HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY – VNU HCMC
OFFICE FOR INTERNATIONAL STUDY PROGRAM
FACULTY OF ELECTRICAL AND ELECTRONIC ENGINEERING

_____ * _____

DIGITAL SYSTEMS (LAB)
EXPERIMENTAL REPORT (PreLab 4)

Lecturer : **Mr. Nguyễn Tuấn Hùng**
Subject : **Digital Systems**
Class : **TT06**
Name : **Lương Triển Thắng**
Student ID : **2051194**

Ho Chi Minh City, 7th June, 2022

Contents

IV Pre Laboratory 4

Finite State Machines

1. Create state diagram and build the circuit for a finite state machine

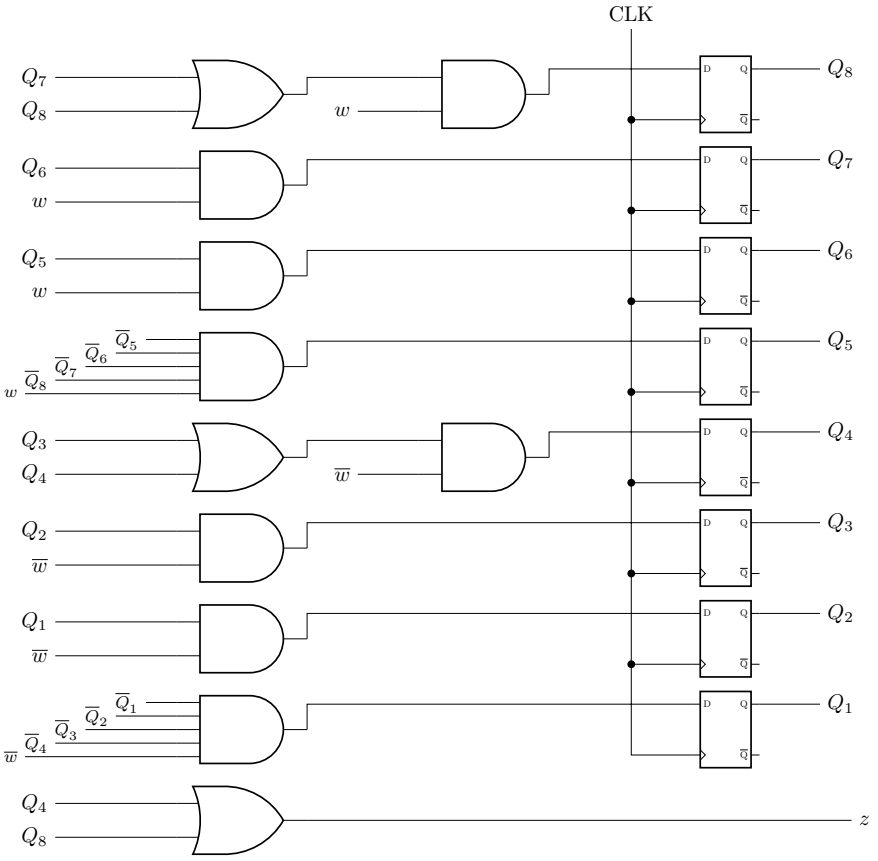
a. Next state table

Present State		Next State $w=0$		Next State $w=1$		Output $w=0$	Output $w=1$
A	000000001	B	000000010	F	000100000	0	0
B	000000010	C	000000100	F	000100000	0	0
C	000000100	D	000001000	F	000100000	0	0
D	000001000	E	000010000	F	000100000	0	0
E	000010000	E	000010000	F	000100000	1	0
F	000100000	B	000000010	G	001000000	0	0
G	001000000	B	000000010	H	010000000	0	0
H	010000000	B	000000010	I	100000000	0	0
I	100000000	B	000000010	I	100000000	0	1
Others	×	Others	×	Others	×	×	×

b. Functions for flip-flops

$$D_8 = w(Q_7 + Q_8) \qquad D_5 = \overline{Q}_5\overline{Q}_6\overline{Q}_7\overline{Q}_8 w \qquad D_2 = Q_1\overline{w}$$
$$D_7 = Q_6 w \qquad D_4 = \overline{w}(Q_3 + Q_4) \qquad D_1 = \overline{Q}_1\overline{Q}_2\overline{Q}_3\overline{Q}_4\overline{w}$$
$$D_6 = Q_5 w \qquad D_3 = Q_2\overline{w} \qquad z = \overline{Q}_4 + \overline{Q}_8$$

c. Circuit diagram



2. Describe FSM using VHDL behavioral expressions

Exc2.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY Exc2 IS PORT (
        w, clk, rst : IN STD_LOGIC;
        x : OUT STD_LOGIC
);
END Exc2;

ARCHITECTURE Behavior OF Exc2 IS
        TYPE State_type IS (A, B, C, D, E, F, G, H, I);
        ATTRIBUTE syn_encoding : STRING;
        ATTRIBUTE syn_encoding OF State_type : TYPE IS "0000 0001 0010 0011 0100 0101 0110 0111 1000";

        SIGNAL y_Q, Y_D : State_type;
BEGIN
        PROCESS (w, y_D)
        BEGIN
                CASE y_D IS
                        WHEN A =>
                                IF (w = '0') THEN
                                        y_Q <= B;
                                ELSE
                                        y_Q <= F;
                                END IF;

                        WHEN B =>
                                IF (w = '0') THEN
                                        y_Q <= C;
                                ELSE
                                        y_Q <= F;
                                END IF;

                        WHEN C =>
                                IF (w = '0') THEN
                                        y_Q <= D;
                                ELSE
                                        y_Q <= F;
                                END IF;

                        WHEN D =>
                                IF (w = '0') THEN
                                        y_Q <= E;
                                ELSE
                                        y_Q <= F;
                                END IF;

                        WHEN E =>
                                IF (w = '0') THEN
                                        y_Q <= E;
                                ELSE
                                        y_Q <= F;
                                END IF;

                                -------------------------
                        WHEN F =>
                                IF (w = '1') THEN
                                        y_Q <= G;
                                ELSE
                                        y_Q <= B;
```

```
                        END IF;

                WHEN G =>
                        IF (w = '1') THEN
                                y_Q <= H;
                        ELSE
                                y_Q <= B;
                        END IF;

                WHEN H =>
                        IF (w = '1') THEN
                                y_Q <= I;
                        ELSE
                                y_Q <= B;
                        END IF;

                WHEN I =>
                        IF (w = '1') THEN
                                y_Q <= I;
                        ELSE
                                y_Q <= B;
                        END IF;
        END CASE;
    END PROCESS;
    x <= '1' WHEN y_D = E OR y_D = I ELSE '0';

    PROCESS (clk, rst)
    BEGIN
        IF rst = '1' THEN
                Y_D <= A;
        ELSE
                IF rising_edge(clk) THEN
                        y_D <= y_Q;
                END IF;
        END IF;
    END PROCESS;
END Behavior;
```

3. Implement a shift register

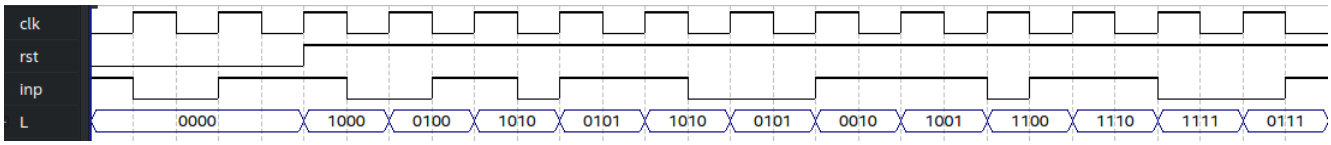
a. Code

Exc3.vhd

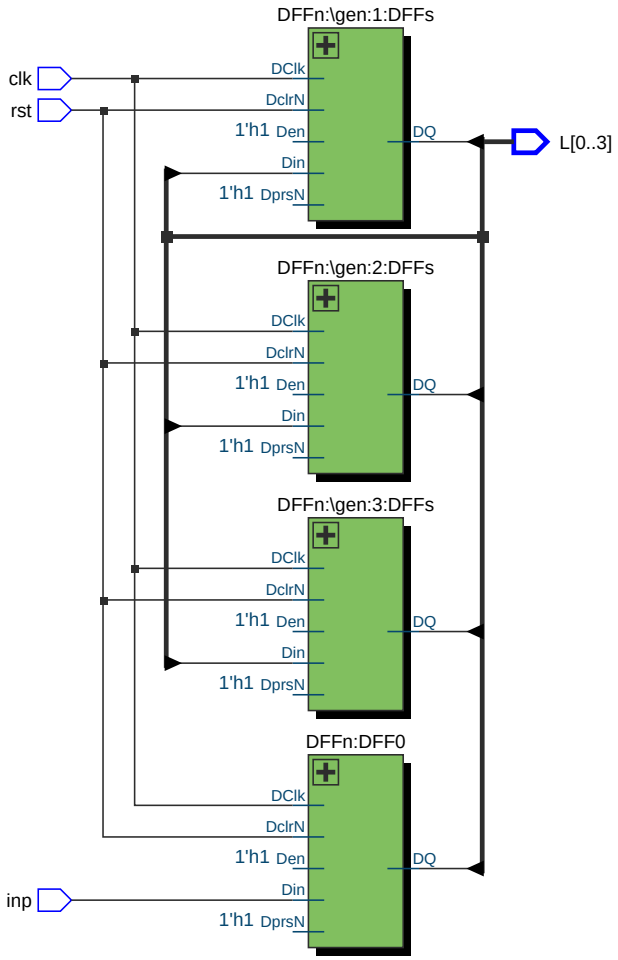
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY Exc3 IS
    PORT (
        clk, inp, rst : IN STD_LOGIC;     -- rst: active low
        L : OUT STD_LOGIC_VECTOR(0 TO 3)
    );
END Exc3;

ARCHITECTURE arch OF Exc3 IS
    SIGNAL q : STD_LOGIC_VECTOR(0 TO 3);
    COMPONENT DFFn IS
        PORT (
            Din, DClk, DprsN, DclrN, Den : IN STD_LOGIC; -- clr and prs are active low.
            DQ : OUT STD_LOGIC);
    END COMPONENT;
```

```
BEGIN
        DFF0 : DFFn PORT MAP(
                Din => inp, DClk => clk, DprsN => '1',
                DclrN => rst, Den => '1', DQ => q(0));
        gen : FOR i IN 1 TO 3 GENERATE
                DFFs : DFFn PORT MAP(
                        Din => q(i - 1), DClk => clk, DprsN => '1',
                        DclrN => rst, Den => '1', DQ => q(i));
        END GENERATE;
        L <= q;
END ARCHITECTURE;
```

b. Waveform



c. Result of RTL viewer



4. Implement a periodic signal

a. Code

Exc4.vhd

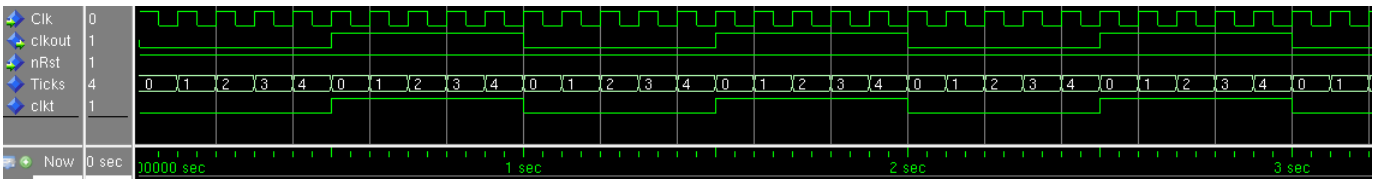
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY Exc4 IS
    GENERIC (rate : INTEGER); -- rate = oldFreq / newFreq
    PORT (
        Clk : IN STD_LOGIC;
        clkout : OUT STD_LOGIC;
        nRst : IN STD_LOGIC -- Negative reset
    );
END ENTITY;

ARCHITECTURE rtl OF Exc4 IS
    -- Signal for counting clock periods
    SIGNAL Ticks : INTEGER := 0;
    SIGNAL clkt : STD_LOGIC := '0';

BEGIN
    clkout <= clkt;
    PROCESS (Clk) IS
    BEGIN
        IF rising_edge(Clk) THEN
            -- If the negative reset signal is active
            IF nRst = '0' THEN
                Ticks <= 0;
            ELSE
                IF Ticks = (rate / 2) - 1 THEN
                    Ticks <= 0;
                    clkt <= NOT (clkt);
                ELSE
                    Ticks <= Ticks + 1;
                END IF;
            END IF;
        END IF;
    END PROCESS;

END ARCHITECTURE;
```

Exc4_tb.vhd

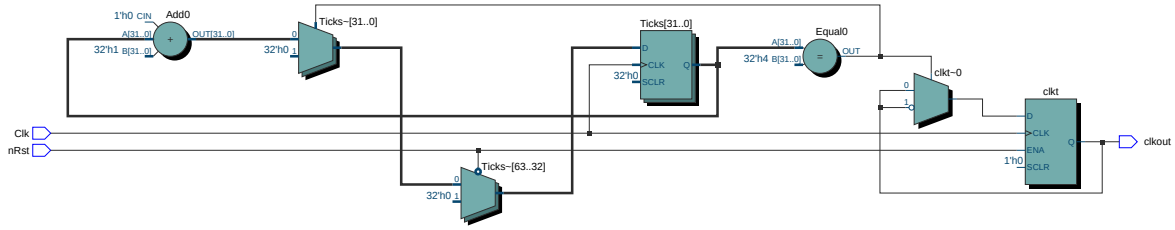
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY Exc4_tb IS
END ENTITY;

ARCHITECTURE sim OF Exc4_tb IS

    -- We're slowing down the clock to speed up simulation time
    CONSTANT ClockFrequencyHz : INTEGER := 10; -- 10 Hz; 50 MHz or 25 MHz for the clock on DE10
    CONSTANT ClockPeriod : TIME := 1000 ms / ClockFrequencyHz;

    SIGNAL Clk : STD_LOGIC := '1';
    SIGNAL nRst : STD_LOGIC := '1';
    SIGNAL clkout : STD_LOGIC;
```

```
BEGIN

    -- The Device Under Test (DUT)
    CD : ENTITY work.Exc4(rtl)
        GENERIC MAP(rate => 10)
        PORT MAP(
            Clk => Clk,
            nRst => nRst,
            clkout => clkout);

    -- Process for generating the clock
    Clk <= NOT Clk AFTER ClockPeriod / 2;

END ARCHITECTURE;
```

b. Waveform



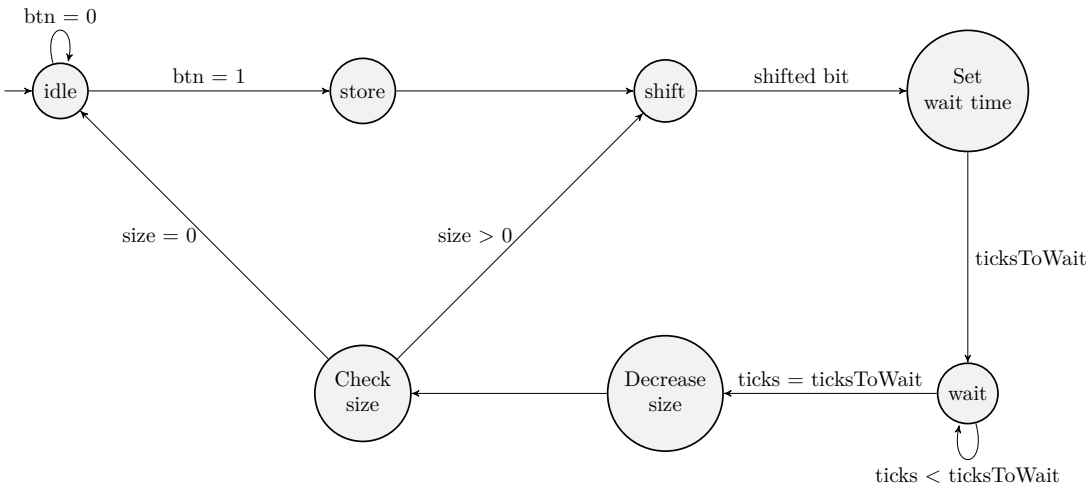
c. Result of RTL viewer



5. Know how to implement a digital circuit using an FSM

In this exercise, I modifed the circuit, so when there is a dot, LED will on for 0.25 second; if there is a dash, LED will on for 1 second.

a. FSM diagram



b. Details

- **Idle:** In this state, the circuit waits for user pressing the button.
- **Store:** The circuit will store morse code and morse length according to the SW[2:0].
- **Shift:** The circuit shift the bit array to the right.
- **Set wait time:**
  + If shifted `bit` = 0 (dot), set wait time = 0.25s.
  + If shifted `bit` = 1 (dash), set wait time = 1s.
- **Wait:** The circuit will count `ticks` until `ticks = ticksToWait`.
- **Decrease size:** Decrease size of morse length.
- **Check size:**
  + If `size` = 0, return to idle state.
  + If `size` > 0, return to shift state.

c. Expected waveform