LAB 0: BIT COMPARATOR

Brandon Chin

CSC343 - Instructor: Prof. Izidor Gertner

Brandon Chin **CSC343**

Lab 0: Intro to VHDL (Comparator)

Objective:

To learn the basics of VHDL and ModelSim by designing and testing 1-bit, 2-bit, and 8-bit comparators.

Functionality and Specifications:

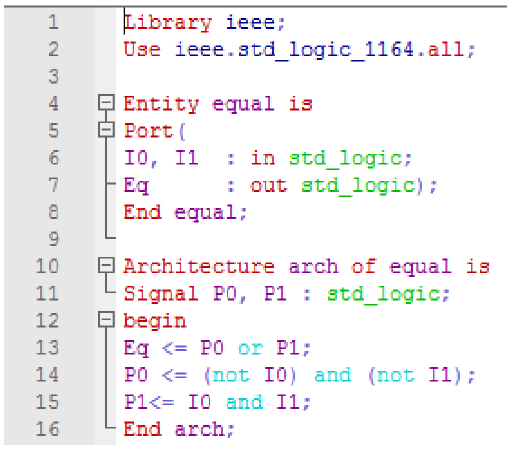
1-Bit Comparator

A comparator is designed to take in two inputs and check for equality. The comparator then returns the boolean true or false (0 = false, 1 = true). In this lab, we are comparing two bits, and checking if they are equal. We begin with 1-bit inputs, each having a value of either 0 or 1. The truth table is shown below,

Input		Output
I0	I1	Eq
0	0	1
0	1	0
1	0	0
1	1	1

Truth table of a one-bit comparator

Our design uses only logical operators and represents a gate-level combinational circuit, composed of logic gates. The logical expression is **Eq = [I0 and I1] or [(not I0) and (not I1)]**. Here is the VHDL code describing the above expression:



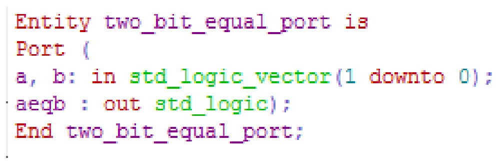
```
Library ieee;
Use ieee.std_logic_1164.all;

Entity equal is
Port(
I0, I1   : in std_logic;
Eq       : out std_logic);
End equal;

Architecture arch of equal is
Signal P0, P1 : std_logic;
begin
Eq <= P0 or P1;
P0 <= (not I0) and (not I1);
P1<= I0 and I1;
End arch;
```

In the Entity block, we have defined an entity called equal, as well as its inputs and outputs.

The architecture of equal includes the logical expression stated above, which defines the behavior of the circuit.

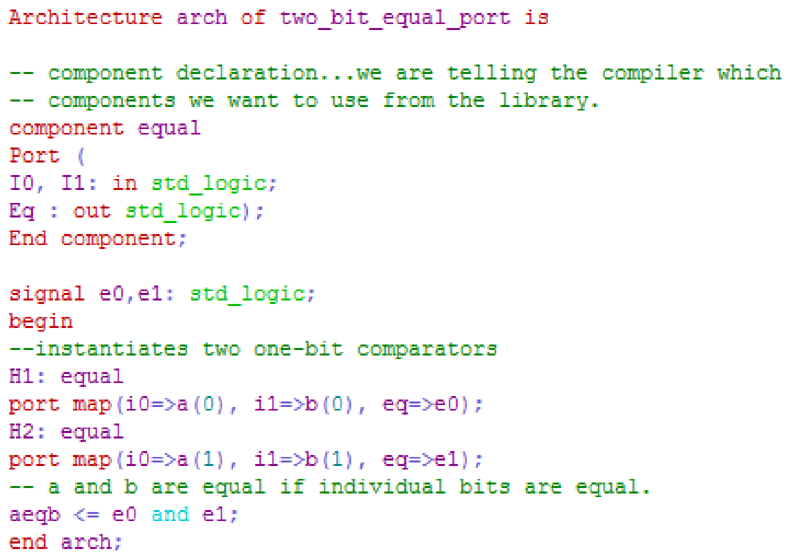
2-Bit Comparator

Now, we're going to expand upon our 1-bit comparator and compare 2-bit numbers. This time we have twice as many possible inputs that our circuit could receive (00,01,10,11). There are two ways we can approach this: we can either derive a new logical expression for this circuit (we get **Eq = [(not a(1))(not b(1))(not a(0))(not b(0))] + [(not a(1))(not b(1))a(0)b(0)] + [a(1)b(1)(not a(0))(not b(0))] + [a(1)b(1)a(0)b(0)]**), or we can use port maps to combine two 1-bit comparators to make a 2-bit comparator. The first method can become complicated, as you might imagine, especially if we try to make larger comparators (*i.e.* 4-bit comparators, or 8-bit comparators, etc.), therefore, we will use the second method, and implement port maps. Before doing so, we must first redefine our entity:



```
Entity two_bit_equal_port is
Port (
a, b: in std_logic_vector(1 downto 0);
aeqb : out std_logic);
End two_bit_equal_port;
```

We are using a vector to store multiple bits for each input (*i.e.* input 'a' has two bits, a(1) and a(0)).



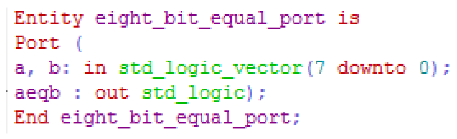
```
Architecture arch of two_bit_equal_port is

-- component declaration...we are telling the compiler which
-- components we want to use from the library.
component equal
Port (
I0, I1: in std_logic;
Eq : out std_logic);
End component;

signal e0,e1: std_logic;
begin
--instantiates two one-bit comparators
H1: equal
port map(i0=>a(0), i1=>b(0), eq=>e0);
H2: equal
port map(i0=>a(1), i1=>b(1), eq=>e1);
-- a and b are equal if individual bits are equal.
aeqb <= e0 and e1;
end arch;
```

Inside the architecture of this new entity, we must include the component of the 1-bit comparator from before.

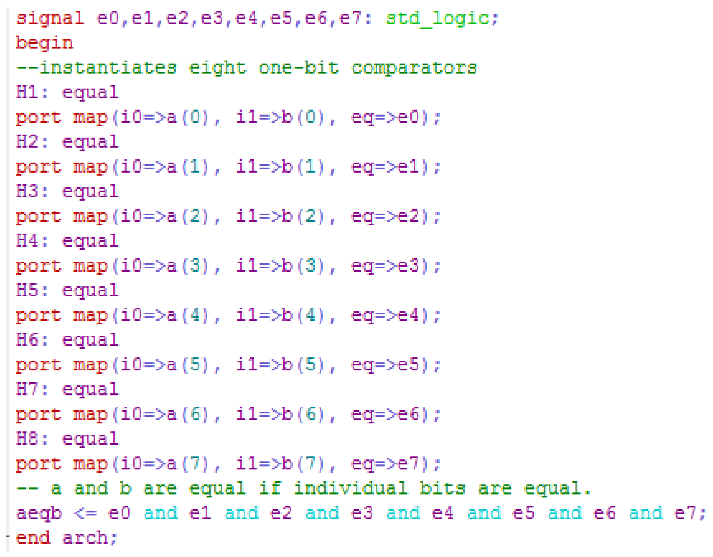
Finally, we implement the port maps between each component.

8-Bit Comparator

For the 8-bit comparator, we use the same method as the 2-bit comparator. Modified segments of the VHDL is shown below:



```
Entity eight_bit_equal_port is
Port (
a, b: in std_logic_vector(7 downto 0);
aeqb : out std_logic);
End eight_bit_equal_port;
```

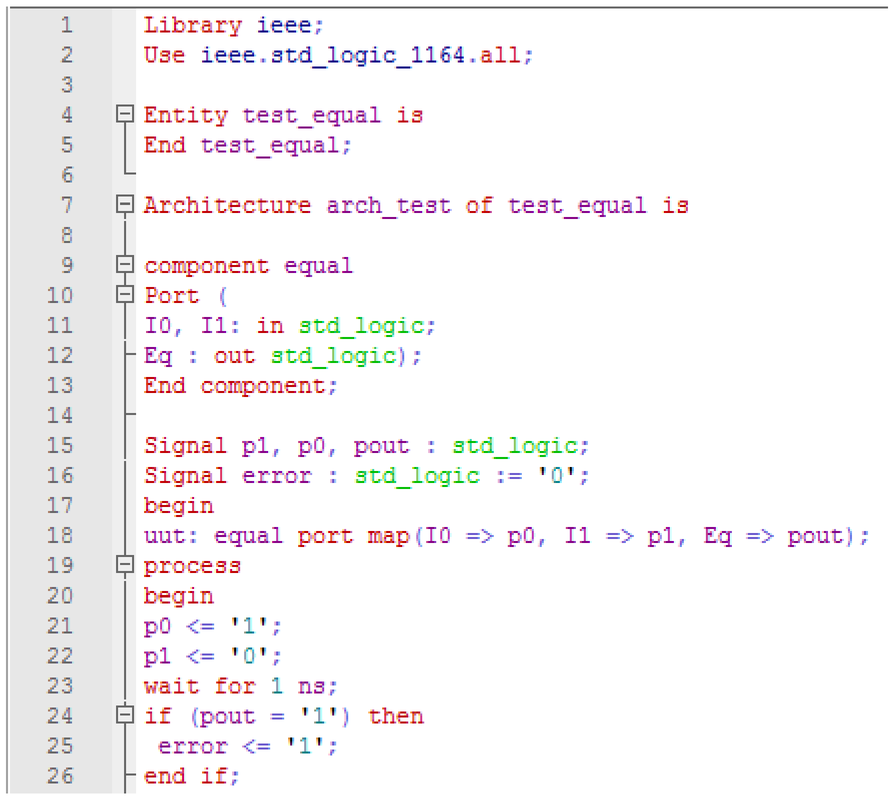


```
signal e0,e1,e2,e3,e4,e5,e6,e7: std_logic;
begin
--instantiates eight one-bit comparators
H1: equal
port map(i0=>a(0), i1=>b(0), eq=>e0);
H2: equal
port map(i0=>a(1), i1=>b(1), eq=>e1);
H3: equal
port map(i0=>a(2), i1=>b(2), eq=>e2);
H4: equal
port map(i0=>a(3), i1=>b(3), eq=>e3);
H5: equal
port map(i0=>a(4), i1=>b(4), eq=>e4);
H6: equal
port map(i0=>a(5), i1=>b(5), eq=>e5);
H7: equal
port map(i0=>a(6), i1=>b(6), eq=>e6);
H8: equal
port map(i0=>a(7), i1=>b(7), eq=>e7);
-- a and b are equal if individual bits are equal.
aeqb <= e0 and e1 and e2 and e3 and e4 and e5 and e6 and e7;
end arch;
```

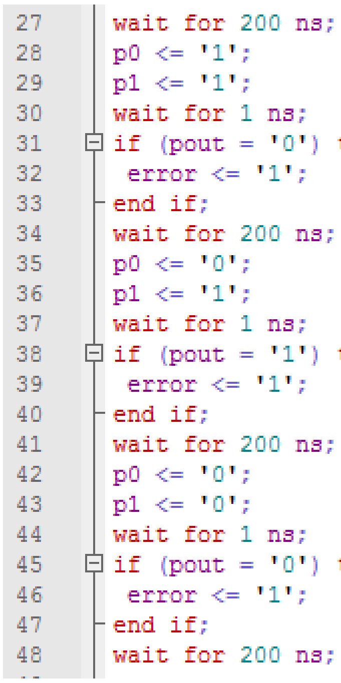
Simulation:

We simulated and tested our circuit designs using test files written in VHDL, and running them through a waveform simulation on ModelSim.

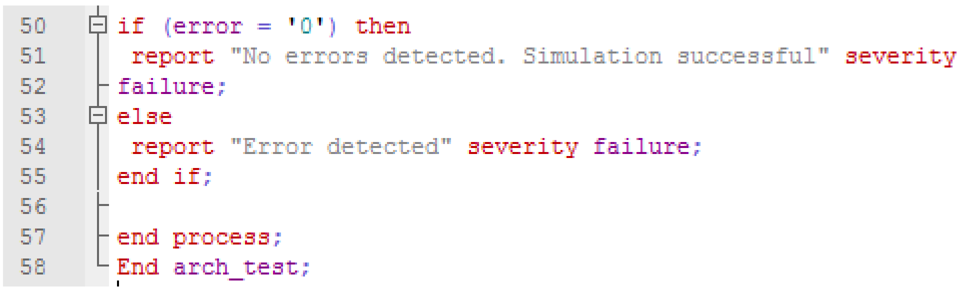
1-Bit Comparator



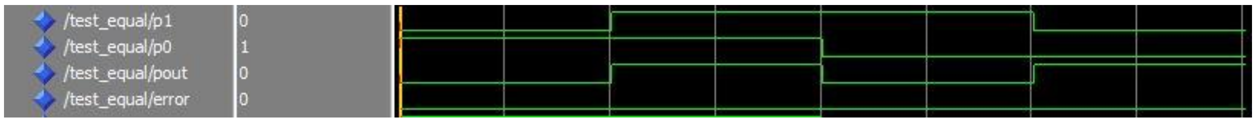
```
Library ieee;
Use ieee.std_logic_1164.all;

Entity test_equal is
End test_equal;

Architecture arch_test of test_equal is

component equal
Port (
I0, I1: in std_logic;
Eq : out std_logic);
End component;

Signal p1, p0, pout : std_logic;
Signal error : std_logic := '0';
begin
uut: equal port map(I0 => p0, I1 => p1, Eq => pout);
process
begin
p0 <= '1';
p1 <= '0';
wait for 1 ns;
if (pout = '1') then
 error <= '1';
end if;
```



```vhdl
wait for 200 ns;
p0 <= '1';
p1 <= '1';
wait for 1 ns;
if (pout = '0') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= '0';
p1 <= '1';
wait for 1 ns;
if (pout = '1') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= '0';
p1 <= '0';
wait for 1 ns;
if (pout = '0') then
 error <= '1';
end if;
wait for 200 ns;
```

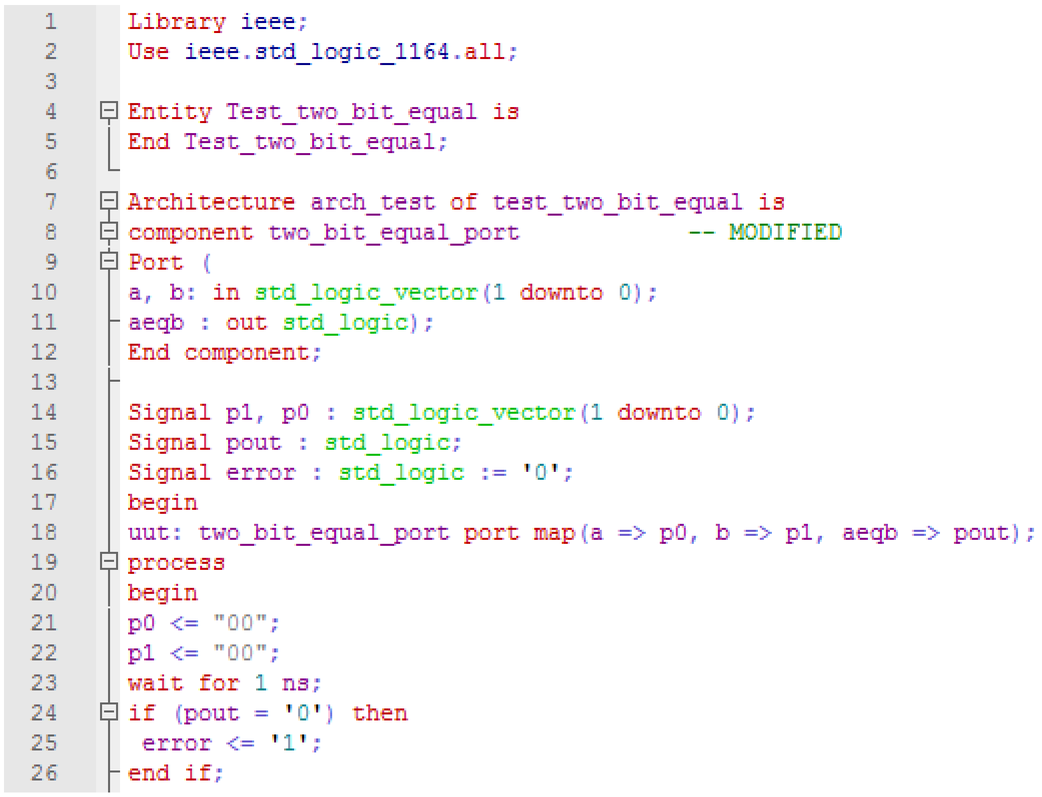


```vhdl
if (error = '0') then
 report "No errors detected. Simulation successful" severity
failure;
else
 report "Error detected" severity failure;
end if;

end process;
End arch_test;
```

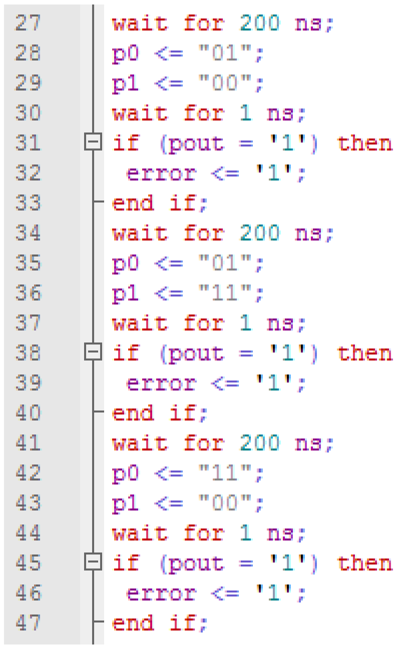


```
/test_equal/p1     0
/test_equal/p0     1
/test_equal/pout   0
/test_equal/error  0
```

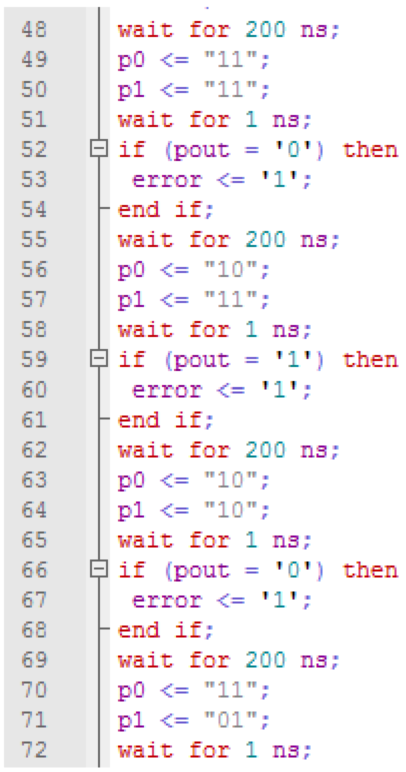
2-Bit Comparator



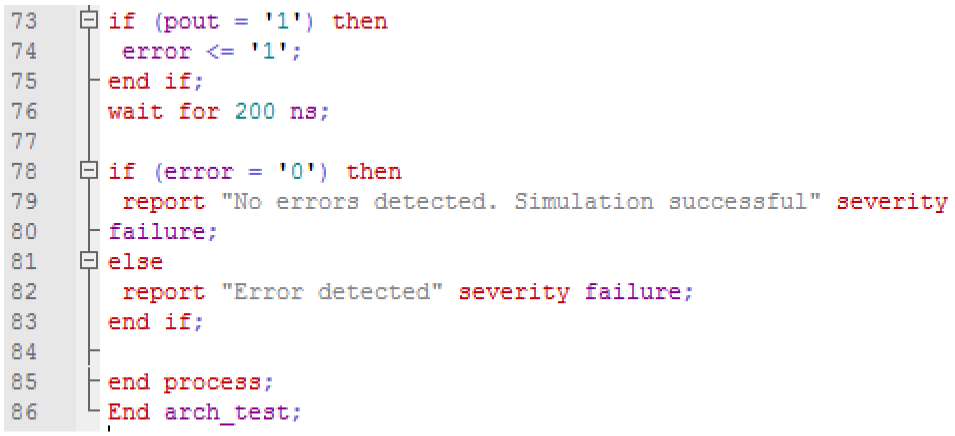
```
Library ieee;
Use ieee.std_logic_1164.all;

Entity Test_two_bit_equal is
End Test_two_bit_equal;

Architecture arch_test of test_two_bit_equal is
component two_bit_equal_port              -- MODIFIED
Port (
a, b: in std_logic_vector(1 downto 0);
aeqb : out std_logic);
End component;

Signal p1, p0 : std_logic_vector(1 downto 0);
Signal pout : std_logic;
Signal error : std_logic := '0';
begin
uut: two_bit_equal_port port map(a => p0, b => p1, aeqb => pout);
process
begin
p0 <= "00";
p1 <= "00";
wait for 1 ns;
if (pout = '0') then
 error <= '1';
end if;
```



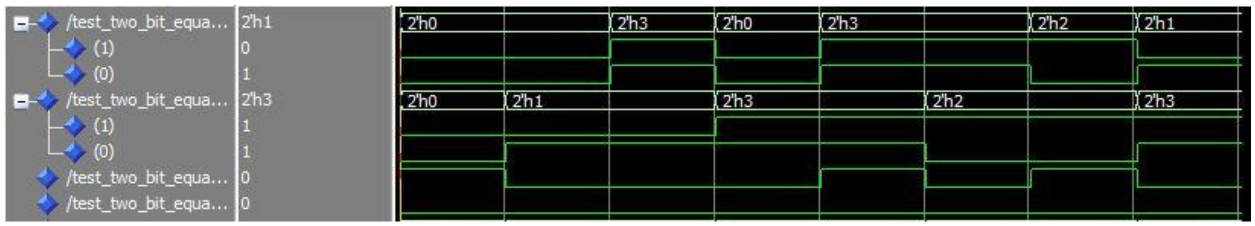
```
wait for 200 ns;
p0 <= "01";
p1 <= "00";
wait for 1 ns;
if (pout = '1') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= "01";
p1 <= "11";
wait for 1 ns;
if (pout = '1') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= "11";
p1 <= "00";
wait for 1 ns;
if (pout = '1') then
 error <= '1';
end if;
```



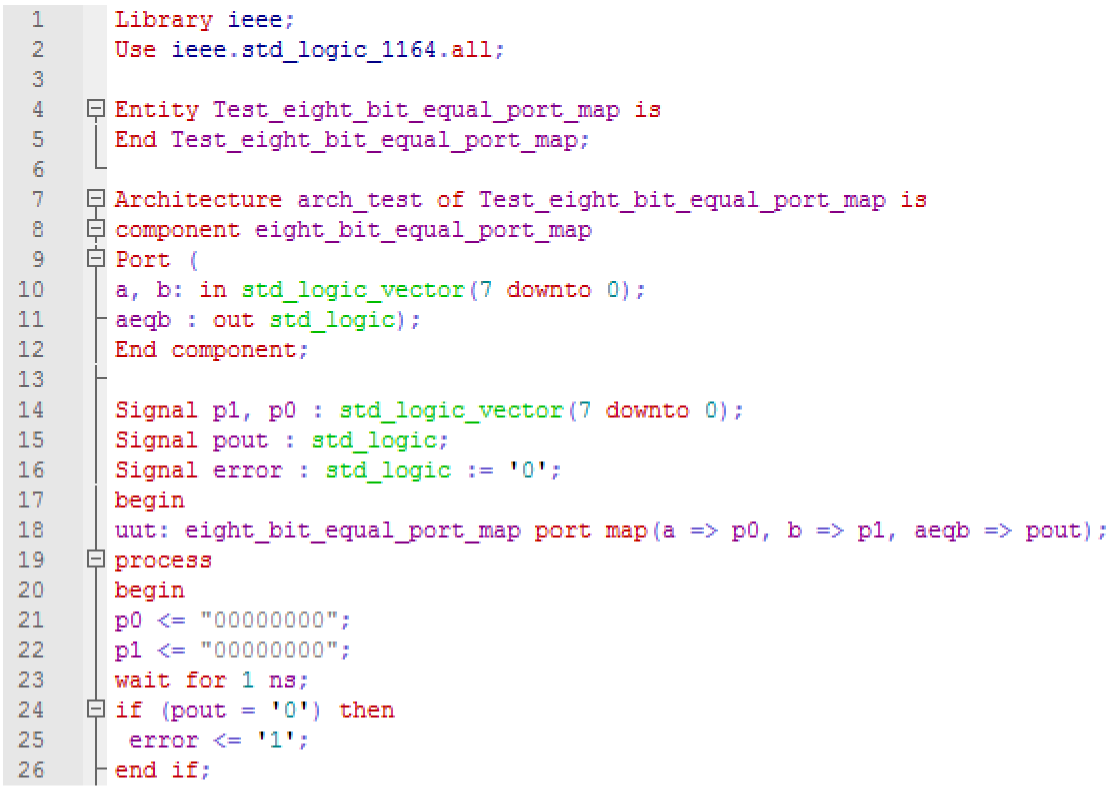
```
wait for 200 ns;
p0 <= "11";
p1 <= "11";
wait for 1 ns;
if (pout = '0') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= "10";
p1 <= "11";
wait for 1 ns;
if (pout = '1') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= "10";
p1 <= "10";
wait for 1 ns;
if (pout = '0') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= "11";
p1 <= "01";
wait for 1 ns;
```



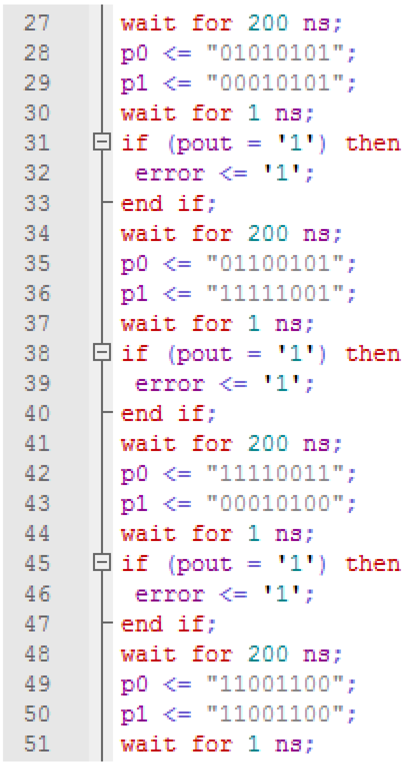
```
if (pout = '1') then
 error <= '1';
end if;
wait for 200 ns;

if (error = '0') then
 report "No errors detected. Simulation successful" severity
failure;
else
 report "Error detected" severity failure;
end if;

end process;
End arch_test;
```



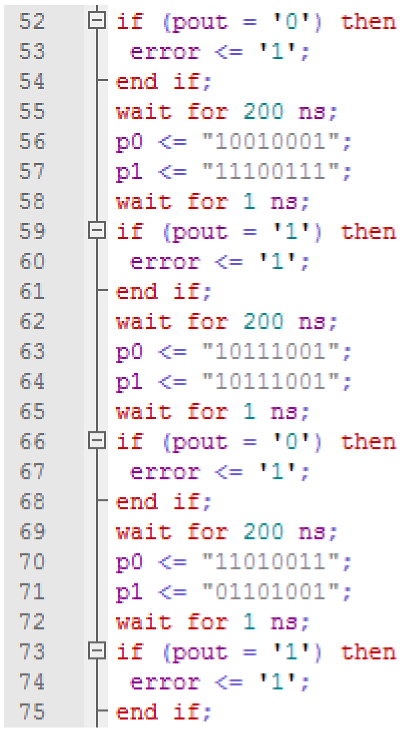
8-Bit Comparator



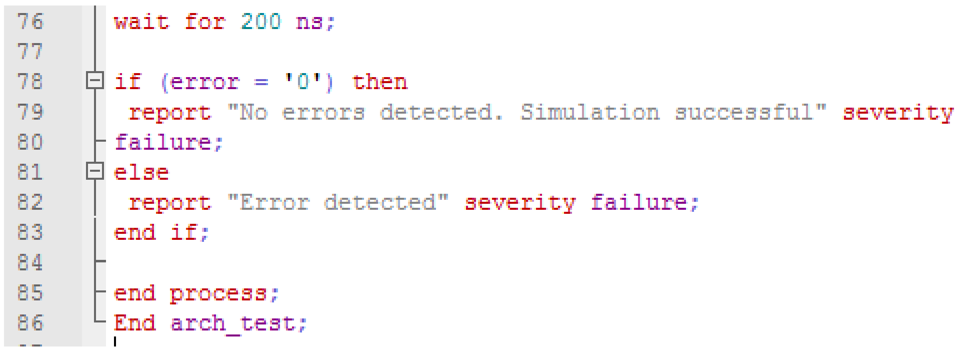
```vhdl
Library ieee;
Use ieee.std_logic_1164.all;

Entity Test_eight_bit_equal_port_map is
End Test_eight_bit_equal_port_map;

Architecture arch_test of Test_eight_bit_equal_port_map is
component eight_bit_equal_port_map
Port (
a, b: in std_logic_vector(7 downto 0);
aeqb : out std_logic);
End component;

Signal p1, p0 : std_logic_vector(7 downto 0);
Signal pout : std_logic;
Signal error : std_logic := '0';
begin
uut: eight_bit_equal_port_map port map(a => p0, b => p1, aeqb => pout);
process
begin
p0 <= "00000000";
p1 <= "00000000";
wait for 1 ns;
if (pout = '0') then
 error <= '1';
end if;
```



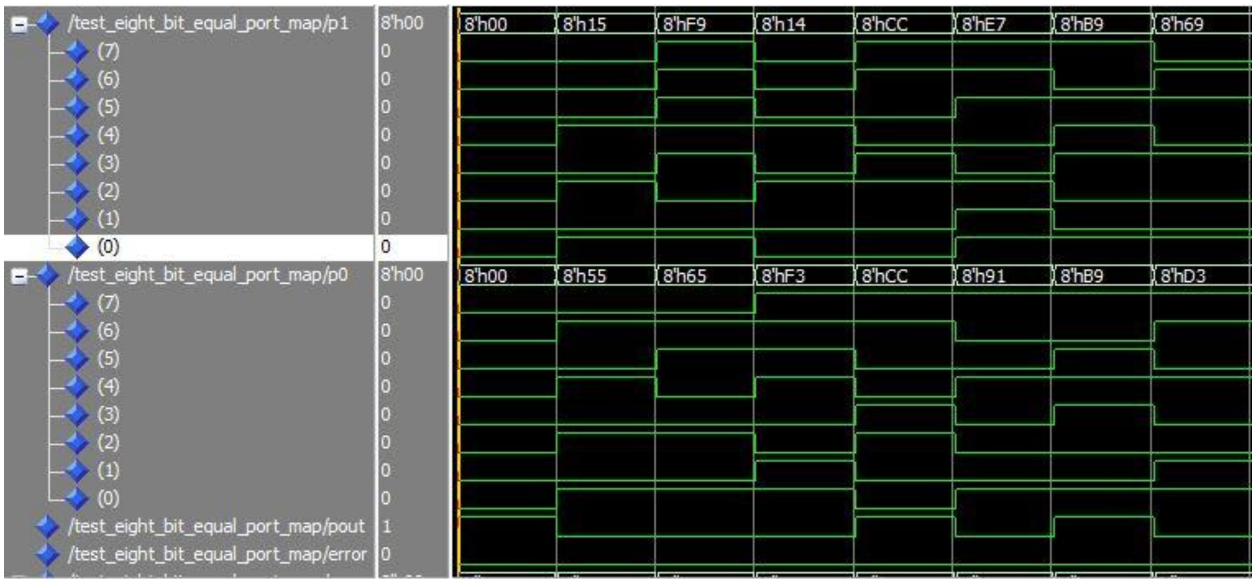
```vhdl
wait for 200 ns;
p0 <= "01010101";
p1 <= "00010101";
wait for 1 ns;
if (pout = '1') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= "01100101";
p1 <= "11111001";
wait for 1 ns;
if (pout = '1') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= "11110011";
p1 <= "00010100";
wait for 1 ns;
if (pout = '1') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= "11001100";
p1 <= "11001100";
wait for 1 ns;
```



```
if (pout = '0') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= "10010001";
p1 <= "11100111";
wait for 1 ns;
if (pout = '1') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= "10111001";
p1 <= "10111001";
wait for 1 ns;
if (pout = '0') then
 error <= '1';
end if;
wait for 200 ns;
p0 <= "11010011";
p1 <= "01101001";
wait for 1 ns;
if (pout = '1') then
 error <= '1';
end if;
```



```
wait for 200 ns;

if (error = '0') then
 report "No errors detected. Simulation successful" severity
failure;
else
 report "Error detected" severity failure;
end if;

end process;
End arch_test;
```



```
/test_eight_bit_equal_port_map/p1     8'h00   8'h00 8'h15 8'hF9 8'h14 8'hCC 8'hE7 8'hB9 8'h69
  (7)  0
  (6)  0
  (5)  0
  (4)  0
  (3)  0
  (2)  0
  (1)  0
  (0)  0
/test_eight_bit_equal_port_map/p0     8'h00   8'h00 8'h55 8'h65 8'hF3 8'hCC 8'h91 8'hB9 8'hD3
  (7)  0
  (6)  0
  (5)  0
  (4)  0
  (3)  0
  (2)  0
  (1)  0
  (0)  0
/test_eight_bit_equal_port_map/pout   1
/test_eight_bit_equal_port_map/error  0
```

Conclusion:

We began by building our 1-bit comparator based off a derived logical expression. Then we used this knowledge to make a 2-bit comparator. However, we learned that we did not need to derive another logical expression in order to make this. Instead, we learned how to optimize our circuit by using port maps in order to combine two 1-bit comparators together. Finally, we used port maps once again to expand our design into an 8-bit comparator. This technique became very useful because it allowed us to connect multiple comparators together so that we can build a unit that can compare binary strings of larger magnitudes. The concept behind port mapping can be used for many other types of circuits as well, for example, large full bit adders and subtractors.