**Arpita Sikder**

**500726025**

**COE 328**

**SECTION 041**

**DESIGN OF A SIMPLE GENERAL PURPOSE PROCESSOR**

**DESIGN OF A SIMPLE GENERAL PURPOSE PROCESSOR**

***Table Of Content:***

1. ***Introduction***
2. ***Materials used***
3. ***Procedures***
4. ***Discussion and Conclusion***
5. ***Appendix***
6. ***Table 1***
7. ***VHDL Codes***

* ***Latch***
* ***ASU\_core***
* ***ASU\_core 2***
* ***ASU\_core 3***
* ***Decoder***
* ***Synchronous\_upcount***

1. ***Schematic circuit***
2. ***Pin assignment***
3. ***Simulation result***

* ***Latch***
* ***Decoder***
* ***Synchronous\_upcount***
* ***ASU\_core***
* ***ASU\_core 2***
* ***ASU\_core 3***

**Objective and Introduction**

The main objective of this experiment was to design and create a simple Processor using VHDL coding of Quartus II software which is capable of performing common and easy functions for two 8 bit inputs. The functions in table1**. Page7** were designed and executed using Microcodes, an ALU processing core, a 3-8 decoder, a synchronous up counter, and registers (latches). Once these functions were executed the following results were shown using LED display of the ALTERA FPGA BOARD.

**Equipment and materials**

1. Quartus II software
2. Altera FPGA board
3. Ryerson Lab manual 6 [1]
4. FUNDAMENTALS OF DIGITAL LOGIC WITH VHDL DESIGN, THIRD EDITION by Stephen Brown and Zvonko Vranesic

**Theory**

**ALU (Arithmetic Logic Unit) Processing code**: A simple ALU processor consists of the control unit, the bus, the latches and the ALU. The control unit which is made up of the decoder and the up-counter receives the instructions from the two eight bit inputs and the single eight bit microcodes are used to choose the desired functions. The bus helps get access to these inputs. The latches temporarily store these data until the ALU system finally performs the arithmetic and logical functions on the inputs. In this lab, the ALU with eight functions were implemented using VHDL coding.

**3-8 Decoder**: A decoder acts as a code converter. In this lab, the 3-8 decoder takes 3 inputs from the synchronous up-counter and produces an eight bit output which in turn represents the microcodes for the eight different functions. The Truth table of the decoder is provided in **Table 2. Page 7**

**Synchronous Up-counter**: In this lab, the up-counter has two inputs, clock and reset. Its output acts as the input for the 3-8 decoder. The up-counter is synchronous with the clock, as a result it helps to change the output of the decoder as the clock counts. For example, in the first count, only Function 1 is active, then in the second count, function 2 is active and so on. Therefore for eight counts of the clock, eight different functions are produced. On the eighth count, the up-counter resets itself to zero and the whole cycle starts again.

**Latches with robes (Registers):** These are storage elements which registers the inputs once they enter the system. Their purpose is to store the two 8 bit input numbers until a series of functions are executed. Otherwise after the first function is being applied, the values of the inputs will change and the next function will use different values of A and B. But once they receive instructions from the ALU core, the values are being released.

**Display module**

The output of the arithmetic functions are displayed here in seven segment. The eight bit output value is split into two 4 bits hexadecimal display.

**Procedure:**

1) At the beginning of the lab, we started by writing the VHDL codes for each of the components of the processor inside a single Project named “Lab6”.

2) The code for the latch was written, so that each latch can take one 8 bit input. The resulting output will also be 8 bits. The latch also has a clock so that it is synchronous with other components and as a result there are no delays. The reset helps to make the value of 8 bit input to ‘’00000000’’. Therefore the user can input new 8 bit values. **The code for the latch is in page8 and the waveform inpage20**

3) The code for the synchronous up-counter was already provided to us in the Ryerson lab 6 manual. <http://www.ee.ryerson.ca/~courses/coe328/lab6.pdf>. It has one three bit output named “Q”. When clock changes from 0 to 1 (positive edge) and reset is 0, the value of Q increments by 1 and finally stops when they become “111”. But if the reset is set to 1 at the positive edge of the clock, the Q is set to”000”. **The code for the up-counter is in page8 and the waveform inpage8**

4) The code of the decoder was written to convert 3 bit input ‘w’ to 8 bit output ‘y’. Every time the clock counts, the values of ‘w’ changes which in turn changes the ‘y’, the function to be performed. The code was written with the help of table 1 and table 2. **The code is provided in page and the waveform in page**

5) The structure of the ALU processing core was already provided to us in the lab6 manual Appendix. The code inside the “process” was modified to carry out the functions: ADD, SUB, NOT, NAND, NOR, AND, OR and XOR as stated in table 1. **The code is provided in page**

6) Then a schematic circuit (block diagram) with same name as the project was created. The circuit contains two latches, one up-counter, one decoder and the ALU unit. The following schematic diagram was then updated and compiled. After the successful compilation, the pins in the Altera FPGA board were then assigned for the all the inputs and outputs. The results were manually done and shown to our respected TA.

7) Then a simulation report was done for this part and the resulting **successful waveform is given in page**

8) For the next problem “2a”, the new file was created with the same ALU structure. But the “process” was modified again so that it performs the following:

a) Take input two 8 bit numbers (A and B) to produce their difference, keep the output constant for the next 3 clock cycles/periods and then alternate the output to produce sum for last 4 cycles.

**The ALU code, waveform are given in page 14 and 18 respectively.**

9)The same procedure was repeated for the problem “3e” which performed the following: Take input two 8 bit numbers (A and B) to generate the decrement A by e) Take input two 8 bit numbers (A and B) to generate the rotate right A by 2 (ror). Hold the output for the next 3 clock periods before changing the output to produce the logical shift right (srl) A by 2 for the remaining clock periods. **The ALU code, waveform are given in page 10 and respectively.**

**Discussion and Conclusion**

All the individual components in the processor were tested and ran properly. After the processor was created, pins were assigned to all the inputs and outputs, which when ran executed almost perfectly according to the requirements. But we encountered some minor errors while running the processor. Most of them were examined properly and later fixed.

The first problem we encountered despite successful compilation was the reset of the up-counter. As a result we were not getting the proper outputs. Then after going through the codes of latches and up-counter, we realized that the Reset of up-counter was set to 1 whereas that of latch was set to zero. Then we set all Reset to 1 and compiled and simulated the ALU core again. This time, the outputs were correct except at the first clock. After observing the waveforms we figured out that it was due to the delay of the signals at the start. That is because the signals were arriving at slightly different time. This might have happened because of the really sensitive switches of the Altera Board as the clock was changed manually.

**References**

[1] Ryerson University, Department of Electrical and Computer Engineering, COE328 – Digital Systems URL: <http://www.ee.ryerson.ca/~courses/coe328/lab6.pdf>

[2] FUNDAMENTALS OF DIGITAL LOGIC WITH VHDL DESIGN, THIRD EDITION by Stephen Brown and Zvonko Vranesic

Table1. Microcode’s generated by decoder. [1]



Table 2. Truth table for the 3-8 decoder.[1]



**CODE FOR LATCH (REGISTER)**

LIBRARY ieee ;

USE ieee.std\_logic\_1164.all ;

ENTITY latchAB IS

PORT ( D : IN STD\_LOGIC\_VECTOR(7 DOWNTO 0) ;

Resetn, Clock : IN STD\_LOGIC ;

Q : OUT STD\_LOGIC\_VECTOR(7 DOWNTO 0) ) ;

END latchAB ;

ARCHITECTURE Behavior OF latchAB IS

BEGIN

PROCESS ( Resetn, Clock )

BEGIN

IF Resetn = '1' THEN

Q <= "00000000" ;

ELSIF Clock 'EVENT AND Clock = '1' THEN

Q <= D ;

END IF ;

END PROCESS ;

END Behavior ;

**CODE FOR SYNCHRONOUS UPCOUNTER**

library ieee;

use ieee.std\_logic\_1164.ALL;

use ieee.std\_logic\_unsigned.ALL;

Entity upcounter is

port (reset, clock: IN std\_logic;

Q : BUFFER std\_logic\_vector (2 downto 0));

end upcounter;

architecture behaviour of upcounter is

begin

upcounter: process (clock)

begin

if (clock'event and clock = '1') then

if reset = '1' then

Q <= "000";

Else

Q <= Q + '1';

End if;

End if;

End process;

End behaviour;

**CODE FOR DECODER**

LIBRARY ieee ;

USE ieee.std\_logic\_1164.all ;

ENTITY decoder IS

PORT ( w : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0) ;

y : OUT STD\_LOGIC\_VECTOR(7 DOWNTO 0) ) ;

END decoder ;

ARCHITECTURE Behavior OF decoder IS

SIGNAL Enw : STD\_LOGIC\_VECTOR(2 DOWNTO 0) ;

BEGIN

Enw <= w ;

WITH Enw SELECT

y <= "00000001" WHEN "000",

"00000010" WHEN "001",

"00000100" WHEN "010",

"00001000" WHEN "011",

"00010000" WHEN "100",

"00100000" WHEN "101",

"01000000" WHEN "110",

"10000000" WHEN "111",

"00000000" WHEN OTHERS;

END Behavior ;

**CODE FOR ALU PROCESSOR (PART 1A STEP 5)**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity ALU\_core is

port( Clk : in std\_logic; --clock signal

A,B : in signed(7 downto 0); --input operands

Op : in unsigned(7 downto 0); --Operation to be performed

neg: out std\_logic;

R : out signed(8 downto 0) --output of ALU

);

end ALU\_core;

architecture Behavioral of ALU\_core is

--temporary signal declaration.

signal Reg1,Reg2,ab : signed(8 downto 0) := (others => '0');

begin

Reg1 <= ('0' & A);

Reg2 <= ('0' & B);

process(ab)

begin

if Reg1 > Reg2 THEN

ab<= (Reg1) + ((Reg2 NAND "11111111" ) + "00000001");

elsif Reg1 < Reg2 THEN

ab<= (((Reg1 + ((Reg2 NAND "11111111" ) + "00000001")) NAND "11111111") + "00000001");

end if;

end process;

process(Clk, Op)

begin

if(rising\_edge(Clk)) then --Do the calculation at the negative edge of clock cycle.

case Op is

when "00000001" => R <= Reg1 + Reg2;

when "00000010" => R <= Reg1 - Reg2;

when "00000100" => R <= Reg1 NAND "11111111";

when "00001000" => R <= Reg1 NAND Reg2;

when "00010000" => R <= Reg1 NOR Reg2;

when "00100000" => R <= Reg1 AND Reg2;

when "01000000" => R <= Reg1 OR Reg2;

when "10000000" => R <= Reg1 XOR Reg2;

when OTHERS => R <= "---------";

end case;

end if;

end process;

process(Op)

begin

if Op = "00000010" THEN

if Reg1> Reg2 THEN neg<= '0';

elsif Reg1 < Reg2 THEN neg<= '1';

end if;

else

neg<= '0';

end if;

end process;

end Behavioral;

**Display module**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

ENTITY displaymodule IS

PORT (neg : IN Std\_LOGIC;

neg1 : out Std\_LOGIC;

R : IN STD\_LOGIC\_VECTOR(8 DOWNTO 0);

leds, leds2: OUT STD\_LOGIC\_VECTOR(1 TO 7));

END displaymodule;

ARCHITECTURE Behavior OF displaymodule IS

BEGIN

PROCESS (R, neg)

BEGIN

CASE R(3 downto 0) IS

WHEN "0000" => leds <= "0000001";

WHEN "0001" => leds <= "1001111";

WHEN "0010" => leds <= "0010010";

WHEN "0011" => leds <= "0000110";

WHEN "0100" => leds <= "1001100";

WHEN "0101" => leds <= "0100100";

WHEN "0110" => leds <= "0100000";

WHEN "0111" => leds <= "0001111";

WHEN "1000" => leds <= "0000000";

WHEN "1001" => leds <= "0001100";

WHEN "1010" => leds <= "0001000";

WHEN "1011" => leds <= "1100000";

WHEN "1100" => leds <= "0110010";

WHEN "1101" => leds <= "1000010";

WHEN "1110" => leds <= "0110000";

WHEN "1111" => leds <= "0111000";

END CASE;

CASE R(7 downto 4) IS

WHEN "0000" => leds2 <= "0000001";

WHEN "0001" => leds2 <= "1001111";

WHEN "0010" => leds2 <= "0010010";

WHEN "0011" => leds2 <= "0000110";

WHEN "0100" => leds2 <= "1001100";

WHEN "0101" => leds2 <= "0100100";

WHEN "0110" => leds2 <= "0100000";

WHEN "0111" => leds2 <= "0001111";

WHEN "1000" => leds2 <= "0000000";

WHEN "1001" => leds2 <= "0001100";

WHEN "1010" => leds2 <= "0001000";

WHEN "1011" => leds2 <= "1100000";

WHEN "1100" => leds2 <= "0110010";

WHEN "1101" => leds2 <= "1000010";

WHEN "1110" => leds2 <= "0110000";

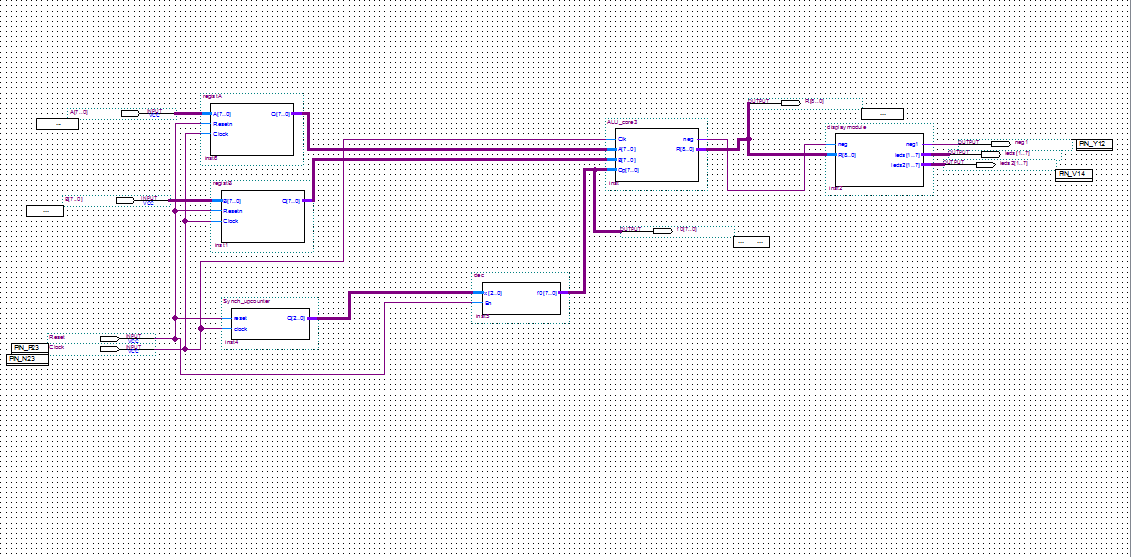
WHEN "1111" => leds2 <= "0111000";

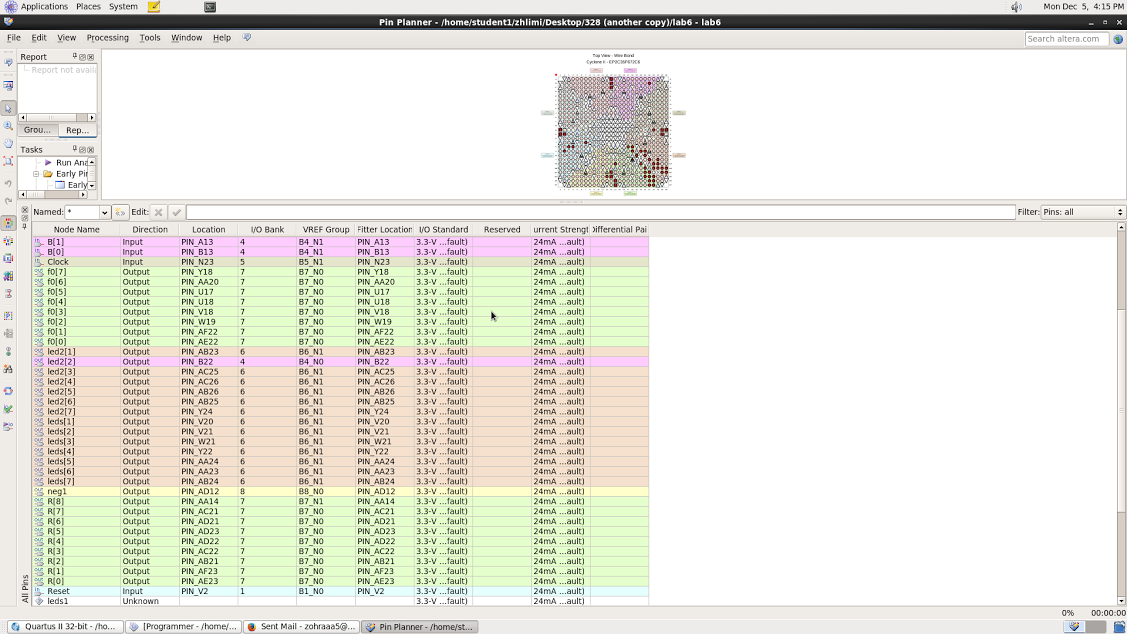
END CASE;

neg1 <= neg;

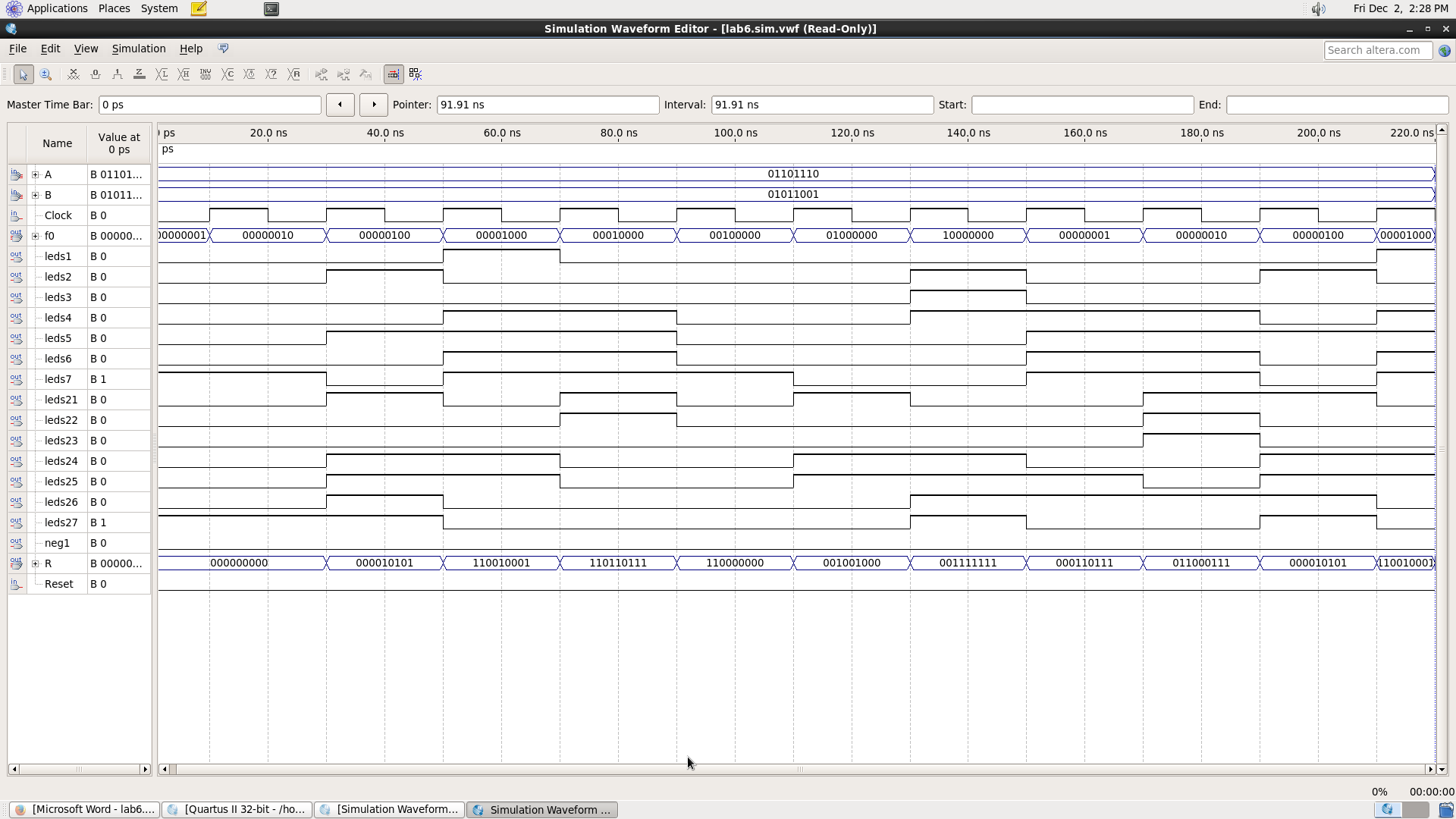
END PROCESS;

END Behavior;

**Block diagram for ALU core**



**Waveform for alu\_core**



**QUESTION 2 (A) ASSIGNED**

**VHDL CODE**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity ALU\_core2 is

port( Clk : in std\_logic; --clock signal

A,B : in signed(7 downto 0); --input operands

Op : in unsigned(7 downto 0); --Operation to be performed

neg: out std\_logic;

R : out signed(8 downto 0) --output of ALU

);

end ALU\_core2;

architecture Behavioral of ALU\_core2 is

--temporary signal declaration.

signal Reg1,Reg2,ab : signed(8 downto 0) := (others => '0');

begin

Reg1 <= ('0' & A);

Reg2 <= ('0' & B);

process(ab)

begin

if Reg1 > Reg2 THEN

ab<= (Reg1) + ((Reg2 NAND "11111111" ) + "00000001");

elsif Reg1 < Reg2 THEN

ab<= (((Reg1 + ((Reg2 NAND "11111111" ) + "00000001")) NAND "11111111") + "00000001");

end if;

end process;

process(Clk, Op)

begin

if(rising\_edge(Clk)) then --Do the calculation at the negative edge of clock cycle.

case Op is

when "00000001" => R <= Reg1 - Reg2;

when "00000010" => R <= Reg1 - Reg2;

when "00000100" => R <= Reg1 - Reg2;

when "00001000" => R <= Reg1 - Reg2;

when "00010000" => R <= Reg1 + Reg2;

when "00100000" => R <= Reg1 + Reg2;

when "01000000" => R <= Reg1 + Reg2;

when "10000000" => R <= Reg1 + Reg2;

when OTHERS => R <= "---------";

end case;

end if;

end process;

process(Op)

begin

if Op = "00000010" THEN

if Reg1> Reg2 THEN neg<= '0';

elsif Reg1 < Reg2 THEN neg<= '1';

end if;

else

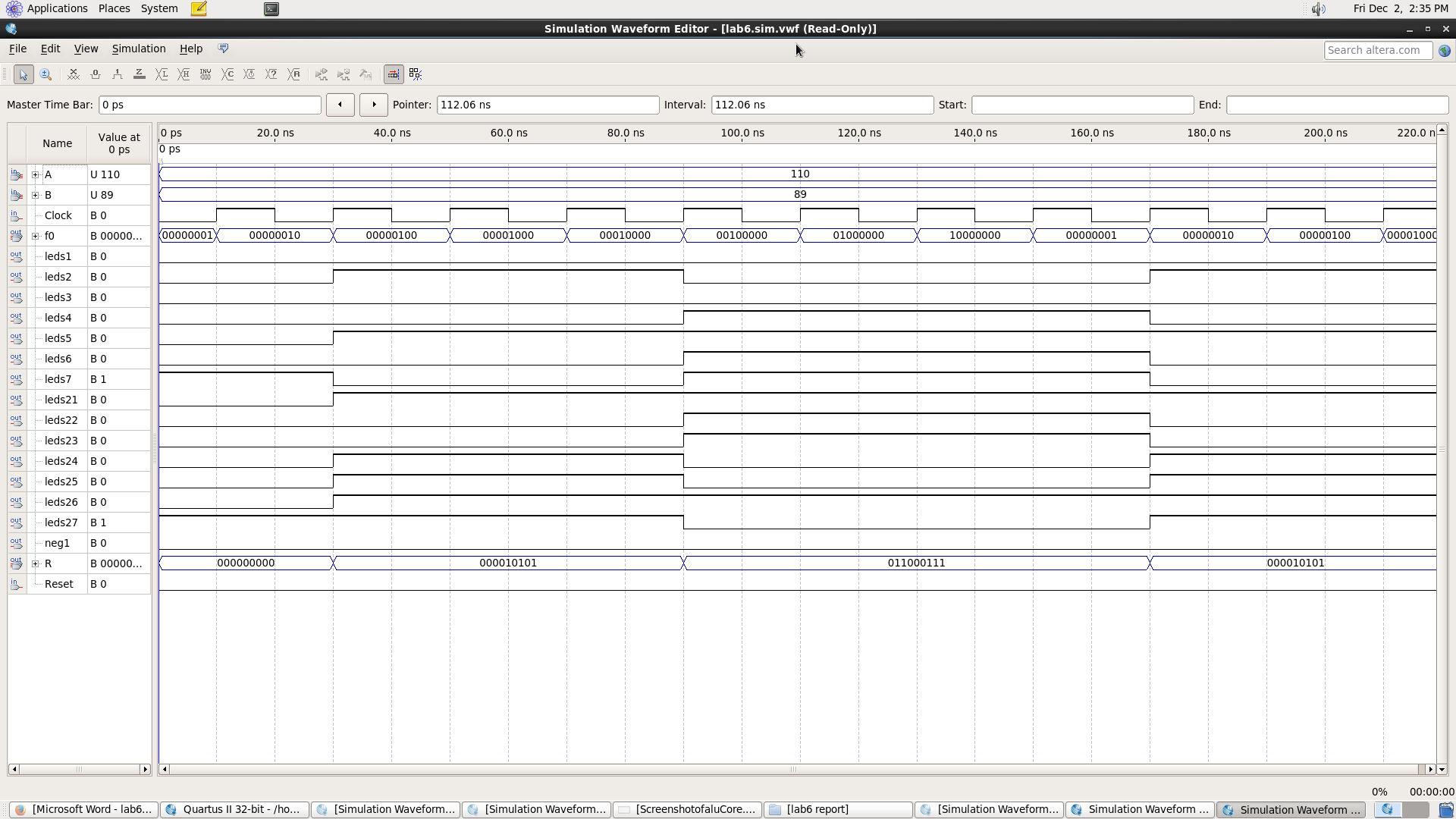
neg<= '0';

end if;

end process;

end Behavioral;

**WAVEFORM FOR ALU 2**



BLOCK DIAGRAM:

**Block diagram for ALU 2 looks similar to ALU CORE in page 12**

**QUESTION 3 (e ASSIGNED)**

**VHDL CODING**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity ALU\_core3 is

port( Clk : in std\_logic; --clock signal

A,B : in signed(7 downto 0); --input operands

Op : in unsigned(7 downto 0); --Operation to be performed

neg: out std\_logic;

R : out signed(8 downto 0) --output of ALU

);

end ALU\_core3;

architecture Behavioral of ALU\_core3 is

--temporary signal declaration.

signal Reg1,Reg2,ab : signed(8 downto 0) := (others => '0');

begin

Reg1 <= ('0' & A);

Reg2 <= ('0' & B);

process(ab)

begin

if Reg1 > Reg2 THEN

ab<= (Reg1) + ((Reg2 NAND "11111111" ) + "00000001");

elsif Reg1 < Reg2 THEN

ab<= (((Reg1 + ((Reg2 NAND "11111111" ) + "00000001")) NAND "11111111") + "00000001");

end if;

end process;

process(Clk, Op)

begin

if(rising\_edge(Clk)) then --Do the calculation at the negative edge of clock cycle.

case Op is

when "00000001" =>

R <= (Reg1 ror 2);

when "00000010" =>

R <= (Reg1 ror 2);

when "00000100" =>

R <= (Reg1 ror 2);

when "00001000" =>

R <= (Reg1 ror 2);

when "00010000" =>

R <= (Reg1 srl 2);

when "00100000" =>

R <= (Reg1 srl 2);

when "01000000" =>

R <= (Reg1 srl 2);

when others =>

R <= (Reg1 srl 2);

end case;

end if;

end process;

process(Op)

begin

if Op = "00000010" THEN

if Reg1> Reg2 THEN neg<= '0';

elsif Reg1 < Reg2 THEN neg<= '1';

end if;

else

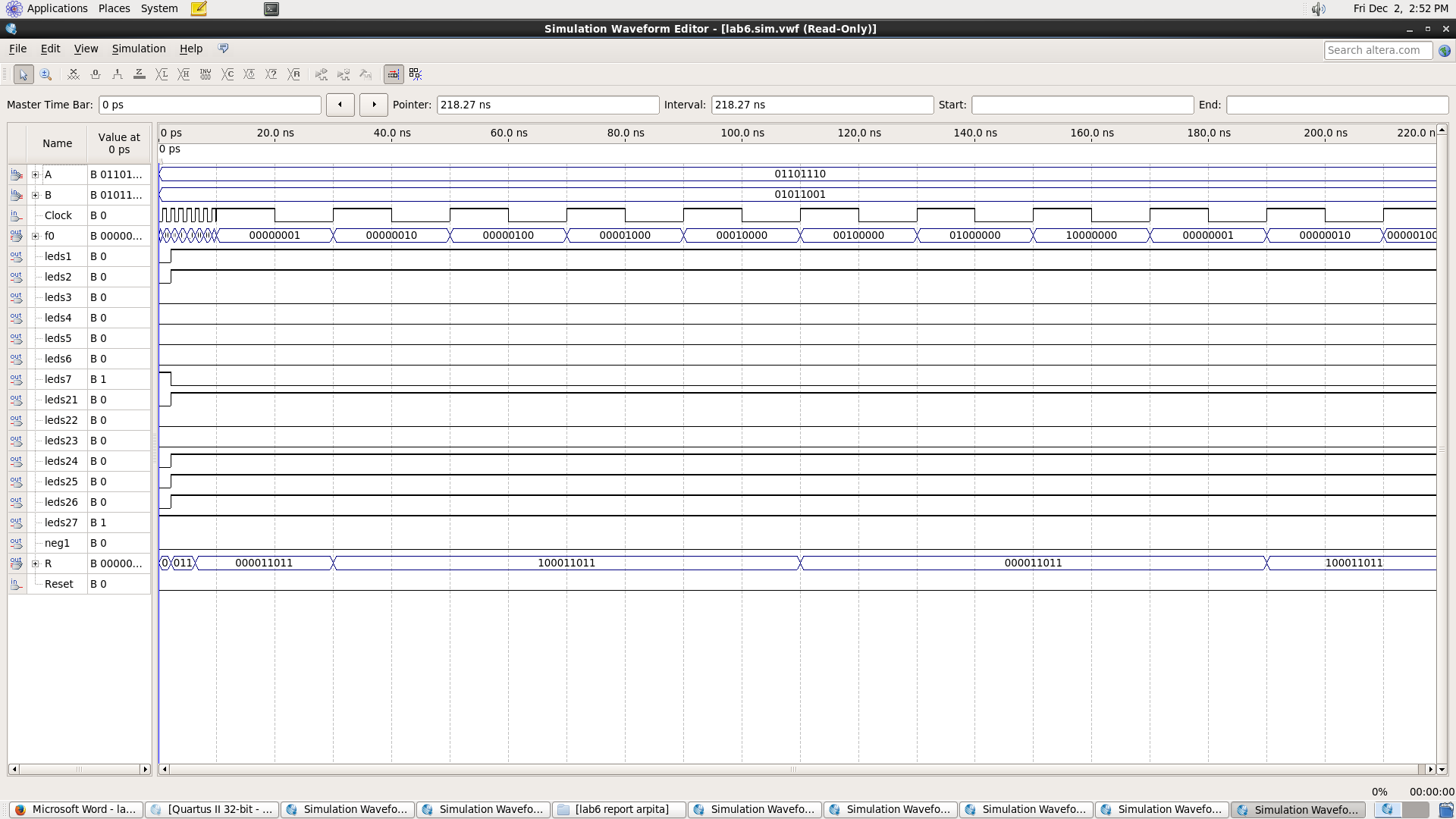
neg<= '0';

end if;

end process;

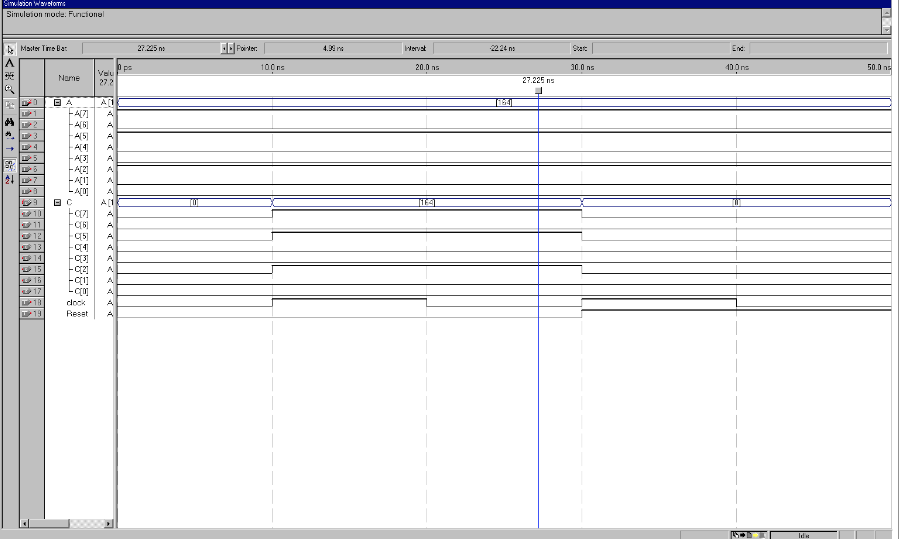
end Behavioral;

WAVEFORM FOR ALU CORE 3

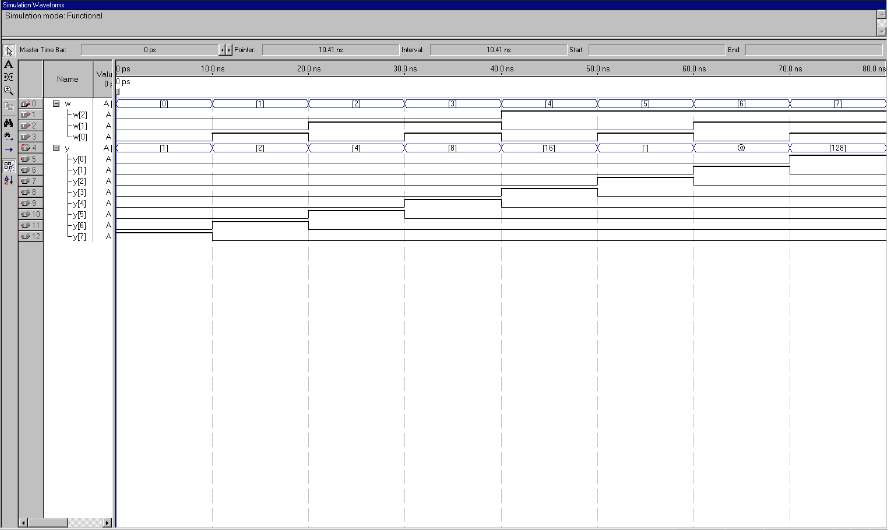


**Block diagram for ALU 3 looks similar to ALU CORE in page 12**

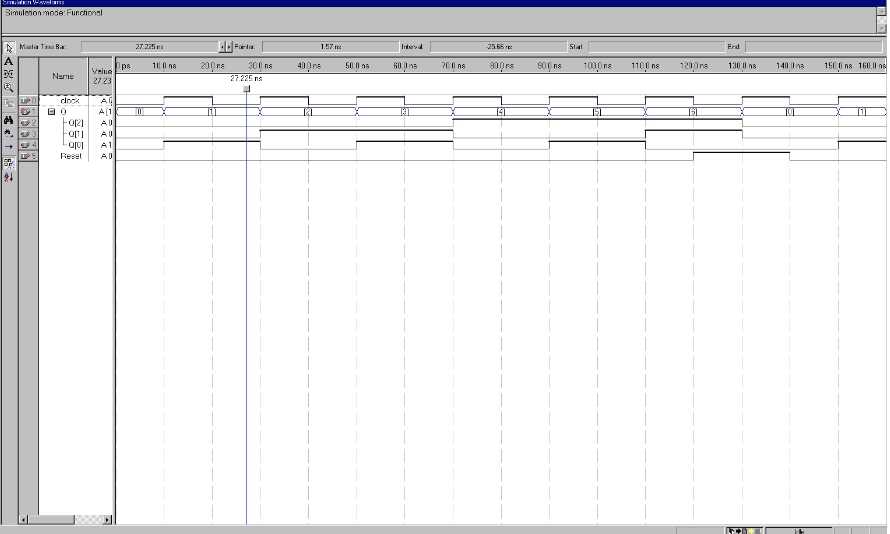
**Latch waveform**

****

**Decoder**

****

**Synchronous Upcounter**

****