Lab 1

EGCP 381

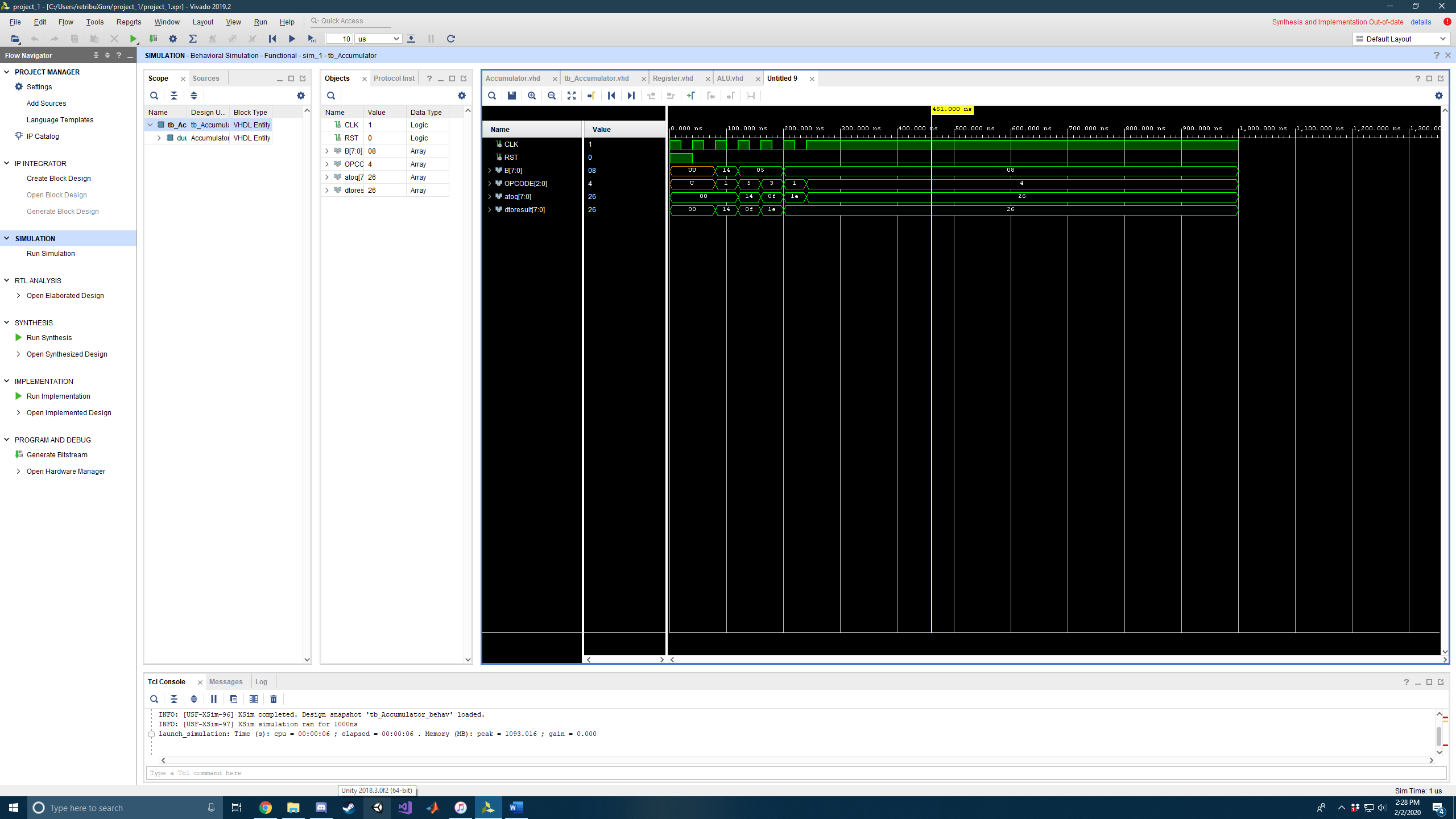
By Levi Randall

Introduction

In this lab, we were to re-sharpen our VHDL skills and combine our previous knowledge to create an ALU with a Register. This ALU is designed to have 8 different functions that are operated by an input called opcode. Computers contain both an ALU and Registers that function similarly to the ones in the VHDL code.

Procedure

I was successful in simulating the lab with the desired result.



We first created four separate files, one for the ALU, one for the register, a test bench file, and a final one to add the two components together. The ALU contained a MUX if statement that uses the opcode to decide the output of the ALU. The register file simply contained a clock, a reset, and an output that works off the rising edge of the clock. The test bench just puts inputs into the opcode and also initializes our input A and changes our input B.

*What operations are required by your processor to first****set the accumulator, A, to 100****, and then calculate****(A & 0x0F)/4****?*

The computer must first read in the values for A and for 0x0F. Then the computer will read in the opcode and determine that the operation is a concatenation. After, the computer must read in the 4 and read in the opcode to determine the operation is a divide.

*What is the final result of these operations?*

The result is 79/4 = 19.75 which is then truncated and rounded down to 19.

How would you change the design of this processor to add additional registers? E.g. instead of only having one accumulator register, how would you modify the design to have four registers?

To have more than one register, I would use two multiplexers where one would choose the operation and the other would determine which register the output would be fed too. All the registers would then be in parallel with one another.

How would you determine which of these four registers to read/write?

The multiplexer would take in another opcode to determine which output it was reading or writing to. This would function very similarly to how regular ‘4 to 1’ multiplexers work.

Conclusion

This lab showed the basic mechanisms of a computers ALU. It helped us to understand how the components of multiplexers and registers can be used with one another to create a mathematical calculator. Also, we were thoroughly retaught how to use most of the basic functions of VHDL code such as processes, components, test benches, and entities in order to complete this lab.

My only recommendation is that in the future you review how to map ports more thoroughly as it was the hardest part of this lab and I struggled the most with that section.

I don’t have any other questions or further comments about this lab. It was fun and a good review!

References

I used James Samawi as a reference and helper with debugging. And no other resources other than old code I created in 281.

Appendix

--Test Bench----------------------------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

entity tb\_Accumulator is

end tb\_Accumulator;

architecture Behavioral of tb\_Accumulator is

component Accumulator is

port(

signal CLK : in STD\_LOGIC;

signal RST : in STD\_LOGIC;

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

signal OPCODE : in std\_logic\_vector(2 downto 0);

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

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

end component;

--Signals

signal CLK : STD\_LOGIC;

signal RST : STD\_LOGIC;

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

signal OPCODE : std\_logic\_vector(2 downto 0);

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

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

begin

duuut: Accumulator port map(

B => B,

CLK => CLK,

RST => RST,

OPCODE => OPCODE,

atoq => atoq,

dtoresult => dtoresult

);

--(A-5)\*2+8

process

begin

CLK <= '1';

RST <= '1';

wait for 20 ns;

CLK <= '0';

wait for 20 ns;

CLK <= '1';

RST <= '0';

wait for 20 ns;

CLK <= '0';

wait for 20 ns;

CLK <= '1';

B <= "00010100"; --Initialize 20

OPCODE <= "001";

wait for 20 ns;

CLK <= '0';

wait for 20 ns;

CLK <= '1';

B <= "00000101";

OPCODE <= "101"; --Subtract 5

wait for 20 ns;

CLK <= '0';

wait for 20 ns;

CLK <= '1';

OPCODE <= "011"; --Multiply by 2

wait for 20 ns;

CLK <= '0';

wait for 20 ns;

CLK <= '1';

B <= "00001000";

OPCODE <= "001"; --ADD 8

wait for 20 ns;

CLK <= '0';

wait for 20 ns;

CLK <= '1';

OPCODE <= "100";

wait for 1000 ns;

end process;

end Behavioral;

--Accumulator --------------------------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

entity Accumulator is

port(

signal CLK : in STD\_LOGIC;

signal RST : in STD\_LOGIC;

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

signal OPCODE : in std\_logic\_vector(2 downto 0);

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

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

end Accumulator;

architecture Behavioral of Accumulator is

component Registers is

port (

d : in STD\_LOGIC\_vector(7 downto 0);

clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

q : out STD\_LOGIC\_vector( 7 downto 0 ));

end component;

component ALU is

port (

a : in std\_logic\_vector(7 downto 0);

b : in std\_logic\_vector(7 downto 0);

result : out std\_logic\_vector(7 downto 0);

opcode : in std\_logic\_vector(2 downto 0));

end component;

begin

dut:Registers port map(

q => atoq,

d => dtoresult,

clk => CLK,

rst => RST

);

duut:ALU port map (

a => atoq,

b => B,

result => dtoresult,

opcode => OPCODE

);

-- process

-- begin

-- A <= Q;

-- D <= RESULT;

-- end process;

end Behavioral;

--ALU--------------------------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity ALU is

port (

a : in std\_logic\_vector(7 downto 0);

b : in std\_logic\_vector(7 downto 0);

result : out std\_logic\_vector(7 downto 0);

opcode : in std\_logic\_vector(2 downto 0));

end ALU;

architecture Behavioral of ALU is

signal apb, aab, am2, asb, aob, ad2 : std\_logic\_vector (7 downto 0);

begin

apb <= a + b;

aab <= a AND b;

am2 <= a(6 downto 0) & '0';

asb <= a - b;

aob <= a OR b;

ad2 <= '0' & a(7 downto 1);

--MUX

result <= "00000000" when opcode = "000" else

apb when opcode = "001" else

aab when opcode = "010" else

am2 when opcode = "011" else

a when opcode = "100" else

asb when opcode = "101" else

aob when opcode = "110" else

ad2 when opcode = "111" else

"00000000";

end Behavioral;

--Register--------------------------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

entity Registers is

port (

d : in STD\_LOGIC\_vector ( 7 downto 0);

clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

q : out STD\_LOGIC\_vector ( 7 downto 0 ));

end Registers;

architecture Behavioral of Registers is

begin

process(clk, rst)

begin

if(rst = '1') then

q <= "00000000";

elsif(rising\_edge(clk)) then

q <= d;

end if;

end process;

end Behavioral;