Laboratory 1: Circuit Designs and Testing

Shein Htike Brandon Vasquez

CSC 343 Spring 2025

Contents

1 Exercise A: 8x1 Multiplexer Using Logic Gates

1.1 Objective

The goal of this exercise is to create an 8x1 multiplexer in two ways: using logic gates and using VHDL code.

1.2 Functionality and Specifications

1.2.1 Logic

The output of the 8×1 multiplexer is given by the following logic equation:

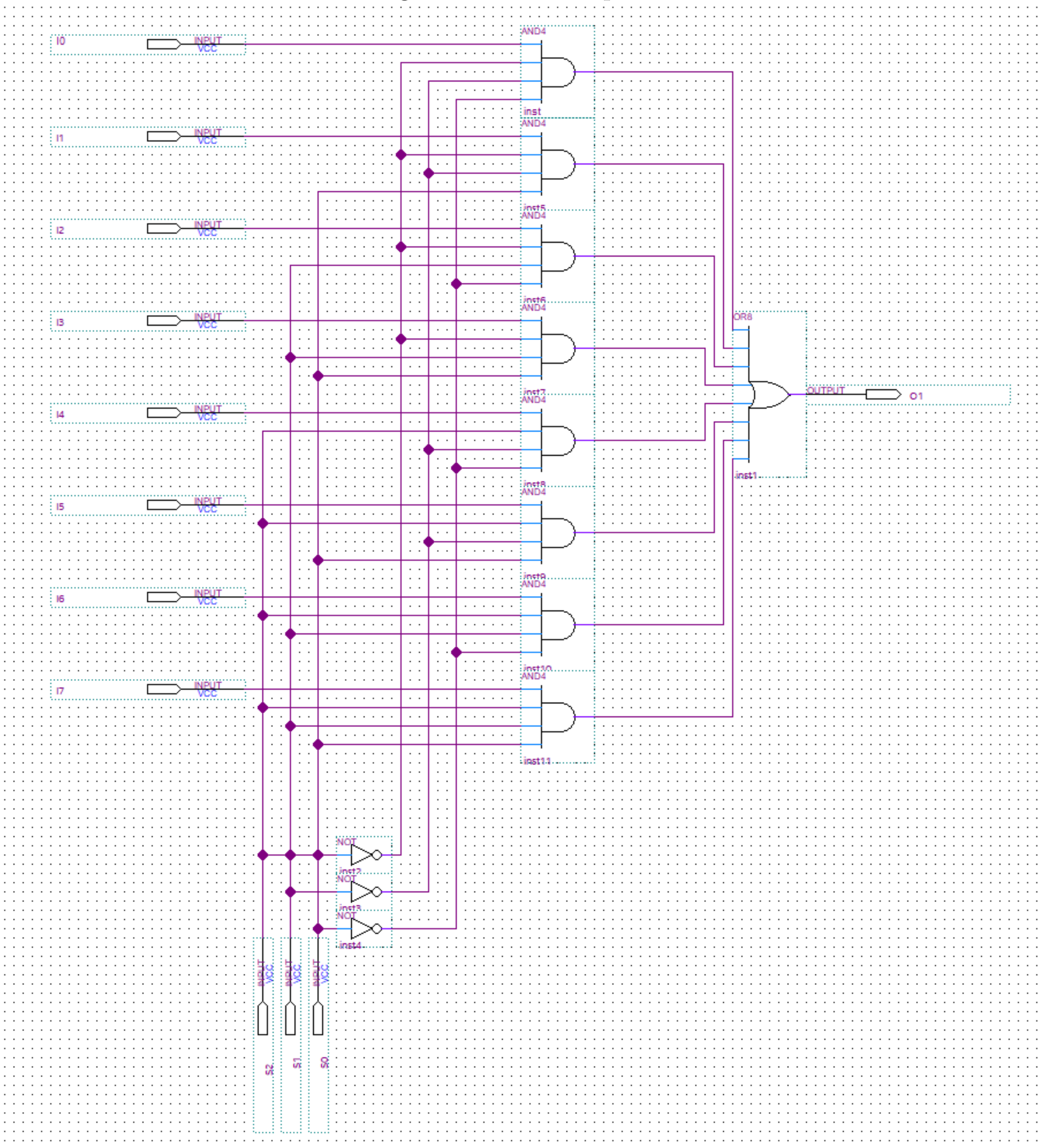
$$\begin{aligned} \text{output} = {} & \overline{S_2}\,\overline{S_1}\,\overline{S_0}\,I_0 \ \lor\ \overline{S_2}\,\overline{S_1}\,S_0\,I_1 \ \lor \\ & \overline{S_2}\,S_1\,\overline{S_0}\,I_2 \ \lor\ \overline{S_2}\,S_1\,S_0\,I_3 \ \lor \\ & S_2\,\overline{S_1}\,\overline{S_0}\,I_4 \ \lor\ S_2\,\overline{S_1}\,S_0\,I_5 \ \lor \\ & S_2\,S_1\,\overline{S_0}\,I_6 \ \lor\ S_2\,S_1\,S_0\,I_7 \end{aligned} \tag{1}$$

In this multiplexer design, we select one of the eight inputs, I_0 through I_7 and connect it to a single output based on the binary value of the three select signals, S_2, S_1, and S_0 (with S_2 being the most significant bit).

1.2.2 Circut Design

In order to implement the 8x1 multiplexer, we created this circuit design in Intel Quartus Prime. This circuit was then compiled into VHDL and imported into ModelSim in order to simulate and test our design.

Figure 1: 8x1 multiplexer



1.2.3 VHDL Code

I also redesigned this multiplexer in VHDL using behavioral modeling.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity multiplexer8x1v2 is
    port (
        I : in  std_logic_vector(7 downto 0);
        S : in  std_logic_vector(2 downto 0);
        O : out std_logic
    );
end multiplexer8x1v2;

architecture Behavioral of multiplexer8x1v2 is
begin
    with S select
        O <= I(0) when "000",
             I(1) when "001",
             I(2) when "010",
             I(3) when "011",
             I(4) when "100",
             I(5) when "101",
             I(6) when "110",
             I(7) when "111",
             '0'    when others;
end Behavioral;
```

1.3 Simulation

I wrote VHDL to simulate both versions of the circuit.

1.3.1 Structural Model Test Bench

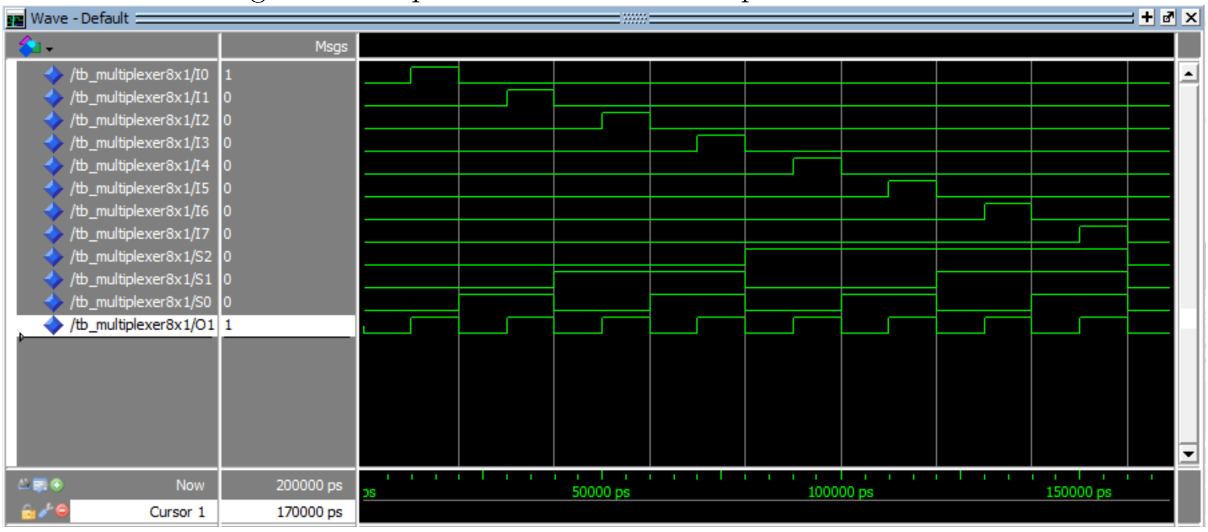
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE std.textio.ALL;

ENTITY tb_multiplexer8x1 IS
END tb_multiplexer8x1;

ARCHITECTURE test OF tb_multiplexer8x1 IS
    SIGNAL I0 : STD_LOGIC := '0';
    SIGNAL I1 : STD_LOGIC := '0';
    SIGNAL I2 : STD_LOGIC := '0';
    SIGNAL I3 : STD_LOGIC := '0';
    SIGNAL I4 : STD_LOGIC := '0';
    SIGNAL I5 : STD_LOGIC := '0';
    SIGNAL I6 : STD_LOGIC := '0';
    SIGNAL I7 : STD_LOGIC := '0';
    SIGNAL S2 : STD_LOGIC := '0';
    SIGNAL S1 : STD_LOGIC := '0';
    SIGNAL S0 : STD_LOGIC := '0';
    SIGNAL O1 : STD_LOGIC;

    COMPONENT mux8to1_structural IS
        PORT (
            I0 : IN STD_LOGIC;
            I2 : IN STD_LOGIC;
            I3 : IN STD_LOGIC;
            I1 : IN STD_LOGIC;
            I4 : IN STD_LOGIC;
            I5 : IN STD_LOGIC;
            I6 : IN STD_LOGIC;
            I7 : IN STD_LOGIC;
            S2 : IN STD_LOGIC;
            S1 : IN STD_LOGIC;
            S0 : IN STD_LOGIC;
            O1 : OUT STD_LOGIC
        );
    END COMPONENT;

BEGIN
    uut : mux8to1_structural
    PORT MAP(I0, I2, I3, I1, I4, I5, I6, I7, S2, S1, S0, O1);
```

```
PROCESS
BEGIN
    S2 <= '0';
    S1 <= '0';
    S0 <= '0';
    WAIT FOR 10 ns;
    I0 <= '1';
    WAIT FOR 10 ns;
    I0 <= '0';

    S2 <= '0';
    S1 <= '0';
    S0 <= '1';
    WAIT FOR 10 ns;
    I1 <= '1';
    WAIT FOR 10 ns;
    I1 <= '0';

    S2 <= '0';
    S1 <= '1';
    S0 <= '0';
    WAIT FOR 10 ns;
    I2 <= '1';
    WAIT FOR 10 ns;
    I2 <= '0';

    S2 <= '0';
    S1 <= '1';
    S0 <= '1';
    WAIT FOR 10 ns;
    I3 <= '1';
    WAIT FOR 10 ns;
    I3 <= '0';

    S2 <= '1';
    S1 <= '0';
    S0 <= '0';
    WAIT FOR 10 ns;
    I4 <= '1';
    WAIT FOR 10 ns;
    I4 <= '0';

    S2 <= '1';
    S1 <= '0';
    S0 <= '1';
    WAIT FOR 10 ns;
    I5 <= '1';
    WAIT FOR 10 ns;
```

```
        I5 <= '0';

        S2 <= '1';
        S1 <= '1';
        S0 <= '0';
        WAIT FOR 10 ns;
        I6 <= '1';
        WAIT FOR 10 ns;
        I6 <= '0';

        S2 <= '1';
        S1 <= '1';
        S0 <= '1';
        WAIT FOR 10 ns;
        I7 <= '1';
        WAIT FOR 10 ns;
        I7 <= '0';
    END PROCESS;
END test;
```

This code selects inputs 0 through 7 and toggles them while they are selected to show that the output corresponds to the selected input.

Figure 2: Output for Structural Multiplexer Test Bench



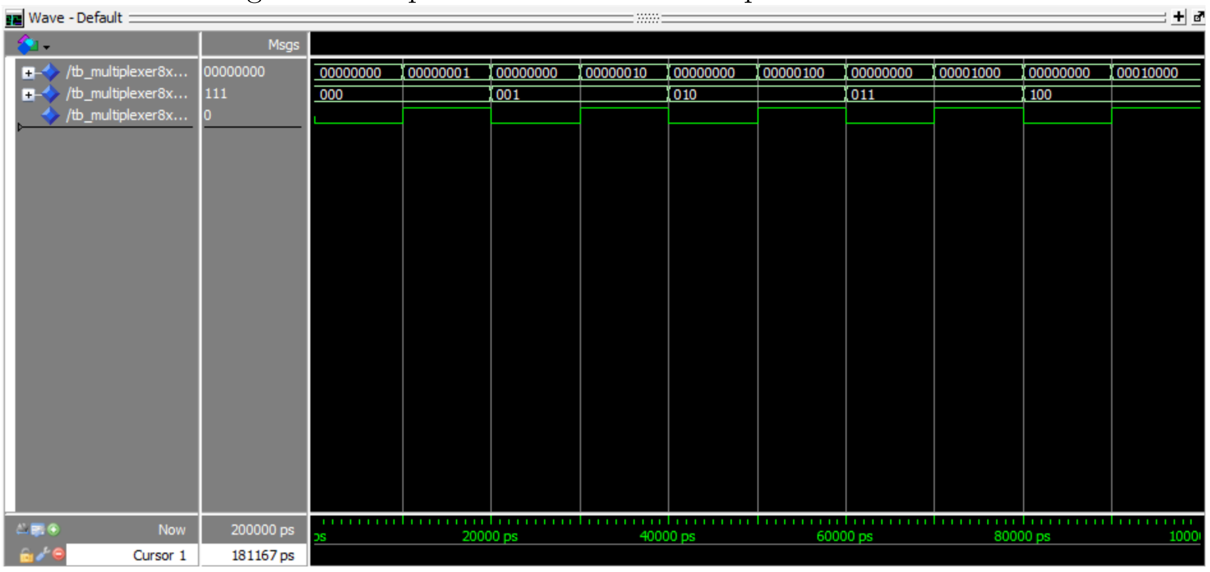
```
/tb_multiplexer8x1/I0   1
/tb_multiplexer8x1/I1   0
/tb_multiplexer8x1/I2   0
/tb_multiplexer8x1/I3   0
/tb_multiplexer8x1/I4   0
/tb_multiplexer8x1/I5   0
/tb_multiplexer8x1/I6   0
/tb_multiplexer8x1/I7   0
/tb_multiplexer8x1/S2   0
/tb_multiplexer8x1/S1   0
/tb_multiplexer8x1/S0   0
/tb_multiplexer8x1/O1   1
Now        200000 ps
Cursor 1   170000 ps
```

1.3.2 Behavioral Model Test Bench

I also wrote test bench code for the behavioral model multiplexer. This code was essentially the same as the previous test bench for the structural model. The only difference is that the inputs and the select signals were vectors.

```
    LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY tb_multiplexer8x1v2 IS
END tb_multiplexer8x1v2;

ARCHITECTURE test OF tb_multiplexer8x1v2 IS
    SIGNAL I : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
    SIGNAL S : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000";
    SIGNAL O : STD_LOGIC;
    COMPONENT multiplexer8x1v2 IS
        PORT (
            I : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
            S : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
            O : OUT STD_LOGIC
        );
    END COMPONENT;
BEGIN
    uut : multiplexer8x1v2
        PORT MAP (
            I => I,
            S => S,
            O => O
        );
    PROCESS
    BEGIN
        S <= "000";
        I <= (OTHERS => '0');
        WAIT FOR 10 ns;
        I(0) <= '1';
        WAIT FOR 10 ns;
        I(0) <= '0';
        S <= "001";
        I <= (OTHERS => '0');
        WAIT FOR 10 ns;
        I(1) <= '1';
        WAIT FOR 10 ns;
        I(1) <= '0';
        S <= "010";
        I <= (OTHERS => '0');
        WAIT FOR 10 ns;
        I(2) <= '1';
        WAIT FOR 10 ns;
        I(2) <= '0';
        S <= "011";
        I <= (OTHERS => '0');
        WAIT FOR 10 ns;
        I(3) <= '1';
        WAIT FOR 10 ns;
        I(3) <= '0';
        S <= "100";
        I <= (OTHERS => '0');
        WAIT FOR 10 ns;
        I(4) <= '1';
        WAIT FOR 10 ns;
        I(4) <= '0';
        S <= "101";
        I <= (OTHERS => '0');
        WAIT FOR 10 ns;
        I(5) <= '1';
        WAIT FOR 10 ns;
        I(5) <= '0';
        S <= "110";
        I <= (OTHERS => '0');
        WAIT FOR 10 ns;
        I(6) <= '1';
        WAIT FOR 10 ns;
        I(6) <= '0';
        S <= "111";
        I <= (OTHERS => '0');
        WAIT FOR 10 ns;
        I(7) <= '1';
        WAIT FOR 10 ns;
        I(7) <= '0';
        WAIT;
    END PROCESS;
END test;
```

Figure 3: Output for Behavioral Multiplexer Test Bench



2 Exercise B: 1x8 De-Multiplexer Using 1x4 and 1x2 De-Multiplexers

2.1 Objective

2.2 Functionality and Specifications

2.3 Simulation

3 Exercise C: 3-to-8 Decoder

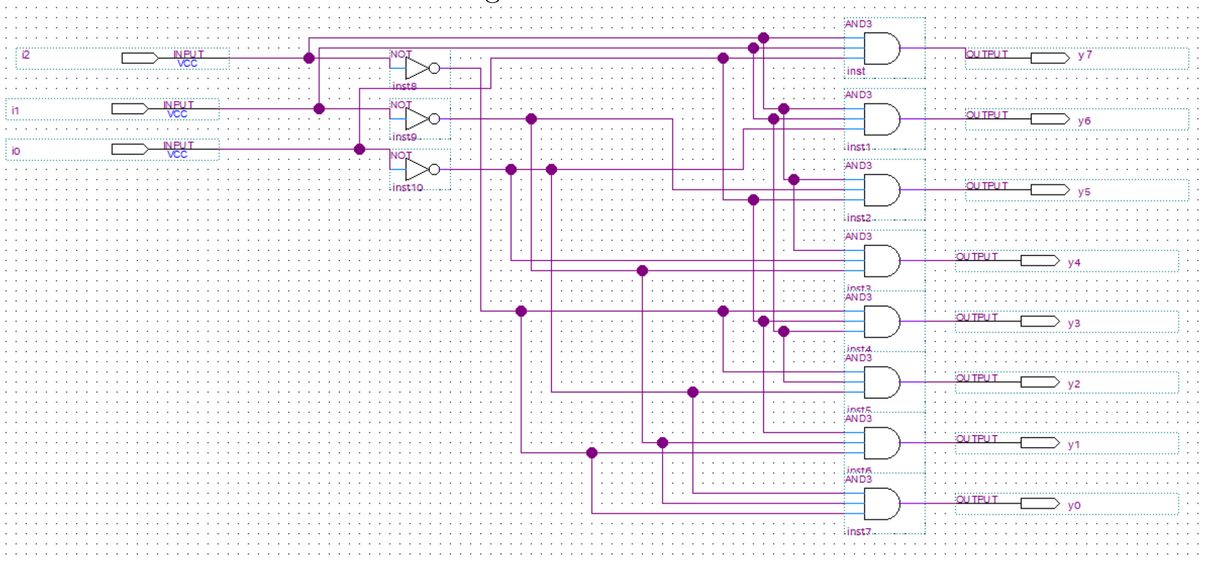
3.1 Objective

The purpose of this exercise was to create a 3x8 decoder circuit from its respective combinational logic. Now having compiled and generated the relevant VHDL for the circuit one was able to simulate the circuit in action.

3.2 Functionality and Specifications

Here we want to describe the combinational logic functions (Boolean equations, if applicable)

Figure 4: 3x8 decoder



3.2.1 VHDL Code generated from the design file

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY lab_assign_1 IS
        PORT
        (
                i2 :  IN  STD_LOGIC;
                i1 :  IN  STD_LOGIC;
                i0 :  IN  STD_LOGIC;
                y7 :  OUT  STD_LOGIC;
                y6 :  OUT  STD_LOGIC;
                y5 :  OUT  STD_LOGIC;
                y4 :  OUT  STD_LOGIC;
                y3 :  OUT  STD_LOGIC;
                y2 :  OUT  STD_LOGIC;
                y1 :  OUT  STD_LOGIC;
                y0 :  OUT  STD_LOGIC
        );
END lab_assign_1;

ARCHITECTURE bdf_type OF lab_assign_1 IS

SIGNAL        SYNTHESIZED_WIRE_12 :  STD_LOGIC;
SIGNAL        SYNTHESIZED_WIRE_13 :  STD_LOGIC;
SIGNAL        SYNTHESIZED_WIRE_14 :  STD_LOGIC;


BEGIN

y7 <= i2 AND i1 AND i0;
y6 <= i2 AND i1 AND SYNTHESIZED_WIRE_12;

SYNTHESIZED_WIRE_12 <= NOT(i0);
y5 <= i2 AND SYNTHESIZED_WIRE_13 AND i0;
y4 <= i2 AND SYNTHESIZED_WIRE_12 AND SYNTHESIZED_WIRE_13;
y3 <= SYNTHESIZED_WIRE_14 AND i0 AND i1;
y2 <= SYNTHESIZED_WIRE_14 AND i1 AND SYNTHESIZED_WIRE_12;
y1 <= i0 AND SYNTHESIZED_WIRE_13 AND SYNTHESIZED_WIRE_14;
y0 <= SYNTHESIZED_WIRE_12 AND SYNTHESIZED_WIRE_13 AND SYNTHESIZED_WIRE_14;

SYNTHESIZED_WIRE_14 <= NOT(i2);
SYNTHESIZED_WIRE_13 <= NOT(i1);
END bdf_type;
```

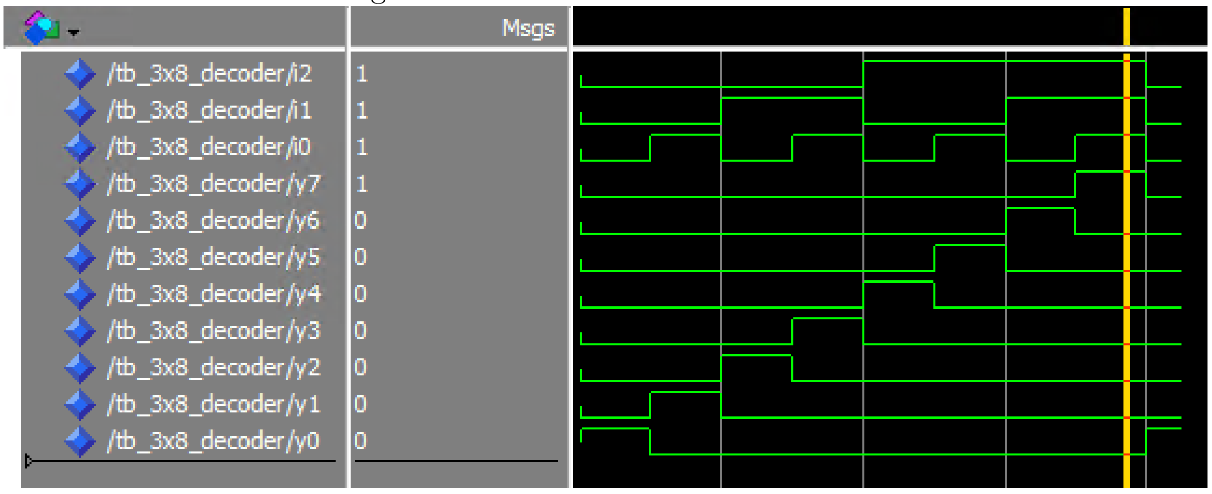
3.2.2 VHDL code for Test bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use std.textio.all;

entity tb_3x8_decoder is
end tb_3x8_decoder;

architecture test of tb_3x8_decoder is
    signal i2 :  STD_LOGIC;
    signal i1 :  STD_LOGIC;
    signal i0 :  STD_LOGIC;
    signal y7 :  STD_LOGIC;
    signal y6 :  STD_LOGIC;
    signal y5 :  STD_LOGIC;
    signal y4 :  STD_LOGIC;
    signal y3 :  STD_LOGIC;
    signal y2 :  STD_LOGIC;
    signal y1 :  STD_LOGIC;
    signal y0 :  STD_LOGIC;

    component lab_assign_1
        Port
        (
                i2 :  IN  STD_LOGIC;
                i1 :  IN  STD_LOGIC;
                i0 :  IN  STD_LOGIC;
                y7 :  OUT  STD_LOGIC;
                y6 :  OUT  STD_LOGIC;
                y5 :  OUT  STD_LOGIC;
                y4 :  OUT  STD_LOGIC;
                y3 :  OUT  STD_LOGIC;
                y2 :  OUT  STD_LOGIC;
                y1 :  OUT  STD_LOGIC;
                y0 :  OUT  STD_LOGIC
        );
    end component;

begin
    uut: lab_assign_1 port map(i2, i1, i0, y7, y6, y5, y4, y3, y2, y1,y0);
    PROCESS
    BEGIN
        i2 <= '0';
        i1 <= '0';
        i0 <= '0';
        WAIT for 10 ns;
        i0 <= '1';
        WAIT for 10 ns;
        i1 <= '1';
        i0 <= '0';
        WAIT for 10 ns;
        i0 <= '1';
        WAIT for 10 ns;
        i2 <= '1';
        i1 <= '0';
        i0 <= '0';
        WAIT for 10 ns;
        i0 <= '1';
        WAIT for 10 ns;
        i1 <= '1';
        i0 <= '0';
        WAIT for 10 ns;
        i0 <= '1';
```

```
            WAIT for 10 ns;
        END PROCESS;

    end test;
```

3.3 Simulation

Figure 5: 3x8 decoder wave simulation



```
Msgs
/tb_3x8_decoder/i2  1
/tb_3x8_decoder/i1  1
/tb_3x8_decoder/i0  1
/tb_3x8_decoder/y7  1
/tb_3x8_decoder/y6  0
/tb_3x8_decoder/y5  0
/tb_3x8_decoder/y4  0
/tb_3x8_decoder/y3  0
/tb_3x8_decoder/y2  0
/tb_3x8_decoder/y1  0
/tb_3x8_decoder/y0  0
```

4 Exercise D: 8-to-3 Priority Encoder

4.1 Objective

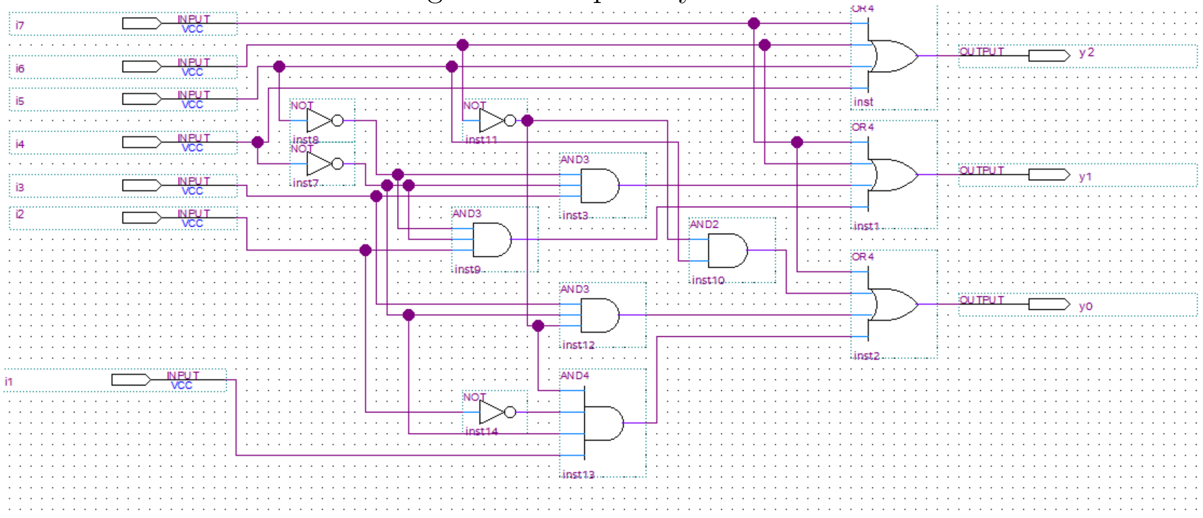
The purpose of this exercise was to create a 8x3 priority encoder circuit from its respective combinational logic. Now having compiled and generated the relevant VHDL for the circuit one was able to simulate the circuit in action. Thus resulting in a better understanding of designing a 8x3 priority encoder circuit from its respective combinational equation, and testing that created circuit.

4.2 Functionality and Specifications

$$Y_2 = D_7 + D_6 + D_5 + D_4,$$
$$Y_1 = D_7 + D_6 + \overline{D_7 + D_6}\,(D_3 + D_2),$$
$$Y_0 = D_7 + \overline{D_7 + D_6}\,D_5 + \overline{D_7 + D_6 + D_5}\,D_3 + \overline{D_7 + D_6 + D_5 + D_4}\,D_1.$$

Here we want to describe the combinational logic functions (our equations essentially)

Figure 6: 8x3 priority encoder



VHDL code generated from design file

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY \8x3_p_enc\ IS
        PORT
        (
                i2 :  IN  STD_LOGIC;
                i1 :  IN  STD_LOGIC;
                i3 :  IN  STD_LOGIC;
                i4 :  IN  STD_LOGIC;
                i5 :  IN  STD_LOGIC;
                i6 :  IN  STD_LOGIC;
                i7 :  IN  STD_LOGIC;
                y2 :  OUT  STD_LOGIC;
                y1 :  OUT  STD_LOGIC;
                y0 :  OUT  STD_LOGIC
        );
END \8x3_p_enc\;

ARCHITECTURE bdf_type OF \8x3_p_enc\ IS

SIGNAL        SYNTHESIZED_WIRE_0 :  STD_LOGIC;
SIGNAL        SYNTHESIZED_WIRE_1 :  STD_LOGIC;
SIGNAL        SYNTHESIZED_WIRE_15 :  STD_LOGIC;
SIGNAL        SYNTHESIZED_WIRE_16 :  STD_LOGIC;
SIGNAL        SYNTHESIZED_WIRE_6 :  STD_LOGIC;
SIGNAL        SYNTHESIZED_WIRE_8 :  STD_LOGIC;
SIGNAL        SYNTHESIZED_WIRE_9 :  STD_LOGIC;
SIGNAL        SYNTHESIZED_WIRE_10 :  STD_LOGIC;
SIGNAL        SYNTHESIZED_WIRE_17 :  STD_LOGIC;


BEGIN



y2 <= i7 OR i5 OR i4 OR i6;


y1 <= i7 OR SYNTHESIZED_WIRE_0 OR SYNTHESIZED_WIRE_1 OR i6;


SYNTHESIZED_WIRE_10 <= SYNTHESIZED_WIRE_15 AND i5;


SYNTHESIZED_WIRE_15 <= NOT(i6);



SYNTHESIZED_WIRE_8 <= i3 AND SYNTHESIZED_WIRE_16 AND SYNTHESIZED_WIRE_15;


SYNTHESIZED_WIRE_9 <= SYNTHESIZED_WIRE_15 AND SYNTHESIZED_WIRE_6 AND SYNTHESIZED_


SYNTHESIZED_WIRE_6 <= NOT(i2);
```

```
y0 <= i7 OR SYNTHESIZED_WIRE_8 OR SYNTHESIZED_WIRE_9 OR SYNTHESIZED_WIRE_10;


SYNTHESIZED_WIRE_0 <= SYNTHESIZED_WIRE_17 AND SYNTHESIZED_WIRE_16 AND i3;


SYNTHESIZED_WIRE_16 <= NOT(i4);



SYNTHESIZED_WIRE_17 <= NOT(i5);



SYNTHESIZED_WIRE_1 <= SYNTHESIZED_WIRE_17 AND SYNTHESIZED_WIRE_16 AND i2;


END bdf_type;
```

4.2.1 VHDL code for Test bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use std.textio.all;

entity tb_8x3_p_encoder is
end tb_8x3_p_encoder;

architecture test of tb_8x3_p_encoder is
    signal i1 : STD_LOGIC;
    signal i2 : STD_LOGIC;
    signal i3 : STD_LOGIC;
    signal i4 : STD_LOGIC;
    signal i5 : STD_LOGIC;
    signal i6 : STD_LOGIC;
    signal i7 : STD_LOGIC;
    signal y2 : STD_LOGIC;
    signal y1 : STD_LOGIC;
    signal y0 : STD_LOGIC;

    component \8x3_p_enc\
        Port ( i1 : in STD_LOGIC;
               i2 : in STD_LOGIC;
               i3 : in STD_LOGIC;
               i4 : in STD_LOGIC;
```

```
                i5 : in STD_LOGIC;
                i6 : in STD_LOGIC;
                i7 : in STD_LOGIC;
                y2 : out STD_LOGIC;
                y1 : out STD_LOGIC;
                y0 : out STD_LOGIC);
    end component;

begin
    uut: \8x3_p_enc\ port map(i1, i2, i3, i4, i5, i6, i7, y2, y1, y0);
    PROCESS
    BEGIN
        -- No inputs active
        i1 <= '0';
        i2 <= '0';
        i3 <= '0';
        i4 <= '0';
        i5 <= '0';
        i6 <= '0';
        i7 <= '0';
        WAIT for 20 ns;
        -- i1 active
        i1 <= '1';
        i2 <= '0';
        i3 <= '0';
        i4 <= '0';
        i5 <= '0';
        i6 <= '0';
        i7 <= '0';
        WAIT for 20 ns;
        -- i2 active
        i1 <= '0';
        i2 <= '1';
        i3 <= '0';
        i4 <= '0';
        i5 <= '0';
        i6 <= '0';
        i7 <= '0';
        WAIT for 20 ns;
        -- i3 active
        i1 <= '0';
        i2 <= '0';
        i3 <= '1';
        i4 <= '0';
        i5 <= '0';
        i6 <= '0';
        i7 <= '0';
        WAIT for 20 ns;
```

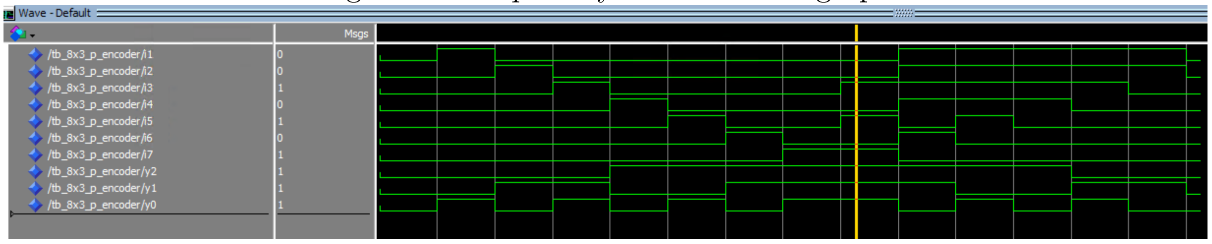
```
-- i4 active
i1 <= '0';
i2 <= '0';
i3 <= '0';
i4 <= '1';
i5 <= '0';
i6 <= '0';
i7 <= '0';
WAIT for 20 ns;
-- i5 active
i1 <= '0';
i2 <= '0';
i3 <= '0';
i4 <= '0';
i5 <= '1';
i6 <= '0';
i7 <= '0';
WAIT for 20 ns;
-- i6 active
i1 <= '0';
i2 <= '0';
i3 <= '0';
i4 <= '0';
i5 <= '0';
i6 <= '1';
i7 <= '0';
WAIT for 20 ns;
-- i7 active
i1 <= '0';
i2 <= '0';
i3 <= '0';
i4 <= '0';
i5 <= '0';
i6 <= '0';
i7 <= '1';
WAIT for 20 ns;
-- multiple active, output should be: (Highest priority, i7)
i1 <= '0';
i2 <= '0';
i3 <= '1';
i4 <= '0';
i5 <= '1';
i6 <= '0';
i7 <= '1';
WAIT for 20 ns;
-- multiple active, output should be: (Highest priority, i6)
i1 <= '1';
i2 <= '1';
```

```
        i3 <= '1';
        i4 <= '1';
        i5 <= '0';
        i6 <= '1';
        i7 <= '0';
        WAIT for 20 ns;
        -- multiple active, output should be: (Highest priority, i5)
        i1 <= '1';
        i2 <= '1';
        i3 <= '1';
        i4 <= '1';
        i5 <= '1';
        i6 <= '0';
        i7 <= '0';
        WAIT for 20 ns;
        -- multiple active, output should be: (Highest priority, i4)
        i1 <= '1';
        i2 <= '1';
        i3 <= '1';
        i4 <= '1';
        i5 <= '0';
        i6 <= '0';
        i7 <= '0';
        WAIT for 20 ns;
        -- multiple active, output should be: (Highest priority, i3)
        i1 <= '1';
        i2 <= '1';
        i3 <= '1';
        i4 <= '0';
        i5 <= '0';
        i6 <= '0';
        i7 <= '0';
        WAIT for 20 ns;
        -- multiple active, output should be: (Highest priority, i2)
        i1 <= '1';
        i2 <= '1';
        i3 <= '0';
        i4 <= '0';
        i5 <= '0';
        i6 <= '0';
        i7 <= '0';
        WAIT for 20 ns;

    END PROCESS;

end test;
```

4.3 Simulation

In this simulation part regarding the (Draw truth tables based on the waveforms.) part we just compare the wave form against the

Figure 7: 8x3 priority encoder wave graph



```
/tb_8x3_p_encoder/i1   0
/tb_8x3_p_encoder/i2   0
/tb_8x3_p_encoder/i3   1
/tb_8x3_p_encoder/i4   0
/tb_8x3_p_encoder/i5   1
/tb_8x3_p_encoder/i6   0
/tb_8x3_p_encoder/i7   1
/tb_8x3_p_encoder/y2   1
/tb_8x3_p_encoder/y1   1
/tb_8x3_p_encoder/y0   1
```

5 Exercise E: Set-Reset Flip-Flop & D Flip-Flop (Positive Edge Trigger)

5.1 Objective

Up until now we've modeled both the set reset flip-flop and the D flip-flop using logic gates. The goal of this section was to instead model both flip-flops behaviorally in VHDL.

5.2 Functionality and Specifications

5.2.1 Set-Reset Flip-Flop Truth Table

The following truth table describes the behavior of our SR flip-flop.

CLK	R	S	Q	$\overline{Q}$
0	X	X	NO CHG	NO CHG
1	X	X	NO CHG	NO CHG
$\downarrow$	X	X	NO CHG	NO CHG
$\uparrow$	0	0	NO CHG	NO CHG
$\uparrow$	0	1	SET	RESET
$\uparrow$	1	0	RESET	SET
$\uparrow$	1	1	ILLEGAL	ILLEGAL

During the rising edge of the clock when the clock transitions from 0 to 1, the outputs Q and $\overline{Q}$ update according to these rules:

- If $S = 1$, then the flip-flop is set: Q becomes 1 and $\overline{Q}$ becomes 0.
- If $R = 1$, then the flip-flop is reset: Q becomes 0 and $\overline{Q}$ becomes 1.
- If both R and S are inactive (i.e., $R = 0$ and $S = 0$), the outputs remain unchanged.
- The condition where both $R = 1$ and $S = 1$ is considered illegal for this circuit.

5.2.2 D Flip-Flop Truth Table

The following truth table illustrates the behavior of the D flip-flop:

CLK	D	Q
0	X	NO CHG
1	X	NO CHG
$\downarrow$	X	NO CHG
$\uparrow$	0	RESET
$\uparrow$	1	SET

Here, Q changes to be equal to D whenever the clock is rising.

5.2.3 Set-Reset Flip-Flop VHDL Code

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

ENTITY sr_flip_flop IS
        PORT (
                s : IN STD_LOGIC;
                r : IN STD_LOGIC;
                clk : IN STD_LOGIC;
                q : OUT STD_LOGIC;
                qc : OUT STD_LOGIC
        );
END sr_flip_flop;

ARCHITECTURE behavior OF sr_flip_flop IS
        SIGNAL S_Q : STD_LOGIC;
BEGIN
        q <= S_Q;
        qc <= NOT S_Q;
        PROCESS (clk)
        BEGIN
                IF rising_edge(clk) THEN
                        IF s = '1' AND r = '0' THEN
                                S_Q <= '1';
                        ELSIF s = '0' AND r = '1' THEN
                                S_Q <= '0';
                        END IF;
                END IF;
        END PROCESS;
END behavior;
```

5.2.4 D Flip-Flop VHDL Code

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

entity d_flip_flop is
    Port (
        D   : in STD_LOGIC;   -- Data input
        CLK : in STD_LOGIC;   -- Clock signal
        Q   : out STD_LOGIC   -- Output
    );
end d_flip_flop;

architecture Behavioral of d_flip_flop is
begin
    process(CLK)
    begin
        if rising_edge(CLK) then
            Q <= D;
        end if;
    end process;
end Behavioral;
```

5.3 Simulation

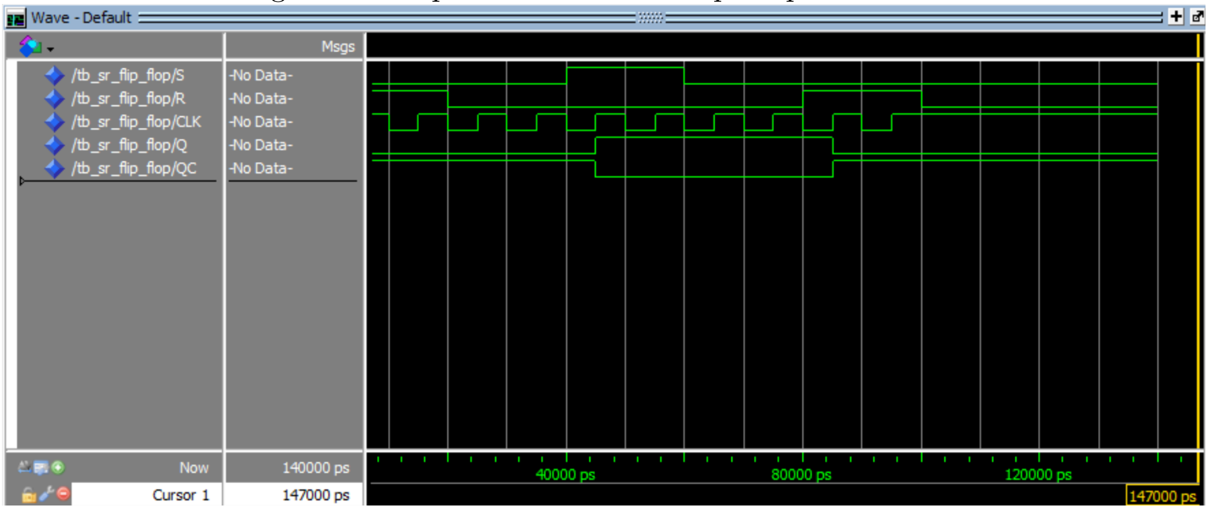
5.3.1 Set-Reset Flip-Flop Test Bench

The following is the test match code for the set reset flip-flop.

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY tb_sr_flip_flop IS
END tb_sr_flip_flop;

ARCHITECTURE testbench OF tb_sr_flip_flop IS
    SIGNAL S : STD_LOGIC := '0'; -- Signal for data input
    SIGNAL R : STD_LOGIC := '0'; -- Signal for data input
    SIGNAL CLK : STD_LOGIC := '0'; -- Clock signal
    SIGNAL Q : STD_LOGIC; -- Output signal
    SIGNAL QC : STD_LOGIC; -- Output signal
BEGIN
    -- Instantiate the SR Flip-Flop
    uut : ENTITY work.sr_flip_flop
        PORT MAP(S, R, CLK, Q, QC);

        -- Clock Generation Process
        CLK_Process : PROCESS
        BEGIN
            WHILE now < 100 ns LOOP -- Limit the clock process to 100 ns
            CLK <= '0';
            WAIT FOR 5 ns;
            CLK <= '1';
            WAIT FOR 5 ns;
        END LOOP;
        WAIT; -- Stops the clock process after 100 ns
        END PROCESS;

        -- Test Process
        Stimulus_Process : PROCESS
        BEGIN
            R <= '1';
            WAIT FOR 20 ns;
            R <= '0';
            WAIT FOR 20 ns;

            S <= '1';
            WAIT FOR 20 ns;
            S <= '0';
            WAIT FOR 20 ns;

            R <= '1';
            WAIT FOR 20 ns;
            R <= '0';
            WAIT FOR 20 ns;

            WAIT;


        END PROCESS;
END testbench;
```

Here is the output for this code. In the waveform, you can see that the output for Q and QC only get updated on the rising edge of the clock when either S or R are set according to the rules outlined earlier.

Figure 8: Output for Set-Reset Flip-Flop Test Bench

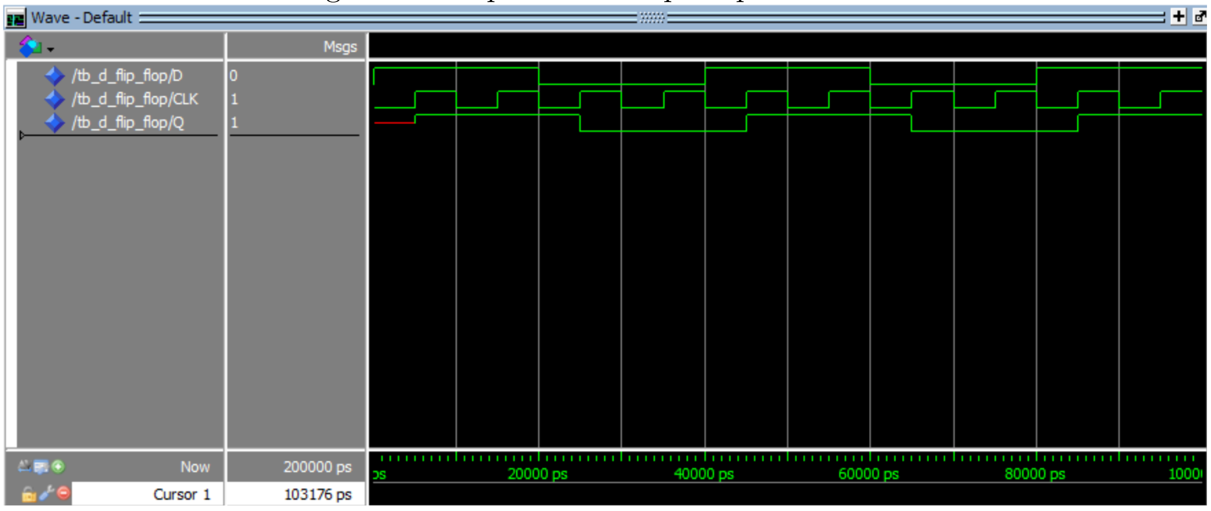


5.3.2 D Flip-Flop Test Bench

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

entity tb_d_flip_flop is
end tb_d_flip_flop;

architecture Testbench of tb_d_flip_flop is
    signal D   : STD_LOGIC := '0'; -- Signal for data input
    signal CLK : STD_LOGIC := '0'; -- Clock signal
    signal Q   : STD_LOGIC;        -- Output signal
begin
    -- Instantiate the D Flip-Flop
    uut: entity work.d_flip_flop
        Port Map ( D, CLK, Q );

    -- Clock Generation Process
    CLK_Process: process
    begin
        while now < 100 ns loop  -- Limit the clock process to 100 ns
            CLK <= '0';
            wait for 5 ns;
            CLK <= '1';
            wait for 5 ns;
        end loop;
        wait; -- Stops the clock process after 100 ns
    end process;

    -- Test Process
    Stimulus_Process: process
    begin
        D <= '1';
        wait for 20 ns;
        D <= '0';
        wait for 20 ns;

        D <= '1';
        wait for 20 ns;
        D <= '0';
        wait for 20 ns;

        D <= '1';
        wait for 20 ns;
        D <= '0';
        wait for 20 ns;

        wait;

    end process;
end Testbench;
```

This code results in the waveform shown in Figure 9.

Figure 9: Output for D Flip-Flop test bench.



Here, the value of Q gets updated to match the value of D whenever the clock rises.

5.4 Differences between the Set-Reset Flip-Flop and the D Flip-Flop

The D flip-flop is very simple: on a rising clock edge,

$$Q \leftarrow D.$$

In contrast, the SR flip-flop responds on the rising clock edge as follows:

- If $S = 1$, then $Q = 1$ and $\overline{Q} = 0$.
- If $R = 1$, then $Q = 0$ and $\overline{Q} = 1$.
- If $S = 0$ and $R = 0$, then Q remains unchanged.

6 Conclusions

We learned about the VHDL language, including data types, input/output/intermediate signals, and its parallel execution model. We observed parallel computation outside processes and sequential operations within processes.

Initially, we struggled with `when` and `if` statements, learning through experimentation that `when` is for outside processes and `if` for inside processes. We also learned processes re-evaluate outputs only when specified inputs change.

We recognized that VHDL's standard logic datatype can hold values other than '0' and '1' such as uninitialized, unknown, and 'don't care' states, which can allow the compiler to create more efficient designs.

We also learned the difference between structural vs. behavioral modeling. Structural modeling is akin to wiring components inside of code, while behavioral modeling describes logic for the compiler to implement.

Challenges included ModelSim and Intel Quartus' project structure, making file management cumbersome. ModelSim's file update behavior, requiring manual saving and recompilation unlike IDEs like IntelliJ IDEA, caused inefficiencies and frustration as well.