
BME328 LAB7

Implementation of Simple Processor

40 Marks (3 weeks) Due **Date:** Week 13

Objective:

- Design and implementation of simple 8-bit microprocessor
- The processor consists of ALU, Registers to store data, Control unit to execute instructions pointed to by program counter (PC)

Summary of System Operation:

An abstract view of system is illustrated in Fig.1**.**

1. 8-bit input data is entered in the simulation phase through the Quartus II waveform editor, and stored in register R1 when instruction LD-R1 is executed
2. 8-bit ALU perform operation on input A and input B based on OP code of executed instruction
3. ALU input A is connected to R1, input B is connected to Accumulator AC.
4. Results of each operation is displayed on two 7 Seg display unit connected to output of AC
5. RC register stores 8-bit input data when LD-RC instruction is executed and RC output is used as a conditional register for control flow
6. PC is a counter that is used to point to next instruction to be executed and starts from Instruction 0 to Instruction N. Each clock cycle it increments PC to point to next instruction
7. Combinational circuit stores the Instructions starting from address 0 to address N. Each instruction has its op code to be used by ALU.

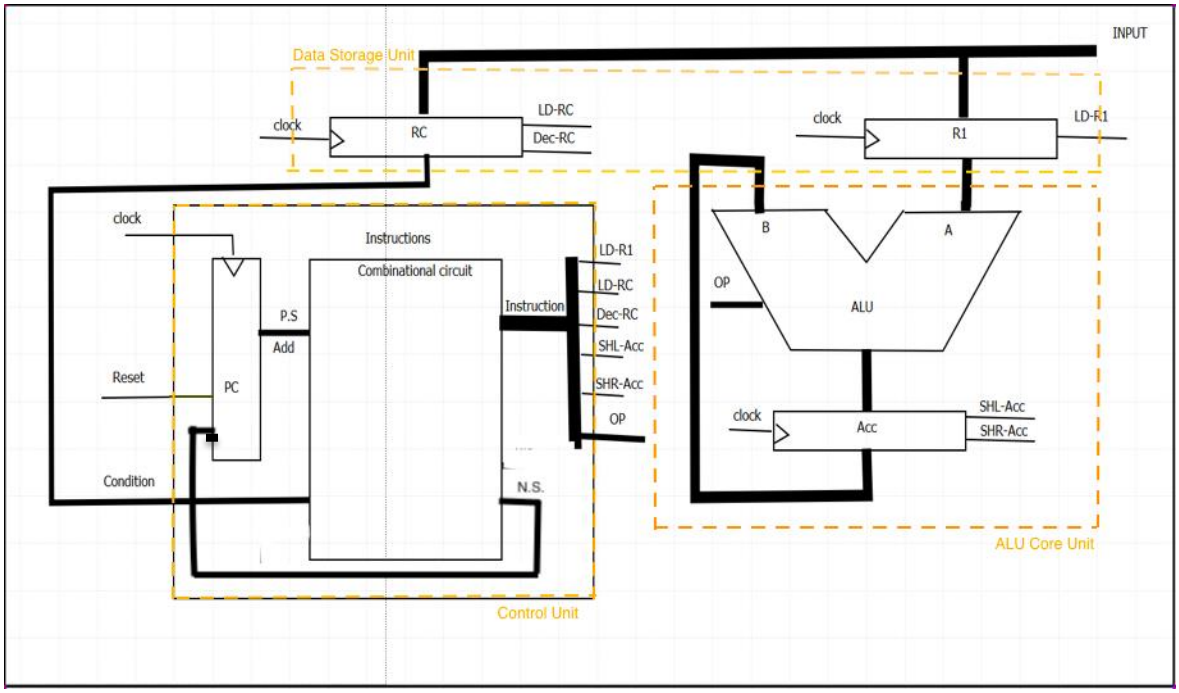


Fig.1 SIMPLE PROCESSOR Organization

Lab Procedure:

This processor consists of different components that functions in specific sequence to generate the desired output according to the instruction being executed. A processor is usually divided to 4 distinct sub-units Memory Unit, Control Unit, Data storage and ALU core unit. The Memory Unit performs the fetching of instructions. Data storage unit stores data from input switches and results of ALU unit to registers R1, RC and Acc. The ALU Core performs the arithmetic and logical operations on desired inputs and produces the required outputs. The Control unit decode the instruction and activates the control signals to execute it. In this project, we will be implementing distinct tasks by varying logic of control unit.

Description of system functioning and implementation details:

Part 1: Data Storage Unit

Input data is entered in the simulation phase through the Quartus II waveform editor. The input data bus is connected to two 8-bits registers namely R1 (Data register) and RC (Control Register). The data from input bus is loaded into either register R1 or register RC depending on the control signals from the control unit. The control signals associated with storing input data are as follows:

- LD-R1 => 1: Load register R1 from input bus
- LD-RC => 1: Load register RC from input bus
- Dec-RC => 1: Decrement content of register RC by 1.

Suggested implementation for register R1 is as follows:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
use IEEE.numeric_std.all;
ENTITY RC IS
PORT ( D: IN unsigned(7 DOWNTO 0) ;
LD_RC, Dec_RC : IN STD_LOGIC ;
Q: OUT unsigned(7 DOWNTO 0) );
END RC;

ARCHITECTURE Behavior OF RC IS
Signal D_signal,Q_signal: unsigned(7 DOWNTO 0);
BEGIN
D_signal <= D;
PROCESS (LD_RC, Dec_RC)
BEGIN

--- Insert your code here
-
END PROCESS ;
Q <= Q_signal;
END Behavior ;
```

Suggested implementation for register RC is as follows:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
use IEEE.numeric_std.all;
ENTITY RC IS
PORT ( D: IN unsigned(7 DOWNTO 0) ;
LD_RC, Dec_RC : IN STD_LOGIC ;
Q: OUT unsigned(7 DOWNTO 0) );
END RC;

ARCHITECTURE Behavior OF RC IS
Signal D_signal,Q_signal: unsigned(7 DOWNTO 0);
BEGIN
D_signal <= D;
PROCESS (LD_RC, Dec_RC)
BEGIN

--- Insert your code here
-
END PROCESS ;
Q <= Q_signal;
END Behavior ;
```

Part 2: Arithmetic Logic Unit Core

The heart of every processor unit is the ALU core where all arithmetic and logical operations are to be implemented and applied as required. In this part students are required to implement all functionalities and operations using VHDL syntax compatible with Altera FPGA boards. ALU core consists of two components ALU and accumulator (ACC). ACC is a register in which intermediate arithmetic and logic results are stored and fed back to the ALU.

The ALU will take two 8-bit inputs (A and B) and a 4-bit opcode from Control unit. The opcode input from controller unit is the operation-selector signal, deciding the operation that is to be applied on the inputs A and B based on the instruction being executed.

Suggested implementation for ALU is as follows:

```
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ALU IS
Port (
A, B : in STD_LOGIC_VECTOR(7 downto 0); -- 2 inputs 8-bit
ALU_Opcode : in STD_LOGIC_VECTOR(3 downto 0); -- 1 input 4-bit for selecting function
Output_bus : out STD_LOGIC_VECTOR(7 downto 0)); -- control signals
end ALU;
architecture Behavioral of ALU is
signal ALU_Result : std_logic_vector (7 downto 0);
begin
process(A,B,ALU_Opcode)
begin

case(ALU_Opcode) is
when "0000" => --Load R1 from input_bus
       ALU_Result <= B; -- ALU idle
when "0001" => -- Load RC from input_bus
       ALU_Result <= B;
when "0010" => -- ACC = ACC + R1
       ALU_Result <= A + B ;
when "0011" => --Acc= ACC-R1
       ALU_Result <= A - B ;
when "0100" => -- Acc=Acc + 1
       ALU_Result <= B + 1;
when "0101" => -- Acc= Acc -1
       ALU_Result <= B - 1;
when "0110" => -- RC= RC-1
       ALU_Result <= B;
when "0111" => -- Acc=Acc&R1
       ALU_Result <= A and B;
when "1000" => -- Acc | R1
       ALU_Result <= A or B;
when "1001" => -- Acc XOR R1  A or B;
       ALU_Result <= A xor B;
when "1010" => -- Acc NAND R1
       ALU_Result <= A nand B;
when "1011" => --- Shift Left Acc
       ALU_Result <= B;
when "1100" => --- Shift Right Acc
       ALU_Result <= B;
when "1101" => ---  branch to start if RC !=0; goto state
       ALU_Result <= B;
when "1110" => --- load the content of ACC to the outout bus
       ALU_Result<= B;-- ALU out
when "1111" => --- not defined
       ALU_Result <= B;
end case;
end process;
Output_bus <= ALU_Result;

end Behavioral;
```

Suggested implementation for ACC is as follows:

```
  LIBRARY ieee ;
  USE ieee.std_logic_1164.all ;
  use ieee.numeric_std.all ;
ENTITY ACC IS
PORT (D: IN unsigned(7 downto 0) ;
 SHL_ACC,SHR_ACC,Clock,Reset : IN STD_LOGIC ;
 Q: OUT unsigned(7 DOWNTO 0) );
 END ACC ;

ARCHITECTURE Behavior OF ACC IS
 signal store_signal : unsigned(7 downto 0) := (others => '0');
BEGIN
PROCESS (SHL_ACC, Clock, Reset, SHR_ACC,D)
 BEGIN
IF Reset = '0' then
    store_signal <= "00000000";
ELSIF((Reset = '1') and (falling_edge(Clock)))then --raising edge of clock
 store_signal<= D;
    IF SHL_ACC = '1' THEN
       store_signal <= store_signal sll 1 ;
    END IF;
    IF SHR_ACC = '1' THEN
       store_signal <= store_signal srl 1 ;
    END IF ;
END IF;
END PROCESS ;
Q <= store_signal;
 END Behavior ;
```

Part 3: Control Unit

The Control unit decides the microcode that is to be delivered to the ALU core and will act as the operations-selector for the ALU core. The Control Unit produces an output OP, which is passed to ALU core as the operations selector. This component consists of program counter (PC) and combinational circuit. The control unit uses Finite State Machine (FSM) to implement PC and combinational circuit is an instruction memory. The details of instruction set to be implemented by control unit are given in following table.

Instruction Set

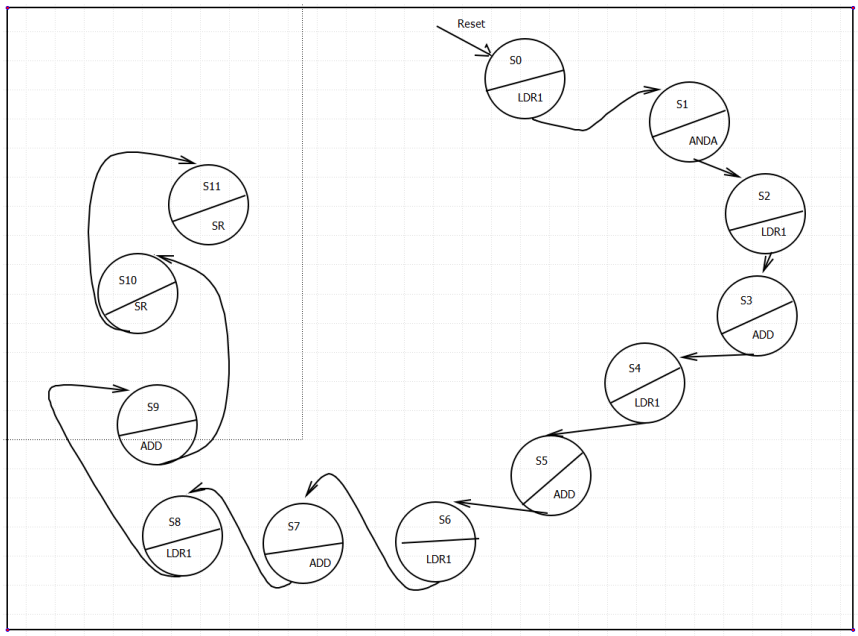
Mnemonics	Description	Opcode
Register Transfer		
LDR1	Load R1 from input; R1=INPUT	0
LDRC	Load RC from input; RC= INPUT	1
Arithmetic Operations:		
ADDA	Add R1 to Acc; Acc= Acc + R1	2
SUBA	Sub R1 from Acc; Acc= ACC-R1	3
INCA	Increment Acc; Acc=Acc + 1	4
DECA	Dec Acc; Acc= Acc -1	5
DEC-RC	Dec RC; RC= RC-1	6
Logic Operations		
ANDA	And Acc with R1; Acc=Acc&R1	7
ORA	Or Acc with R1; Acc = Acc | R1	8
XORA	XOR Acc with R1; Acc =Acc XOR R1	9
NAND	Invert Acc; Acc = Acc NAND R1	A
SLA	Shift Left Acc; Acc=Acc*2	B
SRA	Shift Right Acc; Acc = Acc/2	C
Control flow		
BNEQZ	branch to start if RC !=0; goto state	D

Task-1: Calculate average of 4 numbers entered from input x1, x2, x3, x4

Instruction Sequence to execute the task-1

Sequence No.	Instruction	Result from execution of the instruction
1	LDR1 0	R1 =0;
2	ANDA	Acc=0
3	LDR1 X1	R1=X1
4	ADD	Acc=X1
5	LDR1 X2	R1=X2
6	ADD	Acc= X1+X2
7	LDR1 X3	R1=X3
8	ADD	Acc=X1+X2+X3
9	LDR1 X4	R1=X4
10	ADD	ACC=X1+X2+X3+X4
11	SR	Acc= (X1+X2+X3+X4)/2
12	SR	Acc= (X1+X2+X3+X4)/4

FSM implementation of PC for task-1



Suggested implementation of PC and instruction memory for task-1 is as follows: *Please note that next two figures are part of a single vhdl code file. Please focus on bottom of first figure and top of second figure to find continuation between them.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PC_comb_ckt_example1 IS
PORT (Clock, Reset : IN STD_logic;
z : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
RC : IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- used for branch instruction
LD_R1, LD_RC, Dec_RC, SHL_Acc, SHR_Acc, ACC_output_bus_select : out std_logic);
-- RC_select = 1 --> input connected to RC
-- RC_select = 0 --> input connected to R1
-- A_select = 1 --> ACC connected to input A of ALU
-- A_select = 0 --> ACC connected to output bus connected to 7-segment signal
END PC_comb_ckt_example1;
ARCHITECTURE Behavior OF PC_comb_ckt_example1 IS
TYPE State_type IS (S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10);
SIGNAL y: State_type;
SIGNAL control_signal: STD_LOGIC_VECTOR(9 DOWNTO 0);
BEGIN
PROCESS (Reset, Clock)
BEGIN
IF Reset='0' THEN
y<=S0;
ELSIF(Clock'EVENT AND Clock='1') THEN
CASE y IS
WHEN S0=> y <= S1;
WHEN S1=> y <=S2;
WHEN S2=> y <=S3;
WHEN S3=> y <=S4;
WHEN S4=> y <=S5;
WHEN S5=> y<=S6;
WHEN S6=> y<=S7;
WHEN S7=>  y<=S8;
WHEN S8=>  y<=S9;
WHEN S9=>  y<=S10;
WHEN S10=> y<=S0;
END CASE;
END IF;
END PROCESS;
PROCESS(y)
BEGIN
CASE y IS
WHEN S0=>
control signal <= "0000010000"; --Load R1 from input X1
```

```
WHEN S9=>  y<=S10;
WHEN S10=> y<=S0;
END CASE;
END IF;
END PROCESS;
PROCESS(y)
BEGIN
CASE y IS
WHEN S0=>
control_signal <= "0000010000"; --Load R1 from input X1
WHEN S1=>
control_signal <= "0000000010"; -- Acc = Acc + R1
WHEN S2=>
control_signal <= "0000010000"; --Load R1 from input X2, set input to X2
WHEN S3=>
control_signal <= "0000000010"; -- Acc = Acc + R1
WHEN S4=>
control_signal <= "0000010000"; --Load R1 from input X3
WHEN S5=>
control_signal <= "0000000010"; -- Acc = Acc + R1
WHEN S6=>
control_signal <= "0000010000"; --Load R1 from input X4
WHEN S7=>
control_signal <= "0000000010"; -- Acc = Acc + R1
WHEN S8=>
control_signal <= "0100000000"; --SRA
WHEN S9=>
control_signal <= "0100000000"; --SRA
WHEN S10=>                    -- Connect output of ACC to 7 segment display
control_signal <= "1000000000";

END CASE;
END PROCESS;

z <= control_signal(3 DOWNTO 0);
LD_R1 <= control_signal(4);
LD_RC <= control_signal(5);
Dec_RC <= control_signal(6);
SHL_Acc <= control_signal(7);
SHR_Acc <= control_signal(8);
ACC_output_bus_select <= control_signal(9);
END Behavior;
```

Task- 2:

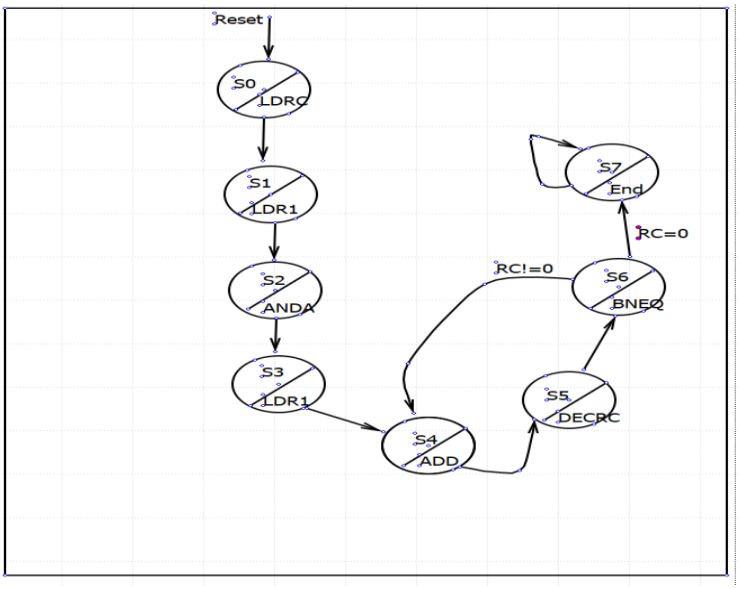
Executing a Loop for number of times. For example

$$for\ (i = 5\ ;\ i \geq 0\ ;\ i - -)\ \{result = result + x\}$$

Instruction Sequence to execute the task-2

Sequence No.	Instruction	Result from execution of the instruction
1	LDRC 5	RC =5;
2	LDR1 0	R1=0
3	ANDA	ACC=ACC and 0; ACC = 0;
4	LDR1 X	R1= X
5	LOOP: ADD	ACC = ACC+R1
6	DECRC	RC = RC-1;
7	BNEQZ LOOP	PC = address of “LOOP”

FSM implementation of PC for task-2



Suggested implementation of PC and instruction memory for task-2 is as follows: *Please note that next two figures are part of a single vhdl code file. Please focus on bottom of first figure and top of second figure to find continuation between them.*

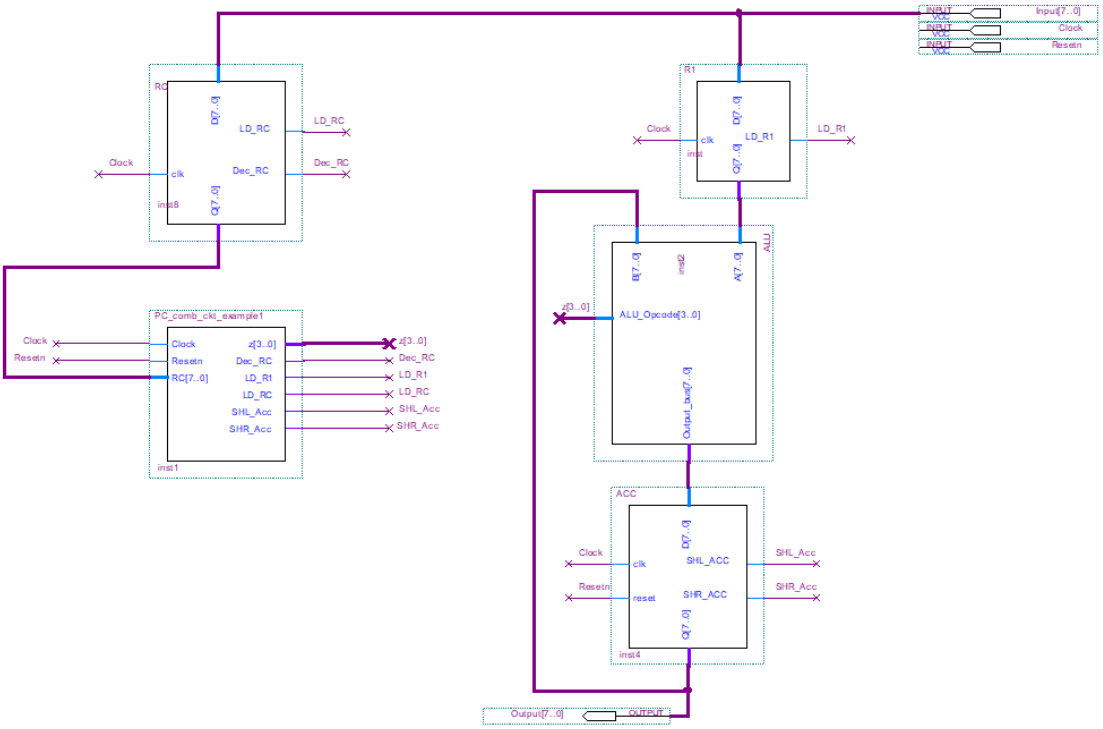
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY PC_comb_ckt_example1 IS
PORT (Clock, Reset : IN STD_logic;
  z : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
  RC : IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- used for branch instruction
 LD_R1, LD_RC, Dec_RC, SHL_Acc, SHR_Acc, ACC_output_bus_select : out std_logic);
 END PC_comb_ckt_example1;
ARCHITECTURE Behavior OF PC_comb_ckt_example1 IS
 TYPE State_type IS (S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10);
 SIGNAL y: State_type;
 SIGNAL control_signal: STD_LOGIC_VECTOR(9 DOWNTO 0);
BEGIN
PROCESS (Reset, Clock)
 BEGIN
IF Reset='0' THEN
 y<=S0;
ELSIF(Clock'EVENT AND Clock='1') THEN
CASE y IS
 WHEN S0=> y <= S1;
 WHEN S1=> y <=S2;
 WHEN S2=> y <=S3;
 WHEN S3=> y <=S4;
 WHEN S4=> y <=S5;
 WHEN S5=> y<=S6;
 WHEN S6=> y<=S7;
 WHEN S7=>  y<=S8;
 WHEN S8=>  y<=S9;
 WHEN S9=>  y<=S10;
 WHEN S10=> y<=S0;
 END CASE;
 END IF;
 END PROCESS;
PROCESS(y)
 BEGIN
CASE y IS
 WHEN S0=>
 control_signal <= "0000010000"; --Load R1 from input X1
 WHEN S1=>
 control_signal <= "0000000010"; -- Acc = Acc + R1
 WHEN S2=>
 control_signal <= "0000010000"; --Load R1 from input X2, set input to X2
 WHEN S3=>
END PROCESS;
PROCESS(y)
BEGIN
CASE y IS
WHEN S0=>
control_signal <= "0000010000"; --Load R1 from input X1
WHEN S1=>
control_signal <= "0000000010"; -- Acc = Acc + R1
WHEN S2=>
control_signal <= "0000010000"; --Load R1 from input X2, set input to X2
WHEN S3=>
control_signal <= "0000000010"; -- Acc = Acc + R1
WHEN S4=>
control_signal <= "0000010000"; --Load R1 from input X3
WHEN S5=>
control_signal <= "0000000010"; -- Acc = Acc + R1
WHEN S6=>
control_signal <= "0000010000"; --Load R1 from input X4
WHEN S7=>
control_signal <= "0000000010"; -- Acc = Acc + R1
WHEN S8=>
control_signal <= "0100000000"; --SRA
WHEN S9=>
control_signal <= "0100000000"; --SRA
WHEN S10=>                      -- Connect output of ACC to 7 segment display
control_signal <= "1000000000";

END CASE;
END PROCESS;

z <= control_signal(3 DOWNTO 0);
LD_R1 <= control_signal(4);
LD_RC <= control_signal(5);
Dec_RC <= control_signal(6);
SHL_Acc <= control_signal(7);
SHR_Acc <= control_signal(8);
ACC_output_bus_select <= control_signal(9);
END Behavior;
```

Displaying Output

The final result of the task which corresponds to the content of ACC during final state of task sequence should be clearly displayed in the simulation waveforms. Please refer to the final schematic on the following page.


Input[7..0]
Clock
Resetin
RC
D[7..0]
LD_RC
Clock
clk
Dec_RC
Q[7..0]
inst8
R1
D[7..0]
Clock
clk
LD_R1
LD_R1
Q[7..0]
inst
ALU
B[7..0]
inst2
A[7..0]
z[0..0]
ALU_Opcode[3..0]
Output_bus[7..0]
PC_comb_clk_example1
Clock
Resetin
RC[7..0]
z[3..0]
Dec_RC
LD_R1
LD_RC
SHL_Acc
SHR_Acc
z[3..0]
Dec_RC
LD_R1
LD_RC
SHL_Acc
SHR_Acc
inst1
ACC
D[7..0]
Clock
clk
SHL_ACC
SHL_Acc
Resetin
reset
SHR_ACC
SHR_Acc
Q[7..0]
inst4
Output[7..0]
OUTPUT

Assigned Tasks: -

Logic Operations: -

1-Enter X1, X2, X3, X4 from switches then perform the following: -
Result= (X1 AND X2) OR X3; Sum of products function

2-Enter X1, X2, X3, X4 from switches then perform the following: -
Result= (X1 OR X2) AND X3; Product of sums function

3-Enter X, Y from switches then perform the following: -
Result= (X XNOR Y); A comparator if(X=Y) Result=1;

Arithmetic Operations: -

1-Enter X1, X2, X3, X4 from switches then perform the following: -
Result= 2*(X1+X2+X3+X4); ADD then SHL

2-Enter X1, X2, Y1, Y2 from switches then perform the following: -
Result= (X1-Y1) + (X2-Y2); Difference between two sets of numbers

3-Enter X1, X2, X3, X4 from switches then perform the following: -
Result= (X1+1) + (X2+1) + (X3+1) + (X4+1); Inc each number and find total

4-Enter X1, Y1, X2, Y2 from switches then perform the following: -
Result= 2*(X1-Y1) +2*(X2-Y2); Calculate double the difference between two sets