FSM Short Guideline

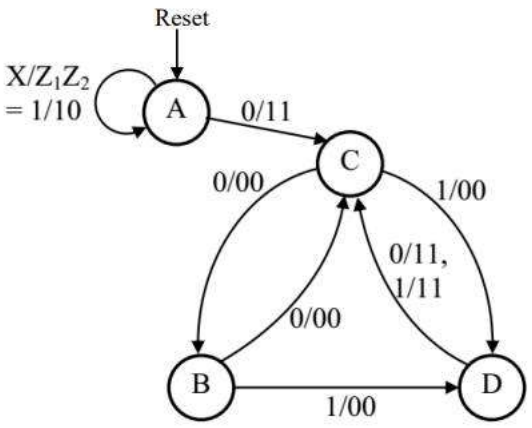
Page: VLSI Technology

Author: Quan Nguyen

Câu hỏi 2) (2.0đ):

Cho trước FSM có giản đồ trạng thái (graph trạng thái) như hình vẽ bên. Hệ có 1 ngõ vào là X, 2 ngõ ra Z_1 và Z_2. Khi có xung clock cạnh lên thì hệ chuyển trạng thái. Ngõ vào bất đồng bộ Reset tích cực cao (khi hệ bị Reset sẽ về trạng thái **A**). Gán trạng thái như sau: **A = 00**, **B = 11**, **C = 01**, và **D = 10**.

c) (1.0đ) Thành lập bảng chuyển trạng thái.
d) (1.0đ) Viết mã Verilog mô tả FSM này.



MSSV: ..... Họ và tên SV: ..... *Trang 1/2*

The above example is used to explain the coding method of a Mealy Finite State MAchine (FSM). First of all, you must understand the FSM theory and Mealy FSM. It can be found at the links:

FSM theory: https://www.youtube.com/watch?v=v5_JvWzJFA0

FSM structure: https://www.youtube.com/watch?v=SK0jWc-k-Ew

FSM coding: https://www.youtube.com/watch?v=I6TqiRhBank

Or you can refer to https://nguyenquanicd.blogspot.com/2017/08/verilogsystem-verilog-may-trang-thai.html .

Now, I assume that you understand the FSM structure and the difference between Moore and Mealy FSM.

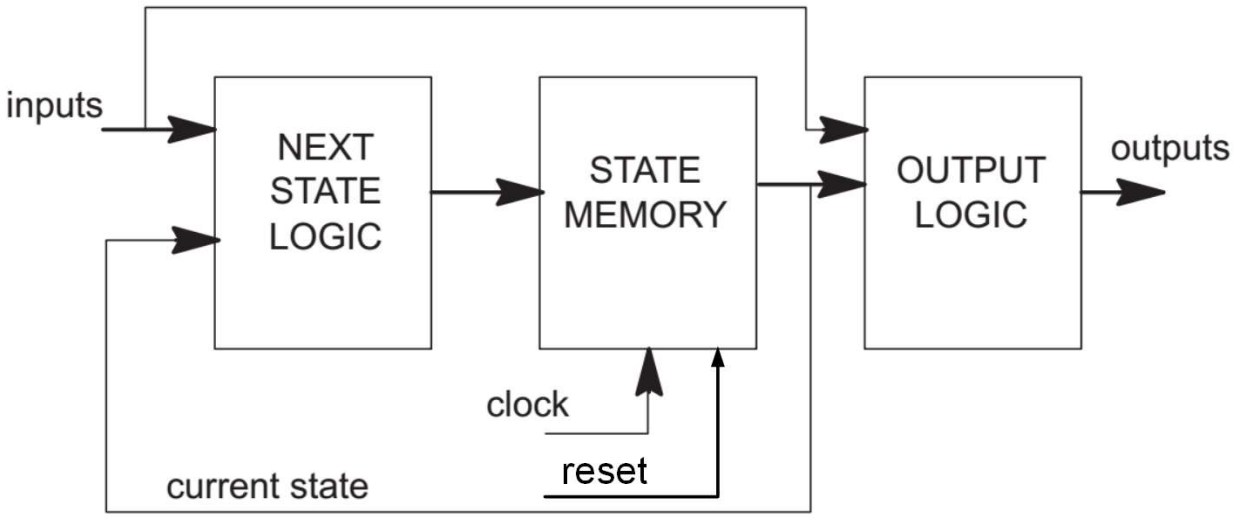


Figure 1: Mealy FSM

If you would like to code a Mealy FSM, you must write the RTL code for 3 parts that corresponds to 3 blocks (Next state logic, state memory/register, and output logic) in Figure 1: Mealy FSM. Some special notes are as follows:

- "Next state logic" is COMBINATIONAL circuit depends on inputs and the current state which is output of "state memory/register"
- "state memory/register" is SEQUENTIAL circuit depends on reset, clock and the next state which is output of "Next state logic"
- "output logic" is COMBINATIONAL circuit depends on inputs and the current state which is output of "state memory/register"

A common coding style is showed in the below.

1. State definition is described in the example requirement.

```
parameter A = 2'b00;

parameter B = 2'b10;

parameter C = 2'b01;
```

```
parameter D = 2'b10;
```

2. Declare the state register (current_state) and next state signal (input of current_state)

```
reg [1:0] next_state, current_state;
```

3. Next state logic

Ignore the value of outputs (Z1, Z2) and only focus on the state transition.

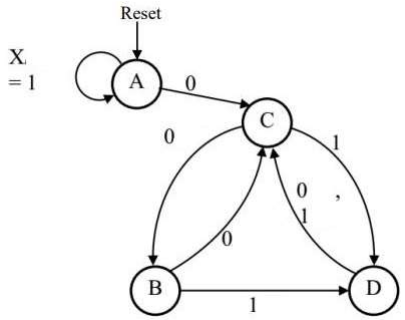


Figure 2: State transition of Mealy FSM (value of outputs are removed)

In "Figure 2: State transition of Mealy FSM (value of outputs are removed)", after the value of outputs is removed, the state transition is in below:

- If current_state is A
  - next_state is C if the input X is 0
  - next_state is A if the input X is 1
- If current_state is B
  - next_state is C if the input X is 0
  - next_state is D if the input X is 1
- If current_state is C
  - next_state is B if the input X is 0
  - next_state is D if the input X is 1

- If current_state is D
    - next_state is C if the input X is 0 or 1 -> It means D always jump to C regardless of the value of the input X.

You can see that the state transition is a “case statement” or “if-else statement”.

```
always @ (*) begin
  case (current_state[1:0])
    A: next_state[1:0] = X? A: C;
    B: next_state[1:0] = X? D: C;
    C: next_state[1:0] = X? D: B;
    D: next_state[1:0] = C;
    default: next_state[1:0] = 2'bxx;
  endcase
end
```

4. State memory/register

RTL code of “state memory/register” is implemented the reset description and connected with the “next state logic”.

Khi có xung clock cạnh lên thì hệ chuyển trạng thái. Ngõ vào bất đồng bộ Reset tích cực cao (khi hệ bị Reset sẽ về trạng thái **A**). Gán trạng thái như sau: **A = 00**, **B = 11**, **C = 01**, và **D = 10**.

```
always @ (posedge Clk or posedge Reset) begin
  if (Reset) current_state[1:0] <= A;
```

```
  else current_state[1:0] <= next_state[1:0];

end
```

5. Output logic

Ignore the state transition and focus on the current state and value of inputs.

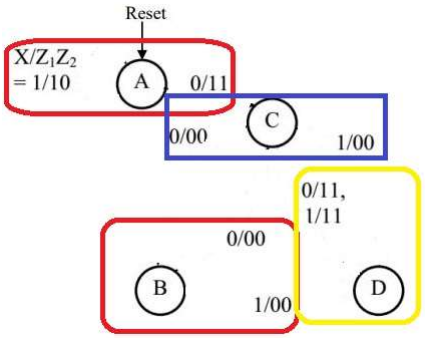


Figure 3: Value of outputs depends on the value of inputs and the current state

In “Figure 3: Value of outputs depends on the value of inputs and the current state”, after the value of outputs is removed, the state transition is in below:

- If current_state is A
  - Z1Z2 = 11 if the input X is 0
  - Z1Z2 = 10 if the input X is 1
- If current_state is B
  - Z1Z2 = 00 if the input X is 0
  - Z1Z2 = 00 if the input X is 1
- If current_state is C
  - Z1Z2 = 00 if the input X is 0
  - Z1Z2 = 00 if the input X is 1
- If current_state is D
  - Z1Z2 = 11 if the input X is 0

- Z1Z2 = 11 if the input X is 1

You can see that the output logic is a “case statement” or “if-else statement”.

```
always @ (*) begin
  case (current_state[1:0])
    A: {Z1, Z2} = X? 2’b10: 2’b11;
    B: {Z1, Z2} = 2’b00;
    C: {Z1, Z2} = 2’b00;
    D: {Z1, Z2} = 2’b11;
    default: {Z1, Z2} = 2’bxx;
  endcase
end
```

Note, this is only a common coding style. In this case, RLT code can be shorter.