Digital Design Lab

ECE 316

Lab/Project 3

Simple ALU with Hardware Implementation

Group 1

Karysse Hay, Nikeem Dunkelly-Allen, Johnny Brown

University of Miami

10/18/2022

Overview

The premise of this lab was to create and implement an 8 bit calculator with the ability to take two 8-bit signed values, X and Y (in binary), and perform eight different operations to the given inputs. The concepts required for the successful completion of this lab included implementing a previously created 8-bit register, creating a simple Arithmetic Logic Unit (internal) which allowed for the different calculation processes and overflow checks, configuring a display circuit that was compatible with the DE1-SoC board and pin mapping inputs and outputs. Understanding the functionalities and capabilities of quartus prime was also a key component of this lab, and we faced issues mapping the displays in regards to this new software. Despite various issues faced throughout the completion of this lab, we gained an understanding of register splitting, display connection, as well as a much better conceptualization of the process of creating, debugging, and implementing a doscript.

Software and Hardware

- Quartus Prime
- DE1-SoC board
- ModelSim

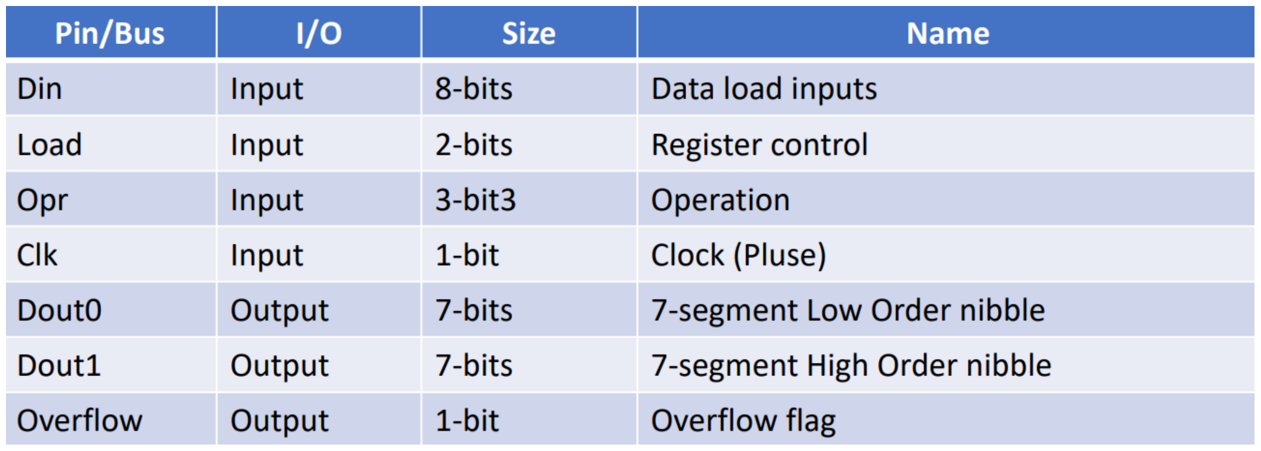
Description

For this Lab, two 8-bit vector inputs controlled the calculation entity- core8, enabling the usage of a prior code that was created for the 8-bit register. The register allowed for two different inputs to be loaded using only one input signal in the Top Level circuit. In the core8 entity, we created two inputs, X and Y, with 8 bits to give the values to the calculator, and in addition created a 3-bit opcode input to choose between operations using X and Y. In regards to the different operations, case statements were created to address each operation and within these case statements, conditional “if-else” statements were utilized in order to take care of the different occurrences of overflow. The basis of this Overflow code is that when the signs of X and Y are different, there is no overflow. However, if the signs of the two inputs are the same and the result is different, then there will be overflow. Overflow will also occur if X is at a maximum and the operation of “X+1” is called.

The last step in this process was the implementation of the display code, which was provided to us in order to display the results on the DE1-SoC board. The first step in this display process was to create a top level entity, and when creating the architecture for this entity we implemented the 8-bit register, calculation processing including the overflow check (core8), and display circuit. A do script was created where we were able to simulate some combinations to test the working of the circuit on ModelSim. When this was perfected, the transfer of our files from ModelSim to QuartusPrime was the next step.

The Top Level components were then used in order to portmap to the DE1-SoC board. Next, the input values were hard coded, and the port map was used to do the calculation process, as well as check overflow. To display the result on the board, we split the 8-bits into a left and right register. This catered for the upper and lower bits of the result. This enabled each part of the result to be sent to a separate single seven segment display for easy reading. This was then demoed to the Professor for signoff. The results of testing on ModelSim can be seen in the results section.

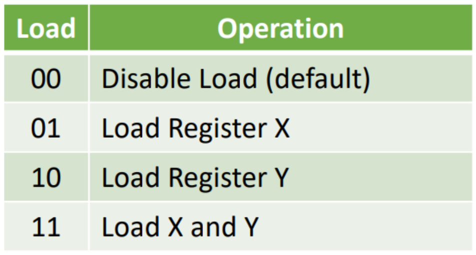
Specifications



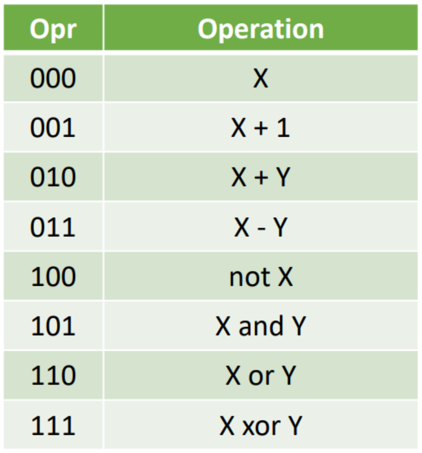
Pin/Bus	I/O	Size	Name
Din	Input	8-bits	Data load inputs
Load	Input	2-bits	Register control
Opr	Input	3-bit3	Operation
Clk	Input	1-bit	Clock (Pluse)
Dout0	Output	7-bits	7-segment Low Order nibble
Dout1	Output	7-bits	7-segment High Order nibble
Overflow	Output	1-bit	Overflow flag

Figure 1: Interface specifications for ALU

Design Synthesis



Load	Operation
00	Disable Load (default)
01	Load Register X
10	Load Register Y
11	Load X and Y



Opr	Operation
000	X
001	X + 1
010	X + Y
011	X - Y
100	not X
101	X and Y
110	X or Y
111	X xor Y

Figure 2: Load and Operation input designation

A majority of this project consisted of designing the functionality of the calculator, achieved this through the use of "if-else" statements for the conditionals. Signals were created for each operation code that mapped the variable equation into our result. For example, when the opcode was "101" we mapped "X and Y" into our result. Similarly when the opcode was "110" we mapped "X or Y" into our result. This was done for all operation codes given. Another feature that needed to be programmed was our logic for overflow, which was done by adding two numbers that would cause an overflow in order to evaluate what condition would be required to check for. An entity was then created with the purpose of determining if an overflow flag should be set for any result given the situation. After all of the results were verified and split into their appropriate upper and lower registers, the next step was to pin each register to a single 7 segment display. This was done by utilizing the DE1 user manual, in order to understand what pins correlated to what segment, and then map them to our left and right displays.

Complete Diagram

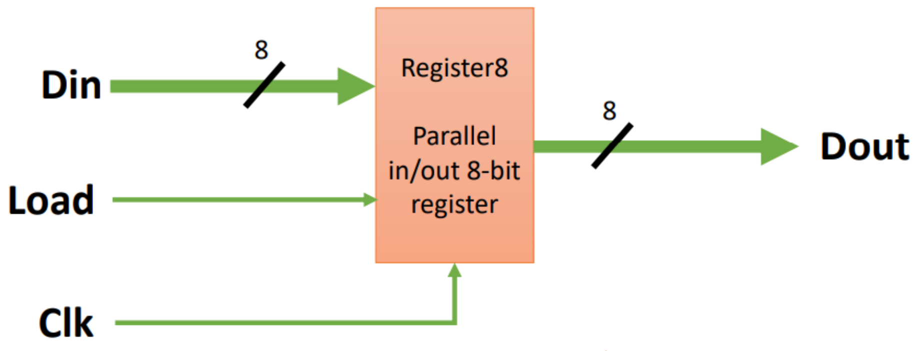


Figure 3: 8-bit Register

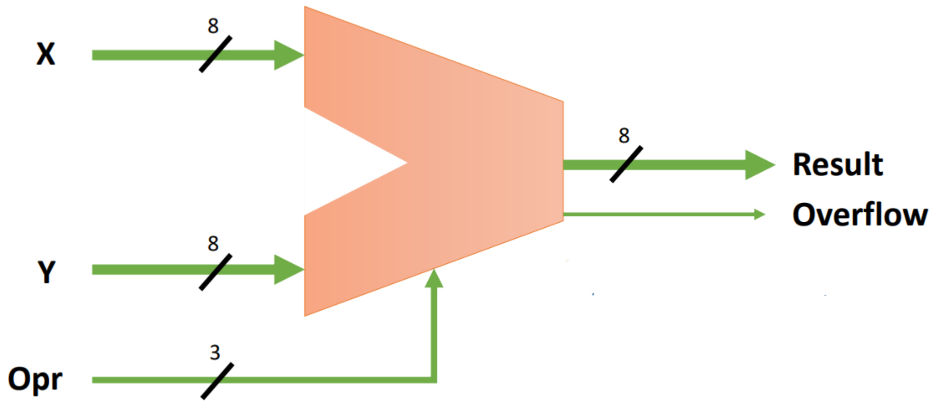


Figure 4: Core 8 internal inputs and outputs

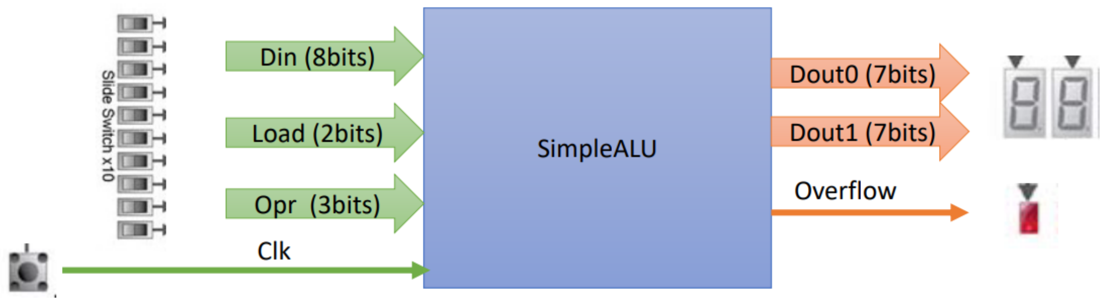


Figure 4: External View of ALU

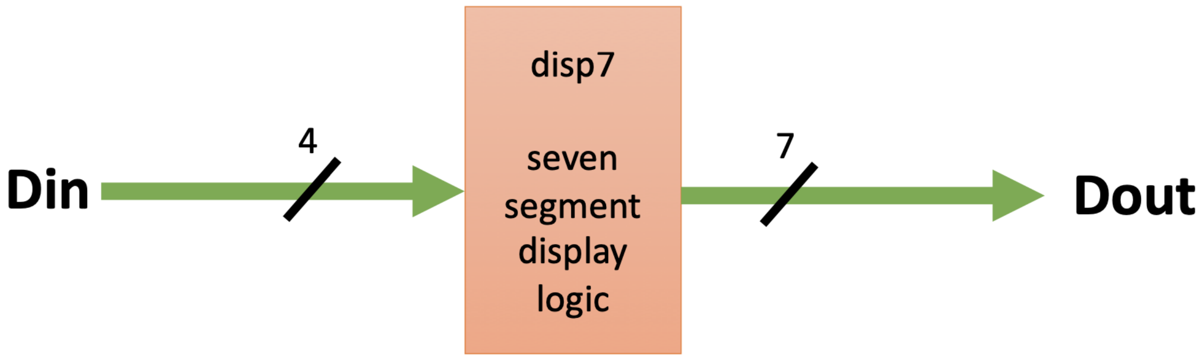


Figure 5: Seven Segment Display (two are needed for the left and right numbers)

Results and Simulations

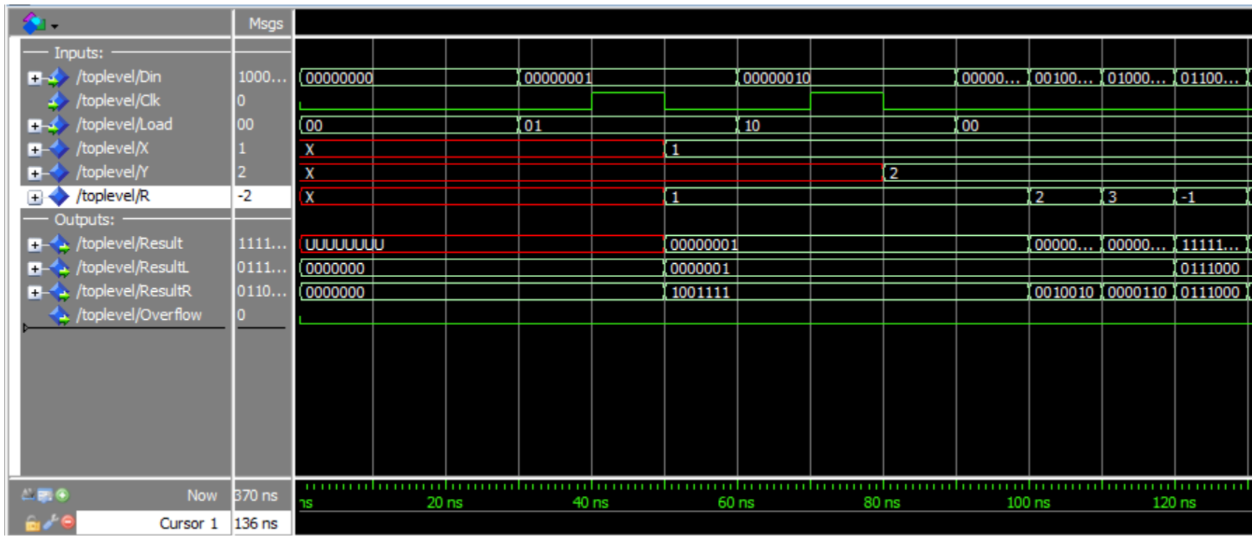


Figure 6: Do file going through operations 000 to 011

Using the do file, everything is set to 0, then X and Y are loaded to 1 and 2 respectively using the Din and the Load input. After each value is loaded to Din the clock pulses in order to save the number in its respective register. Then operations 000 to 011 are done consecutively, the result can be seen as decimal in the Result output.

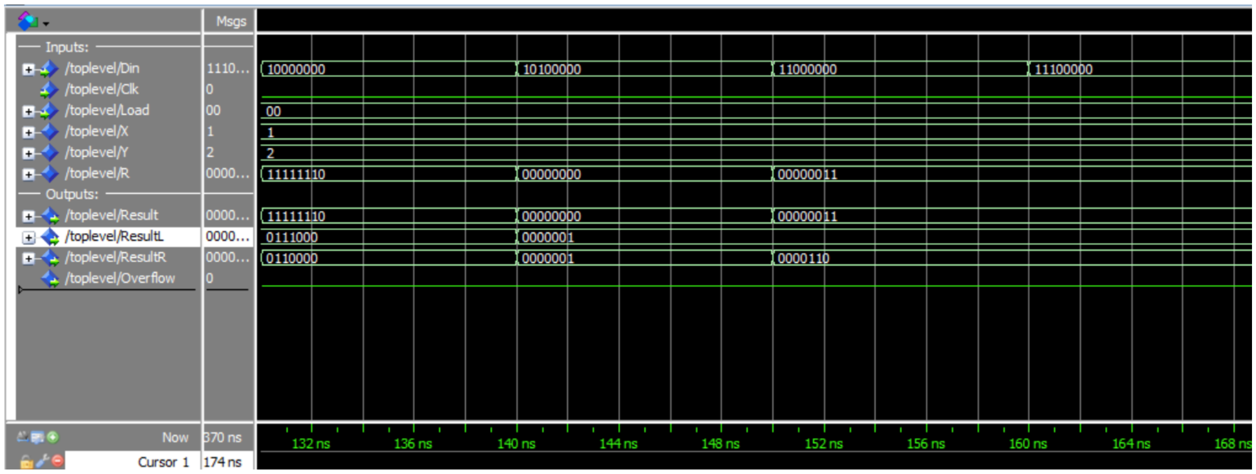


Figure 7: Do file going through operations 100 to 111

Still using the do file with inputs from *Figure 6* operations 100 to 111 are done consecutively, the result is in binary due to bitwise operations in the Result output.

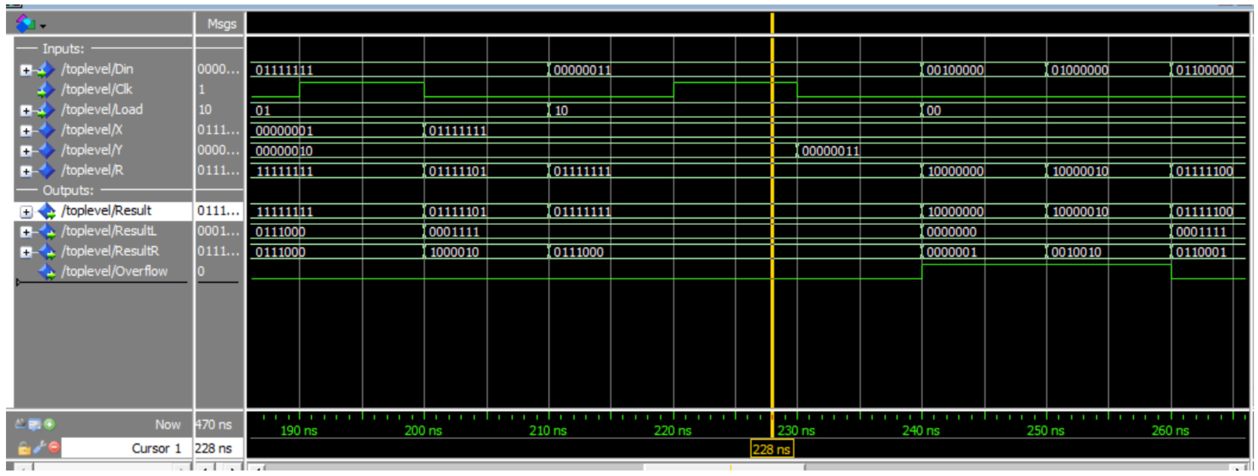


Figure 8: Testing the Overflow check

Continuing testing for functionality X and Y are loaded into Din respectively using Load, then operations 001, 010, and 011 are performed, the result can be seen in binary in the Result output wave (which is highlighted). As you can see when X was set to 01111111 and the operation was 001, the overflow bit was set high. Another example of the overflow bit being set high is the addition of X+Y when X is 011111111 and Y is 00000011.

Conclusion

The implementation of this assignment was fairly straightforward given the slides provided from the lecture. The process flow of this lab consisted of mapping all of the inputs and outputs together including the 8-bit register, calculation process entity with the overflow check (core8), display circuit and the top level circuit. This process required us to follow the same sequence as our entity to make sure everything was going to the correct location.

Some problems we encountered was getting QuartusPrime to be compatible with our personal computer. After all the files were downloaded and put into the correct place, another issue arose which was that the DE1-Soc board was not being recognized by QuartusPrime. Due to time constraints, we then decided to use the allocated computers in the University's lab to demo our completed calculator.

From this lab, we were able to gain a better understanding of the quartus software, as well as a struggle point for us which was pinning our results to a segment in our display. Additionally, we also got a better understanding of the reason behind splitting up registers in the process of sending them to individual displays, as well as the limitations of which one display can represent in hex. In conclusion, the process of this lab was considerably more difficult than previous labs, but quite insightful in terms of the scope of concepts learned and implemented.

Signed OFF

Please see attached.

Appendix

CORE 8

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;


entity core8 is
        port (X, Y: in std_logic_vector (7 downto 0);
                Opr: in std_logic_vector (2 downto 0);
                Result: out std_logic_vector (7 downto 0);
                Overflow: out std_logic);
end core8;

architecture behav of core8 is
        signal R: std_logic_vector (7 downto 0);
        --alias R: std_logic_vector (7 downto 0) is Result;
begin
        PR_core8: process(X, Y, Opr, R)
        begin
                Overflow <= '0'; --initialize it
                case Opr is
                        when "000" => R <= X;
                        when "001" => R <= X+1;
                              if (X = "01111111") then
                                  Overflow <= '1';
                              else
                                   Overflow <= '0';
                              end if;
                        when "010" => R <= X+Y;
                              if (X(7) /= Y(7)) then
                                  Overflow <= '0';
                              elsif (X(7) /= R(7)) then
                                  Overflow <= '1';
                              else
                                  Overflow <= '0';
                              end if;
                        when "011" => R <= X-Y;
                              if (X(7) = Y(7)) then
                                  Overflow <= '0';
                              elsif (X(7) = R(7)) then
                                  Overflow <= '1';
                              else
                                  Overflow <= '0';
                              end if;
                        when "100" => R <= NOT X;
                        when "101" => R <= X AND Y;
                        when "110" => R <= X OR Y;
                        when "111" => R <= X XOR Y;
                        when others => R <= "00000000";
                end case;
                Result<=R;

        end process PR_core8;

end behav;
```

8 BIT REGISTER

```
library ieee;
use ieee.std_logic_1164.all;

Entity Register8 is
    Port(Din : in std_logic_vector(7 downto 0);
        Load, Clk : in std_logic;
        Dout : out std_logic_vector(7 downto 0));
end Register8;

Architecture behav of Register8 is
    Signal storage : std_logic_vector(7 downto 0);
        begin
        process(Din, Load, Clk)
        begin
            if (Clk'event and Clk = '1' and Load = '1') then
                storage <= Din;
            elsif (Clk'event and Clk = '0') then
                Dout <= storage;
            end if;
        end process;
```

```
		end;
```

DISPLAY CKT

```
library ieee;
use ieee.std_logic_signed.all;
use ieee.std_logic_1164.all;

entity DisplayCkt is
	port(Input: in std_logic_vector(3 downto 0);
		SevenSeg : out std_logic_vector(6 downto 0));

end DisplayCkt;

Architecture behav_display of DisplayCkt is
begin
process (Input)
Begin
 case Input is
	when"0000"=> SevenSeg <= "0000001"; -- 0 --change all these around
	when"0001"=> SevenSeg <= "1001111"; -- 1
	when"0010"=> SevenSeg <= "0010010"; -- 2
	when"0011"=> SevenSeg <= "0000110"; -- 3
	when"0100"=> SevenSeg <= "1001100"; -- 4
	when"0101"=> SevenSeg <= "0100100"; -- 5
	when"0110"=> SevenSeg <= "0100000"; -- 6
	when"0111"=> SevenSeg <= "0001111"; -- 7
	when"1000"=> SevenSeg <= "0000000"; -- 8
	when"1001"=> SevenSeg <= "0000100"; -- 9
	when"1010"=> SevenSeg <= "0001000"; -- 10 A
	when"1011"=> SevenSeg <= "1100000"; -- 11 b
	when"1100"=> SevenSeg <= "0110001"; -- 12 c
	when"1101"=> SevenSeg <= "1000010"; -- 13 d
	when"1110"=> SevenSeg <= "0110000"; -- 14 e
	when"1111"=> SevenSeg <= "0111000"; -- 15 f
	when others => SevenSeg <= "0000000";
end case;
end process;
end behav_display;
```

TOP LEVEL

```
Library ieee;

use ieee.std_logic_signed.all;
use ieee.std_logic_1164.all;
```

```
entity TopLevel is
        port ( Din: in std_logic_vector(7 downto 0);
           Clk: in std_logic;
           Load: in std_logic_vector(1 downto 0);
           --Opr : in std_logic_vector(2 downto 0);
           Result: out std_logic_vector(7 downto 0);
           ResultL: out std_logic_vector(6 downto 0);
           ResultR: out std_logic_vector(6 downto 0);
           Overflow: out std_logic);
end entity;

architecture behav of TopLevel is

component Register8
        Port(Din : in std_logic_vector(7 downto 0);
                Load, Clk : in std_logic;
                Dout : out std_logic_vector(7 downto 0));
end component;

component core8
        port (X, Y: in std_logic_vector (7 downto 0);
                Opr: in std_logic_vector (2 downto 0);
                Result: out std_logic_vector (7 downto 0);
                Overflow: out std_logic);
end component;

component DisplayCkt is
        port(Input: in std_logic_vector(3 downto 0);
                SevenSeg : out std_logic_vector(6 downto 0));
end component;

signal X: std_logic_vector ( 7 downto 0);
signal Y: std_logic_vector ( 7 downto 0);
signal R: std_logic_vector ( 7 downto 0);

begin

RegisterX: Register8 port map ( Din, Load(0), Clk, X);
RegisterY: Register8 port map ( Din, Load(1), Clk, Y);
Calculator: core8 port map ( X, Y, Din (7 downto 5), R, Overflow); -- no overflow
Result<=R;

outL: DisplayCkt port map (R(7 downto 4),ResultL);
outR: DisplayCkt port map (R(3 downto 0),ResultR);
```

```
end behav;
```

DO SCRIPT FOR TOP LEVEL

```
# close any existing simulations
quit -sim

# compile our VHDL file
vcom TopLevel.vhd

# start the simulation - set timescale to nanoseconds
vsim -t ns TopLevel

# add the inputs to the wave
# first add a divider with a label
add wave -divider Inputs:
add wave Din Clk Load X Y R

# add the outputs to the wave
# first add a divider with a label
add wave -divider Outputs:
add wave Result ResultL ResultR Overflow

# do initial run with default values - optional
#run 10 ns

# force values

force Din "00000000"
force Clk 0
force Load "00"

run 30ns
#initialize

#load X reg
#force Din to X

force Din "00000001"
force Load "01"
run 10ns

force Clk 1
run 10ns
force Clk 0
```

```
run 10ns

#load y to register
force Din "00000010"
force Load "10"
run 10ns

force Clk 1
run 10ns
force Clk 0
run 10ns

#load opr
force Din "00000000"
force Load "00"
run 10ns

#load opr
force Din "00100000"
force Load "00"
run 10ns

#load opr
force Din "01000000"
force Load "00"
run 10ns

#load opr
force Din "01100000"
force Load "00"
run 10ns

#load opr
force Din "10000000"
force Load "00"
run 10ns

#load opr
force Din "10100000"
force Load "00"
run 10ns

#load opr
force Din "11000000"
force Load "00"
run 10ns

#load opr
force Din "11100000"
force Load "00"
run 10ns

#-----to test overflow

force Din "00000000"
force Clk 0
force Load "00"

run 10ns
#initialize

#load X reg
#force Din to X

force Din "01111111"
force Load "01"
run 10ns

force Clk 1
run 10ns
force Clk 0
run 10ns

#load y to register
force Din "00000011"
force Load "10"
run 10ns

force Clk 1
run 10ns
force Clk 0
run 10ns

#load opr
force Din "00100000"
force Load "00"
run 10ns
```

```
#load opr
force Din "01000000"
force Load "00"
run 10ns

#load opr
force Din "01100000"
force Load "00"
run 10ns

#set load to load X 01
#clk is higrun 10, then low for 10 / dont keep repeating
#force din to y
#set load to Y
#pulse clk
#put opr
#then run it
#can leave same values as X and Y
#for opr- force Din "10100000"


# run
run 200ns
```