Imperial College
London

Lecture 6

Finite State Machine – Part 1

Peter Cheung
Department of Electrical & Electronic Engineering
Imperial College London

URL: www.ee.imperial.ac.uk/pcheung/teaching/ee2_digital
E-mail: p.cheung@imperial.ac.uk

In this section of the course, we will consider the design and specification of finite state machine (FSM). FSM is one of the most important topics in digital electronics. It provides a formal methodology for a designer to translate specification of a digital control circuit to actual circuits.

Lecture Objectives

- Review the definition of a synchronous finite state machine (FSM or SSM)
- Learn how to construct the state table and state diagram of a state machine from its circuit diagram
- Appreciate the alternative ways of drawing the state diagram
- Learn how to draw the output waveforms of a state machine given its initial state and input waveforms
- Understand the causes of glitches in state machine outputs
- To learn how to design a state machine to meet specific objectives
- To learn how to design a FSM without the use of CAD tools (i.e. manually)

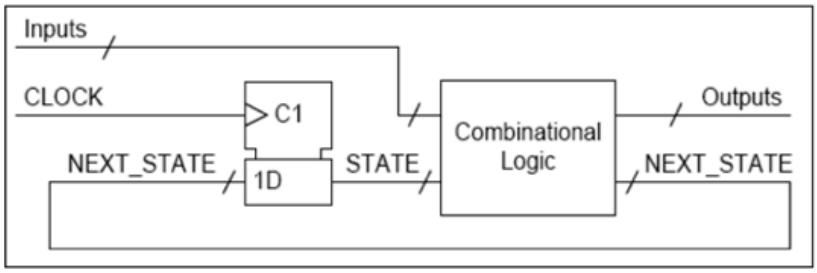
In the previous lecture, we examined how to use counters and shift registers to produce arbitrary digital signals that could be used as control signals to a digital system. While such circuits could be fast (particularly with shift registers), the design process is not systematic. For complicated control logic, it is far better to design such circuit as a synchronous (or finite) state machine.

In this lecture, we will examine how we can analyze the working of a finite state machine (FSM) through three different representations: state table, state diagram and digital waveforms.

(FSM is sometimes known as synchronous state machine or SSM. The two are synonymous.)

Synchronous State Machines

- **Synchronous State Machine (also called Finite State Machine FSM)**
  = Register + Logic



  - The ***state*** is defined by the register contents
  - Register has *n* flipflops ⇒ 2^n states
  - The state only ever changes on CLOCK↑
    - We stay in a state for an exact number of CLOCK cycles
  - The state is the only memory of the past

Rules:

- Never mess around with the clock signal
- Never use ***asynchronous*** SET/RESET inputs to register (*asynchronous* = independent of CLOCK)

PYKC 22 Oct 2019 E2.1 Digital Electronics Lecture 6 Slide 3

Here is a simplified generic diagram of a finite (or synchronous) state machine (FSM or SSM). A set of D-flipflips are used to store the current state value. The current state together with external inputs are fed to a combinational logic circuit to evaluate two things: the **next state** and the **current outputs**.

With an n-bit register and using binary state encoding (i.e. coding states as binary numbers), such machine can have a maximum of 2^n states.

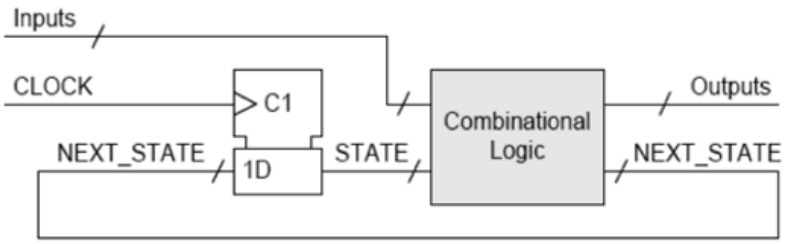
This is a synchronous state machine because the transition to the next state is synchronous with the rising edge of the clock signal. Therefore all output signals are synchronized.

There are two basic rules in designing a FSM that operates reliably:

1. Do not put logic in front of the clock signal. Doing so is likely to cause timing issues when the FSM is used in conjunction with the rest of the system.

2. Do not use asynchronous SET or RESET signals. Doing so would make the rest of the system NOT synchronous to the CLOCK signal.

Combinational Logic Block



- The combinational logic outputs specify two things:
  - **the output signals during the current state**
    These may change during the state if the inputs change
  - **which state to go to at the next CLOCK**
    This too may change during a state but the only thing that matters is its value just before CLOCK
- ***combinational*** logic has no internal feedback loops ⇒ no memory
  - combinational logic outputs are entirely determined by the **current STATE** and the **current Inputs**

PYKC 22 Oct 2019 E2.1 Digital Electronics Lecture 6 Slide 4

The combinational logic circuit in a FSM performs two separate tasks:

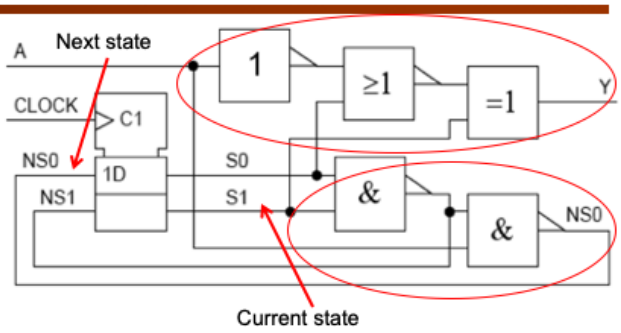
1. It determines what **the output signals** should be. This derived by the current state value STATE and the current inputs. Therefore such output signals could change in the middle of a clock cycle if input signals are NOT synchronized with the CLOCK.
2. It determines what the **next state value** should be, i.e. the state transition of the FSM.

The combinational logic block (by definition) contains no memory (or register) circuit.

Analysing a State Machine

State Table:

- Truth table for the combinational logic:
  - One row per state: n flipflops $\Rightarrow 2^n$ rows
  - One column per input combination: m input signals $\Rightarrow 2^m$ columns
  - Each cell specifies the ***next state*** and the ***output signals during the current state***
    - for clarity, we separate the two using a /



NS1,NS0/Y		
S1,S0	A=0	A=1
00	11/0	10/1
01	11/0	10/0
10	11/1	10/0
11	01/1	01/1

Shown here is a simple FSM in details. The upper group of gates are used to compute the output signal Y. The lower group of gates are used to work out the next state values NS0 and NS1.

We will now analyse how this circuit works. One powerful tool that we can use is to the state transition table. It is similar to the truth table used for combinational circuit, but is used to show the function of the FSM.

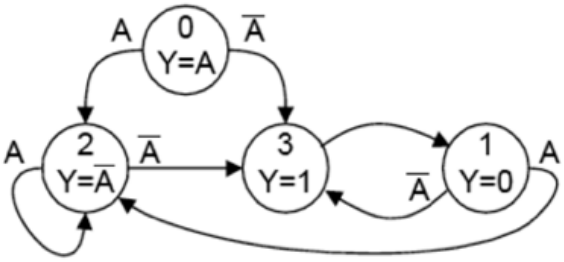
Each row in this table represents one state. Since this FSM has 2 state bits, there are 4 possible states.

There is one column devoted to each input combination. In this case, there is only one input A. There would be four columns if there were two inputs.

The contents of the table shows the next state transition, followed by the output signal(s) during the current state. A '/' character is used to separate the two.

Drawing the State Diagram

- Split state table into two parts: next state table and output table



NS1,NS0/Y

S1,S0	A=0	A=1
00	11/0	10/1
01	11/0	10/0
10	11/1	10/0
11	01/1	01/1

Next State: NS1:0

S1:0	A=0	A=1
0	3	2
1	3	2
2	3	2
3	1	1

- Transition arrows are marked with Boolean expressions saying when they occur
  - Every input combination has exactly one destination.
  - Unlabelled arrows denote unconditional transitions
- Output Signals: Boolean expressions within each state

Output Signal: /Y

S1:0	A=0	A=1	
0	/0	/1	Y=A
1	/0	/0	Y=0
2	/1	/0	Y=!A
3	/1	/1	Y=1

Another very powerful tool to show the function of a FSM is to use state diagram (one that uses “bubbles”). For clarity, let us split the state transition table into two tables: one for next state NS1:0, and another for the output signal Y.

We now draw a bubble for each state and label this with the state name (which happens in this case to be the same as the state value). Transition arrows are draw between the states with a Boolean expression as a label to indicate the condition required for the transition to occur ON THE ACTIVE CLOCK EDGE (positive edge in this case). The transitions are derived direclty from the next state table. Consider state 0, on rising edge of CLOCK, if A=0, go to state 3, else if A=1, go to state 2.

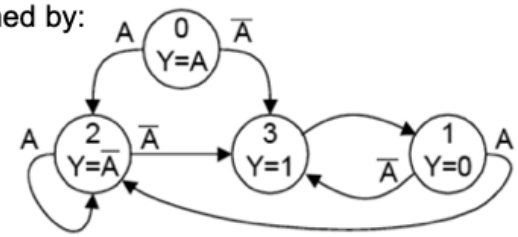
Inside the bubble, we now indicate the value of Y as another Boolean expression.

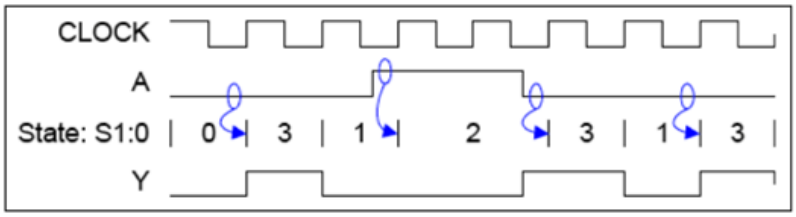
In this example, we perform analysis of a circuit designed by someone else. Therefore we derive the transition table from the circuit, then the state diagram from the state transition table.

When we are designing a FSM from a specification, we usual do this the other way round, i.e. design the state diagram from the specification, then draw up the state transition table as required and derive the circuit from that.

Timing Diagram

- State machine behaviour is entirely determined by:
  - The initial state
  - The input signal waveforms
- **State Sequence:**
  - *Determine this first.* Next state depends on input values just before CLOCK





- **Output Signals:**
  Defined by Boolean expressions within each state.
  If all the expressions are constant 0 or 1 then outputs only ever change on clock . (***Moore machine***)
  If any expressions involve the inputs (e.g. Y=A) then it is possible for the outputs to change in the middle of a state. (***Mealy machine***)

PYKC 22 Oct 2019 E2.1 Digital Electronics Lecture 6 Slide 7

It is important to note that the behaviour of a FSM is determined by the initial state. Given the state diagram and the initial state (assumed here to be state 0), and waveform of the input A, we can easily trace the subsequent states S1:0 and the output Y.

For our course, we are exclusively using FPGA. For us, state initialization is part of the FPGA configuration process – we can use the Verilog "**initial**" statement to perform this initialization. (More later.)

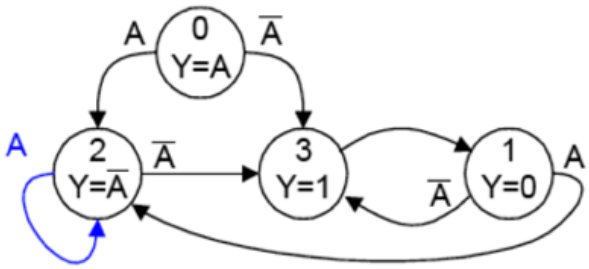
There are two types of state machines:

1. **Moore** machine – the outputs are constant 0 or 1 while inside a state even if input changes. The output only changes on the active clock edge. An example of a Moore machine is a counter. For our designs with Verilog, our FSMs are usually a Moore machine (as will be seen in a later lecture).
2. **Mealy** machine – the outputs could changes if input changes even without an active clock edge.

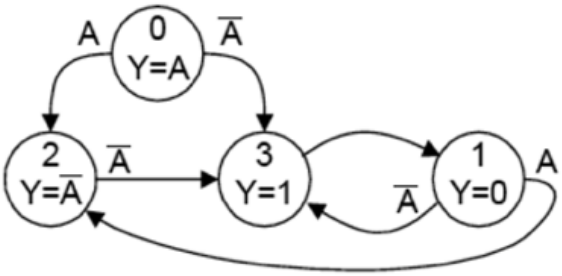
The FSM here is a Mealy machine because the output Y inside state 0 and 2 are Boolean expressions. If A changes in the middle of a clock cycle, the output Y will change immediately. So the output is NOT dependent on the state of the machine alone.

Self-Transitions

- We can omit transitions from a state to itself
  - Aim: to save clutter on the diagram



- The state machine remains in its current state if none of the transition-arrow conditions are satisfied
  - From state 2, we go to state 3 if !A occurs, otherwise we remain in state 2



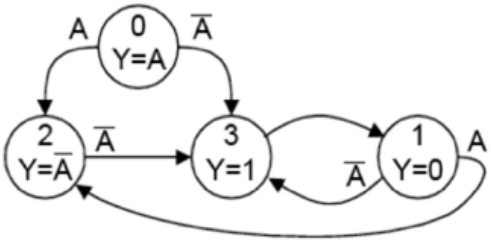
PYKC 22 Oct 2019 E2.1 Digital Electronics Lecture 6 Slide 8

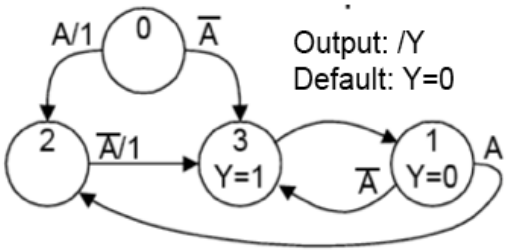
In order to make the state diagram less cluttered, you can omit the self transition arrows. Therefore the rule is that a state machine stays in its current state unless the conditions of an exiting arrow is satisfied.

In this example, we stay in state 2 until A = 0 on the rising edge of CLOCK. Then we go to state 3.

Output Expressions on Arrows

- It may make the diagram clearer to put output expressions on the arrows instead of within the state circles:
  - Useful if the same Boolean expression determines both the ***next state*** and the ***output signals***
  - For each state, the output specification must be ***either*** inside the circle *or else* on ***every*** emitted arrow
  - If self transitions are omitted, we must declare default values for the outputs





Output: /Y
Default: Y=0

- Outputs written on an arrow apply to the state ***emitting*** the arrow.
- Outputs still apply for the entire time spent in a state
- This does not affect the Moore/Mealy distinction
- This is a notation change only

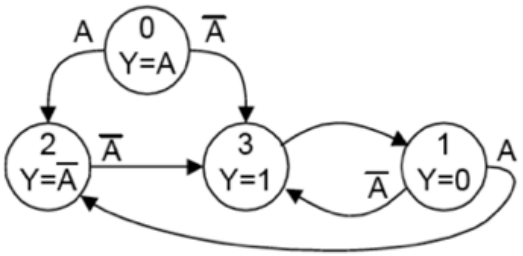
PYKC 22 Oct 2019 E2.1 Digital Electronics Lecture 6 Slide 9

Instead of specifying outputs inside the state bubble, it is also possible to specify outputs on the transition arrow. There are a few rules that you must follow:

1. For each state, you must specify the output either inside the bubble or on EVERY emitted arrow from the state.
2. You can mix the two conventions in a state diagram, but you must use only one method for each (and not mixing them).
3. If you use self transition, as in state 2 here, you must declare the default values for each outputs.
4. Output written on an arrow always applies to the state **EMITTING** the arrow (i.e. source not destination).

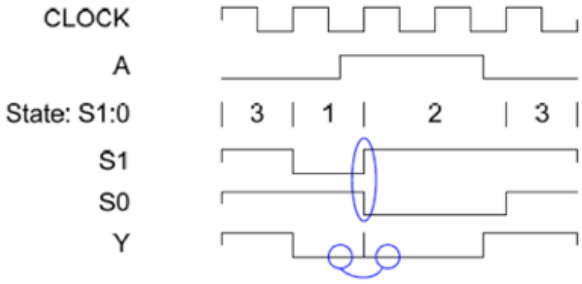
Output Glitches

- When making a transition from one state to another, the logic is likely to generate a glitch on an output if:
  - two or more state bits change
  - the output has the same value in both states
  - some combination of the changing state bits would cause the output to change



In changing from state 1 to state 2:

- the two states differ in both S0 and S1
- the output is low in both states
- if S0 and S1 both went high then the output would change.



PYKC 22 Oct 2019 E2.1 Digital Electronics Lecture 6 Slide 10

Let us examine the state machines output Y in detail. When we transit from state 1 to state 2, it is possible that we pass through state 0 for a very short time. This is because when S1:0 = 01 moves to S1:0 = 10, S1 could be slightly slower than S0 in the transition. Hence S1:0 = 00 occurs temporarily.

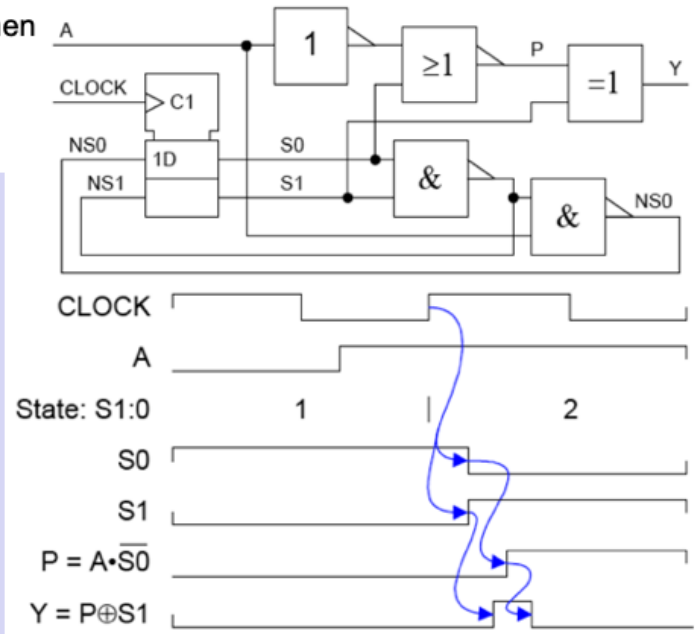
In this case, Y could produce a short glitch (because A=1, and Y=A) during the transition.

In general, a glitch could be produced if:

1. Two or more state bits change values when transiting states
2. The output has the same value in both states

Cause of Output Glitches

- Look in detail at the logic when going from state 1 to 2:



- The two inputs to the XOR gate (P and S1) are meant to change simultaneously.
- In fact S1 changes first because of the delay through the NOR gate.
- The XOR gate "sees" the effect of S1 changing before it "sees" the effect of S0 changing. It is as if we went briefly into state 3.

PYKC 22 Oct 2019 E2.1 Digital Electronics Lecture 6 Slide 11

Let us consider the circuit of this FSM in detail to discover why a glitch may be produced at the Y output.

Y is P XOR S1. Since P is produce by the NOR gate, it will change later than S1. Therefore the XOR gate will see the short duration that P and S1 are different. Hence a short pulse will be produced.

Often such glitches are NOT important unless it is used as a clock input signal to another circuit. However, it is important to be aware of such glitches.

You could eliminate such glitches by putting another set of D-FF at the output signals. However, this will also produce a one-cycle delay to the output. We will consider this in a later lecture.

Quiz Questions

1. What is the definition of a Moore machine?
2. What does it mean if an arrow in a state diagram has no Boolean expression attached to it?
3. To which state does an output value refer when it is marked on an arrow in a state diagram? Is it the state the arrow points *towards* or the state the arrow points *away from*?
4. Is the next state determined by the value that the input signals have just *before* or just *after* the CLOCK↑?
5. If transitions from a state to itself have been omitted from a state diagram, how can you tell when such a transition occurs?
6. What are the three conditions that give rise to output glitches?

Answers are all in the notes.