

Design of a Simple General Purpose Processor

Objective

This objective of this laboratory experiment is to design and construct a simple Processor/CPU in VHDL

Upon completion of this laboratory, the student should be able to:

- Design and build all functions of a simple Processor.
- Design, simulate and program a simple Processor using VHDL on an Altera FPGA board (Based on Functional Simulation in Quartus Simulator).
- Output the results on the LED Display.

Part I: Procuring data from same DIP switches for processing

The ALU performs a set of functions on two 8-bit inputs (A and B). In our design, these values must be set by the user and changed dynamically at wish.

After completing the design follow these steps:

1. Import and assign all pins. Refer to the pin instructions manual (provided in earlier labs).
2. Compile, simulate, and test your design.
3. Download the program on the Altera board. You can input a value on the DIP switch and By assigning the inputs to the switches and output to the LED's.
4. Show your TA, the working program, simulation results and implementation results.

Part II: Description of a simple ALU processing core:

The processor of a computer is usually divided into at least four components, the *control* unit which performs the fetching the signals/instructions, the *bus* which controls access to the data and address bus, *latches* for temporary storage, and the *arithmetic/logic unit (ALU)* that performs arithmetic and logical operations. In this laboratory we will focus on developing all the four components. The processor will be tested by performing the functions of all. The heart of every computer is an Arithmetic Logic Unit (ALU). This is the part of the computer which performs arithmetic operations on numbers, *e.g.* addition, subtraction, *etc.* In this lab you will use VHDL to implement an ALU with eight functions.

Instructions:

- The ALU must consist of two 8-bit input values and 8-bit select input that chooses which operation is to be performed on the two values, as shown below:
- Your ALU will perform eight functions on two 8-bit inputs (A and B). These inputs could represent either unsigned numbers or simply bit patterns. The ALU will generate a 4-bit output (R).
- To select which of the eight functions to implement you will need 8-bit select inputs. A block diagram is shown in Figure 1.
 - 1) Synthesize, simulate and verify the functionality of the ALU core.

- 2) Create/update symbol from schematic for being used as a functional block for use later in the schematic design of your simple processor.

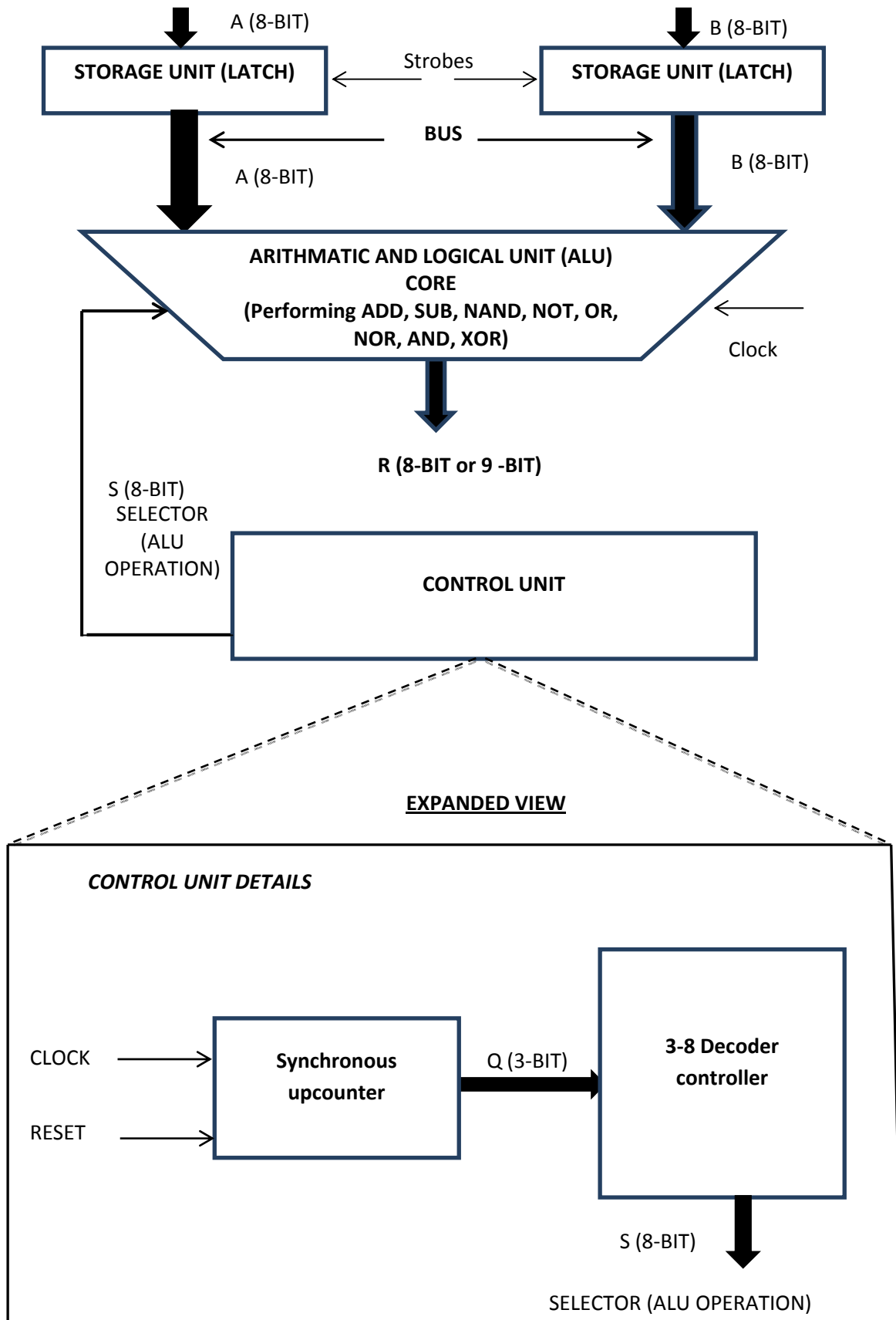


Figure1. The Block Diagram of a simple 8-bit Processor

Part III: Controller Design: The microcode's controlling the ALU core is given as follows in Table 1. Implement the microcode in VHDL to design the decoder (which in turn is being controlled by upcounter). The microcode's act as the selector for the ALU functions/instructions which produces the respective output as the microcode values change.

Table1 Microcode's generated by decoder

Number of functions	Microcode's	Instructions/Functions (arithmetic or logical)
1	00000001	Add (Addition)
2	00000010	Sub (Subtraction)
3	00000100	NOT (Inverter)
4	00001000	NAND (Invert and-ing)
5	00010000	NOR (Invert or-ing)
6	00100000	AND (and-ing)
7	01000000	OR (or-ing)
8	10000000	XOR (ex-oring)

Part III: a) Description of a 3-8 Decoder unit (controlling the selector of the ALU):

A 3 to 8 decoder consists of three inputs and eight outputs, truth table and symbols of which is shown below. The 3-8 Decoder generates 8 different output bits to control the 8 different functions of an ALU. The truth table and symbol (Figure.2) are provided below:

The Black box diagram of a 3-8 Decoder unit:

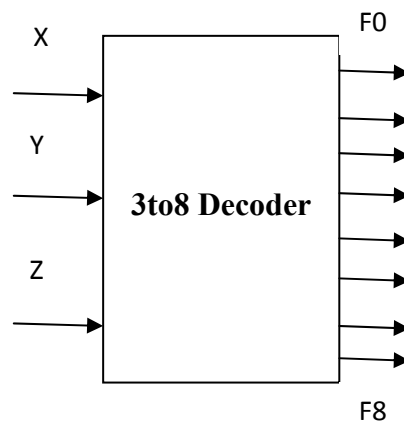


Figure.2

Truth Table:

X	Y	Z	F0	F1	F2	F3	F4	F5	F6	F7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Instructions:

- 1) From the truth Table above, draw the circuit diagram of the 3-8 decoder (not using two 2-4 decoders) and implement it in the Quartus schematic.
- 2) Synthesize, simulate and verify the functionality of the decoder unit.
- 3) Create/update symbol from schematic for being used as a functional block for use later in the schematic design of your simple processor.

Part III: b) Description of a synchronous upcounter (controlling the 3-8 Decoder)-

The Synchronous upcounter is a component of the controller and is synchronous with the clock. It is used in conjunction with the decoder to generate the 8 different output bits for 8 different functions (Each of the decoder output represents a function. In the first count, the first function F0 is active (which in turn drives the first function of ALU e.g. add), in the second count, the second function F1 is active (which in turn drives the first function of ALU e.g. Subtract) and so on.

Instructions:

- 1) Use the VHDL code given in Figure below (Figure 3.) for designing the upcounter.
- 2) Synthesize, simulate and verify the functionality of the Upcounter unit.
- 3) Create/update symbol from schematic for being used as a functional block for use later in the schematic design of your simple processor.

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_unsigned.ALL;
```

```
Entity Synchronous_upcount is
port (reset, clock: IN std_logic;
      Q : BUFFER std_logic_vector (2 downto 0));
```

```
end Synchronous_upcount;
architecture behaviour of Synchronous_upcount is
begin
```

```

Synchronous_upcount: 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;

```

Figure 3

Part IV: b) Description of a Storage Elements (Latches with strobe):

The storage elements/latches are used to store temporarily the input values for 8-bit A and B. It holds the values until the strobe instructs (at logic 1) them to release the data for use by ALU core.

Instructions:

- 1) Use the VHDL code done in previous labs (and modify if required) for designing the latches.
- 2) Synthesize, simulate and verify the functionality of the latches.
- 3) Create/update symbol from schematic for being used as a functional block for use later in the schematic design of your simple processor.

Part V: Port mapping of all the components of a processor

Once all the symbols for the subunits (ALU core, Controller – Decoder and Upcounter, Registers and busses) have been generated, from its respective VHDL code/schematic, then the blocks needs to be port mapped in a final circuit.

Instructions:

- 1) Open a new schematic file and connect all the components of the processor (with proper data integrity) based on figure1 (complete block diagram). Name the busses/inputs as e.g. A[7..0] by right clicking on the bus symbol and going to properties for renaming the bus. Name the wires as e.g. Q[1] by right clicking on the wire symbol and going to properties for renaming the wire. The complete schematic must look similar to Figure I (in appendix)
- 2) Synthesize, simulate and verify the functionality of the complete processor circuit by generating the test bench waveform (A snapshot of a sample simulation result is given in Figure 4).
- 3) Show the results of the simulation in Binary format to the TA.

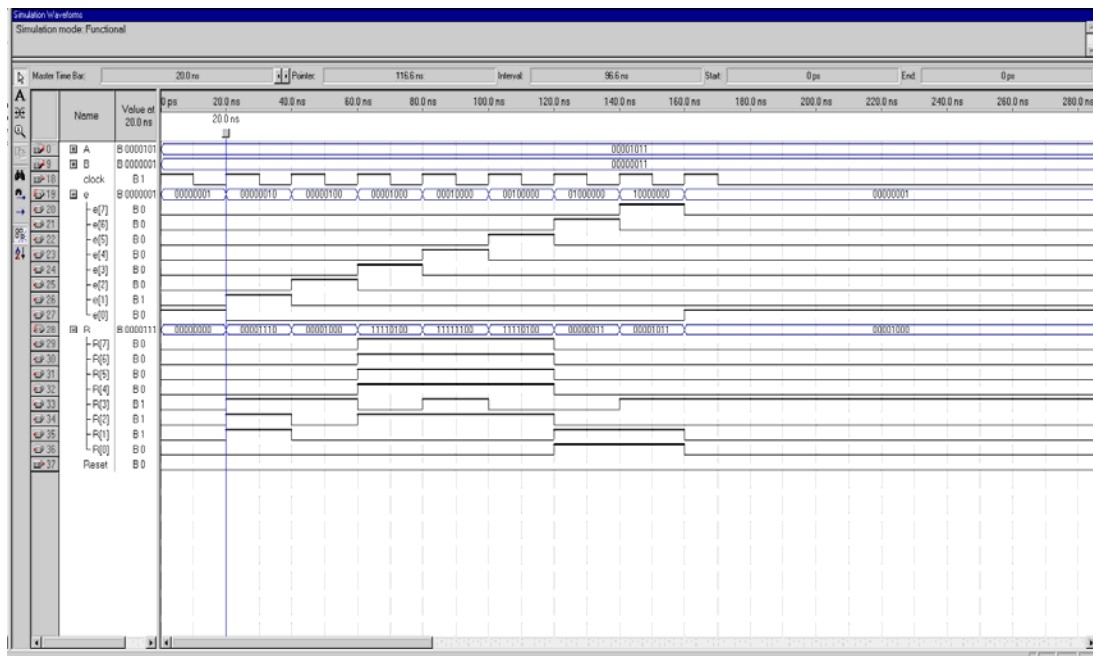


Figure4. The snapshot of a sample simulation result of the complete

Part VI: Implementation in Altera FPGA:

1) Compile your design

2) Assign the following signals to the dedicated I/O-pins of the FPGA chip on the prototype board (see lab 3 for the data sheets of the pin assignment and the FPGA chip on the board to know its name), using either *Pin assignments* menu (see Quartus Tutorial 2):

- a) A7, A6, A5, A4, A3, A2, A1, A0 to SW7, SW6, SW5, SW4, SW3, SW2, SW1, SW0. While, B7, B6, B5, B4, B3, B2, B1, B0 to SW15, SW14, SW13, SW12, SW11, SW10, SW9, SW8.
- b) Clock to a push button switch.
- c) R7, R6, R5, R4, R3, R2, R1, R0 to LEDG8, LEDG7, LEDG6, LEDG5, LEDG4, LEDG3, LEDG2, LEDG1, LEDG0.
- d) Reset to the toggle switch, SW [17]

Note: 1. All the LEDs of common anode type (active LOW)

3) Recompile the design and implement/program/download into the FPGA on the board.

4) Test your design and demonstrate results to the instructor (TA).

Part VII: Addressing the Problem exercise for the Projects

1) Follow the sample ALU template given in Figure II (in Appendix).

2) Fill the logic inside the 'case' statement according to your assigned group programs (problem set 2 ,3) and the assigned general program for all (problem set 1).

3) Once the general logic inside 'case' has been developed and the logics statements inside case need to be modified each time the custom program (assigned program: (problem set 2 and 3)) is made to implement.

Problem sets for the Projects:

1. **Each group should produce the output of the processor in LED's of the FPGA board in the sequence as provided with the gradual in clock cycles or application of new clock pulse** (Note: The output of the LED's will change on the rising edge of the clock pulse): i) addition ii) Subtraction iii) inverting (NOT), iv) NAND, v) NOR vi) AND vii) OR viii) XOR

For e.g. 1st clock cycle (cc)/period shows LEDs having the output of an addition followed with the output values of Subtraction in 2nd clock period, invert, nand, nor, and, or, xor in next consecutive clock cycles/periods (maintain the sequence). Students should fill the vacant portion in the template of the ALU provided in the appendix below to exhibit this pattern.

2. **Each group will be assigned a task/program by the TA from the list below. Note: the outputs should be generated in the sequence mentioned in the question.**

- 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.
- b) Take input two 8 bit numbers (A and B) to alternate between inverting output of A and B in consecutive clock cycle/periods.
- c) Take input two 8 bit numbers (A and B) to alternate between NAND for 1st 4 clock periods and OR outputs in remaining 4 clock periods.
- d) Take input two 8 bit numbers (A and B) to produce the OR output for 2 consecutive clock periods, XOR for next 2 consecutive clock periods, SUM for next 2 clock periods and INV for next 2 clock periods.
- e) Take input two 8 bit numbers (A and B) to produce no result for consecutive 7 clock cycles/clock periods and generate a SUM in the next clock period. The output pattern should repeat following the SUM.
- f) Take input two 8 bit numbers (A and B) to produce the OR output for 3 consecutive clock periods, NAND for next 3 consecutive clock periods and 'no result' for the remaining clock periods. (Note: NULL keyword does not produce no result. Use of some numeric value should be made to drive the output to no result i.e. LEDs should be turned off)
- g) Take input two 8 bit numbers (A and B) to generate the XOR result in the first clock period. Hold the result in the remaining 7 clock periods.

- h) Take input two 8 bit numbers (A and B) to generate the OR result followed by AND result in first 2 clock periods. Hold the result in the next 6 clock periods. Repeat the OR followed by AND result pattern.

3. Each group will be assigned a task/program by the TA from the list below. *Note: the outputs should be generated in the sequence mentioned in the question.*

- a) Take input two 8 bit numbers (A and B) to produce an alternate output of the single incremented values of A and single decremented value of A in consecutive clock periods.
- b) Take input two 8 bit numbers (A and B) to produce the sum in first clock period, followed by increment of A by 1, increment by 2, increment by 3, increment by 4, increment by 5, increment by 6 and increment by 7 in consecutive clock periods.
- c) Take input two 8 bit numbers (A and B) to produce the sum, difference, decrement of A by 2, logical shift right (*srl*) A by 3, rotate left A by 2 (*rol*), rotate right A by 2 (*ror*), logical shift left (*sll*) A by 2 and increment B by 1 in consecutive clock periods.
- d) Take input two 8 bit numbers (A and B) to generate the rotate left A by 2 (*rol*). Hold the output for the next 3 clock cycles before changing the output to produce the logical shift left (*sll*) A by 2 for the remaining 4 clock periods.
- 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.
- f) Take input two 8 bit numbers (A and B) to generate the decrement A by 2. Hold the output for the next 4 clock periods before changing the output to produce the increment A by 2 for the remaining clock periods.
- g) Take input two 8 bit numbers (A and B) to generate the rotate left A by 3 (*rol*) followed by rotate right A by 3 (*ror*). Alternate this output in the remaining clock periods.
- h) Take input two 8 bit numbers (A and B) to generate the logical shift left (*sll*) A by 3 (*sll*) followed by logical shift right by 3 (*srl*). Alternate this output in the remaining 6 clock periods.

FORMAL REQUIREMENTS FOR THE REPORT:

- 1. Brief the technical aspects/Abstract that you learnt through this lab.
- 2. Include step by step description of your work in this project lab.
- 3. Include all the VHDL codes that you coded and all the snapshot of the schematic circuit that you designed.

4. Include the simulation result of each of the blocks of the design that you built followed by inclusion of the complete simulation result for exercise 1, exercise 2 and exercise 3 assigned to each group.
5. Include the pin assignment details in the report and successful compilation report.

Note: Check Appendix Next Page

Appendix

1. Simple General Purpose Processor

The design of your designed processor should look similar to the figure below.

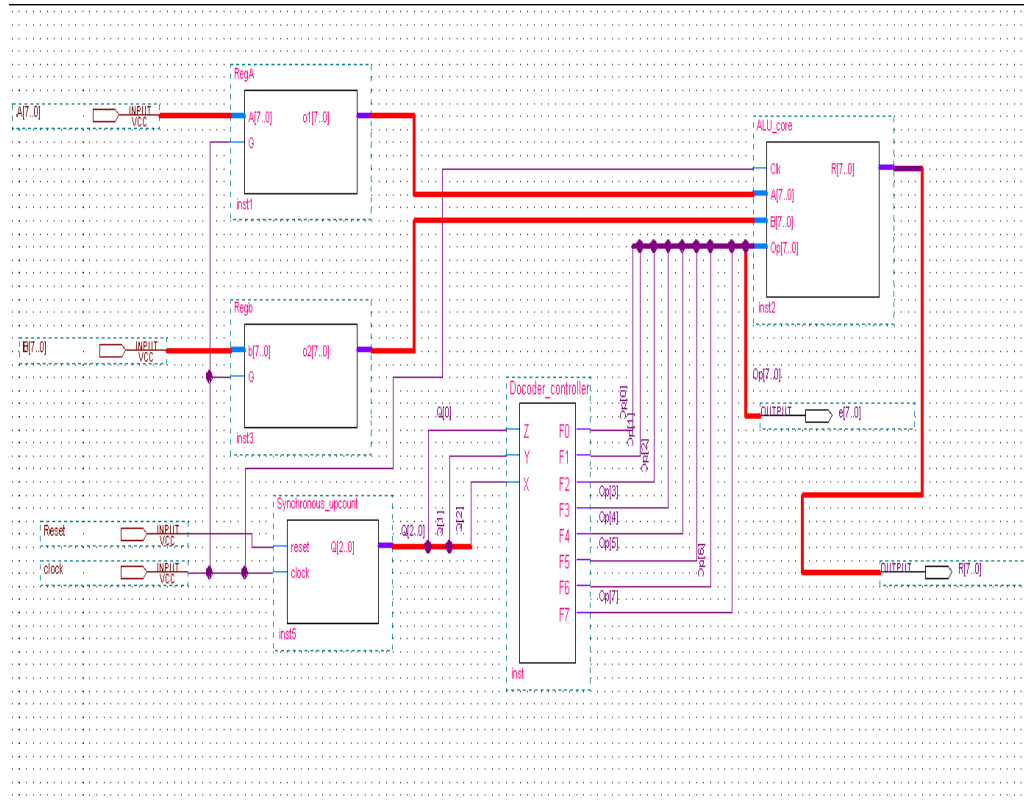


Figure I

2. Simple ALU processing core

The template for your ALU core is given below. You should fill up the logic for the vacant portion.

```
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
      R : out signed(7 downto 0) --output of ALU
    );
end ALU_core;
```

architecture Behavioral of ALU_core is

--temporary signal declaration.

signal Reg1,Reg2,Reg3 : signed(7 downto 0) := (others => '0');

begin

Reg1 <= A;

Reg2 <= B;

R <= Reg3;

process(Clk, Op)

begin

if(rising_edge(Clk)) then --Do the calculation at the positive edge of clock cycle.

case Op is

end case;

end if;

end process;

end Behavioral;

Figure II