Poorman’s ALU: Milestone 5

Project Iteration 2 Release

04.30.2023

Ryan Woodward

Grand Canyon University
SWE450 - Embedded Systems 2
Prof. Mark Reha

Table of Contents

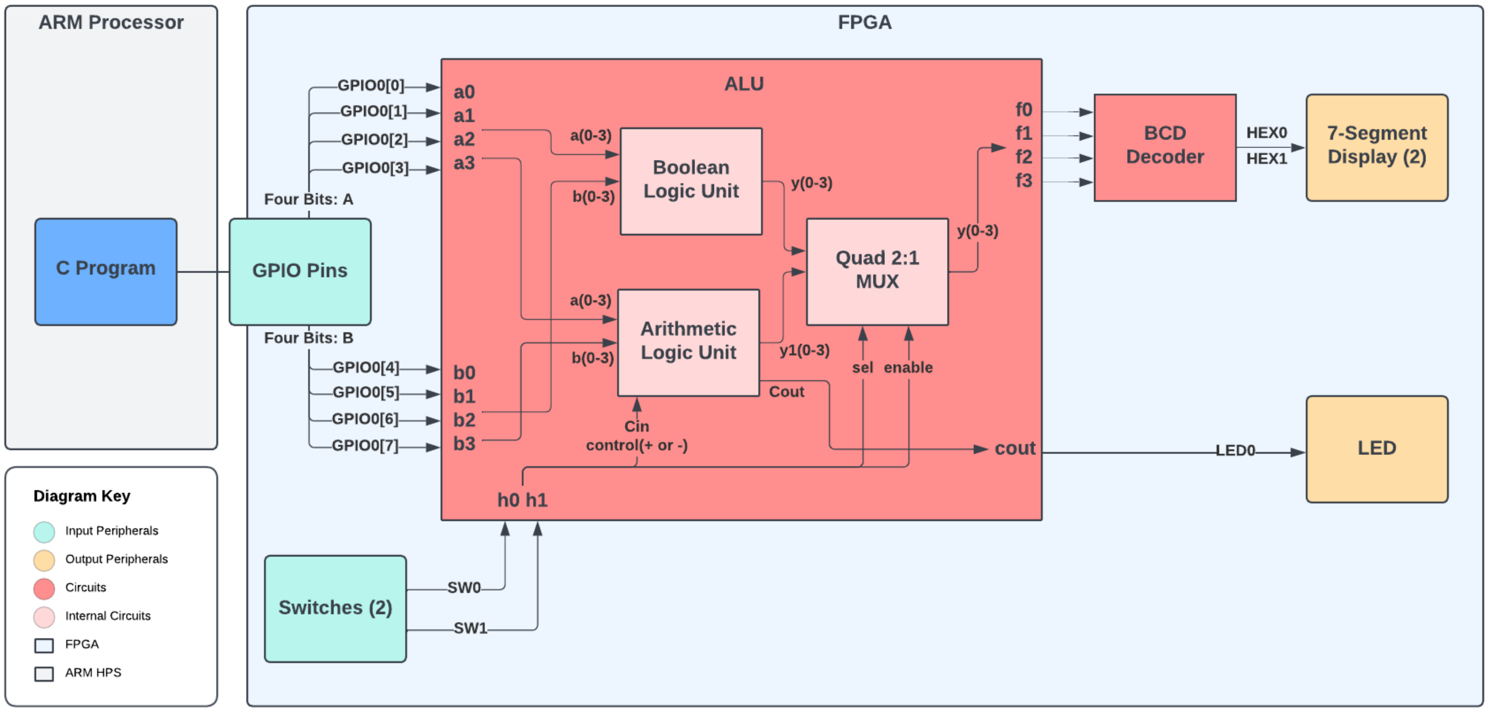
Topic	Milestone 4: Project Iteration Release 3
Date	April 30, 2023
Revision	**4.0**
Milestone Summary	
Github URL	[Ryanjwoodward/Poormans_ALU: Dedicated Repository for my SWE450 (Embedded Systems 2) Poormans ALU Project (github.com)]()
Screencast URL	https://youtu.be/qGbF3a6-ZJk

User Story/Task	**Hours Worked**	**Hours Remaining**
Code Refinement	**2+**	**0**
Arm Assembly Code	**4+**	**1+**
IP Core Addition	**2+**	**0**

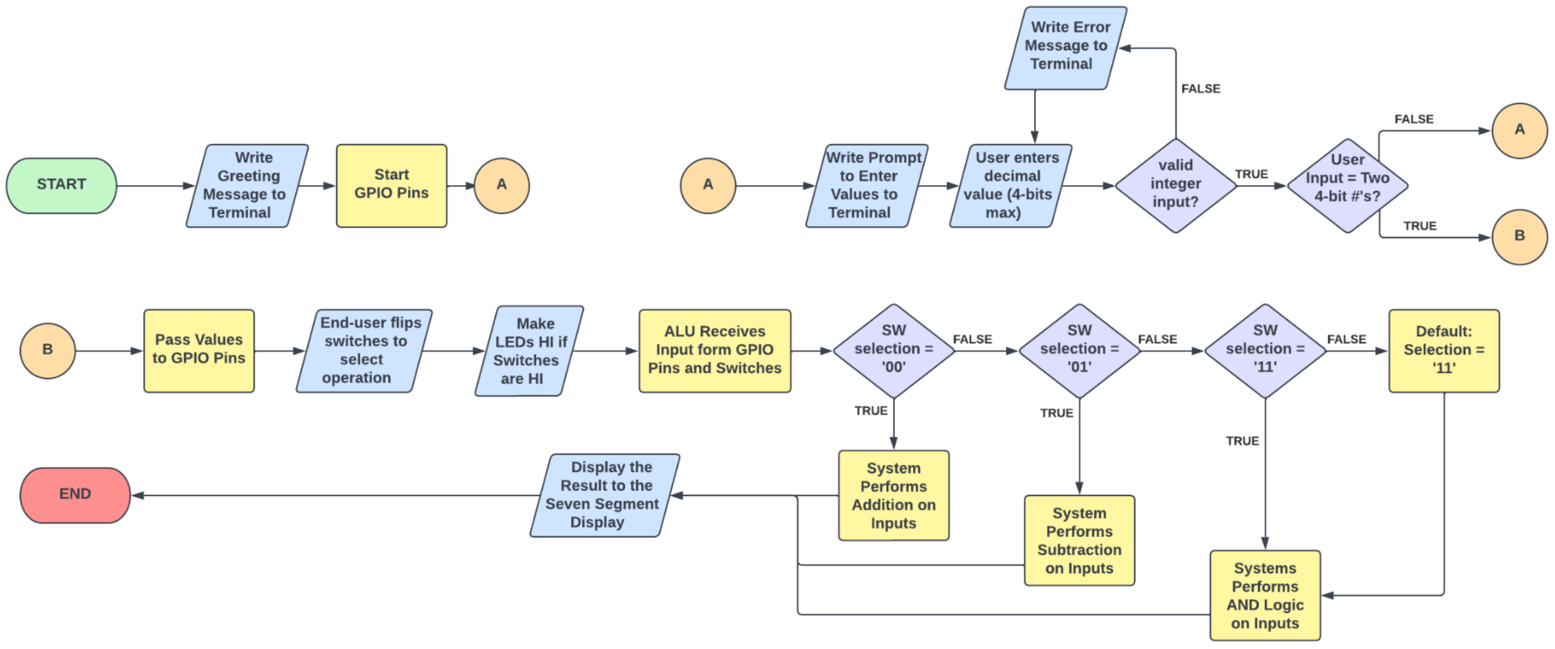
General Technical Approach

As this is the final release of the project there was initially little more than tasks of refining the project such as adding further comments to the code and setting up another repository that was clean and navigable. However, after activity 7 I wanted to add more to the project namely an IP block and some Arm assembly to the C-file. Those tasks are complete and integrated with the project. This includes testing the project as a whole with these new portions integrated. Furthermore, as a part of the final release of this project I completed the project using Verilog as well and tested that out on the board to ensure that the project worked in both RTLs

System Design



Application Design


START
Write Greeting Message to Terminal
Start GPIO Pins
A
A
Write Prompt to Enter Values to Terminal
User enters decimal value (4-bits max)
valid integer input?
FALSE
Write Error Message to Terminal
TRUE
User Input = Two 4-bit #'s?
FALSE
A
TRUE
B
B
Pass Values to GPIO Pins
End-user flips switches to select operation
Make LEDs HI if Switches are HI
ALU Receives Input form GPIO Pins and Switches
SW selection = '00'
FALSE
SW selection = '01'
FALSE
SW selection = '11'
FALSE
Default: Selection = '11'
TRUE
System Performs Addition on Inputs
TRUE
System Performs Subtraction on Inputs
TRUE
Systems Performs AND Logic on Inputs
Display the Result to the Seven Segment Display
END

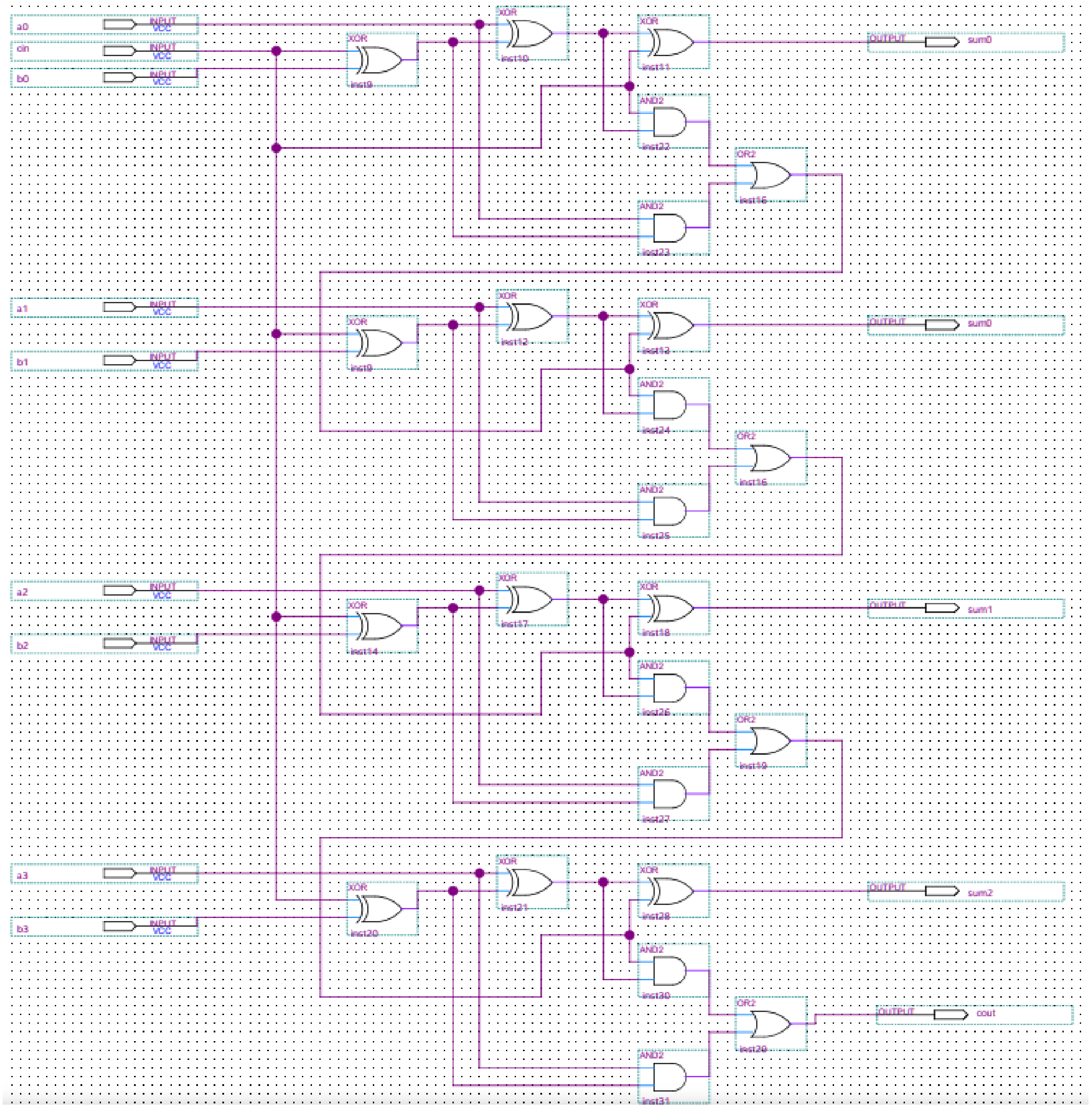
Note: Karnaugh Maps are most suitable for 2-5 variables each of the following circuits is comprised of smaller variants of the circuit (i.e Quad 2:1 MUX to a 2:1 MUX and 4-bit adder/subtractor to 1-bit adder/subtractor) as such I create K-maps and boolean expressions for the smaller circuits making up each of the larger ones. This is because the principle expression and function of each larger circuit can still be demonstrated.

4-bit Adder/Subtractor (Arithmetic Unit)

Arithmetic Unit Truth Table

a3	a2	a1	a0	b3	b2	b1	b0	cin	cout	s3	s2	s1	s0
Cin = ‘0’ (Addition)													
1	0	0	0	1	0	0	0	0	1	0	0	0	0
0	1	0	0	0	1	0	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0	0	1	0	1	0	0
0	1	1	0	0	1	1	0	0	0	1	1	0	0
1	1	1	0	1	1	1	0	0	1	1	1	0	0
0	1	0	1	0	1	0	1	0	0	1	0	1	0
1	1	0	1	1	1	0	1	0	1	1	0	1	0
0	1	1	1	0	1	1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1	0	1	1	1	1	0
Cin = ‘1’ (Subtraction)													
1	0	0	0	0	1	1	0	1	1	0	0	1	0
0	1	0	0	0	0	1	0	1	1	0	0	1	0
1	1	0	0	1	0	0	0	1	1	0	1	0	0
1	0	0	1	0	1	1	1	1	1	0	0	1	0
1	1	1	0	1	0	1	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1	1	0	1	1	1	1
1	0	0	0	0	1	1	1	1	1	0	0	0	1
1	1	0	1	1	0	0	1	1	1	0	1	0	0
1	1	1	1	1	1	1	1	1	1	0	0	0	0

4-bit Adder/Subtractor (Arithmetic Unit) Schematic Capture



4-bit Adder/Subtractor Karnaugh Maps

Sum/Difference

A \ B, Cin	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Cout

A \ B, Cin	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Bout

A \ B, Cin	00	01	11	10
0	0	1	1	1
1	0	0	1	0

Explanation:

The K-map on the left serves to represent the output variables Sum or Difference in a 1-bit adder or subtractor. The middle map represents the Carry out value from a 1-bit adder and the right map represents the Borrow out from a 1-bit subtractor. The difference is the Cin bit is used as a selector to either operation. Extending these maps to a 4-bit wide A and B is simple. Similar to how if A=1 and B=0 the output of Sum is 1 we might see in 4-bits A=1000, B=0000 the result is 1000. Were we to do this for Cout the result would be 0 in both cases because there is no carry. However, for the Bout Table the result is 1 when A=1 and B=0 due to the subtraction operation.

Boolean Expressions

Sum/Difference K-Map

S/D = A’B’Cin + AB’Cin’ + ABCin + A’BCin’

Carry Out K-Map

Cout = AB’Cin + ABCin + ABCin’ + A’BCin

Borrow Out K-Map

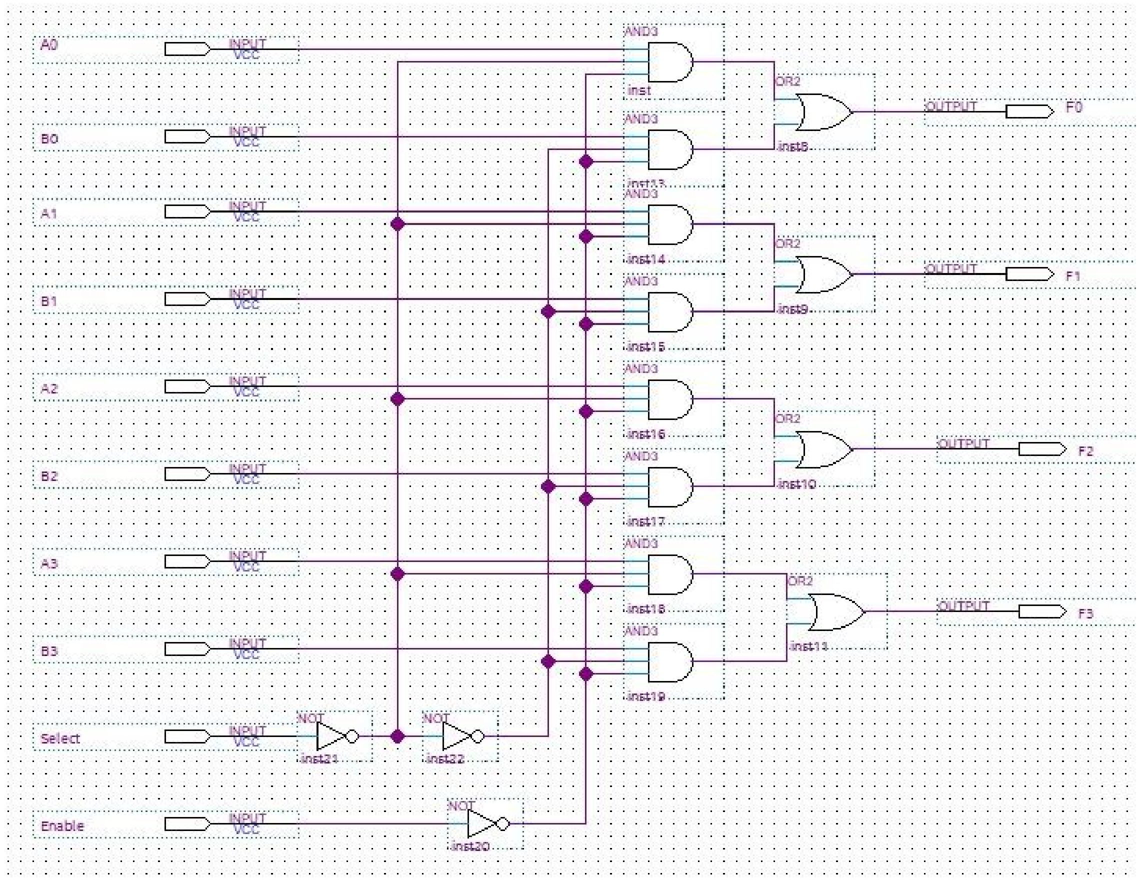
Bout = A’B’Cin + A’BCin + ABCin + ABCin’

Quad 2:1 Multiplexer

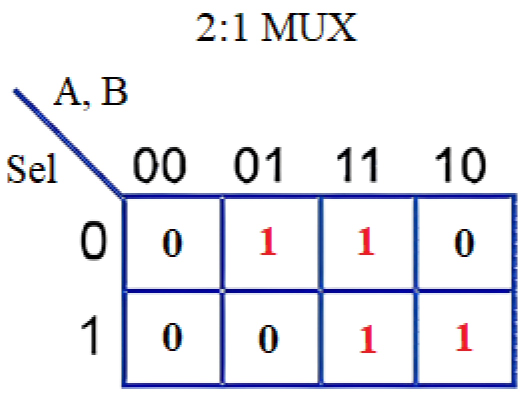
Quad 2:1 MUX Truth Table

a3	a2	a1	a0	b3	b2	b1	b0	S	E	f3	f2	f1	f0
1	1	0	1	0	1	0	1	0	0	1	1	0	1
1	1	0	1	0	1	0	1	1	0	0	1	0	1
0	1	1	1	1	0	1	1	0	0	0	1	1	1
0	1	1	1	1	0	1	1	1	0	1	0	1	1
1	1	0	1	0	1	0	1	0	1	0	0	0	0
1	1	0	1	0	1	0	1	1	1	0	0	0	0
0	1	1	1	1	0	1	1	0	1	0	0	0	0
0	1	1	1	1	0	1	1	1	1	0	0	0	0

Quad 2:1 MUX Schematic Capture



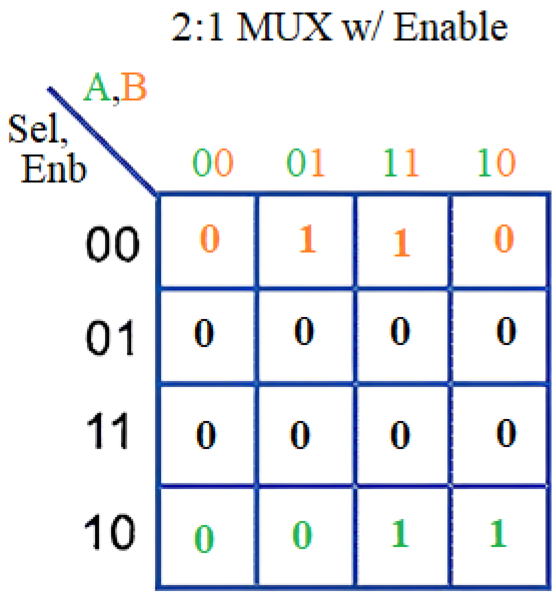
Karnaugh Maps


2:1 MUX

Sel \ A, B	00	01	11	10
0	0	1	1	0
1	0	0	1	1

<u>Explanation</u>

For a 2:1 MUX the output is determined entirely by the Select input, essentially acting as an If-then logic logic block. If Select is '1' then the output will be A and if Select is Low the output will be B. In my Quad 2:1 MUX this is reversed (Sel =1 Output = B) and there is an additional input of the enable switch when high makes the output 0 regardless of the other inputs. If we include an 'enable' input the K-map looks like:


2:1 MUX w/ Enable

Sel, Enb \ A,B	00	01	11	10
00	0	1	1	0
01	0	0	0	0
11	0	0	0	0
10	0	0	1	1

Boolean Expression

<u>2:1 MUX</u>

F = Sel'A'B + Sel'AB + SelAB + SelAB'

Reduce this to: F = Sel'B + SelA

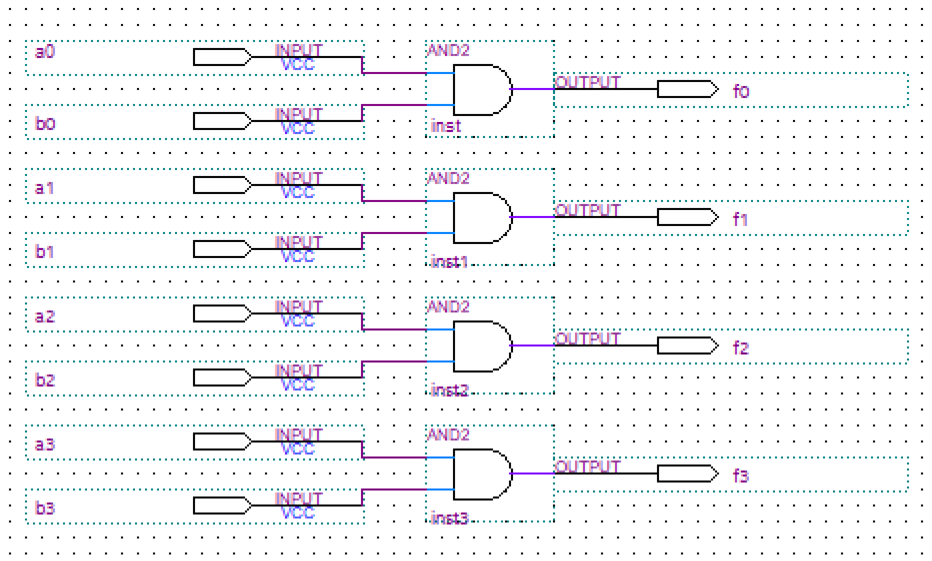
<u>2:1 MUX with Enable</u>

F = Sel'B.Enb' + SelA.Enb'

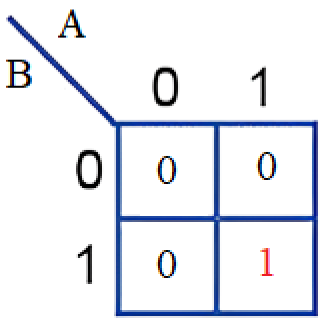
Boolean Logic Unit Truth Table

a3	a2	a1	a0	b3	b2	b1	b0	f3	f2	f1	f0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	0	1	0	0
1	0	0	0	1	0	0	0	1	0	0	0
1	1	0	0	1	1	0	0	1	1	0	0
1	0	1	0	0	1	0	1	0	0	0	0
1	0	1	0	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0

Boolean Logic Unit Schematic Capture



Karnaugh Map



Explanation:

Admittedly this one is somewhat unnecessary to do because the Logic unit consists of two 4-bit numbers being AND'ed together. There is a single AND gate for each pair or A and B. The result of the operation will only be true if both A and B are high.

Boolean Expression

F= AB

BCD Decoder

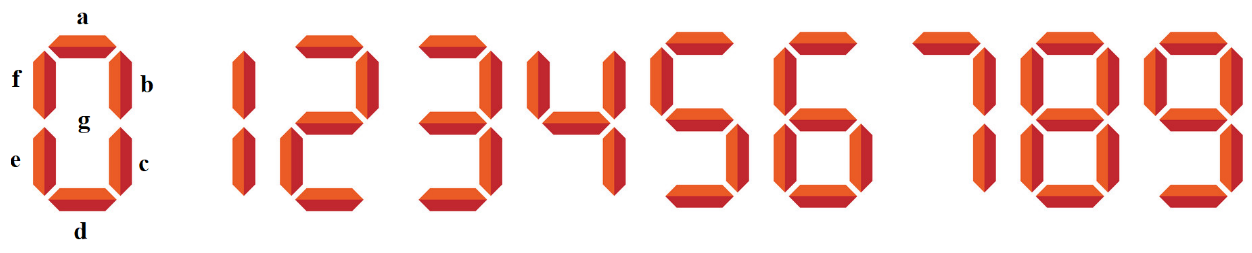
Truth Table

A	B	C	D	a	b	c	d	e	f	g	#
0	0	0	0	1	1	1	1	1	1	0	**0**
0	0	0	1	0	1	1	0	0	0	0	**1**
0	0	1	0	1	1	0	1	1	0	1	**2**
0	0	1	1	1	1	1	1	0	0	1	**3**
0	1	0	0	0	1	1	0	0	1	1	**4**
0	1	0	1	1	0	1	1	0	1	1	**5**
0	1	1	0	1	0	1	1	1	1	1	**6**
0	1	1	1	1	1	1	0	0	0	0	**7**
1	0	0	0	1	1	1	1	1	1	1	**8**
1	0	0	1	1	1	1	1	0	1	1	**9**

Karnaugh Map

<u>Explanation</u>

Each output of the BCD Decoder Requires it's own K-map and Boolean Expression. The K-maps for each letter a-g (output) Represent their value for each number that would be displayed on the Seven segment display. For instance the input 'a' will be low only for the values 1 and 4 (see below).



output: a

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	0

output: b

CD \ AB	00	01	11	10
00	1	1	1	1
01	1	0	1	1
11	1	1	1	1
10	1	0	1	1

output: c

CD \ AB	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	0	1	1	1

output: d

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	0
11	1	1	0	0
10	1	1	0	1

output: e

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

output: f

CD \ AB	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	0	1	0	1

output: g

CD \ AB	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	1	0	1	1
10	1	1	1	1

Boolean Expression:

a = (A' & B' & C' & D') (A' & B' & C & D') | (A' & B & C' & D') | (A & B' & C' & D') | (A & B & C& D)

b = (A' & B' & C' & D') | (A' & B & C & D') | (A & B & C' & D') | (A & B' & C' & D) | (A & B & C & D)

c = (A' & B' & C' & D') | (A' & B & C' & D') | (A' & B & C & D') | (A & B' & C & D) | (A & B & C & D)

d = (A' & B' & C' & D') | (A' & B' & C & D') | (A & B' & C & D') | (A & B & C' & D) | (A & B & C & D)

e = (A' & B' & C' & D') | (A' & B & C' & D') | (A' & B & C & D') | (A & B & C' & D') | (A & B & C & D)

f = (A' & B' & C' & D') | (A' & B' & C' & D) | (A' & B & C & D') | (A & B & C' & D') | (A & B & C & D)

g = (A' & B' & C' & D) | (A' & B & C' & D) | (A & B' & C' & D) | (A & B & C' & D) | (A & B & C & D)

NOTE: Above expression and truth table were built using the provided BCD Seven Segment Decoder (SWE-350) and the reference (GeeksforGeeks, 2019).

Truth Table

Mux Out	Arith Op													
sel1	sel0	a3	a2	a1	a0	b3	b2	b1	b0	f3	f2	f1	f0	cout
sel1 = ‘1’ Output = Result from Boolean Logic Unit														
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	1	0	0	0	1	0
1	0	0	0	1	1	0	0	0	1	0	0	0	0	0
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0
1	0	0	1	0	1	0	1	1	1	0	1	0	1	0
1	0	0	1	1	1	0	1	1	1	0	1	1	1	0
sel1 = ‘0’ Output = Result from Arithmetic Unit, sel0 = ‘0’ (Addition)														
0	1	1	0	0	0	1	0	0	0	0	0	0	0	1
0	1	0	1	0	0	0	1	0	0	1	0	0	0	1
0	1	0	1	1	1	0	1	1	1	1	1	1	0	1
0	1	1	1	0	1	1	1	0	1	1	0	1	0	1
0	1	0	0	0	1	0	0	1	0	0	0	1	1	0
sel1 = ‘0’ Output = Result from Arithmetic Unit, sel0 = ‘1’ (Subtraction)														
0	1	1	0	0	0	0	1	1	0	0	0	1	0	1
0	1	0	1	0	0	0	0	1	0	0	0	1	0	1
0	1	1	1	1	1	1	1	1	1	0	0	0	0	1
0	1	1	1	0	1	1	0	0	1	0	1	0	0	1
0	1	0	0	0	1	0	0	0	1	0	0	0	0	1

Karnaugh Map & Boolean Expression

I omitted this K-map and boolean expression. The basis for each possible result (addition, subtraction, AND), are explained in the K-maps for each individual component circuit's K-map above. As such I did not think it was necessary to write a K-map or boolean expression for this circuit as the desired results are described in each of the above truth tables and k-maps.

Truth Table

Mux Out	Arith Op																							
sel1	sel0	a3	a2	a1	a0	b3	b2	b1	b0	f6	f5	f4	f3	f2	f1	f0	d6	d5	d4	d3	d2	d1	d0	cout
sel1 = ‘1’ Output = Result from Boolean Logic Unit																								
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
1	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1
1	0	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1
1	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0	0	1	0	0	1	0	1
sel1 = ‘0’ Output = Result from Arithmetic Unit, sel0 = ‘0’ (Addition)																								
0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
0	1	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	0
0	1	0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
0	1	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
sel1 = ‘0’ Output = Result from Arithmetic Unit, sel0 = ‘1’ (Subtraction)																								
0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
0	1	1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

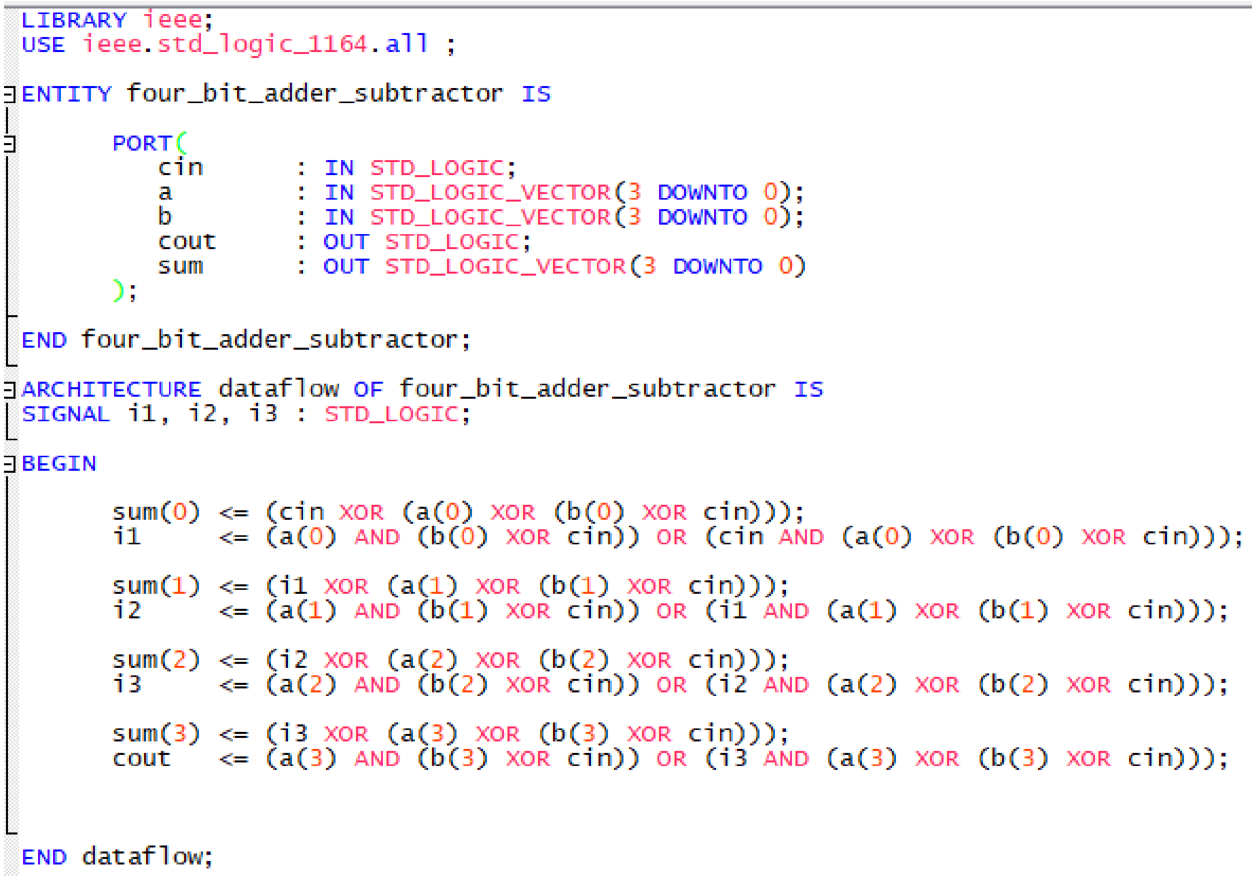
Karnaugh Map & Boolean Expression

I omitted this K-map and boolean expression. The basis for each possible result (addition, subtraction, AND), are explained in the K-maps for each individual component circuit's K-map above. As such I did not think it was necessary to write a K-map or boolean expression for this circuit as the desired results are described in each of the above truth tables and k-maps.

4-bit Adder Subtractor (Arithmetic Unit)

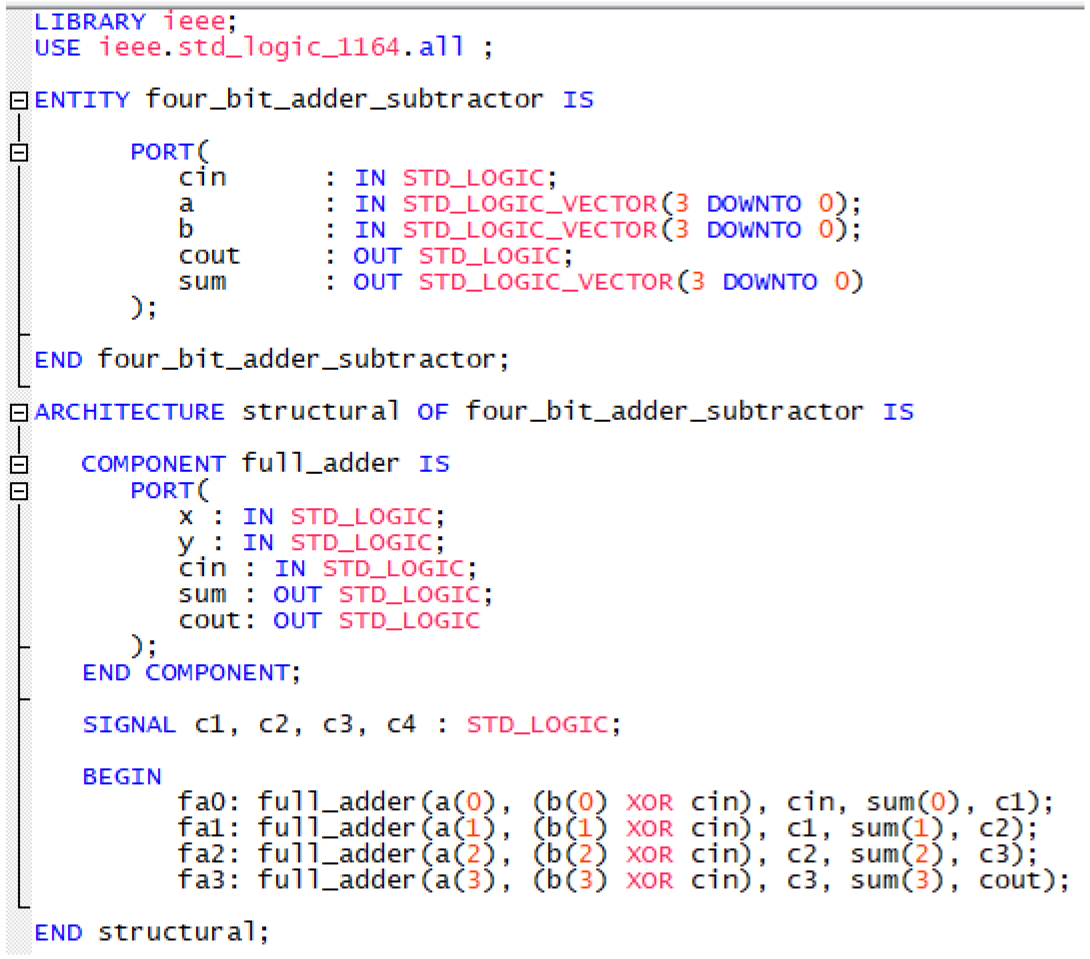
Arithmetic Unit VHDL Code Screenshots

Dataflow



```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;

ENTITY four_bit_adder_subtractor IS

      PORT(
         cin      : IN STD_LOGIC;
         a        : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         b        : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         cout     : OUT STD_LOGIC;
         sum      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
      );

END four_bit_adder_subtractor;

ARCHITECTURE dataflow OF four_bit_adder_subtractor IS
SIGNAL i1, i2, i3 : STD_LOGIC;

BEGIN

      sum(0) <= (cin XOR (a(0) XOR (b(0) XOR cin)));
      i1     <= (a(0) AND (b(0) XOR cin)) OR (cin AND (a(0) XOR (b(0) XOR cin)));

      sum(1) <= (i1 XOR (a(1) XOR (b(1) XOR cin)));
      i2     <= (a(1) AND (b(1) XOR cin)) OR (i1 AND (a(1) XOR (b(1) XOR cin)));

      sum(2) <= (i2 XOR (a(2) XOR (b(2) XOR cin)));
      i3     <= (a(2) AND (b(2) XOR cin)) OR (i2 AND (a(2) XOR (b(2) XOR cin)));

      sum(3) <= (i3 XOR (a(3) XOR (b(3) XOR cin)));
      cout   <= (a(3) AND (b(3) XOR cin)) OR (i3 AND (a(3) XOR (b(3) XOR cin)));


END dataflow;
```

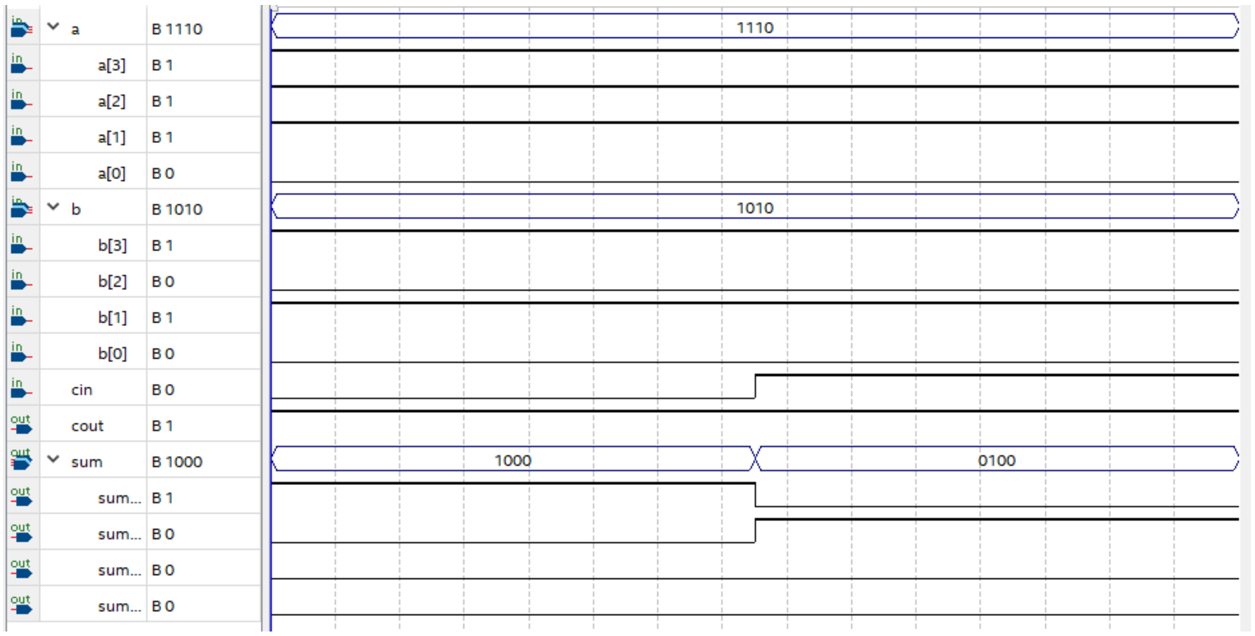
Structural



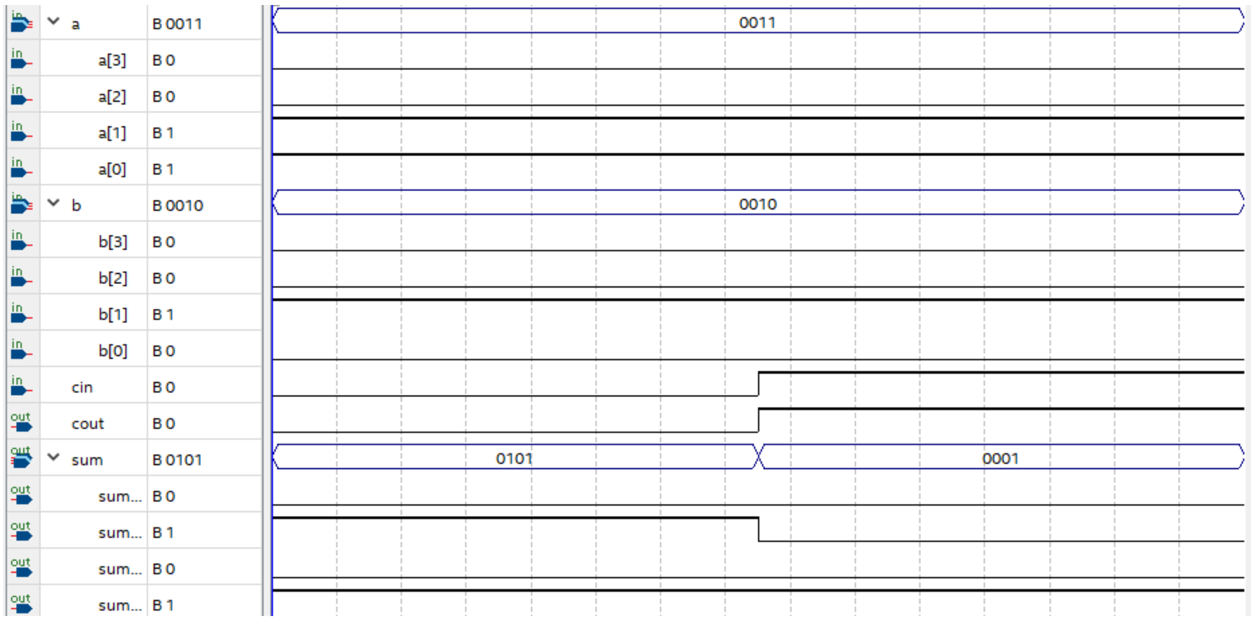
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all ;

ENTITY four_bit_adder_subtractor IS

      PORT(
         cin       : IN STD_LOGIC;
         a         : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         b         : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         cout      : OUT STD_LOGIC;
         sum       : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
      );

END four_bit_adder_subtractor;

ARCHITECTURE structural OF four_bit_adder_subtractor IS

   COMPONENT full_adder IS
      PORT(
         x : IN STD_LOGIC;
         y : IN STD_LOGIC;
         cin : IN STD_LOGIC;
         sum : OUT STD_LOGIC;
         cout: OUT STD_LOGIC
      );
   END COMPONENT;

   SIGNAL c1, c2, c3, c4 : STD_LOGIC;

   BEGIN
         fa0: full_adder(a(0), (b(0) XOR cin), cin, sum(0), c1);
         fa1: full_adder(a(1), (b(1) XOR cin), c1, sum(1), c2);
         fa2: full_adder(a(2), (b(2) XOR cin), c2, sum(2), c3);
         fa3: full_adder(a(3), (b(3) XOR cin), c3, sum(3), cout);

END structural;
```

Arithmetic Unit Simulations

(add: cin = 0 a+b, sub: cin = 1 a-b) a = 1110, b = 1010

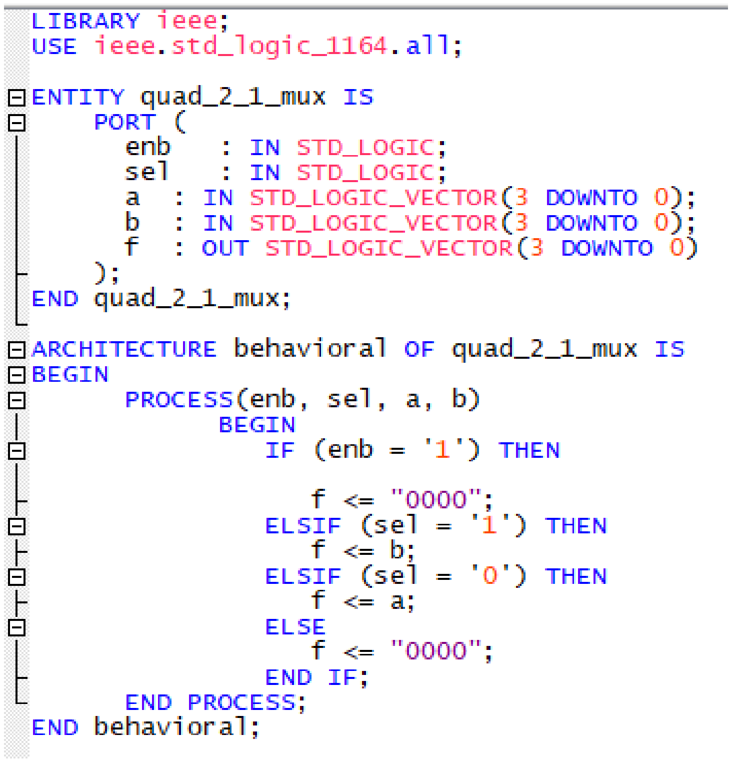


a=0011, b = 0010



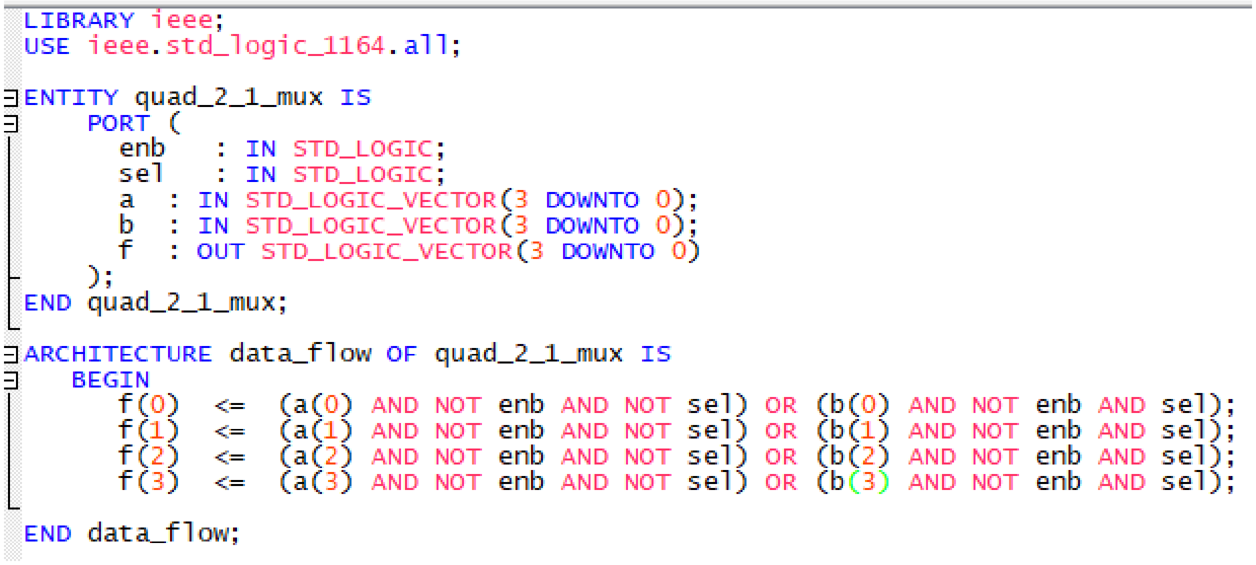
Quad 2:1 MUX VHDL Code Screenshots

Behavioral Model



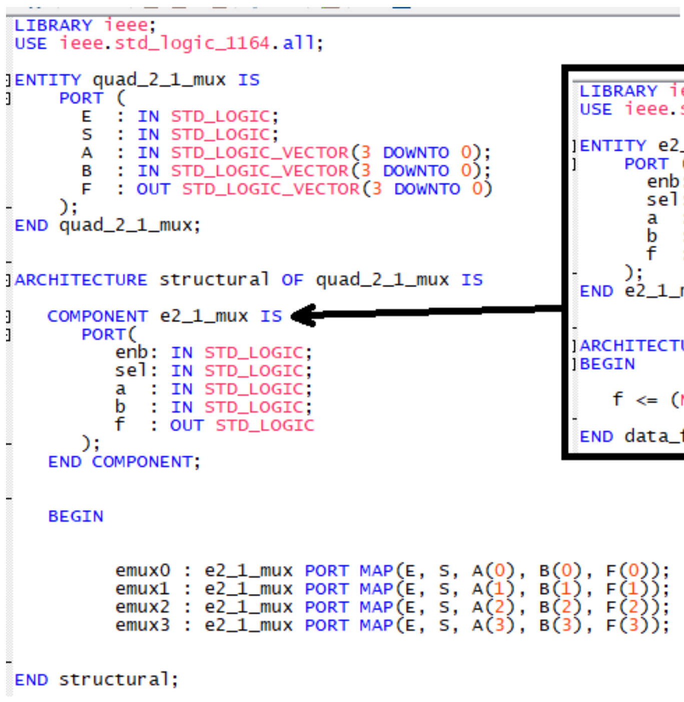
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY quad_2_1_mux IS
    PORT (
        enb    : IN STD_LOGIC;
        sel    : IN STD_LOGIC;
        a  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        b  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        f  : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );
END quad_2_1_mux;

ARCHITECTURE behavioral OF quad_2_1_mux IS
BEGIN
        PROCESS(enb, sel, a, b)
              BEGIN
                 IF (enb = '1') THEN

                    f <= "0000";
                 ELSIF (sel = '1') THEN
                    f <= b;
                 ELSIF (sel = '0') THEN
                    f <= a;
                 ELSE
                    f <= "0000";
                 END IF;
        END PROCESS;
END behavioral;
```

Dataflow Model

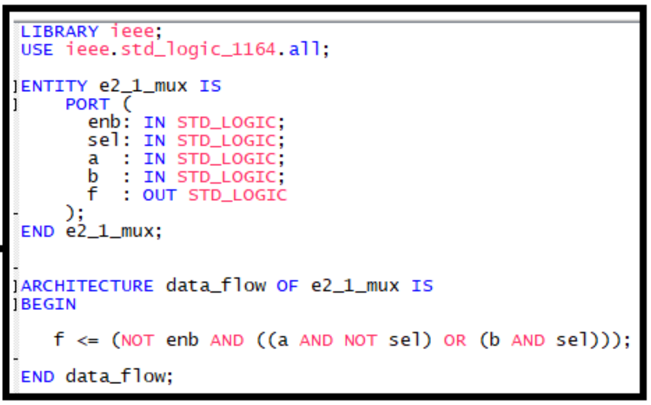


```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY quad_2_1_mux IS
    PORT (
        enb    : IN STD_LOGIC;
        sel    : IN STD_LOGIC;
        a  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        b  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        f  : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );
END quad_2_1_mux;

ARCHITECTURE data_flow OF quad_2_1_mux IS
   BEGIN
      f(0)  <=  (a(0) AND NOT enb AND NOT sel) OR (b(0) AND NOT enb AND sel);
      f(1)  <=  (a(1) AND NOT enb AND NOT sel) OR (b(1) AND NOT enb AND sel);
      f(2)  <=  (a(2) AND NOT enb AND NOT sel) OR (b(2) AND NOT enb AND sel);
      f(3)  <=  (a(3) AND NOT enb AND NOT sel) OR (b(3) AND NOT enb AND sel);

END data_flow;
```

Structural



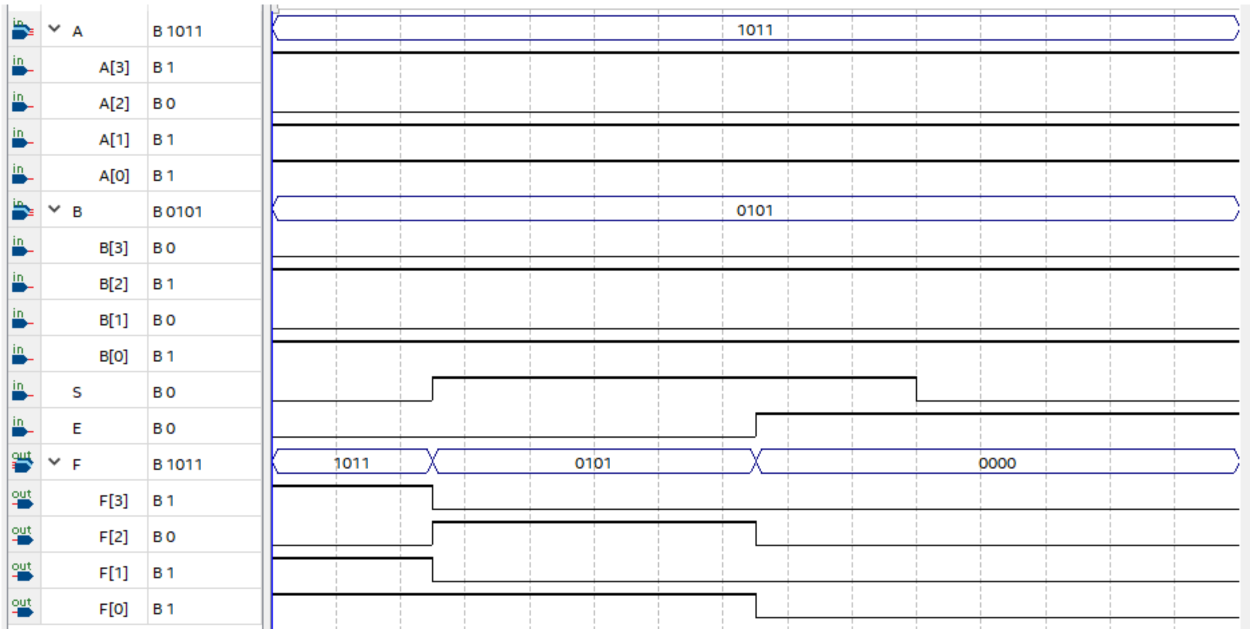
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY quad_2_1_mux IS
    PORT (
        E   : IN STD_LOGIC;
        S   : IN STD_LOGIC;
        A   : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        B   : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        F   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );
END quad_2_1_mux;

ARCHITECTURE structural OF quad_2_1_mux IS

   COMPONENT e2_1_mux IS
      PORT(
         enb: IN STD_LOGIC;
         sel: IN STD_LOGIC;
         a  : IN STD_LOGIC;
         b  : IN STD_LOGIC;
         f  : OUT STD_LOGIC
      );
   END COMPONENT;

   BEGIN

         emux0 : e2_1_mux PORT MAP(E, S, A(0), B(0), F(0));
         emux1 : e2_1_mux PORT MAP(E, S, A(1), B(1), F(1));
         emux2 : e2_1_mux PORT MAP(E, S, A(2), B(2), F(2));
         emux3 : e2_1_mux PORT MAP(E, S, A(3), B(3), F(3));

END structural;
```



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY e2_1_mux IS
    PORT (
        enb: IN STD_LOGIC;
        sel: IN STD_LOGIC;
        a  : IN STD_LOGIC;
        b  : IN STD_LOGIC;
        f  : OUT STD_LOGIC
    );
END e2_1_mux;

ARCHITECTURE data_flow OF e2_1_mux IS
BEGIN

   f <= (NOT enb AND ((a AND NOT sel) OR (b AND sel)));

END data_flow;
```

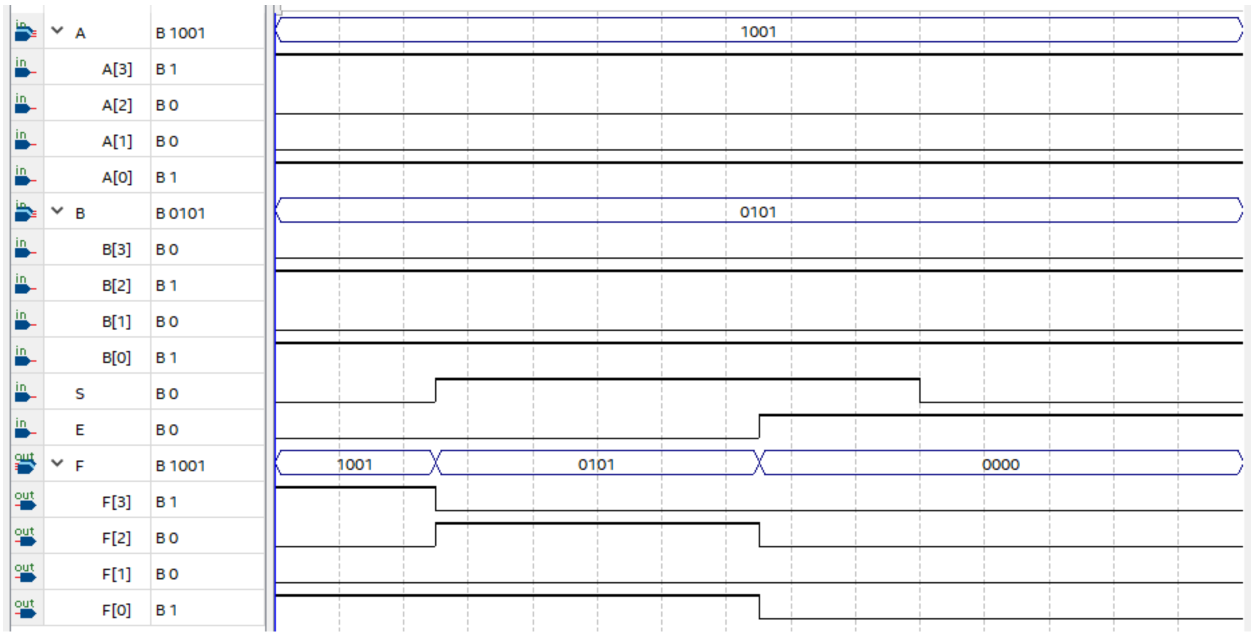
Quad 2:1 MUX Simulations:

a = 1011 b = 0101 (output = b if sel(0) = 0) **Note**: a(0) & b(0) are LSB

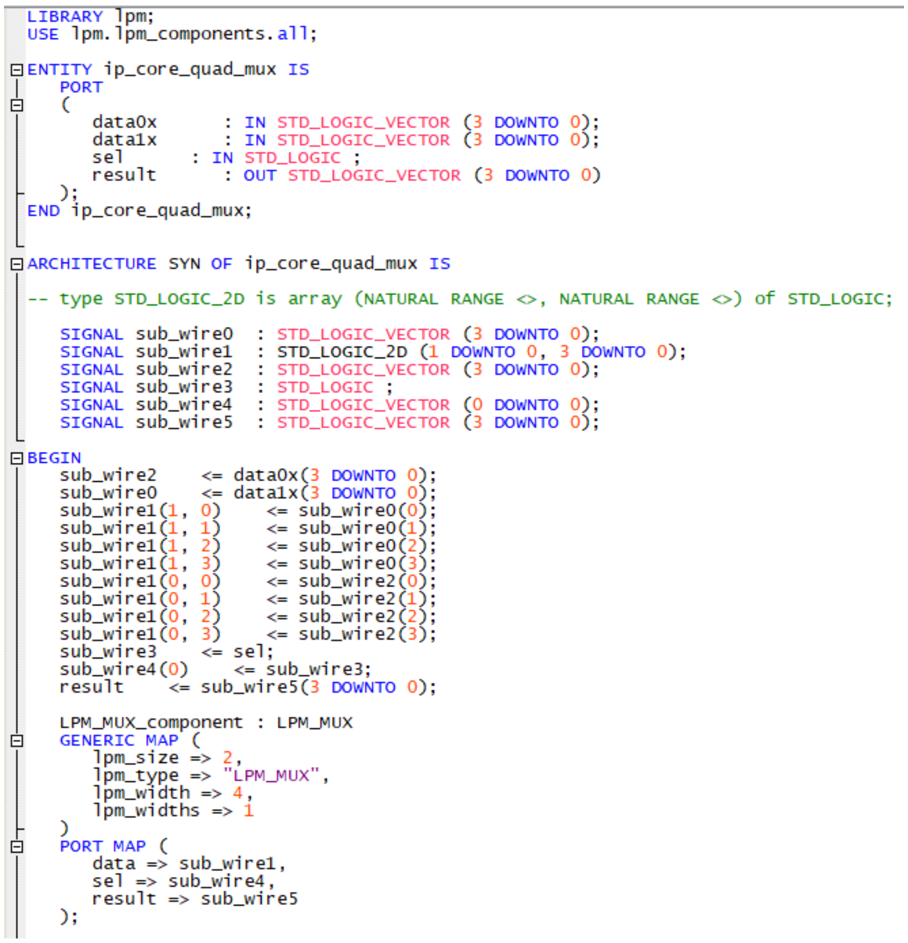


```
A     B 1011   1011
A[3]  B 1
A[2]  B 0
A[1]  B 1
A[0]  B 1
B     B 0101   0101
B[3]  B 0
B[2]  B 1
B[1]  B 0
B[0]  B 1
S     B 0
E     B 0
F     B 1011   1011 | 0101 | 0000
F[3]  B 1
F[2]  B 0
F[1]  B 1
F[0]  B 1
```

a = 1001 b = 1101



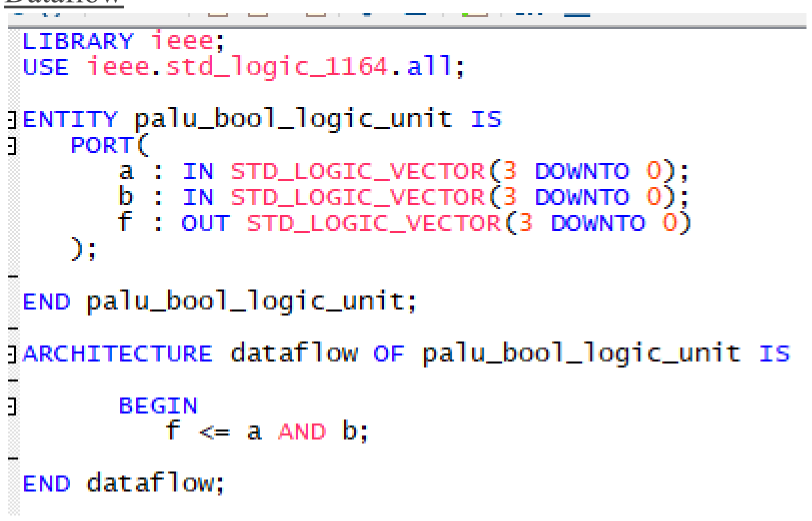
IP Core Code Screenshot



```
LIBRARY lpm;
USE lpm.lpm_components.all;

ENTITY ip_core_quad_mux IS
    PORT
    (
        data0x      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        data1x      : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        sel     : IN STD_LOGIC ;
        result      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
    );
END ip_core_quad_mux;

ARCHITECTURE SYN OF ip_core_quad_mux IS

-- type STD_LOGIC_2D is array (NATURAL RANGE <>, NATURAL RANGE <>) of STD_LOGIC;

    SIGNAL sub_wire0  : STD_LOGIC_VECTOR (3 DOWNTO 0);
    SIGNAL sub_wire1  : STD_LOGIC_2D (1 DOWNTO 0, 3 DOWNTO 0);
    SIGNAL sub_wire2  : STD_LOGIC_VECTOR (3 DOWNTO 0);
    SIGNAL sub_wire3  : STD_LOGIC ;
    SIGNAL sub_wire4  : STD_LOGIC_VECTOR (0 DOWNTO 0);
    SIGNAL sub_wire5  : STD_LOGIC_VECTOR (3 DOWNTO 0);

BEGIN
    sub_wire2    <= data0x(3 DOWNTO 0);
    sub_wire0    <= data1x(3 DOWNTO 0);
    sub_wire1(1, 0)    <= sub_wire0(0);
    sub_wire1(1, 1)    <= sub_wire0(1);
    sub_wire1(1, 2)    <= sub_wire0(2);
    sub_wire1(1, 3)    <= sub_wire0(3);
    sub_wire1(0, 0)    <= sub_wire2(0);
    sub_wire1(0, 1)    <= sub_wire2(1);
    sub_wire1(0, 2)    <= sub_wire2(2);
    sub_wire1(0, 3)    <= sub_wire2(3);
    sub_wire3    <= sel;
    sub_wire4(0)    <= sub_wire3;
    result    <= sub_wire5(3 DOWNTO 0);

    LPM_MUX_component : LPM_MUX
    GENERIC MAP (
        lpm_size => 2,
        lpm_type => "LPM_MUX",
        lpm_width => 4,
        lpm_widths => 1
    )
    PORT MAP (
        data => sub_wire1,
        sel => sub_wire4,
        result => sub_wire5
    );
```

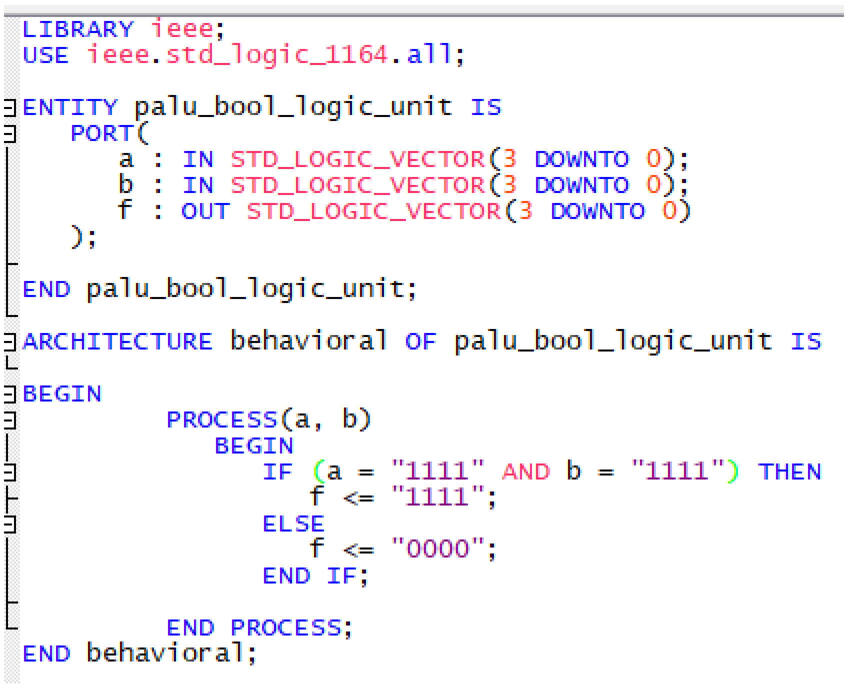
Boolean Logic Unit VHDL Code Screenshots

Dataflow



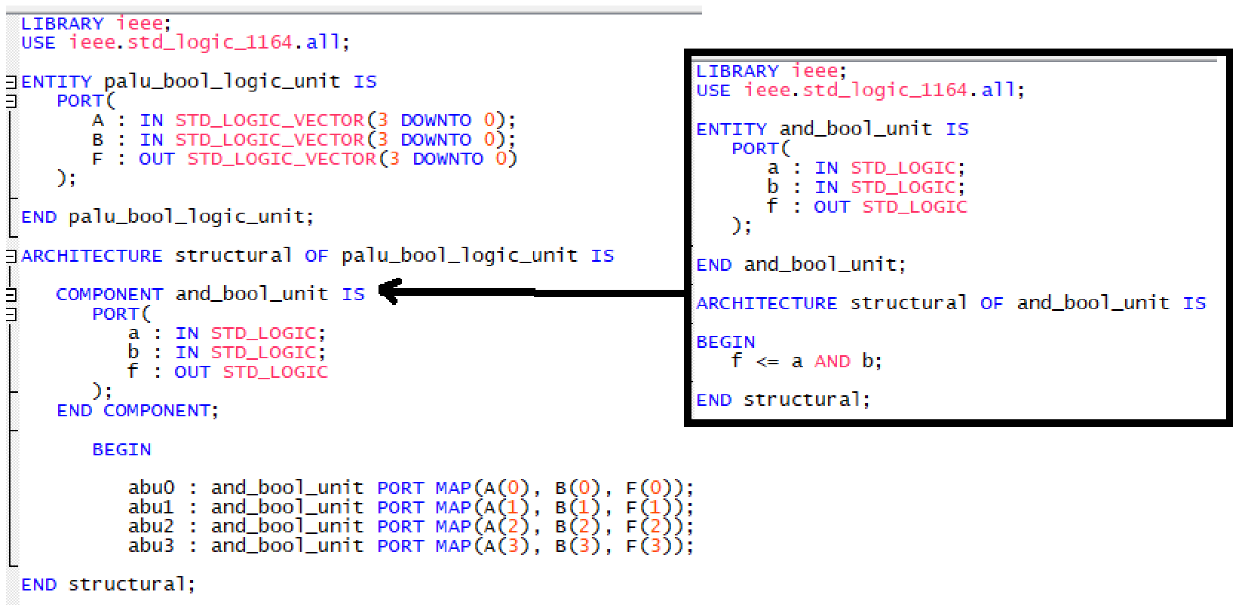
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY palu_bool_logic_unit IS
    PORT(
        a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        b : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        f : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );

END palu_bool_logic_unit;

ARCHITECTURE dataflow OF palu_bool_logic_unit IS

        BEGIN
            f <= a AND b;

END dataflow;
```

Behavioral



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY palu_bool_logic_unit IS
    PORT(
        a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        b : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        f : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );

END palu_bool_logic_unit;

ARCHITECTURE behavioral OF palu_bool_logic_unit IS

BEGIN
         PROCESS(a, b)
            BEGIN
                IF (a = "1111" AND b = "1111") THEN
                    f <= "1111";
                ELSE
                    f <= "0000";
                END IF;

         END PROCESS;
END behavioral;
```

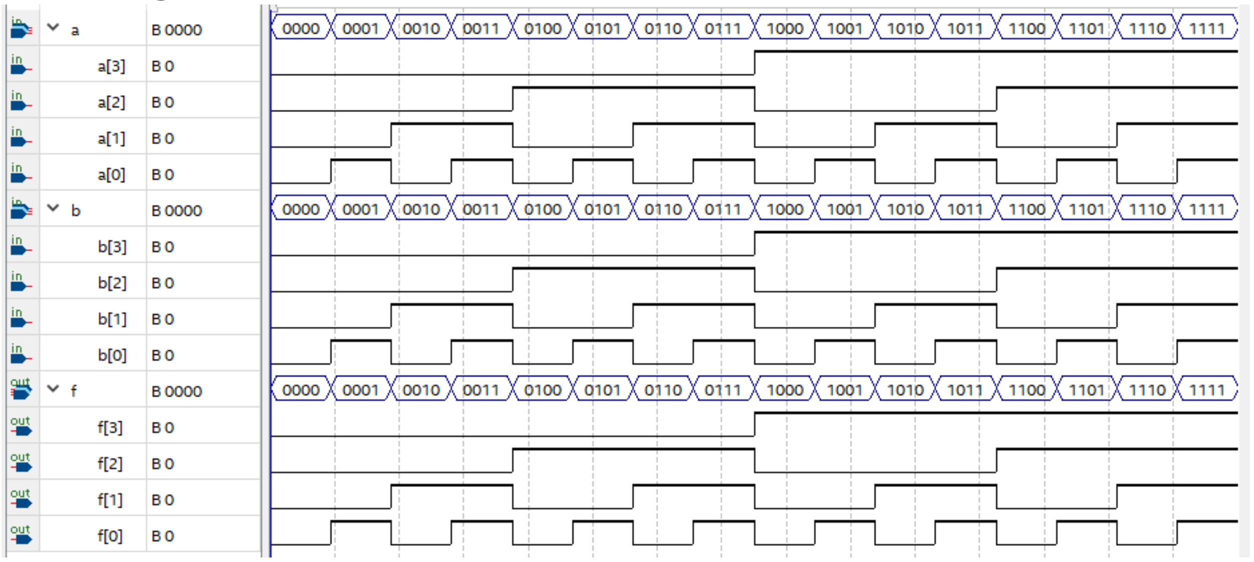
Structural

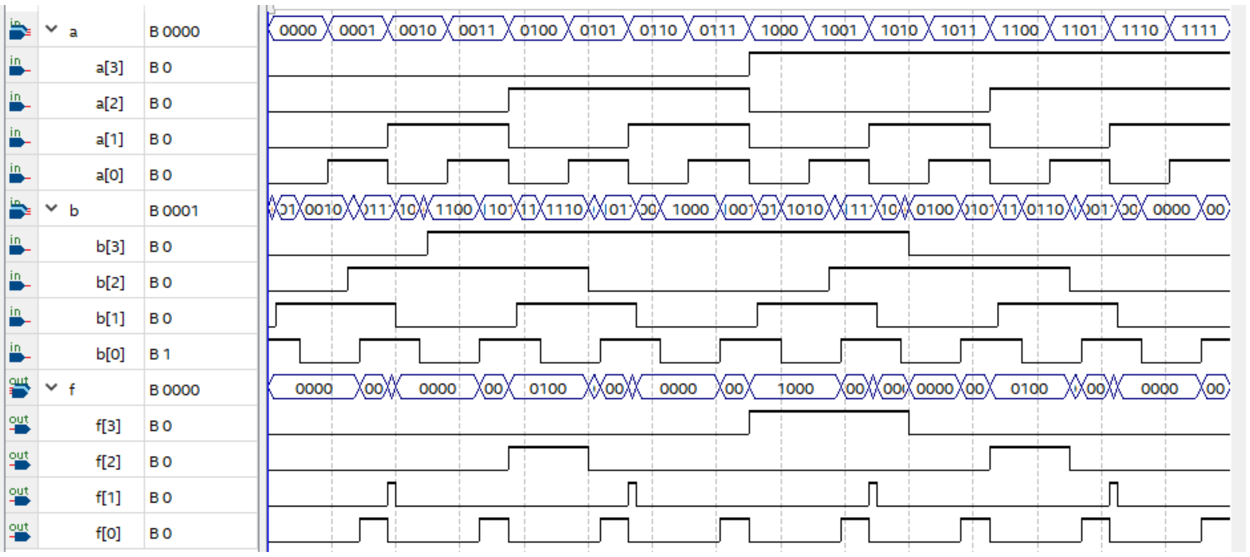


```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY palu_bool_logic_unit IS
    PORT(
        A : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        F : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );

END palu_bool_logic_unit;

ARCHITECTURE structural OF palu_bool_logic_unit IS

    COMPONENT and_bool_unit IS
        PORT(
            a : IN STD_LOGIC;
            b : IN STD_LOGIC;
            f : OUT STD_LOGIC
        );
    END COMPONENT;

        BEGIN

            abu0 : and_bool_unit PORT MAP(A(0), B(0), F(0));
            abu1 : and_bool_unit PORT MAP(A(1), B(1), F(1));
            abu2 : and_bool_unit PORT MAP(A(2), B(2), F(2));
            abu3 : and_bool_unit PORT MAP(A(3), B(3), F(3));

END structural;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY and_bool_unit IS
    PORT(
        a : IN STD_LOGIC;
        b : IN STD_LOGIC;
        f : OUT STD_LOGIC
    );

END and_bool_unit;

ARCHITECTURE structural OF and_bool_unit IS

BEGIN
    f <= a AND b;

END structural;
```

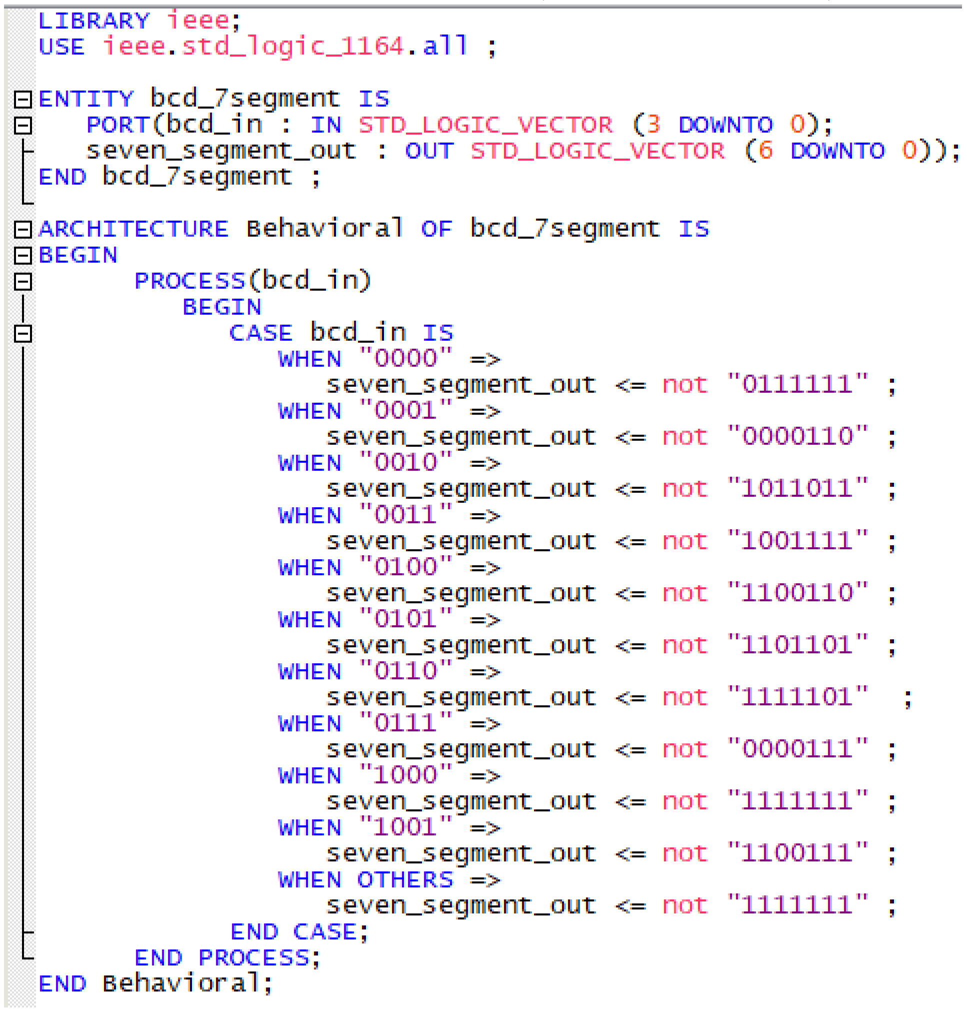
Boolean Logic Unit Simulations





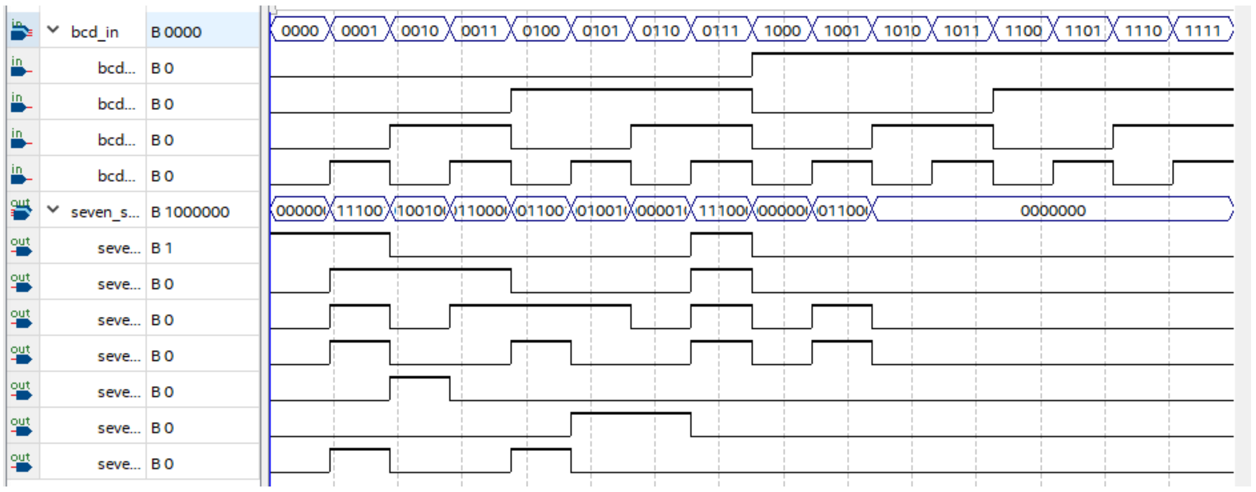
BCD Decoder

BCD Decoder VHDL Code Screenshot (Code from SWE-350)



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all ;

ENTITY bcd_7segment IS
   PORT(bcd_in : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
   seven_segment_out : OUT STD_LOGIC_VECTOR (6 DOWNTO 0));
END bcd_7segment ;

ARCHITECTURE Behavioral OF bcd_7segment IS
BEGIN
      PROCESS(bcd_in)
         BEGIN
            CASE bcd_in IS
               WHEN "0000" =>
                  seven_segment_out <= not "0111111" ;
               WHEN "0001" =>
                  seven_segment_out <= not "0000110" ;
               WHEN "0010" =>
                  seven_segment_out <= not "1011011" ;
               WHEN "0011" =>
                  seven_segment_out <= not "1001111" ;
               WHEN "0100" =>
                  seven_segment_out <= not "1100110" ;
               WHEN "0101" =>
                  seven_segment_out <= not "1101101" ;
               WHEN "0110" =>
                  seven_segment_out <= not "1111101"  ;
               WHEN "0111" =>
                  seven_segment_out <= not "0000111" ;
               WHEN "1000" =>
                  seven_segment_out <= not "1111111" ;
               WHEN "1001" =>
                  seven_segment_out <= not "1100111" ;
               WHEN OTHERS =>
                  seven_segment_out <= not "1111111" ;
            END CASE;
      END PROCESS;
END Behavioral;
```

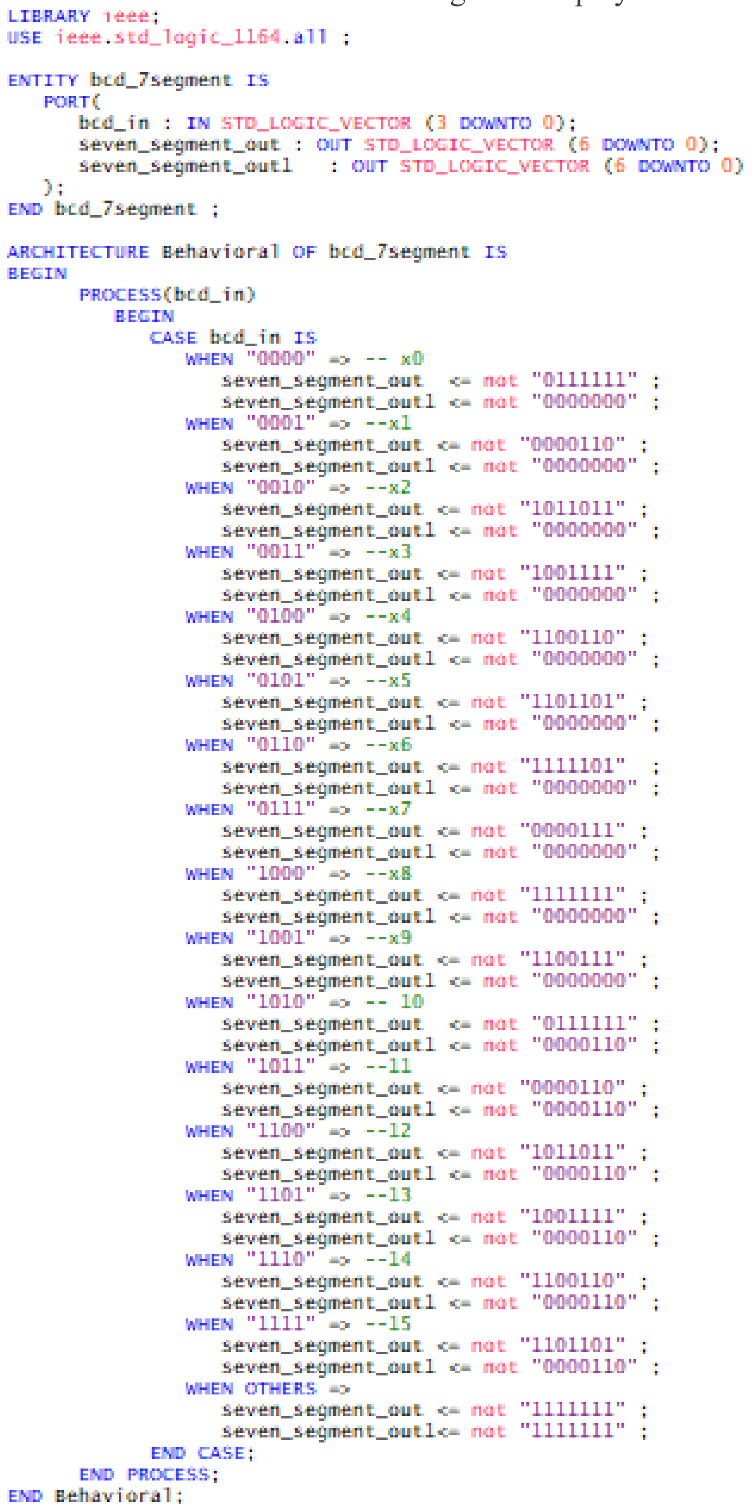
BCD Decoder Simulation



```
bcd_in      B 0000    0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
  bcd...    B 0
  bcd...    B 0
  bcd...    B 0
  bcd...    B 0
seven_s...  B 1000000 00000( 11100( 10010( 11000( 01100 01001( 00001( 11100( 00000( 01100( 0000000
  seve...   B 1
  seve...   B 0
  seve...   B 0
  seve...   B 0
  seve...   B 0
  seve...   B 0
  seve...   B 0
```

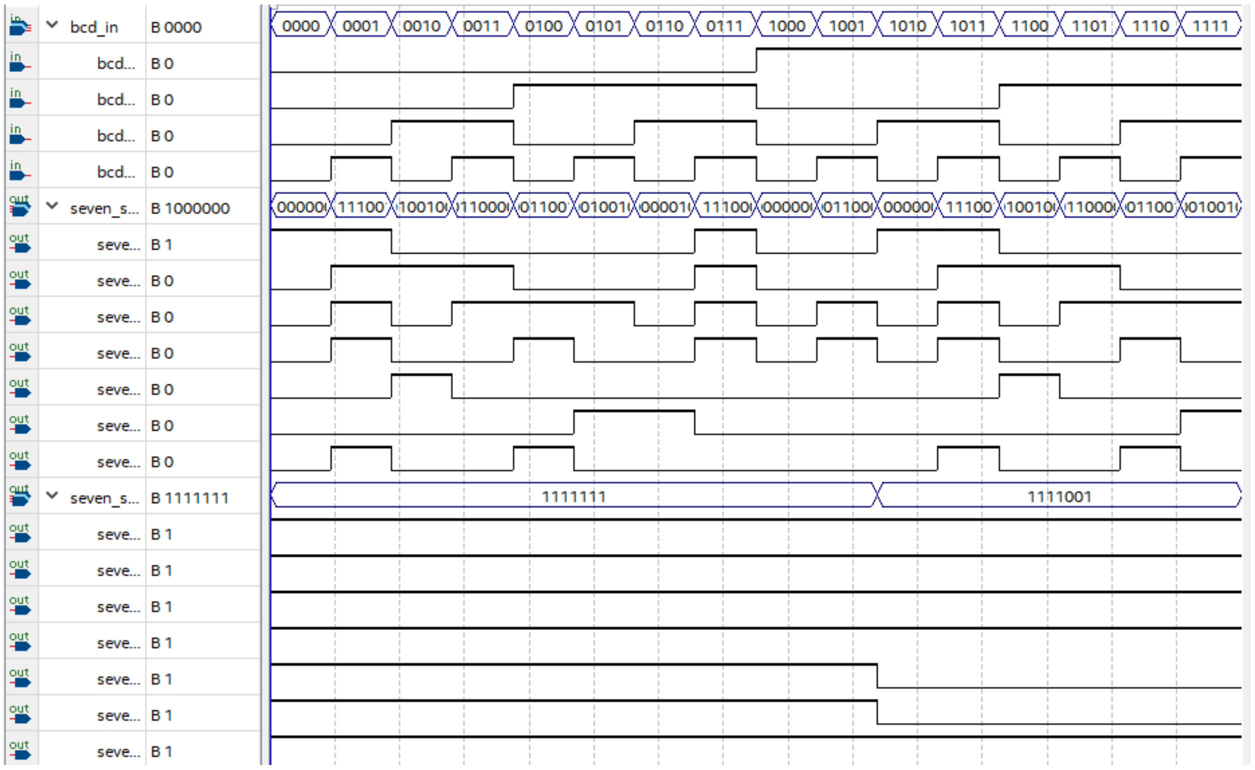
BCD Decoder(extended) VHDL Code Screenshot

Note: This code is derived from the code above (from SWE350) but includes some additions to handle numbers that are 4-bits wide with two seven segment displays

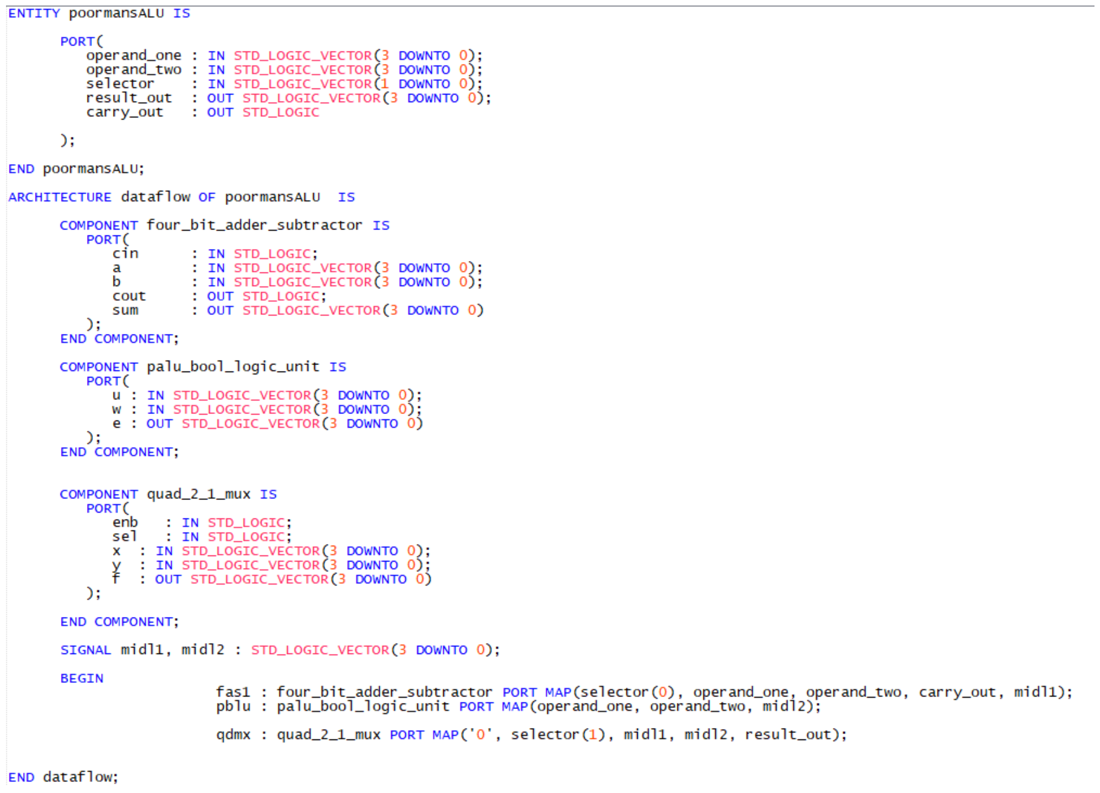


```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;

ENTITY bcd_7segment IS
    PORT(
        bcd_in : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        seven_segment_out : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
        seven_segment_out1    : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
    );
END bcd_7segment ;

ARCHITECTURE Behavioral OF bcd_7segment IS
BEGIN
        PROCESS(bcd_in)
            BEGIN
                CASE bcd_in IS
                    WHEN "0000" => -- x0
                        seven_segment_out  <= not "0111111" ;
                        seven_segment_out1 <= not "0000000" ;
                    WHEN "0001" => --x1
                        seven_segment_out <= not "0000110" ;
                        seven_segment_out1 <= not "0000000" ;
                    WHEN "0010" => --x2
                        seven_segment_out <= not "1011011" ;
                        seven_segment_out1 <= not "0000000" ;
                    WHEN "0011" => --x3
                        seven_segment_out <= not "1001111" ;
                        seven_segment_out1 <= not "0000000" ;
                    WHEN "0100" => --x4
                        seven_segment_out <= not "1100110" ;
                        seven_segment_out1 <= not "0000000" ;
                    WHEN "0101" => --x5
                        seven_segment_out <= not "1101101" ;
                        seven_segment_out1 <= not "0000000" ;
                    WHEN "0110" => --x6
                        seven_segment_out <= not "1111101"  ;
                        seven_segment_out1 <= not "0000000" ;
                    WHEN "0111" => --x7
                        seven_segment_out <= not "0000111" ;
                        seven_segment_out1 <= not "0000000" ;
                    WHEN "1000" => --x8
                        seven_segment_out <= not "1111111" ;
                        seven_segment_out1 <= not "0000000" ;
                    WHEN "1001" => --x9
                        seven_segment_out <= not "1100111" ;
                        seven_segment_out1 <= not "0000000" ;
                    WHEN "1010" => -- 10
                        seven_segment_out  <= not "0111111" ;
                        seven_segment_out1 <= not "0000110" ;
                    WHEN "1011" => --11
                        seven_segment_out <= not "0000110" ;
                        seven_segment_out1 <= not "0000110" ;
                    WHEN "1100" => --12
                        seven_segment_out <= not "1011011" ;
                        seven_segment_out1 <= not "0000110" ;
                    WHEN "1101" => --13
                        seven_segment_out <= not "1001111" ;
                        seven_segment_out1 <= not "0000110" ;
                    WHEN "1110" => --14
                        seven_segment_out <= not "1100110" ;
                        seven_segment_out1 <= not "0000110" ;
                    WHEN "1111" => --15
                        seven_segment_out <= not "1101101" ;
                        seven_segment_out1 <= not "0000110" ;
                    WHEN OTHERS =>
                        seven_segment_out <= not "1111111" ;
                        seven_segment_out1<= not "1111111" ;
                END CASE;
        END PROCESS;
END Behavioral;
```

(extended) BCD Decoder Simulation

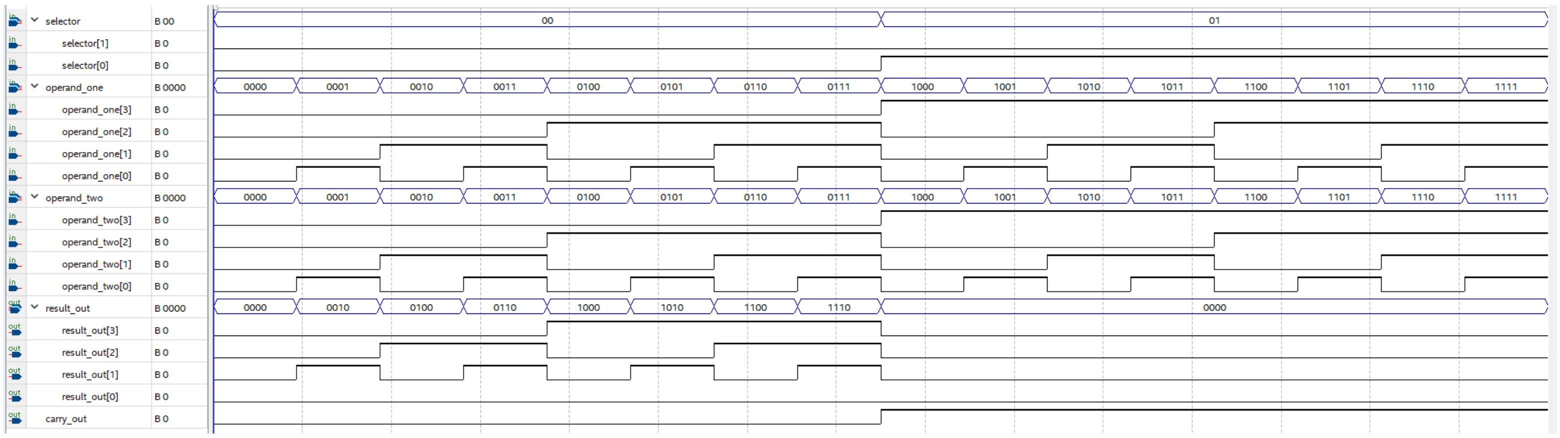


PoormansALU VHDL Code Screenshot

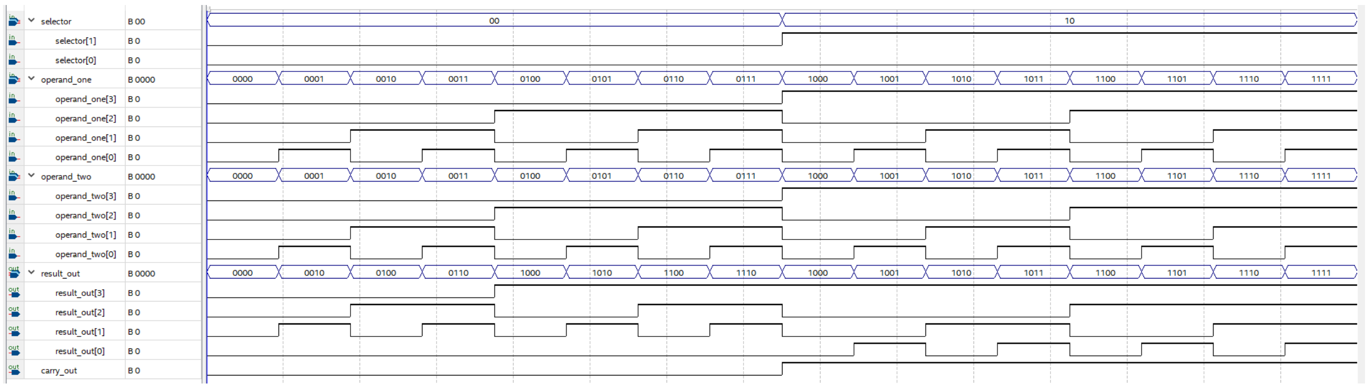


```
ENTITY poormansALU IS

      PORT(
         operand_one : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         operand_two : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         selector    : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
         result_out  : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
         carry_out   : OUT STD_LOGIC

      );

END poormansALU;

ARCHITECTURE dataflow OF poormansALU  IS

      COMPONENT four_bit_adder_subtractor IS
         PORT(
            cin      : IN STD_LOGIC;
            a        : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            b        : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            cout     : OUT STD_LOGIC;
            sum      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
         );
      END COMPONENT;

      COMPONENT palu_bool_logic_unit IS
         PORT(
            u : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            w : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            e : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
         );
      END COMPONENT;


      COMPONENT quad_2_1_mux IS
         PORT(
            enb   : IN STD_LOGIC;
            sel   : IN STD_LOGIC;
            x   : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            y   : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            f   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
         );

      END COMPONENT;

      SIGNAL midl1, midl2 : STD_LOGIC_VECTOR(3 DOWNTO 0);

      BEGIN
                        fas1 : four_bit_adder_subtractor PORT MAP(selector(0), operand_one, operand_two, carry_out, midl1);
                        pblu : palu_bool_logic_unit PORT MAP(operand_one, operand_two, midl2);

                        qdmx : quad_2_1_mux PORT MAP('0', selector(1), midl1, midl2, result_out);

END dataflow;
```

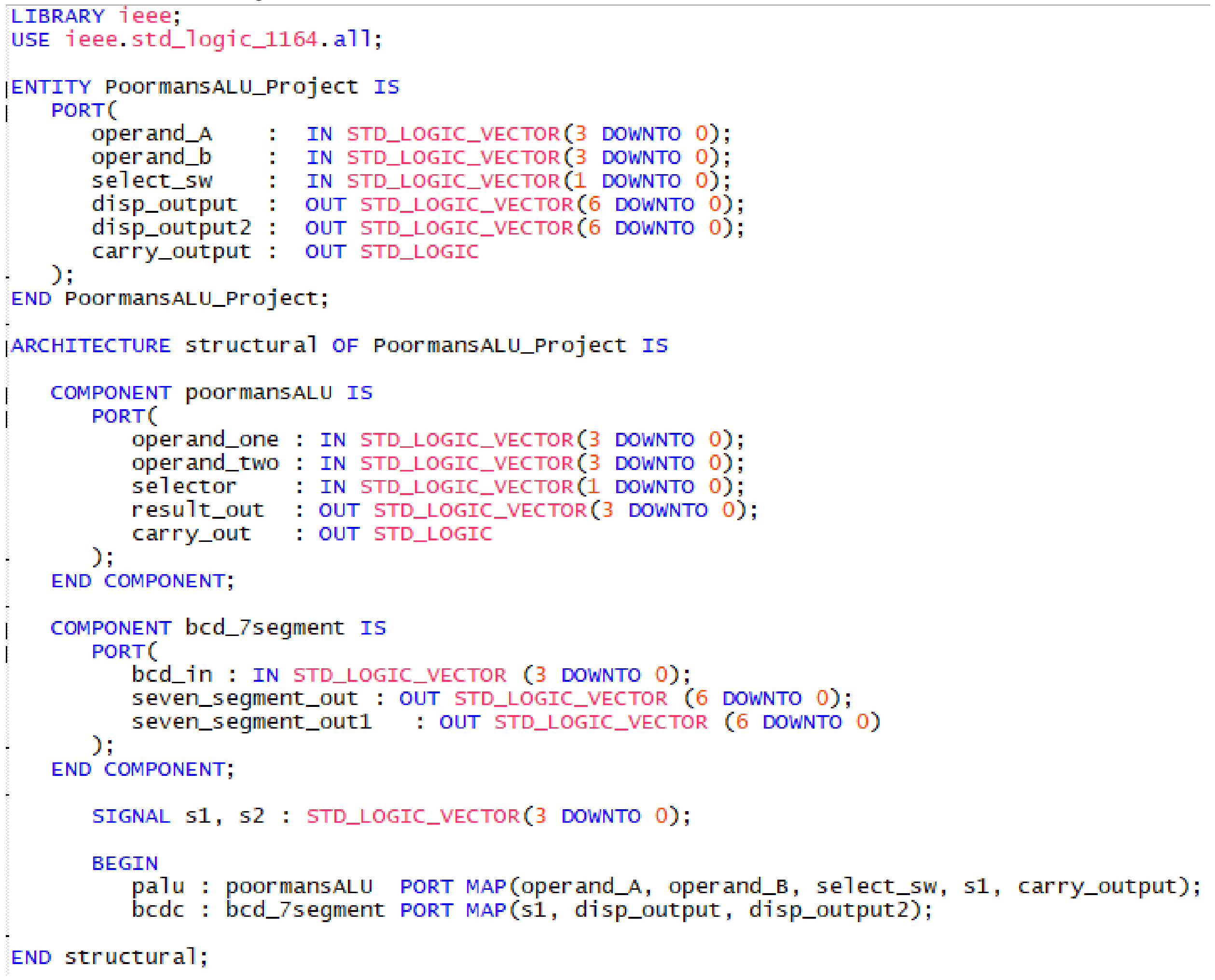
PoormansALU Waveform Simulation (addition & subtraction)



PoormansALU Waveform Simulation (boolean logic & addition)

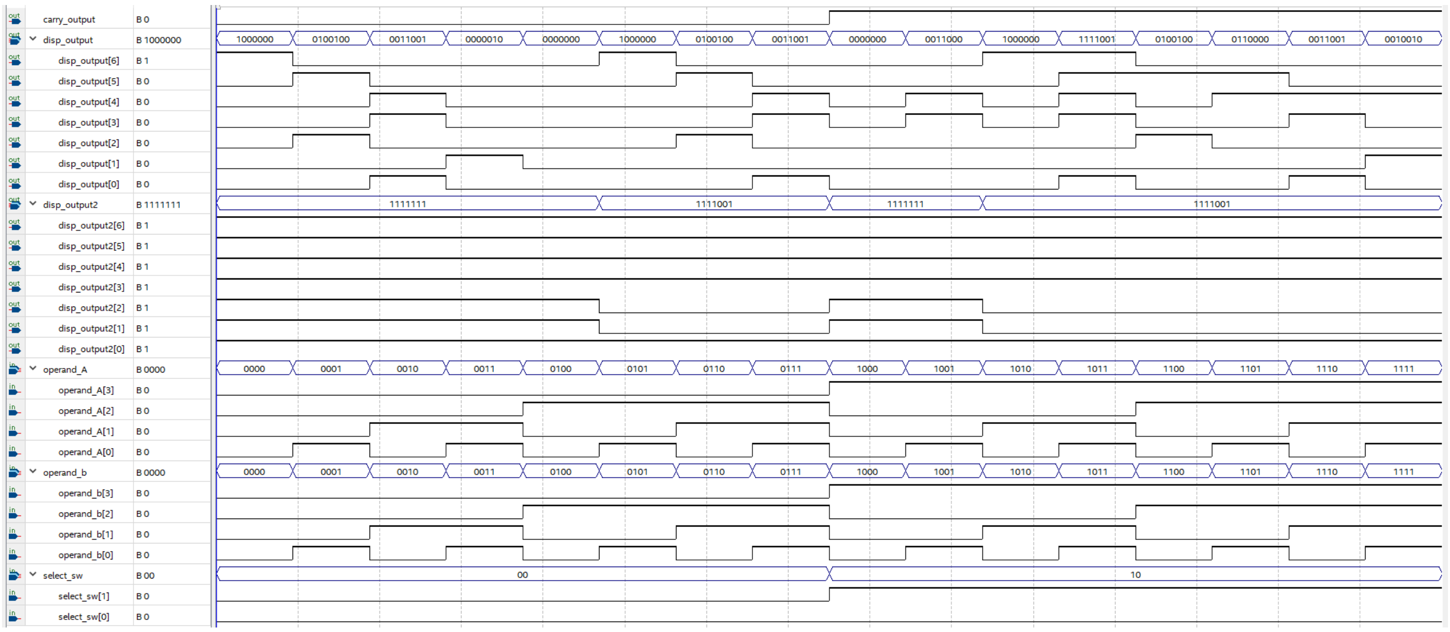


PoormansALU Project VHDL Code Screenshot



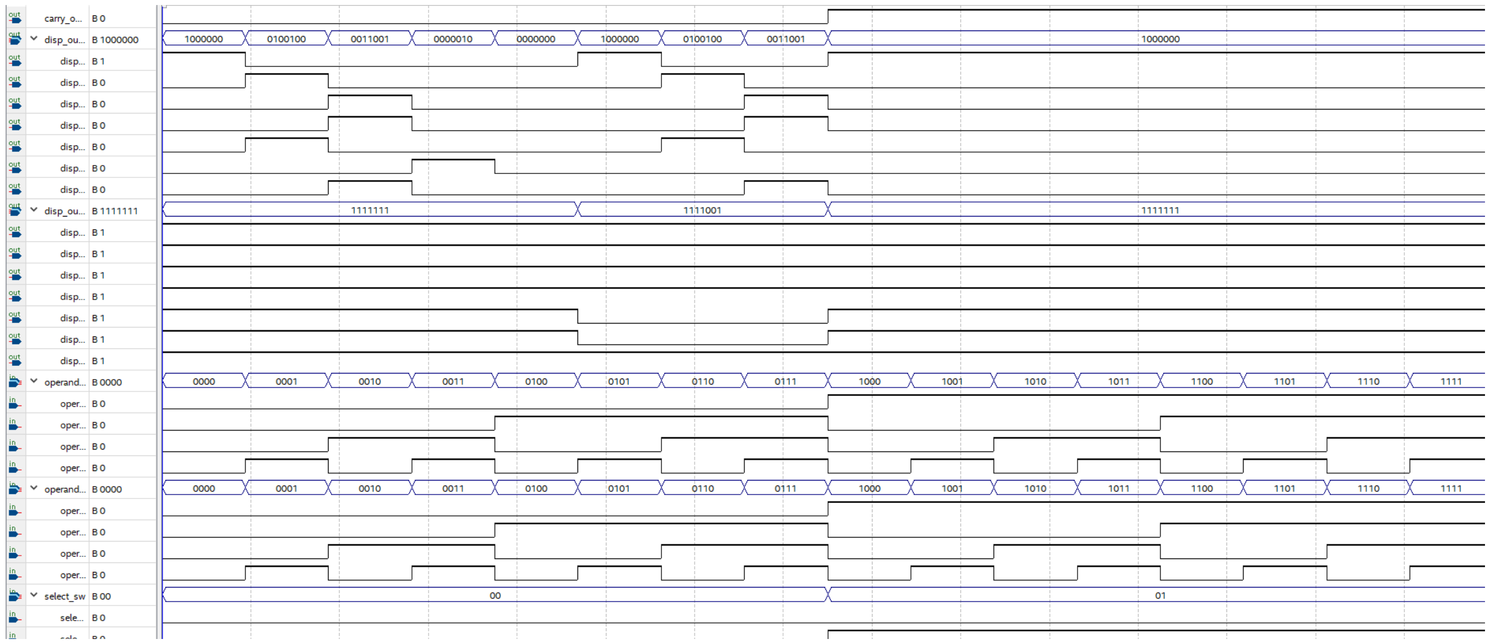
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY PoormansALU_Project IS
   PORT(
      operand_A     :  IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      operand_b     :  IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      select_sw     :  IN STD_LOGIC_VECTOR(1 DOWNTO 0);
      disp_output   :  OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      disp_output2 :  OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      carry_output :  OUT STD_LOGIC
   );
END PoormansALU_Project;

ARCHITECTURE structural OF PoormansALU_Project IS

   COMPONENT poormansALU IS
      PORT(
         operand_one : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         operand_two : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         selector    : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
         result_out  : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
         carry_out   : OUT STD_LOGIC
      );
   END COMPONENT;

   COMPONENT bcd_7segment IS
      PORT(
         bcd_in : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
         seven_segment_out : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
         seven_segment_out1    : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
      );
   END COMPONENT;

      SIGNAL s1, s2 : STD_LOGIC_VECTOR(3 DOWNTO 0);

      BEGIN
         palu : poormansALU  PORT MAP(operand_A, operand_B, select_sw, s1, carry_output);
         bcdc : bcd_7segment PORT MAP(s1, disp_output, disp_output2);

END structural;
```

PoormansALU Project Waveform Simulation (addition & subtraction)



```
carry_output        B 0
disp_output         B 1000000
  disp_output[6]    B 1
  disp_output[5]    B 0
  disp_output[4]    B 0
  disp_output[3]    B 0
  disp_output[2]    B 0
  disp_output[1]    B 0
  disp_output[0]    B 0
disp_output2        B 1111111
  disp_output2[6]   B 1
  disp_output2[5]   B 1
  disp_output2[4]   B 1
  disp_output2[3]   B 1
  disp_output2[2]   B 1
  disp_output2[1]   B 1
  disp_output2[0]   B 1
operand_A           B 0000
  operand_A[3]      B 0
  operand_A[2]      B 0
  operand_A[1]      B 0
  operand_A[0]      B 0
operand_b           B 0000
  operand_b[3]      B 0
  operand_b[2]      B 0
  operand_b[1]      B 0
  operand_b[0]      B 0
select_sw           B 00
  select_sw[1]      B 0
  select_sw[0]      B 0
```

PoormansALU Project Waveform Simulation (boolean logic & addition)

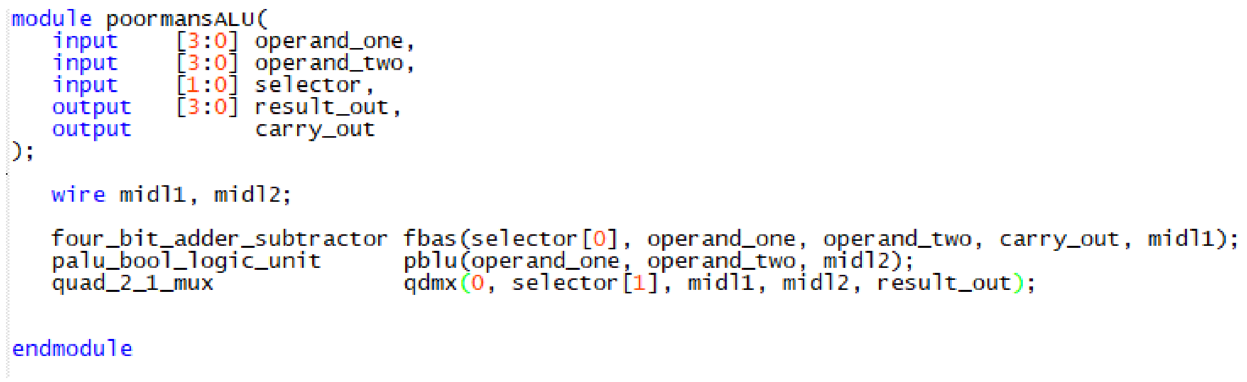


Risks & Issues

N/A. - this is the final release for this project.

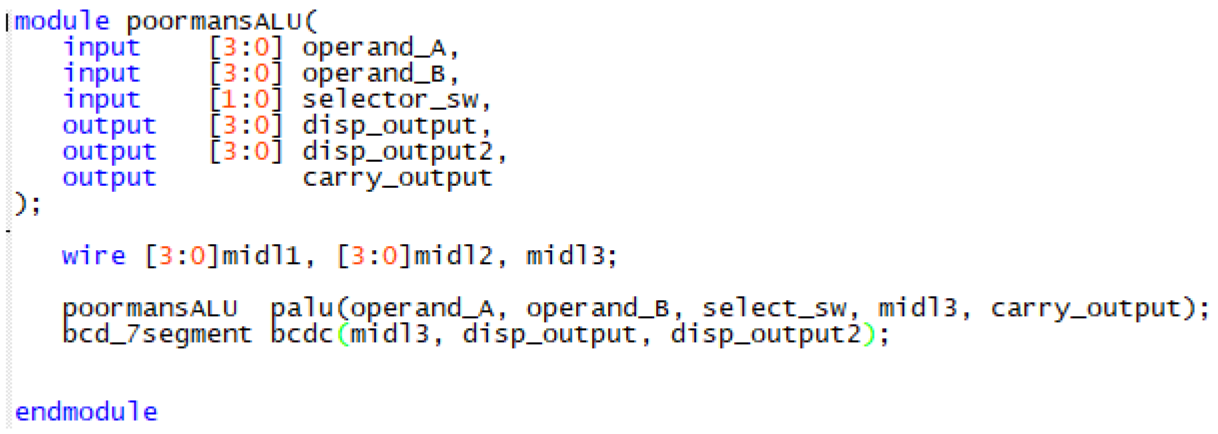
Other Documentation

PoormansALU - Verilog



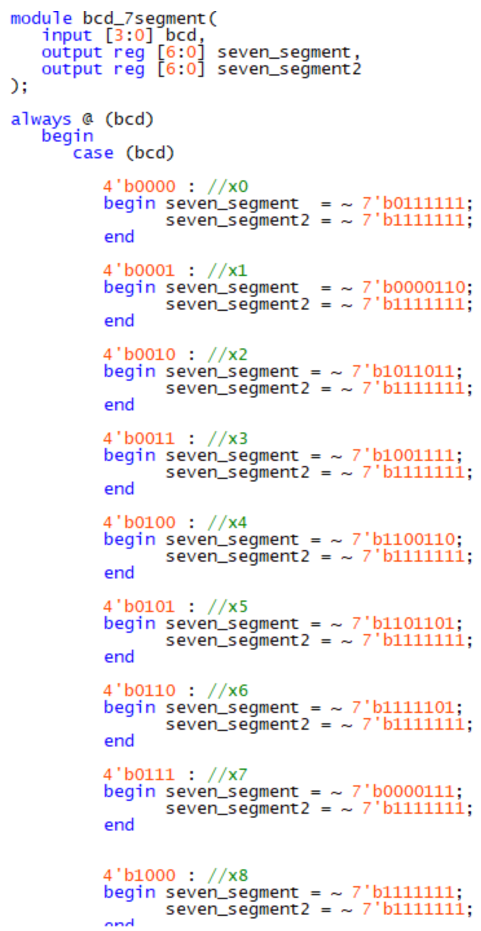
```
module poormansALU(
    input     [3:0] operand_one,
    input     [3:0] operand_two,
    input     [1:0] selector,
    output    [3:0] result_out,
    output          carry_out
);

    wire midl1, midl2;

    four_bit_adder_subtractor fbas(selector[0], operand_one, operand_two, carry_out, midl1);
    palu_bool_logic_unit      pblu(operand_one, operand_two, midl2);
    quad_2_1_mux              qdmx(0, selector[1], midl1, midl2, result_out);


endmodule
```

PoormansALU_Project - Verilog

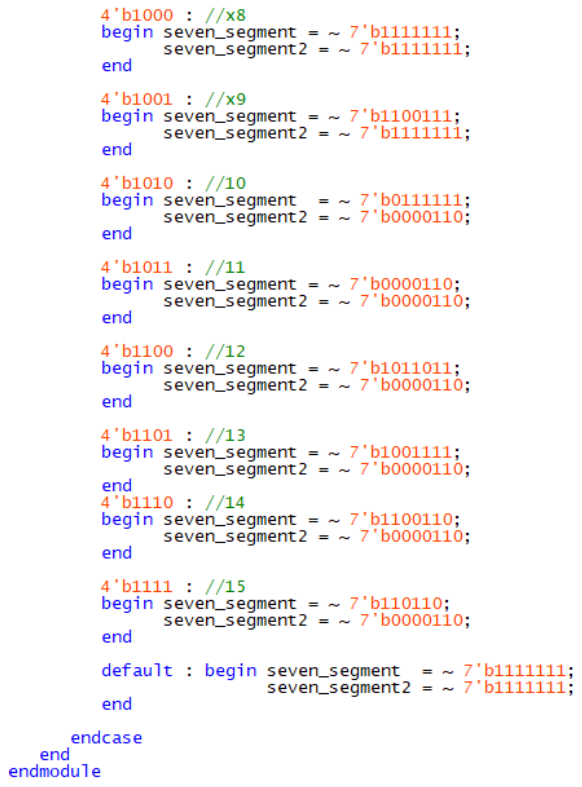


```
module poormansALU(
    input     [3:0] operand_A,
    input     [3:0] operand_B,
    input     [1:0] selector_sw,
    output    [3:0] disp_output,
    output    [3:0] disp_output2,
    output          carry_output
);

    wire [3:0]midl1, [3:0]midl2, midl3;

    poormansALU  palu(operand_A, operand_B, select_sw, midl3, carry_output);
    bcd_7segment bcdc(midl3, disp_output, disp_output2);


endmodule
```

Extended BCD Decoder - Verilog

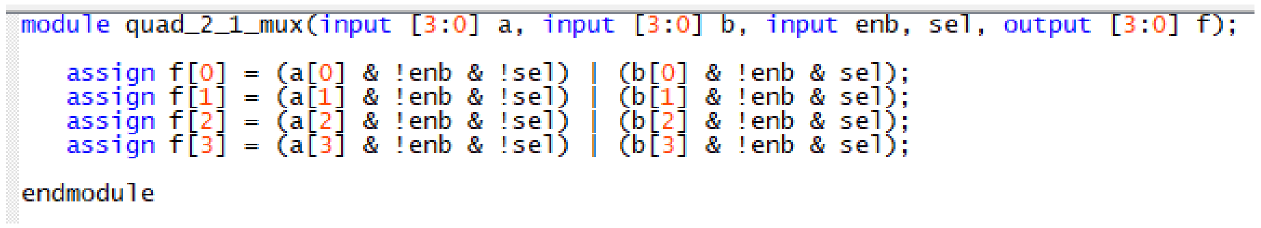


```verilog
module bcd_7segment(
    input [3:0] bcd,
    output reg [6:0] seven_segment,
    output reg [6:0] seven_segment2
);

always @ (bcd)
    begin
        case (bcd)

            4'b0000 : //x0
            begin seven_segment  = ~ 7'b0111111;
                    seven_segment2 = ~ 7'b1111111;
            end

            4'b0001 : //x1
            begin seven_segment  = ~ 7'b0000110;
                    seven_segment2 = ~ 7'b1111111;
            end

            4'b0010 : //x2
            begin seven_segment = ~ 7'b1011011;
                    seven_segment2 = ~ 7'b1111111;
            end

            4'b0011 : //x3
            begin seven_segment = ~ 7'b1001111;
                    seven_segment2 = ~ 7'b1111111;
            end

            4'b0100 : //x4
            begin seven_segment = ~ 7'b1100110;
                    seven_segment2 = ~ 7'b1111111;
            end

            4'b0101 : //x5
            begin seven_segment = ~ 7'b1101101;
                    seven_segment2 = ~ 7'b1111111;
            end

            4'b0110 : //x6
            begin seven_segment = ~ 7'b1111101;
                    seven_segment2 = ~ 7'b1111111;
            end

            4'b0111 : //x7
            begin seven_segment = ~ 7'b0000111;
                    seven_segment2 = ~ 7'b1111111;
            end


            4'b1000 : //x8
            begin seven_segment = ~ 7'b1111111;
                    seven_segment2 = ~ 7'b1111111;
```



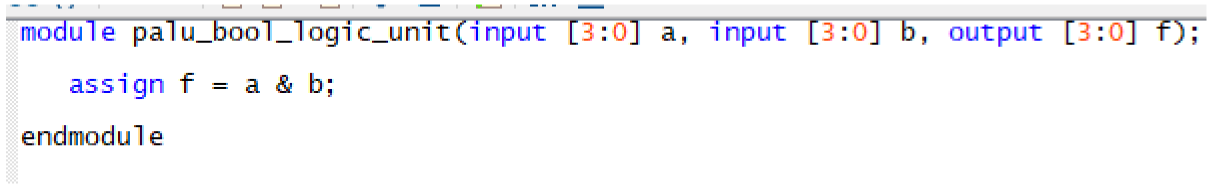
```verilog
            4'b1000 : //x8
            begin seven_segment = ~ 7'b1111111;
                    seven_segment2 = ~ 7'b1111111;
            end

            4'b1001 : //x9
            begin seven_segment = ~ 7'b1100111;
                    seven_segment2 = ~ 7'b1111111;
            end

            4'b1010 : //10
            begin seven_segment  = ~ 7'b0111111;
                    seven_segment2 = ~ 7'b0000110;
            end

            4'b1011 : //11
            begin seven_segment = ~ 7'b0000110;
                    seven_segment2 = ~ 7'b0000110;
            end

            4'b1100 : //12
            begin seven_segment = ~ 7'b1011011;
                    seven_segment2 = ~ 7'b0000110;
            end

            4'b1101 : //13
            begin seven_segment = ~ 7'b1001111;
                    seven_segment2 = ~ 7'b0000110;
            end
            4'b1110 : //14
            begin seven_segment = ~ 7'b1100110;
                    seven_segment2 = ~ 7'b0000110;
            end

            4'b1111 : //15
            begin seven_segment = ~ 7'b110110;
                    seven_segment2 = ~ 7'b0000110;
            end

            default : begin seven_segment  = ~ 7'b1111111;
                            seven_segment2 = ~ 7'b1111111;
            end

        endcase
    end
endmodule
```

Quad 2:1 MUX - Verilog



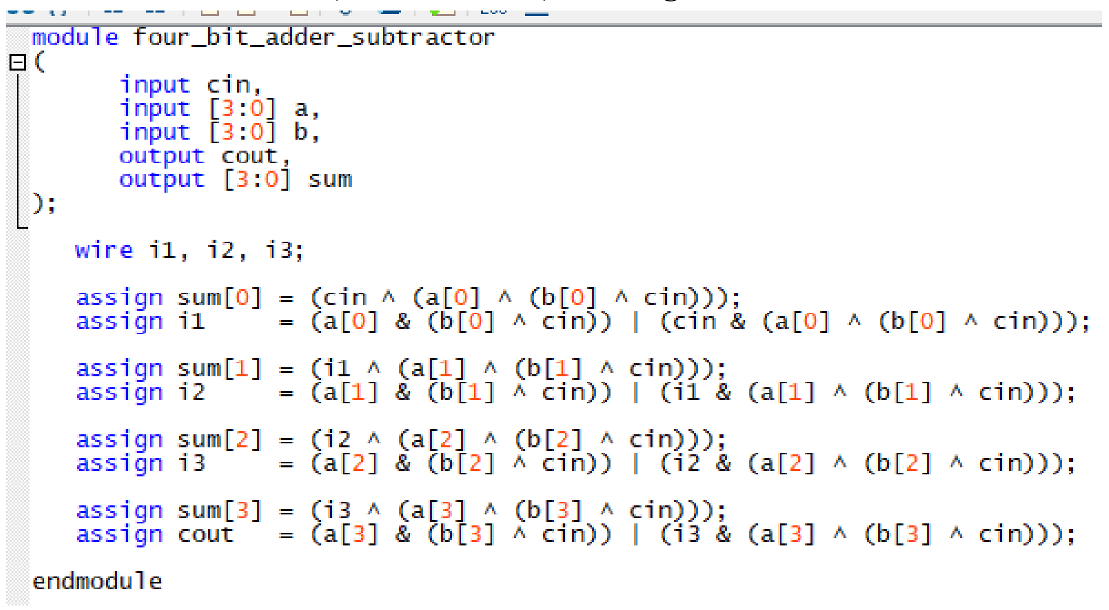
```verilog
module quad_2_1_mux(input [3:0] a, input [3:0] b, input enb, sel, output [3:0] f);

   assign f[0] = (a[0] & !enb & !sel) | (b[0] & !enb & sel);
   assign f[1] = (a[1] & !enb & !sel) | (b[1] & !enb & sel);
   assign f[2] = (a[2] & !enb & !sel) | (b[2] & !enb & sel);
   assign f[3] = (a[3] & !enb & !sel) | (b[3] & !enb & sel);

endmodule
```

Boolean Logic Unit - Verilog (AND two 4-bit inputs)



```verilog
module palu_bool_logic_unit(input [3:0] a, input [3:0] b, output [3:0] f);

   assign f = a & b;

endmodule
```

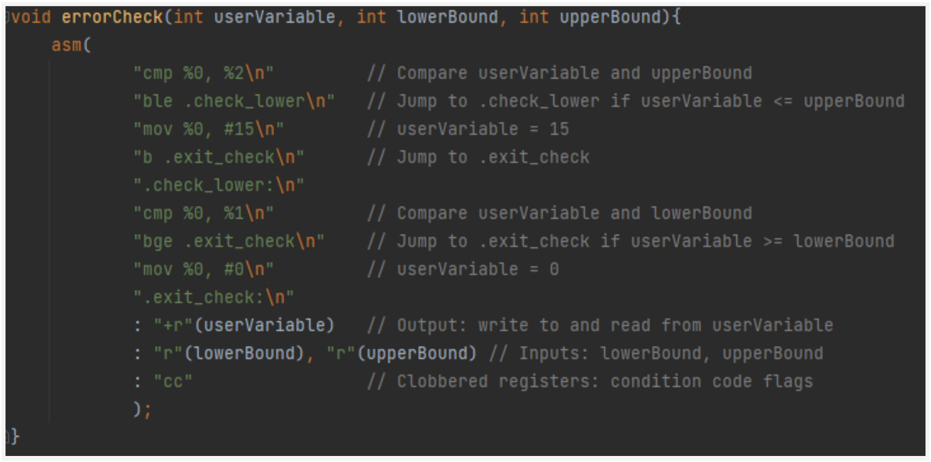
4-bit Adder/Subtractor (arithmetic unit) - Verilog



```verilog
module four_bit_adder_subtractor
(
      input cin,
      input [3:0] a,
      input [3:0] b,
      output cout,
      output [3:0] sum
);

   wire i1, i2, i3;

   assign sum[0] = (cin ^ (a[0] ^ (b[0] ^ cin)));
   assign i1     = (a[0] & (b[0] ^ cin)) | (cin & (a[0] ^ (b[0] ^ cin)));

   assign sum[1] = (i1 ^ (a[1] ^ (b[1] ^ cin)));
   assign i2     = (a[1] & (b[1] ^ cin)) | (i1 & (a[1] ^ (b[1] ^ cin)));

   assign sum[2] = (i2 ^ (a[2] ^ (b[2] ^ cin)));
   assign i3     = (a[2] & (b[2] ^ cin)) | (i2 & (a[2] ^ (b[2] ^ cin)));

   assign sum[3] = (i3 ^ (a[3] ^ (b[3] ^ cin)));
   assign cout   = (a[3] & (b[3] ^ cin)) | (i3 & (a[3] ^ (b[3] ^ cin)));

endmodule
```

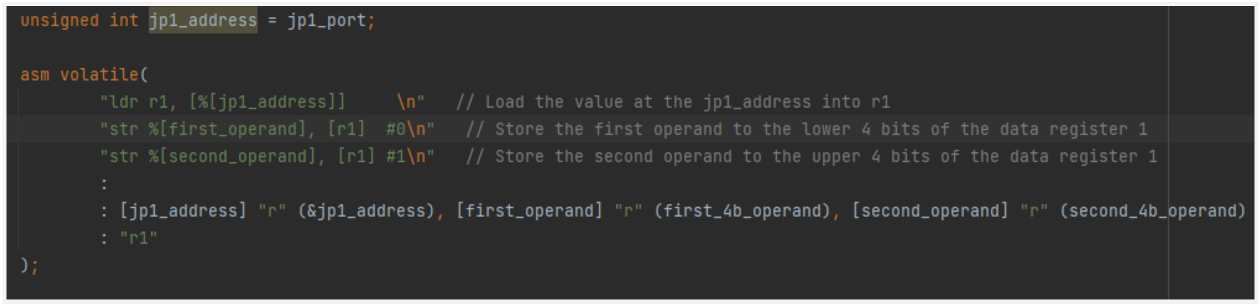
Arm Assembly

Error Check Function - alu_module



```
void errorCheck(int userVariable, int lowerBound, int upperBound){
    asm(
            "cmp %0, %2\n"          // Compare userVariable and upperBound
            "ble .check_lower\n"    // Jump to .check_lower if userVariable <= upperBound
            "mov %0, #15\n"         // userVariable = 15
            "b .exit_check\n"       // Jump to .exit_check
            ".check_lower:\n"
            "cmp %0, %1\n"          // Compare userVariable and lowerBound
            "bge .exit_check\n"     // Jump to .exit_check if userVariable >= lowerBound
            "mov %0, #0\n"          // userVariable = 0
            ".exit_check:\n"
            : "+r"(userVariable)    // Output: write to and read from userVariable
            : "r"(lowerBound), "r"(upperBound) // Inputs: lowerBound, upperBound
            : "cc"                  // Clobbered registers: condition code flags
            );
}
```

WriteToParallelPort Function - hardware_module



```
unsigned int jp1_address = jp1_port;

asm volatile(
        "ldr r1, [%[jp1_address]]     \n"   // Load the value at the jp1_address into r1
        "str %[first_operand], [r1]  #0\n"   // Store the first operand to the lower 4 bits of the data register 1
        "str %[second_operand], [r1] #1\n"   // Store the second operand to the upper 4 bits of the data register 1
        :
        : [jp1_address] "r" (&jp1_address), [first_operand] "r" (first_4b_operand), [second_operand] "r" (second_4b_operand)
        : "r1"
);
```

Resources

Documentation – Arm Developer. (n.d.).

https://developer.arm.com/documentation/dui0473/c/writing-arm-assembly-language

Writing ARM Assembly (Part 1). (n.d.). Azeria-Labs.

https://azeria-labs.com/writing-arm-assembly-part-1/