ECE 4300
Discrete System Design Using VHDL

Midterm 2

Choi Tim Antony Yung (ID:013499993)

April 14, 2021

1 Circuit from RTL Schematic

The addition block and the increment block can be modeled with the + operator given that appropriate library be included and that the type is a vector. The multiplexer can be modeled with a case statement. The two registers can be modeled with a process block.

Code

4x1 Multiplexer

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

ENTITY mux4x1 IS
    PORT (
        x : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        sel : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
        y : OUT STD_LOGIC
    );
END mux4x1;

ARCHITECTURE Behavioral OF mux4x1 IS

BEGIN

    PROCESS (x, sel)
    BEGIN
        CASE sel IS
            WHEN "00" => y <= x(0);
            WHEN "01" => y <= x(1);
            WHEN "10" => y <= x(2);
            WHEN "11" => y <= x(3);
        END CASE;
    END PROCESS;

END Behavioral;
```

Main Circuit

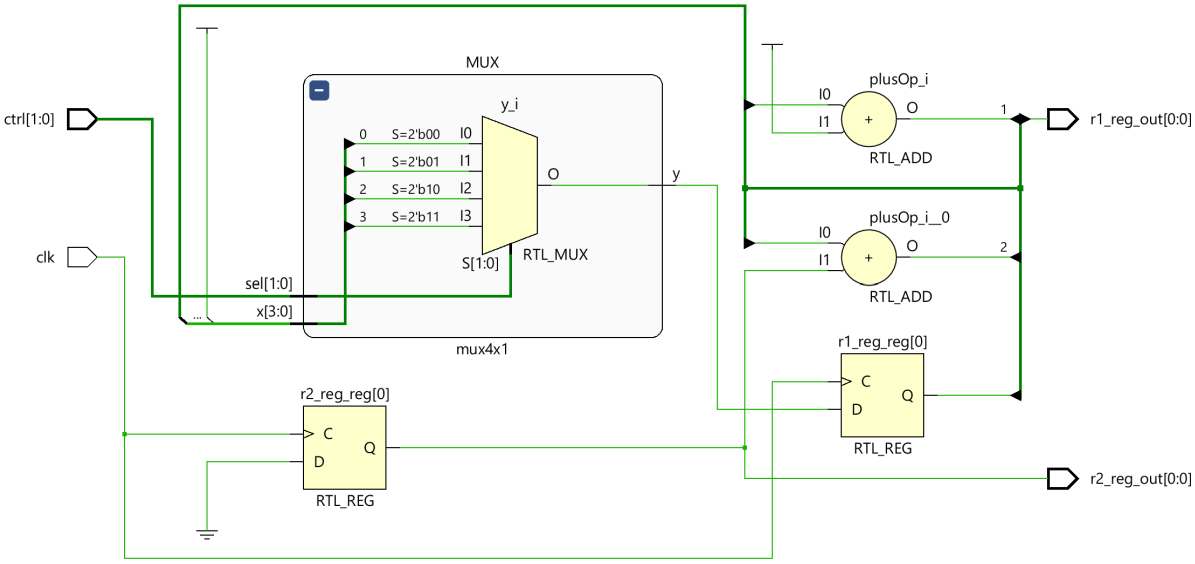
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY ECE4304_Midterm2_1 IS
    PORT (
        clk : IN STD_LOGIC;
        ctrl : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        r1_reg_out : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
        r2_reg_out : OUT STD_LOGIC_VECTOR(0 DOWNTO 0)
    );
END ECE4304_Midterm2_1;
```

```
ARCHITECTURE Behavioral OF ECE4304_Midterm2_1 IS

    SIGNAL r1_reg : STD_LOGIC_VECTOR(0 DOWNTO 0);
    SIGNAL r1_next : STD_LOGIC_VECTOR(0 DOWNTO 0);
    SIGNAL r2_reg : STD_LOGIC_VECTOR(0 DOWNTO 0);

    COMPONENT mux4x1 IS
        PORT (
            x : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            sel : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
            y : OUT STD_LOGIC
        );
    END COMPONENT;

    SIGNAL sum : STD_LOGIC_VECTOR(0 DOWNTO 0);
    SIGNAL inc : STD_LOGIC_VECTOR(0 DOWNTO 0);

BEGIN

    -- mux here
    MUX : mux4x1
    PORT MAP(
        x => r1_reg & sum & inc & "1",
        sel => ctrl,
        y => r1_next(0)
    );

    -- arithmetics (+ and +1)
    sum <= r1_reg + r2_reg;
    inc <= r1_reg + "1";

    -- clock r1 and r2
    PROCESS (clk)
    BEGIN
        IF (rising_edge(clk)) THEN
            r1_reg <= r1_next;
            r2_reg <= "Z";
        END IF;
    END PROCESS;

    -- produce output to produce RTL schematic
    r1_reg_out <= r1_reg;
    r2_reg_out <= r2_reg;

END Behavioral;
```

RTL Schematic

As can be seen in figure 1, the RTL schematic resembles the given schematic.

Figure 1: RTL Schematic of the Circuit



2 D Flip-Flop with Asynchronous Reset

Code

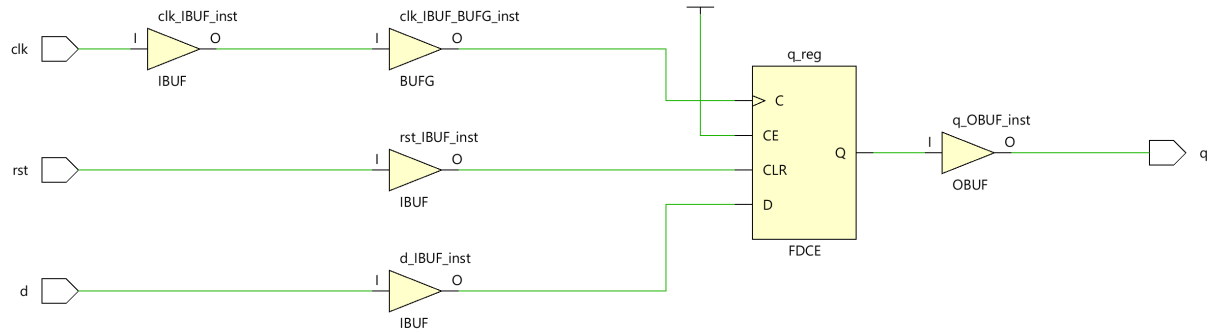
D Flip-Flop

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

ENTITY dffr IS
    PORT (
        clk : IN STD_LOGIC;
        rst : IN STD_LOGIC;
        d : IN STD_LOGIC;
        q : OUT STD_LOGIC
    );
END dffr;

ARCHITECTURE arch OF dffr IS

BEGIN

    PROCESS (clk, rst)
    BEGIN
        IF (rst = '1') THEN -- asynchronous reset
            q <= '0';
        ELSIF (rising_edge(clk)) THEN -- d get clocked to q when rising and not reset
            q <= d;
        END IF;
    END PROCESS;

END arch;
```

Schematic

As can be seen in figure 2, the code models a D Flip-Flop with Asynchronous Reset.

Figure 2: Schematic of the Circuit



3

3.a **Assume $D_1 = 0$, $D_2 = 5$, and D_1 changes to 1 at time $= 10$ ns. What are the values of D_1 and D_2 after the following code has been executed once? Do the values of D_1 and D_2 swap?**

```
RUN: process (D1)
begin
    D2 <= D1;
    D1 <= D2;
end process;
```

Assuming it is synthesizable, as statements executes at the same time, when D1 change to be specific, D2 will get loaded with D1's value immediate at 10 ns and vice versa, D1 will be loaded 5 and D2 will be loaded 1. In other words, it swaps after the process block was executed once.

3.b Simplify the following code segment, i.e., what is it equivalent to?

```
process(a, b, c, d)
   begin
   y <= a or c;
   y <= a and b;
   y <= c and d;
end process;
```

It is equivalent to `y <= c and d;`.

As only the last statement will be the final assignment to y, it can be simplified to the following:

```
process(a, b, c, d)
   begin
   y <= c and d;
end process;
```

As change in a and b alone does not result in any change in y since it depend solely in c and d, it can then be further simplified to the following:

```
process(c, d)
   begin
   y <= c and d;
end process;
```

As there is only one line inside the process and the sensitivity functionality of process block is redundant with it consist of solely the signals which y is dependent on, the statement can be taken out of the process block while functioning exactly the same way, thus the final result:

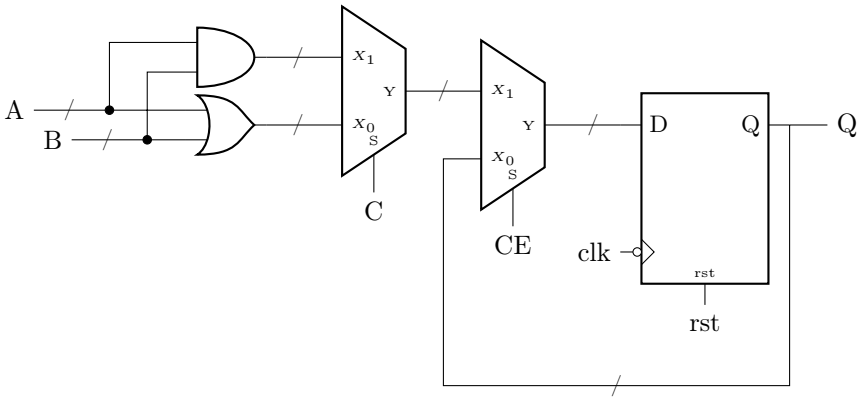
```
   y <= c and d;
```

4

```
TST : process (clk, rst)
begin
    if (rst = '1') then
        Q <= (others => '0');
    elsif (falling_edge(clk)) then
        if (CE = '1') then
            if (C = '0') then
                Q <= A or B;
            else
                Q <= A and B;
            end if;
        end if;
    end if;
end process TST;
```

4.i Draw the circuit represented by the following Verilog process:

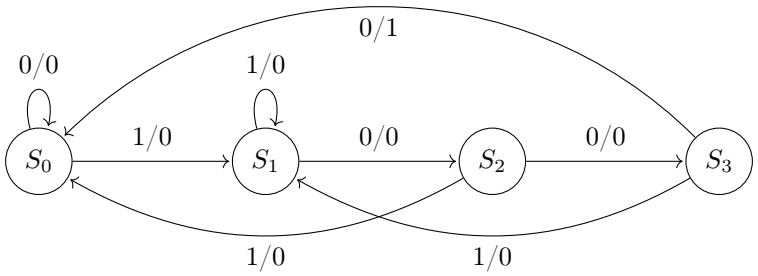
The first three line inside the process block resembles the structure of a flipflop with asynchronous reset triggers on falling edge. The nested if statements inside the **elsif** block can then be modeled with multiplexer controlled with C going into another multiplexer controlled by CE going into D. As the **else** of CE is not specified, it is assume that Q remain unchanged when CE = 0, meaning Q go into X_0 of mux controlled by CE. From these information the following schematic can be constructed. Also, the `Q <= (others => '0');` line indicate that Q is a bus, thus A and B must also be bus as it would result in type mismatch otherwise.



4.ii Why rst is on the sensitivity list whereas C is not?

As everything inside the elsif block happen only at the falling edge of the clock, the clk on the sensitivity already cover that block, thus no need to put C on sensitivity list. However, since the reset is intended to be asynchronous, it cannot happen only when clk change, thus the necessity to put rst on the sensitivity list such that reset can be trigger independent of the clock.

5 Write the VHDL code that implements the graphical representation of the finite state machine (FSM) below using asynchronous modeling.



Code

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

ENTITY ECE4304_Midterm2_5 IS
    PORT (
        fsm_clk : IN STD_LOGIC;
        fsm_rst : IN STD_LOGIC;
        fsm_in : IN STD_LOGIC;
        fsm_out : OUT STD_LOGIC
    );
END ECE4304_Midterm2_5;

ARCHITECTURE Behavioral OF ECE4304_Midterm2_5 IS

    -- state definition
    TYPE state_type IS (
        s0,
        s1,
        s2,
        s3
    );

    -- state registers
    SIGNAL ps_state : state_type; -- present state register
    SIGNAL ns_state : state_type; -- next state register
    SIGNAL fsm_out_next : STD_LOGIC; -- next output

BEGIN

    -- state transition logic
    STATE_TRAN : PROCESS (fsm_clk, fsm_rst)
    BEGIN
        IF (fsm_rst = '1') THEN -- reset state
            ps_state <= s0;
        ELSIF (rising_edge(fsm_clk)) THEN
```

```
            ps_state <= ns_state; -- state transition
            fsm_out <= fsm_out_next; -- output transition
        END IF;
    END PROCESS;

    -- (asynchronous) next state logic
    STATE_NEXT : PROCESS (ps_state, fsm_in)
    BEGIN
        fsm_out_next <= '0';
        CASE ps_state IS
            WHEN s0 =>
                IF (fsm_in = '1') THEN
                    ns_state <= s1;
                ELSE
                    ns_state <= s0;
                END IF;
            WHEN s1 =>
                IF (fsm_in = '1') THEN
                    ns_state <= s1;
                ELSE
                    ns_state <= s2;
                END IF;
            WHEN s2 =>
                IF (fsm_in = '1') THEN
                    ns_state <= s0;
                ELSE
                    ns_state <= s3;
                END IF;
            WHEN s3 =>
                IF (fsm_in = '1') THEN
                    ns_state <= s1;
                ELSE
                    ns_state <= s0;
                    fsm_out_next <= '1';
                END IF;
            WHEN OTHERS =>
        END CASE;
    END PROCESS;

END Behavioral;
```

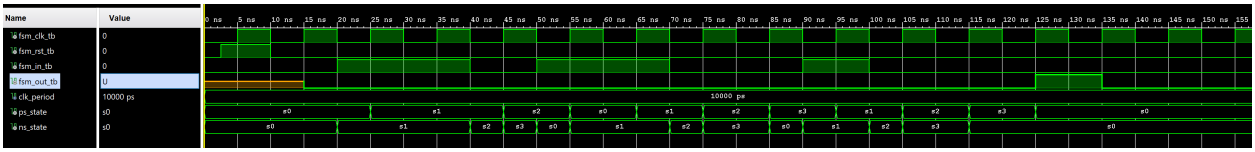
Testbench

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

ENTITY ECE4304_Midterm2_5_tb IS
    --  Port ( );
END ECE4304_Midterm2_5_tb;

ARCHITECTURE Behavioral OF ECE4304_Midterm2_5_tb IS
```

```
    COMPONENT ECE4304_Midterm2_5 IS
        PORT (
            fsm_clk : IN STD_LOGIC;
            fsm_rst : IN STD_LOGIC;
            fsm_in : IN STD_LOGIC;
            fsm_out : OUT STD_LOGIC
        );
    END COMPONENT;

    SIGNAL fsm_clk_tb : STD_LOGIC;
    SIGNAL fsm_rst_tb : STD_LOGIC;
    SIGNAL fsm_in_tb  : STD_LOGIC;
    SIGNAL fsm_out_tb : STD_LOGIC;

    CONSTANT clk_period : TIME := 10ns;

BEGIN

    UUT: ECE4304_Midterm2_5
    PORT MAP(
        fsm_clk => fsm_clk_tb,
        fsm_rst => fsm_rst_tb,
        fsm_in  => fsm_in_tb,
        fsm_out => fsm_out_tb
    );

    CLK_GEN : PROCESS
    BEGIN
        fsm_clk_tb <= '0';
        WAIT FOR 0.5 * clk_period;
        fsm_clk_tb <= '1';
        WAIT FOR 0.5 * clk_period;
    END PROCESS;

    RST_GEN : PROCESS
    BEGIN
        fsm_rst_tb <= '0';
        WAIT FOR 0.25 * clk_period;
        fsm_rst_tb <= '1';
        WAIT FOR 0.75 * clk_period;
        fsm_rst_tb <= '0';
        WAIT;
    END PROCESS;

    MAIN : PROCESS
    BEGIN
        -- goal is to test each edge
        fsm_in_tb <= '0';
        WAIT FOR clk_period;
        -- now in s0 after reset
        WAIT FOR clk_period; -- 0/0, s0 to s0, new
        fsm_in_tb <= '1';
        WAIT FOR clk_period; -- 1/0, s0 to s1, new
        WAIT FOR clk_period; -- 1/0, s1 to s1, new
```

```
        fsm_in_tb <= '0';
        WAIT FOR clk_period; -- 0/0, s1 to s2, new
        fsm_in_tb <= '1';
        WAIT FOR clk_period; -- 1/0, s2 to s0, new
        WAIT FOR clk_period; -- 1/0, s0 to s1
        fsm_in_tb <= '0';
        WAIT FOR clk_period; -- 0/0, s1 to s2
        WAIT FOR clk_period; -- 0/0, s2 to s3, new
        fsm_in_tb <= '1';
        WAIT FOR clk_period; -- 1/0, s3 to s1, new
        fsm_in_tb <= '0';
        WAIT FOR clk_period; -- 0/0, s1 to s2
        WAIT FOR clk_period; -- 0/0, s2 to s3
        WAIT FOR clk_period; -- 0/1, s3 to s0, new, only clock cycle with output = 1
        WAIT; -- output should revert back to 0
        -- all 8 transitions were tested


    END PROCESS;

END Behavioral;
```

Figure 3: Simulation Waveform of the FSM



All 8 edges were tested:

- $0/0$, S_0 to S_0 at 15 ns
- $1/0$, S_0 to S_1 at 25 ns
- $1/0$, S_1 to S_1 at 35 ns
- $0/0$, S_1 to S_2 at 45 ns
- $1/0$, S_2 to S_0 at 55 ns
- $0/0$, S_2 to S_3 at 85 ns
- $1/0$, S_3 to S_1 at 95 ns
- $0/1$, S_3 to S_0 at 125 ns