Advanced Digital Logic

ENGR – UH 2310

Spring 2025

Instructor: Muhammad Hassan Jamil

Demarce Williams (dsw9740)
Aman Sunesh (as18181)

Task 1

4-Input Nand Gate

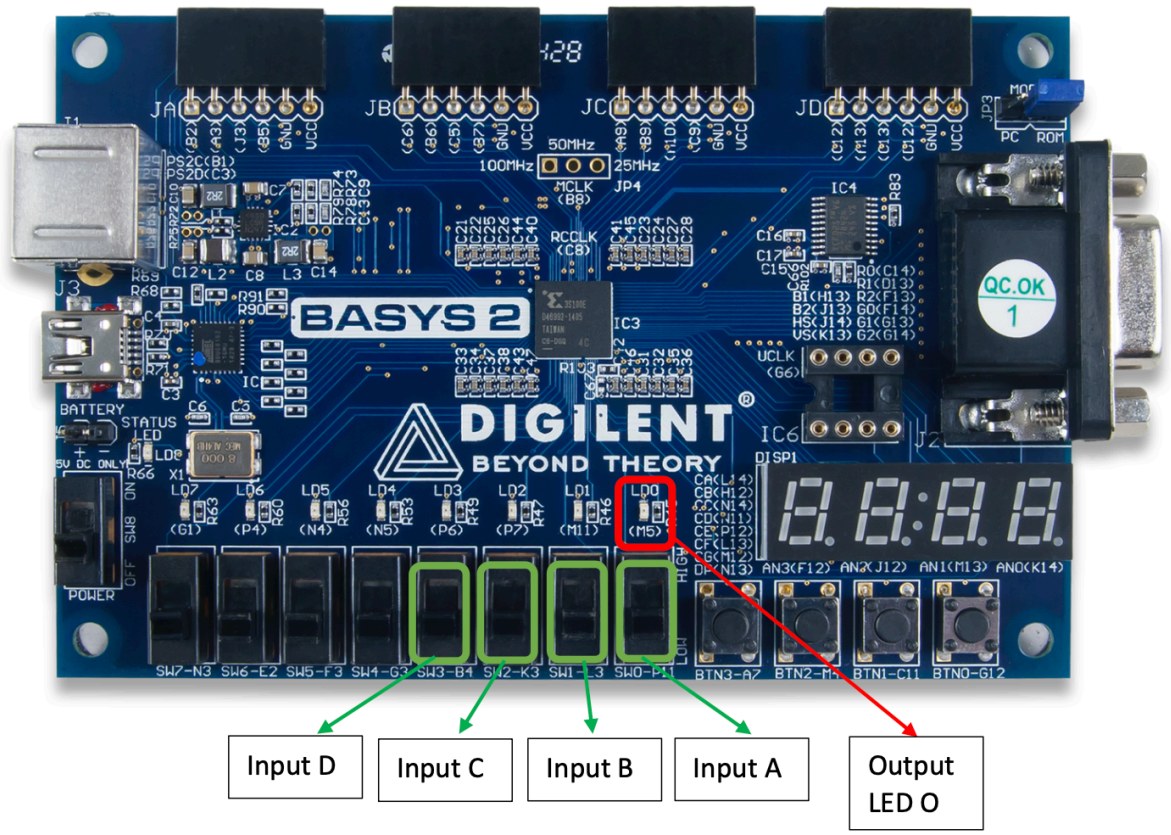
VHDL code

```
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    12:20:54 08/02/2025
-- Design Name:    4-Input NAND Gate Implementation
-- Module Name:    NAND - Behavioral
-- Project Name:   Advanced Digital Logic Task 1 - NAND Gate
-- Target Devices: FPGA
-- Tool versions:  Xilinx
-- Description:
-- This module implements a 4-input NAND gate using VHDL behavioral modeling.
-- The output (O) is the logical negation of the AND operation of four inputs (A, B, C, D).
-- This design can be tested on simulation tools and further synthesized on FPGA devices.
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.

-- Revision:
-- Revision 0.01 - File Created

-- Additional Comments:
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration for 4-input NAND gate
entity NAND4 is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           C : in  STD_LOGIC;
           D : in  STD_LOGIC;
           O : out  STD_LOGIC);
end NAND4;

-- Architecture defining the behavior of the 4-input NAND gate
architecture Behavioral of NAND4 is

begin
   -- The output O is the negation of the AND operation on inputs A, B, C, and D
   O <= not(A and B and C and D);

end Behavioral;


```

Constraint File Implementation:

```
NET "A" LOC = "P11";
NET "B" LOC = "L3";
NET "C" LOC = "K3";
NET "D" LOC = "B4";

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

FPGA Board


BASYS 2
DIGILENT
BEYOND THEORY
Input D
Input C
Input B
Input A
Output
LED O

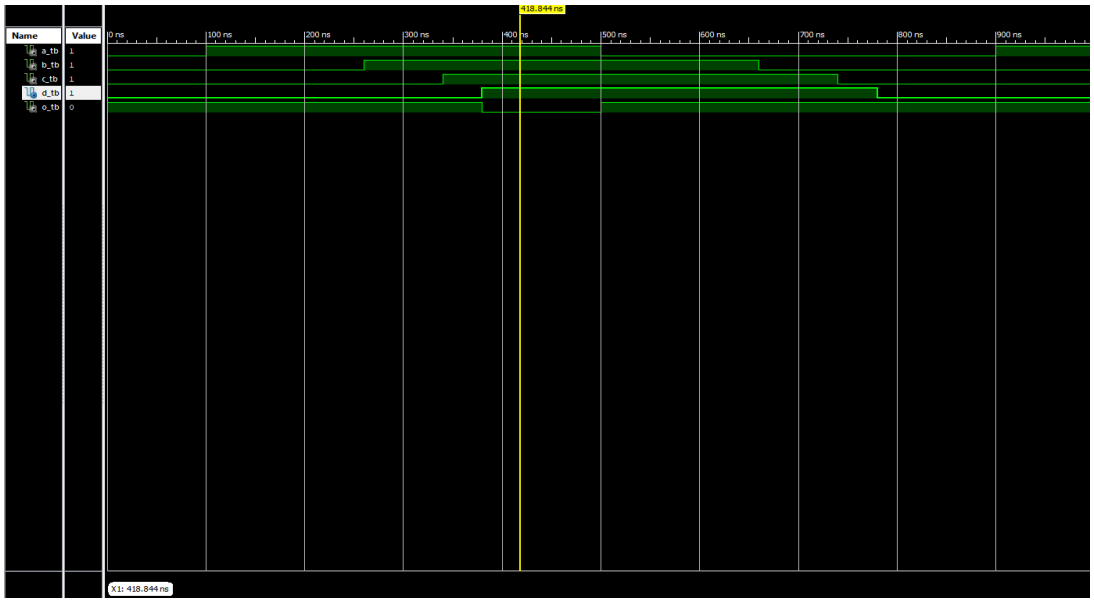
Testbench:

```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    12:22:43 08/02/2025
-- Design Name:    NAND4 Testbench
-- Module Name:    D:/NAND4/NAND_TB.vhd
-- Project Name:   Advanced Digital Logic Task 1 - NAND
-- Target Device:  FPGA
-- Tool versions:  Xilinx
-- Description:
-- VHDL Test Bench created to test the functionality of the 4-input NAND gate (NAND4).
-- The testbench verifies the output (O_tb) based on different input combinations of A_tb, B_tb, C_tb, and D_tb.
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
----------------------------------------------------------------------------------
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 declaration for the testbench
ENTITY NAND_TB IS
END NAND_TB;

-- Architecture defining the behavior of the testbench
ARCHITECTURE behavior OF NAND_TB IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT NAND4
    PORT(
         A : IN  std_logic;  -- First input of the NAND gate
         B : IN  std_logic;  -- Second input of the NAND gate
         C : IN  std_logic;  -- Third input of the NAND gate
         D : IN  std_logic;  -- Fourth input of the NAND gate
         O : OUT std_logic   -- Output of the NAND gate
        );
    END COMPONENT;


    -- Input signals to test the NAND4 gate
    signal A_tb : std_logic := '0';  -- Test signal for input A
    signal B_tb : std_logic := '0';  -- Test signal for input B
    signal C_tb : std_logic := '0';  -- Test signal for input C
    signal D_tb : std_logic := '0';  -- Test signal for input D

    -- Output signal to capture the output of the NAND4 gate
    signal O_tb : std_logic;
```

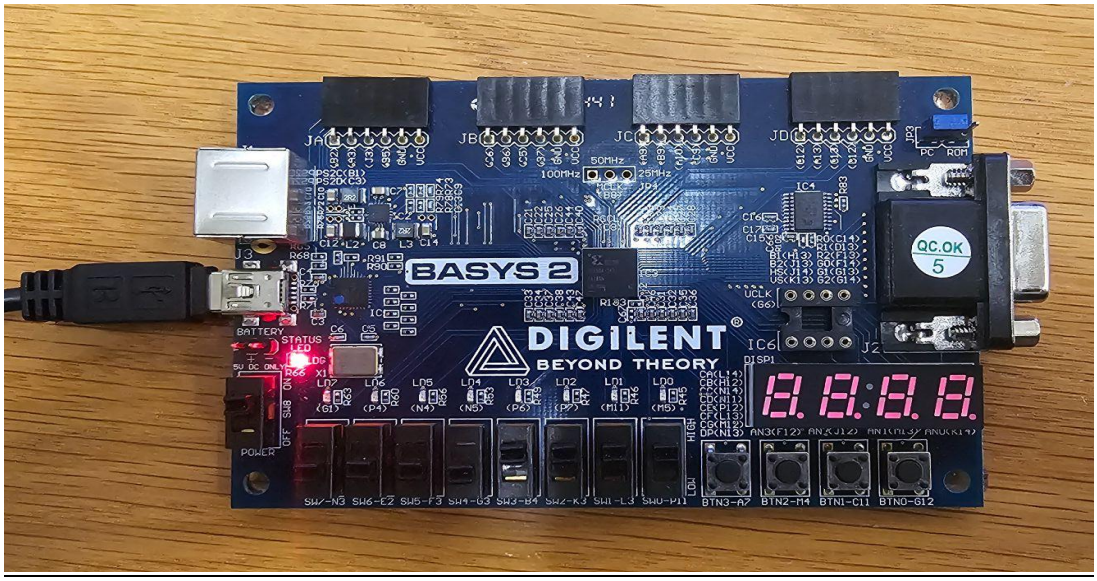
```vhdl
BEGIN

    -- Instantiate the Unit Under Test (UUT) and connect it to test signals
    uut: NAND4 PORT MAP (
         A => A_tb,
         B => B_tb,
         C => C_tb,
         D => D_tb,
         O => O_tb
    );

    -- Stimulus process: apply input combinations and observe the output
    stim_proc: process
    begin
         -- Hold the initial state for 100 ns
         wait for 100 ns;

         -- Toggle A_tb and wait 160 ns
         A_tb <= not A_tb;
         wait for 160 ns;

         -- Toggle B_tb and wait 80 ns
         B_tb <= not B_tb;
         wait for 80 ns;

         -- Toggle C_tb and wait 40 ns
         C_tb <= not C_tb;
         wait for 40 ns;

         -- Toggle D_tb and wait 20 ns
         D_tb <= not D_tb;
         wait for 20 ns;

         --wait;
    end process;

END behavior;
```

Test Case 1

Input Values: A = 1, B = 1, C = 1, D = 1
Output Value: O = 0 (as expected)
Simulation Time: 418 ns
Observation: The output line is low while the input lines are high, confirming the expected behavior of the NAND operation.

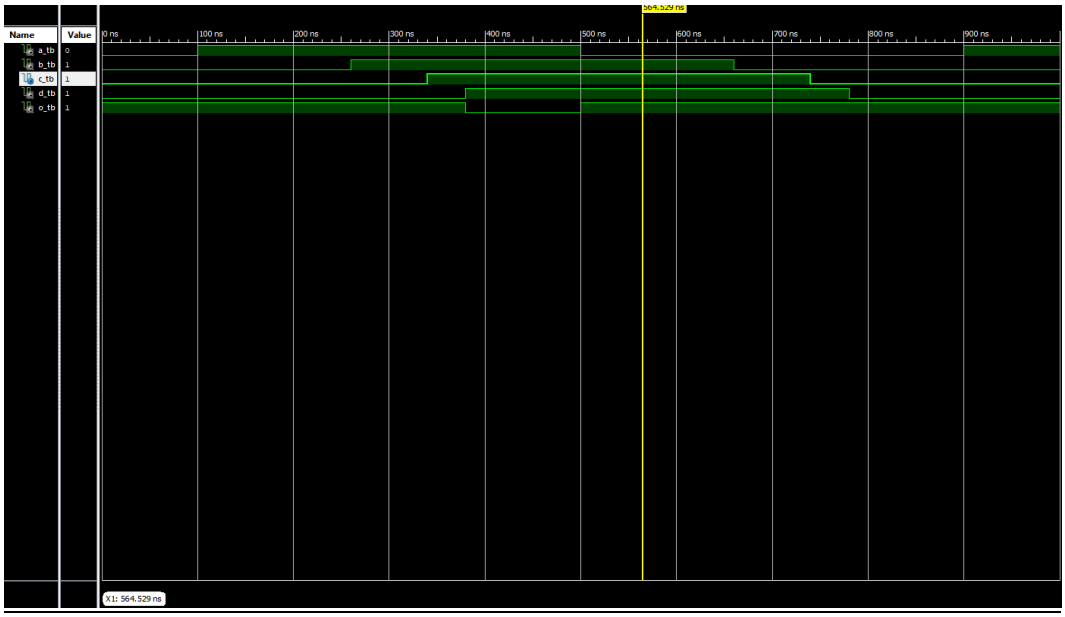


The board also exhibits a low output, as evidenced by the LED being in an off.

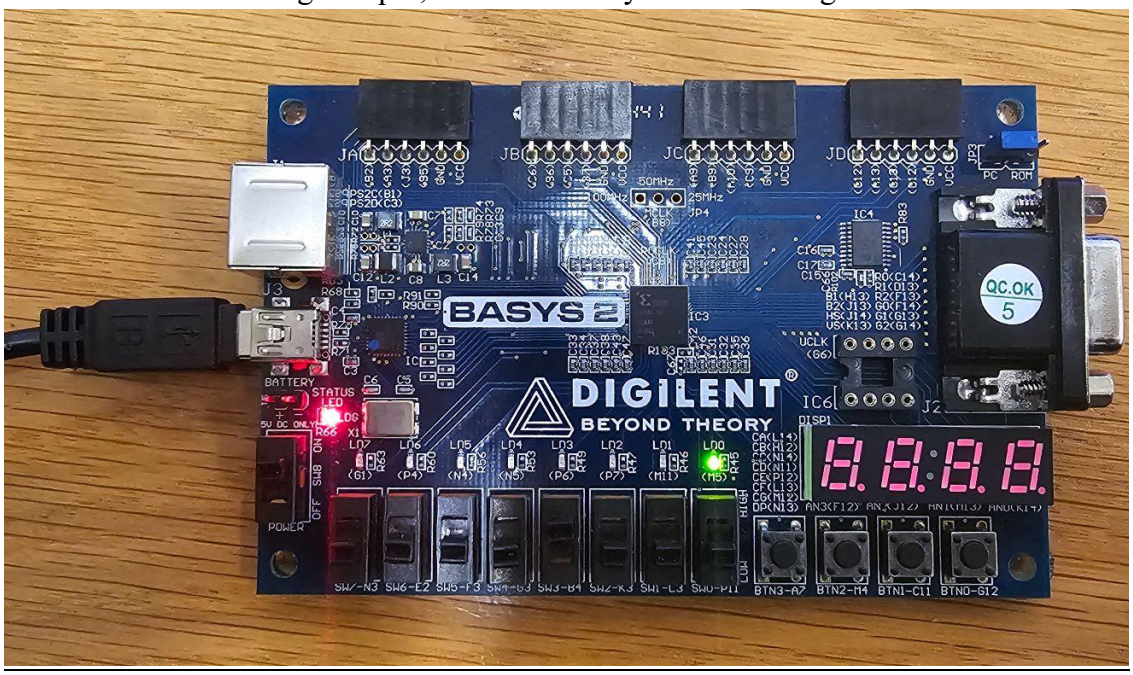


Test Case 2

Input Values: A = 0, B = 1, C = 1, D = 1
Output Value: O = 1 (as expected)
Simulation Time: 564 ns
Observation: The output line is high while the input line A is low and all other input lines are high, confirming the expected behavior of the NAND operation.

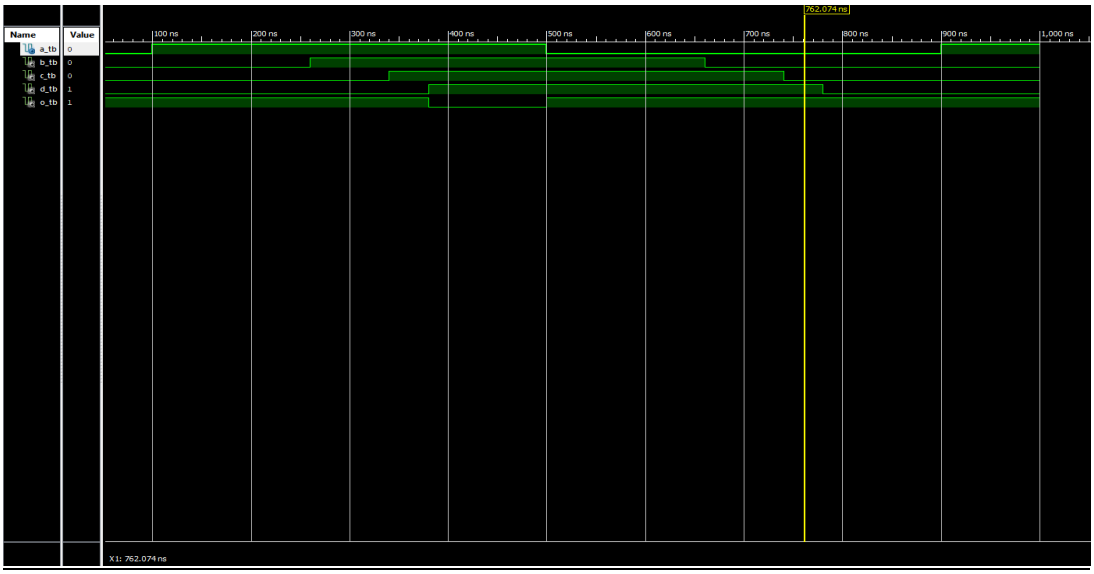


The board also exhibits a high output, as evidenced by the LED being in an on.

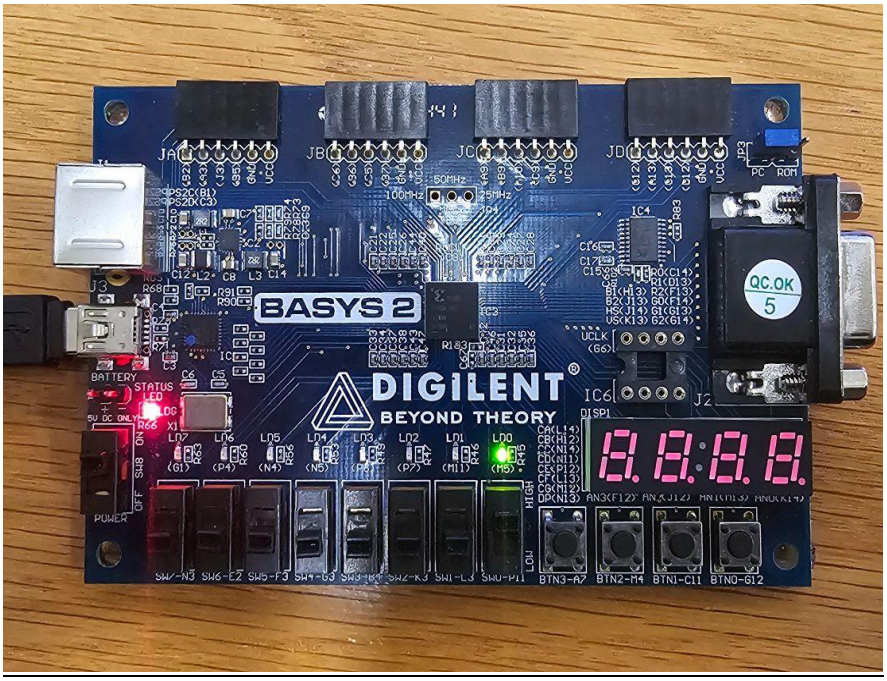


Test Case 3

Input Values: A = 0, B = 0, C = 0, D = 1
Output Value: O = 1 (as expected)
Simulation Time: 762 ns
Observation: The output line is high while the input line D is high and all other input lines are low, confirming the expected behavior of the NAND operation.



The board also exhibits a high output, as evidenced by the LED being on.



Task 2

Half-Adder

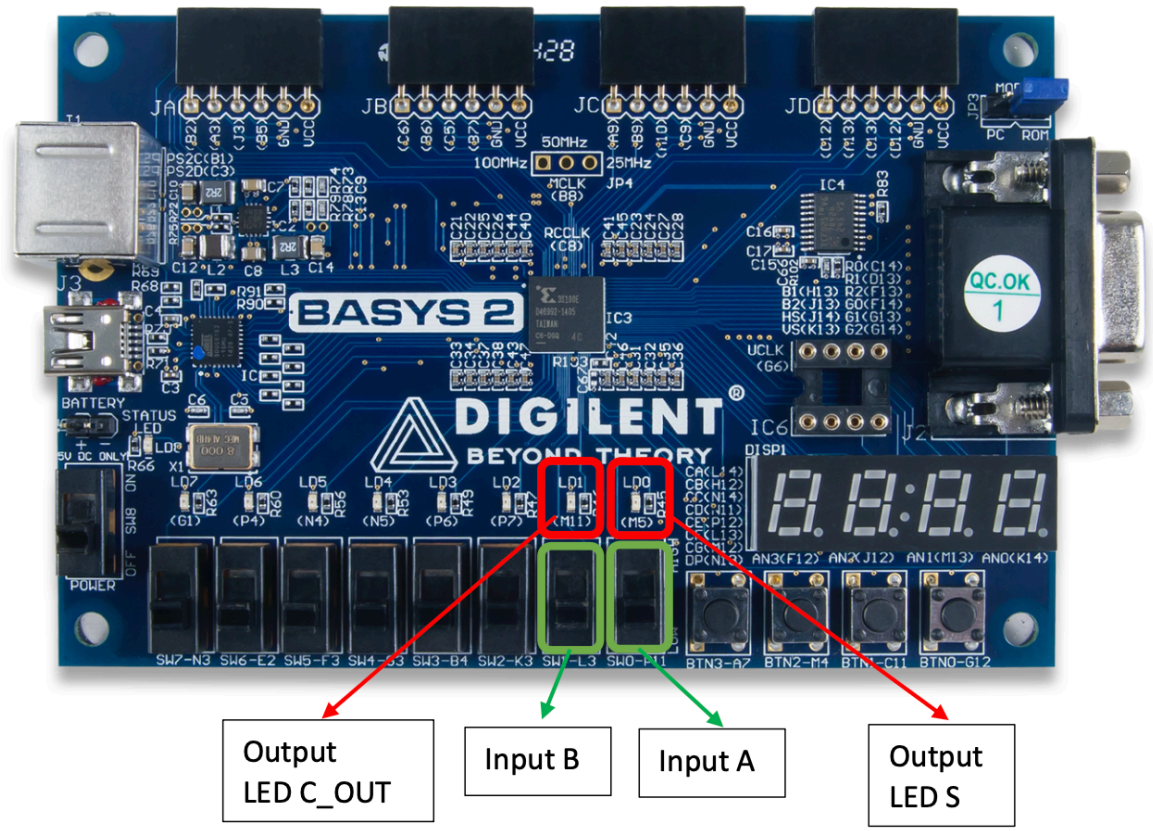
VHDL code

```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    12:35:14 08/02/2025
-- Design Name:    Half Adder Implementation
-- Module Name:    HalfAdder - Behavioral
-- Project Name:   Advanced Digital Logic Task 2 - Half Adder
-- Target Devices: FPGA
-- Tool versions:  Xilinx
-- Description:
-- This VHDL module implements a Half Adder using behavioral modeling.
-- A Half Adder performs the addition of two single-bit binary numbers (A and B)
-- and produces a sum output (Sum) and a carry-out (Carry).
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
--
-- Revision:
-- Revision 0.01 - File Created
--
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


-- Entity declaration for the Half Adder
entity HalfAdder is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           S : out  STD_LOGIC;
           C_OUT : out  STD_LOGIC);
end HalfAdder;


-- Architecture defining the behavior of the Half Adder
architecture Behavioral of HalfAdder is

begin
   -- Sum is the result of XOR operation on A and B
   S <= A xor B;

   -- Carry is the result of AND operation on A and B
   C_OUT <= A and B;

end Behavioral;
```

Constraint File Implementation:

```
NET "A" LOC = "L3";
NET "B" LOC = "P11";

NET "S" LOC = "M5";
NET "C_OUT" LOC = "M11";
```

FPGA Board


BASYS 2
DIGILENT
BEYOND THEORY
Output
LED C_OUT
Input B
Input A
Output
LED S

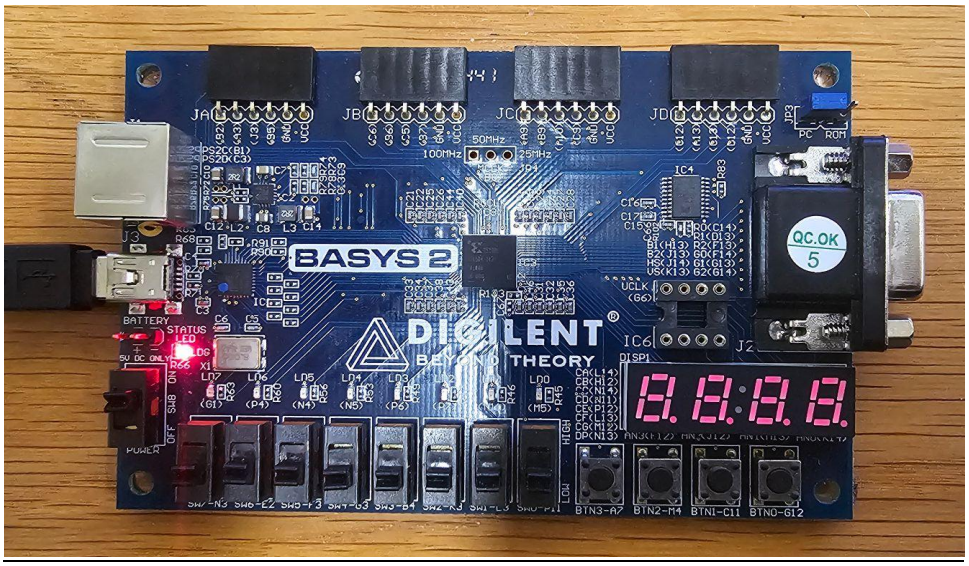
Testbench:

```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    12:36:54 08/02/2025
-- Design Name:    Half Adder Testbench
-- Module Name:    HalfAdder_TB - Behavioral
-- Project Name:   Advanced Digital Logic Task 2 - Half Adder
-- Target Devices: FPGA
-- Tool versions:  Xilinx
-- Description:
-- VHDL Test Bench created to test the functionality of the Half Adder (HalfAdder).
-- The testbench verifies the sum (S) and carry-out (C_OUT) based on different input
-- combinations of A and B.
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
--
-- Revision:
-- Revision 0.01 - File Created
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration for the testbench
entity HalfAdder_TB is
end HalfAdder_TB;

architecture Behavioral of HalfAdder_TB is

    -- Component Declaration for the Unit Under Test (UUT)
    component HalfAdder
        port(
            A : in  std_logic;
            B : in  std_logic;
            S : out std_logic;
            C_OUT : out std_logic
        );
    end component;

    -- Signals to provide test inputs and capture the outputs
    signal A_tb : std_logic := '0';  -- Test signal for input A
    signal B_tb : std_logic := '0';  -- Test signal for input B
    signal S_tb : std_logic;         -- Output signal for Sum
    signal C_OUT_tb : std_logic;     -- Output signal for Carry-Out

begin

    -- Instantiate the Unit Under Test (UUT)
    UUT: HalfAdder port map (
        A => A_tb,
        B => B_tb,
        S => S_tb,
        C_OUT => C_OUT_tb
    );

    -- Stimulus process: Apply various input combinations and monitor outputs
    stim_proc: process
    begin
        -- Hold the initial state for 100 ns
        wait for 100 ns;

        -- Apply input combination 00 and wait for 160 ns
        A_tb <= '0'; B_tb <= '0';
        wait for 160 ns;

        -- Apply input combination 01 and wait for 80 ns
        A_tb <= '0'; B_tb <= '1';
        wait for 80 ns;

        -- Apply input combination 10 and wait for 40 ns
        A_tb <= '1'; B_tb <= '0';
        wait for 40 ns;

        -- Apply input combination 11 and wait for 20 ns
        A_tb <= '1'; B_tb <= '1';
        wait for 20 ns;

        --wait;
    end process;
end Behavioral;
```

Test Case 1

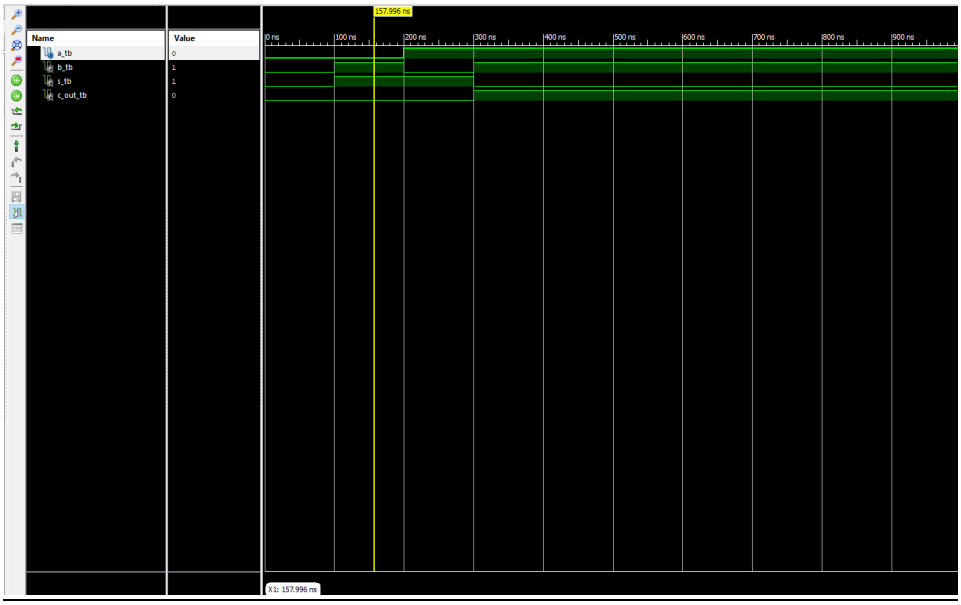
Input Values: A = 0, B = 0
Output Value: S = 0, C_OUT=0 (as expected)
Simulation Time: 39 ns
Observation: Both the sum and carry-out output lines are low while the input lines are all low, confirming the expected behavior of the Half-Adder operation.

The board also exhibit low outputs, as evidenced by the LEDs being off.

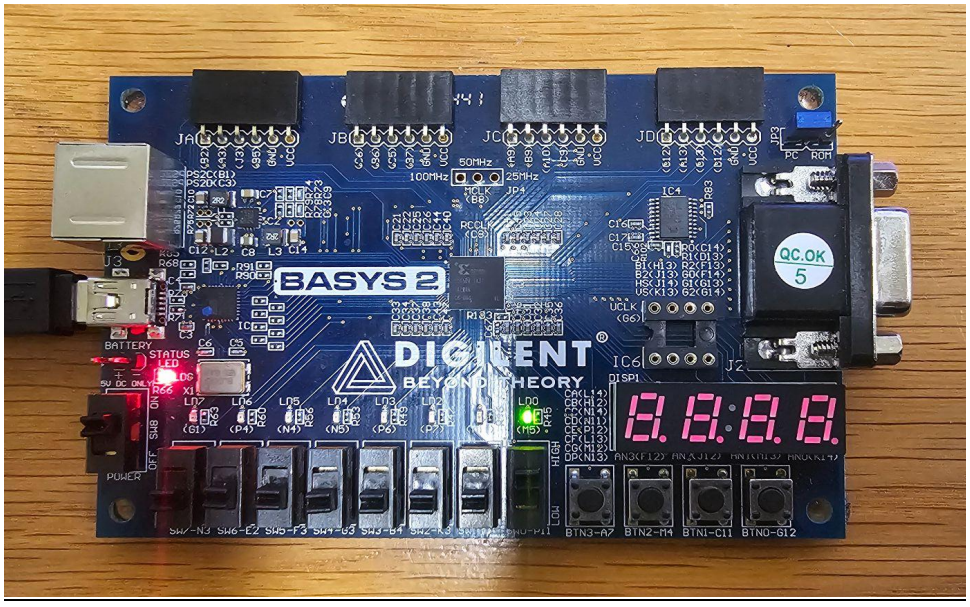


Test Case 2

Input Values: A = 0, B = 1
Output Value: S = 1, C_OUT=0 (as expected)
Simulation Time: 158 ns
Observation: The sum output line is high and the carry-out output line is low while the input line A is low and the line B is high, confirming the expected behavior of the Half-Adder operation.

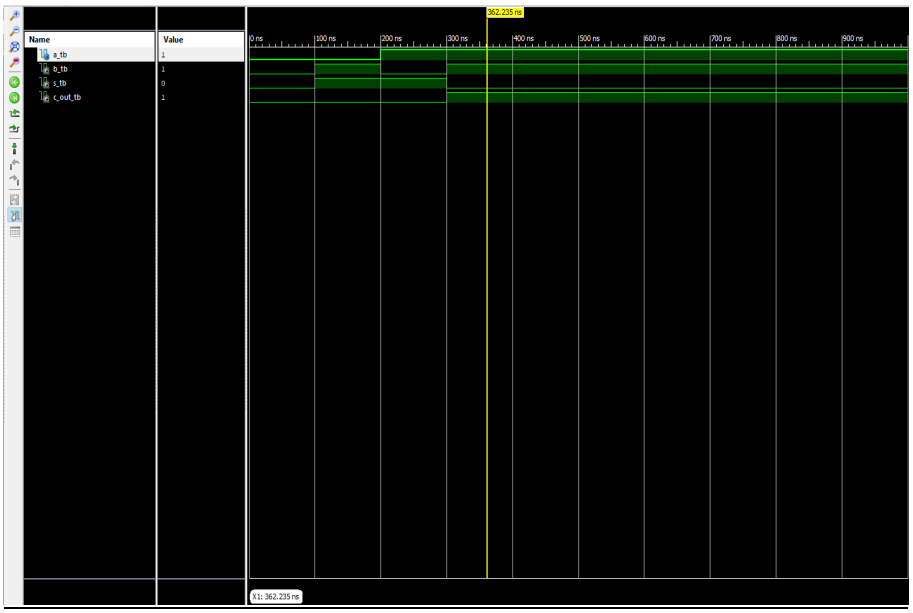


The board also exhibit low output for carry-out(LED M11), and high output for the sum (LED M5).

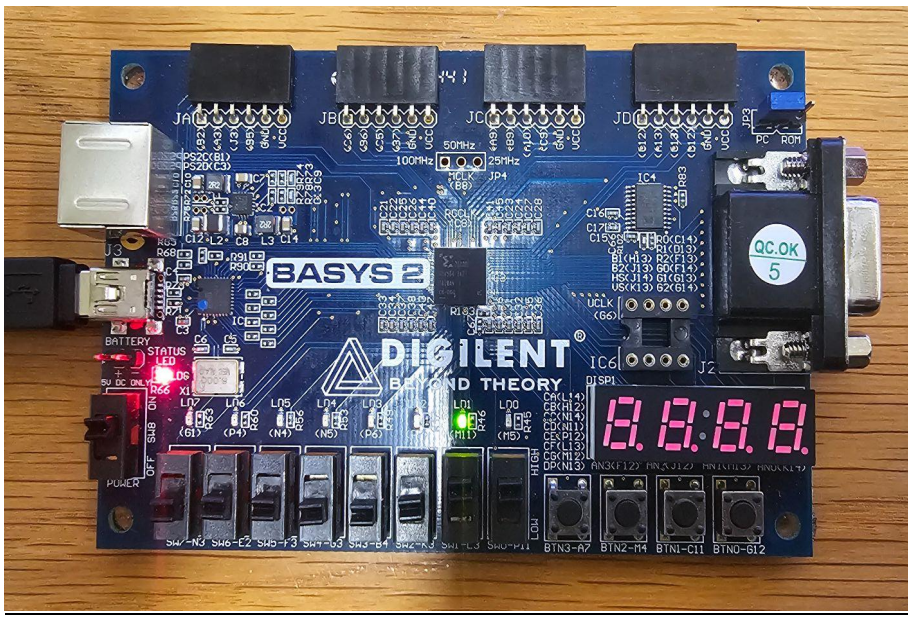


Test Case 3

Input Values: A = 1, B = 1
Output Value: S = 0, C_OUT=1 (as expected)
Simulation Time: 362 ns
Observation: The sum output line is low and the carry-out output line is high while all the input lines are high, confirming the expected behavior of the Half-Adder operation.



The board also exhibit low output for sum-out (LED M5), and high output for the carry-out (LED M11).



Task 3

4-Bit Binary Adder

VHDL code

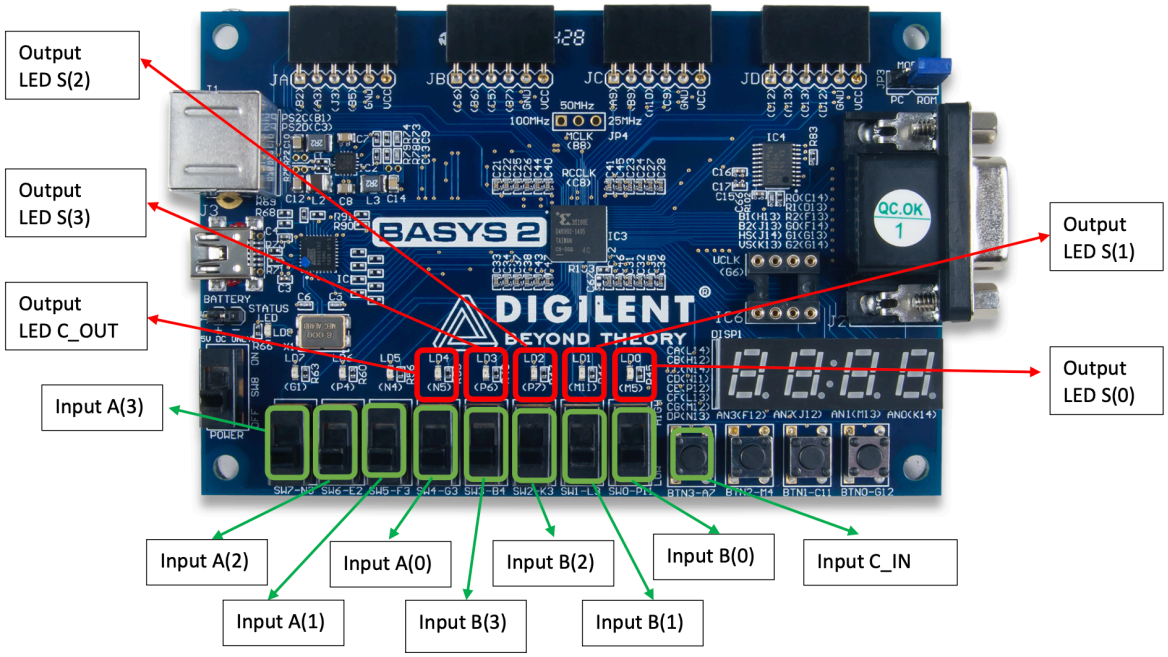
```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    12:43:03 08/02/2025
-- Design Name:    4-Bit Binary Adder Implementation
-- Module Name:    FourBitBinaryAdder - Behavioral
-- Project Name:   Advanced Digital Logic Task 3 - Four Bit Binary Adder
-- Target Devices: FPGA
-- Tool versions:  Xilinx
-- Description:
-- This VHDL module implements a 4-bit binary adder using behavioral modeling.
-- It adds two 4-bit binary numbers (A and B) and an input carry (C_IN) to produce
-- a 4-bit sum (S) and an output carry (C_OUT).
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
--
-- Revision:
-- Revision 0.01 - File Created
--
-- Additional Comments:
--
----------------------------------------------------------------------------------

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;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

-- Entity declaration for the 4-bit binary adder
entity FourBitBinaryAdder is
    Port ( A : in  STD_LOGIC_VECTOR(3 downto 0);   -- First 4-bit binary input
           B : in  STD_LOGIC_VECTOR(3 downto 0);   -- Second 4-bit binary input
           C_IN : in  STD_LOGIC;                   -- Carry-in input
           S : out  STD_LOGIC_VECTOR(3 downto 0);  -- 4-bit sum output
           C_OUT : out  STD_LOGIC);                -- Carry-out output
end FourBitBinaryAdder;


-- Architecture defining the behavior of the 4-bit binary adder
architecture Behavioral of FourBitBinaryAdder is
```

```vhdl
begin
    -- Process block to compute the sum and carry
    process(A, B, C_IN)
        variable temp_carry : STD_LOGIC;  -- Variable to store the carry for each bit position
    begin
        -- Initialize the carry with the input carry
        temp_carry := C_IN;

        -- Loop through each bit position (from 0 to 3)
        for i in 0 to 3 loop
            -- Sum for bit i using XOR operation for A(i), B(i), and temp_carry
            S(i) <= A(i) xor B(i) xor temp_carry;

            -- Compute the carry for the next bit position
            -- Carry is generated if both A(i) and B(i) are 1 (A(i) and B(i))
            -- or if temp_carry is 1 and one of A(i) or B(i) is 1 ((A(i) xor B(i)) and temp_carry)
            temp_carry := (A(i) and B(i)) or ((A(i) xor B(i)) and temp_carry);
        end loop;

        -- Assign the final carry-out value
        C_OUT <= temp_carry;
    end process;

end Behavioral;
```

Constraint File Implementation:

```
# Specify the I/O standard to match the external device voltage levels.
# This ensures proper signal integrity, compatibility, and power optimization.

NET "A(3)" LOC = "N3" | IOSTANDARD = LVCMOS33;
NET "A(2)" LOC = "E2" | IOSTANDARD = LVCMOS33;
NET "A(1)" LOC = "F3" | IOSTANDARD = LVCMOS33;
NET "A(0)" LOC = "G3" | IOSTANDARD = LVCMOS33;

NET "B(3)" LOC = "B4" | IOSTANDARD = LVCMOS33;
NET "B(2)" LOC = "K3" | IOSTANDARD = LVCMOS33;
NET "B(1)" LOC = "L3" | IOSTANDARD = LVCMOS33;
NET "B(0)" LOC = "P11" | IOSTANDARD = LVCMOS33;

NET "C_IN" LOC = "A7" | IOSTANDARD = LVCMOS33;

NET "S(3)" LOC = "P6" | IOSTANDARD = LVCMOS33;
NET "S(2)" LOC = "P7" | IOSTANDARD = LVCMOS33;
NET "S(1)" LOC = "M11" | IOSTANDARD = LVCMOS33;
NET "S(0)" LOC = "M5" | IOSTANDARD = LVCMOS33;

NET "C_OUT" LOC = "N5" | IOSTANDARD = LVCMOS33;
```

FPGA Board


Output
LED S(2)
Output
LED S(3)
Output
LED C_OUT
Input A(3)
Output
LED S(1)
Output
LED S(0)
Input A(2)
Input A(0)
Input B(2)
Input B(0)
Input C_IN
Input A(1)
Input B(3)
Input B(1)
BASYS 2
DIGILENT
BEYOND THEORY

Testbench:

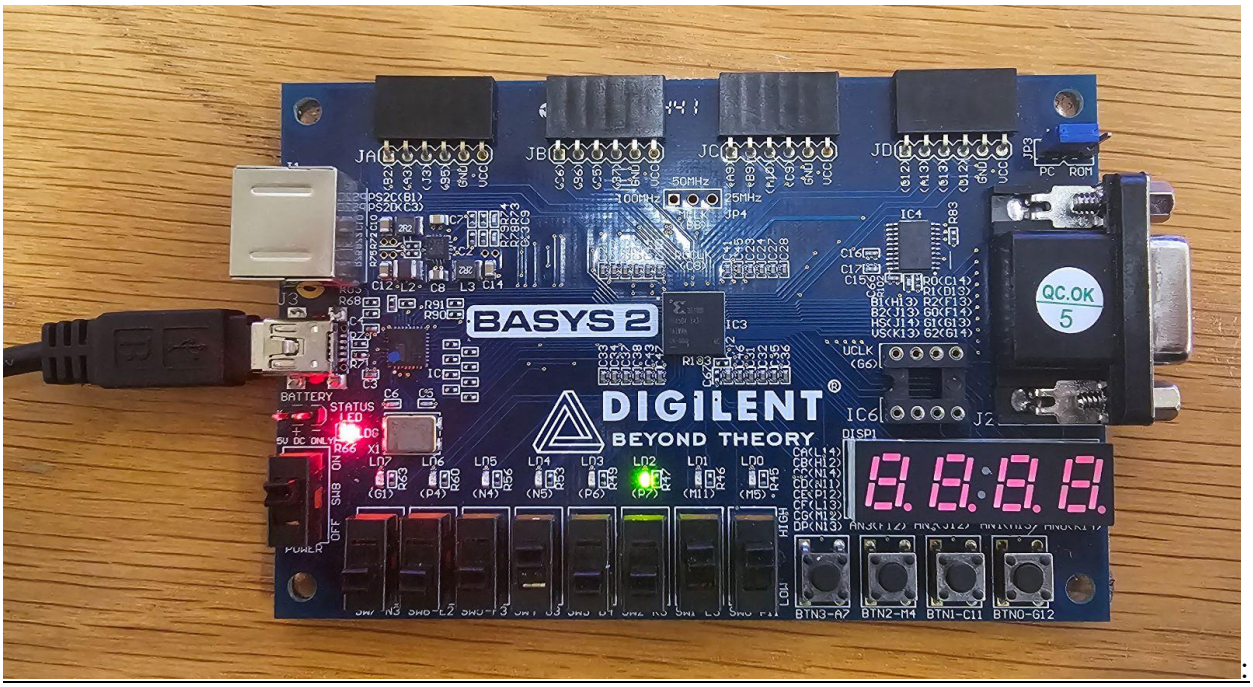
```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    12:48:43 08/02/2025
-- Design Name:    4-Bit Binary Adder Testbench
-- Module Name:    FourBitBinaryAdder_TB - Behavioral
-- Project Name:   Advanced Digital Logic Task 3 - Four Bit Binary Adder
-- Target Device:  FPGA
-- Tool versions:  Xilinx
-- Description:
-- VHDL Test Bench created to test the functionality of the 4-bit binary adder (FourBitBinaryAdder).
-- The testbench verifies the output (S_tb) and carry-out (C_OUT_tb) based on different input combinations of A_tb, B_tb, and C_IN_tb.
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
----------------------------------------------------------------------------------

-- Import standard IEEE library for logic operations
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using arithmetic functions
--USE ieee.numeric_std.ALL;

-- Entity declaration for the testbench
ENTITY FourBitBinaryAdder_TB IS
END FourBitBinaryAdder_TB;

-- Architecture defining the behavior of the testbench
ARCHITECTURE behavior OF FourBitBinaryAdder_TB IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT FourBitBinaryAdder
    PORT(
         A : IN  std_logic_vector(3 downto 0);  -- First 4-bit binary input
         B : IN  std_logic_vector(3 downto 0);  -- Second 4-bit binary input
         C_IN : IN  std_logic;                  -- Carry-in input
         S : OUT std_logic_vector(3 downto 0);  -- 4-bit sum output
         C_OUT : OUT std_logic                  -- Carry-out output
    );
    END COMPONENT;

    -- Input signals to test the FourBitBinaryAdder
    signal A_tb : std_logic_vector(3 downto 0) := "0000";  -- Test signal for input A
    signal B_tb : std_logic_vector(3 downto 0) := "0000";  -- Test signal for input B
    signal C_IN_tb : std_logic := '0';                     -- Test signal for carry-in

    -- Output signals to capture the results of the addition
    signal S_tb : std_logic_vector(3 downto 0);  -- Signal to store the sum output
    signal C_OUT_tb : std_logic;                 -- Signal to store the carry-out
```

```vhdl

BEGIN

    -- Instantiate the Unit Under Test (UUT) and connect it to the test signals
    UUT: FourBitBinaryAdder PORT MAP (
        A => A_tb,
        B => B_tb,
        C_IN => C_IN_tb,
        S => S_tb,
        C_OUT => C_OUT_tb
    );

    -- Stimulus process: apply input combinations and observe the output
    stim_proc: process
    begin
        -- Hold the initial state for 100 ns
        wait for 100 ns;

        -- Test case 1: A = 0001, B = 0011, C_IN = 0 (expect S = 0100, C_OUT = 0)
        A_tb <= "0001"; B_tb <= "0011"; C_IN_tb <= '0';
        wait for 160 ns;

        -- Test case 2: A = 1010, B = 0101, C_IN = 1 (expect S = 1110, C_OUT = 0)
        A_tb <= "1010"; B_tb <= "0101"; C_IN_tb <= '1';
        wait for 80 ns;

        -- Test case 3: A = 1111, B = 1111, C_IN = 0 (expect S = 1110, C_OUT = 1)
        A_tb <= "1111"; B_tb <= "1111"; C_IN_tb <= '0';
        wait for 40 ns;

        -- Test case 4: A = 0110, B = 0011, C_IN = 1 (expect S = 1010, C_OUT = 0)
        A_tb <= "0110"; B_tb <= "0011"; C_IN_tb <= '1';
        wait for 20 ns;

        --wait;
    end process;

END behavior;
```

Test Case 1

Input Values: A = 0001, B = 0011
Output Value: S = 0100, C_OUT=0 (as expected)
Simulation Time: 564 ns
Observation: The sum output line shows binary number 4 and the carry-out output line low while the binary numbers 1 and 3 are submitted at the A and B input lines, respectively, confirming the expected behavior of the 4-bit Binary Adder operation.

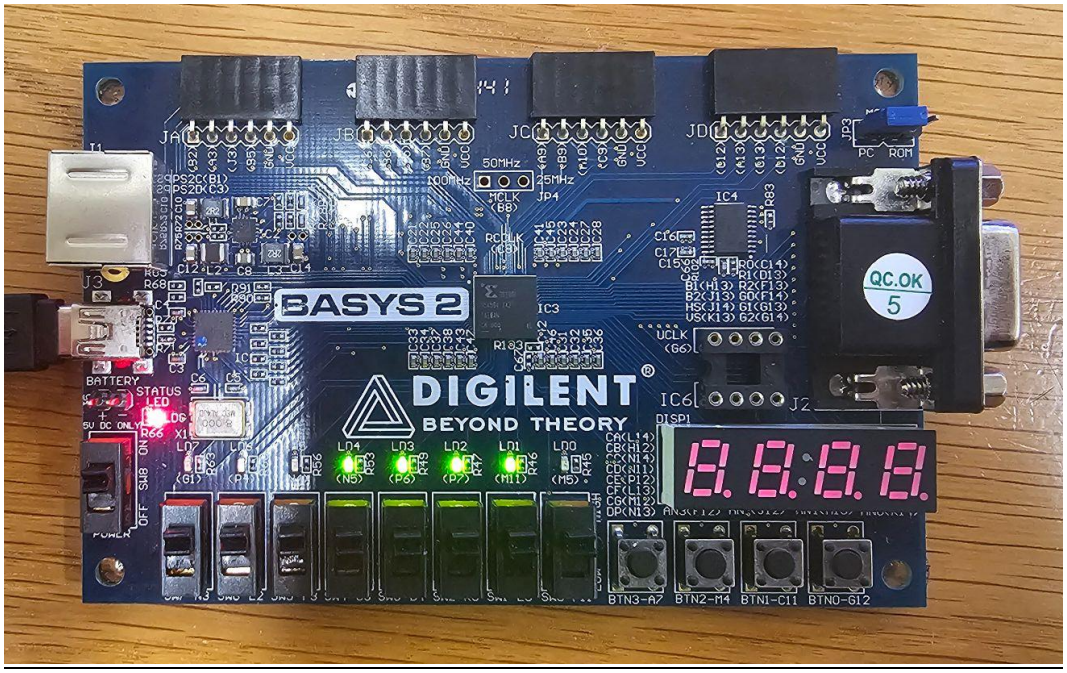
The board also exhibit an off LED output for carry-out (N5) and a sum output of off, on, off, off LEDS (P6, P7, M11, M5).



Test Case 2

Input Values: A = 1111, B = 1111
Output Value: S = 1110, C_OUT=1(as expected)
Simulation Time: 360 ns
Observation: The sum output line shows binary number 14 and the carry-out output line high while the binary number 15 is submitted at both the A and B input lines, confirming the expected behavior of the 4-bit Binary Adder operation.

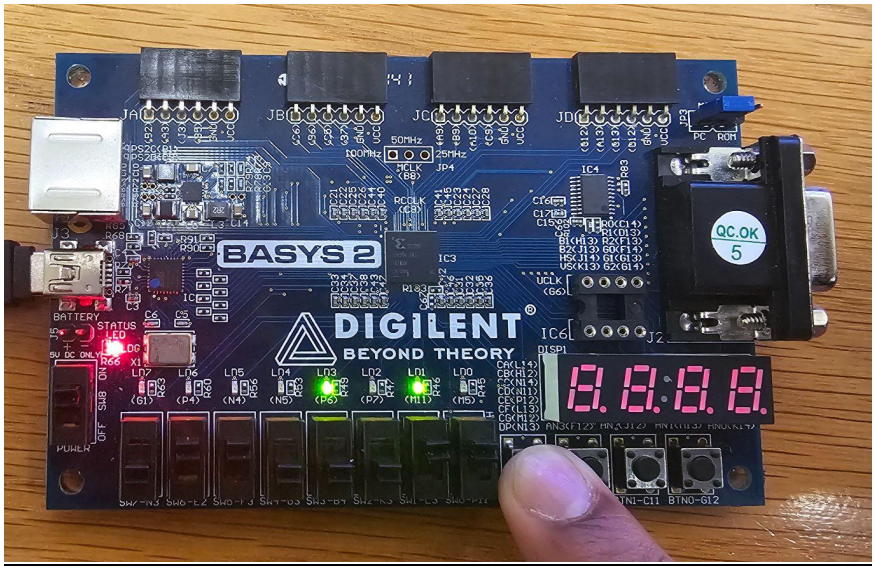
The board also exhibit on LED output for carry-out (N5), and a sum output of 3 on LEDs followed by an off (P6, P7, M11, M5).



Test Case 3

Input Values: A = 0110, B = 0011
Output Value: S = 1010, C_OUT=0 (as expected)
Simulation Time: 360 ns
Observation: The sum output line shows binary number 14 and the carry-out output line high while the binary number 15 is submitted at both the A and B input lines, confirming the expected behavior of the 4-bit Binary Adder operation.

The board also exhibit off LED output for carry-out (N5), and a sum output of on, off, on and off LEDs (P6, P7, M11, M5).



Task 4

Minterms Function

VHDL code for Concurrent Signals

```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    12:52:53 10/02/2025
-- Design Name:    Minterm-Based Function Implementation
-- Module Name:    Minterms - Behavioral
-- Project Name:   Advanced Digital Logic Task 4 - Minterm Using When-Else
-- Target Devices: FPGA
-- Tool versions:  Xilinx
-- Description:
-- This VHDL module implements a function based on minterm selection.
-- It uses four individual STD_LOGIC inputs (A, B, C, D).
-- The output F is '1' for the minterms "0001", "0011", "1001", and "1011"
-- (assuming A is the MSB and D is the LSB), and '0' otherwise.
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
--
-- Revision:
-- Revision 0.01 - File Created
--
-- Additional Comments:
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


-- Entity declaration for the minterm-based function
entity Minterms is
    Port (
        A : in  STD_LOGIC;  -- MSB (bit 3)
        B : in  STD_LOGIC;  -- bit 2
        C : in  STD_LOGIC;  -- bit 1
        D : in  STD_LOGIC;  -- LSB (bit 0)
        F : out STD_LOGIC
    );
end Minterms;

-- Architecture defining the behavior of the minterm-based function
architecture Behavioral of Minterms is
begin

    -- Concurrent conditional signal assignment:
    -- The output F is assigned '1' if the input combination matches any of
    -- the specified minterms, otherwise F is assigned '0'.
    --
    -- Minterm "0001": A = '0', B = '0', C = '0', D = '1'
    -- Minterm "0011": A = '0', B = '0', C = '1', D = '1'
    -- Minterm "1001": A = '1', B = '0', C = '0', D = '1'
    -- Minterm "1011": A = '1', B = '0', C = '1', D = '1'

    F <= '1' when ((A = '0') and (B = '0') and (C = '0') and (D = '1')) else
         '1' when ((A = '0') and (B = '0') and (C = '1') and (D = '1')) else
         '1' when ((A = '1') and (B = '0') and (C = '0') and (D = '1')) else
         '1' when ((A = '1') and (B = '0') and (C = '1') and (D = '1')) else
         '0';

end Behavioral;
```

VHDL Code for Case-When

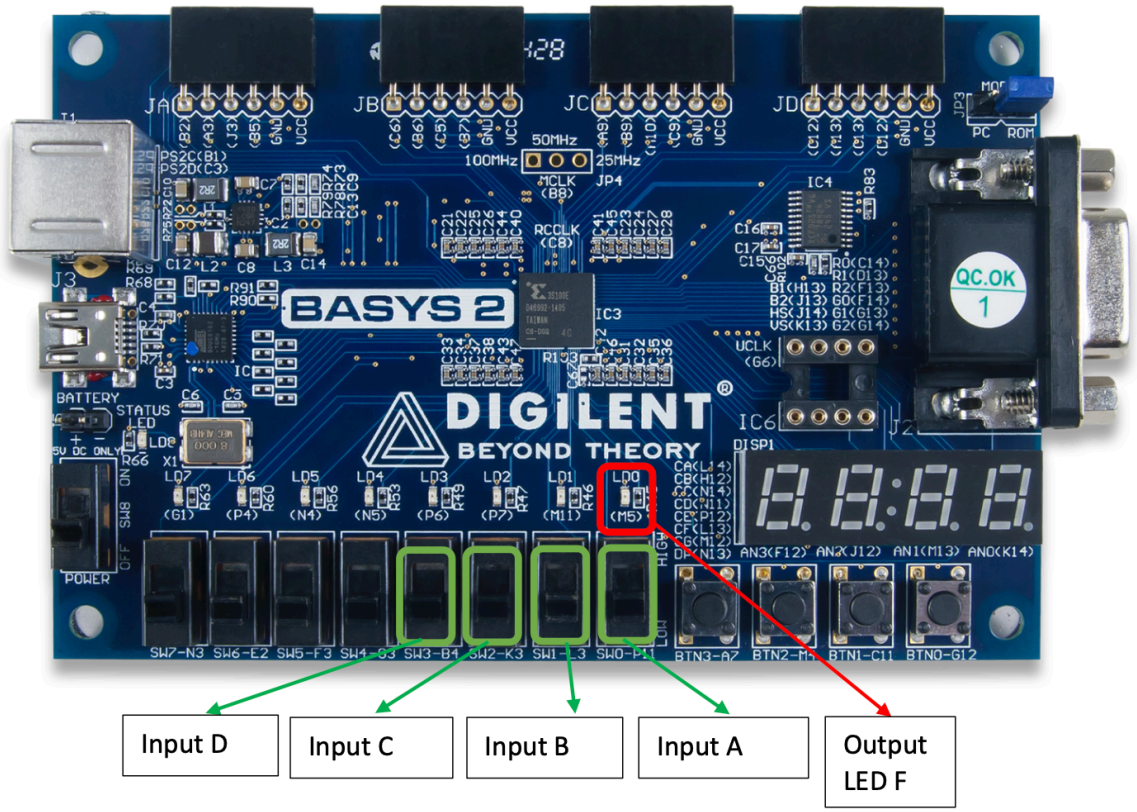
```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    12:52:53 10/02/2025
-- Design Name:    Minterm-Based Function Implementation
-- Module Name:    Minterms - Behavioral
-- Project Name:   Advanced Digital Logic Task 4 - Minterm Using Case-When
-- Target Devices: FPGA
-- Tool versions:  Xilinx
-- Description:
-- This VHDL module implements a function based on minterm selection.
-- It uses four individual STD_LOGIC inputs (A, B, C, D).
-- The output F is '1' for the minterms "0001", "0011", "1001", and "1011" (assuming
-- A is the MSB and D is the LSB), and '0' otherwise.
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
--
-- Revision:
-- Revision 0.01 - File Created
--
-- Additional Comments:
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration for the minterm-based function
entity Minterms is
    Port (
        A : in  STD_LOGIC;  -- MSB (bit 3)
        B : in  STD_LOGIC;  -- bit 2
        C : in  STD_LOGIC;  -- bit 1
        D : in  STD_LOGIC;  -- LSB (bit 0)
        F : out STD_LOGIC
    );
end Minterms;


-- Architecture defining the behavior of the minterm-based function
architecture Behavioral of Minterms is

-- Internal signal to group A,B,C,D into a 4-bit vector
signal Inputs : STD_LOGIC_VECTOR(3 downto 0);

begin

    -- Concatenate A,B,C,D to form a 4-bit vector, assuming A is MSB and D is LSB
    Inputs <= A & B & C & D;

    -- Process block to determine the output F based on the 4-bit value of Inputs
    process(Inputs)
    begin
        case Inputs is
            when "0001" =>  -- A=0, B=0, C=0, D=1   Minterm corresponding to binary 1
                F <= '1';
            when "0011" =>  -- A=0, B=0, C=1, D=1   Minterm corresponding to binary 3
                F <= '1';
            when "1001" =>  -- A=1, B=0, C=0, D=1   Minterm corresponding to binary 9
                F <= '1';
            when "1011" =>  -- A=1, B=0, C=1, D=1   Minterm corresponding to binary 11
                F <= '1';
            when others =>
                F <= '0';    -- Default case for inputs not matching the specfied minterms
        end case;
    end process;

end Behavioral;

```

Design Summary (Programming File Generated) | Minterm.vhd* | Testbench.vhd | Constraints.ucf

Constraint File Implementation:

```
# Specify the I/O standard to match the external device voltage levels.
# This ensures proper signal integrity, compatibility, and power optimization.

NET "A" LOC = "B4" | IOSTANDARD = LVCMOS33;
NET "B" LOC = "K3" | IOSTANDARD = LVCMOS33;
NET "C" LOC = "L3" | IOSTANDARD = LVCMOS33;
NET "D" LOC = "P11" | IOSTANDARD = LVCMOS33;

NET "F" LOC = "M5" | IOSTANDARD = LVCMOS33;


```

FPGA Board

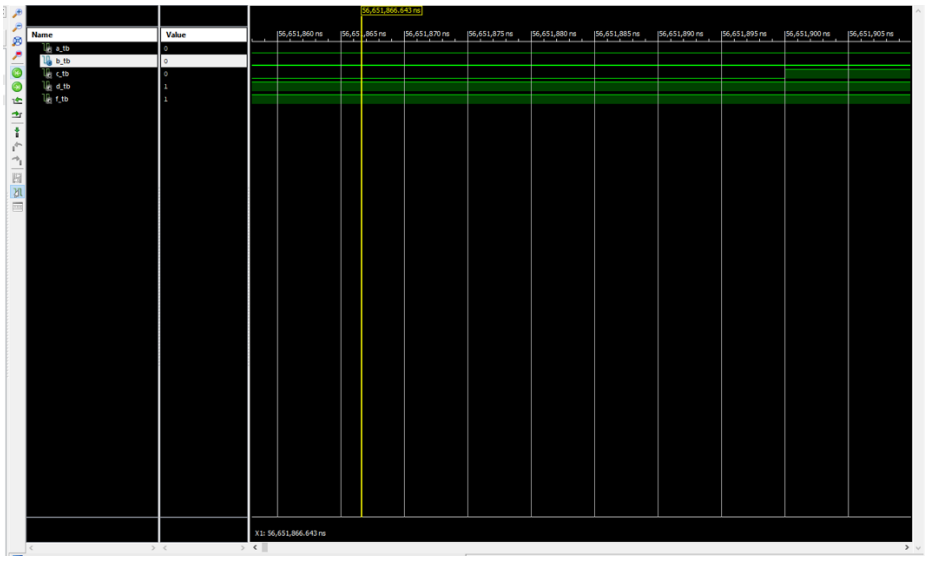

BASYS 2
DIGILENT
BEYOND THEORY
Input D
Input C
Input B
Input A
Output
LED F

Testbench:

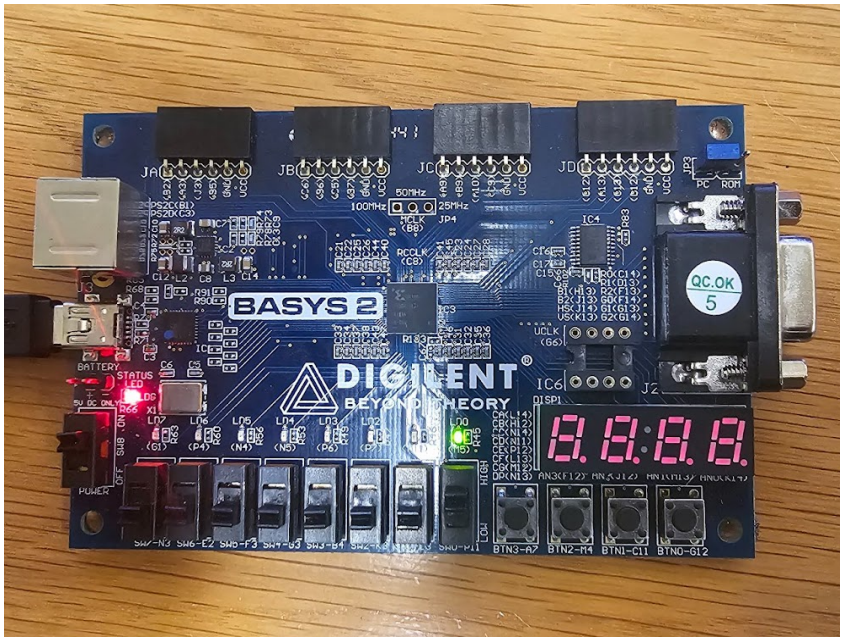
```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    12:55:53 10/08/2025
-- Design Name:    Minterms Testbench
-- Module Name:    Minterms_TB - Behavioral
-- Project Name:   Advanced Digital Logic Task 4 - Minterms
-- Target Device:  FPGA
-- Tool versions:  Xilinx
-- Description:
-- VHDL Test Bench created to test the functionality of the minterm-based function
-- (Minterms) which uses separate inputs A, B, C, D. The testbench verifies
-- the output (F_tb) based on different input combinations of A_tb, B_tb, C_tb, and D_tb.
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration for the testbench
entity Minterms_TB is
end Minterms_TB;

architecture Behavioral of Minterms_TB is


 --  Component Declaration for the Unit Under Test (UUT)
 component Minterms
     port(
         A : in  std_logic;  -- MSB (bit 3)
         B : in  std_logic;  -- bit 2
         C : in  std_logic;  -- bit 1
         D : in  std_logic;  -- LSB (bit 0)
         F : out std_logic
     );
 end component;


 -- Signals to drive the UUT inputs/outputs
 signal A_tb : std_logic := '0';  -- Test input for bit 3
 signal B_tb : std_logic := '0';  -- Test input for bit 2
 signal C_tb : std_logic := '0';  -- Test input for bit 1
 signal D_tb : std_logic := '0';  -- Test input for bit 0
 signal F_tb : std_logic;         -- Output signal for the function

begin


 --  Instantiate the Unit Under Test (UUT)
 UUT: Minterms
     port map (
         A => A_tb,
         B => B_tb,
         C => C_tb,
         D => D_tb,
         F => F_tb
     );

 --  Stimulus Process: apply different combinations and observe output

 stim_proc: process
 begin
     -- Hold reset state for 100 ns
     wait for 100 ns;

     -- Test case 1: "0001" => A=0, B=0, C=0, D=1 => F should be '1'
     A_tb <= '0';  B_tb <= '0';  C_tb <= '0';  D_tb <= '1';
     wait for 100 ns;

     -- Test case 2: "0011" => A=0, B=0, C=1, D=1 => F should be '1'
     A_tb <= '0';  B_tb <= '0';  C_tb <= '1';  D_tb <= '1';
     wait for 100 ns;

     -- Test case 3: "1001" => A=1, B=0, C=0, D=1 => F should be '1'
     A_tb <= '1';  B_tb <= '0';  C_tb <= '0';  D_tb <= '1';
     wait for 100 ns;

     -- Test case 4: "1011" => A=1, B=0, C=1, D=1 => F should be '1'
     A_tb <= '1';  B_tb <= '0';  C_tb <= '1';  D_tb <= '1';
     wait for 100 ns;

     -- Test case 5: "0100" => A=0, B=1, C=0, D=0 => F should be '0'
     A_tb <= '0';  B_tb <= '1';  C_tb <= '0';  D_tb <= '0';
     wait for 100 ns;

     -- Test case 6: "1100" => A=1, B=1, C=0, D=0 => F should be '0'
     A_tb <= '1';  B_tb <= '1';  C_tb <= '0';  D_tb <= '0';
     wait for 100 ns;

     --wait;
 end process;

end Behavioral;
```

Test Case 1

Input Values: A = 0, B=0, C=0, D=1
Output Value: F = 1 (as expected)
Simulation Time: 56,651,866 ns
Observation: The output line is high while the binary number 1 is at the input line, confirming the operation of the Minterms function

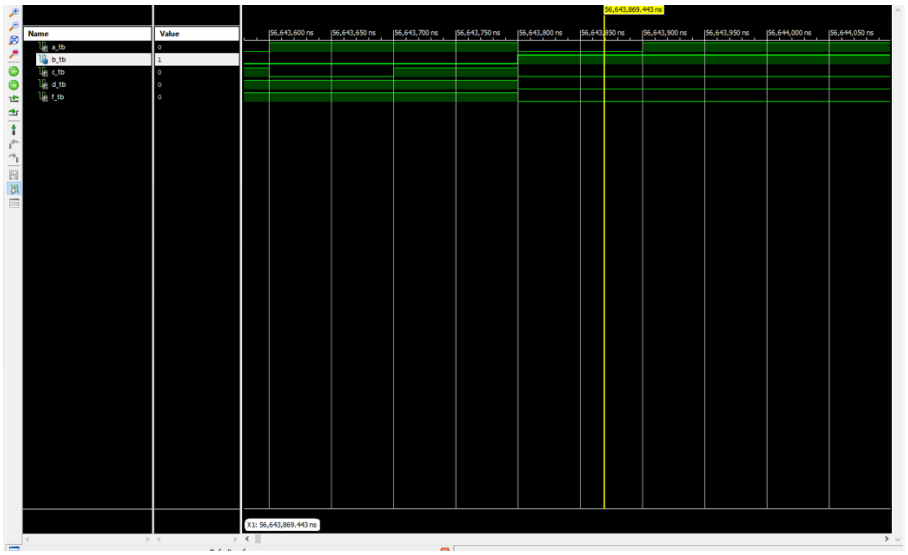


The board exhibits on LED (M5) output.

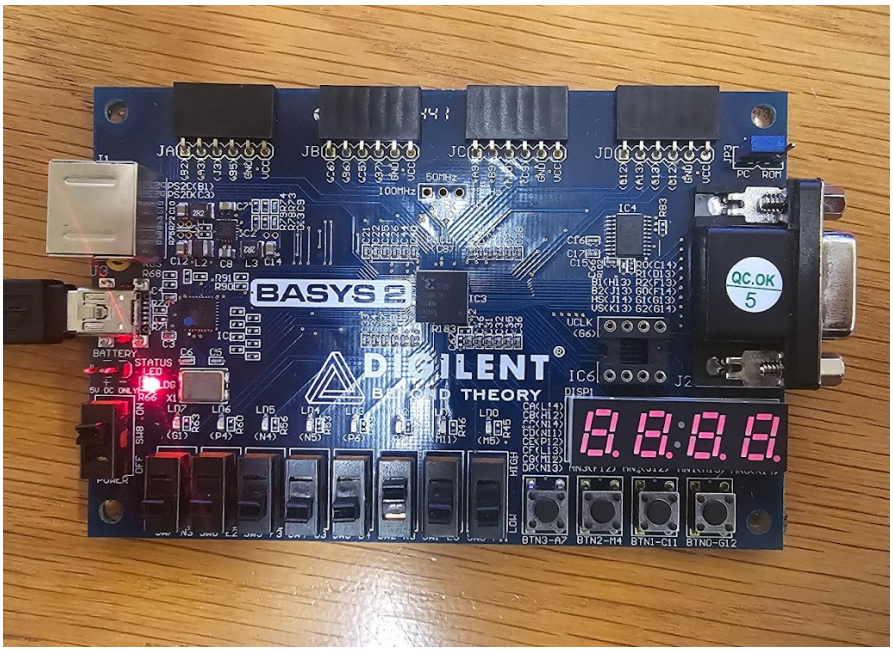


Test Case 2

Input Values: A = 0, B=1, C=0, D=0
Output Value: F = 0 (as expected)
Simulation Time: 56,643,869 ns
Observation: The output line is low while the binary number 4 is at the input line, confirming the operation of the Minterms function.

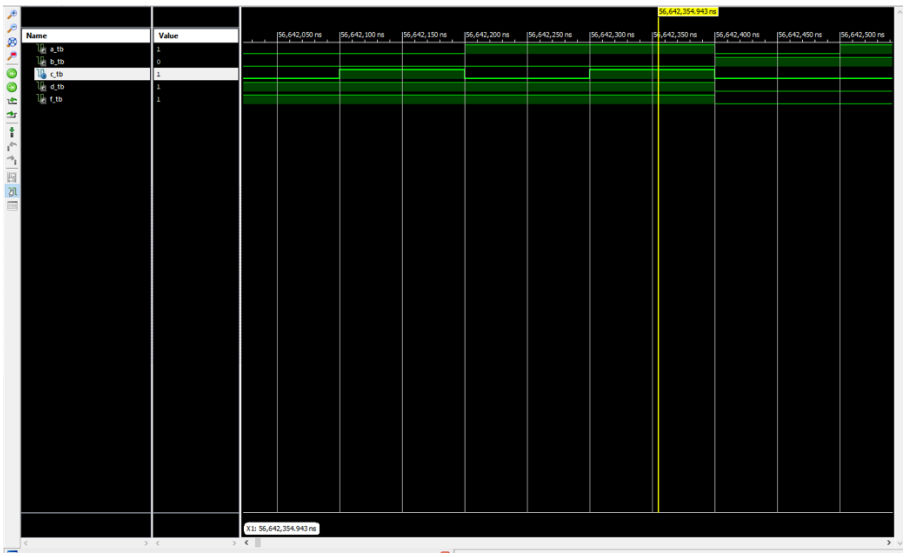


The board exhibits off LED (M5) output.

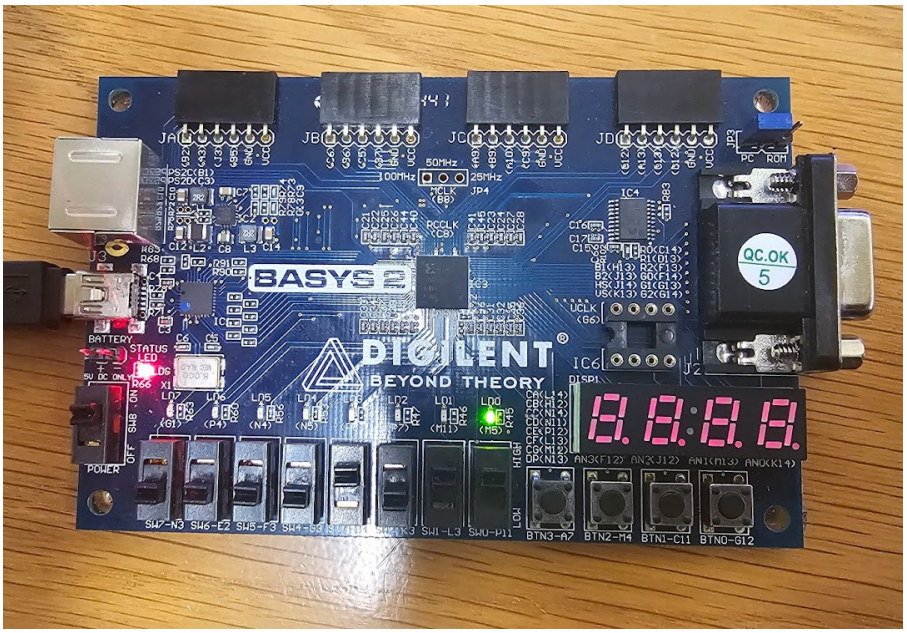


Test Case 3

Input Values: A = 1, B=0, C=1, D=1
Output Value: F = 1 (as expected)
Simulation Time: 56,642,354 ns
Observation: The output line is high while the binary number 11 is at the input lines, confirming the operation of the Minterms function.



The board exhibits on LED (M5) output.



Task 5

8:1 Multiplexer

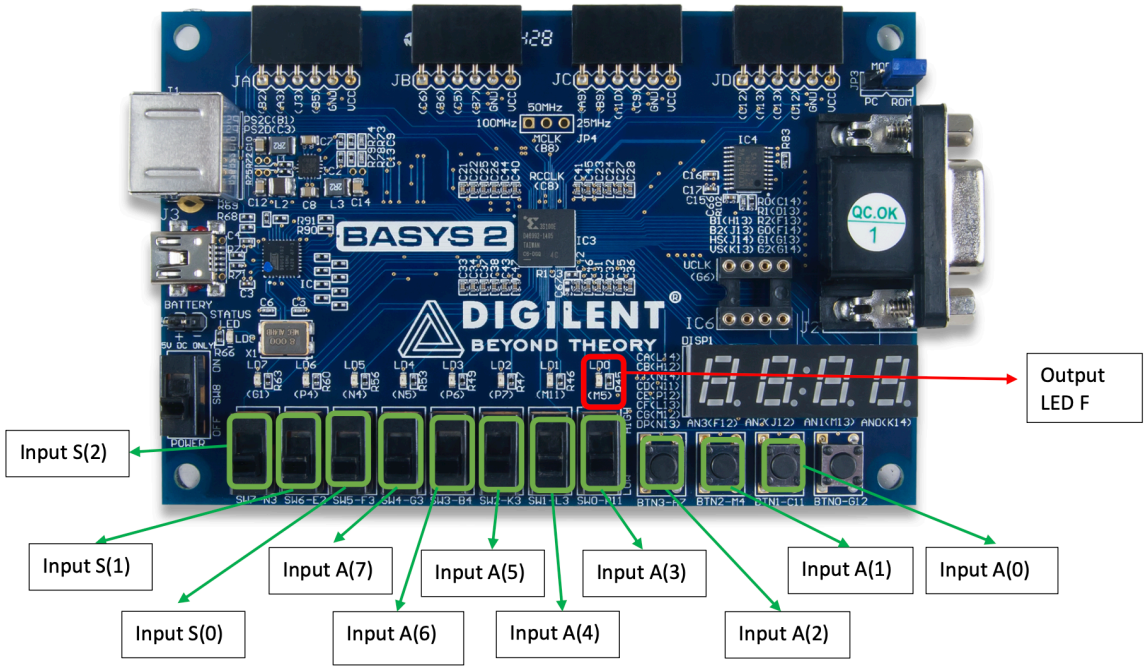
VHDL code

```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    23:36:12 09/02/2025
-- Design Name:    8-to-1 Multiplexer Implementation
-- Module Name:    Multiplexer - Behavioral
-- Project Name:   Advanced Digital Logic Task 5 - Multiplexer
-- Target Devices: FPGA
-- Tool versions:  Xilinx
-- Description:
-- This VHDL module implements an 8-to-1 multiplexer using behavioral modeling.
-- The multiplexer selects one of the 8 input signals (A) based on the 3-bit select input (S)
-- and routes the selected input to the output (F).
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
--
-- Revision:
-- Revision 0.01 - File Created
--
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration for the 8-to-1 multiplexer
entity Multiplexer is
    Port ( A : in  STD_LOGIC_VECTOR(7 downto 0);   -- 8-bit input vector (A(0) to A(7))
           S : in  STD_LOGIC_VECTOR(2 downto 0);   -- 3-bit select input to choose one of the inputs
           F : out  STD_LOGIC);                    -- Output signal representing the selected input
end Multiplexer;


-- Architecture defining the behavior of the 8-to-1 multiplexer
architecture Behavioral of Multiplexer is
begin
    -- Process block to select the appropriate input based on the value of S
    process(S, A)
    begin
        -- Use a case statement to route the appropriate input bit to F
        case S is
            when "000" =>  -- Select input A(0) when S = "000"
                F <= A(0);
            when "001" =>  -- Select input A(1) when S = "001"
                F <= A(1);
            when "010" =>  -- Select input A(2) when S = "010"
                F <= A(2);
            when "011" =>  -- Select input A(3) when S = "011"
                F <= A(3);
            when "100" =>  -- Select input A(4) when S = "100"
                F <= A(4);
            when "101" =>  -- Select input A(5) when S = "101"
                F <= A(5);
            when "110" =>  -- Select input A(6) when S = "110"
                F <= A(6);
            when "111" =>  -- Select input A(7) when S = "111"
                F <= A(7);
            when others =>  -- Default case: output '0' if S is outside the expected range
                F <= '0';
        end case;
    end process;
end Behavioral;
```

Constraint File Implementation:

```
# Specify the I/O standard to match the external device voltage levels.
# This ensures proper signal integrity, compatibility, and power optimization.

NET "S(2)" LOC = "N3" | IOSTANDARD = LVCMOS33;
NET "S(1)" LOC = "E2" | IOSTANDARD = LVCMOS33;
NET "S(0)" LOC = "F3" | IOSTANDARD = LVCMOS33;


NET "A(7)" LOC = "G3" | IOSTANDARD = LVCMOS33;
NET "A(6)" LOC = "B4" | IOSTANDARD = LVCMOS33;
NET "A(5)" LOC = "K3" | IOSTANDARD = LVCMOS33;
NET "A(4)" LOC = "L3" | IOSTANDARD = LVCMOS33;
NET "A(3)" LOC = "P11" | IOSTANDARD = LVCMOS33;
NET "A(2)" LOC = "A7" | IOSTANDARD = LVCMOS33;
NET "A(1)" LOC = "M4" | IOSTANDARD = LVCMOS33;
NET "A(0)" LOC = "C11" | IOSTANDARD = LVCMOS33;

NET "F" LOC = "M5" | IOSTANDARD = LVCMOS33;

```

FPGA Board


BASYS 2
DIGILENT
BEYOND THEORY
Output LED F
Input S(2)
Input S(1)
Input S(0)
Input A(7)
Input A(6)
Input A(5)
Input A(4)
Input A(3)
Input A(2)
Input A(1)
Input A(0)

Testbench:

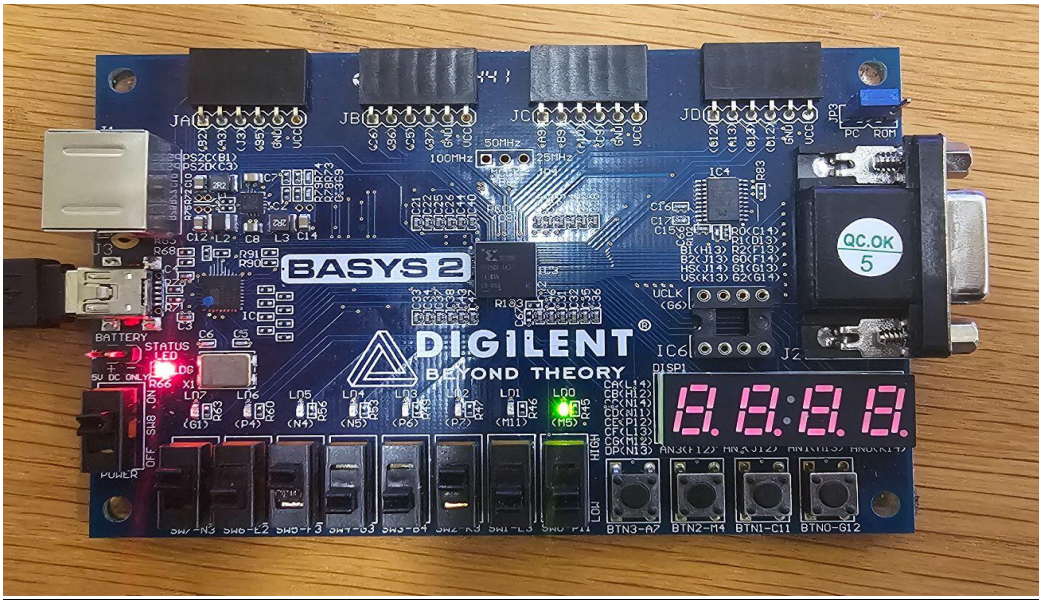
```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    23:50:12 09/02/2025
-- Design Name:    Multiplexer Testbench
-- Module Name:    Multiplexer_TB - Behavioral
-- Project Name:   Advanced Digital Logic Task 5 - Multiplexer
-- Target Device:  FPGA
-- Tool versions:  Xilinx
-- Description:
-- VHDL Test Bench created to test the functionality of the 8-to-1 multiplexer (Multiplexer).
-- The testbench verifies the output (F_tb) based on different combinations of the input vector (A_tb)
-- and the select input (S_tb).
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration for the testbench
entity Multiplexer_TB is
end Multiplexer_TB;

-- Architecture defining the behavior of the testbench
architecture Behavioral of Multiplexer_TB is

    -- Component Declaration for the Unit Under Test (UUT)
    component Multiplexer
        port(
            A : in  std_logic_vector(7 downto 0);  -- 8-bit input
            S : in  std_logic_vector(2 downto 0);  -- 3-bit select input
            F : out std_logic                      -- Output signal for the selected input
        );
    end component;

    -- Input signals to test the multiplexer
    signal A_tb : std_logic_vector(7 downto 0) := "00000000";  -- Test input vector
    signal S_tb : std_logic_vector(2 downto 0) := "000";       -- Test select input
    signal F_tb : std_logic;                                   -- Test output signal

begin

    -- Instantiate the Unit Under Test (UUT) and connect it to the test signals
    UUT: Multiplexer port map (
        A => A_tb,
        S => S_tb,
        F => F_tb
    );
```

```vhdl

    -- Stimulus process: apply input combinations and observe the output
    stim_proc: process
    begin
        -- Hold the initial state for 100 ns
        wait for 100 ns;

        -- Test case 1: A = "00000001", S = "000" (expect F_tb = '1')
        A_tb <= "00000001"; S_tb <= "000";
        wait for 1024 ns;

        -- Test case 2: A = "00000010", S = "001" (expect F_tb = '1')
        A_tb <= "00000010"; S_tb <= "001";
        wait for 512 ns;

        -- Test case 3: A = "00000100", S = "010" (expect F_tb = '1')
        A_tb <= "00000100"; S_tb <= "010";
        wait for 256 ns;

        -- Test case 4: A = "00001000", S = "011" (expect F_tb = '1')
        A_tb <= "00001000"; S_tb <= "011";
        wait for 128 ns;

        -- Test case 5: A = "00010000", S = "100" (expect F_tb = '1')
        A_tb <= "00010000"; S_tb <= "100";
        wait for 64 ns;

        -- Test case 6: A = "00100000", S = "101" (expect F_tb = '1')
        A_tb <= "00100000"; S_tb <= "101";
        wait for 32 ns;

        -- Test case 7: A = "01000000", S = "110" (expect F_tb = '1')
        A_tb <= "01000000"; S_tb <= "110";
        wait for 16 ns;

        -- Test case 8: A = "10000000", S = "111" (expect F_tb = '1')
        A_tb <= "10000000"; S_tb <= "111";
        wait for 8 ns;

        -- Test case 9: A = "00000000", S = "010" (expect F_tb = '0')
        A_tb <= "00000000"; S_tb <= "010";
        wait for 4 ns;

        --wait;
    end process;

end Behavioral;

```

Test Case 1

Input Values: A = 00100000, S = 101
Output Value: F = 1 (as expected)
Simulation Time: 186 ns
Observation: The output line is high showing that the 6th input of A (right to left) is propagated out when binary number 5 is at the selector input line, confirming the expected behavior of the 8:1 Multiplexer operation.

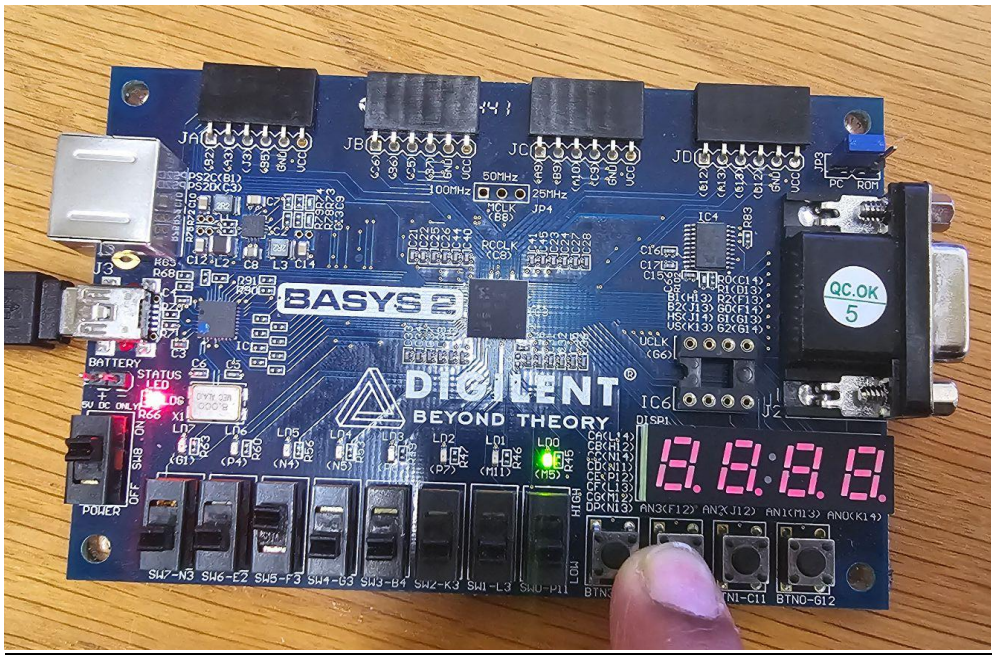
The board exhibits on LED (M5) output.



Test Case 2

Input Values: A = 00000010, S = 001
Output Value: F = 1 (as expected)
Simulation Time: 186 ns
Observation: The output line is high showing that the 2nd input of A (right to left) is propagated out when binary number 1 is at the selector input line, confirming the expected behavior of the 8:1 Multiplexer operation.

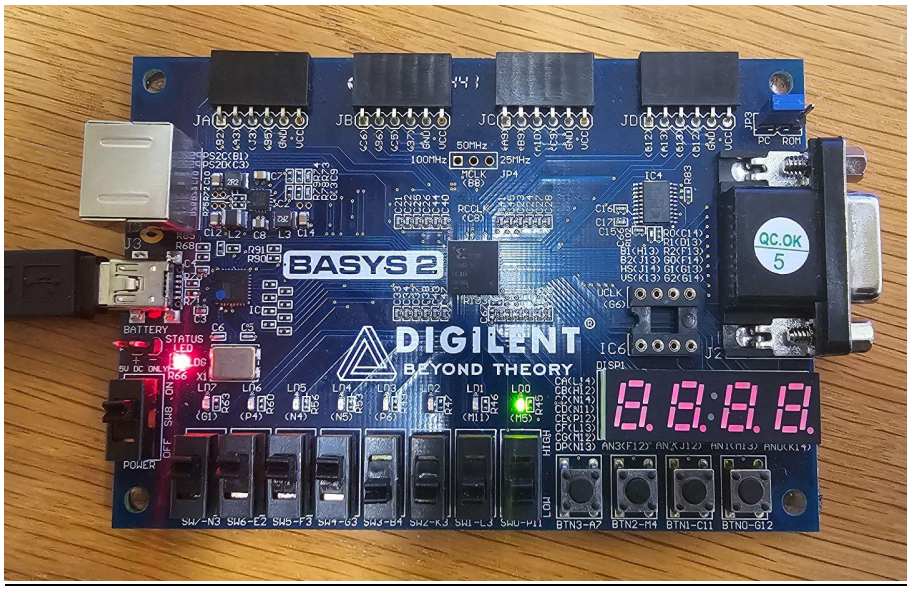
The board exhibits on LED (M5) output.



Test Case 3

Input Values: A = 10000000, S = 111
Output Value: F = 1 (as expected)
Simulation Time: 186 ns
Observation: The output line is high showing that the 8th input of A (right to left) is propagated out when binary number 7 is at the selector input line, confirming the expected behavior of the 8:1 Multiplexer operation.

The board exhibits on LED (M5) output.

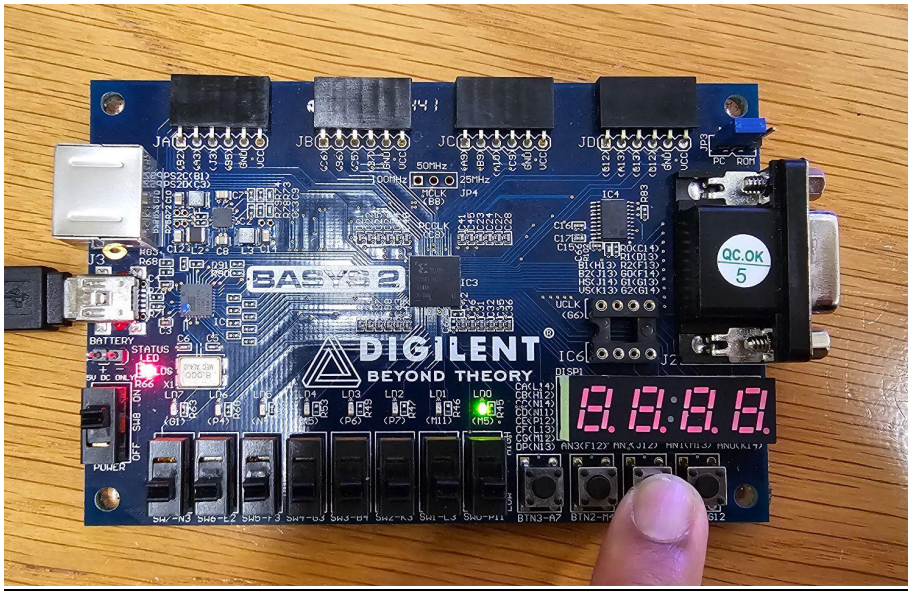


Using “With-Select” Statement:

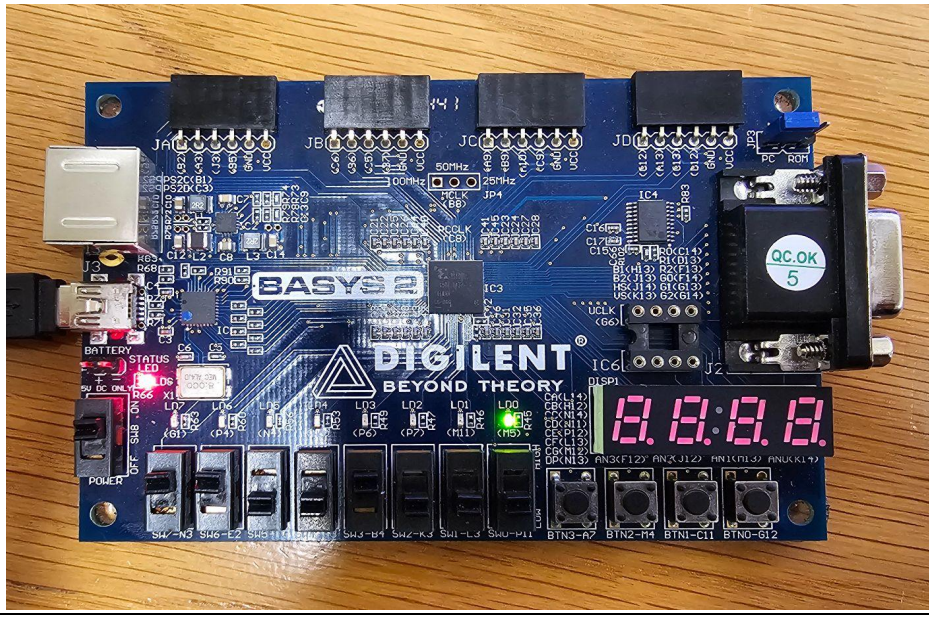
```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    23:50:33 09/02/2025
-- Design Name:    8-to-1 Multiplexer Using "With-Select"
-- Module Name:    Multiplexer_With_Select - Behavioral
-- Project Name:   Advanced Digital Logic Task 5 - Multiplexer Using "With-Select"
-- Target Devices: FPGA
-- Tool versions:  Xilinx
-- Description:
-- This VHDL module implements an 8-to-1 multiplexer using the "with-select" statement.
-- The multiplexer selects one of the 8 input signals (A) based on the 3-bit select input (S)
-- and routes the selected input to the output (F).
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
--
-- Revision:
-- Revision 0.01 - File Created
--
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration for the 8-to-1 multiplexer
entity Multiplexer_With_Select is
    Port ( A : in  STD_LOGIC_VECTOR(7 downto 0);   -- 8-bit input vector (A(0) to A(7))
           S : in  STD_LOGIC_VECTOR(2 downto 0);   -- 3-bit select input to choose one of the inputs
           F : out  STD_LOGIC);                    -- Output signal representing the selected input
end Multiplexer_With_Select;


-- Architecture defining the behavior of the 8-to-1 multiplexer using "with-select"
architecture Behavioral of Multiplexer_With_Select is

begin
    -- With-select construct: based on the value of S, route the appropriate input to F
    with S select
        F <= A(0) when "000",  -- Select input A(0) when S = "000"
             A(1) when "001",  -- Select input A(1) when S = "001"
             A(2) when "010",  -- Select input A(2) when S = "010"
             A(3) when "011",  -- Select input A(3) when S = "011"
             A(4) when "100",  -- Select input A(4) when S = "100"
             A(5) when "101",  -- Select input A(5) when S = "101"
             A(6) when "110",  -- Select input A(6) when S = "110"
             A(7) when "111",  -- Select input A(7) when S = "111"
             '0'  when others; -- Default case: output '0' if S is outside the expected range

end Behavioral;


```

Test cases

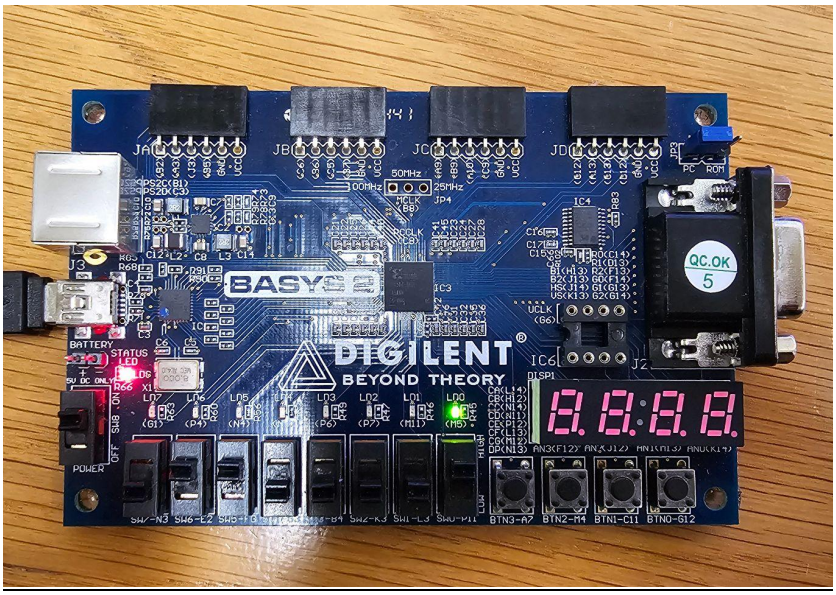
i. Selector Signal: 000
Input Signal Propagated Out: 0



ii. Selector Signal: 110
Input Signal Propagated Out: 6



iii. Selector Signal: 011
Input Signal Propagated Out: 3

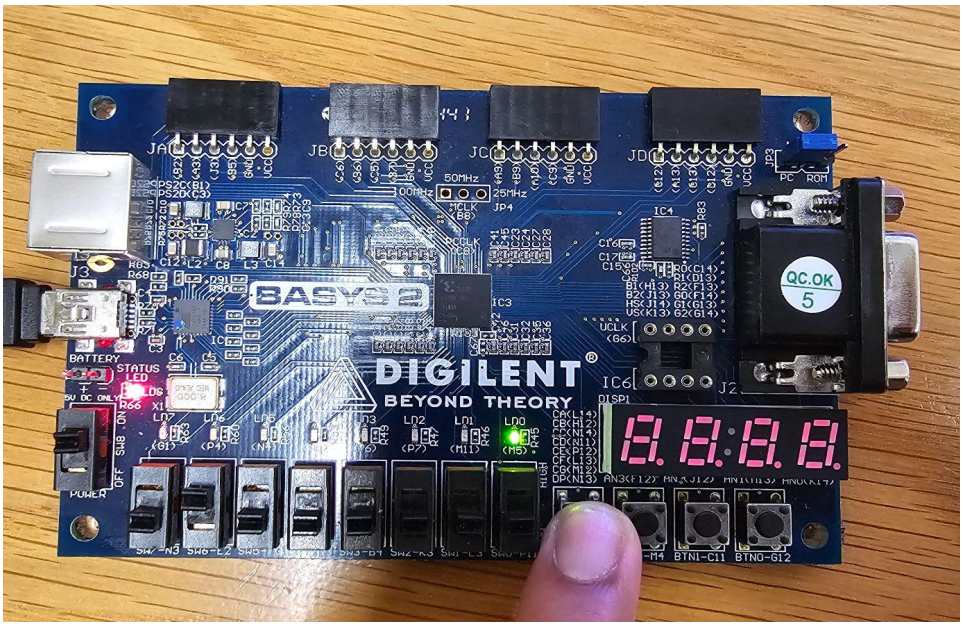


Using “When-Else” Statement:

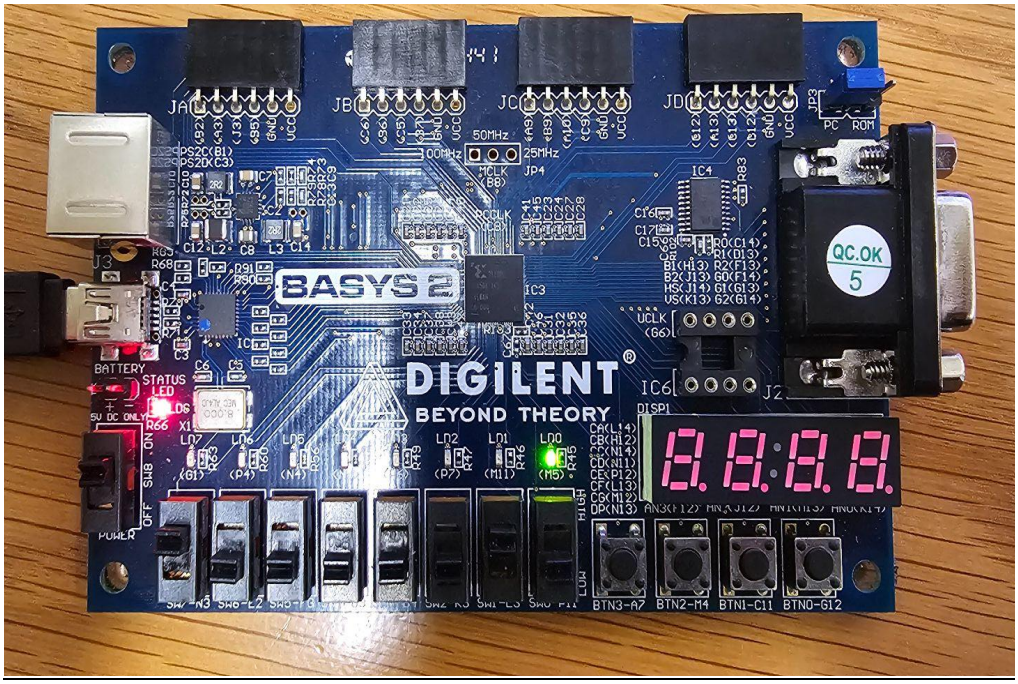
```vhdl
----------------------------------------------------------------------------------
-- Company: NYU Abu Dhabi
-- Engineer: Aman Sunesh, Demarce Williams
--
-- Create Date:    00:00:15 09/02/2025
-- Design Name:    8-to-1 Multiplexer Using "When-Else"
-- Module Name:    Multiplexer_When_Else - Behavioral
-- Project Name:   Advanced Digital Logic Task 5 - Multiplexer Using "When-Else"
-- Target Devices: FPGA
-- Tool versions:  Xilinx
-- Description:
-- This VHDL module implements an 8-to-1 multiplexer using the "when-else" statement.
-- The multiplexer selects one of the 8 input signals (A) based on the 3-bit select input (S)
-- and routes the selected input to the output (F).
--
-- Dependencies:
-- Standard IEEE 1164 logic libraries.
--
-- Revision:
-- Revision 0.01 - File Created
--
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration for the 8-to-1 multiplexer
entity Multiplexer_When_Else is
    Port ( A : in  STD_LOGIC_VECTOR(7 downto 0);
           S : in  STD_LOGIC_VECTOR(2 downto 0);
           F : out  STD_LOGIC);
end Multiplexer_When_Else;

-- Architecture defining the behavior of the 8-to-1 multiplexer using "when-else"
architecture Behavioral of Multiplexer_When_Else is

begin

    -- When-else statement: based on the value of S, route the appropriate input to F
    F <= A(0) when S = "000" else  -- Select input A(0) when S = "000"
         A(1) when S = "001" else  -- Select input A(1) when S = "001"
         A(2) when S = "010" else  -- Select input A(2) when S = "010"
         A(3) when S = "011" else  -- Select input A(3) when S = "011"
         A(4) when S = "100" else  -- Select input A(4) when S = "100"
         A(5) when S = "101" else  -- Select input A(5) when S = "101"
         A(6) when S = "110" else  -- Select input A(6) when S = "110"
         A(7) when S = "111" else  -- Select input A(7) when S = "111"
         '0';                      -- Default case: output '0' if S is outside the expected range

end Behavioral;
```

Test Cases:

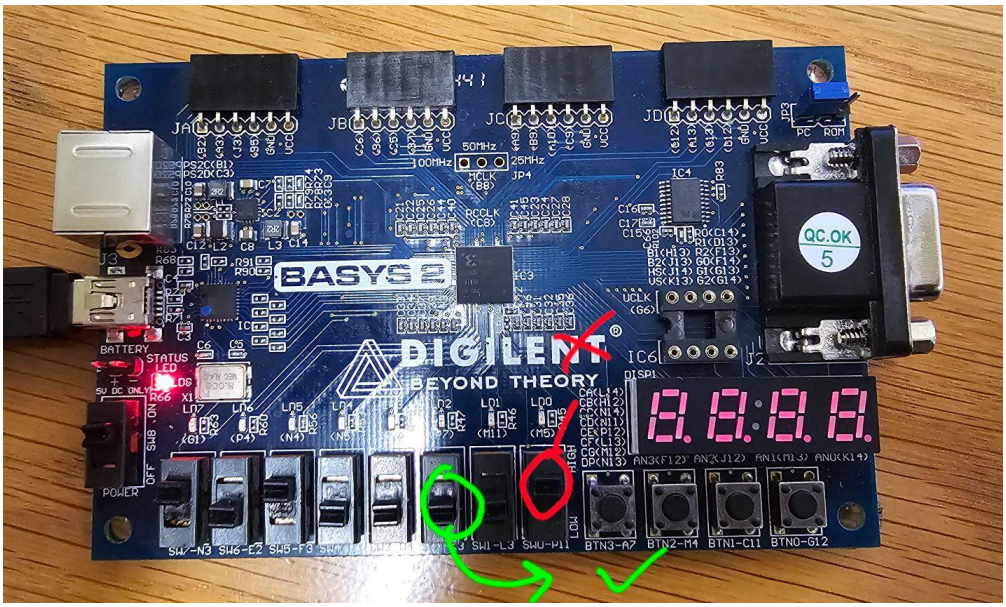
i. Selector Signal: 010
Input Signal Propagated Out: 2



ii. Selector Signal: 100
Input Signal Propagated Out: 4



iii. Selector Signal: 101
Input Signal Propagated Out: 5
Since bit 5 is low, the LED stays off. Despite bit 3 being, on its signal does not propagate out as the selector signal for three (011) is not at the input



Q.1.)What VHDL programming difference in architecture you have seen between Case statements and the With-Select/When-Else statements. Are they all initialized and implemented in the same way within the Architecture, or implemented differently?

Ans: In VHDL, case statements typically appear inside a process and handle multiple branches in a structured way (e.g., “when X => do something”). On the other hand, with-select and when-else are concurrent statements that assign an output based on the input condition without requiring a process. “With-select” provides a compact way to assign a single output, whereas “when-else” can tend to be more readable when handling a series of conditions. Despite these syntactical differences, all three approaches implement exactly the same multiplexer functionality. They differ mainly in how the code is written and the logic is organized, not in the final functionality or initialization.