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 (Lab 4)

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

Ho Chi Minh City, 14th June, 2022

Contents

IV Laboratory 4

Finite State Machines

1. Know how to implement a FSM circuit and download the cicuit into the FPGA chip

a. Code

DFFn.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY DFFn IS
        PORT (
                Din, DClk, Dprs, Dclr, Den : IN STD_LOGIC;
                DQ : OUT STD_LOGIC);
END DFFn;
ARCHITECTURE Behavior OF DFFn IS
BEGIN
        PROCESS (Dprs, Dclr, DClk)
        BEGIN
                IF Dclr = '1' AND Dprs = '0' THEN
                        DQ <= '0';
                ELSIF Dclr = '0' AND Dprs = '1' THEN
                        DQ <= '1';
                ELSE
                        IF rising_edge(DClk) AND Den = '1' THEN
                                DQ <= Din;
                        END IF;
                END IF;
        END PROCESS;
END Behavior;
```

NineDFF.vhd

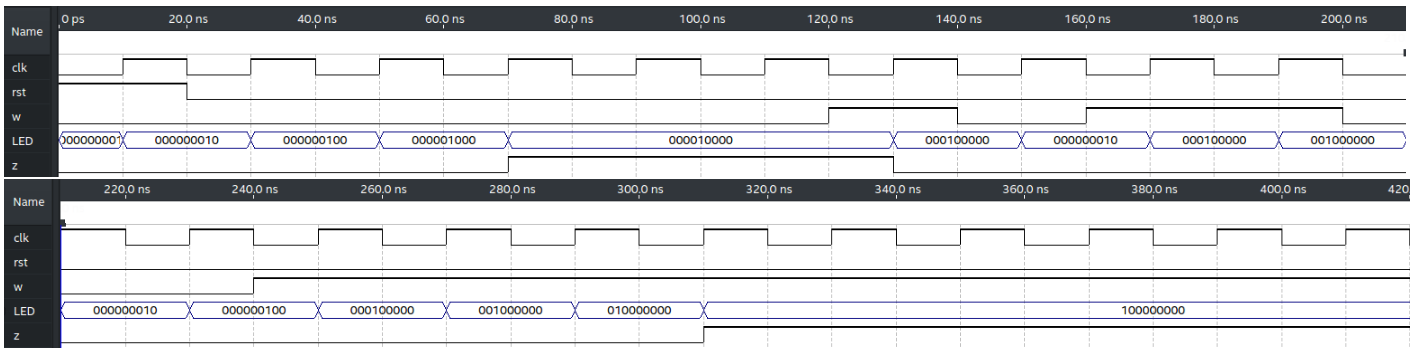
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY NineDFF IS
        PORT (
                NClk, Nen : IN STD_LOGIC;
                ND, Nprs, Nclr : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
                NQ : OUT STD_LOGIC_VECTOR(8 DOWNTO 0)
        );
END NineDFF;

ARCHITECTURE arch OF NineDFF IS
        COMPONENT DFFn IS
                PORT (
                        Din, DClk, Dprs, Dclr, Den : IN STD_LOGIC;
                        DQ : OUT STD_LOGIC);
        END COMPONENT;
BEGIN
        gen : FOR i IN 8 DOWNTO 0 GENERATE
                DFFs : DFFn PORT MAP(
                        Din => ND(i), DClk => NClk, Dprs => Nprs(i),
                        Dclr => Nclr(i), Den => Nen, DQ => NQ(i));
        END GENERATE;
END ARCHITECTURE;
```

Exc1.vhd

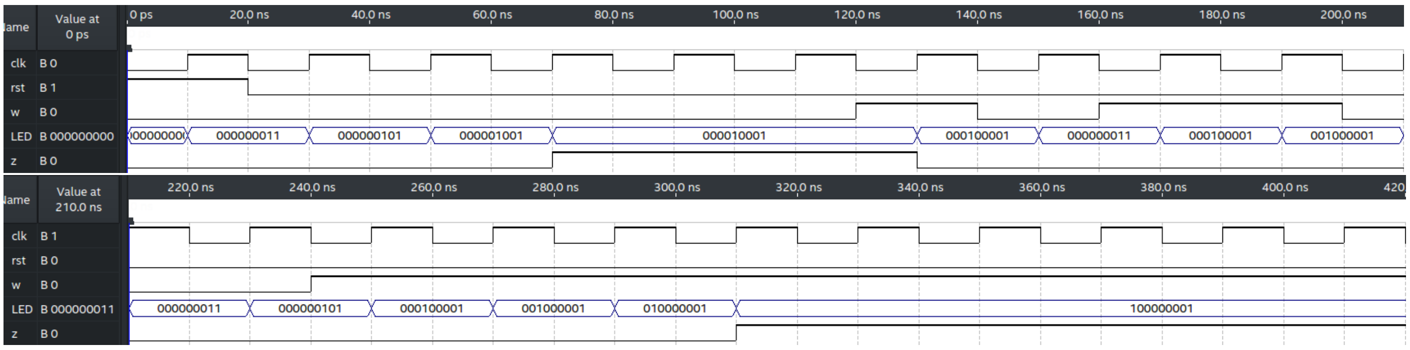
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY Exc1 IS
        PORT (
                clk, w, rstN : IN STD_LOGIC;
                LED : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
                z : OUT STD_LOGIC
        );
END Exc1;

ARCHITECTURE arch OF Exc1 IS
        SIGNAL D, Q, prsV, clrV : STD_LOGIC_VECTOR(8 DOWNTO 0);
        SIGNAL rst : STD_LOGIC;
        COMPONENT NineDFF IS
                PORT (
                        NClk, Nen : IN STD_LOGIC;
                        ND, Nprs, Nclr : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
                        NQ : OUT STD_LOGIC_VECTOR(8 DOWNTO 0)
                );
        END COMPONENT;
BEGIN
        rst <= NOT(rstN);

        D(8) <= w AND (Q(7) OR Q(8));
        D(7) <= Q(6) AND w;
        D(6) <= Q(5) AND w;
        D(5) <= w AND NOT(Q(5) OR Q(6) OR Q(7) OR Q(8));
        D(4) <= NOT(w) AND (Q(3) OR Q(4));
        D(3) <= Q(2) AND NOT(w);
        D(2) <= Q(1) AND NOT(w);
        D(1) <= NOT(Q(1) OR Q(2) OR Q(3) OR Q(4) OR w);
        D(0) <= '0';
        z <= Q(4) OR Q(8);

        -- FOR MODIFIED ONE-HOT
        --        D(8) <= w AND (Q(7) OR Q(8));
        --        D(7) <= Q(6) AND w;
        --        D(6) <= Q(5) AND w;
        --        D(5) <= w AND (NOT(Q(5) OR Q(6) OR Q(7) OR Q(8)));
        --        D(4) <= NOT(w) AND (Q(3) OR Q(4));
        --        D(3) <= Q(2) AND NOT(w);
        --        D(2) <= Q(1) AND NOT(w);
        --        D(1) <= NOT(Q(1) OR Q(2) OR Q(3) OR Q(4) OR w);
        --        D(0) <= '1';
        --        z <= Q(4) OR Q(8);

        LED <= Q;

        prsV <= "000000001" WHEN rst = '1' ELSE "000000000";
        clrV <= "111111110" WHEN rst = '1' ELSE "000000000";

        -- FOR MODIFIED ONE-HOT
        --        prsV <= "000000000" WHEN rst = '1' ELSE "000000000";
        --        clrV <= "111111111" WHEN rst = '1' ELSE "000000000";

        NDFF : NineDFF PORT MAP(
                NClk => clk, Nprs => prsV, Nclr => clrV,
                Nen => '1', ND => D, NQ => Q
        );
END ARCHITECTURE;
```

b. Waveform

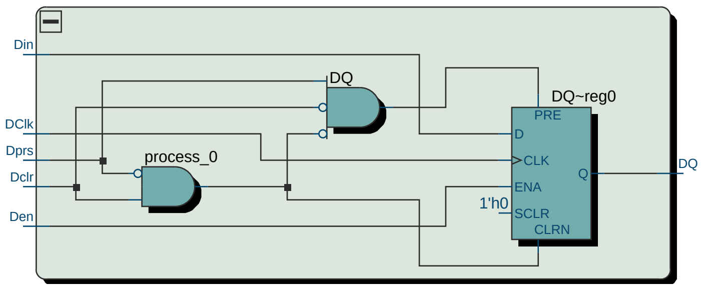
One-hot



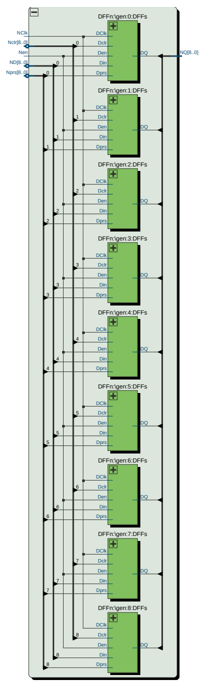
Modified one-hot



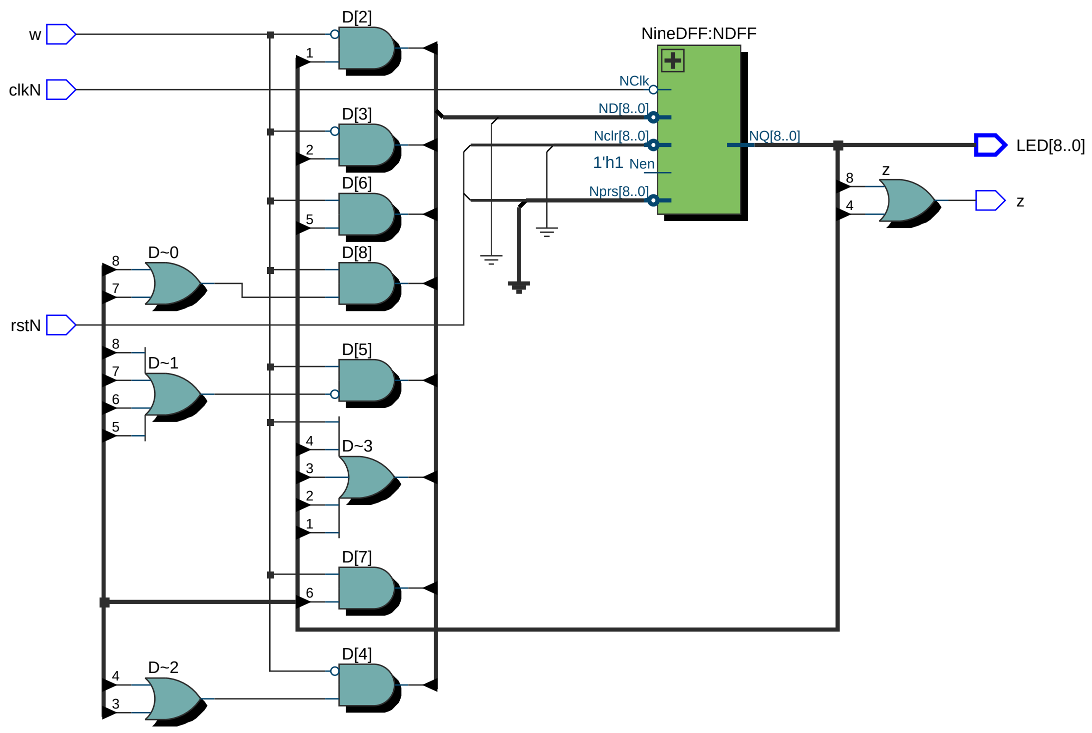
c. Result of RTL viewer



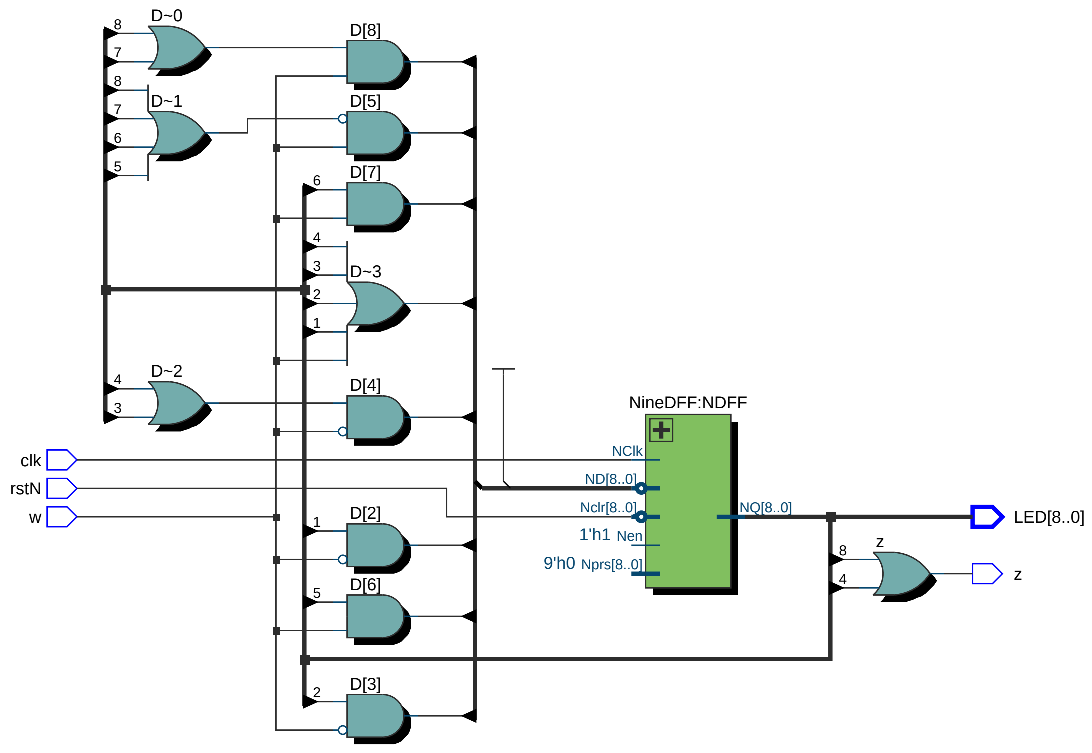
D flip-flop



Connected 9 D flip-flops



Top level (using one-hot encoding)



Top level (using modified one-hot encoding)

2. Know how to implement a FSM circuit using VHDL behavioral expressions and download the cicuit into the FPGA chip

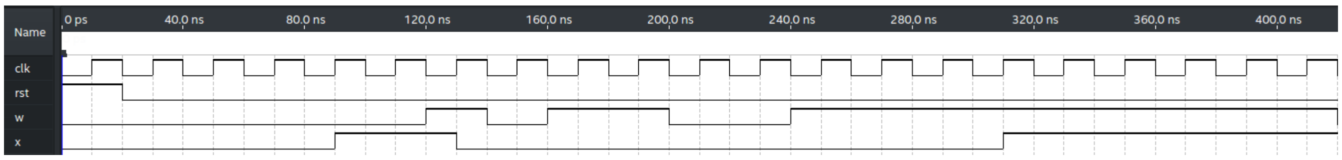
a. Code

Exc2.vhd

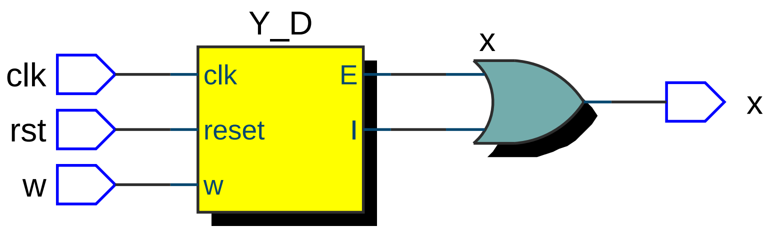
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY Exc2 IS PORT (
        w, clk, rst : IN STD_LOGIC;
        x : OUT STD_LOGIC
);
END Exc2;

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;
```

b. Waveform

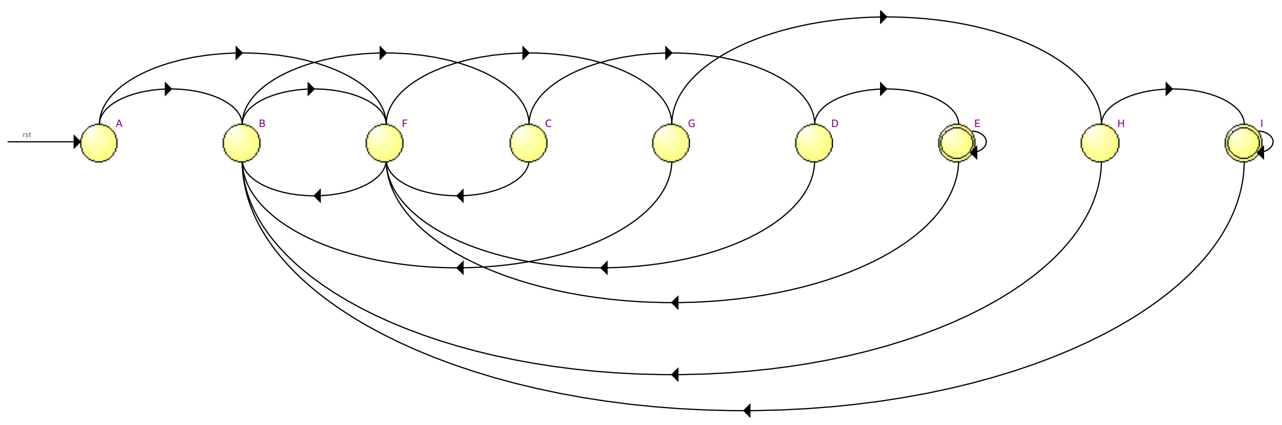


c. Result of RTL viewer

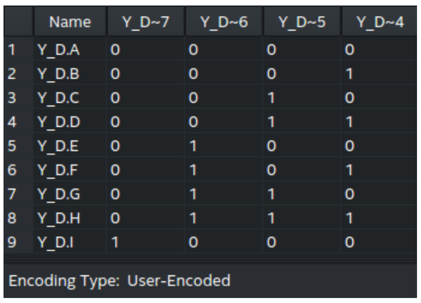


Top level

d. State machine diagram and encoding types

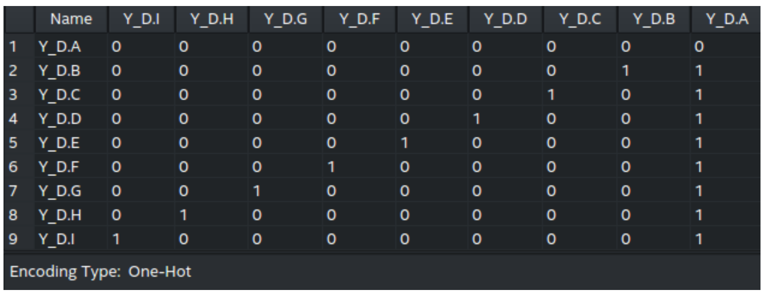


State machine diagram



	Name	Y_D~7	Y_D~6	Y_D~5	Y_D~4
1	Y_D.A	0	0	0	0
2	Y_D.B	0	0	0	1
3	Y_D.C	0	0	1	0
4	Y_D.D	0	0	1	1
5	Y_D.E	0	1	0	0
6	Y_D.F	0	1	0	1
7	Y_D.G	0	1	1	0
8	Y_D.H	0	1	1	1
9	Y_D.I	1	0	0	0

Encoding Type: User-Encoded



	Name	Y_D.I	Y_D.H	Y_D.G	Y_D.F	Y_D.E	Y_D.D	Y_D.C	Y_D.B	Y_D.A
1	Y_D.A	0	0	0	0	0	0	0	0	0
2	Y_D.B	0	0	0	0	0	0	0	1	1
3	Y_D.C	0	0	0	0	0	0	1	0	1
4	Y_D.D	0	0	0	0	0	1	0	0	1
5	Y_D.E	0	0	0	0	1	0	0	0	1
6	Y_D.F	0	0	0	1	0	0	0	0	1
7	Y_D.G	0	0	1	0	0	0	0	0	1
8	Y_D.H	0	1	0	0	0	0	0	0	1
9	Y_D.I	1	0	0	0	0	0	0	0	1

Encoding Type: One-Hot

Encoding types

3. Know how to implement sequence detector using shift registers

In this exercise, using two shift registers is a waste, so I will use only one shift register and connect it with some logic gates, that is enough.

a. Code

DFFn.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY DFFn IS
        PORT (
                Din, DClk, DprsN, DclrN, Den : IN STD_LOGIC; -- clr and prs are active low.
                DQ : OUT STD_LOGIC);
END DFFn;
ARCHITECTURE Behavior OF DFFn IS
BEGIN
        PROCESS (DprsN, DclrN, DClk)
        BEGIN
                IF DclrN = '0' AND DprsN = '1' THEN
                        DQ <= '0';
                ELSIF DclrN = '1' AND DprsN = '0' THEN
                        DQ <= '1';
                ELSE
                        IF rising_edge(DClk) AND Den = '1' THEN
                                DQ <= Din;
                        END IF;
                END IF;
        END PROCESS;
END Behavior;
```

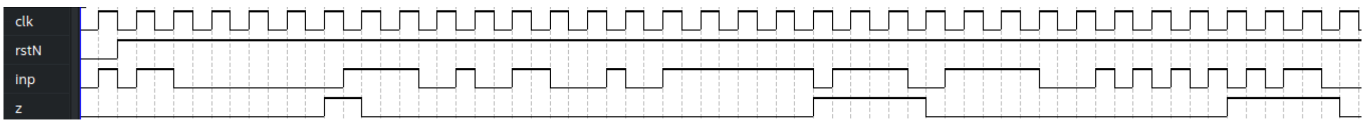
FourBitShiftReg.vhd

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

ENTITY FourBitShiftReg IS
        PORT (
                SRegclk, SReginp, SRegrstN : IN STD_LOGIC;
                SRegL : OUT STD_LOGIC_VECTOR(0 TO 3)
        );
END FourBitShiftReg;

ARCHITECTURE arch OF FourBitShiftReg IS
        SIGNAL q : STD_LOGIC_VECTOR(0 TO 3) := "XXXX";
        COMPONENT DFFn IS
                PORT (
                        Din, DClk, DprsN, DclrN, Den : IN STD_LOGIC;
                        DQ : OUT STD_LOGIC);
        END COMPONENT;
BEGIN
        DFF0 : DFFn PORT MAP(
                Din => SReginp, DClk => SRegclk, DprsN => '1',
                DclrN => SRegrstN, Den => '1', DQ => q(0));
        gen : FOR i IN 1 TO 3 GENERATE
                DFFs : DFFn PORT MAP(
                        Din => q(i - 1), DClk => SRegclk, DprsN => '1',
                        DclrN => SRegrstN, Den => '1', DQ => q(i));
        END GENERATE;
        SRegL <= q;
END ARCHITECTURE;
```

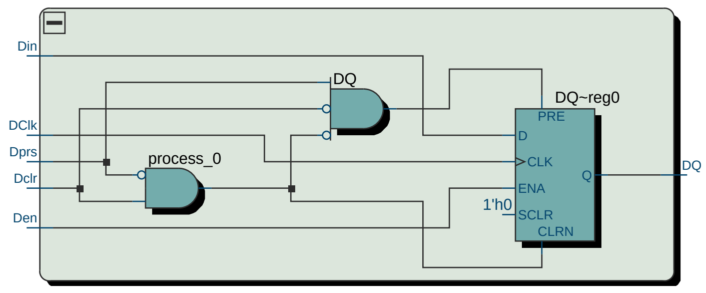
Exc2.vhd

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

ENTITY Exc3 IS
        PORT (
                inp, clk, rstN : IN STD_LOGIC;
                z : OUT STD_LOGIC
        );
END Exc3;

ARCHITECTURE arch OF Exc3 IS
        SIGNAL L : STD_LOGIC_VECTOR(0 TO 3);
        SIGNAL waitFourClocks : UNSIGNED(2 DOWNTO 0) := "100";
        SIGNAL en : STD_LOGIC;
        COMPONENT FourBitShiftReg IS
                PORT (
                        SRegclk, SReginp, SRegrstN : IN STD_LOGIC;
                        SRegL : OUT STD_LOGIC_VECTOR(0 TO 3)
                );
        END COMPONENT;
BEGIN
        PROCESS (clk)
        BEGIN
                IF rstN = '0' THEN
                        waitFourClocks <= "100";
                ELSIF rising_edge(clk) AND waitFourClocks /= 0 THEN
                        waitFourClocks <= waitFourClocks - 1;
                END IF;
        END PROCESS;
        en <= '1' WHEN waitFourClocks = 0 ELSE '0';
        SR1 : FourBitShiftReg PORT MAP(SRegclk => clk, SReginp => inp, SRegrstN => rstN, SRegL => L);

        z <= '1' WHEN (L1 = "1111" OR L1 = "0000") AND en = '1' ELSE '0';
END ARCHITECTURE;
```

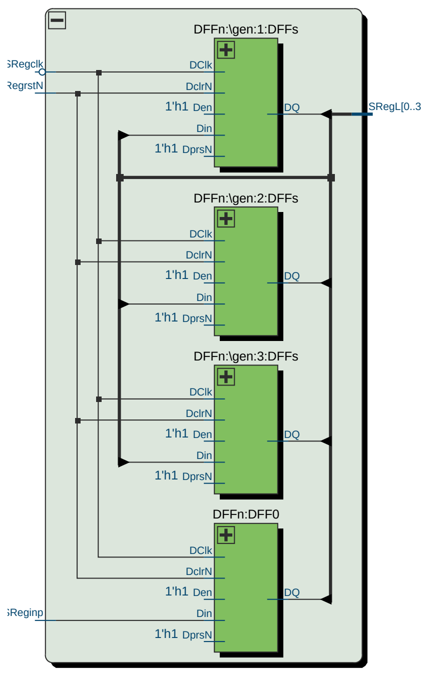
b. Waveform



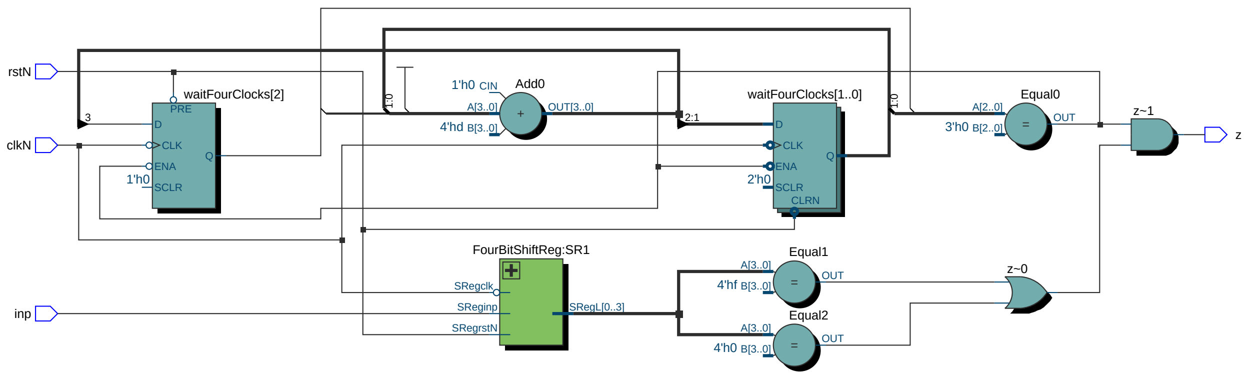
c. Result of RTL viewer



D flip-flop



Four-bit shift register

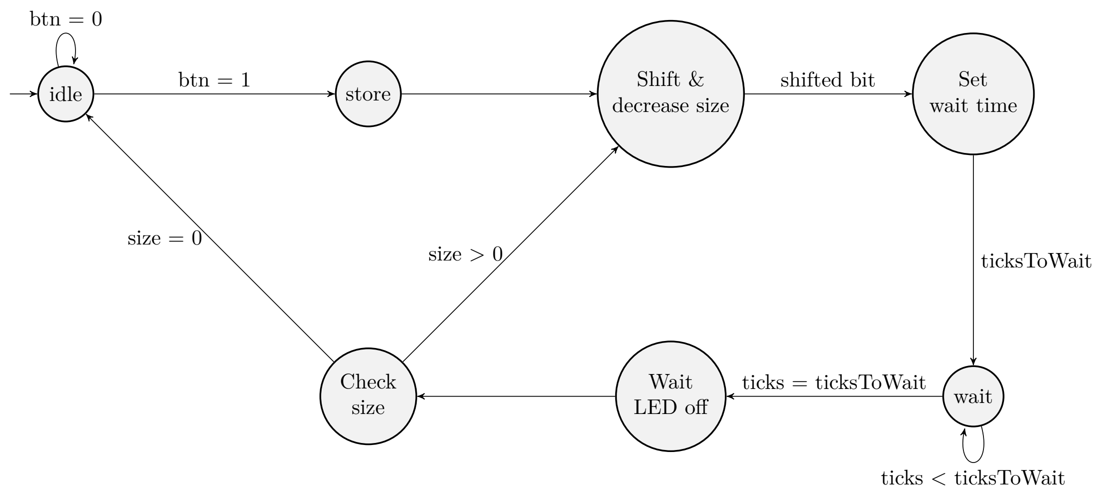


Top level

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

As I stated in PreLab 4, in this exercise, instead of using two LEDR and LEDG, I modifed the circuit, so when there is a dot, LED will on for 0.25 second; when there is a dash, LED will on for 1 second.

a. FSM diagram



b. Code

Exc4.vhd

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

ENTITY Exc4 IS
  PORT (
    letter : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    LED : OUT STD_LOGIC;
    clk, btnN, rstN : IN STD_LOGIC
  );
END Exc4;

ARCHITECTURE arch OF Exc4 IS
  SIGNAL morseCode : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL morseLength : STD_LOGIC_VECTOR(2 DOWNTO 0);
  SIGNAL bitIn, btn, rst : STD_LOGIC := '0';

  SIGNAL ticks, ticksToWait : INTEGER RANGE 0 TO 200 := 0; -- Change 200 to 50000000 if running on DE10.

  TYPE State_type IS (idle, store, shift, setWaitInterval, waitT, waitLEDOff, checkSize);
  SIGNAL state : State_type := idle;
  COMPONENT DFFn IS
    PORT (
      Din, DClk, Den : IN STD_LOGIC;
      DQ : OUT STD_LOGIC);
  END COMPONENT;

BEGIN
  btn <= NOT(btnN);
  rst <= NOT(rstN);
  PROCESS (clk, rst) IS
  BEGIN
```

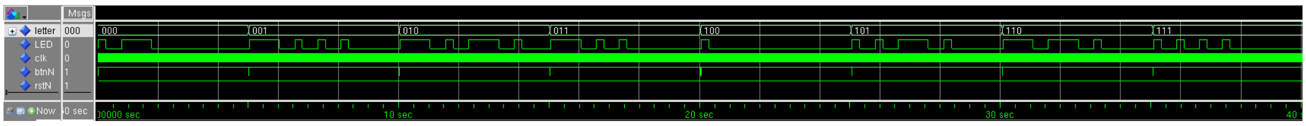
```
IF rst = '1' THEN
  state <= idle;
ELSE
  IF rising_edge(clk) THEN
    CASE state IS
      WHEN idle =>
        LED <= '0';
        IF btn = '1' THEN
          state <= store;
        ELSE
          state <= idle;
        END IF;

      WHEN store =>
        CASE letter IS
          WHEN "000" =>
            morseCode <= "0010";
            morseLength <= "010";
          WHEN "001" =>
            morseCode <= "0001";
            morseLength <= "100";
          WHEN "010" =>
            morseCode <= "0101";
            morseLength <= "100";
          WHEN "011" =>
            morseCode <= "0001";
            morseLength <= "011";
          WHEN "100" =>
            morseCode <= "0000";
            morseLength <= "001";
          WHEN "101" =>
            morseCode <= "0100";
            morseLength <= "100";
          WHEN "110" =>
            morseCode <= "0011";
            morseLength <= "011";
          WHEN "111" =>
            morseCode <= "0000";
            morseLength <= "100";
          WHEN OTHERS =>
            morseCode <= "0000";
            morseLength <= "000";
        END CASE;
        state <= shift;

      WHEN shift =>
        bitIn <= morseCode(0);
        morseCode <= '0' & morseCode(3 DOWNTO 1); -- Shift;
        morseLength <= STD_LOGIC_VECTOR(UNSIGNED(morseLength) - 1);
        state <= setWaitInterval;

      WHEN setWaitInterval =>
        -- clk at 200Hz
        -- ticksToWait = oldFreq / newFreq
        -- dot = 4Hz (0.25s), dash = 1Hz (1s)
        IF (bitIn = '0') THEN
          ticksToWait <= 50; -- On DE10: 50MHz / 4Hz = 12500000
        ELSE
          ticksToWait <= 200; -- On DE10: 50MHz / 1Hz = 50000000
        END IF;

        ticks <= 0; -- Reset ticks
        LED <= '1'; -- LED on
```

```
          state <= waitT;

        WHEN waitT =>
          IF ticks = ticksToWait - 1 THEN
            LED <= '0'; -- LED off
            ticks <= 0; -- Reset ticks
            state <= waitLEDOff;
            ticksToWait <= 100;
          ELSE
            ticks <= ticks + 1;
            state <= waitT;
          END IF;

        WHEN waitLEDOff =>
          IF ticks = ticksToWait - 1 THEN
            state <= checkSize;
            ticks <= 0; -- Reset ticks
          ELSE
            ticks <= ticks + 1;
            state <= waitLEDOff;
          END IF;

        WHEN checkSize =>
          IF morseLength = "000" THEN
            state <= idle;
          ELSE
            state <= shift;
          END IF;
      END CASE;
    END IF;
  END IF;
 END PROCESS;
END ARCHITECTURE;
```

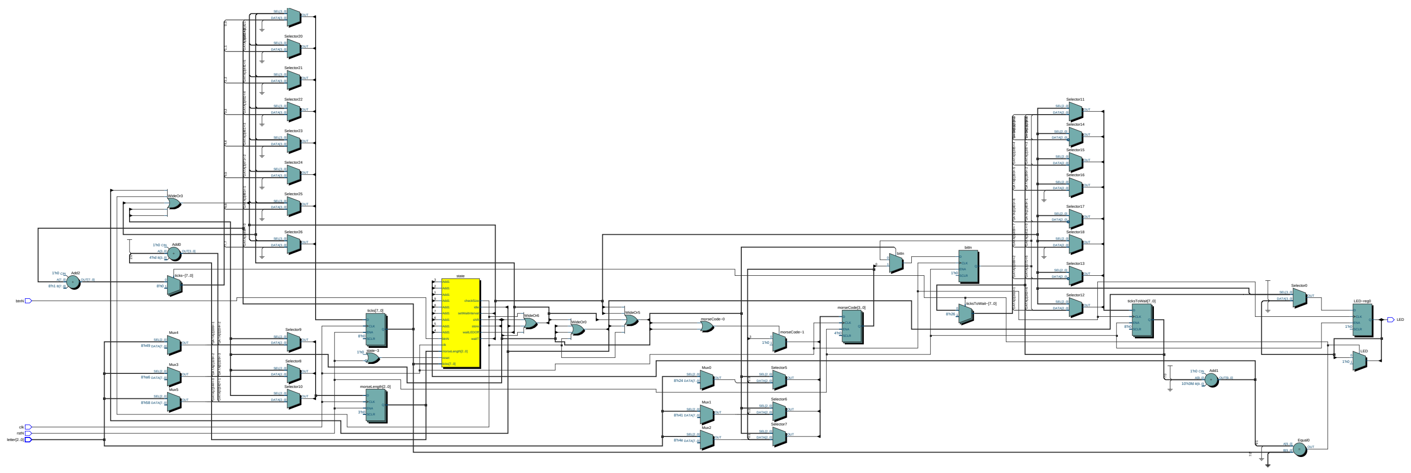
Exc4_tb.vhd

```
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 := 200; -- 200 Hz
        CONSTANT ClockPeriod : TIME := 1000 ms / ClockFrequencyHz;

        SIGNAL letter : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000";
        SIGNAL LED : STD_LOGIC;
        SIGNAL clk : STD_LOGIC := '1';
        SIGNAL btnN : STD_LOGIC := '0';
        SIGNAL rstN : STD_LOGIC := '1';
BEGIN
        -- The Device Under Test (DUT)
        morse : ENTITY work.Exc4(arch)
                PORT MAP(
                        letter => letter,
                        LED => LED,
                        clk => clk,
                        btnN => btnN,
                        rstN => rstN
                );
        -- Process for generating the clock
        Clk <= NOT Clk AFTER ClockPeriod / 2;
        PROCESS IS
        BEGIN
                WAIT UNTIL rising_edge(clk);
                btnN <= '0';
                WAIT UNTIL rising_edge(clk);
                btnN <= '1';
                WAIT FOR 5000 ms;
                letter <= STD_LOGIC_VECTOR(UNSIGNED(letter) + 1);
        END PROCESS;
END ARCHITECTURE;
```

c. Waveform

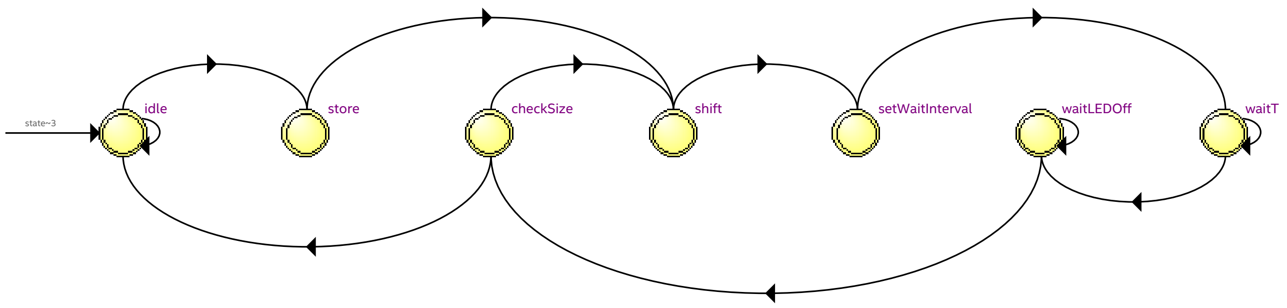


d. Result of RTL viewer



Top level (zoom in for better viewing)

e. State machine diagram


state~3
idle
store
checkSize
shift
setWaitInterval
waitLEDOff
waitT