Advanced Digital Logic
ENGR – UH 2310

# Lab 1

Group 1
Spring, 2025

**Submitted by:**

| **Name** | **Net ID** |
|---|---|
| Damiane Kapanadze | dk4770 |
| Seoyoon Jung | sj4260 |
| Sipan Hovsepian | sh7437 |

# Contents

## 1 Task 1: 4-input NAND gate

### 1.1 VHDL Code

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

entity NANDGATE_4 is      -- NANDGATE_4 is identifier name for a 4
    input NAND GATE
    Port ( A : in  STD_LOGIC;                    --Input A of the
       NAND Gate
           B : in  STD_LOGIC;
           C : in  STD_LOGIC;
           D : in  STD_LOGIC;
           O : out  STD_LOGIC);          --Output O
end NANDGATE_4;

architecture Behavioral of NANDGATE_4 is

begin
    -- The output O is assigned as the negation of the AND
       operation of inputs A, B, C, and D

O <= not (A and B and C and D);

end Behavioral;
```

Listing 1: VHDL code for Lab 1_1

### 1.2 Simulation: VHDL Testbench Code

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

 -- Testbench entity for the 4-input NAND gate. No ports are
    needed since this is a self-contained testbench.

ENTITY lab1_tb IS
END lab1_tb;

ARCHITECTURE behavior OF lab1_tb IS

     -- Component declaration for the 4-input NAND gate under
        test.

    COMPONENT NANDGATE_4
    PORT(
         A : IN  std_logic;
         B : IN  std_logic;
         C : IN  std_logic;
         D : IN  std_logic;
         O : OUT  std_logic             --result of NAND operation
        );
    END COMPONENT;


    -- Testbench signals for providing inputs to the NAND gate.

   signal A_tb : std_logic := '0';
   signal B_tb : std_logic := '0';
   signal C_tb : std_logic := '0';
   signal D_tb : std_logic := '0';

        --  Testbench signal to capture the output from the NAND gate.
   signal O_tb : std_logic;



BEGIN
 -- Instantiation of the NANDGATE_4 component.

   uut: NANDGATE_4 PORT MAP (
          A => A_tb,
          B => B_tb,
          C => C_tb,
          D => D_tb,
          O => O_tb
        );

   -- Stimulus process for input A:
    -- Waits for 200 ns and then inverts signal A_tb.


        stim_proc: process
   begin
   wait for 200 ns;
        A_tb <= not A_tb;
   end process;

         stim_proc2: process
   begin
   wait for 100 ns;
        B_tb <= not B_tb;
   end process;

         stim_proc3: process
   begin
   wait for 50 ns;
        C_tb <= not C_tb;
    end process;

         stim_proc4: process
    begin
    wait for 25 ns;
        D_tb <= not D_tb;
    end process;
END;
```

Listing 2: Simulation code for Lab 1_1

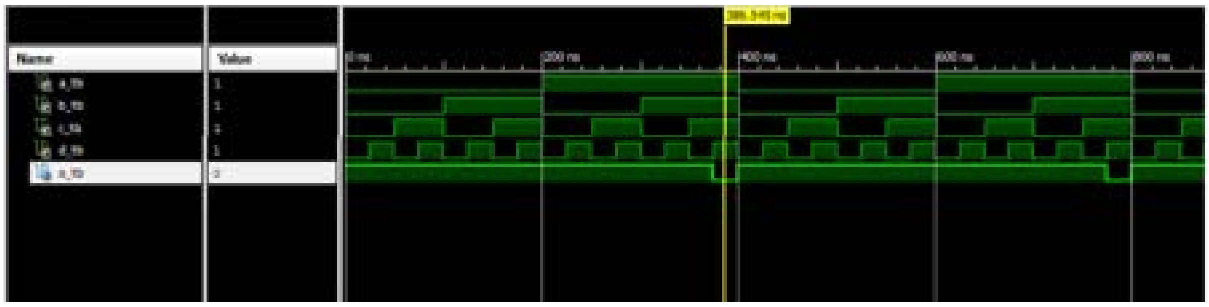


Figure 1: Simulation for test cases in task 1: Output=0

As expected, NAND gate outputs 0 if and only if all the inputs are 1.

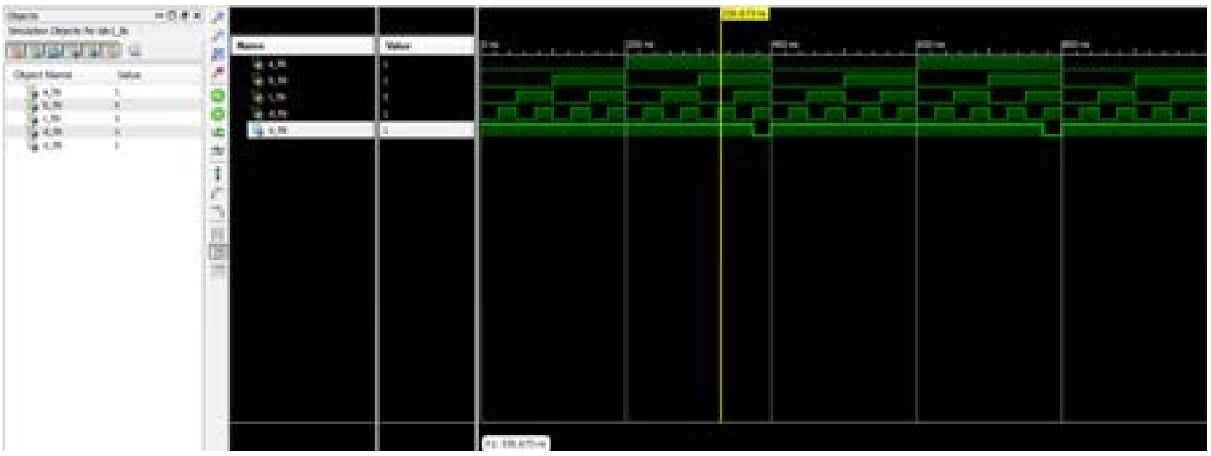


Figure 2: Simulation for testcases in task 1: Output=1

In all other cases, the output of NAND Gate is 1.

## 1.3 Constraints

```
NET "A" LOC = "P11";
NET "B" LOC = "L3";
NET "c" LOC = "K3";
NET "D" LOC = "B4";
NET "O" LOC = "P4";
```

Listing 3: Constraints file for Lab 1_1

## 1.4 Test Cases on FPGA Board

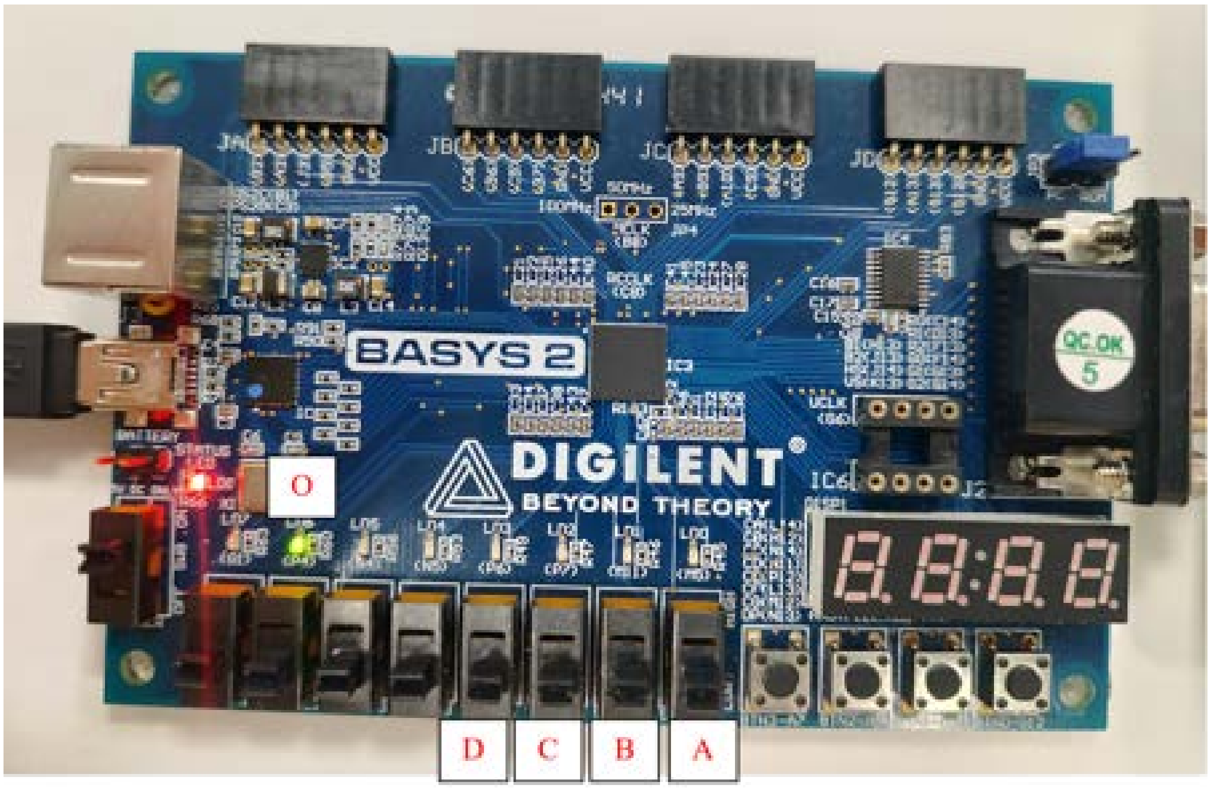


Figure 3: NAND(0000) - expected 1 ✓

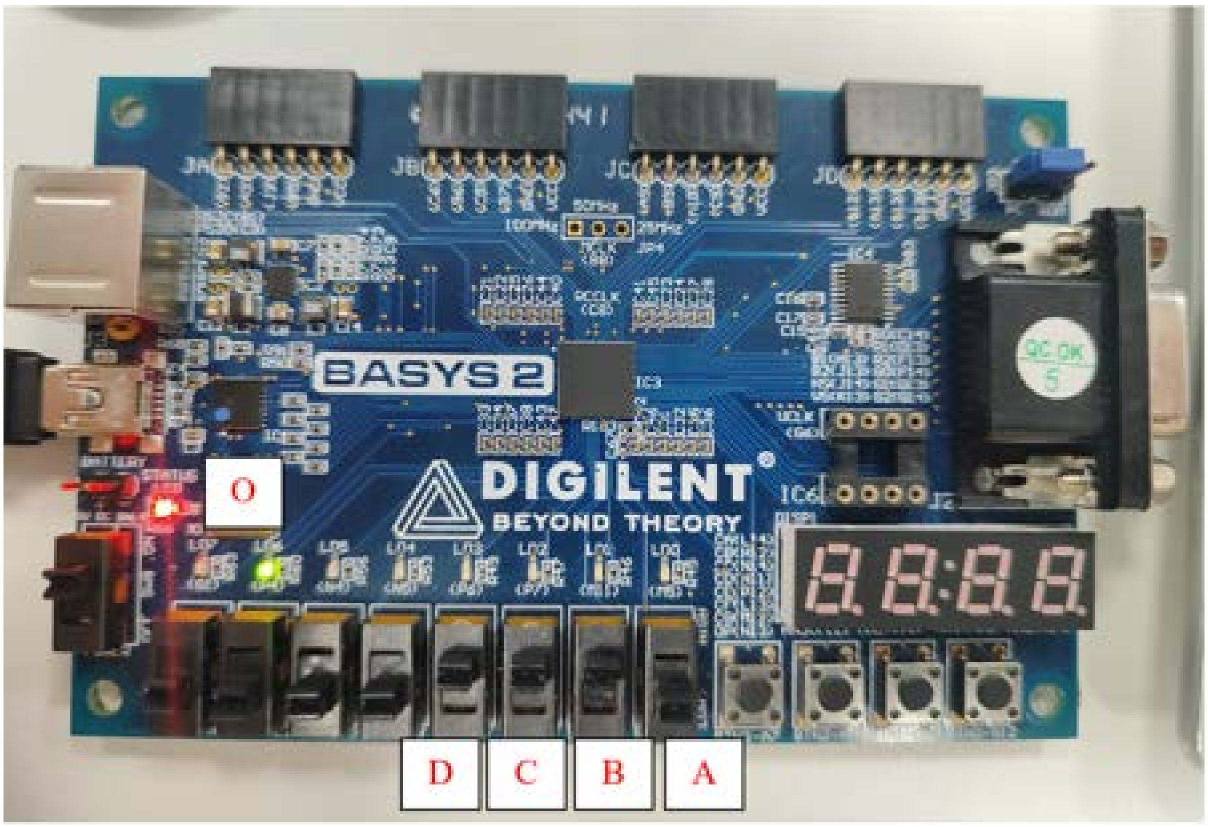


Figure 4: NAND(0111) - expected 0 ✓

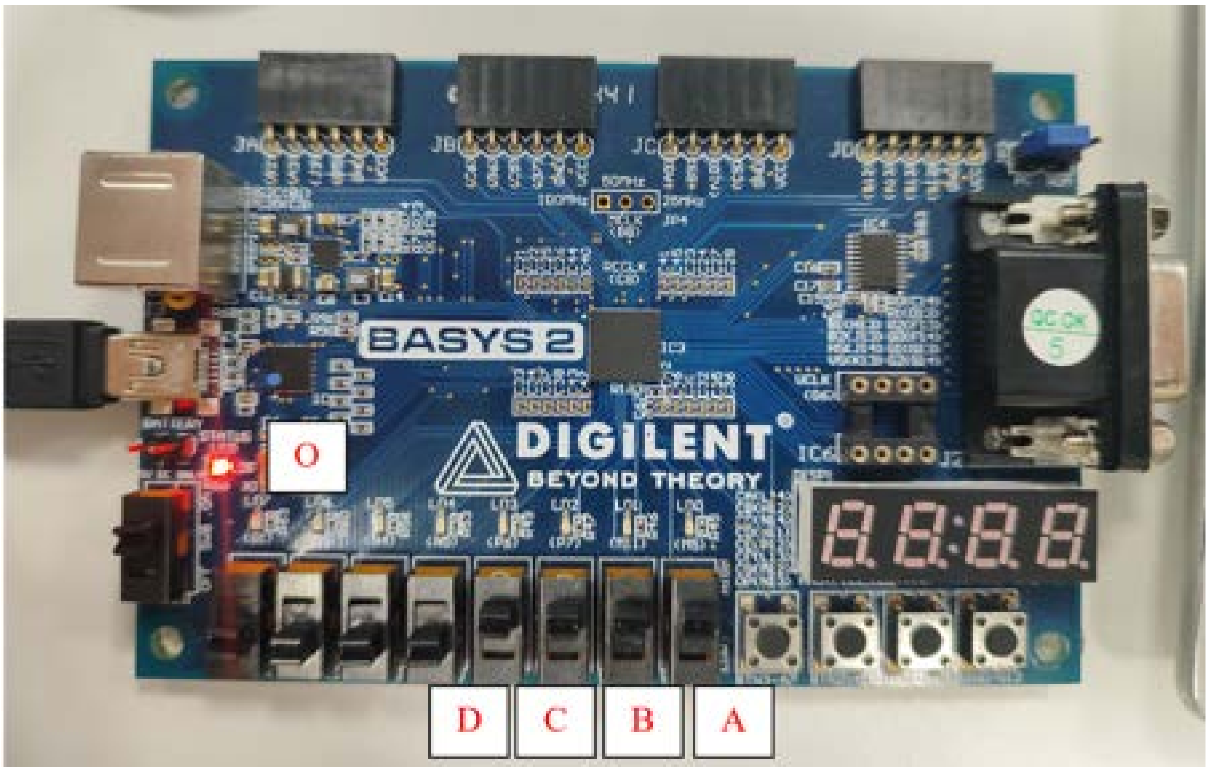


Figure 5: NAND(1111) - expected 0 ✓

## 2 Task 2: Half adder

### 2.1 VHDL Code

```vhdl

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity half_adder is              --half_adder is identifier name
    Port ( A : in  STD_LOGIC;         --first input
           B : in  STD_LOGIC;         --second input
           S : out  STD_LOGIC;        --sum
           C : out  STD_LOGIC);       --carry
end half_adder;

architecture Behavioral of half_adder is

begin
        S <= A xor B;       --the XOR relation can be derived from
            truth table
        C <= A and B;
end Behavioral;
```

Listing 4: VHDL code for Lab 1_2

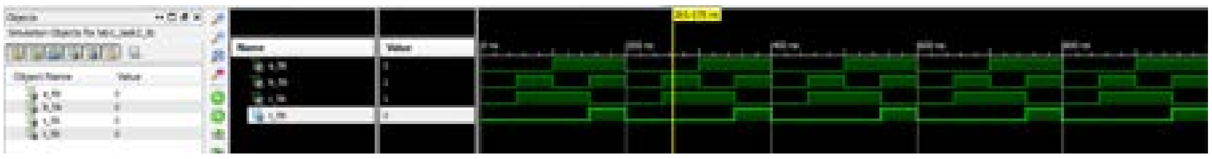
### 2.2 Simulation: VHDL Testbench Code

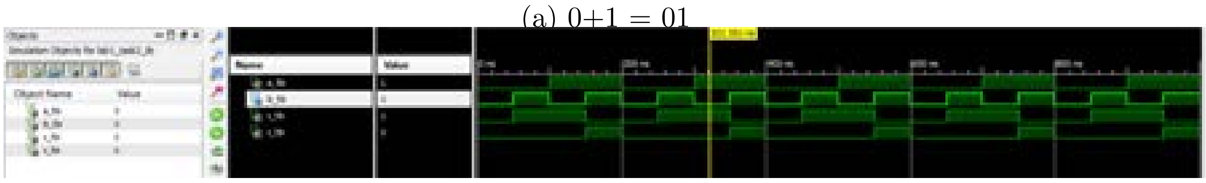
```vhdl

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

                -- testbench enitiy for half adder, no ports are
                   needed since this is a self-contained
                   testbench

ENTITY lab1_task2_tb IS
END lab1_task2_tb;

ARCHITECTURE behavior OF lab1_task2_tb IS

    -- Component Declaration for the half adder

    COMPONENT half_adder
    PORT(
         A : IN  std_logic;
         B : IN  std_logic;
         S : OUT  std_logic;
         C : OUT  std_logic
        );
    END COMPONENT;
```

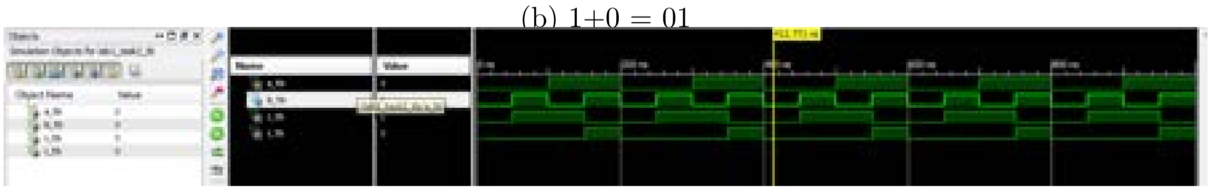
```

        -- testbench signals for providing inputs to the half
           adder

   --Inputs
   signal A_tb : std_logic := '0';
   signal B_tb : std_logic := '0';

                 -- testbench signals of outputs
        --Outputs
   signal S_tb : std_logic;
   signal C_tb : std_logic;


BEGIN

        -- Instantiation of half adder
   uut: half_adder PORT MAP (
          A => A_tb,
          B => B_tb,
          S => S_tb,
          C => C_tb
        );


   -- Stimulus process, times are assigned to ensure different
      values for inputs
   stim_proc1: process
   begin
      wait for 100 ns;
                A_tb <= not A_tb;
   end process;

        stim_proc2: process
   begin
      wait for 50 ns;
                B_tb <= not B_tb;
   end process;

END;
```

Listing 5: Simulation file for Lab 1_2

The simulation results for all 4 cases match the expected results.


(a) 0+1 = 01


(b) 1+0 = 01


(c) 0+0 = 00

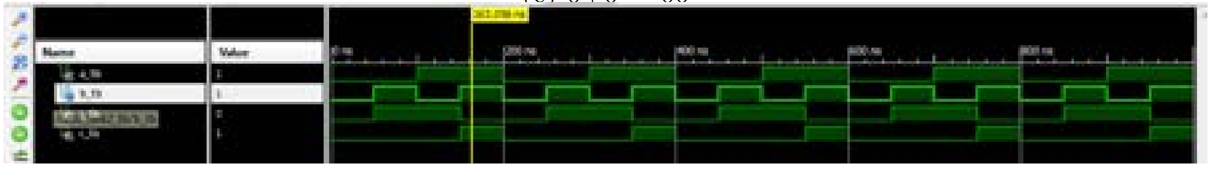

(d) 1+1 = 10

Figure 6: Simulation for test cases in task 2

## 2.3 Constraints

```
NET "A" LOC = "L3";
NET "B" LOC = "P11";
NET "S" LOC = "M11";
NET "C" LOC = "M5";
```

Listing 6: Constraints file for Lab 1_2

## 2.4 Test Cases on FPGA Board

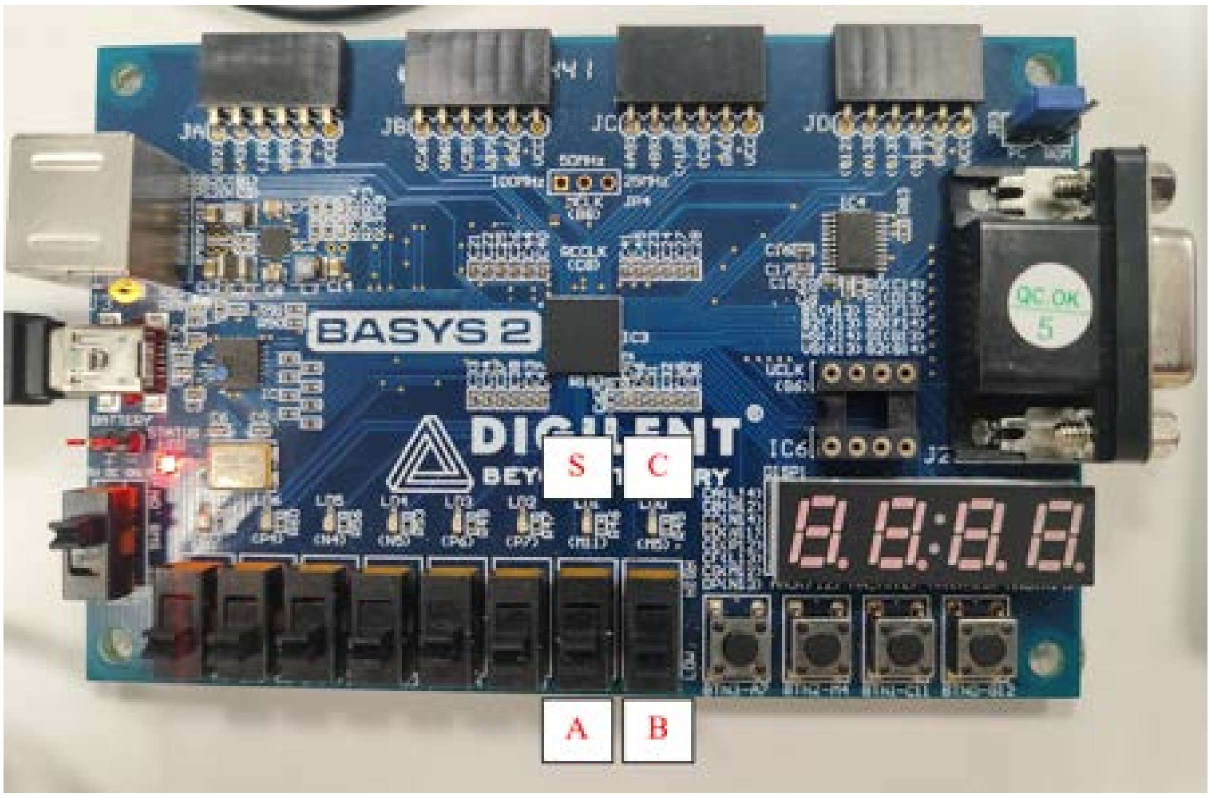


Figure 7: 0 + 0 - expected 00 ✓

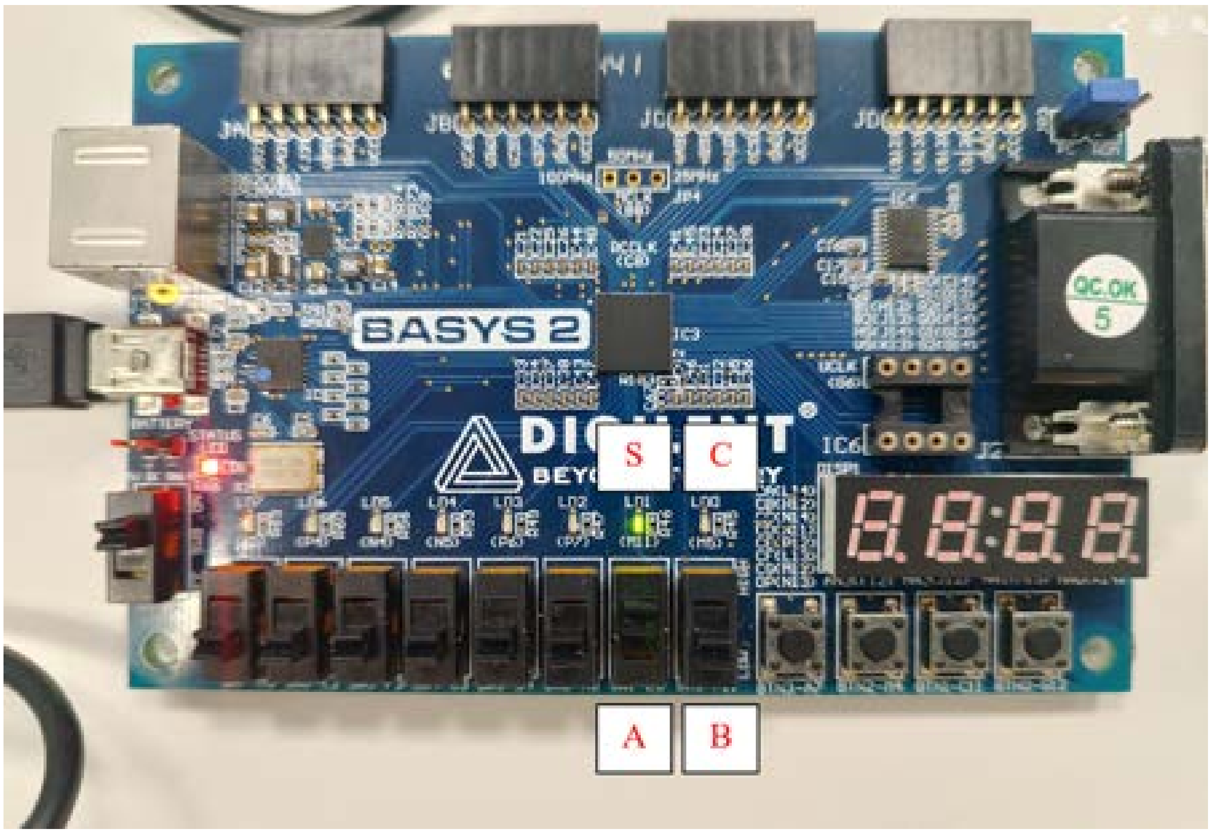


Figure 8: $1 + 0$ - expected 01 ✓

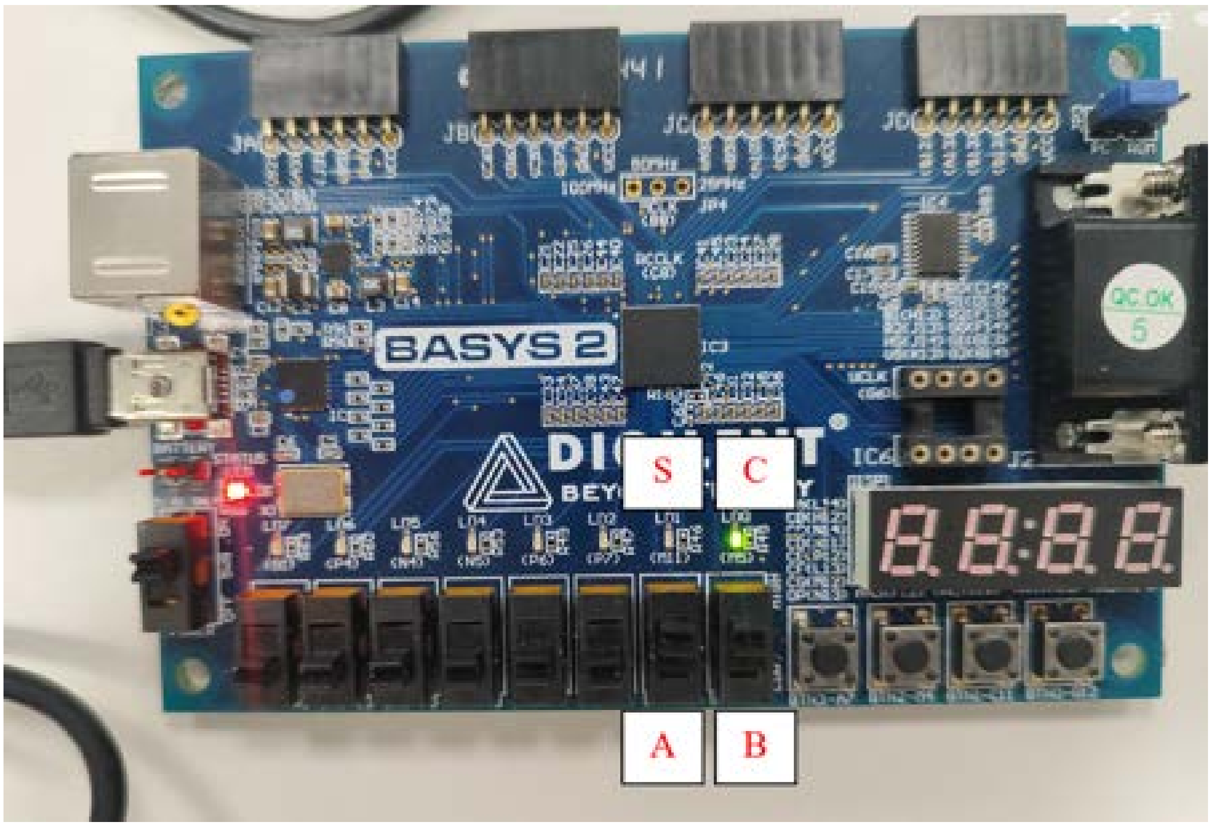


Figure 9: $1 + 1$ - expected 10 ✓

## 3 Task 3: 4-bit binary adder

### 3.1 VHDL Code

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


entity adder_4bit is                        --adder_4bit is identifier name
    Port ( A0 : in  STD_LOGIC;
           A1 : in  STD_LOGIC;
           A2 : in  STD_LOGIC;
           A3 : in  STD_LOGIC;
           B0 : in  STD_LOGIC;
           B1 : in  STD_LOGIC;
           B2 : in  STD_LOGIC;
           B3 : in  STD_LOGIC;
           S0 : out  STD_LOGIC;
           S1 : out  STD_LOGIC;
           S2 : out  STD_LOGIC;
           S3 : out  STD_LOGIC;
           C0 : out  STD_LOGIC); -- carry
end adder_4bit;

architecture Behavioral of adder_4bit is

begin
        S0 <= A0 xor B0;
        S1  <= (A1 xor B1) xor (A0 and B0);
   S2  <= (A2 xor B2) xor ((A1 and B1) or ((A0 and B0) and (A1 xor B1)));
   S3  <= (A3 xor B3) xor ((A2 and B2) or (((A1 and B1) or ((A0 and B0) and (A1 xor B1))) and (A2 xor B2)));
   C0  <= (A3 and B3) or (((A2 and B2) or (((A1 and B1) or ((A0 and B0) and (A1 xor B1))) and (A2 xor B2))) and (A3 xor B3));

-- relations are derived from truth table

end Behavioral;
```

Listing 7: VHDL code for Lab 1_3

## 3.2 Simulation

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

        -- testbench entity for 4 bit adder, no ports are needed
           since this is a self contained testbench

ENTITY lab1_task3_tb IS
END lab1_task3_tb;

ARCHITECTURE behavior OF lab1_task3_tb IS

    -- Component Declaration for the 4 bit adder

    COMPONENT adder_4bit
    PORT(
         A0 : IN  std_logic;
         A1 : IN  std_logic;
         A2 : IN  std_logic;
         A3 : IN  std_logic;
         B0 : IN  std_logic;
         B1 : IN  std_logic;
         B2 : IN  std_logic;
         B3 : IN  std_logic;
         S0 : OUT  std_logic;
         S1 : OUT  std_logic;
         S2 : OUT  std_logic;
         S3 : OUT  std_logic;
         C0 : OUT  std_logic
        );
    END COMPONENT;

         --testbench signals for providing inputs to the 4 bit
            adder

   --Inputs
   signal A0_tb : std_logic := '0';
   signal A1_tb : std_logic := '0';
   signal A2_tb : std_logic := '0';
   signal A3_tb : std_logic := '0';
   signal B0_tb : std_logic := '0';
   signal B1_tb : std_logic := '0';
   signal B2_tb : std_logic := '0';
   signal B3_tb : std_logic := '0';

        --Outputs
   signal S0_tb : std_logic;
   signal S1_tb : std_logic;
   signal S2_tb : std_logic;
```

```
    signal S3_tb : std_logic;
    signal C0_tb : std_logic;



BEGIN


    uut: adder_4bit PORT MAP (
           A0 => A0_tb,
           A1 => A1_tb,
           A2 => A2_tb,
           A3 => A3_tb,
           B0 => B0_tb,
           B1 => B1_tb,
           B2 => B2_tb,
           B3 => B3_tb,
           S0 => S0_tb,
           S1 => S1_tb,
           S2 => S2_tb,
           S3 => S3_tb,
           C0 => C0_tb
         );



    -- Stimulus process
    stim_proc: process

         -- 3 randomly chosen test cases were selected (otherwise
            it could be more

    begin
       --test 1
         A0_tb <= '1';
         A1_tb <= '1';
         A2_tb <= '1';
         A3_tb <= '1';
         B0_tb <= '0';
         B1_tb <= '0';
         B2_tb <= '1';
         B3_tb <= '0';

       wait for 100 ns;
    --test 2
         A0_tb <= '0';
         A1_tb <= '0';
         A2_tb <= '0';
         A3_tb <= '1';
         B0_tb <= '1';
         B1_tb <= '1';
```

```
        B2_tb <= '1';
        B3_tb <= '1';

      wait for 150 ns;
      --test 3
        A0_tb <= '1';
        A1_tb <= '1';
        A2_tb <= '1';
        A3_tb <= '1';
        B0_tb <= '1';
        B1_tb <= '1';
        B2_tb <= '1';
        B3_tb <= '1';
      wait for 150 ns;


         end process;

END;
```

Listing 8: Simulation code for Lab 1_3

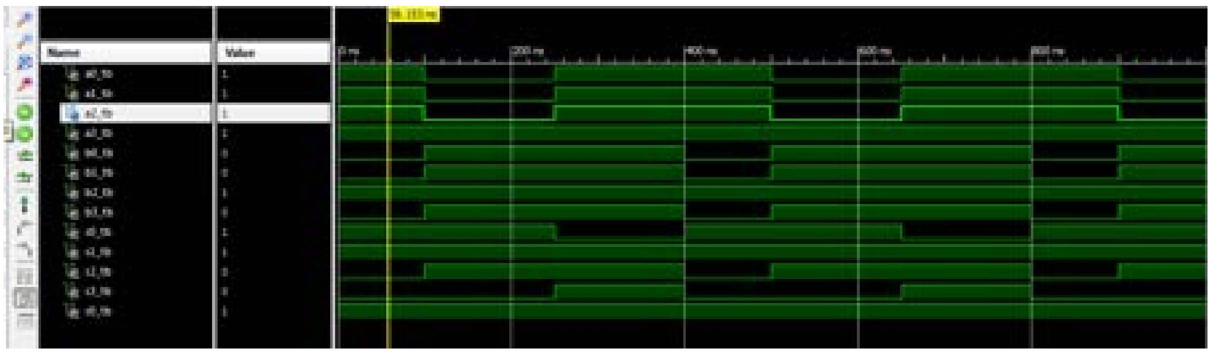


Figure 10: Simulation for test cases in task 3: 1111 + 0100 = 10011

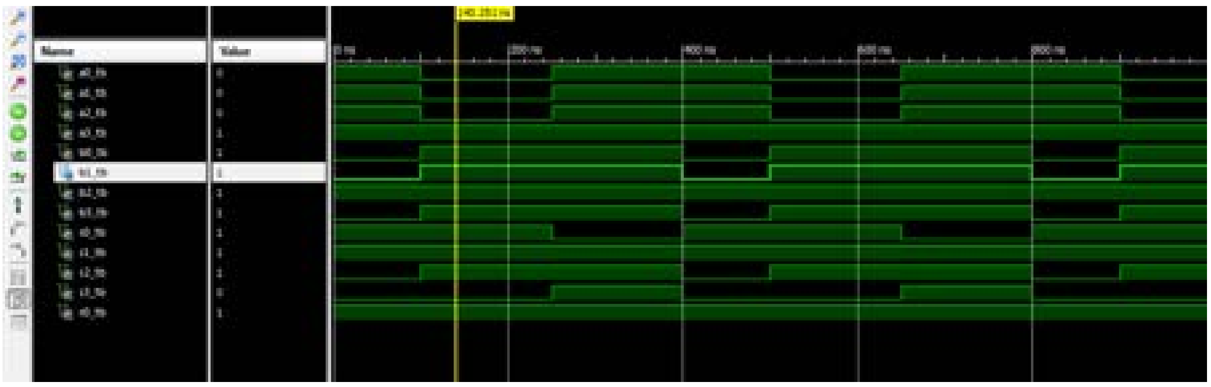


Figure 11: Simulation for test cases in task 3: 1000 + 1111 = 10111

The simulation results for all 3 cases match the expected results.

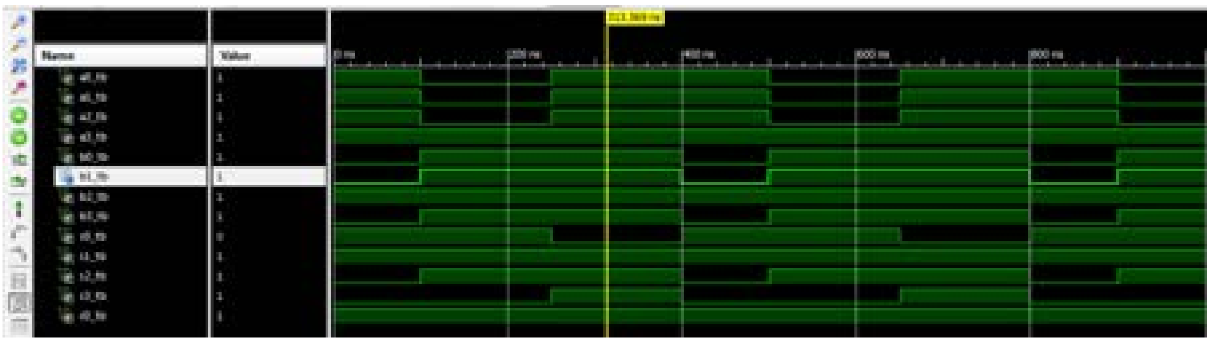


Figure 12: Simulation for test cases in task 3: 1111 + 1111 = 11110

### 3.3 Constraints

```
Net "A3" LOC = "N3";
Net "A2" LOC = "E2";
Net "A1" LOC = "F3";
Net "A0" LOC = "G3";
Net "B3" LOC = "B4";
Net "B2" LOC = "K3";
Net "B1" LOC = "L3";
Net "B0" LOC = "P11";
Net "S3" LOC = "P6";
Net "S2" LOC = "P7";
Net "S1" LOC = "M11";
Net "S0" LOC = "M5";
Net "C0" LOC = "N5";
```

Listing 9: Constraints file for Lab 1_3

## 3.4 Test Cases on FPGA Board

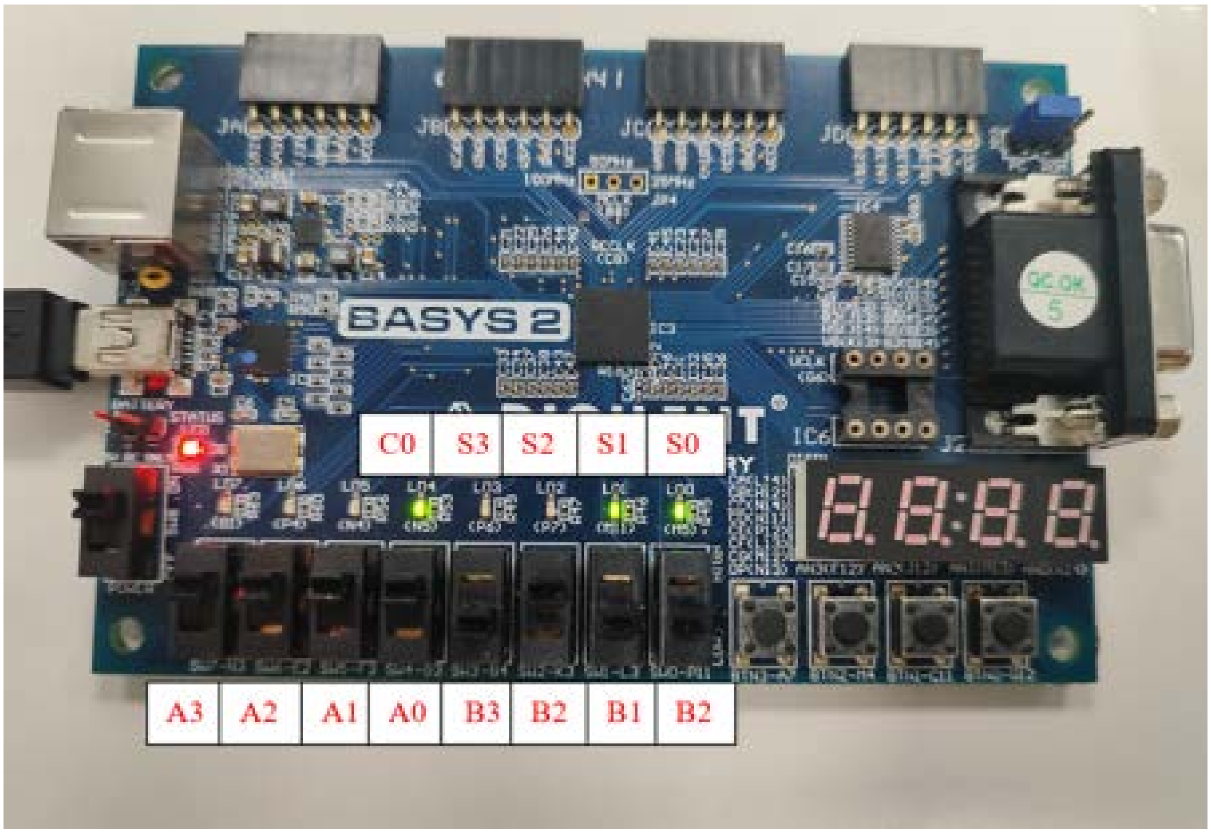


Figure 13: 1111 + 0100 : expected 10011 ✓

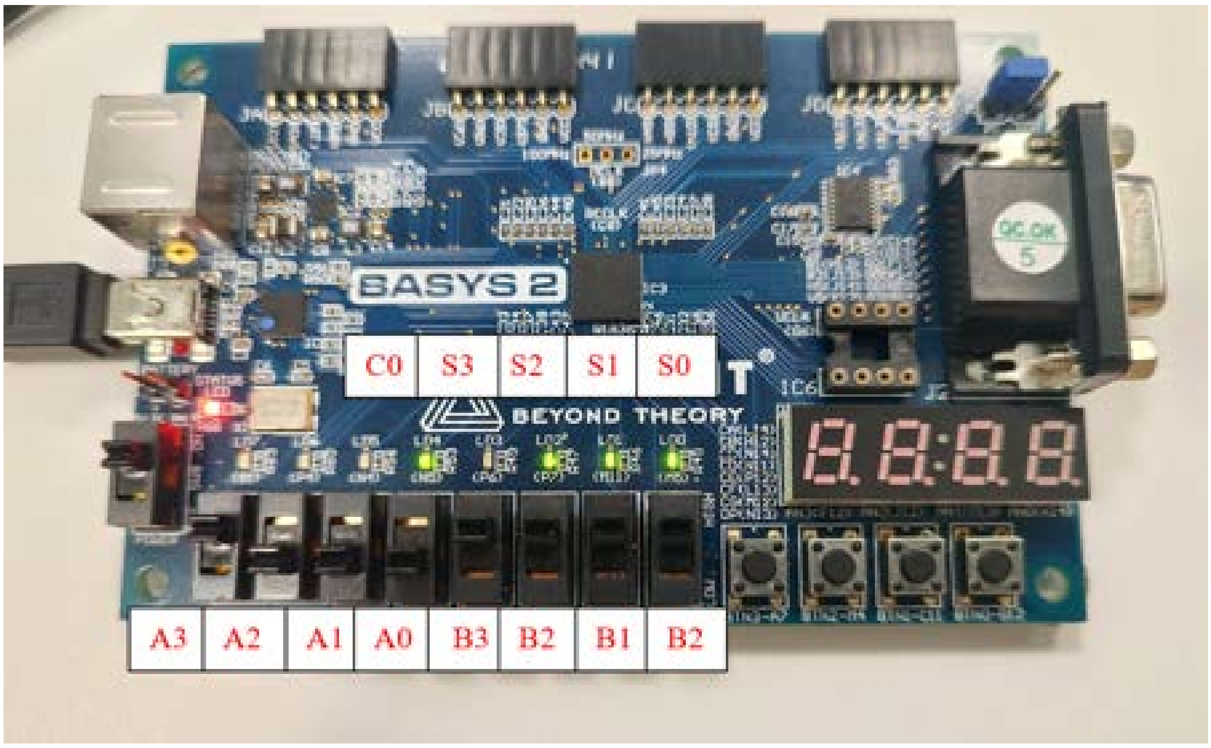


Figure 14: 1000 + 1111 : expected 10111 ✓

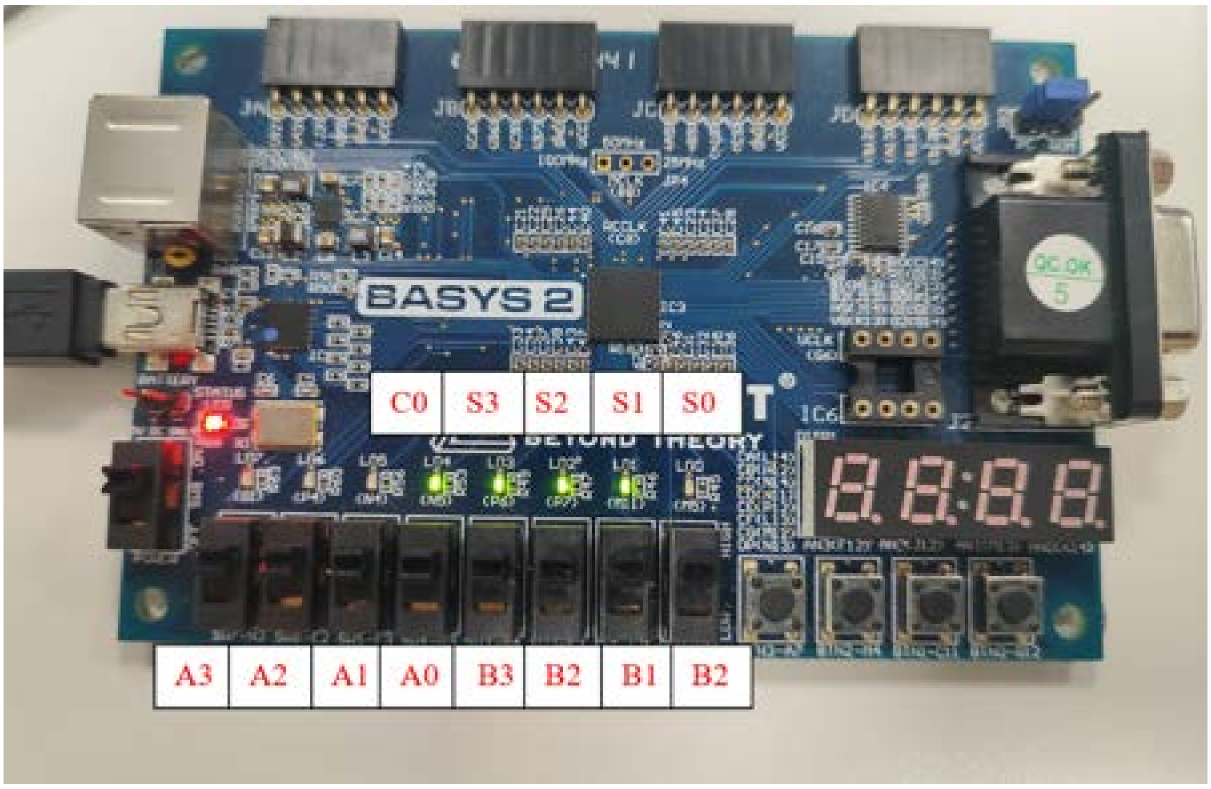


Figure 15: 1111 + 1111 : expected 11110 ✓

## 4 Task 4: Implementing given 4-input Minterms

### 4.1 VHDL code

F1 is implemented by concurrent signal assignment; F2 is implemented by sequential Case-When statements.

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

entity minterm is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           C : in  STD_LOGIC;
           D : in  STD_LOGIC;
           F1 : out  STD_LOGIC;
           F2 : out  STD_LOGIC);
end minterm;

architecture Behavioral of minterm is

begin

 -- 1. Concurrent signal assignment (logic equation)
    F1 <= (not A and not B and not C and D)  -- Minterm 1 (0001)
       or (not A and not B and C and D)      -- Minterm 3 (0011)
       or (A and not B and not C and D)      -- Minterm 9 (1001)
       or (A and not B and C and D);         -- Minterm 11 (1011)

 -- 2. Sequential (Case-When inside a process)
    process(A, B, C, D)
        variable input_vector : std_logic_vector(3 downto 0);
    begin
        input_vector := (A & B & C & D);  -- Convert inputs to a
           4-bit vector
        F2 <= '0';  -- F2 default value
        case input_vector is
            when "0001" => F2 <= '1';  -- Minterm 1
            when "0011" => F2 <= '1';  -- Minterm 3
            when "1001" => F2 <= '1';  -- Minterm 9
            when "1011" => F2 <= '1';  -- Minterm 11
            when others => F2 <= '0';  -- Default case to prevent
                latch
        end case;
    end process;

end Behavioral;
```

Listing 10: VHDL code for Lab 1_4

## 4.2 Simulation: VHDL Testbench Code

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

ENTITY minterm_tb IS
END minterm_tb;

ARCHITECTURE behavior OF minterm_tb IS

    -- Component Declaration
    COMPONENT minterm
    PORT(
         A : IN  std_logic;
         B : IN  std_logic;
         C : IN  std_logic;
         D : IN  std_logic;
         F1 : OUT std_logic;
         F2 : OUT std_logic
        );
    END COMPONENT;

    -- Inputs
    SIGNAL A_tb : std_logic := '0';
    SIGNAL B_tb : std_logic := '0';
    SIGNAL C_tb : std_logic := '0';
    SIGNAL D_tb : std_logic := '0';

    -- Outputs
    SIGNAL F1_tb : std_logic;
    SIGNAL F2_tb : std_logic;

BEGIN
    -- Instantiation
    uut: minterm PORT MAP (
          A => A_tb,
          B => B_tb,
          C => C_tb,
          D => D_tb,
          F1 => F1_tb,
          F2 => F2_tb
        );

    -- Stimulus process
    stim_proc1: process
    begin
        -- Toggle A every 200 ns
        wait for 200 ns;
        A_tb <= not A_tb;
    end process;

    stim_proc2: process
    begin
        -- Toggle B every 100 ns
        wait for 100 ns;
        B_tb <= not B_tb;
    end process;

    stim_proc3: process
    begin
        -- Toggle C every 50 ns
        wait for 50 ns;
        C_tb <= not C_tb;
    end process;

    stim_proc4: process
    begin
        -- Toggle D every 25 ns
        wait for 25 ns;
        D_tb <= not D_tb;
    end process;

end behavior;
```

Listing 11: VHDL Testbench code Lab 1_4

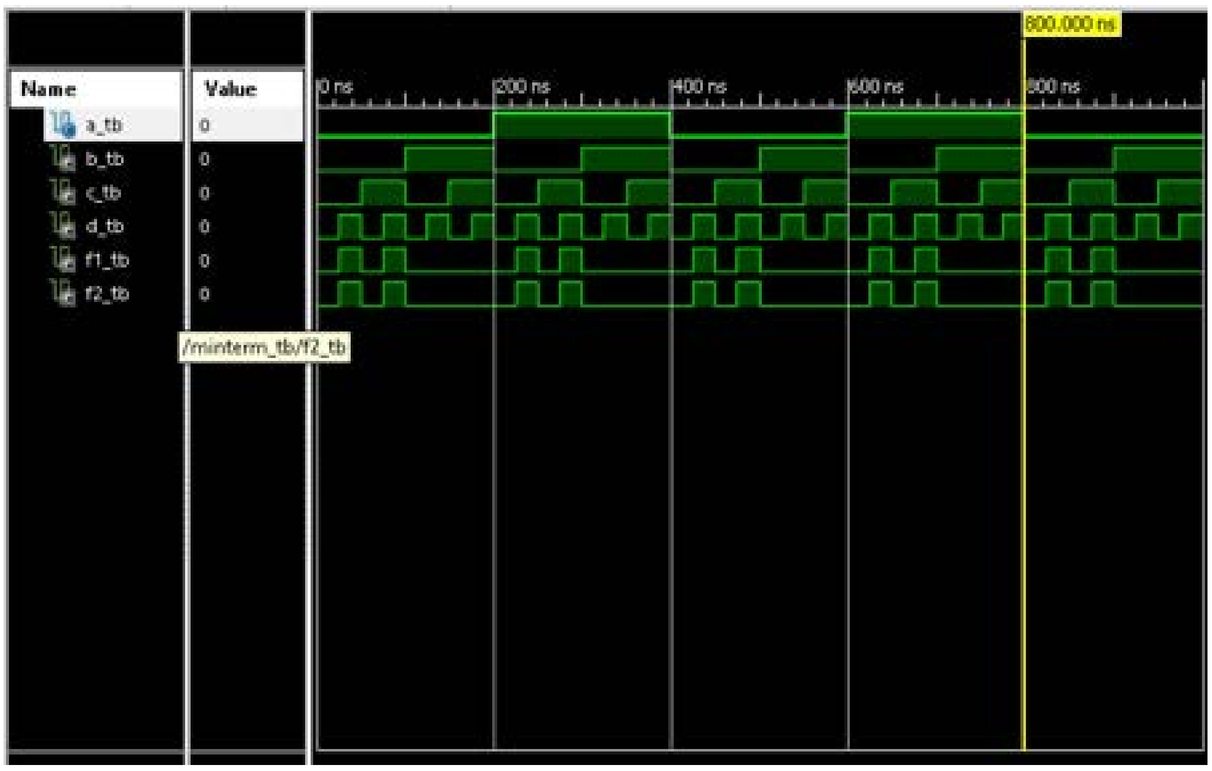


Figure 16: Simulation for task 4

The simulation demonstrates that F1 and F2 are identically activated if and only if the inputs are 0001, 0011, 1001, and 1011.

## 4.3 Constraints

```
Net "A" LOC = "N3";
Net "B" LOC = "E2";
Net "C" LOC = "F3";
Net "D" LOC = "G3";
Net "F1" LOC = "M11";
Net "F2" LOC = "M5";
```

Listing 12: Constrains for Lab 1_4

## 4.4 Test Cases on FPGA Board

Below are testcases for each Minterm: 1(0001), 3(0011), 9(1001), 11(1011), and an extra case that is not a Minterm: 5(0101). The LEDs F1 and F2 are the output signal, where F1 is defined by the Concurrent signal assignment (logic equation) and F2 is defined by the Sequential assignment (Case-When). Note that activation of F1 and F2 are identical, meaning both methods of implementation produce the same result.
Figure 5 is a testcase for a non-minterm (i.e. 5).

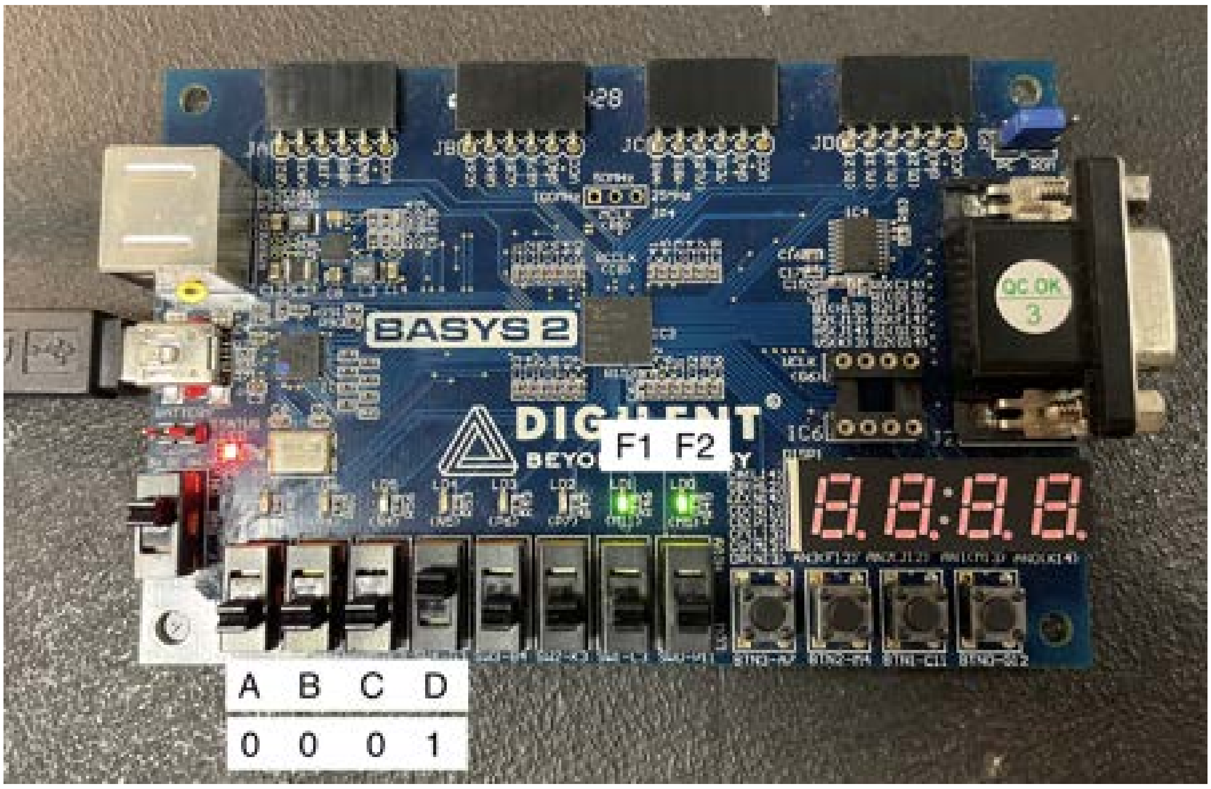


Figure 17: Minterm 1(0001) - expected 1✓

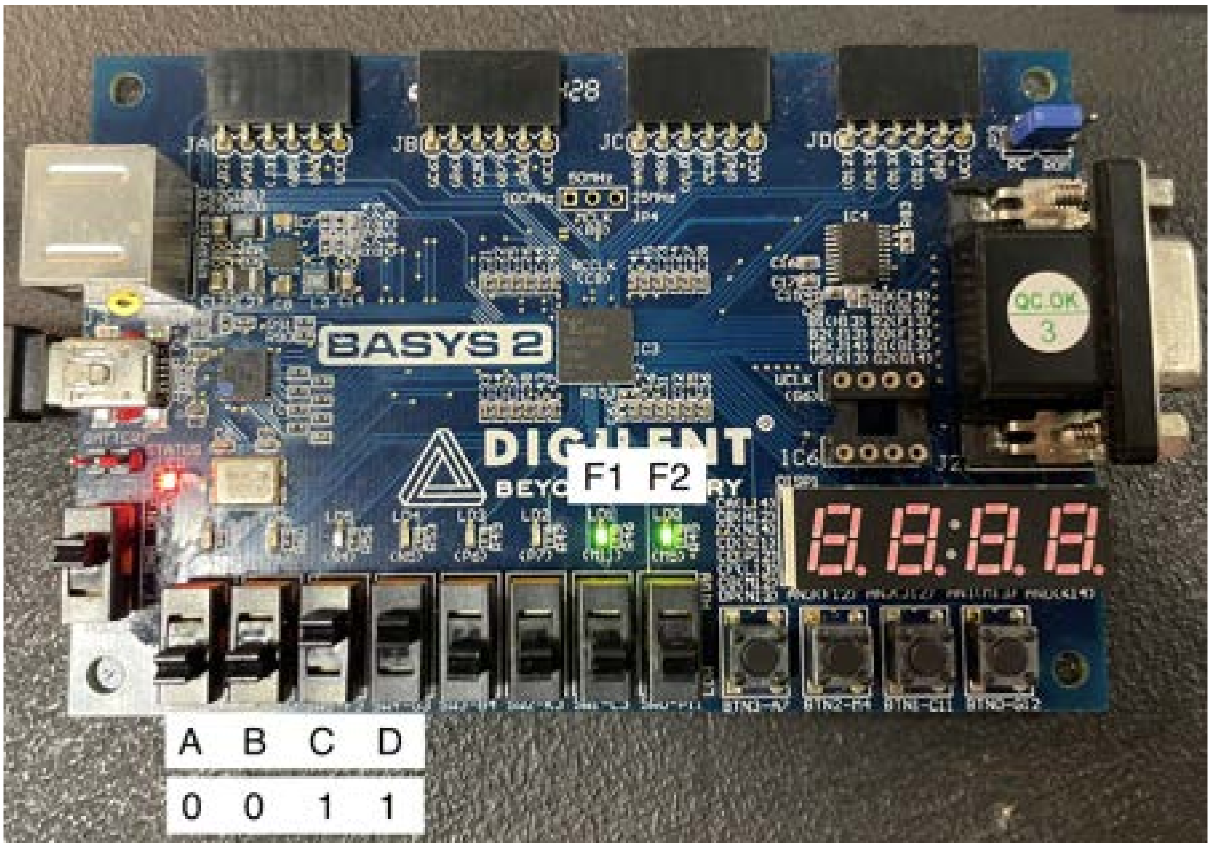


Figure 18: Minterm 3(0011) - expected 1✓

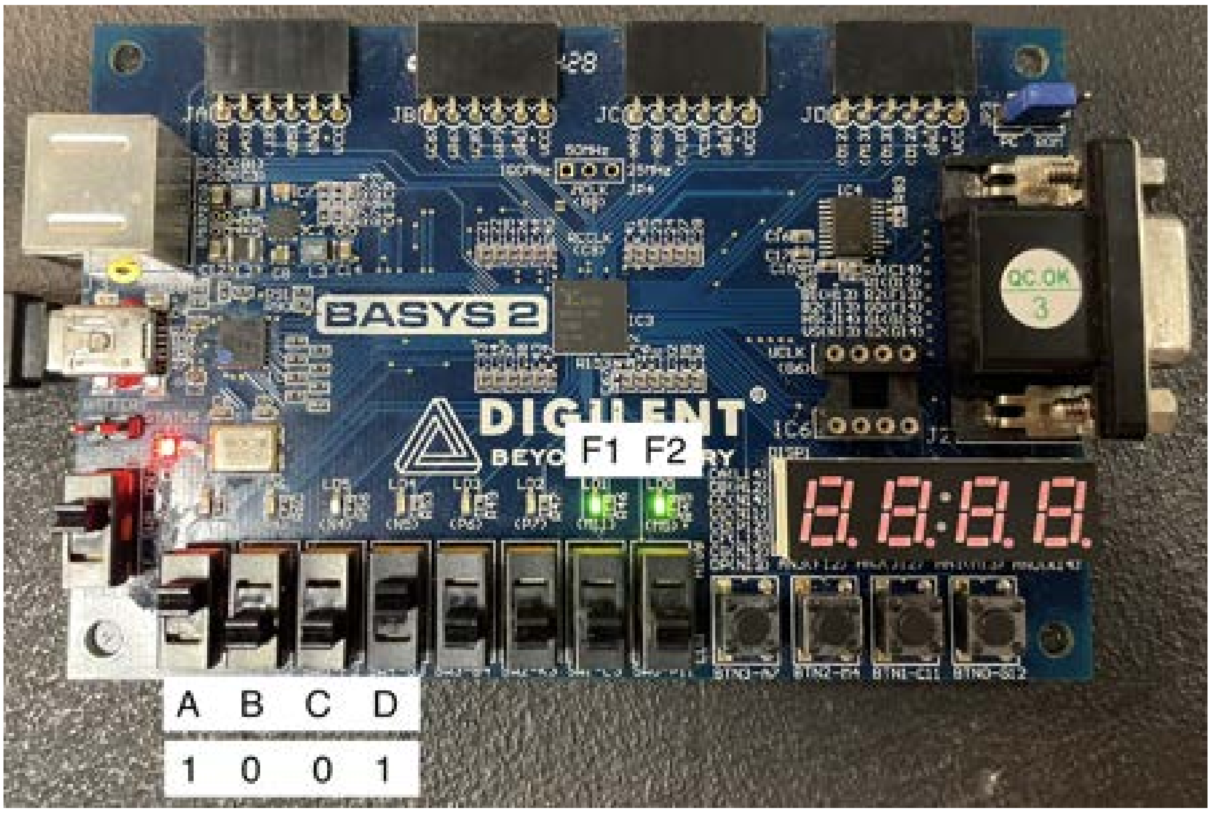


Figure 19: Minterm 9(1001) - expected 1✓

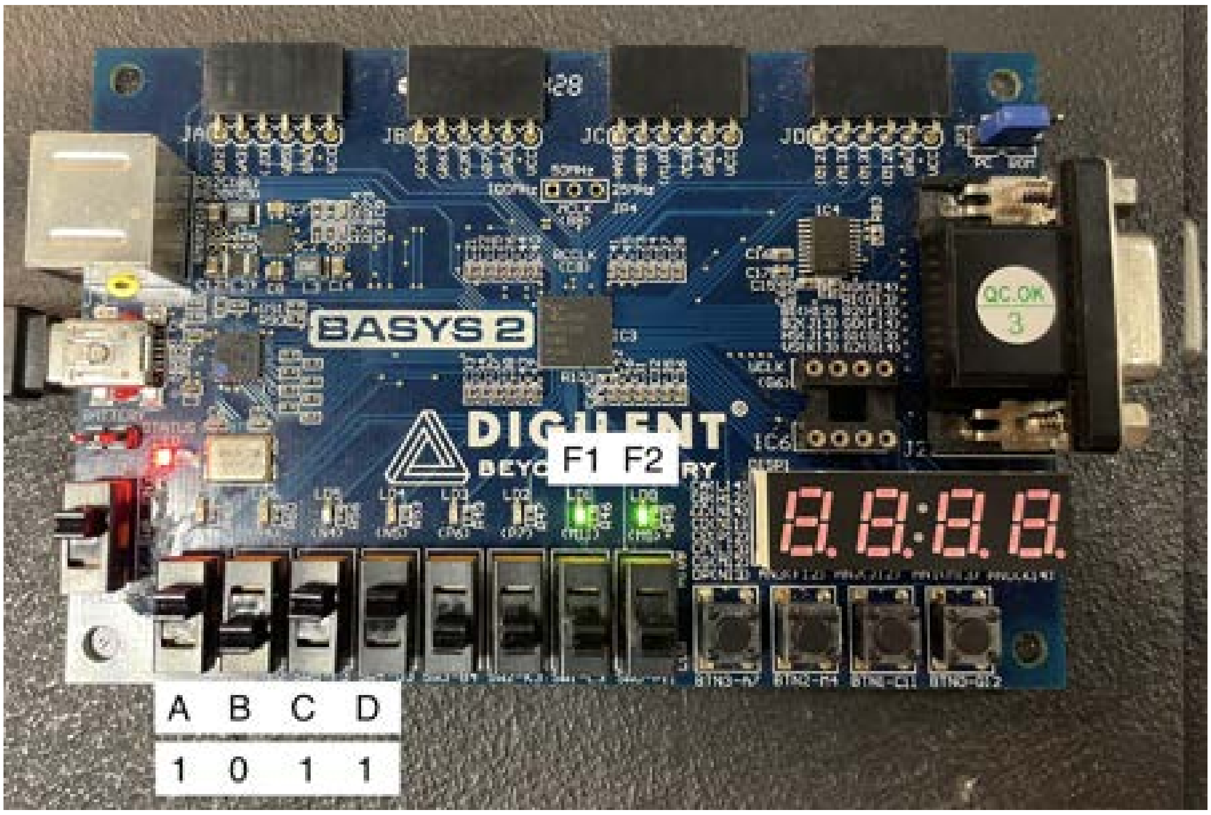


Figure 20: Minterm 11(1011) - expected 1✓

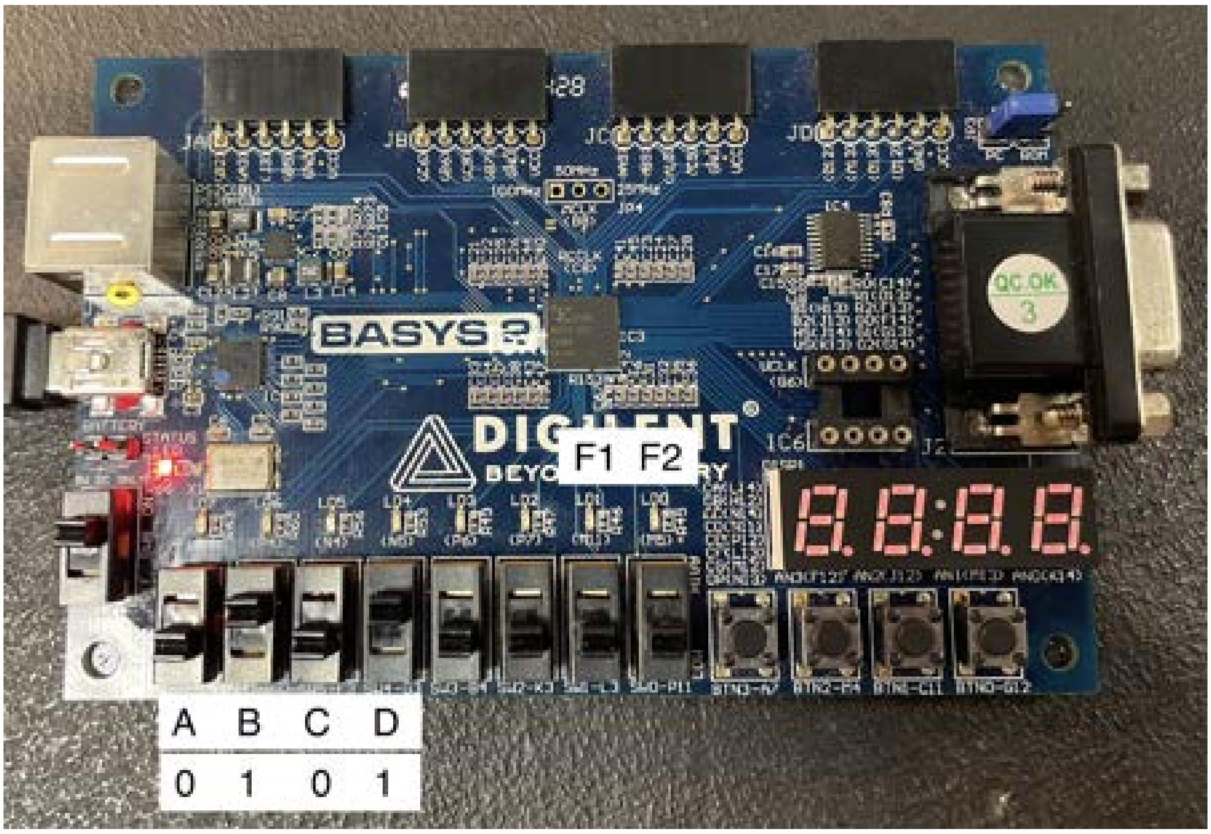


Figure 21: Minterm 5(0101) - expected 0✓

## 5 Task 5: 8:1 MUX

### 5.1 VHDL code

***Case statements***

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

entity Lab1_5_MUX is
    Port ( D0 : in  STD_LOGIC;
           D1 : in  STD_LOGIC;
           D2 : in  STD_LOGIC;
           D3 : in  STD_LOGIC;
           D4 : in  STD_LOGIC;
           D5 : in  STD_LOGIC;
           D6 : in  STD_LOGIC;
           D7 : in  STD_LOGIC;
           S0 : in  STD_LOGIC;
           S1 : in  STD_LOGIC;
                          S2 : in  STD_LOGIC;
           O : out  STD_LOGIC);
end Lab1_5_MUX;

architecture Behavioral of Lab1_5_MUX is
begin
    -- Multiplexer logic using a process
    process (D0, D1, D2, D3, D4, D5, D6, D7, S0, S1, S2)
    begin

        case STD_LOGIC_VECTOR'(S2 & S1 & S0) is
            when "000" => O <= D0;
            when "001" => O <= D1;
            when "010" => O <= D2;
            when "011" => O <= D3;
            when "100" => O <= D4;
            when "101" => O <= D5;
            when "110" => O <= D6;
            when "111" => O <= D7;
            when others => O <= '0';  -- Default case (should not
                normally happen)
        end case;
    end process;
end Behavioral;
```

Listing 13: VHDL using case statements for Lab 1_5 MUX

***With-Select***

```
-- Architecture Definition using With-Select
architecture Behavioral of Lab1_5_MUX is
begin
        -- Use the combined value of S2, S1, and S0 as the
           selection signal
        with STD_LOGIC_VECTOR'(S2 & S1 & S0) select
                O <= D0 when "000",
                          D1 when "001",
                          D2 when "010",
                          D3 when "011",
                          D4 when "100",
                          D5 when "101",
                          D6 when "110",
                          D7 when "111",
                          '0' when others;

end Behavioral;
```

Listing 14: Testbench for Lab 1_5 MUX

***When-Else***

```
-- Architecture Definition using With-Select
architecture Behavioral of Lab1_5_MUX is
begin
        -- Use S2, S1, and S0 to select the appropriate input
        O <= D0 when (S2 = '0' and S1 = '0' and S0 = '0') else
             D1 when (S2 = '0' and S1 = '0' and S0 = '1') else
             D2 when (S2 = '0' and S1 = '1' and S0 = '0') else
             D3 when (S2 = '0' and S1 = '1' and S0 = '1') else
             D4 when (S2 = '1' and S1 = '0' and S0 = '0') else
             D5 when (S2 = '1' and S1 = '0' and S0 = '1') else
             D6 when (S2 = '1' and S1 = '1' and S0 = '0') else
             D7 when (S2 = '1' and S1 = '1' and S0 = '1') else
             '0';
end Behavioral;
```

Listing 15: Testbench for Lab 1_5 MUX

## 5.2 Theoretical question

**VHDL programming difference between Case statements and the With-Select/ When-Else statements:**

Case statements are sequential, meaning they are evaluated only when there is an event on the signal list. They are placed inside a *process* block. Meanwhile, With-select and When-else are both concurrent statements that are always active within the architecture, and they do not require process blocks. With-select is compact and suitable for direct mapping of select signals to outputs, like multiplexers. Unlike with-select, When-else evaluates conditions sequentially from top to bottom, which can introduce priority encoding for evaluating conditions in order.

## 5.3 Simulation: VHDL Testbench Code

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY lab1_5_tb IS
END lab1_5_tb;

ARCHITECTURE behavior OF lab1_5_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Lab1_5_MUX
    PORT(
         D0 : IN  std_logic;
         D1 : IN  std_logic;
         D2 : IN  std_logic;
         D3 : IN  std_logic;
         D4 : IN  std_logic;
         D5 : IN  std_logic;
         D6 : IN  std_logic;
         D7 : IN  std_logic;
         S0 : IN  std_logic;
         S1 : IN  std_logic;
         S2 : IN  std_logic;
         O : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal D0_tb : std_logic := '0';
   signal D1_tb : std_logic := '0';
   signal D2_tb : std_logic := '0';
   signal D3_tb : std_logic := '0';
   signal D4_tb : std_logic := '0';
   signal D5_tb : std_logic := '0';
   signal D6_tb : std_logic := '0';
   signal D7_tb : std_logic := '0';
   signal S0_tb : std_logic := '0';
   signal S1_tb : std_logic := '0';
   signal S2_tb : std_logic := '0';

        --Outputs
   signal O_tb : std_logic;


```

```vhdl
BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: Lab1_5_MUX PORT MAP (
          D0 => D0_tb,
          D1 => D1_tb,
          D2 => D2_tb,
          D3 => D3_tb,
          D4 => D4_tb,
          D5 => D5_tb,
          D6 => D6_tb,
          D7 => D7_tb,
          S0 => S0_tb,
          S1 => S1_tb,
          S2 => S2_tb,
          O => O_tb
        );



   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

      -- Stimulus 1: Select all 0 (S2=0, S1=0, S0=0)
      S0_tb <= '0';
      S1_tb <= '0';
      S2_tb <= '0';
      D0_tb <= '0';
      D1_tb <= '0';
      D2_tb <= '0';
      D3_tb <= '0';
      D4_tb <= '0';
      D5_tb <= '0';
      D6_tb <= '0';
      D7_tb <= '0';
      wait for 100 ns;

      -- Stimulus 2: Select D0 (S2=0, S1=0, S0=0)
      S0_tb <= '0';
      S1_tb <= '0';
      S2_tb <= '0';
      D0_tb <= '1'; -- Set D1 to 1, others to 0
      D1_tb <= '0';
      D2_tb <= '0';
      D3_tb <= '0';
      D4_tb <= '0';
      D5_tb <= '0';
      D6_tb <= '0';
```

```
        D7_tb <= '0';
        wait for 100 ns;

        -- Stimulus 3: Select D2 (S2=0, S1=1, S0=0)
        S0_tb <= '0';
        S1_tb <= '1';
        S2_tb <= '0';
        D0_tb <= '0';
        D1_tb <= '0';
        D2_tb <= '1';  -- Set D2 to 1, others to 0
        D3_tb <= '0';
        D4_tb <= '0';
        D5_tb <= '0';
        D6_tb <= '0';
        D7_tb <= '0';
        wait for 100 ns;

        -- Stimulus 4: Select D3 (S2=0, S1=1, S0=1)
        S0_tb <= '1';
        S1_tb <= '1';
        S2_tb <= '0';
        D0_tb <= '0';
        D1_tb <= '0';
        D2_tb <= '0';
        D3_tb <= '1';  -- Set D3 to 1, others to 0
        D4_tb <= '0';
        D5_tb <= '0';
        D6_tb <= '0';
        D7_tb <= '0';
        wait for 100 ns;



        -- End of test
        wait;
    end process;

END;
```

Listing 16: Testbench for Lab 1_5 MUX

Here we can see that the multiplexer is working correctly, giving the correct Data signal based on S0, S1, S2. It workes the same on the FGPA board as well.

Table 1: Simulation Inputs and Outputs for 8-to-1 Multiplexer

| **S2** | **S1** | **S0** | **D0** | **D1** | **D2** | **D3** | **D4** | **D5** | **D6** | **D7** | **O** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

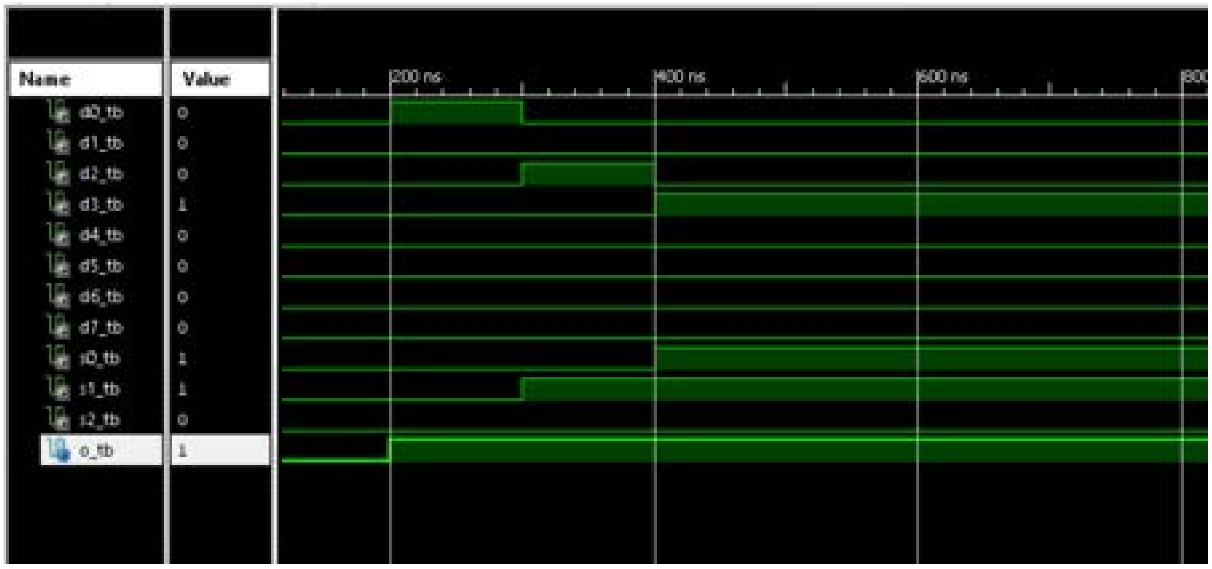


Figure 22: Simulation for 3 test cases for task 5

This table does a good job of visualizing that the multiplexer is working. When select is 0, output is D0, and so on.

## 5.4 Constraints

```
NET "D0"  LOC = "P11";
NET "D1"  LOC = "L3";
NET "D2"  LOC = "K3";
NET "D3"  LOC = "B4";
NET "D4"  LOC = "G3";
NET "D5"  LOC = "F3";
NET "D6"  LOC = "E2";
NET "D7"  LOC = "N3";

NET "S0"  LOC = "G12";
NET "S1"  LOC = "C11";
NET "S2"  LOC = "M4";

NET "O"   LOC = "M5";
```

Listing 17: Pin Assignments for Lab 1 MUX

## 5.5 Test Cases on FPGA Board

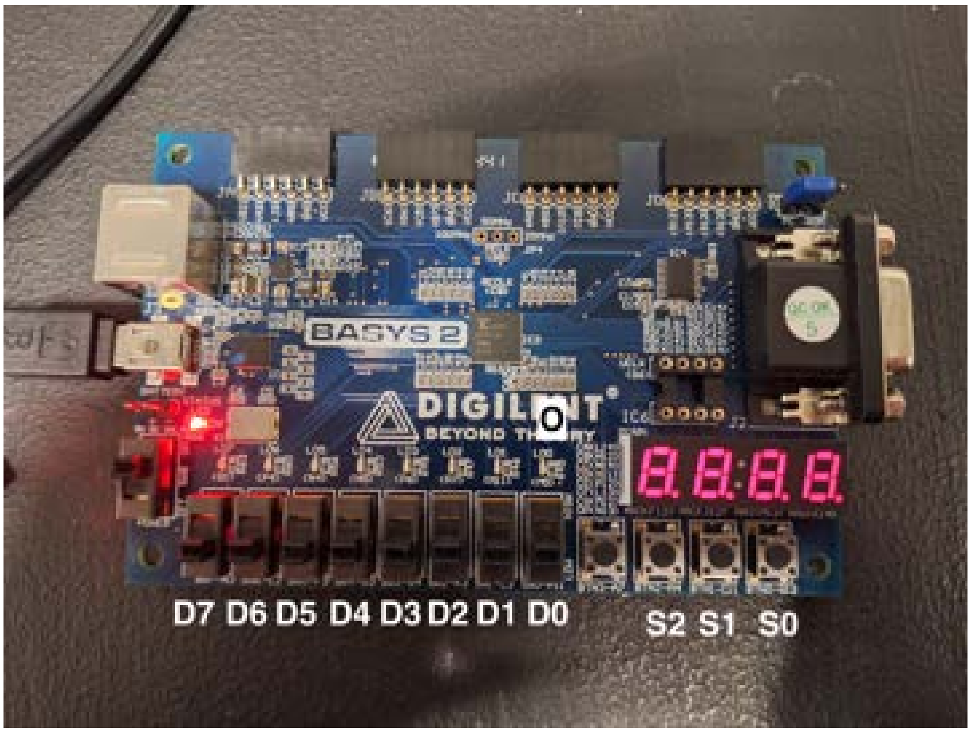


Figure 23: Everything off - expected 0 ✓

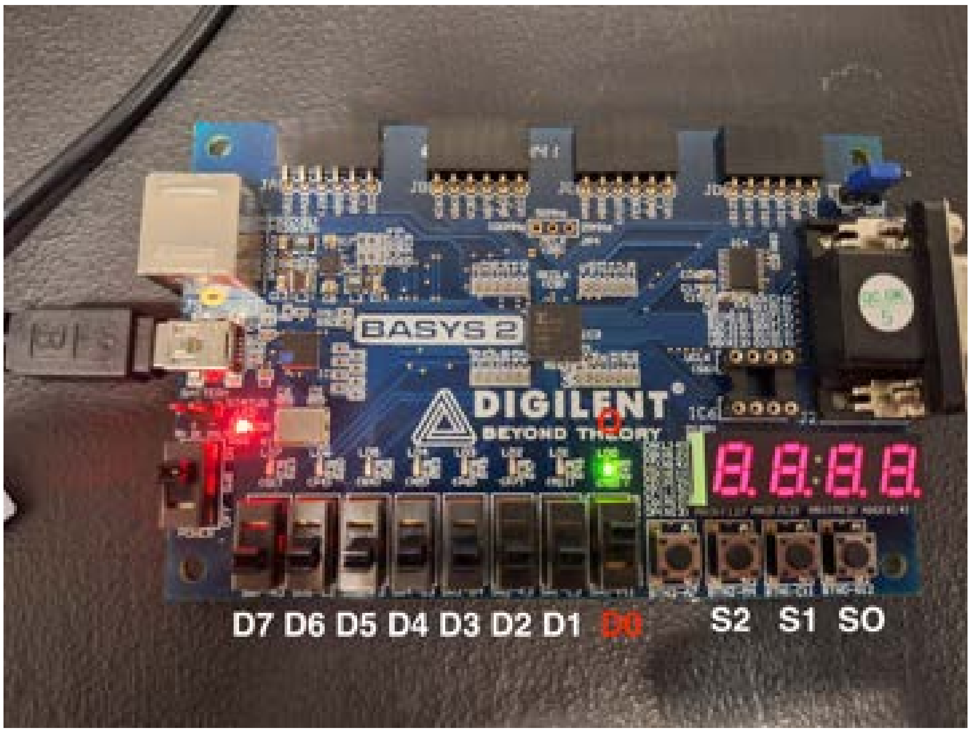


Figure 24: D0 is on - expected 1 ✓

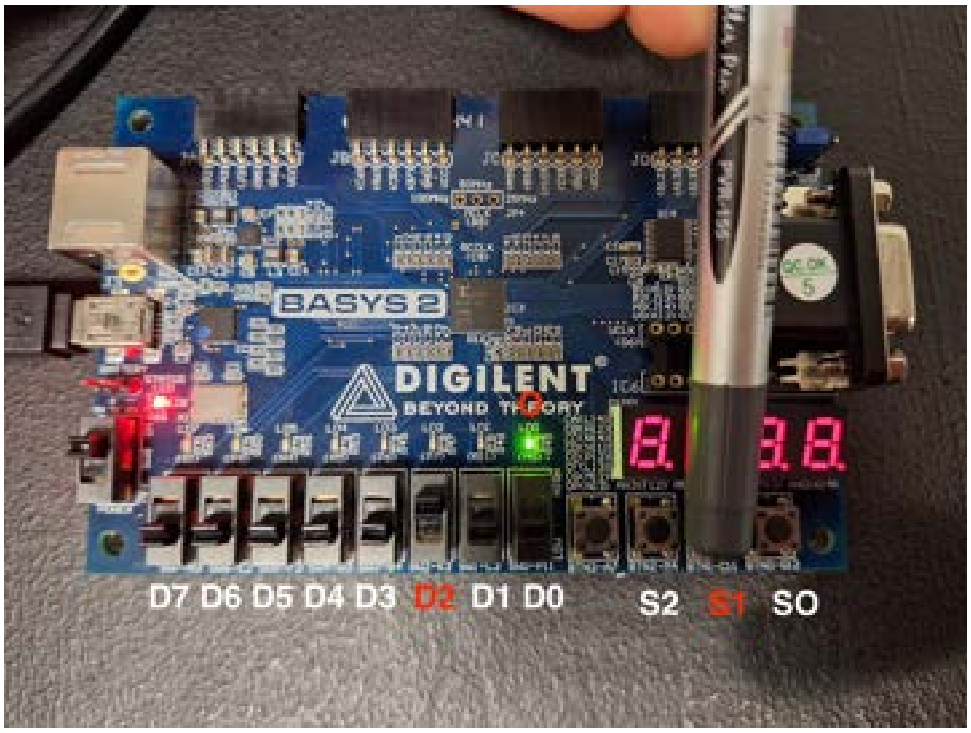


Figure 25: D2 is on, S1 is pressed - expected 1 ✓

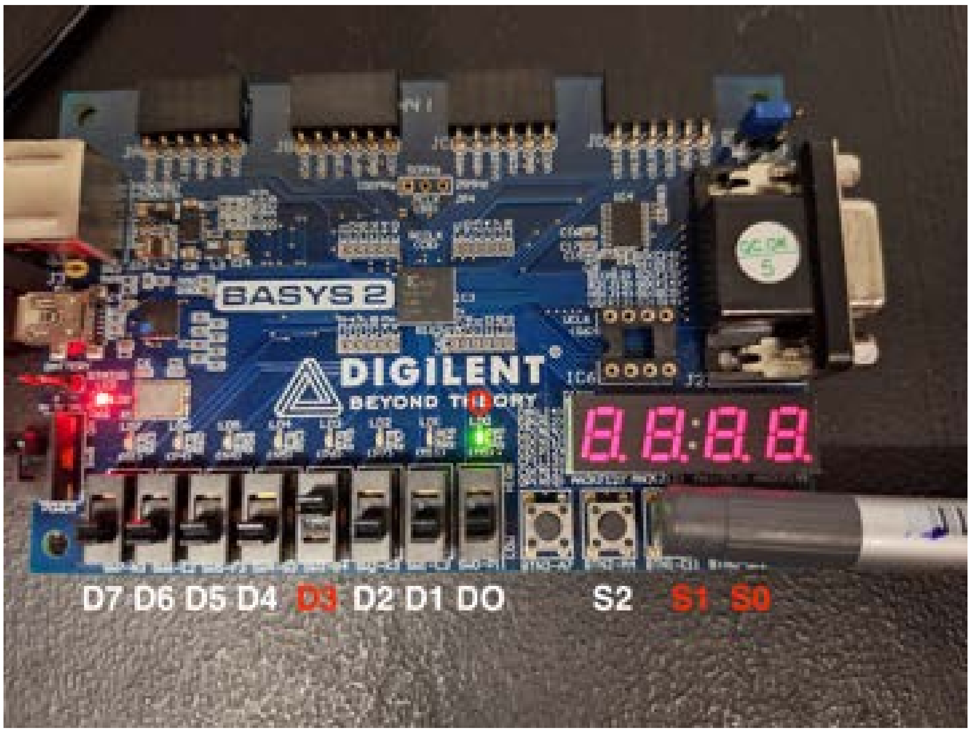


Figure 26: D3 is on, S1 and S0 is pressed - expected 1 ✓