designed together. This will create its entirely own schematic with different logic structure for each one. The code for this circuit will focus on combining all the parts.
    - If each of the separate parts are designed correctly, then I expect that will preform as desired with the zybo.
  + Design
    - VHDL code for counter top

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity counters\_1 is

port(clk :in std\_logic;

output : out std\_logic);

end counters\_1;

architecture countb of counters\_1 is

signal count: std\_logic\_vector(25 downto 0):=(others => '0');

begin

process (clk)

begin

if rising\_edge(clk) then count <= std\_logic\_vector(unsigned(count) +1);

if(count ="11101110011010110010100000") then

output <= '1';

count <= (others => '0');

else

output <= '0';

end if;

end if;

end process;

end countb;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity button is

port(btn,clk :in std\_logic;

output : out std\_logic);

end button;

architecture archi of button is

signal count: std\_logic\_vector(25 downto 0):=(others => '0');

begin

process (clk)

begin

if rising\_edge(clk) then

if(btn = '1') then

if(count ="1001100010010110100000") then

output <= '1';

count <= (others => '0');

else

output <= '0';

end if;

end if;

end if;

end process;

end archi;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity fancy is

port(clk, clk\_en,dir,en,ld,rst,updn : in std\_logic;

val: in std\_logic\_vector(3 downto 0);

cnt : inout std\_logic\_vector(3 downto 0));

end fancy;

architecture archi of fancy is

signal tmp: std\_logic;

begin

process(clk)

begin

if (en = '1') then

if(rising\_edge(clk)) then

if (rst = '0') then

if (clk\_en = '1') then

if (updn ='1') then

tmp <= dir;

end if;

if (ld = '1') then

CNT <= val;

else if (tmp = '1') then

if (CNT = val) then

CNT <= "0000";

else

CNT <= std\_logic\_vector (unsigned(cnt) + 1);

end if;

else if( tmp = '0') then

if (CNT = "0000") then

CNT <= val;

else

CNT <= std\_logic\_vector (unsigned(cnt) - 1);

end if;

end if;

end if;

end if;

end if;

else if (rst = '1') then

cnt <= "0000";

if (clk\_en = '1') then

if (updn = '1') then

tmp <= dir;

end if;

end if;

end if;

end if;

end if;

end if;

end process;

end archi;

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity counter\_top is

Port (

btn: in std\_logic\_vector(3 downto 0);

clk: in std\_logic;

sw: in std\_logic\_vector(3 downto 0);

led: inout std\_logic\_vector(3 downto 0));

end counter\_top;

architecture archi of counter\_top is

component counters\_1

port(clk : in std\_logic;

output : out std\_logic);

end component;

component button

Port ( btn: in std\_logic;

clk: in std\_logic;

output: out std\_logic);

end component;

component fancy

port (

clk, clk\_en, dir, en, ld, rst, updn: in std\_logic;

val : in std\_logic\_vector(3 downto 0);

cnt: inout std\_logic\_vector(3 downto 0));

end component;

signal but1, but2, but3, but4, clockout : std\_logic;

begin

u1: button port map( btn => btn(0),clk => clk,output => but1);

u2: button port map( btn => btn(1),clk => clk,output => but2);

u3: button port map( btn => btn(2), clk => clk, output => but3);

u4: button port map( btn => btn(3), clk => clk, output => but4);

u5: counters\_1 port map( clk => clk, output => clockout);

u6: fancy port map( clk => clk, clk\_en => clockout, dir => sw(0), rst => but1, en => but2,updn => but3,ld => but4,val => sw, cnt => led);

end archi;

* Design
  + Schematic for Counter Top

schematic2.pdf

* Discussion
  + Lab Manual Questions
    - 1. How much do we need to divide our input by to get from 125 MHz to 2 Hz?
      * 62.5 MHz
    - 2. How many bits are required to store a counter that can count up to the value obtained in
      * We had to convert 62.5 to binary digits. 26 bits.
    - 3. What is the value of the button when it is pressed for the Zybo?
      * 1
    - 4. If we want our debounce time to be 20 ms, and our system clock is 125 MHz, how many ticks do we need a steady ‘1’ to be read for it to count as a ‘1’ ?
      * 2,500,000 ticks
    - 5. How many bits are required for a counter that can go that high?
    - 22 bits.
  + Oberservations/Discoveries
    - I learned a lot in this lab even though it was really difficult. Part 4 was really nice because I could see all the parts coming together.
  + Questions
    - I would still like more review in regards to the zybo and implementing different sources. The learning curve from the synthesis lab to this was extremely high.