Daniel Ackuaku

Hardware Lab 2: Basic Clocked Registers

ENGR 304

9th April 2019.

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

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

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

-- Add a USE statement for the package containing function calls that support

-- the arithmetic operations on unsigned data. Similar libraries are available

-- for signed (2's complement) arithmetic operations.

USE ieee.std\_logic\_unsigned.ALL;

-- Add a USE statement for the package containing function calls that support

-- the 7-segment displays on the DE2 board.

USE work.SevenSeg\_pkg.ALL;

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

-- ENTITY: HWLab2

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

ENTITY HWLab2 IS

PORT (

-- The entity port names are intended to match the names found in the pin

-- assignment file. Matching the names makes pin assignment easier, but

-- it is not required.

SW : IN std\_logic\_vector(15 downto 0);

KEY : IN std\_logic\_vector(3 downto 0);

HEX0 : OUT std\_logic\_vector(6 downto 0);

HEX1 : OUT std\_logic\_vector(6 downto 0);

HEX2 : OUT std\_logic\_vector(6 downto 0);

HEX3 : OUT std\_logic\_vector(6 downto 0);

HEX4 : OUT std\_logic\_vector(6 downto 0);

HEX5 : OUT std\_logic\_vector(6 downto 0);

HEX6 : OUT std\_logic\_vector(6 downto 0);

HEX7 : OUT std\_logic\_vector(6 downto 0)

);

END HWLab2;

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

-- ARCHITECTURE: behav

-- This architecture is implemented with behavioural VHDL

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

ARCHITECTURE behav OF HWLab2 IS

-- This area is used to define types and any internal signals

-- Fill\_In as necessary to define signals like "Result", etc

-- Defining the signals for input, output and timing control of the progamme

signal registerClock , resultClock, Reset : std\_logic; -- timing control signals

signal registerA : std\_logic\_vector(7 downto 0); -- input register

signal registerB : std\_logic\_vector(7 downto 0); -- input reigster

signal sumRegister : std\_logic\_vector(11 downto 0); -- output sumRegister

BEGIN

-- concurrent signal assignments placed here

-- Assign internal signals to external port names (e.g. SW15..8 = InputB)

-- use the conversion function to display the least signif 4-bits of Result

-- Assigning the signal values to theri respective Hardware components

registerClock <= KEY(3); -- triggers the update of registers with values read in from the SW

resultClock <= KEY(2); -- triggers the sum computation.

Reset <= KEY(0); -- resets the system

-- Fill\_In as necessary to drive the remaining HEX displays

-- this section updades the HEX displays after resultClock &/ registerClock is pressed

HEX7 <= convert\_to\_7seg(registerA(7 downto 4));

HEX6 <= convert\_to\_7seg(registerA(3 downto 0));

HEX5 <= convert\_to\_7seg(registerB(7 downto 4));

HEX4 <= convert\_to\_7seg(registerB(3 downto 0));

-- handling the overflow bit.

-- if sumRegister is greater that 255 or ( 1111111 ) bits then HEX2 <= sumRegister(11 downto 8) <=

HEX2 <= "1111111" when sumRegister(11 downto 8) = "0000" else convert\_to\_7seg(sumRegister(11 downto 8));

HEX1 <= convert\_to\_7seg(sumRegister(7 downto 4));

HEX0 <= convert\_to\_7seg(sumRegister(3 downto 0));

-- use "others" to drive busses of wires (e.g. turning off hex displays)

-- setting HEX3 to 1

HEX3 <= "1111111";

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

-- process

-- reading the input into registerA & registerB

-- Based on the binary value of the fisrt 8 SWs and next 8SWs.

-- A 4bit unsigned bit is generated.

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

process (registerClock, Reset) is

begin

if ( Reset = '0') then -- if reset is pushed

registerA <= "00000000" ; -- set registerA to 0

registerB <= "00000000"; -- set registerB to 0

elsif (registerClock'EVENT AND registerClock = '1') then -- when Key3 is pushed

registerA <= SW(15 downto 8); -- set registerA

registerB <= SW(7 downto 0); -- set registerB

end if;

end process;

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

-- process

-- SumRegister ... computes the sum of the 8 bit ( the sum signal is zero

-- paded with 4 zeros so it fits into convert\_to\_7seg function)

-- this block calculates the Sum of the hex digits num1 and num2

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

process (resultClock, Reset, registerA, registerB) is

begin

if ( Reset = '0') then -- if reset is pushed

sumRegister <= "000000000000" ; -- set sumRegister to 0

elsif (resultClock'EVENT AND resultClock = '1') then -- when Key3 is pushed

-- compute the sum

sumRegister <= (( "0000" & registerA ) + ( "0000" & registerB ));

end if;

end process;

END behav;

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

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

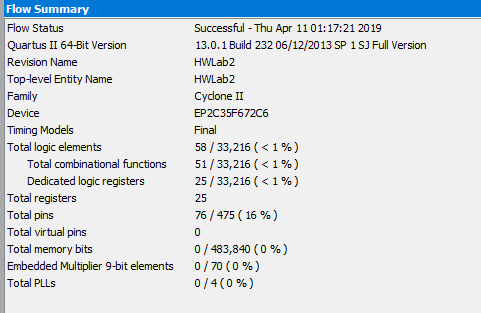


Figure The compilation summary report from Quartus



Figure compilation warnings in Quartus

The purpose of the lab was to program the Altera DE2 boards to receive two 8-bit unsigned numbers, find the sum and place the result in a 9-bit result register and displays the answer on the HEX display. The control flow of the program was managed using Keys on the DE2 board with key Key0 as reset, Key3 as a clock for the input registers and Key2 as a clock for the result register.

To tackle this program, I created a block diagram that detailed the necessary processes required.

RegisterA – the input register for the 1st 8-bit number. The value stored in this register was driven by the lower 8 switches. The register was updated with switch inputs when Key3 was pressed. The value of the register was set to zero when Key0 was pressed.

RegisterB - the input register for the 2nd 8-bit number. The value stored in this register was driven by the next 8 switches on the board. This register was also updated with switch inputs when Key3 was pressed. The value of the register was set to zero when Key0 was pressed.

The process block representing the adder was incorporated into the RegisterResult.

RegisterResult – this 9-bit register performs the sum of the 2 inputs. Its value is updated anytime Key2 is pressed. It is also reset on Key0.

Check Carryout – this check if the size of the result is greater that 255. If it is it sets the additional HEX display to 1 else, it turns of the display.

Predicted number of registers = 8 + 8 + 9 = 25

Actual number of registers = 25

Testing.

I employed an iterative testing approach to test my code. After each major process I ran sudo tests to ensure a working program. I made sure the register clocks worked and the reset worked. Next, I tested the arithmetic portion of the program. I started off with the extreme cases of FF + FF = 1FE and the zero case 00 + 00 = 00. Finally, I tested the overflow case 01 + FF = 100. All the results were correct.