Hoàng Như Vĩnh – Đại học FPT (Greenwich)

VinhHN7@fe.edu.vn

Báo cáo Thực hành

**ELT2041: Lab 1**

Table of Contents

[Abstract 2](#_Toc164068333)

[Phần 1 3](#_Toc164068334)

[Phần 2 6](#_Toc164068335)

[Phần 3 9](#_Toc164068336)

[Phần 4 12](#_Toc164068337)

[Cải tiến & Kết luận 17](#_Toc164068338)

Table of Figures

[Figure 1 - Kết quả chạy testbench cho mux21 4](#_Toc163986682)

[Figure 2 - Waveform của testbench mux21 4](#_Toc163986683)

[Figure 3 - Kết quả chạy testbench mux21\_tb\_nand 7](#_Toc163986684)

[Figure 4 - Waveform của mux21\_tb\_nand 7](#_Toc163986685)

[Figure 5 - Kết quả chạy testbench mux21\_8bit\_tb 10](#_Toc163986686)

[Figure 6 - Waveform của mux21\_8bit\_tb 10](#_Toc163986687)

[Figure 7 - Kết quả chạy testbench mux21\_nbit\_tb 14](#_Toc163986688)

[Figure 8 - Waveform của testbench mux21\_nbit\_tb 14](#_Toc163986689)

[Figure 9 - Cấu trúc thư mục dự án sau khi thêm các thư mục design và testbench 16](#_Toc163986690)

[Figure 10 - Cách chạy compiling tất cả design và testbench với 1 cậu lệnh sử dụng "\*" 16](#_Toc163986691)

# Abstract

Bài cáo tập trung vào thực hành và ứng dụng sử dụng ngôn ngữ VHDL trong việc thiết kế và chạy testbench trong việc thiết kế vi mạch. Mục tiêu của bài thực hành còn giúp thực hành sử dụng các công cụ thiết kế vi mạch của Synopsys như VCS cho việc compiling và Verdi cho việc xem waveform.

Trong quá trình thực hành, việc sử dụng VCS và Verdi thông qua kết nối remote đến server VKU gặp khó khan trong việc làm thực hành tại nhà. Để giải quyết vấn đề này, bài báo cáo sẽ sử dụng các công cụ Open Source là GHDL và Gtkwave.

Kết quả của bài thực hành sẽ được cung cấp, trình bày cũng như cung cấp góc nhìn tổng quan cho việc sử dụng VHDL cũng như các công cụ liên quan trong thiết kế vi mạch.

# Phần 1

Theo yêu cầu của yêu cầu Lab1, mux21 sẽ được viết như sau

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY *mux21* IS

  PORT (

    x : IN *STD\_LOGIC*;

    y : IN *STD\_LOGIC*;

    s : IN *STD\_LOGIC*;

    m : OUT *STD\_LOGIC*);

END ENTITY *mux21*;

*ARCHITECTURE df OF mux21 IS*

*BEGIN -- architecture df*

*m <= (y AND s) OR (x AND (NOT s));*

*END ARCHITECTURE df*;

Để thuận tiện cho việc kiểm tra sự hoạt động của mux21, testbench mux21\_tb sẽ được viết để cung cấp các test cases dựa theo truth table đã được cho.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY *mux21\_tb* IS

END ENTITY *mux21\_tb*;

*ARCHITECTURE behavior OF mux21\_tb IS*

*COMPONENT mux21*

*PORT (*

*x : IN STD\_LOGIC;*

*y : IN STD\_LOGIC;*

*s : IN STD\_LOGIC;*

*m : OUT STD\_LOGIC);*

*END COMPONENT;*

*SIGNAL x\_tb : STD\_LOGIC := '0';*

*SIGNAL y\_tb : STD\_LOGIC := '1';*

*SIGNAL s\_tb : STD\_LOGIC := '0';*

*SIGNAL m\_tb : STD\_LOGIC;*

*BEGIN*

*uut\_df : ENTITY work.mux21(df) PORT MAP(*

*x => x\_tb,*

*y => y\_tb,*

*s => s\_tb,*

*m => m\_tb*

*);*

*-- Stimulus process*

*stim\_proc : PROCESS*

*BEGIN*

*-- Test case 1*

*x\_tb <= '0';*

*y\_tb <= '0';*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '0') REPORT "Test Case 1 Failed" SEVERITY ERROR;*

*-- Test case 2*

*x\_tb <= '1';*

*y\_tb <= '0';*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '1') REPORT "Test Case 2 Failed" SEVERITY ERROR;*

*-- Test case 3*

*x\_tb <= '0';*

*y\_tb <= '1';*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '0') REPORT "Test Case 3 Failed" SEVERITY ERROR;*

*-- Test case 4*

*x\_tb <= '1';*

*y\_tb <= '1';*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '1') REPORT "Test Case 4 Failed" SEVERITY ERROR;*

*-- Test case 5*

*x\_tb <= '0';*

*y\_tb <= '0';*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '0') REPORT "Test Case 5 Failed" SEVERITY ERROR;*

*-- Test case 6*

*x\_tb <= '1';*

*y\_tb <= '0';*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '0') REPORT "Test Case 6 Failed" SEVERITY ERROR;*

*-- Test case 7*

*x\_tb <= '0';*

*y\_tb <= '1';*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '1') REPORT "Test Case 7 Failed" SEVERITY ERROR;*

*-- Test case 8*

*x\_tb <= '1';*

*y\_tb <= '1';*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '1') REPORT "Test Case 8 Failed" SEVERITY ERROR;*

*-- End simulation*

*REPORT "END OF SIMULATION" SEVERITY NOTE;*

*WAIT;*

*END PROCESS;*

*END*;

Kết quả chạy mô phỏng được thấy như hình sau kèm theo thông báo tử terminal cho phép biết được tất cả test cases đều đã PASS và không có cases nào FAILED

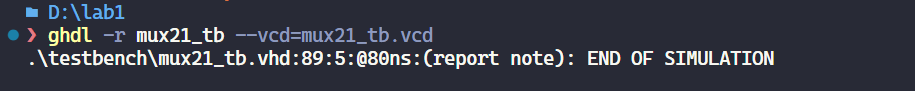


Figure 1 - Kết quả chạy testbench cho mux21

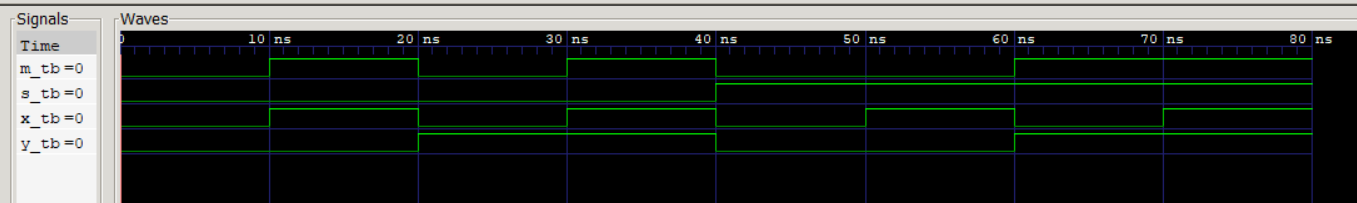


Figure 2 - Waveform của testbench mux21

Dựa theo waveform, kết quả cho thấy đúng với truth table đã được cho

# Phần 2

Theo yêu cầu của đề bài, mux21 được thêm architecture df\_nand để chỉ sử dụng cổng NAND

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY *mux21* IS

  PORT (

    x : IN *STD\_LOGIC*;

    y : IN *STD\_LOGIC*;

    s : IN *STD\_LOGIC*;

    m : OUT *STD\_LOGIC*);

END ENTITY *mux21*;

*ARCHITECTURE df OF mux21 IS*

*BEGIN -- architecture df*

*m <= (y AND s) OR (x AND (NOT s));*

*END ARCHITECTURE df*;

*ARCHITECTURE df\_nand OF mux21 IS*

*BEGIN -- architecture df\_nand*

*m <= ((x NAND (s NAND s)) NAND (y NAND s));*

*END ARCHITECTURE df\_nand*;

Tương tự như phần 1, testbench mux21\_tb\_nand sẽ được viết để kiểm tra kết quả

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY *mux21\_tb\_nand* IS

END ENTITY *mux21\_tb\_nand*;

*ARCHITECTURE behavior OF mux21\_tb\_nand IS*

*COMPONENT mux21*

*PORT (*

*x : IN STD\_LOGIC;*

*y : IN STD\_LOGIC;*

*s : IN STD\_LOGIC;*

*m : OUT STD\_LOGIC);*

*END COMPONENT;*

*SIGNAL x\_tb : STD\_LOGIC := '0';*

*SIGNAL y\_tb : STD\_LOGIC := '1';*

*SIGNAL s\_tb : STD\_LOGIC := '0';*

*SIGNAL m\_tb : STD\_LOGIC;*

*BEGIN*

*uut\_df\_nand : ENTITY work.mux21(df\_nand) PORT MAP( -- use df\_nand architecture of mux21*

*x => x\_tb,*

*y => y\_tb,*

*s => s\_tb,*

*m => m\_tb*

*);*

*-- Stimulus process*

*stim\_proc : PROCESS*

*BEGIN*

*-- Test case 1*

*x\_tb <= '0';*

*y\_tb <= '0';*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '0') REPORT "Test Case 1 Failed" SEVERITY ERROR;*

*-- Test case 2*

*x\_tb <= '1';*

*y\_tb <= '0';*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '1') REPORT "Test Case 2 Failed" SEVERITY ERROR;*

*-- Test case 3*

*x\_tb <= '0';*

*y\_tb <= '1';*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '0') REPORT "Test Case 3 Failed" SEVERITY ERROR;*

*-- Test case 4*

*x\_tb <= '1';*

*y\_tb <= '1';*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '1') REPORT "Test Case 4 Failed" SEVERITY ERROR;*

*-- Test case 5*

*x\_tb <= '0';*

*y\_tb <= '0';*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '0') REPORT "Test Case 5 Failed" SEVERITY ERROR;*

*-- Test case 6*

*x\_tb <= '1';*

*y\_tb <= '0';*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '0') REPORT "Test Case 6 Failed" SEVERITY ERROR;*

*-- Test case 7*

*x\_tb <= '0';*

*y\_tb <= '1';*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '1') REPORT "Test Case 7 Failed" SEVERITY ERROR;*

*-- Test case 8*

*x\_tb <= '1';*

*y\_tb <= '1';*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT (m\_tb = '1') REPORT "Test Case 8 Failed" SEVERITY ERROR;*

*-- End simulation*

*REPORT "END OF SIMULATION" SEVERITY NOTE;*

*WAIT;*

*END PROCESS;*

*END*;

Kết quả chạy testbench và waveform được thể hiện như sau

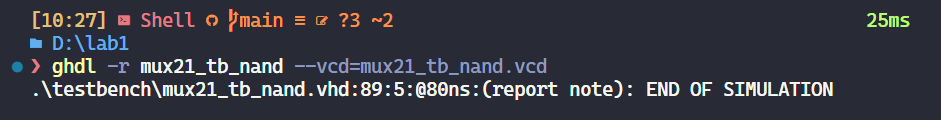


Figure 3 - Kết quả chạy testbench mux21\_tb\_nand

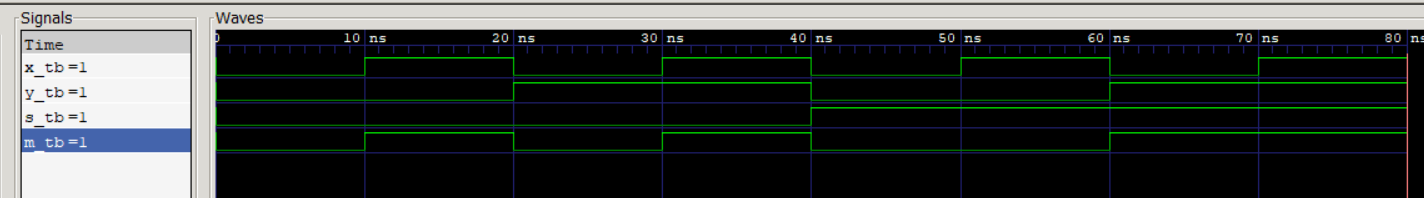


Figure 4 - Waveform của mux21\_tb\_nand

Tương tự như phần 1, waveform thể hiện đúng truth table đã được cho.

# Phần 3

Để hoàn thành được mux21\_8bit, 8 instances của mux21 sẽ được sử dụng theo như code sau

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY *mux21\_8bit* IS

  PORT (

    x : IN *STD\_LOGIC\_VECTOR*(7 DOWNTO 0);

    y : IN *STD\_LOGIC\_VECTOR*(7 DOWNTO 0);

    s : IN *STD\_LOGIC*;

    m : OUT *STD\_LOGIC\_VECTOR*(7 DOWNTO 0)

  );

END ENTITY *mux21\_8bit*;

*ARCHITECTURE df OF mux21\_8bit IS*

*COMPONENT mux21*

*PORT (*

*x : IN STD\_LOGIC;*

*y : IN STD\_LOGIC;*

*s : IN STD\_LOGIC;*

*m : OUT STD\_LOGIC*

*);*

*END COMPONENT mux21;*

*SIGNAL m\_internal : STD\_LOGIC\_VECTOR(7 DOWNTO 0);*

*BEGIN*

*-- Instantiating 8 instances of mux21*

*mux\_inst0 : mux21 PORT MAP(x(0), y(0), s, m\_internal(0));*

*mux\_inst1 : mux21 PORT MAP(x(1), y(1), s, m\_internal(1));*

*mux\_inst2 : mux21 PORT MAP(x(2), y(2), s, m\_internal(2));*

*mux\_inst3 : mux21 PORT MAP(x(3), y(3), s, m\_internal(3));*

*mux\_inst4 : mux21 PORT MAP(x(4), y(4), s, m\_internal(4));*

*mux\_inst5 : mux21 PORT MAP(x(5), y(5), s, m\_internal(5));*

*mux\_inst6 : mux21 PORT MAP(x(6), y(6), s, m\_internal(6));*

*mux\_inst7 : mux21 PORT MAP(x(7), y(7), s, m\_internal(7));*

*-- Assigning the output*

*m <= m\_internal;*

*END ARCHITECTURE df*;

Để kiểm tra tính đúng đắng của thiết kế, testbench mux21\_8bit\_tb được viết như sau với các test cases cơ bản để đảm bảo đầu ra được chọn theo select bit s.

Nếu để kiểm tra tất cả các khả năng của đầu vào thì các test cases sẽ rất nhiều. Do đó trong giai đoạn này thì chỉ có các test cases cơ bản để kiểm tra tính đúng đắn của thiết kế trước.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY *mux21\_8bit\_tb* IS

END ENTITY *mux21\_8bit\_tb*;

*ARCHITECTURE behavior OF mux21\_8bit\_tb IS*

*-- Component declaration for the Unit Under Test (UUT)*

*COMPONENT mux21\_8bit*

*PORT (*

*x : IN STD\_LOGIC\_VECTOR(7 DOWNTO 0);*

*y : IN STD\_LOGIC\_VECTOR(7 DOWNTO 0);*

*s : IN STD\_LOGIC;*

*m : OUT STD\_LOGIC\_VECTOR(7 DOWNTO 0)*

*);*

*END COMPONENT;*

*-- Declare signals*

*SIGNAL x : STD\_LOGIC\_VECTOR(7 DOWNTO 0) := (OTHERS => '0');*

*SIGNAL y : STD\_LOGIC\_VECTOR(7 DOWNTO 0) := (OTHERS => '0');*

*SIGNAL s : STD\_LOGIC := '0';*

*SIGNAL m : STD\_LOGIC\_VECTOR(7 DOWNTO 0);*

*BEGIN*

*-- Instantiate the Unit Under Test (UUT)*

*uut : mux21\_8bit PORT MAP(*

*x => x,*

*y => y,*

*s => s,*

*m => m*

*);*

*-- Stimulus process*

*stim\_proc : PROCESS*

*BEGIN*

*-- Test case 1*

*x <= "00000001";*

*y <= "11111111";*

*s <= '0';*

*WAIT FOR 10 ns;*

*ASSERT m = "00000001" REPORT "Test case 1 failed" SEVERITY ERROR;*

*-- Test case 2*

*x <= "00000010";*

*y <= "10101010";*

*s <= '1';*

*WAIT FOR 10 ns;*

*ASSERT m = "10101010" REPORT "Test case 2 failed" SEVERITY ERROR;*

*-- Test case 3*

*x <= "11111111";*

*y <= "00000000";*

*s <= '1';*

*WAIT FOR 10 ns;*

*ASSERT m = "00000000" REPORT "Test case 3 failed" SEVERITY ERROR;*

*-- End the simulation*

*REPORT "End simulation" SEVERITY NOTE;*

*WAIT;*

*END PROCESS stim\_proc;*

*END ARCHITECTURE behavior*;

Kết quả chạy testbench cũng như waveform được thể hiện như sau

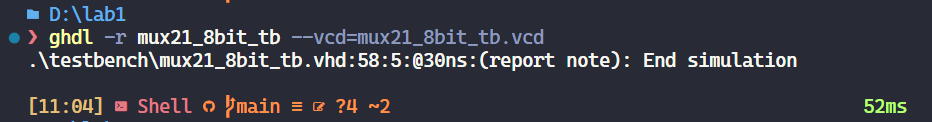


Figure 5 - Kết quả chạy testbench mux21\_8bit\_tb

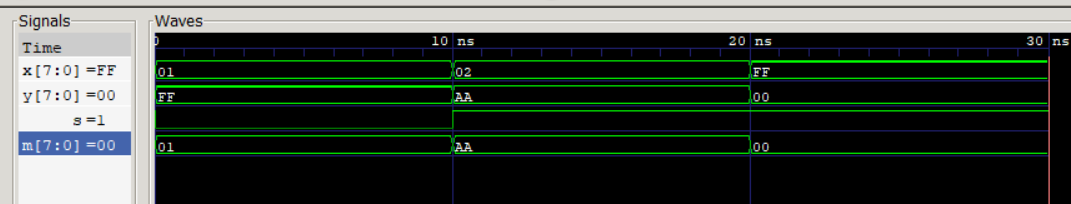


Figure 6 - Waveform của mux21\_8bit\_tb

Dựa theo waveform, m được chọn theo đúng selector s.

# Phần 4

Bộ mux21\_nbit được thiết kế như sau bằng cách dung GENERATE

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY *mux21\_nbit* IS

  GENERIC (

    DATA\_WIDTH : *INTEGER* := 16);

  PORT (

    X : IN *STD\_LOGIC\_VECTOR*(DATA\_WIDTH - 1 DOWNTO 0);

    Y : IN *STD\_LOGIC\_VECTOR*(DATA\_WIDTH - 1 DOWNTO 0);

    s : IN *STD\_LOGIC*;

    m : OUT *STD\_LOGIC\_VECTOR*(DATA\_WIDTH - 1 DOWNTO 0));

END ENTITY *mux21\_nbit*;

*ARCHITECTURE df OF mux21\_nbit IS*

*COMPONENT mux21 IS*

*PORT (*

*x : IN STD\_LOGIC;*

*y : IN STD\_LOGIC;*

*s : IN STD\_LOGIC;*

*m : OUT STD\_LOGIC);*

*END COMPONENT mux21;*

*SIGNAL m\_temp : STD\_LOGIC\_VECTOR(DATA\_WIDTH - 1 DOWNTO 0);*

*BEGIN -- ARCHITECTURE df*

*mux21\_gen : FOR i IN 0 TO DATA\_WIDTH - 1 GENERATE*

*mux21\_i : mux21*

*PORT MAP(*

*x => X(i),*

*y => Y(i),*

*s => s,*

*m => m\_temp(i));*

*END GENERATE mux21\_gen;*

*m <= m\_temp;*

*END ARCHITECTURE df*;

Testbench mux21\_nbit\_tb được viết như sau để kiểm kết quả của design. Các pattern đầu vào được chọn để kiểm tra các pattern cơ bản nhất của input

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY *mux21\_nbit\_tb* IS

END ENTITY *mux21\_nbit\_tb*;

*ARCHITECTURE behavior OF mux21\_nbit\_tb IS*

*-- Constants declaration*

*CONSTANT DATA\_WIDTH : INTEGER := 16;*

*-- Component declaration*

*COMPONENT mux21\_nbit*

*GENERIC (*

*DATA\_WIDTH : INTEGER := 16);*

*PORT (*

*X : IN STD\_LOGIC\_VECTOR(DATA\_WIDTH - 1 DOWNTO 0);*

*Y : IN STD\_LOGIC\_VECTOR(DATA\_WIDTH - 1 DOWNTO 0);*

*s : IN STD\_LOGIC;*

*m : OUT STD\_LOGIC\_VECTOR(DATA\_WIDTH - 1 DOWNTO 0));*

*END COMPONENT mux21\_nbit;*

*-- Signals declaration*

*SIGNAL X\_tb : STD\_LOGIC\_VECTOR(DATA\_WIDTH - 1 DOWNTO 0);*

*SIGNAL Y\_tb : STD\_LOGIC\_VECTOR(DATA\_WIDTH - 1 DOWNTO 0);*

*SIGNAL s\_tb : STD\_LOGIC;*

*SIGNAL m\_tb : STD\_LOGIC\_VECTOR(DATA\_WIDTH - 1 DOWNTO 0);*

*BEGIN*

*-- Instantiate the Unit Under Test (UUT)*

*uut : mux21\_nbit*

*GENERIC MAP(DATA\_WIDTH => DATA\_WIDTH)*

*PORT MAP(*

*X => X\_tb,*

*Y => Y\_tb,*

*s => s\_tb,*

*m => m\_tb);*

*-- Stimulus process*

*stim\_proc : PROCESS*

*BEGIN*

*-- Stimulus 1*

*X\_tb <= "1010101010101010";*

*Y\_tb <= "0101010101010101";*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT m\_tb = X\_tb REPORT "Output mismatch for Stimulus 1" SEVERITY ERROR;*

*-- Stimulus 2*

*X\_tb <= "1111000011110000";*

*Y\_tb <= "0000111100001111";*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT m\_tb = Y\_tb REPORT "Output mismatch for Stimulus 2" SEVERITY ERROR;*

*-- Stimulus 3*

*X\_tb <= "0000000011111111";*

*Y\_tb <= "1111111100000000";*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT m\_tb = X\_tb REPORT "Output mismatch for Stimulus 3" SEVERITY ERROR;*

*-- Stimulus 4*

*X\_tb <= "0101010101010101";*

*Y\_tb <= "1010101010101010";*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT m\_tb = Y\_tb REPORT "Output mismatch for Stimulus 4" SEVERITY ERROR;*

*-- Stimulus 5*

*X\_tb <= (OTHERS => '0');*

*Y\_tb <= (OTHERS => '1');*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT m\_tb = X\_tb REPORT "Output mismatch for Stimulus 5" SEVERITY ERROR;*

*-- Stimulus 6*

*X\_tb <= (OTHERS => '1');*

*Y\_tb <= (OTHERS => '0');*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT m\_tb = Y\_tb REPORT "Output mismatch for Stimulus 6" SEVERITY ERROR;*

*-- Stimulus 7*

*X\_tb <= "1111000000000000";*

*Y\_tb <= "0000111111111111";*

*s\_tb <= '0';*

*WAIT FOR 10 ns;*

*ASSERT m\_tb = X\_tb REPORT "Output mismatch for Stimulus 7" SEVERITY ERROR;*

*-- Stimulus 8*

*X\_tb <= "0000111111111111";*

*Y\_tb <= "1111000000000000";*

*s\_tb <= '1';*

*WAIT FOR 10 ns;*

*ASSERT m\_tb = Y\_tb REPORT "Output mismatch for Stimulus 8" SEVERITY ERROR;*

*-- Add more test cases as needed*

*-- End of stimulus*

*REPORT "Stimulus process finished" SEVERITY NOTE;*

*WAIT;*

*END PROCESS stim\_proc;*

*END ARCHITECTURE behavior*;

Kết quả chạy testbench cùng với waveform được thể hiện như sau

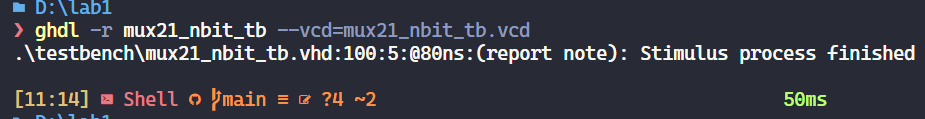


Figure 7 - Kết quả chạy testbench mux21\_nbit\_tb

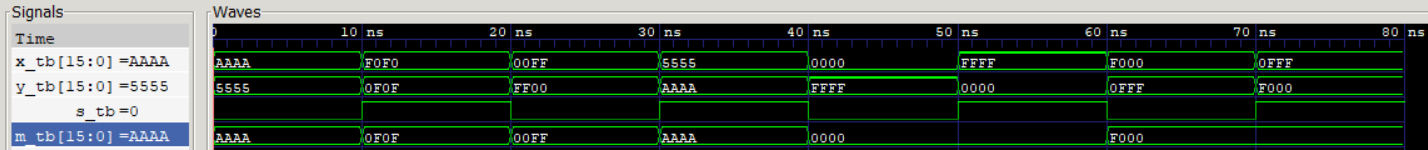


Figure 8 - Waveform của testbench mux21\_nbit\_tb

Dựa theo output của testbench, các test cases đều được PASS và không có trường hợp FAILED.

Dựa theo waveform, output m được lựa chọn đúng them selector bit s

Refactor lại design mux21\_nbit bằng cách dung WHEN-ELSE như sau:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY *mux21\_nbit* IS

  GENERIC (

    DATA\_WIDTH : *INTEGER* := 16);

  PORT (

    X : IN *STD\_LOGIC\_VECTOR*(DATA\_WIDTH - 1 DOWNTO 0);

    Y : IN *STD\_LOGIC\_VECTOR*(DATA\_WIDTH - 1 DOWNTO 0);

    s : IN *STD\_LOGIC*;

    m : OUT *STD\_LOGIC\_VECTOR*(DATA\_WIDTH - 1 DOWNTO 0));

END ENTITY *mux21\_nbit*;

*ARCHITECTURE df OF mux21\_nbit IS*

*COMPONENT mux21 IS*

*PORT (*

*x : IN STD\_LOGIC;*

*y : IN STD\_LOGIC;*

*s : IN STD\_LOGIC;*

*m : OUT STD\_LOGIC);*

*END COMPONENT mux21;*

*SIGNAL m\_temp : STD\_LOGIC\_VECTOR(DATA\_WIDTH - 1 DOWNTO 0);*

*BEGIN -- ARCHITECTURE df*

*WITH s SELECT*

*m\_temp <= X WHEN '0',*

*Y WHEN OTHERS;*

*m <= m\_temp;*

*END ARCHITECTURE df*;

Cách viết design này không thay đổi về behavior và code được clean hơn rất nhiều.

Dựa theo selector bit s thì m\_temp sẽ được gán giá trị tương đương với X khi s = 0 và m\_temp <= ‘0’ khi s = 1.

# Cải tiến & Kết luận

Để thuận tiện cho việc quản lý dự án, cấu trúc của dự án có thể được chia thành 2 thư mục **design** và **testbench.** Như vậy sẽ giúp phân biệt các files cũng như quản lý sự thay đổi dễ dàng hơn.

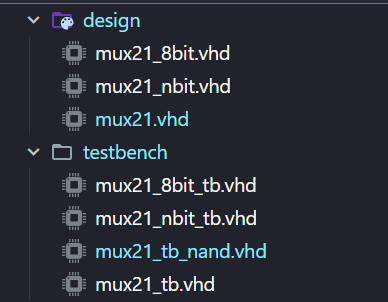
****

Figure 9 - Cấu trúc thư mục dự án sau khi thêm các thư mục design và testbench

Hơn nữa, việc này sẽ giúp cho việc chạy compile nhiều files cùng 1 lúc chỉ với 1 câu lệnh



Figure 10 - Cách chạy compiling tất cả design và testbench với 1 cậu lệnh sử dụng "\*"

Bài lab1 bước đầu đã giúp người học hiểu về việc thiết kế digital với ngôn ngữ VHDL, cách thiết kế và sử dụng testbench trong việc kiểm thử, cách sử dụng waveform để kiểm tra cũng như sử dụng VCS / Verdi / Các công cụ Open Source trong việc thiết kế Front End.