Mustafa Cankan BALCI

22101761

Section 01

8 November 2022

Lab 4: Arithmetic Logic Unit

Purpose

In this lab, Arithmetic Logic Unit (ALU) will aim to design by using VHDL and Basys3 board. Also, the aim of the experiment is creating a hierarchal design in VHDL.

Design Specifications

An arithmetic Logic Unit (ALU) is a combinational circuit that can perform mathematical and logical operations and is an essential building block for most digital devices. Arithmetic Logic Units are designed using a multiplexer and logical gates such as and gate, OR gate, and XOR gate. In this experiment, an 8-bit ALU was designed, with an 8-to-1 multiplexer containing 3 selective inputs (S2, S1, S0) and 8 data inputs. The data inputs of the 8-to-1 multiplexer of ALU is the combinational circuit that are 4-bit adder, 4-bit subtracter, 4-bit left shifter, 4-bit comparator, 4-bit equaler, 4-bit AND gate, 4-bit OR gate and 4-bit XOR gate. Therefore, the output of ALU is a 4-bit binary number. The inputs of combinational circuits in ALU expect a 4-bit left shifter are two 4-bit data inputs (A, B). The input of the 4-bit shifter is a 4-bit data input(A). The truth table of the combinational circuits in the ALU is in the appendix.

Moreover, the selective inputs (S2, S1, S0) are assigned to R2, T1 and U1 switches in orderly at BASYS3. The of data inputs 4-bit A input (A3, A2, A1, A0) are defined W17, W16, V16,

V17 switches in given order. The of data inputs 4-bit B input (B3, B2, B1, B0) are W13, W14, V15, W15 switches in given order. The 4-bit output (O3, O2, O1, O0) of ALU is V19, U19, E19 and V16 LEDS on BASYS3 respectively.

Methodology

Hierarchal design method and bus notation are used to design the 8-bit Arithmetic Logic Unit. Initially, a 4-bit adder is designed with using full adders and half adders. Half adders are designed firstly for creating full adders with one XOR gate and one AND gate. Afterward, full adders designed with 2 half adders and one OR gate. 4-bit adder contains the combination 4 full adders, which are used as a low-level entry in the design. Moreover, 4-bit subtractor is designed by applying the same process in 4-bit adder. The one of the differences between the 4-bit adder and the 4-bit subtractor is one of the inputs of subtractor connected with NOT gate before going in the subtractor module. Also, the carry in of 4-bit adder is 0, but the carry in of 4-bit subtractor is 1.

Furthermore, the 4-bit left shifter is designed with using the low-level entry which is a low shift in the code. The low shift contains x1, x2 and shift inputs and fs output, which consists of AND, NOT and OR gates. In the top-level entry, a shift module is built on 4 low shift modules. Also, the shift inputs of low shifts are assigned to 1 in the shift module code.

Additionally, the 4-bit comparator is constructed by 2-bit comparator and 2-bit equality checking. The 2-bit comparator is designed by using AND, NOT gates. The 2-bit equality checker used XNOR ,AND gates. The first 2-bit comparator is checking the first two most significant nibbles. If the first 2 most significant nibbles are equal, 2-bit least significant nibbles are

compared. Also, 4-bit equaler is designed with XNOR gates. Moreover, the bitwise operations such as AND, OR and XOR are designed in different files. Finally, the all-combinational circuits are used as a data input of the 8-to-1 multiplexer, so all modules used as a low entry of design in ALU.

Results

The RTL schematics of ALU are the same as the expected result. An 8-to-1 multiplexer is designed by using a 2-to-1 multiplexer in RTL schematics. The selective inputs of the 8-to-1 multiplexer are s2, s1, and s0. The data inputs of the 8-to-1 multiplexer are adder, subtracter, shifter, equaler, and_gate, or_gate, and xor_gate. The 4-bit A inputs of the modules are a3, a2, a1, and a0. The 4-bit B input of modules are b3, b2, b1, and b0. The interior designs of the photos of the combinational circuit are in the appendix.

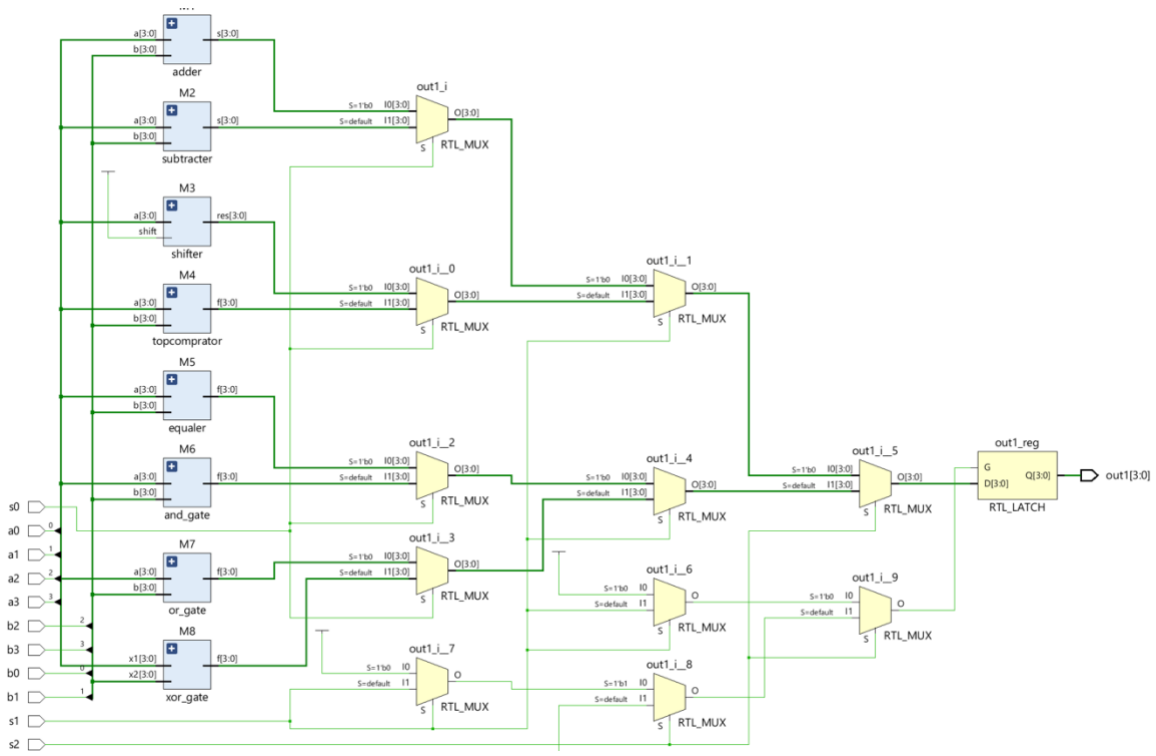

Figure 1: RTL Design of ALU

The testbench of ALU is also give the give same expected result. The graph 1 is one of the example timing diagrams in the simulation. For instance, when s2, s1 and s2 equals zero, 4-bit A and B equals to “0000”, the final result is “0000”.

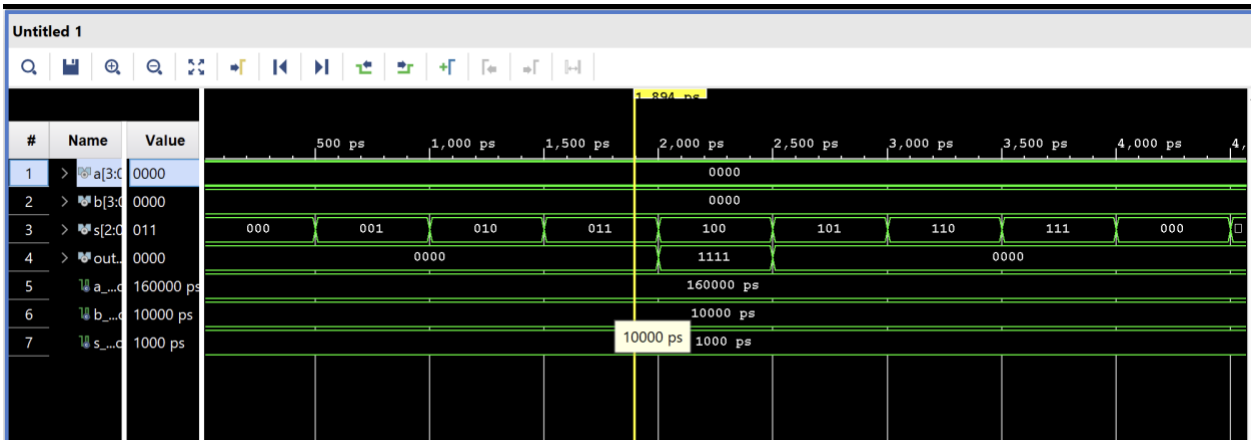

Figure 2: Testbench Graph

The application of the ALU on BASYS 3 is the same as expected. For instance, when the comparator state, A is “0100” and B is”0011”, the output leds turned (Figure 3). There are also example photos of different inputs for a combinational circuit’s BASYS3 implementation.

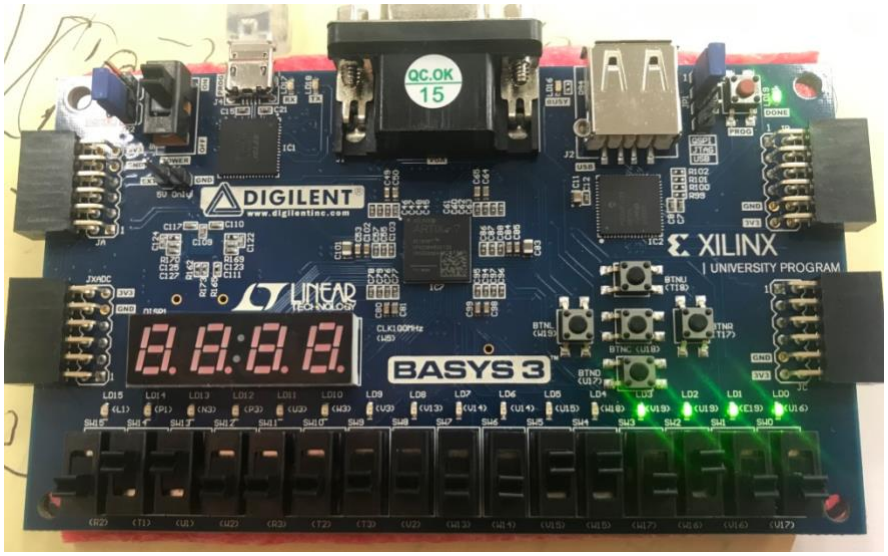

Figure 3: T1, U1, W16, W15 and V15 pulled up, the U16, E19, U19 and V19 leds turned on

Conclusion

The investigation focuses on designing an Arithmetic Logic Unit. The designing of hierarchal design modules is necessary during the design of 8-bit ALU via VHDL. The investigation aimed to obtain a simulation of the combination of combinational circuits and compare the result of the simulation and implantation of the BASYS3 board. The most significant problems faced were the connection between low-level entry outputs and the top-level entry outputs and testbench simulation. Initially, the testbench simulation output shows only subtraction. The problem was solved by adding selector input pins to the process's sensitivity list in the 8-to-1 multiplexer, which is top-level ALU code.

Appendix

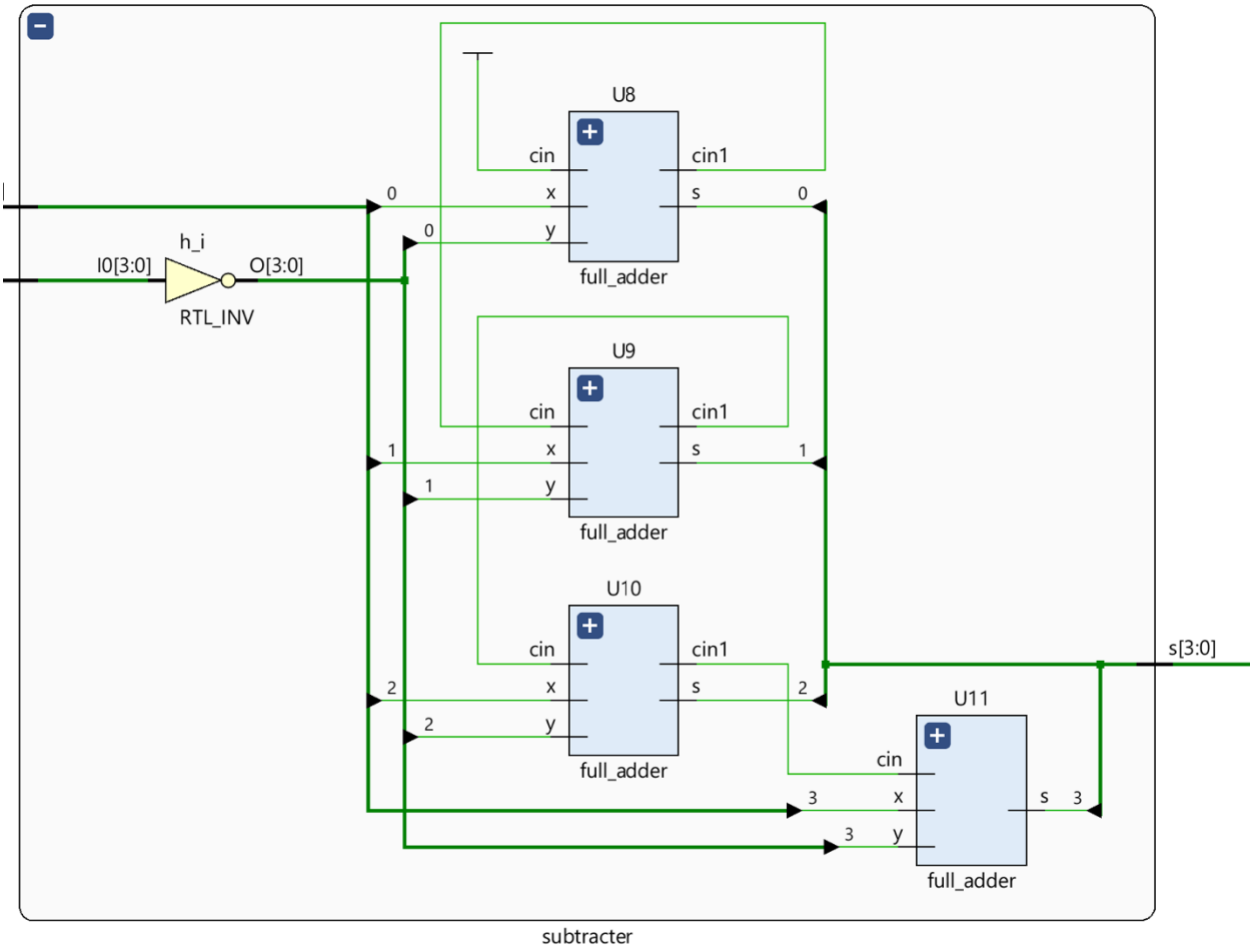

Figure 4: RTL Design of 4-bit Subtractor

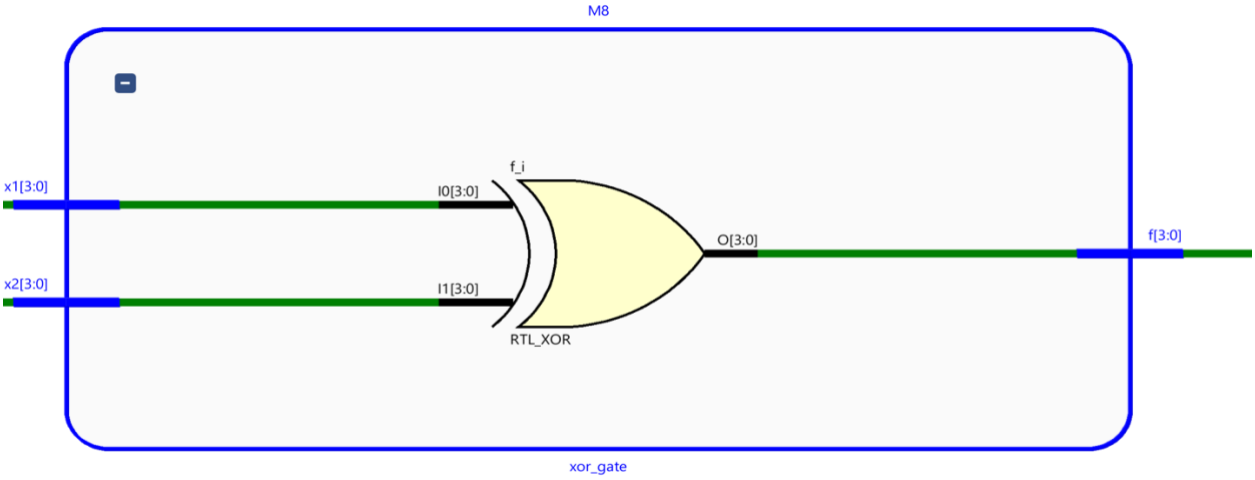

Figure 5: RTL Design of 4-bit XOR gate

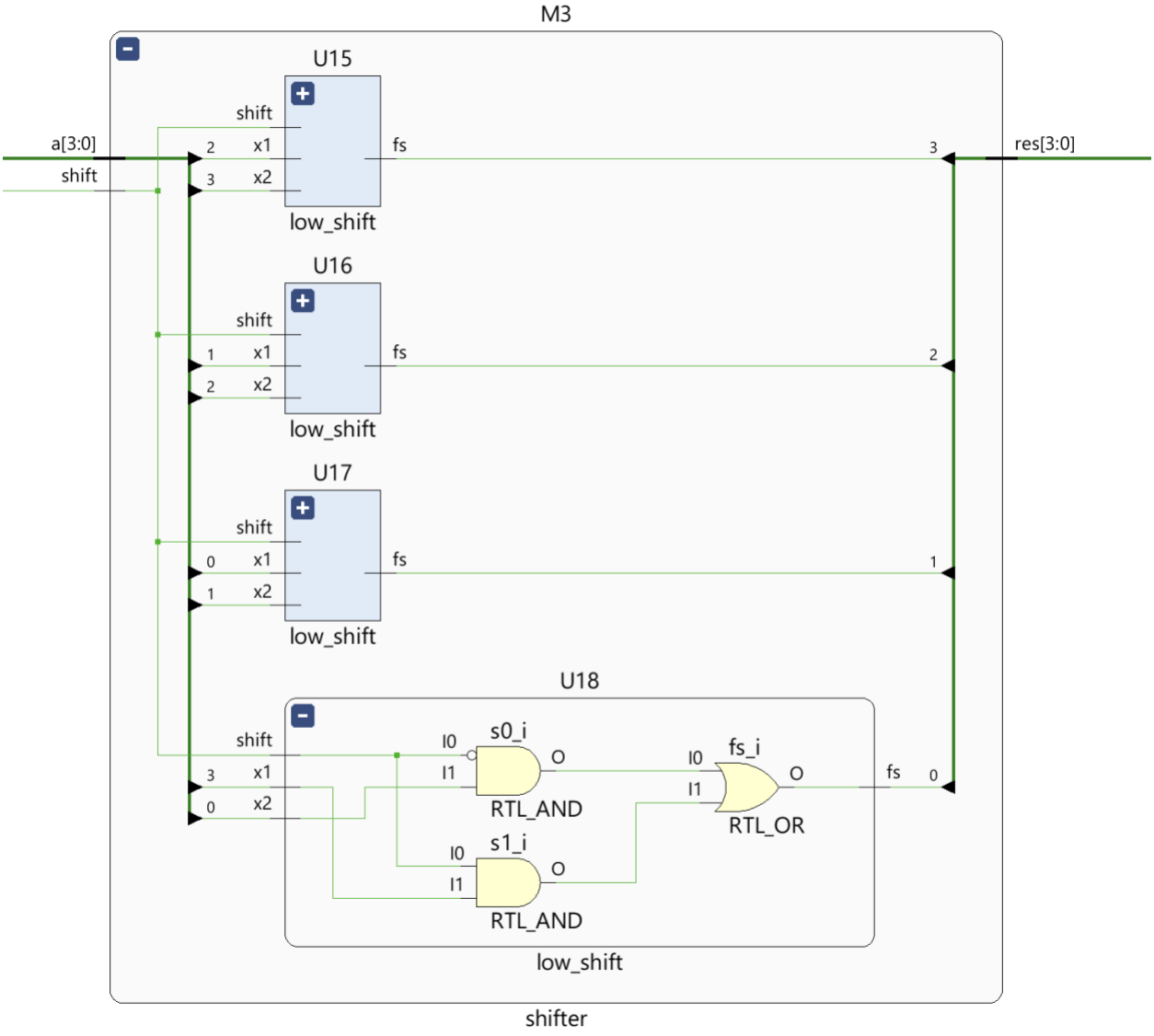

Figure 6: RTL Design of 4-bit left shifter

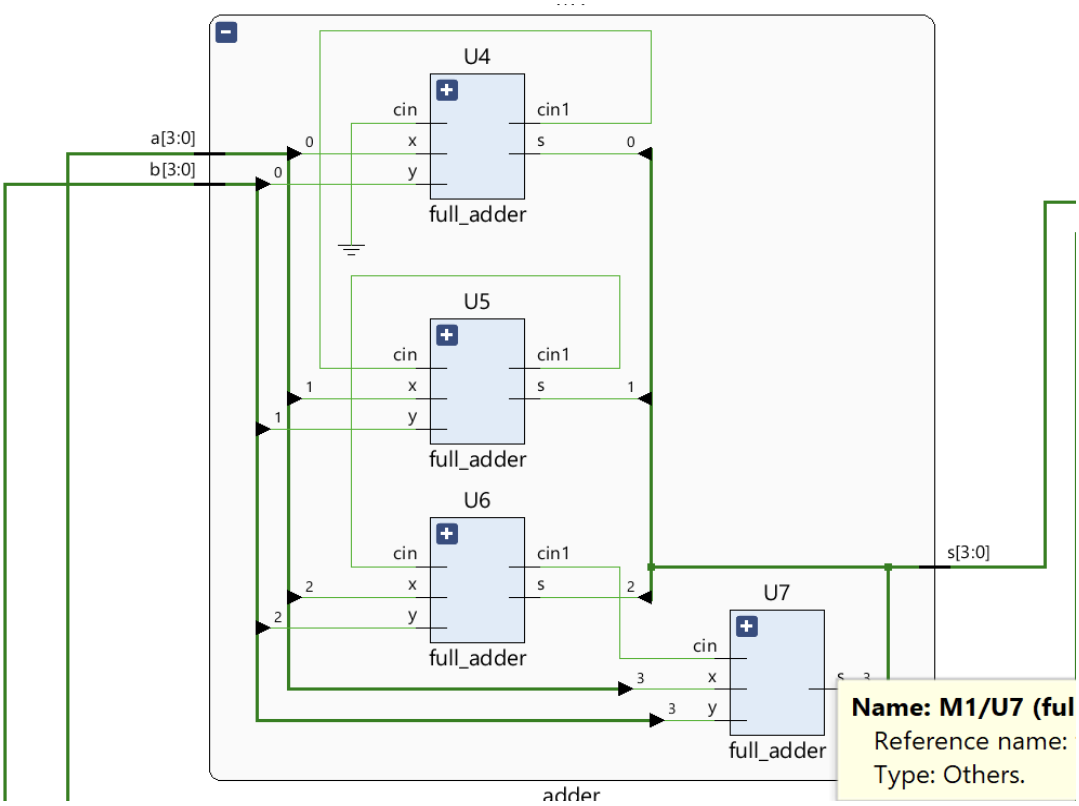

Figure 7: RTL Design of 4-bit adder

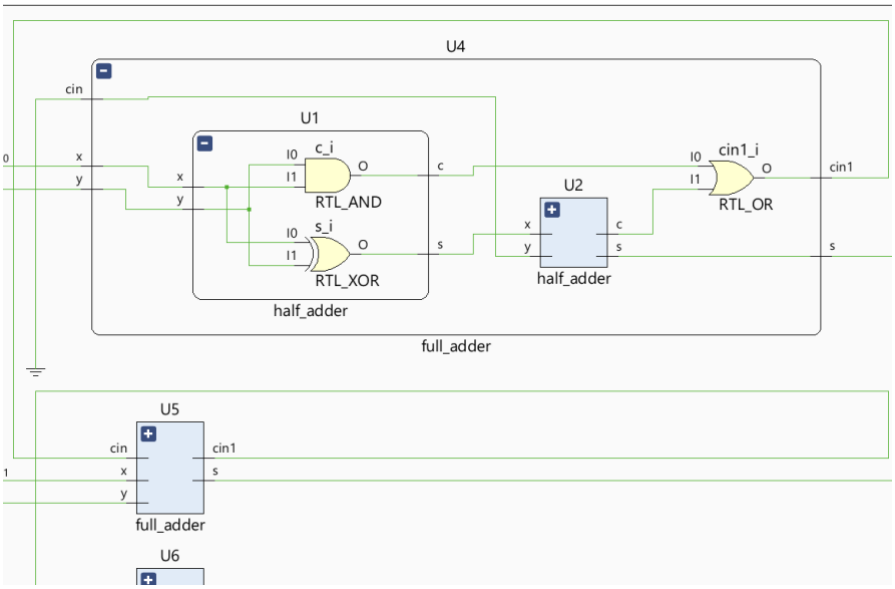

Figure 8: RTL Design of Full adder and Half Adder

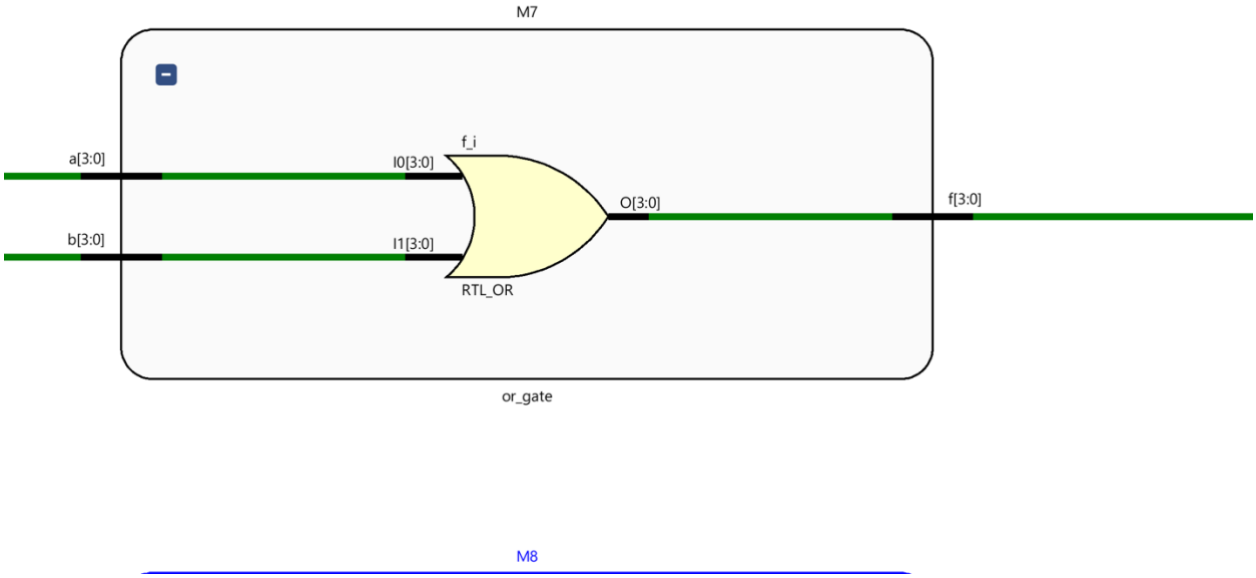

Figure 9: RTL Design of 4-bit XOR gate

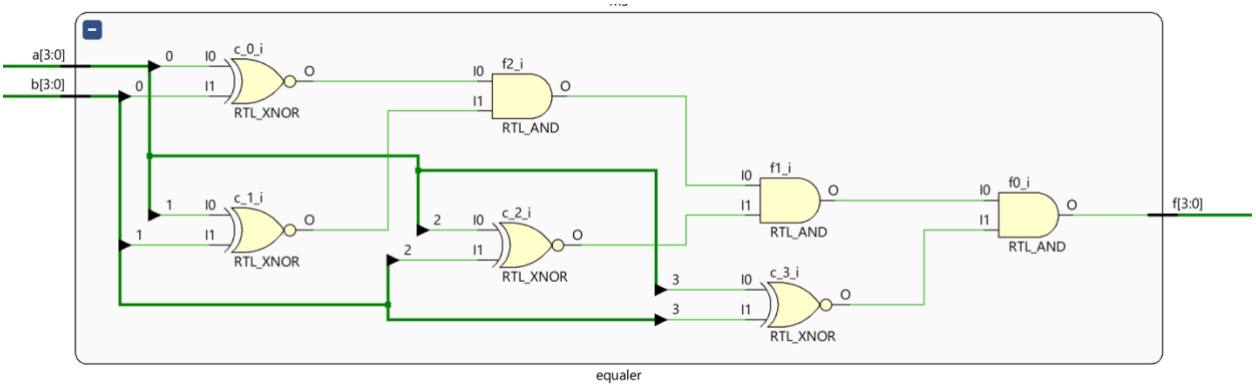

Figure 10: RTL Design of 4-bit Equality Checker

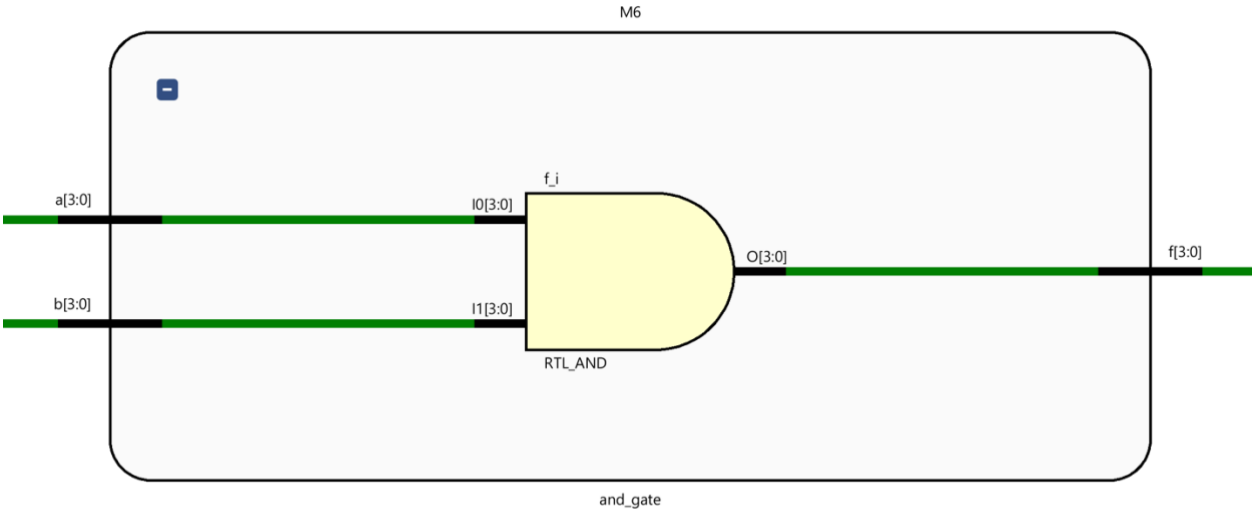

Figure 11: RTL Design of 4-bit And gate

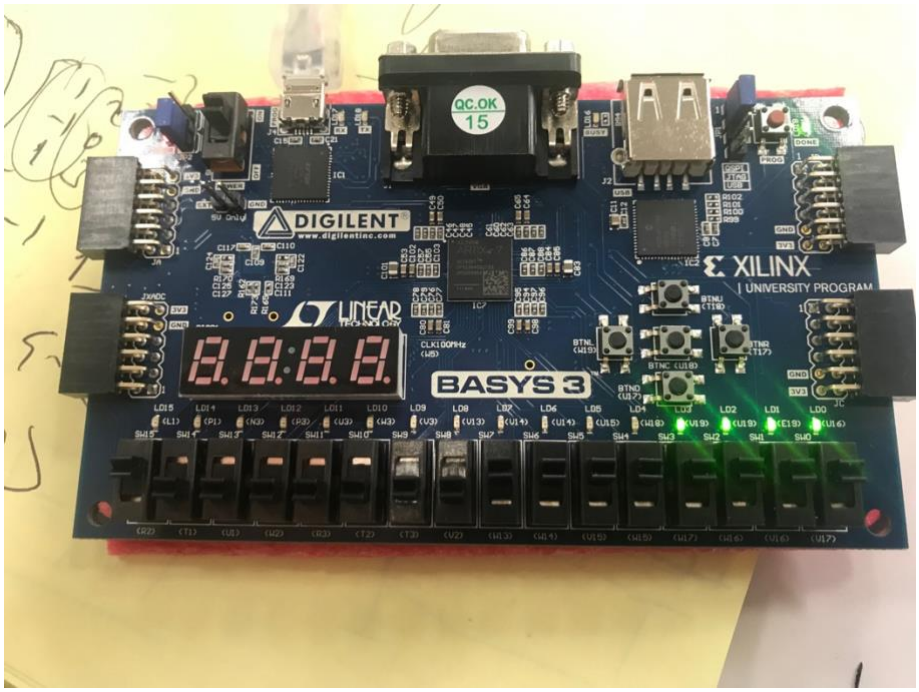

Figure 12: R2, W14,W13, W15, V15,W17,W16,V16 and V17 pulled up, the U16, E19, U19 and V19 leds turned on

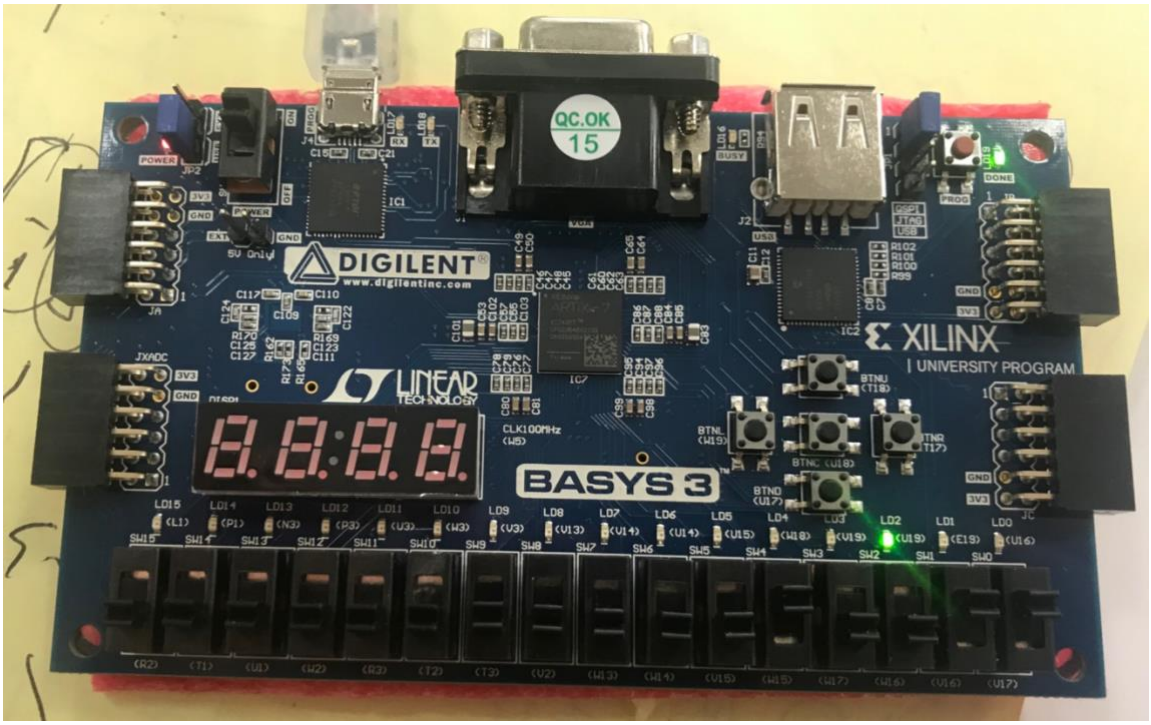

Figure 13: V17, V16 and W15 pulled up the U19 led turned on

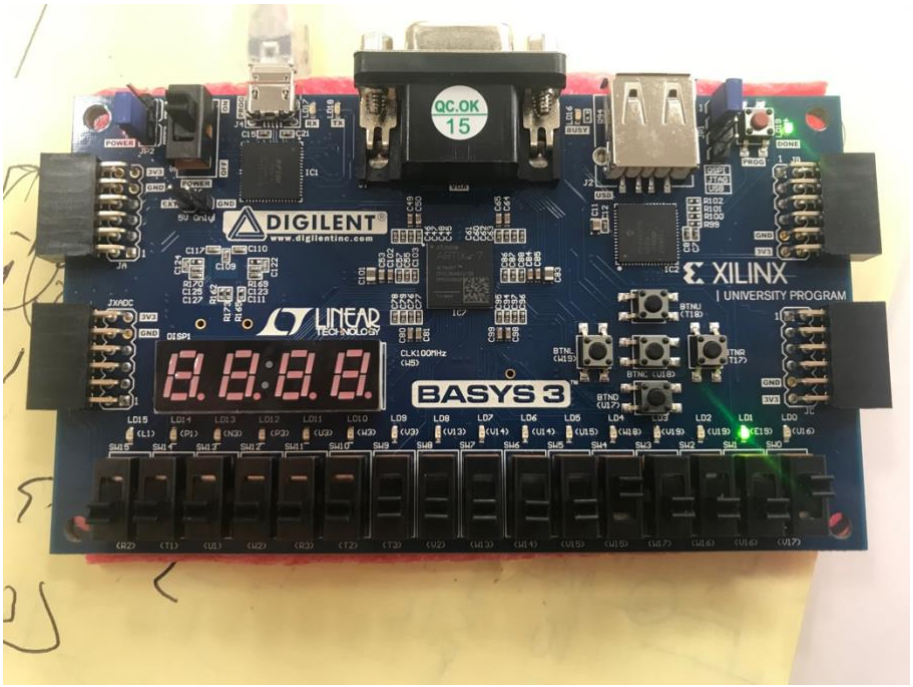

Figure 14: V17, W15 a pulled up E19 led turned on

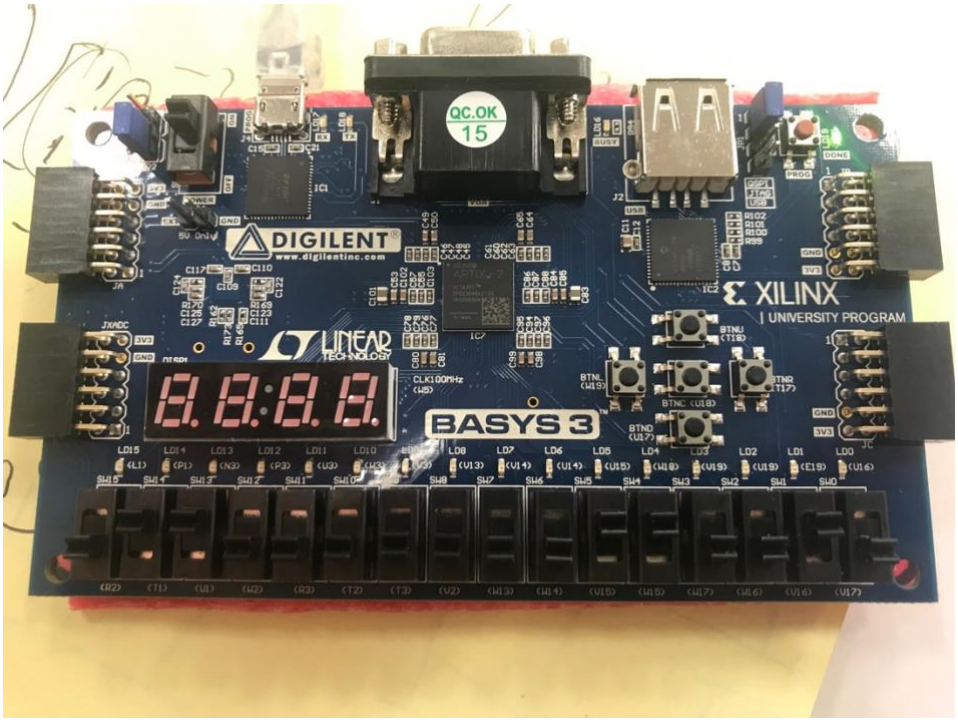

Figure 15: T1, V1, V15, W15 and V16 pulled up, none of the leds turned on

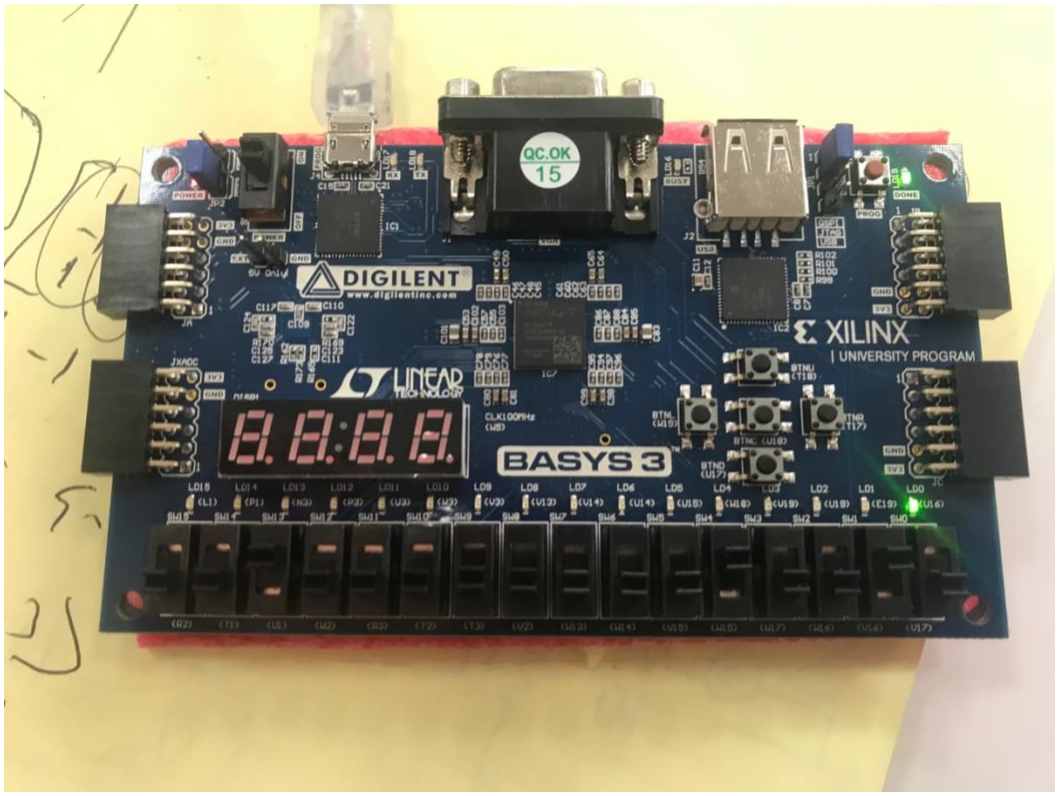

Figure 16: V1, W15, V16 and V17 pulled up, the U16 led turned on

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_top is
   Port ( s0, s1, s2 : in STD_LOGIC;
        b0,b1,b2,b3 : in STD_LOGIC;
        a0, a1,a2,a3 : in STD_LOGIC;
        out1: out STD_LOGIC_vector(3 downto 0)

        );
end mux_top;

architecture Behavioral of mux_top is

-- Input Parameters
signal s : std_logic_vector(2 downto 0);
signal a : std_logic_vector(3 downto 0);
signal b : std_logic_vector(3 downto 0);

-- Output Parameters
Signal out_adder : std_logic_vector(3 downto 0);
Signal out_subtracter: std_logic_vector(3 downto 0);
Signal out_shifter: std_logic_vector(3 downto 0);
Signal out_topcomprator : std_logic_vector(3 downto 0);
Signal out_equaler : std_logic_vector(3 downto 0);
Signal out_and_gate: std_logic_vector(3 downto 0);
Signal out_or_gate: std_logic_vector(3 downto 0);
Signal out_xor_gate: std_logic_vector(3 downto 0);

Component adder is
   Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
        b : in STD_LOGIC_VECTOR (3 downto 0);
        s : out STD_LOGIC_VECTOR (3 downto 0)
        );
End Component;

Component subtracter is
   Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
        b : in STD_LOGIC_VECTOR (3 downto 0);
        s : out STD_LOGIC_VECTOR (3 downto 0)
        );
 End Component;

Component shifter is
```

```
   Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
          shift : in STD_LOGIC;
          res : out STD_LOGIC_VECTOR (3 downto 0));
End Component;

Component topcomprator 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 Component;

Component equaler 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 Component;

Component and_gate 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 Component;

Component or_gate 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 Component;

Component xor_gate is
   Port ( x1 : in STD_LOGIC_VECTOR (3 downto 0);
          x2 : in STD_LOGIC_VECTOR (3 downto 0);
          f : out STD_LOGIC_VECTOR (3 downto 0));
End Component;

begin
a(3)<= a3;
a(2)<= a2;
a(1)<= a1;
a(0)<= a0;

b(3)<= b3;
```

```
b(2)<= b2;
b(1)<= b1;
b(0)<= b0;

s(2)<= s2;
s(1)<= s1;
s(0)<= s0;

M1: adder Port Map(a, b, out_adder);
M2: subtracter Port Map(a,b, out_subtracter);
M3: shifter Port Map(a, '1', out_shifter);
M4: topcomprator Port Map(a, b, out_topcomprator);
M5: equaler Port Map(a,b,out_equaler);
M6: and_gate Port Map(a,b,out_and_gate);
M7: or_gate Port Map(a,b,out_or_gate);
M8: xor_gate Port Map(a,b,out_xor_gate);

Process(s,out_adder, out_subtracter, out_shifter, out_topcomprator, out_equaler,
out_and_gate, out_or_gate, out_xor_gate)
begin

If s(2) = '0' Then
   If s(1) = '0' Then
      If s(0) = '0' Then
         out1 <= out_adder ;
      Else
         out1 <= out_subtracter;
      End If;
   Elsif s(1) = '1' Then
      If s(0) = '0' Then
         out1 <= out_shifter;
      Else
         out1 <= out_topcomprator;
      End If;
    End If;
Elsif s(2) = '1' Then
   If s(1) = '0' Then
      If s(0) = '0' Then
         out1 <= out_equaler ;

      Else
         out1 <= out_and_gate ;
      End If;
   Elsif s(1) = '1' Then
```

```
      If s(0) = '0' Then
         out1 <= out_or_gate;
      Else
         out1 <= out_xor_gate;
      End If;
    End If;
End If;
end Process;
end Behavioral;
```

Code 1: Design Code of ALU

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity half_adder is
   Port ( x : in STD_LOGIC;
          y : in STD_LOGIC;
          s : out STD_LOGIC;
          c : out STD_LOGIC);
end half_adder;

architecture Behavioral of half_adder is
begin
s <= x XOR y;
c <= y and x;
end Behavioral;
```

Code 2: Design Code of Half Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_adder is
   Port ( x : in STD_LOGIC;
          y : in STD_LOGIC;
          cin : in STD_LOGIC;
          s : out STD_LOGIC;
          cin1 : out STD_LOGIC);
end full_adder;

architecture Behavioral of full_adder is
Signal s0, c0, c1 : std_logic;
Component half_adder is
   PORT(
      x, y: in std_logic;
      s, c: out std_logic
   );
```

```
 End Component;
begin
U1: half_adder PORT MAP (x,y,s0,c0);
U2: half_adder PORT MAP (s0,cin, s, c1);
cin1 <= c0 or c1;
end Behavioral;
```

Code 3: Design Code of Full Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity adder is
   Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
        b : in STD_LOGIC_VECTOR (3 downto 0);
        s : out STD_LOGIC_VECTOR (3 downto 0)
        );
end adder;

architecture Behavioral of adder is
Signal c1, c2, c3, c4: std_logic;
Component full_adder is
   Port (
      x, y, cin : IN std_logic;
      s, cin1: out std_logic
   );
   End Component;
begin
U4: full_adder Port Map(a(0), b(0), '0', s(0), c1);
U5: full_adder Port Map(a(1), b(1), c1, s(1), c2);
U6: full_adder Port Map(a(2), b(2), c2, s(2), c3);
U7: full_adder Port Map(a(3), b(3), c3 , s(3), c4);
end Behavioral;
```

Code 4: Design Code of 4-bit Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity subtracter is
   Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
        b : in STD_LOGIC_VECTOR (3 downto 0);
        s : out STD_LOGIC_VECTOR (3 downto 0)

       );
end subtracter;

architecture Behavioral of subtracter is
```

```
Signal c0,c1, c2, c3, c4: std_logic;
Signal h : std_logic_vector(3 downto 0);
Component full_adder is
   Port (
      x, y, cin : IN std_logic;
      s, cin1: out std_logic
   );
   End Component;
begin
c0 <= '1';
h <= not b;
U8: full_adder Port Map(a(0), h(0), c0, s(0), c1);
U9: full_adder Port Map(a(1), h(1), c1, s(1), c2);
U10: full_adder Port Map(a(2), h(2), c2, s(2), c3);
U11: full_adder Port Map(a(3), h(3), c3 , s(3), c4);
end Behavioral;
```

Code 5: Design Code of 4-bit Subtractor

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity shifter is
   Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
        shift : in STD_LOGIC;
        res : out STD_LOGIC_VECTOR (3 downto 0));
end shifter;
architecture Behavioral of shifter is
Component low_shift is
Port(
   x2 : in STD_LOGIC;
   shift : in STD_LOGIC;
   x1 : in STD_LOGIC;
   fs : out STD_LOGIC
   );
End Component;
begin
U15: low_shift Port Map(a(3), shift, a(2), res(3));
U16: low_shift Port Map(a(2), shift, a(1), res(2));
U17: low_shift Port Map(a(1), shift, a(0), res(1));
U18: low_shift Port Map(a(0), shift, a(3), res(0));
end Behavioral;
```

Code 6: Design Code of 4-bit Shifter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity low_shift is
   Port ( x2 : in STD_LOGIC;
          shift : in STD_LOGIC;
          x1 : in STD_LOGIC;
          fs : out STD_LOGIC);
end low_shift;

architecture Behavioral of low_shift is
Signal s0 , s1 : std_logic;
begin
s0 <= (not shift) and x2;
s1 <= shift and x1;
fs <= s0 or s1;
end Behavioral;
```

Code 7: Design Code of Low_Shifter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity comprator is
   Port ( a : in STD_LOGIC_VECTOR (1 downto 0);
          b : in STD_LOGIC_VECTOR (1 downto 0);
          f : out STD_LOGIC);
end comprator;

architecture Behavioral of comprator is
begin
f <= (a(1) and (not b(1))) or ((a(1) xnor b(1)) and (a(0) and (not b(0))));
end Behavioral;
```

Code 8: Design Code of 2-bit Comparator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity same is
   Port ( a : in STD_LOGIC_VECTOR (1 downto 0);
          b : in STD_LOGIC_VECTOR (1 downto 0);
          s : out STD_LOGIC);
end same;

architecture Behavioral of same is
Signal c1 , c2 : std_logic;
begin
c1 <= a(0) xnor b(0);
c2 <= a(1) xnor b(1);
```

```
s <= c1 and c2;
end Behavioral;
```

Code 9: Design Code of 2-bit Equality Checker

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity topcomprator 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 topcomprator;

architecture Behavioral of topcomprator is
Signal s0, s1, s2 : std_logic;
Component comprator is
   Port(
      a, b : in std_logic_vector(1 downto 0);
      f: out std_logic
   );
   End Component;

Component same is
   Port(
      a, b : in std_logic_vector(1 downto 0);
      s: out std_logic
   );
   End Component;

begin

U12: comprator Port Map(a(1 downto 0), b(1 downto 0), s0);
U13: same Port Map(a(3 downto 2), b(3 downto 2), s1);
U14: comprator Port Map (a(3 downto 2), b(3 downto 2), s2);
f(0) <= (s0 and s1) or s2;
f(1) <= (s0 and s1) or s2;
f(2) <= (s0 and s1) or s2;
f(3) <= (s0 and s1) or s2;
end Behavioral;
```

Code 10: Design Code of 4-bit Comparator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity equaler 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 equaler;

architecture Behavioral of equaler is

Signal c: std_logic_vector(3 downto 0);

begin
c(0) <= a(0) xnor b(0);
c(1) <= a(1) xnor b(1);
c(2) <= a(2) xnor b(2);
c(3) <= a(3) xnor b(3);

f(0) <= c(0) and c(1) and c(2) and c(3);
f(1) <= c(0) and c(1) and c(2) and c(3);
f(2) <= c(0) and c(1) and c(2) and c(3);
f(3) <= c(0) and c(1) and c(2) and c(3);
end Behavioral;
```

Code 11: Design Code of 4-bit Equality Checker

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and_gate 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 and_gate;

architecture Behavioral of and_gate is
begin
f <= (a and b);
end Behavioral;
```

Code 12: Design Code of 4-bit And Gate

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity or_gate 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 or_gate;
```

```
architecture Behavioral of or_gate is

begin
f <= a or b;
end Behavioral;
```

Code 13: Design Code of 4-bit Or Gate

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity xor_gate is
   Port ( x1 : in STD_LOGIC_VECTOR (3 downto 0);
          x2 : in STD_LOGIC_VECTOR (3 downto 0);
          f : out STD_LOGIC_VECTOR (3 downto 0));
end xor_gate;

architecture Behavioral of xor_gate is
begin
f <= x1 xor x2;
end Behavioral;
```

Code 14: Design Code of 4-bit Xor Gate

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity testbench is

end testbench;

architecture Behavioral of testbench is

Component mux_top
   Port(
     s0, s1, s2 : in STD_LOGIC;
     b0,b1,b2,b3 : in STD_LOGIC;
     a0, a1,a2,a3 : in STD_LOGIC;
     out1: out STD_LOGIC_vector(3 downto 0)
   );
End Component;



 -- Inputs
 Signal a: std_logic_vector(3 downto 0) := "0000" ;
```

```
 Signal b: std_logic_vector(3 downto 0) := "0000" ;
 Signal s: std_logic_vector(2 downto 0) := "000" ;

 -- Outputs
 Signal out2: std_logic_vector(3 downto 0);



 -- Constant Times of Inputs
 constant a_period : time := 160ns;
 constant b_period : time := 10ns;
 constant s_period : time := 1ns;

begin
uut: mux_top Port Map(
   s0 => s(0),
   s1 => s(1),
   s2 => s(2),
   b0 => b(0),
   b1 => b(1),
   b2 => b(2),
   b3 => b(3),
   a0 => a(0),
   a1 => a(1),
   a2 => a(2),
   a3 => a(3),
   out1 => out2
);

a_process: process
begin
a <= "0000";
wait for a_period/2;
a <= "0001";
wait for a_period/2;
a <= "0010";
wait for a_period/2;
a <= "0011";
wait for a_period/2;
a <= "0100";
wait for a_period/2;
a <= "0101";
wait for a_period/2;
a <= "0110";
```

```
wait for a_period/2;
a <= "0111";
wait for a_period/2;
a <= "1000";
wait for a_period/2;
a <= "1001";
wait for a_period/2;
a <= "1010";
wait for a_period/2;
a <= "1011";
wait for a_period/2;
a <= "1100";
wait for a_period/2;
a <= "1101";
wait for a_period/2;
a <= "1110";
wait for a_period/2;
a <= "1111";
wait for a_period/2;
end process;

b_process: process
begin
b <= "0000";
wait for b_period/2;
b <= "0001";
wait for b_period/2;
b <= "0010";
wait for b_period/2;
b <= "0011";
wait for b_period/2;
b <= "0100";
wait for b_period/2;
b <= "0101";
wait for b_period/2;
b <= "0110";
wait for b_period/2;
b <= "0111";
wait for b_period/2;
b <= "1000";
wait for b_period/2;
b <= "1001";
wait for b_period/2;
b <= "1010";
```

```
wait for b_period/2;
b <= "1011";
wait for b_period/2;
b <= "1100";
wait for b_period/2;
b <= "1101";
wait for b_period/2;
b <= "1110";
wait for b_period/2;
b <= "1111";
wait for b_period/2;
end process;

s_process: process
begin
s <= "000";
wait for s_period/2;
s <= "001";
wait for s_period/2;
s <= "010";
wait for s_period/2;
s <= "011";
wait for s_period/2;
s <= "100";
wait for s_period/2;
s <= "101";
wait for s_period/2;
s <= "110";
wait for s_period/2;
s <= "111";
wait for s_period/2;
end process;

 -- Stimulus process
 stim_proc: process
 begin
   -- hold reset state for 100 ms
   wait for 100 ms;
   wait for a_period*10;
   -- insert stimulus here
  wait;
  end process;
end Behavioral;
```

Code 15: Testbench Code of ALU