Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Computer Organization and Architecture
Course Number:	COE628
Semester/Year (e.g.F2016)	W2024

Instructor:	Khalid Abdel Hafeez

Assignment/Lab Number:	3
Assignment/Lab Title:	Program Counter and Register Set Design

Submission Date:	2024/02/07
Due Date:	2024/02/08

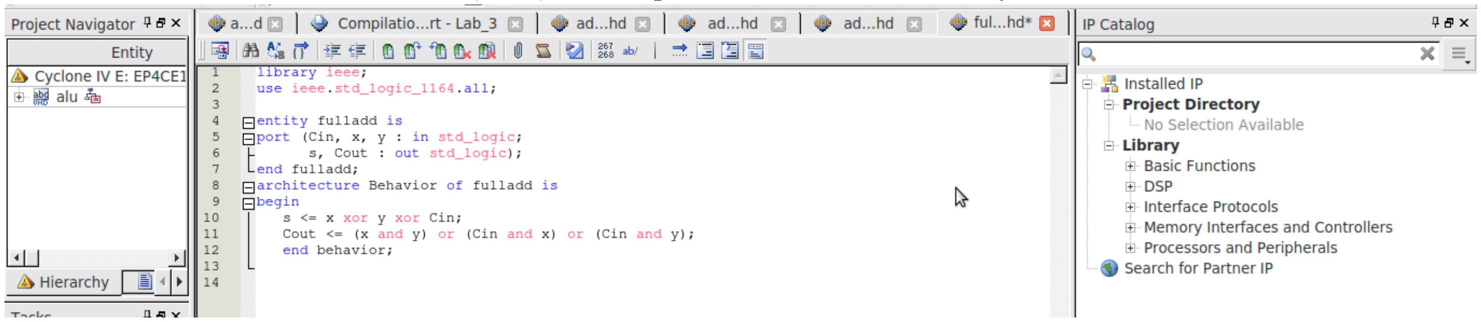
Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Sarim	Shahwar	501109286	12	SS

Lab Objective:

The ALU is a crucial part of the CPU architecture. The primary objective of this lab is to design, implement, simulate, and test the 32-bit ALU system created. The 32-bit ALU system is composed of 1-bit, 4-bit, 16-bit and 32-bit adders/subtractors, with the use of VHDL and the Waveform generator. These adders/subtractors are a fundamental component in the architecture of a 32-bit ALU system. The lab objective is to create a fully functional ALU component, convert the component into an 8-bit operand, and test the ALU, with the use of various LED's and switches, on the FGPA board.

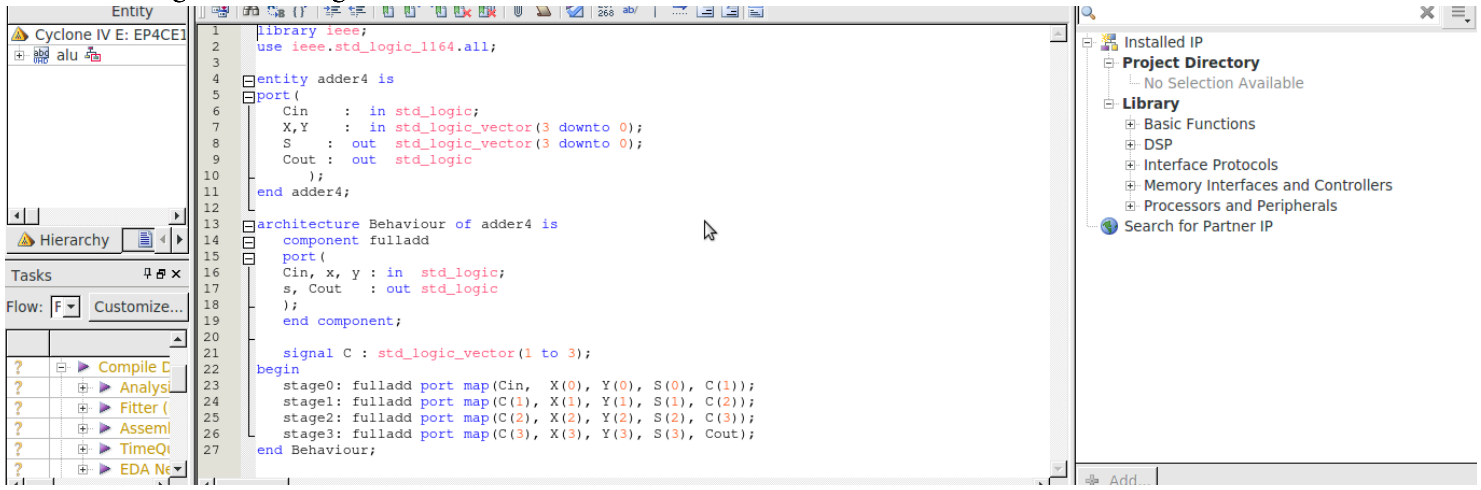
Experiment Details:

1-bit adder: This is the 1-bit adder function VHDL Code. The architecture of this code uses basic logical operations to calculate the 'Sum' as the exclusive OR (XOR) of the inputs and the 'Cout' as the carry-out of the addition



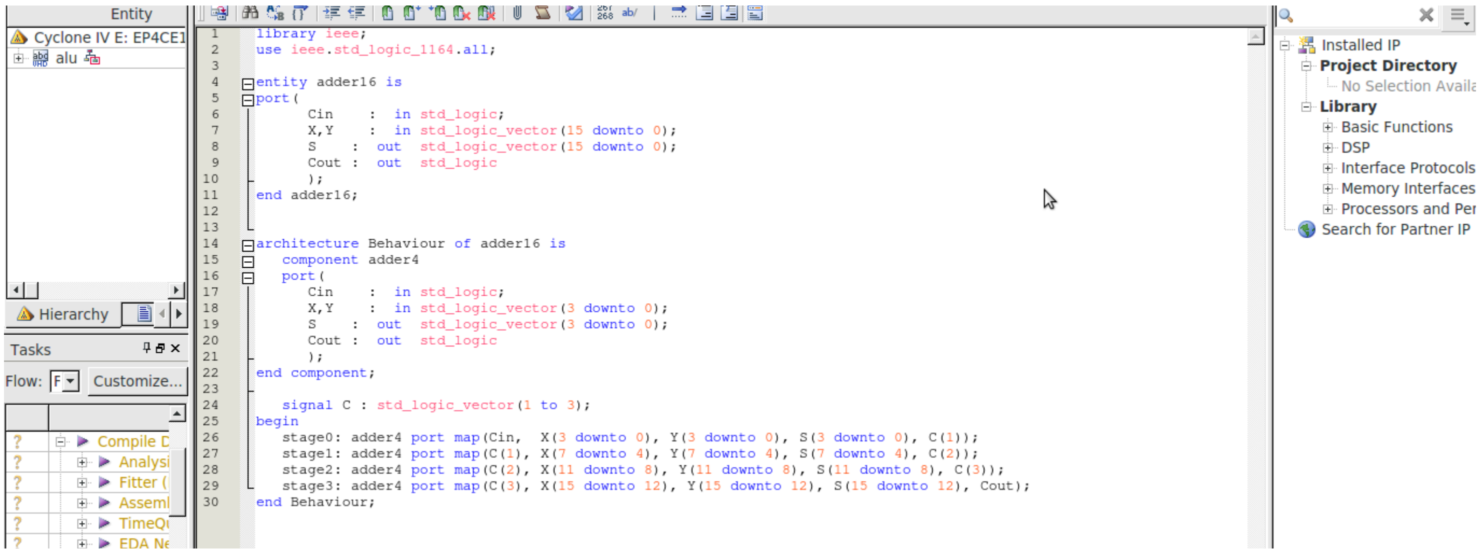
```
library ieee;
use ieee.std_logic_1164.all;

entity fulladd is
port (Cin, x, y : in std_logic;
      s, Cout : out std_logic);
end fulladd;
architecture Behavior of fulladd is
begin
    s <= x xor y xor Cin;
    Cout <= (x and y) or (Cin and x) or (Cin and y);
    end behavior;
```

4-bit adder: This is the 4-bit adder function VHDL Code. The Four full adders are instantiated to create the 4-bit adder, with carries chaining between stages



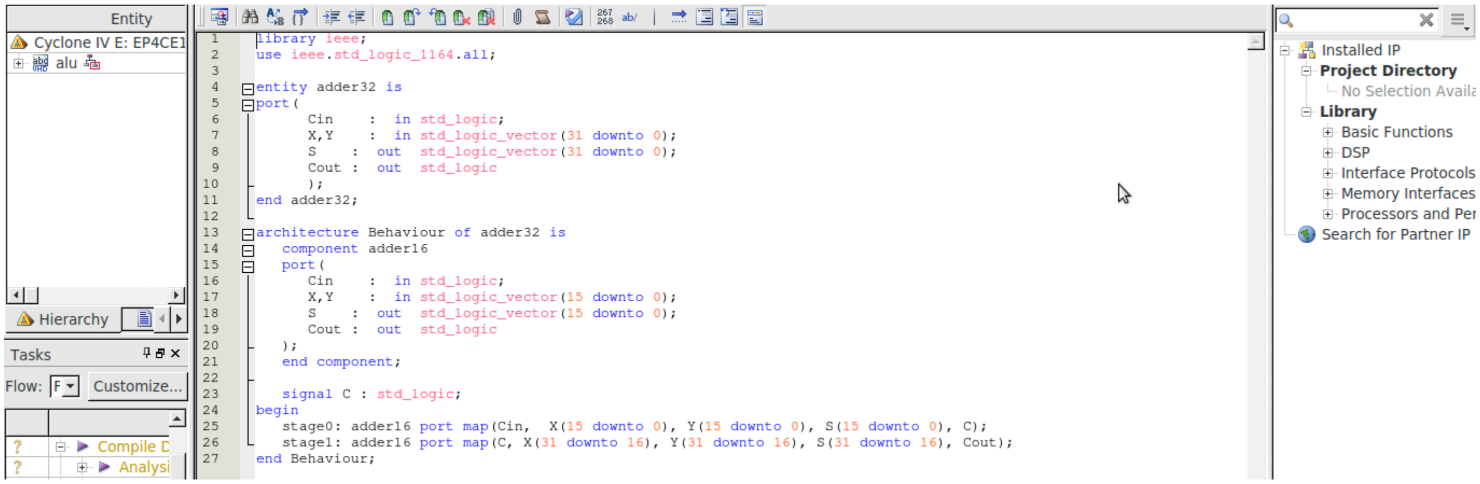
```
library ieee;
use ieee.std_logic_1164.all;

entity adder4 is
port (
    Cin    :  in std_logic;
    X,Y    :  in std_logic_vector(3 downto 0);
    S     :  out  std_logic_vector(3 downto 0);
    Cout :  out  std_logic
      );
end adder4;

architecture Behaviour of adder4 is
    component fulladd
    port (
    Cin, x, y : in  std_logic;
    s, Cout   : out std_logic
    );
    end component;

    signal C : std_logic_vector(1 to 3);
begin
    stage0: fulladd port map(Cin,  X(0), Y(0), S(0), C(1));
    stage1: fulladd port map(C(1), X(1), Y(1), S(1), C(2));
    stage2: fulladd port map(C(2), X(2), Y(2), S(2), C(3));
    stage3: fulladd port map(C(3), X(3), Y(3), S(3), Cout);
end Behaviour;
```

16-bit adder: This is the 16-bit adder function VHDL Code. The VHDL code is for an entity named 'adder16', which implies that it is a 16-bit adder, including a carry-in input (Cin), two 16-bit input vectors (X and Y) for the numbers to be added, a 16-bit output vector (Sum) for the resulting sum of X and Y, and a carry-out output (Cout). The carries between the stages would change, allowing for the sequential addition of each 4-bit segment along with the carry, which is forwarded from the previous stage.



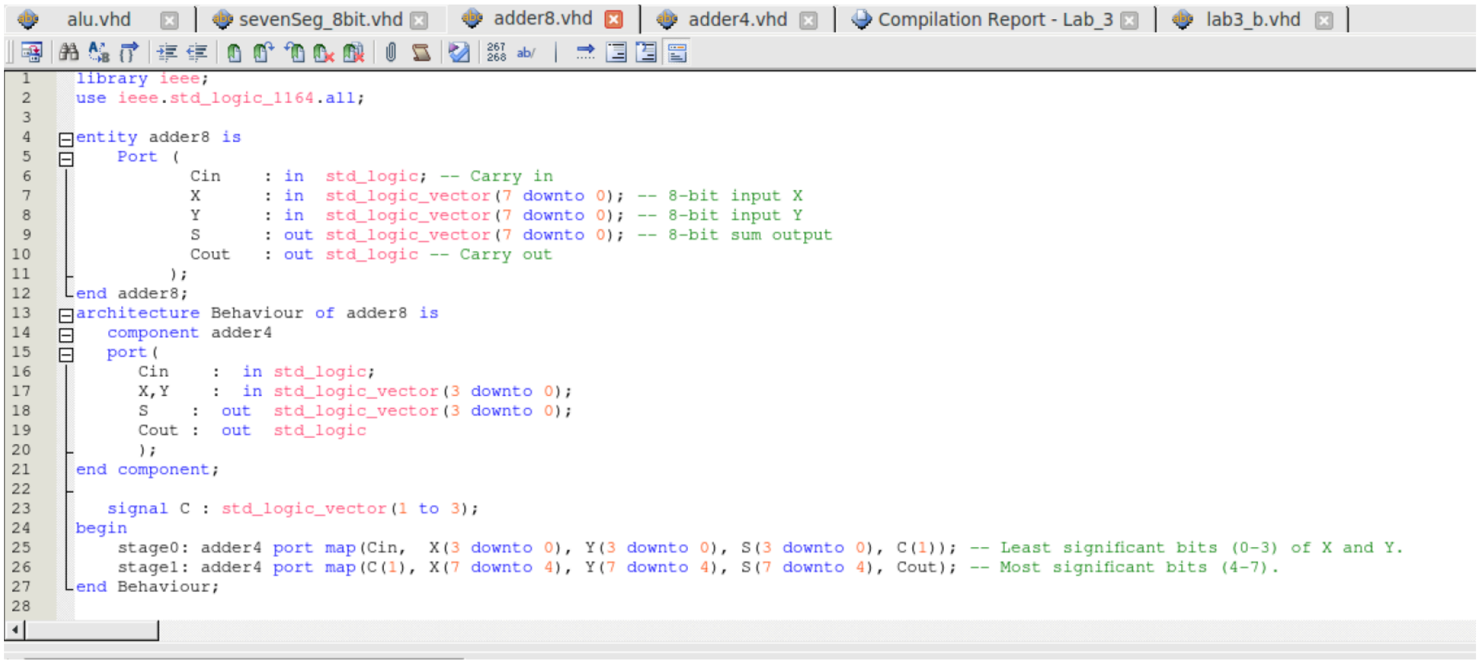
```
library ieee;
use ieee.std_logic_1164.all;

entity adder16 is
port (
      Cin    :  in  std_logic;
      X,Y    :  in  std_logic_vector(15 downto 0);
      S    :  out  std_logic_vector(15 downto 0);
      Cout :  out  std_logic
      );
end adder16;


architecture Behaviour of adder16 is
   component adder4
   port (
      Cin    :  in  std_logic;
      X,Y    :  in  std_logic_vector(3 downto 0);
      S    :  out  std_logic_vector(3 downto 0);
      Cout :  out  std_logic
      );
end component;

   signal C : std_logic_vector(1 to 3);
begin
   stage0: adder4 port map(Cin,  X(3 downto 0), Y(3 downto 0), S(3 downto 0), C(1));
   stage1: adder4 port map(C(1), X(7 downto 4), Y(7 downto 4), S(7 downto 4), C(2));
   stage2: adder4 port map(C(2), X(11 downto 8), Y(11 downto 8), S(11 downto 8), C(3));
   stage3: adder4 port map(C(3), X(15 downto 12), Y(15 downto 12), S(15 downto 12), Cout);
end Behaviour;
```

32-bit adder/subtractor: This is the 32-bit adder/subtractor function VHDL Code. This includes a single-bit standard logic input for the carry-in (Cin), two 32-bit input vectors (X and Y) representing the operands to be added, and two outputs: a 32-bit output vector (Sum) for the addition result and a standard logic output (Cout) for the carry-out. The 32-bit adder is constructed by two 16-bit adders.



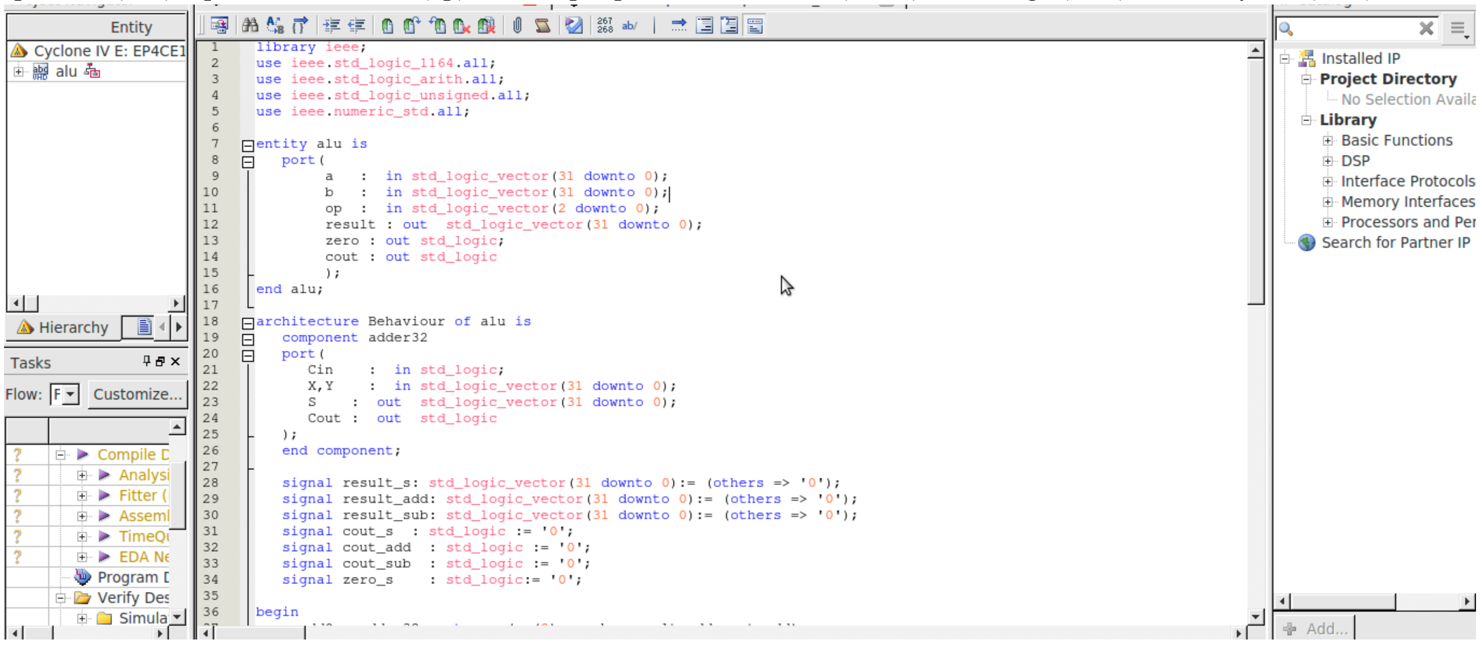
```
library ieee;
use ieee.std_logic_1164.all;

entity adder32 is
port (
      Cin    :  in  std_logic;
      X,Y    :  in  std_logic_vector(31 downto 0);
      S    :  out  std_logic_vector(31 downto 0);
      Cout :  out  std_logic
      );
end adder32;

architecture Behaviour of adder32 is
   component adder16
   port (
      Cin    :  in  std_logic;
      X,Y    :  in  std_logic_vector(15 downto 0);
      S    :  out  std_logic_vector(15 downto 0);
      Cout :  out  std_logic
   );
   end component;

   signal C : std_logic;
begin
   stage0: adder16 port map(Cin,  X(15 downto 0), Y(15 downto 0), S(15 downto 0), C);
   stage1: adder16 port map(C, X(31 downto 16), Y(31 downto 16), S(31 downto 16), Cout);
end Behaviour;
```

8-bit adder: This is the 8-bit adder function VHDL Code. This code was created for part b of this lab, as well as to test the ALU functions on the Cyclone IV FPGA Board. This function uses two 4-bit adders ('adder4' components). The 'adder8' entity has ports for carry-in, two 8-bit input vectors, an 8-bit output vector for the sum, and carry-out. The architecture 'Behaviour' includes component declarations for 'adder4' and an internal signal for a carry action.

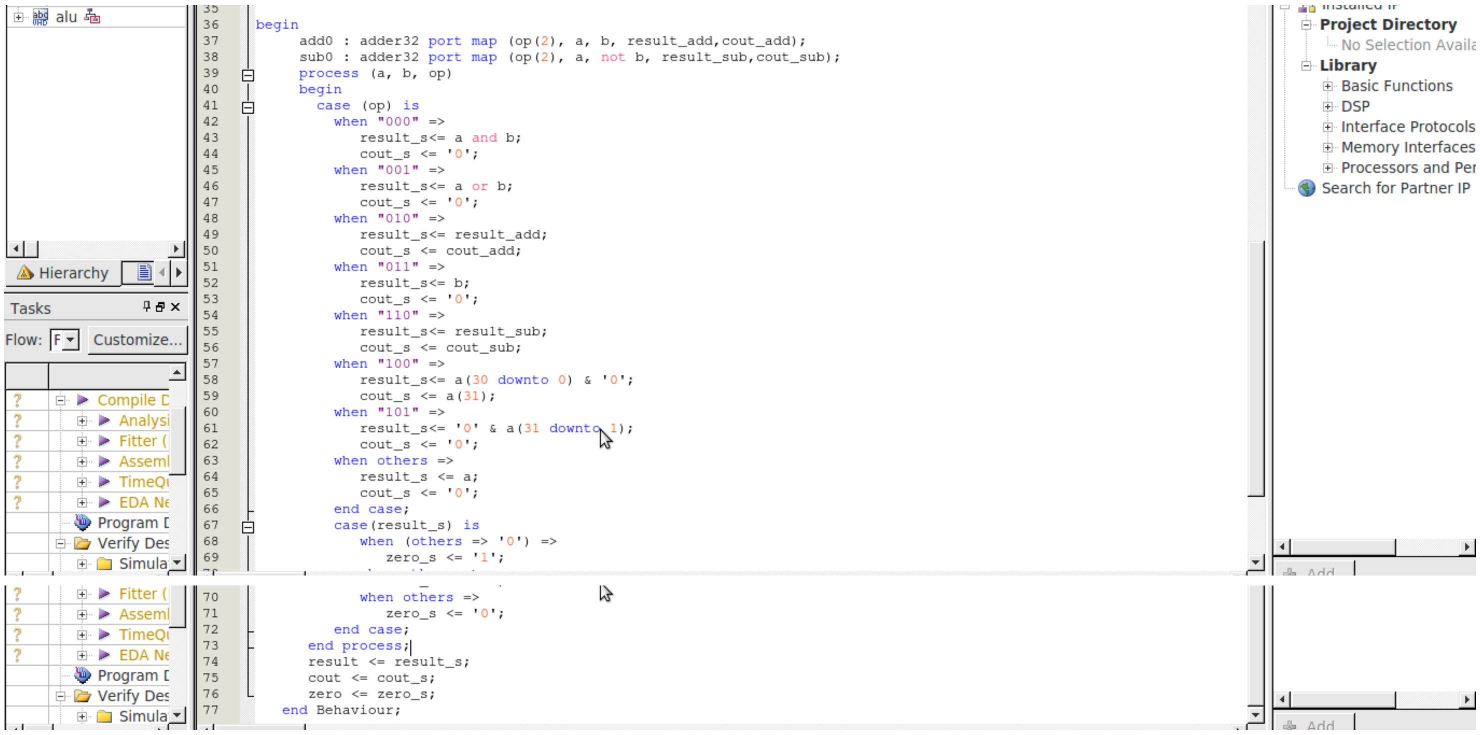


```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity adder8 is
    Port (
           Cin     : in  std_logic; -- Carry in
           X       : in  std_logic_vector(7 downto 0); -- 8-bit input X
           Y       : in  std_logic_vector(7 downto 0); -- 8-bit input Y
           S       : out std_logic_vector(7 downto 0); -- 8-bit sum output
           Cout    : out std_logic -- Carry out
         );
end adder8;
architecture Behaviour of adder8 is
   component adder4
   port(
      Cin    :  in std_logic;
      X,Y    :  in std_logic_vector(3 downto 0);
      S    :  out  std_logic_vector(3 downto 0);
      Cout :  out  std_logic
      );
end component;

   signal C : std_logic_vector(1 to 3);
begin
    stage0: adder4 port map(Cin,  X(3 downto 0), Y(3 downto 0), S(3 downto 0), C(1)); -- Least significant bits (0-3) of X and Y.
    stage1: adder4 port map(C(1), X(7 downto 4), Y(7 downto 4), S(7 downto 4), Cout); -- Most significant bits (4-7).
end Behaviour;

```

ALU: This is the full ALU program in VHDL. This code consists of the adders and subtractors sourced from the four adder/subtractor VHDL codes created previously. This includes the 1-bit, 4-bit, 16-bit, and 34-bit adder/subtractor functions. Functions such as AND, OR, addition, and subtraction operations on 32-bit system. The entity includes input ports for operands (a, b), operation selection (op), and output ports for the result (result), zero flags (zero), and carry out (cout).

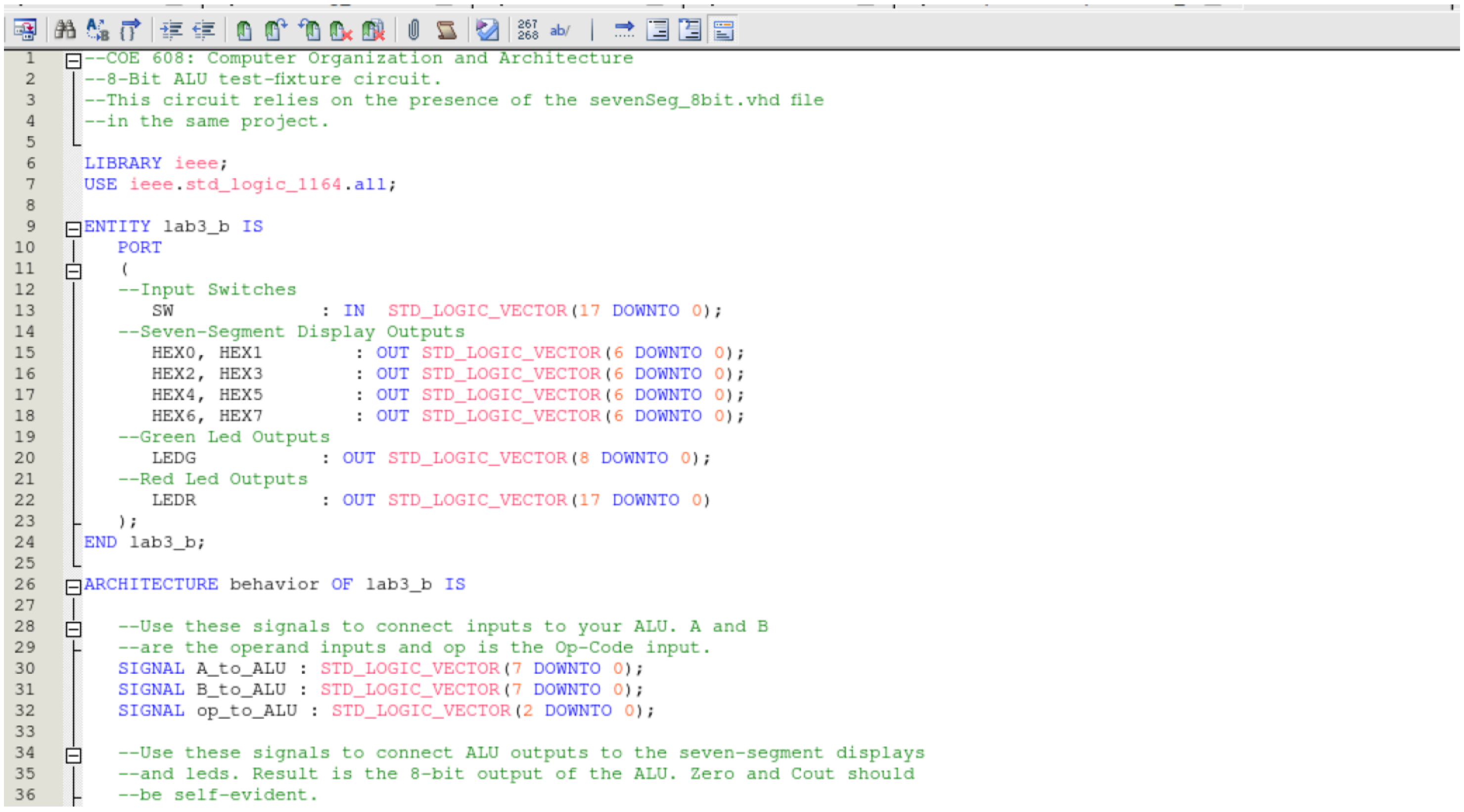


```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity alu is
   port(
         a   :  in std_logic_vector(31 downto 0);
         b   :  in std_logic_vector(31 downto 0);
         op  :  in std_logic_vector(2 downto 0);
         result : out  std_logic_vector(31 downto 0);
         zero : out std_logic;
         cout : out std_logic
         );
end alu;

architecture Behaviour of alu is
   component adder32
   port(
      Cin    :  in std_logic;
      X,Y    :  in std_logic_vector(31 downto 0);
      S    :  out  std_logic_vector(31 downto 0);
      Cout :  out  std_logic
   );
   end component;

   signal result_s: std_logic_vector(31 downto 0):= (others => '0');
   signal result_add: std_logic_vector(31 downto 0):= (others => '0');
   signal result_sub: std_logic_vector(31 downto 0):= (others => '0');
   signal cout_s  : std_logic := '0';
   signal cout_add  : std_logic := '0';
   signal cout_sub  : std_logic := '0';
   signal zero_s    : std_logic:= '0';

begin
```

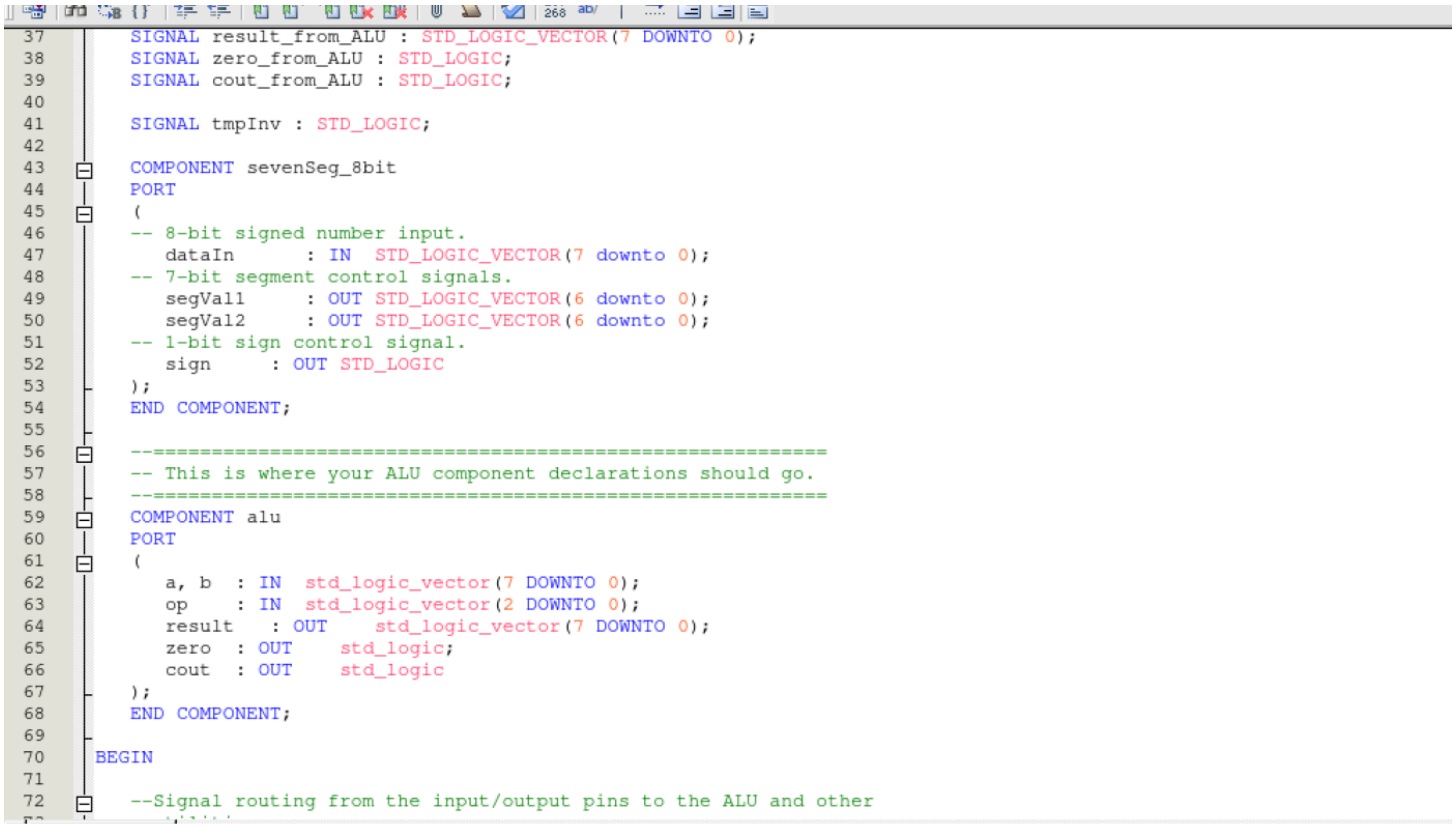


```vhdl
begin
     add0 : adder32 port map (op(2), a, b, result_add,cout_add);
     sub0 : adder32 port map (op(2), a, not b, result_sub,cout_sub);
     process (a, b, op)
     begin
       case (op) is
         when "000" =>
            result_s<= a and b;
            cout_s <= '0';
         when "001" =>
            result_s<= a or b;
            cout_s <= '0';
         when "010" =>
            result_s<= result_add;
            cout_s <= cout_add;
         when "011" =>
            result_s<= b;
            cout_s <= '0';
         when "110" =>
            result_s<= result_sub;
            cout_s <= cout_sub;
         when "100" =>
            result_s<= a(30 downto 0) & '0';
            cout_s <= a(31);
         when "101" =>
            result_s<= '0' & a(31 downto 1);
            cout_s <= '0';
         when others =>
            result_s <= a;
            cout_s <= '0';
         end case;
         case(result_s) is
            when (others => '0') =>
               zero_s <= '1';
            when others =>
               zero_s <= '0';
         end case;
      end process;
      result <= result_s;
      cout <= cout_s;
      zero <= zero_s;
   end Behaviour;
```

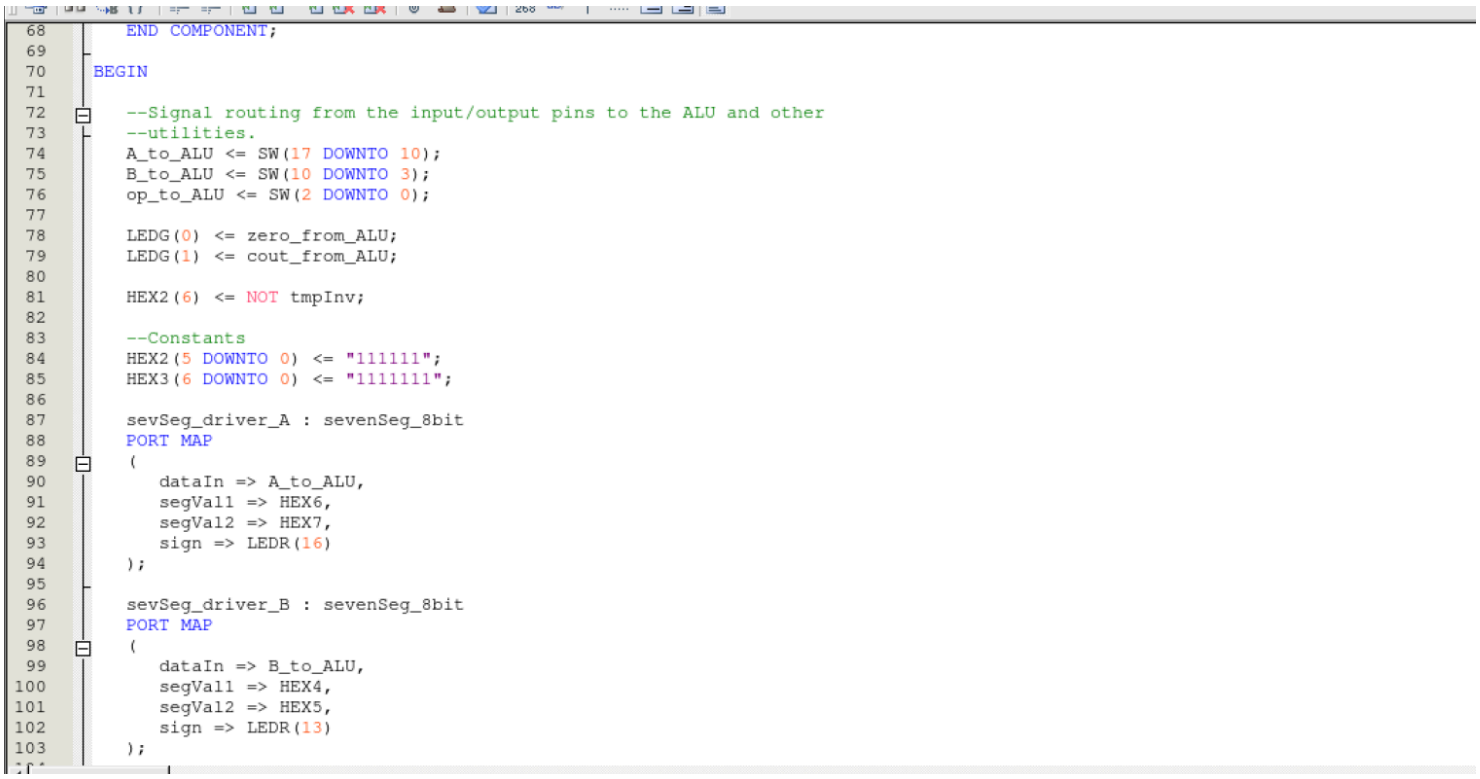
Lab3_b ALU test-fixture circuit: The ALU performs operations based on the input switches' states, and the results are displayed using LEDs and the seven-segment display on the FPGA board.



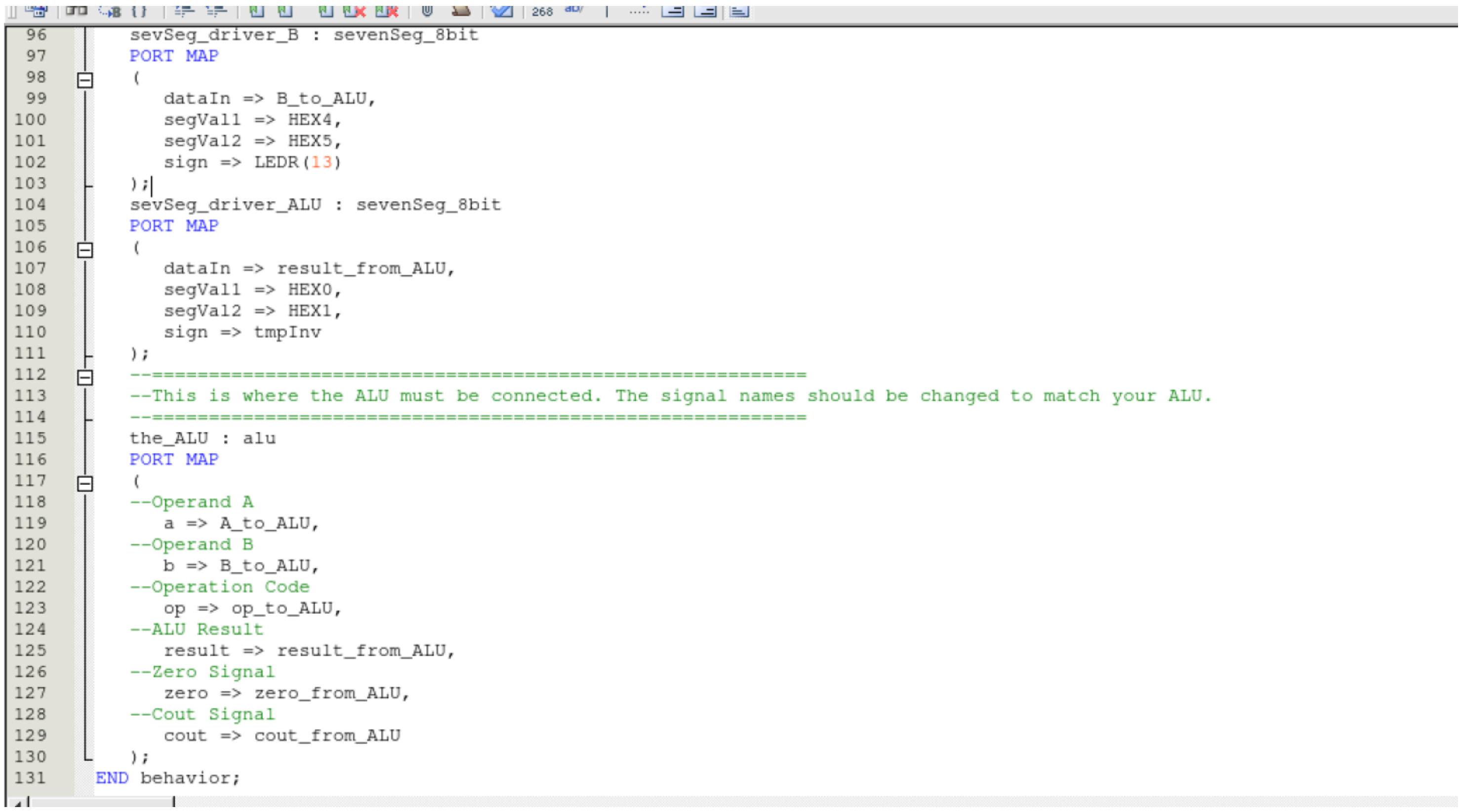
```vhdl
--COE 608: Computer Organization and Architecture
--8-Bit ALU test-fixture circuit.
--This circuit relies on the presence of the sevenSeg_8bit.vhd file
--in the same project.

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY lab3_b IS
   PORT
   (
   --Input Switches
      SW               : IN  STD_LOGIC_VECTOR(17 DOWNTO 0);
   --Seven-Segment Display Outputs
      HEX0, HEX1          : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX2, HEX3          : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX4, HEX5          : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX6, HEX7          : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
   --Green Led Outputs
      LEDG             : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
   --Red Led Outputs
      LEDR             : OUT STD_LOGIC_VECTOR(17 DOWNTO 0)
   );
END lab3_b;

ARCHITECTURE behavior OF lab3_b IS

   --Use these signals to connect inputs to your ALU. A and B
   --are the operand inputs and op is the Op-Code input.
   SIGNAL A_to_ALU : STD_LOGIC_VECTOR(7 DOWNTO 0);
   SIGNAL B_to_ALU : STD_LOGIC_VECTOR(7 DOWNTO 0);
   SIGNAL op_to_ALU : STD_LOGIC_VECTOR(2 DOWNTO 0);

   --Use these signals to connect ALU outputs to the seven-segment displays
   --and leds. Result is the 8-bit output of the ALU. Zero and Cout should
   --be self-evident.
```



```vhdl
   SIGNAL result_from_ALU : STD_LOGIC_VECTOR(7 DOWNTO 0);
   SIGNAL zero_from_ALU : STD_LOGIC;
   SIGNAL cout_from_ALU : STD_LOGIC;

   SIGNAL tmpInv : STD_LOGIC;

   COMPONENT sevenSeg_8bit
   PORT
   (
   -- 8-bit signed number input.
      dataIn       : IN  STD_LOGIC_VECTOR(7 downto 0);
   -- 7-bit segment control signals.
      segVal1      : OUT STD_LOGIC_VECTOR(6 downto 0);
      segVal2      : OUT STD_LOGIC_VECTOR(6 downto 0);
   -- 1-bit sign control signal.
      sign      : OUT STD_LOGIC
   );
   END COMPONENT;

   --==========================================================
   -- This is where your ALU component declarations should go.
   --==========================================================
   COMPONENT alu
   PORT
   (
      a, b   : IN  std_logic_vector(7 DOWNTO 0);
      op     : IN  std_logic_vector(2 DOWNTO 0);
      result    : OUT     std_logic_vector(7 DOWNTO 0);
      zero   : OUT     std_logic;
      cout   : OUT     std_logic
   );
   END COMPONENT;

BEGIN

   --Signal routing from the input/output pins to the ALU and other
```

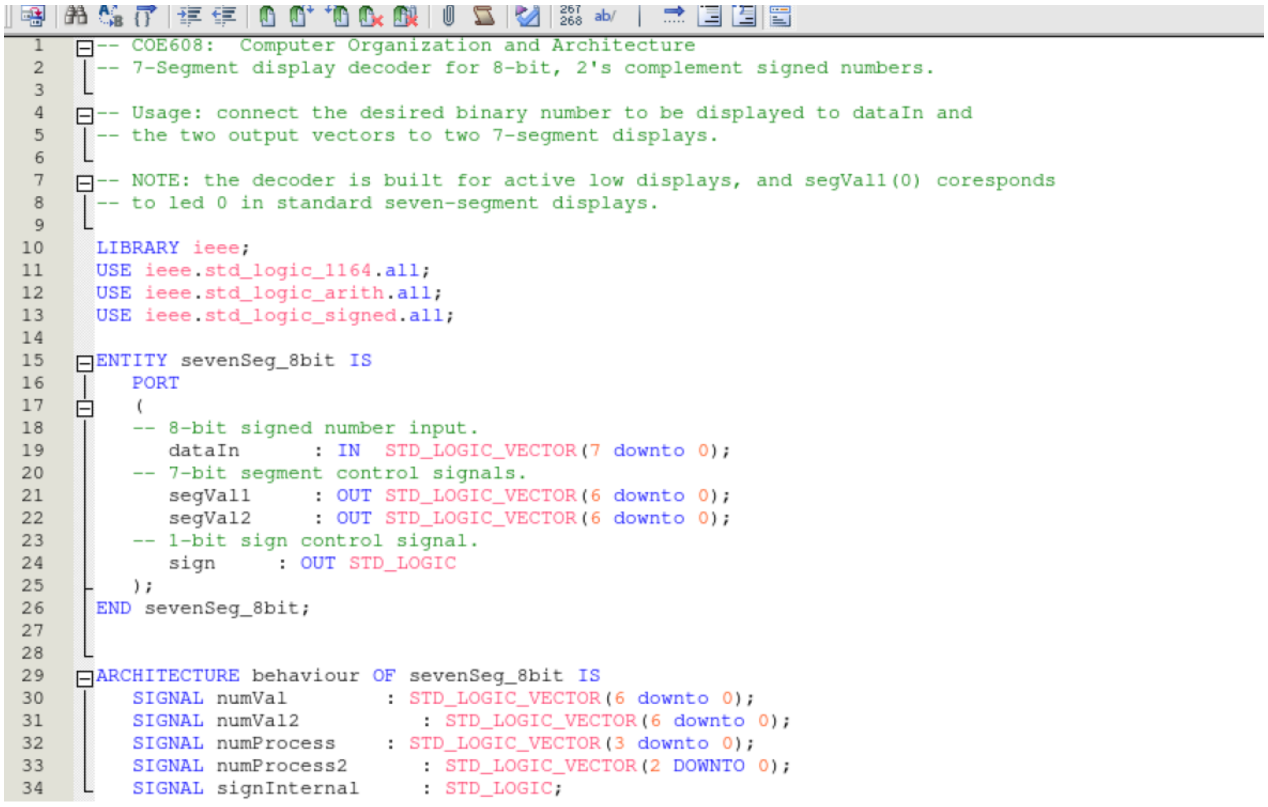


```vhdl
   END COMPONENT;

BEGIN

   --Signal routing from the input/output pins to the ALU and other
   --utilities.
   A_to_ALU <= SW(17 DOWNTO 10);
   B_to_ALU <= SW(10 DOWNTO 3);
   op_to_ALU <= SW(2 DOWNTO 0);

   LEDG(0) <= zero_from_ALU;
   LEDG(1) <= cout_from_ALU;

   HEX2(6) <= NOT tmpInv;

   --Constants
   HEX2(5 DOWNTO 0) <= "111111";
   HEX3(6 DOWNTO 0) <= "1111111";

   sevSeg_driver_A : sevenSeg_8bit
   PORT MAP
   (
      dataIn => A_to_ALU,
      segVal1 => HEX6,
      segVal2 => HEX7,
      sign => LEDR(16)
   );

   sevSeg_driver_B : sevenSeg_8bit
   PORT MAP
   (
      dataIn => B_to_ALU,
      segVal1 => HEX4,
      segVal2 => HEX5,
      sign => LEDR(13)
   );
```

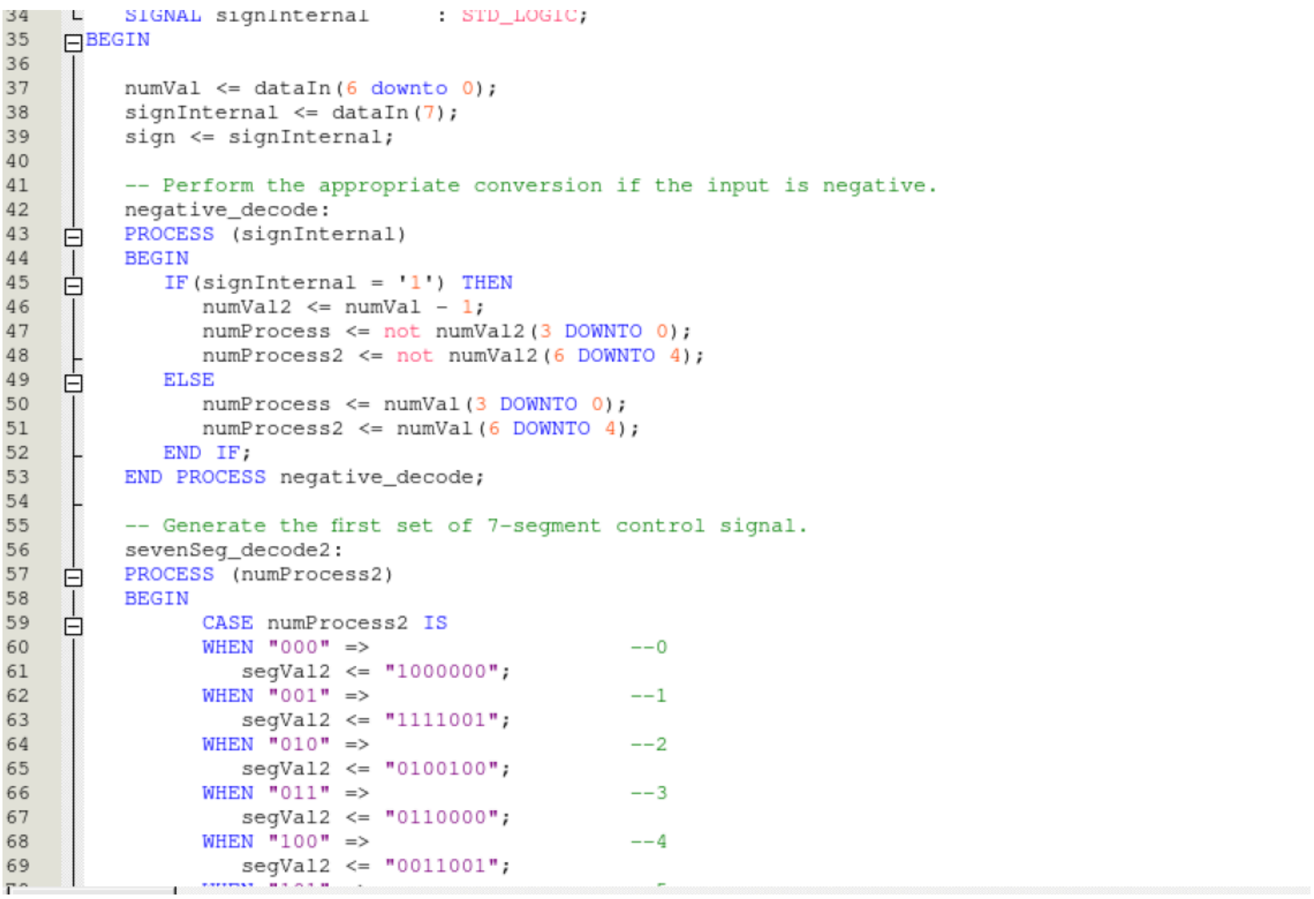


```
    sevSeg_driver_B : sevenSeg_8bit
    PORT MAP
    (
       dataIn => B_to_ALU,
       segVal1 => HEX4,
       segVal2 => HEX5,
       sign => LEDR(13)
    );
    sevSeg_driver_ALU : sevenSeg_8bit
    PORT MAP
    (
       dataIn => result_from_ALU,
       segVal1 => HEX0,
       segVal2 => HEX1,
       sign => tmpInv
    );
    --==========================================================
    --This is where the ALU must be connected. The signal names should be changed to match your ALU.
    --==========================================================
    the_ALU : alu
    PORT MAP
    (
    --Operand A
       a => A_to_ALU,
    --Operand B
       b => B_to_ALU,
    --Operation Code
       op => op_to_ALU,
    --ALU Result
       result => result_from_ALU,
    --Zero Signal
       zero => zero_from_ALU,
    --Cout Signal
       cout => cout_from_ALU
    );
END behavior;
```

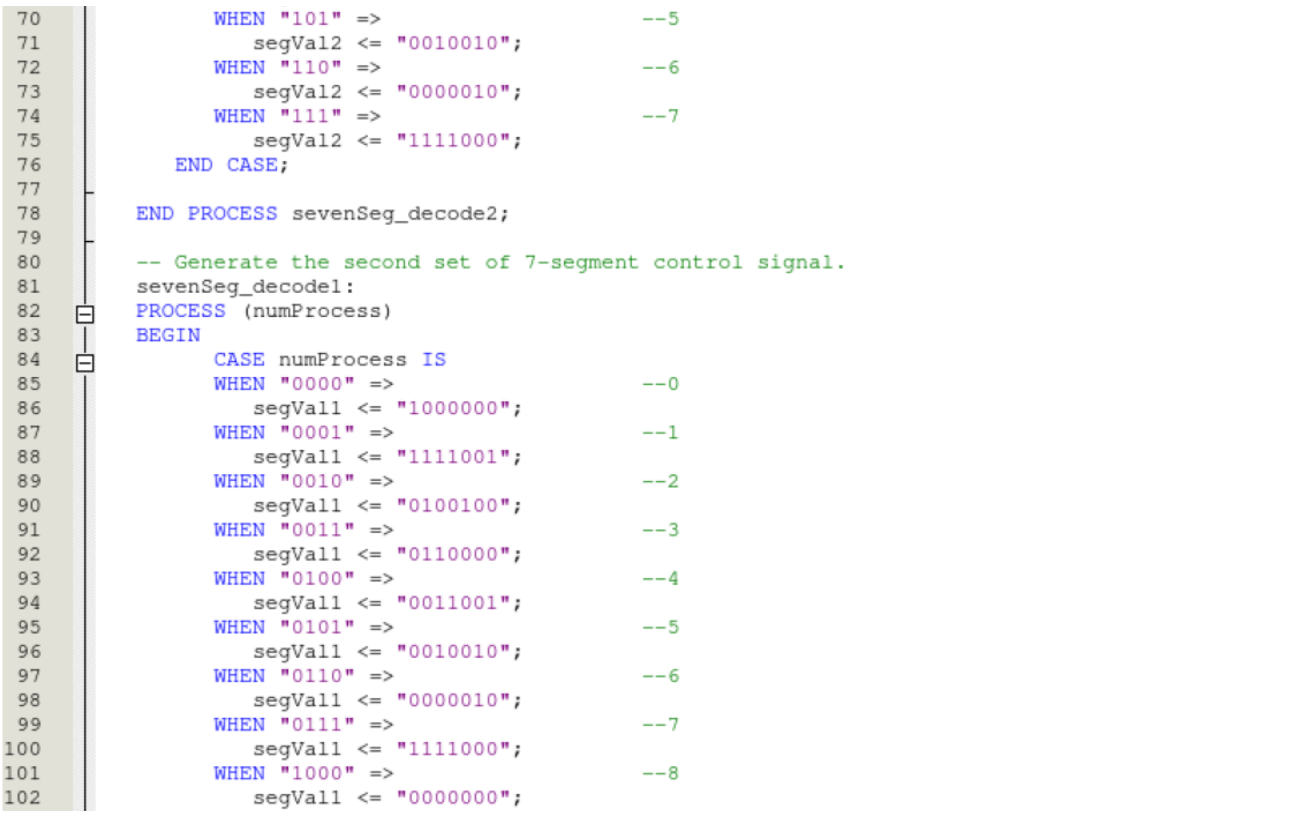
SSEG 8-bit: The SSEG VHDL code is the 7-segment display decoder for an 8-bit function. This is used to connect the binary number and display it to the dataIn port and the two output vectors to the 7-segment display. This program was provided in the lab to allow testing on the FGPA board.



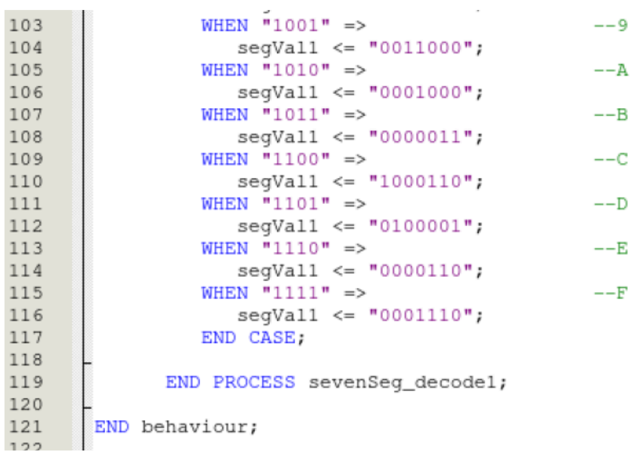
```
-- COE608:  Computer Organization and Architecture
-- 7-Segment display decoder for 8-bit, 2's complement signed numbers.

-- Usage: connect the desired binary number to be displayed to dataIn and
-- the two output vectors to two 7-segment displays.

-- NOTE: the decoder is built for active low displays, and segVal1(0) coresponds
-- to led 0 in standard seven-segment displays.

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;

ENTITY sevenSeg_8bit IS
    PORT
    (
    -- 8-bit signed number input.
       dataIn      : IN  STD_LOGIC_VECTOR(7 downto 0);
    -- 7-bit segment control signals.
       segVal1     : OUT STD_LOGIC_VECTOR(6 downto 0);
       segVal2     : OUT STD_LOGIC_VECTOR(6 downto 0);
    -- 1-bit sign control signal.
       sign      : OUT STD_LOGIC
    );
END sevenSeg_8bit;


ARCHITECTURE behaviour OF sevenSeg_8bit IS
    SIGNAL numVal        : STD_LOGIC_VECTOR(6 downto 0);
    SIGNAL numVal2          : STD_LOGIC_VECTOR(6 downto 0);
    SIGNAL numProcess    : STD_LOGIC_VECTOR(3 downto 0);
    SIGNAL numProcess2      : STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL signInternal     : STD_LOGIC;
```



```
   SIGNAL signInternal      : STD_LOGIC;
BEGIN

   numVal <= dataIn(6 downto 0);
   signInternal <= dataIn(7);
   sign <= signInternal;

   -- Perform the appropriate conversion if the input is negative.
   negative_decode:
   PROCESS (signInternal)
   BEGIN
      IF(signInternal = '1') THEN
         numVal2 <= numVal - 1;
         numProcess <= not numVal2(3 DOWNTO 0);
         numProcess2 <= not numVal2(6 DOWNTO 4);
      ELSE
         numProcess <= numVal(3 DOWNTO 0);
         numProcess2 <= numVal(6 DOWNTO 4);
      END IF;
   END PROCESS negative_decode;

   -- Generate the first set of 7-segment control signal.
   sevenSeg_decode2:
   PROCESS (numProcess2)
   BEGIN
         CASE numProcess2 IS
         WHEN "000" =>                    --0
            segVal2 <= "1000000";
         WHEN "001" =>                    --1
            segVal2 <= "1111001";
         WHEN "010" =>                    --2
            segVal2 <= "0100100";
         WHEN "011" =>                    --3
            segVal2 <= "0110000";
         WHEN "100" =>                    --4
            segVal2 <= "0011001";
```



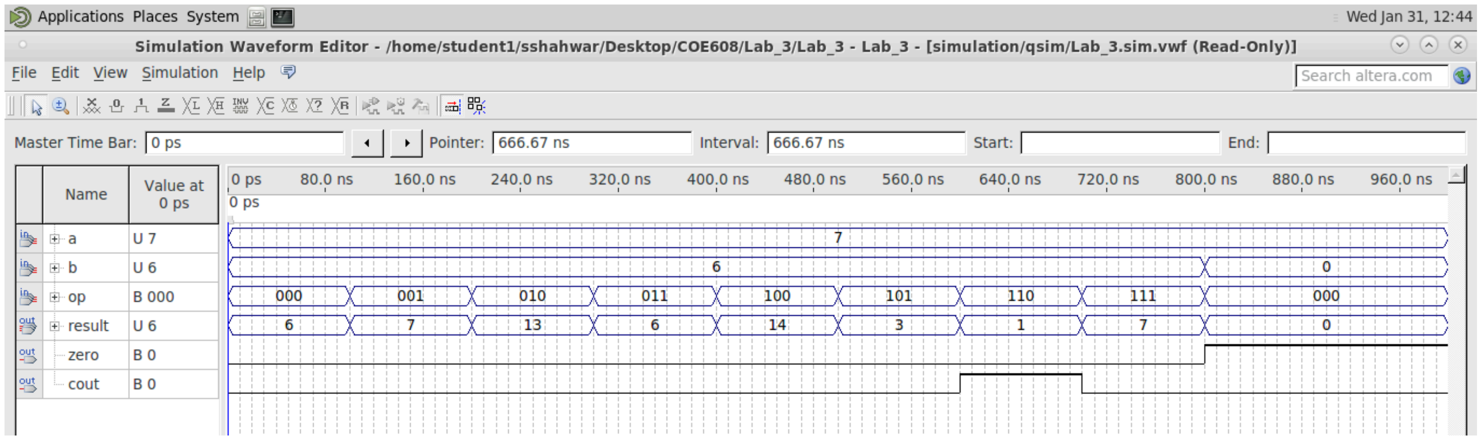
```
         WHEN "101" =>                    --5
            segVal2 <= "0010010";
         WHEN "110" =>                    --6
            segVal2 <= "0000010";
         WHEN "111" =>                    --7
            segVal2 <= "1111000";
      END CASE;

   END PROCESS sevenSeg_decode2;

   -- Generate the second set of 7-segment control signal.
   sevenSeg_decode1:
   PROCESS (numProcess)
   BEGIN
         CASE numProcess IS
         WHEN "0000" =>                   --0
            segVal1 <= "1000000";
         WHEN "0001" =>                   --1
            segVal1 <= "1111001";
         WHEN "0010" =>                   --2
            segVal1 <= "0100100";
         WHEN "0011" =>                   --3
            segVal1 <= "0110000";
         WHEN "0100" =>                   --4
            segVal1 <= "0011001";
         WHEN "0101" =>                   --5
            segVal1 <= "0010010";
         WHEN "0110" =>                   --6
            segVal1 <= "0000010";
         WHEN "0111" =>                   --7
            segVal1 <= "1111000";
         WHEN "1000" =>                   --8
            segVal1 <= "0000000";
```



```
         WHEN "1001" =>                  --9
            segVal1 <= "0011000";
         WHEN "1010" =>                  --A
            segVal1 <= "0001000";
         WHEN "1011" =>                  --B
            segVal1 <= "0000011";
         WHEN "1100" =>                  --C
            segVal1 <= "1000110";
         WHEN "1101" =>                  --D
            segVal1 <= "0100001";
         WHEN "1110" =>                  --E
            segVal1 <= "0000110";
         WHEN "1111" =>                  --F
            segVal1 <= "0001110";
         END CASE;

      END PROCESS sevenSeg_decode1;

END behaviour;
```

Results:

ALU: The image displayed below is the waveform which was generated from the created ALU system. The ALU system has three inputs, which are denoted by 'a, 'b,' and 'operation (op). The two outputs are denoted as 'result,' 'zero,' and 'cout'. The way the waveform functions depends on the input values, the operation being performed, and both the 'zero' and 'cout' values which are used as a function to set all values to 0 and a carry bit.



Simulation Waveform Editor - /home/student1/sshahwar/Desktop/COE608/Lab_3/Lab_3 - Lab_3 - [simulation/qsim/Lab_3.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 666.67 ns Interval: 666.67 ns Start: End:

Name	Value at 0 ps	0 ps	80.0 ns	160.0 ns	240.0 ns	320.0 ns	400.0 ns	480.0 ns	560.0 ns	640.0 ns	720.0 ns	800.0 ns	880.0 ns	960.0 ns
a	U 7	7												
b	U 6	6								0				
op	B 000	000	001	010	011	100	101	110	111	000				
result	U 6	6	7	13	6	14	3	1	7	0				
zero	B 0													
cout	B 0													

Discussion:

The ALU system is to manage the arithmetic that takes place within a system. The values of a and b are set as unsigned and placed as a values of 6 and 7. All the values that are in the operation section consist of different operations that will occur with the input values. Such operations include AND (logic), OR (logic), ADD, SUB, ROL, and ROR. In terms of the waveform, the operation begins by taking the value of 6, which is associated with “000,” and for reference, the number 7 is associated with “111”. The first operation is the OR (logical operation), which displays 7. The second operation is the ADD operation, where 7 and 6 are added together to get 13. The next few operations are followed by SUB, ROL, and ROR. The ‘cout’ value is the carry value for a digit that is below 0. At the end of the waveform, the ‘zeros’ output is high, meaning that all the corresponding values will be set to 0. It is like a reset function in the waveform.