Australian National University | ANU College of ENGINEERING, COMPUTING AND CYBERNETICS

ENGN 4213/6213 Digital Systems and Microprocessors

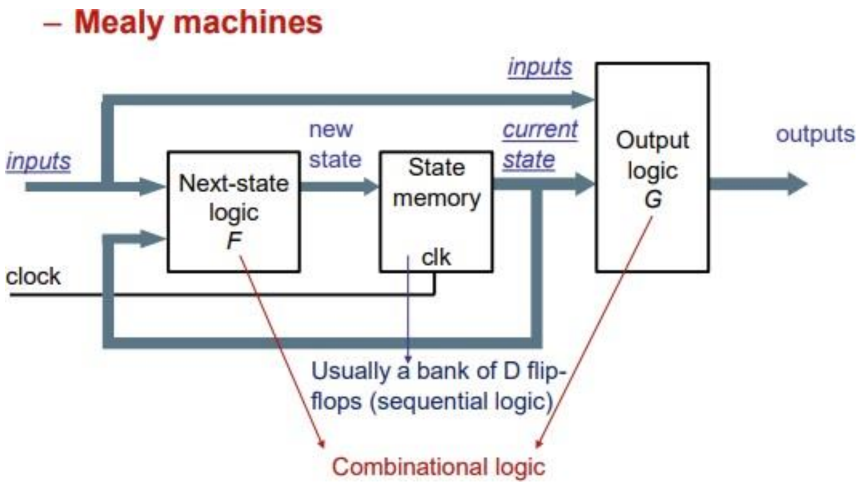
Semester 1, 2023

Tutorial 4 – Worked Solutions

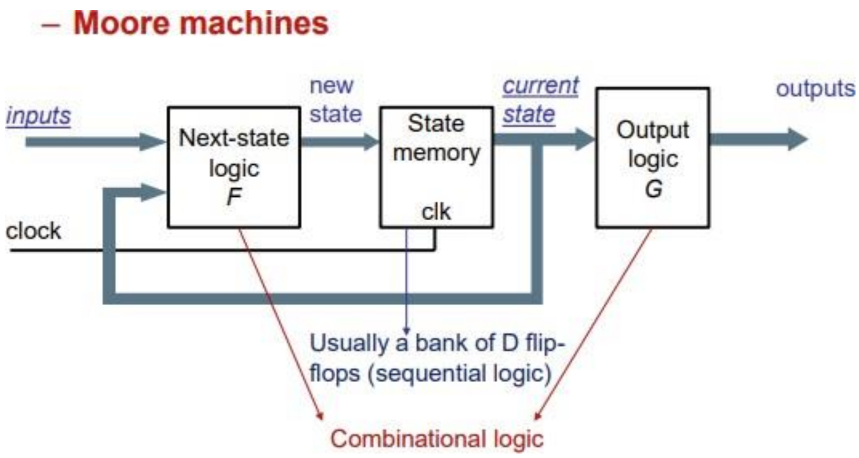
Question 1:

a. Describe the difference between a **Mealy** and a **Moore** machine showing a block diagram for each with the three key blocks and their inputs/outputs. Also, give a simple example for both.

Mealy machine: the output depends on both the state and the input, a simple example could be an elevator.



Moore machine: the output only depends on the state, example could be the FSM question in last tutorial.



b. Why do Mealy machines have possible asynchronous outputs (unlike Moore machines) and how can we solve this (i.e. having synchronous outputs)?

Mealy outputs may change when an input changes (can occur not on a clock edge). This can cause asynchronous behaviour and cause glitches in the output. Can make the inputs synchronous by processing through the same clock as the state memory. Moore machines only depend on current state which always changes on a clock edge, thus doesn't have the same problem.

c. Why might someone choose to encode FSM states as binary, gray, or one-hot? What would the advantages and disadvantages of each be?

Binary encoding: each state is assigned a unique binary code using natural binary sequence

- Requires least number of bits and therefore number of flip-flops to represent the states (consideration if hardware resource is constrained)
- Susceptible to glitches, any change in state requires multiple bit changes

Gray encoding: binary encoding where consecutive state codes differ by only one bit

- Least number of flip-flops
- More resistant to glitches, when bit error occurs the severity may be reduced
- Decoding logic complicated

One-hot encoding: only one of the bits of the state variable is set as '1'

- Decoding straightforward
- Uses more flip-flops

d. What input and state should every FSM design have in order for the user to come back to a "safe" and known behaviour?

Every FSM should have an "idle" state that the system is forced to go to when a reset input is ON. This allows for a safe state when things go wrong whereupon the behaviour of the FSM is known. (Also when you simply want the system to reset)

Question 2:

For this question, follow the order given below to develop a Finite State Machine:

1. Determine the inputs/outputs and the required number of states
2. Draw a state diagram
3. Make a next state table, transition table, and output table

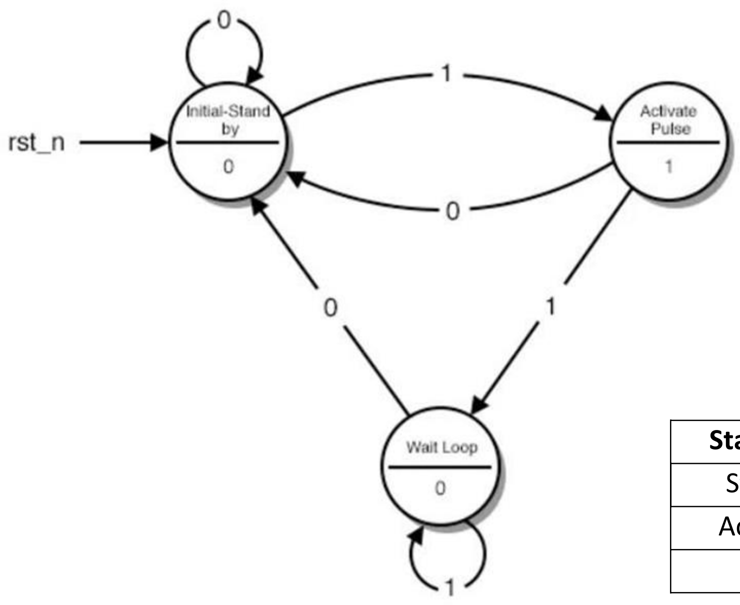
Design an FSM to simply transmit a HIGH pulse with a duration of only one clock cycle when some manual button is pressed and won't transmit another pulse until the button is released and pressed again.

Step 1. Determine inputs/outputs and number of states

This FSM is best designed to be a Moore machine.
It will have a single input being the button (excluding the reset input) and a single output being the HIGH pulse, which we can call an LED.
This FSM will require 3 states:

- Stand-by / idle (waiting for button to be pressed, or reset)
- Activate pulse (occurs when button has been pressed and will remain in state for a single clock cycle before moving back to IDLE or to the next state, depending if button is still pressed or not)
- Wait loop (occurs when button is still pressed and stays in such until button is released)

Step 2. State diagram:



A State Diagram

State name	State value
Stand-by	00
Activation	01
Wait	10

Step 3. Tables

Current State		Input	Next State	
A	B	I	Anext	Bnext
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0

Since this is a Moore machine, the output is only a function of current state:

Current State	Output
00 (Stand-by)	0
01 (Activation pulse)	1
10 (Wait loop)	0

Question 3:

In this question we practice designing a complex multi-module pin-code door access system that is controlled by a state machine. Based on the information and module instantiation provided below, complete the block diagram with the appropriate module name and module input/output.

- Given a very simple keypad (with the keys 0 and 1), the user is to enter an 8-digit code, which is compared with a list of valid pin codes held in the system's memory.
- If the pin is found amongst the valid codes, access is granted and the door may be opened. Once the door is shut again, the system will wait for a new pin input.
- If the pin entered by the user is not valid, access will not be granted; the system will reset and wait

for a new input.

- Buttons A and B represent the 0 and 1 keys, there is also a reset button to cancel inputting a pin if a mistake has been made. Buttons must be debounced where required.
- As each keystroke is entered, a LED should light up indicating how many keystrokes have been entered (i.e., 8 LEDs light up when a full pin has been entered).
- If access is granted, a particular LED combination should indicate this (e.g. having every second LED on).
- The door could be represented by one of the sliding switches. As the door is unlocked by the system, it must be opened (switch up) and closed again (switch back down) for the system to be reset.

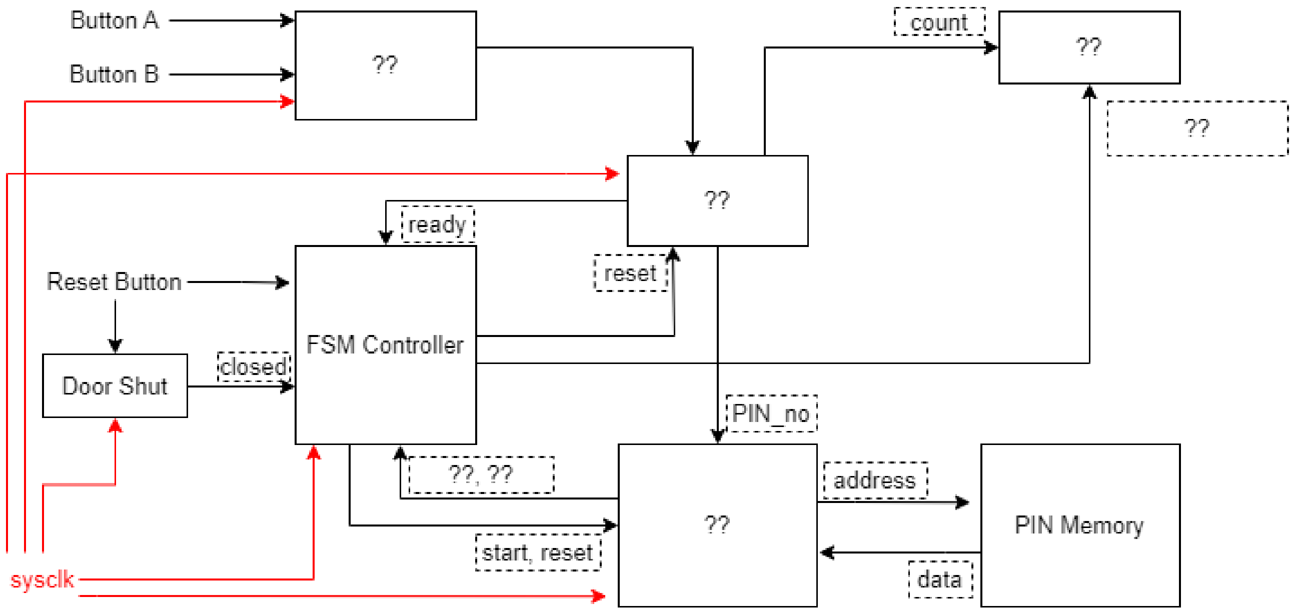


Figure 1 Block diagram of the door access system

```
module fsm_controller(
     input wire sysclk,
     input wire ready,
     input wire done,
     input wire closed_fsm,
     input wire found,
     input wire reset_btn,
     output reg start,
     output reg reset,
     output reg acc_gran);

module doubledebounce (
     input wire X0,
     input wire X1,
     input wire sysclk,
     input wire reset,
     output wire X0_deb,
     output wire X1_deb //X0 represents the input for the "0" key, X1 for
     //the "1" key
     );

module keyproc(
```

```
    input wire X0_deb,
    input wire X1_deb,
    input wire reset,
    input wire sysclk,
    output wire ready,
    output reg [3:0] count,
    output reg [7:0] pin_password);

module LEDproc(
    input wire [3:0] count,
    input wire acc_grant,
    output reg [7:0] LEDS);

module PIN_memory (
    input wire [3:0] addr,
    output reg [7:0] data);

module pincomp(
    input wire start,
    input wire [7:0] pin_code,
    input wire [7:0] data,
    input wire sysclk,
    input wire reset,
    output reg [3:0] addr,
    output wire done,
    output wire found);

module doorshut(
    input wire reset,
    input wire closed_sw,
    input wire sysclk,
    output wire closed_fsm);
```

Completed block diagram: