

City University of Hong Kong

Department of Electrical Engineering

EE 2000 – Lab Manual 4

**Using Finite State Machine (FSM) to design
a 3-bit counter**

Course Leader: CHIN Lip Ket

Objectives:

- Learn how to write sequential logic circuits using VHDL
- Implement a simple finite-state machine FSM with a good coding style
- Implement two buttons to control the state changes of the FSM
- Display the state transition with the LEDs on the BASYS3 Board

There are 5 checkpoints in total.

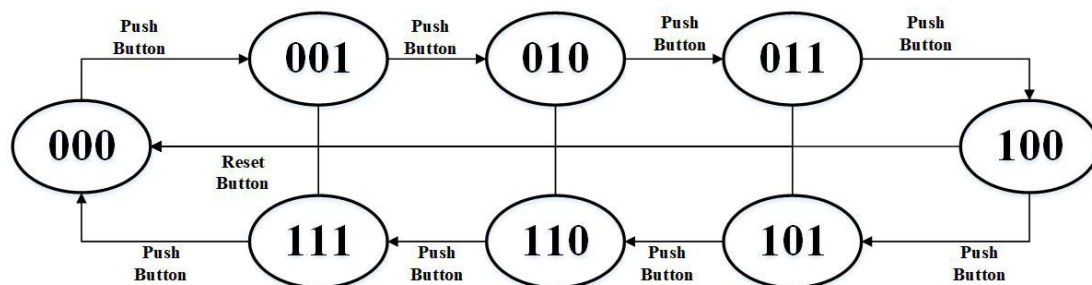
For each checkpoint, please take notes/photos/screenshots for your report.

Please show Checkpoints 1, 3 and 4 to the lab tutor or demonstrator for marking.

Experiment:

A finite-state machine (FSM) is an abstract machine that can be implemented as one of a finite number of states at any given time. The FSM can change from one state to another in response to some external inputs. The change is called a transition. An FSM can be defined by a list of its states, its initial state and the conditions for each transition.

In this experiment, we will implement a simple 3-bit counter with VHDL. The initial state is 000. Every time we push the control button, the counter will add 1 and LEDs will show the current state. The FSM will jump to the initial state 000 when the reset button is pushed. The diagram of this FSM is



The FSM has both sequential logic and combinational logic, it is a good habit to separate these two parts in VHDL. In this experiment, we will implement this FSM step by step.

- i. Determine the inputs and outputs of the module. The module has sequential logic circuits so we need a clock signal **clk** and a global reset signal **rst**. To observe the current state (the 3-bit states), we can use an 8-bit output signal which connects to LEDs. The entity of our FSM module is

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

entity fsm is Port (
  clk: in STD_LOGIC;
  rst: in STD_LOGIC;
  ctrlButton: in STD_LOGIC;
  led: out STD_LOGIC_VECTOR (7 DOWNTO 0)
);
end fsm;
```

- ii. In this experiment, we need to detect when will the control button is pushed. As the time the button is pressed, the **ctrlButton** is equal to '1'. We set a 10ms timer to eliminate any unstable inputs, which is called **button debounce**. Here we provide the key detection source code. Please read and try to understand the code. You are required to know how to write a piece of sequential logic code by yourself.

Architecture Behavioral of fsm is

```
Constant TIME_10ms: integer := 1000000;
Signal key_counter: integer range 0 to 1000001;
Signal btnDetection: boolean;
type states is (zero, one, two, three, four, five, six, seven);
signal present_state, next_state: states;

begin

Key_detection: process (clk) begin
  if rising_edge(clk) then --Clock event, used to describe sequential logic.
    if (ctrlButton='1') then
```

```

        key_counter <= 0;
    elsif (ctrlButton='0') and (key_counter <= TIME_10MS) then

        key_counter <= key_counter + 1;
    end if; end if;
end process;

btnDetection <= (key_counter = TIME_10ms);

```

When control button is pushed, **key_counter** will set to 0. After the button is released, the timer will start counting and if the stable time is long enough (reach to **TIME_10ms**), **btnDetection** sets to true for that cycle.

iii. The sequential logic of our FSM is very simple. When the **btnDetection** is true, the FSM will jump to the next state. We define a type called ***states*** enumerating all possible states. Then we define two signals **present_state** and **next_state** which are both ***states*** type. When the **btnDetection** is true, the **present_state** sets to **next_state**. The assignment of **next_state** will be introduced in the next section.

iv. The combinational logic of our FSM assigns LED outputs and **next_state**.

```

state_transition: process (rst, clk) begin
    if (rst='1') then
        --Write your code here to describe state transition.
    elsif (rising_edge(clk) and btnDetection = true) then
        --Write your code here to describe state transition.
    end if; end process;

```

The LED output is a 3-to-8 decoder. We can see the truth table of the decoder.

Present_state	LED Output
zero	00000001
one	00000010
two	00000100
three	00001000
four	00010000
five	00100000
six	01000000
seven	10000000

The corresponding VHDL template and you can complete the following code.

```
decoder: process (present_state)
begin
    case present_state is
        --Write your code here to assign LED and next_state
    end case;
end process;
end Behavioral;
```

v. Create constraint file.

In this experiment, we use the on-board 100 MHz clock source as the input of **clk** in our design, push-button BTNC as input of **rst**, BTNL as input of **ctrlButton**, LD0-LD7 as output of **led**[0] - **led**[7]. Please refer to the “BASYS3 Board Master Constraints file” to complete the code below.

```
# Timing Constraints
set_property PACKAGE_PIN W5 [get_ports {clk}]
    set_property IOSTANDARD LVCMOS33 [get_ports {clk}]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk}]

#Buttons

set_property PACKAGE_PIN U18 [get_ports {rst}]
    set_property IOSTANDARD LVCMOS33 [get_ports
{rst}]
#Write your code here to complete this constraints file
```

vi. Create testbench.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity fsm_tb is end fsm_tb;

architecture Behavioral of fsm_tb is component fsm is
Port (
    clk : in STD_LOGIC; rst :      in STD_LOGIC;
    ctrlButton : in STD_LOGIC;
    led : out STD_LOGIC_VECTOR (7 DOWNT0 0)
);
end component;
```

```

signal clk : STD_LOGIC;
signal rst : STD_LOGIC;
signal ctrlButton : STD_LOGIC;
signal led : STD_LOGIC_VECTOR (7 DOWNTO 0);

begin
    uut: fsm port map (clk=>clk, rst=>rst, ctrlButton=>ctrlButton,
led=>led);

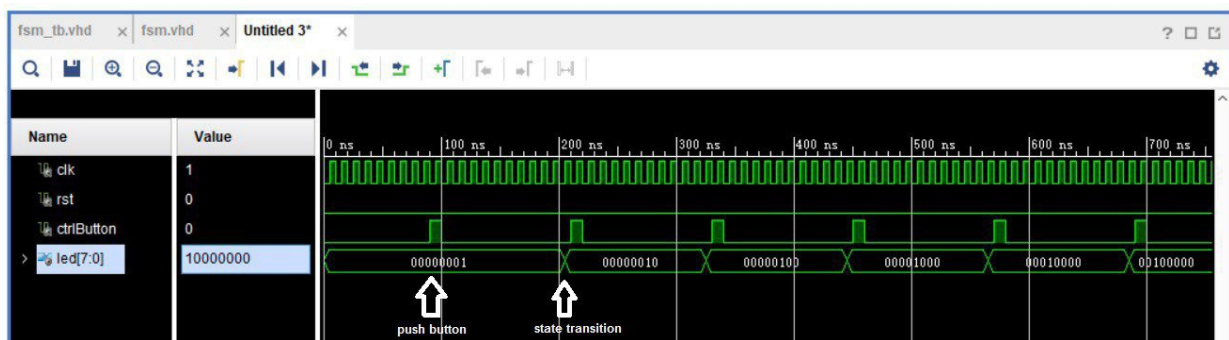
    clk_gen: process
    begin
        clk <= '0';
        wait for 5ns;
        clk <= '1';
        wait for 5ns;
    end process;

    rst <= '0';

    sig_gen: process
    begin
        ctrlButton <= '0';
        wait for 90ns;
        ctrlButton <= '1';
        wait for 10ns;
        ctrlButton <= '0';
        wait for 20ns;
    end process;
end Behavioral;

```

To shorten simulation time, change **TIME_10MS** in fsm.vhd to **10**. After simulation, we can get the waveform like this.

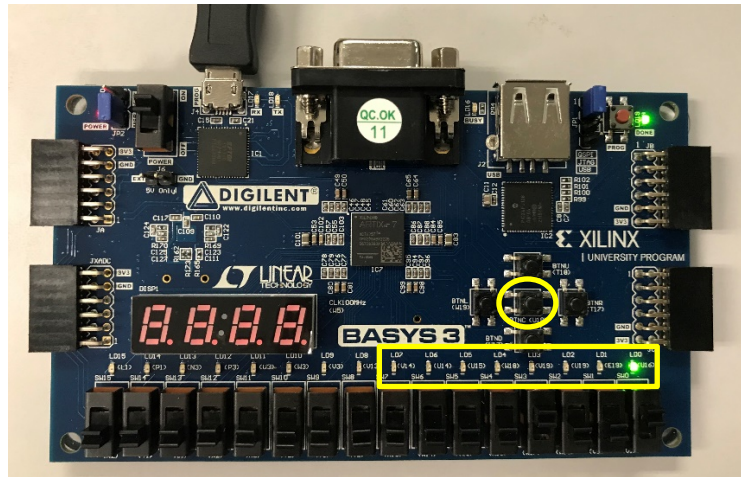


After the control button is pushed, it takes some time to detect and then it jumps to the next state while the LEDs signals show the current state.

Checkpoint 1: Show your simulation results to your demonstrator.

- vii. **Run synthesis, run implementation, and generate bitstream** as in lab 1. (Please don't forget to change the value **TIME_10MS** back in fsm.vhd) Then, we **program** the BASYS3 Board. As shown in the figure below, once you press the yellow circled button, the LED 0-7 will be illuminated one by one.

Checkpoint 2: Why you must change **TIME_10MS** in fsm.vhd in this case? Explain “Button Debounce” in short.



Checkpoint 3: Verify your design and show your result to your demonstrator.

Checkpoint 4: Here you are required to modify the previous FSM, and use your student ID as the states. For example: SID = 50123457, the transition of the LEDs will perform as follows, with the push button operation:

00000000 -> 00000101 -> 00000000 -> 00000001 -> 00000010 ->

00000011 -> 00000100 -> 00000101 -> 00000111 -> Reset -> 00000000

For this checkpoint, show the simulation result to the tutor.

Checkpoint 5: Discuss in the lab report how to modify the VHDL source code to implement a simple 4-bit counter with LEDs showing the state.

~END~