Hardware Lab 3: Iterative Multiplication on the Altera DE2 Board

Prof Jo.

18th April 2019.

Daniel Ackuaku

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

-- FILE: HWLab3

-- This file contains VHDL that implements Lab 2 for the Altera DE2 Board.

-- DESCRIPTION:

-- In this lab, the switches are used to input digits. HEX displays are used

-- to show the hex value of the operation result and the operands.

-- COURSE: Engineering 304

-- DESIGN TOOL: Fill\_In

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

-- MODIFICATION HISTORY:

-- Revision 1.1 Fill\_In

-- Fill\_In

-- Revision 1.0 3/27/18 2:00 PM Prof. Brouwer

-- File template as supplied by the professor.

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

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;

-- MAth library for counter

use IEEE.math\_real.all;

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

-- ENTITY: HWLab3

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

ENTITY HWLab3 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(16 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);

LEDR : OUT std\_logic\_vector(15 downto 0);

LEDG : OUT std\_logic\_vector(7 downto 0)

);

END HWLab3;

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

-- ARCHITECTURE: behav

-- This architecture is implemented with behavioral VHDL

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

ARCHITECTURE behav OF HWLab3 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 inputClock , resultClock, Reset : std\_logic; -- timing control signals

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

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

signal shiftRegister : std\_logic\_vector(7 downto 0); -- output shiftRegister

signal computeResult : std\_logic\_vector(7 downto 0); -- output computeResult

signal count : std\_logic\_vector(7 downto 0); -- counter

signal display : std\_logic\_vector(7 downto 0);

signal Q : std\_logic\_vector(7 downto 0); -- used in multiplicand mux loop

signal A : std\_logic\_vector(7 downto 0); -- used in multiplier mux loop

signal sumVal : std\_logic\_vector(7 downto 0); -- used in multiplier mux loop

signal Result : std\_logic\_vector(8 downto 0);

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

inputClock <= KEY(3); -- triggers the update of registers with values 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 updates the HEX displays after resultClock &/ inputClock is pressed

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

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

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

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

-- this displays the result

HEX3 <= convert\_to\_7seg(computeResult(7 downto 4));

HEX2 <= convert\_to\_7seg(computeResult(3 downto 0));

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

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

-- LEDR assignments

LEDR(7 downto 0) <= shiftRegister(7 downto 0);

LEDR(15 downto 8) <= Result(7 downto 0);

LEDG(7 downto 0) <= count(7 downto 0);

-- Pre-adder multiplicand MUX

-- controls the input to the adder.

sumVal <= SW(15 downto 8) when shiftRegister(0) = '1' else "00000000";

-- Sum

Result <= ('0' & computeResult) + sumVal;

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

-- process

-- reading the input into multiplierReg & multiplicandReg

-- Based on the binary value of the first 8 SWs and next 8SWs, and the state

-- of SW16.

-- Two 8 bit unsigned numbers are generated.

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

process (inputClock, Reset) is

begin

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

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

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

-- mux structure that feeds the 2 input registers

elsif ( inputClock'EVENT AND inputClock = '1') then

if ( SW(16) = '1') then -- when select is high (SW16)

-- input is read from switches on KEY3

-- when Key3 is pushed

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

multiplicandReg <= SW(7 downto 0); -- set multiplicandReg end if;

end if;

end process;

-- shiftRegister MUX controls the input to the computeResult.

Q <= SW(7 downto 0) when SW(16) = '1' else (Result(0) & shiftRegister(7 downto 1));

-- computeResult MUX controls the input to the computeResult.

A <= "00000000" when SW(16) = '1' else Result(8 downto 1);

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

-- process

-- populating the registers.

-- the shift register logic.

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

process (inputClock, Reset) is

begin

if ( Reset = '0') then

shiftRegister <= "00000000";

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

shiftRegister <= Q;

end if;

end process;

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

-- process

-- populating the registers.

-- the compute Result logic.

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

process (inputClock, Reset) is

begin

if (Reset = '0') then

computeResult <= "00000000";

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

computeResult <= A;

end if;

end process;

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

-- process

-- controls the counter LEDGs it increments on each key press of KEY3 to

-- indicate they progress of the computation.

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

process (inputClock, Reset) is

begin

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

display <= "00000000";

count <= "00000000";

elsif (SW(16) = '0') then

if (inputClock'EVENT AND inputClock = '1') then -- when Key3 is pushed

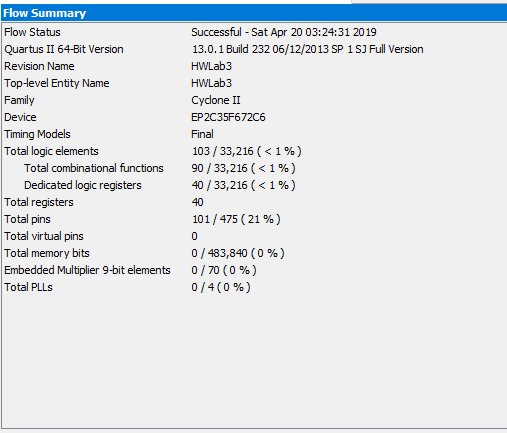
count <= ('1' & count(7 downto 1));

end if;

end if;

end process;

END behav;



The purpose of the lab was to program the Altera DE2 boards to receive two 8-bit unsigned numbers, find the product of the two numbers using the additive multiplication technique. The computation takes place in 9-bit result register that updates and stores its values iteratively in 2 8-bit registers until all bits in the 2 8-bit registers are filled. The computations stored in the 2 8-bit registers are displayed on the lowest 4 HEX displays as well as the LEDRs. To indicate the progress of the computation, the LEDGs were used. 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 SW16 as a MUX selector for the result register MUXes.

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

multiplierReg – 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 if SW16 is high. The value of the register was set to zero when Key0 was pressed.

multiplicandReg - 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. The register was updated with switch inputs when Key3 was pressed if SW16 is high. The value of the register was set to zero when Key0 was pressed.

Since these two registers were both performing parallel functions, I combined them and their respective MUXes into the same process.

A 9-bit signal called Result was created to represent the adder in the block diagram.

compResult – 8-bit registers used in the computation of result. After 8 clock cycles the values in register represent the higher 8 bits of the final solution.

shiftRegister- 8-bit registers used in the computation of result. The highest bit of this register is either filled with the lowest value of result or the multiplier depending on the value of SW16 (MUX). After 8 clock cycles the values in register represent the lower 8 bits of the final solution.

For a complete multiplication to be carried out the computation takes 8 clock cycles (KEY3). In order to display the process of the computation the LEDGs were used. Each time a clock cycle occurs an additional LED is turned on. Once all 8 are on the user knows the answer displayed is the final answer.

Predicted number of registers = 8 + 8 + 8 + 8 + 9 = 41. MY program has 4 8-bit registers and a 9-bit signal that dovetails as a register.

Actual number of registers = 40

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 zero case 00 + 00 = 00 then I tackled the cases in the handout c7 + F7 = c001.