Digital Electronics: Sequential Logic

Synchronous State Machines 1

Introduction

- We have seen how we can use FFs (D-types in particular) to design synchronous counters
- We will now investigate how these principles can be extended to the design of synchronous state machines (of which counters are a subset)
- We will begin with some definitions and then introduce two popular types of machines

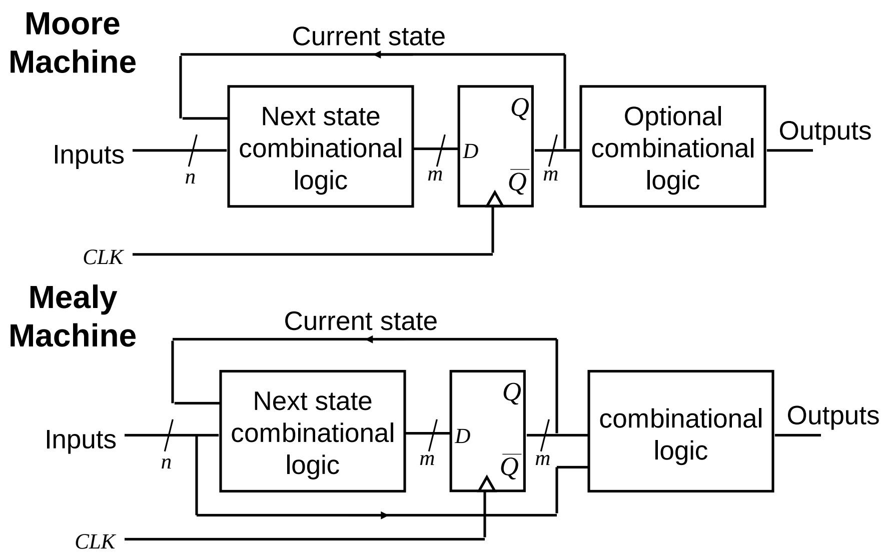
Definitions

- **Finite State Machine (FSM)** – a deterministic machine (circuit) that produces outputs which depend on its internal state and external inputs
- **States** – the set of internal memorised values, shown as circles on the state diagram
- **Inputs –** External stimuli, labelled as arcs on the state diagram
- **Outputs –** Results from the FSM

Types of State Machines

- Two types of state machines are in general use, namely *Moore* machines and *Mealy* machines
- We will see that the state diagrams (and associated state tables) corresponding with the 2 types of machine are slightly different

Machine Schematics

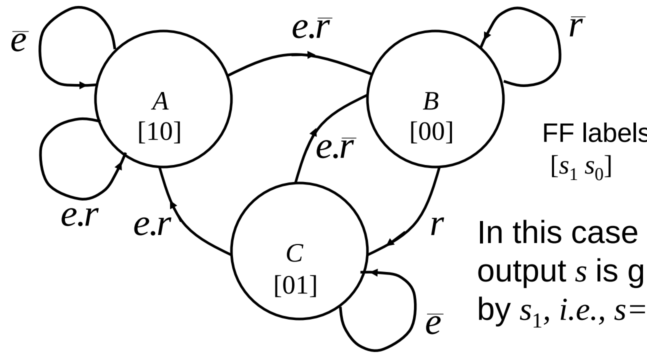


Moore vs. Mealy Machines

- Outputs from Mealy Machines depend upon the timing of the inputs
- Outputs from Moore machines come directly from clocked FFs so:
  - They have guaranteed timing characteristics
  - They are glitch free
- Any Mealy machine can be converted to a Moore machine and vice versa, though their timing properties will be different

Moore Machine State Diagram

- Example FSM has 3 states (*A*, *B* and *C*), inputs *e* and *r*, and output *s*

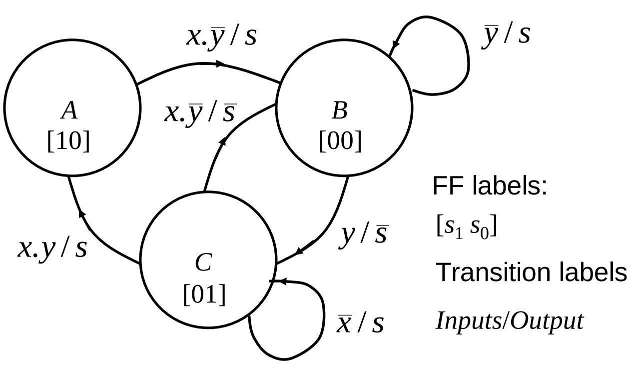


In this case the output *s* is given by s_1, *i.e.*, $s=s_1$

- See **inputs only** appear on transitions between states, i.e., next state is given by current state and current inputs
- Outputs determined from current state via combinational logic (if required)

Mealy Machine State Diagram

- Example FSM has 3 states (*A*, *B* and *C*), inputs *x* and *y*, and output *s*

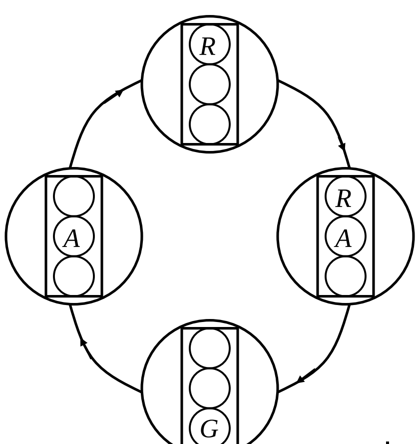


- Inputs **and outputs** appear on transitions between states, i.e., next state is given by current state and current inputs
- Output determined from current state and inputs via combinational logic

Moore Machine - Example

- We will design a Moore Machine to implement a traffic light controller
- In order to visualise the problem it is often helpful to draw the state transition diagram
- This is used to generate the state transition table
- The state transition table is used to generate
  – The next state combinational logic
  – The output combinational logic (if required)

Example – Traffic Light Controller



See we have 4 states

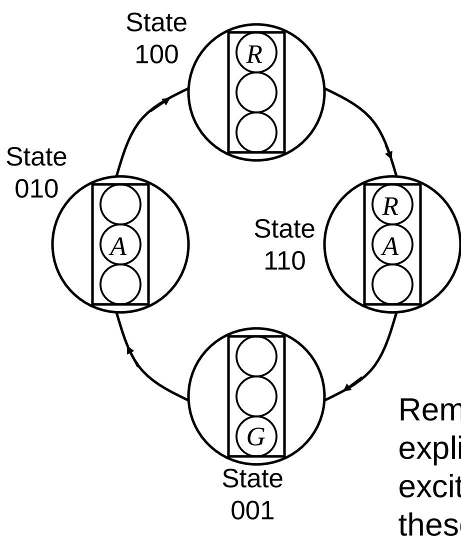
So in theory we could use a minimum of 2 FFs

However, by using 3 FFs we will see that we do not need to use any output combinational logic

So, we will only use 4 of the 8 possible states

In general, state assignment is a difficult problem and the optimum choice is not always obvious

Example – Traffic Light Controller



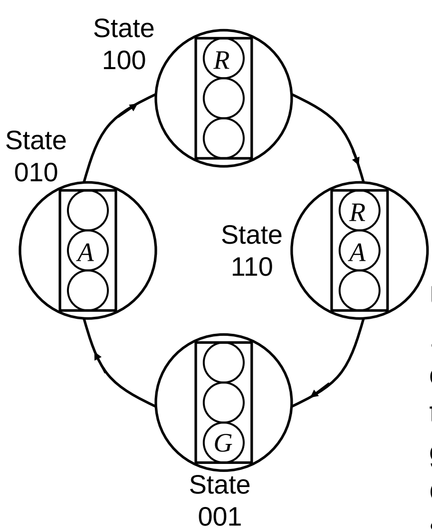
By using 3 FFs (we will use D-types), we can assign one to each of the required outputs (R, A, G), eliminating the need for output logic

We now need to write down the state transition table

We will label the FF outputs R, A and G

Remember we do not need to explicitly include columns for FF excitation since if we use D-types these are identical to the next state

Example – Traffic Light Controller



Current state			Next state		
R	A	G	R'	A'	G'
1	0	0	1	1	0
1	1	0	0	0	1
0	0	1	0	1	0
0	1	0	1	0	0

Unused states, 000, 011, 101 and 111. Since these states will never occur, we don't care what output the next state combinational logic gives for these inputs. These don't care conditions can be used to simplify the required next state combinational logic

Example – Traffic Light Controller

Current state / Next state

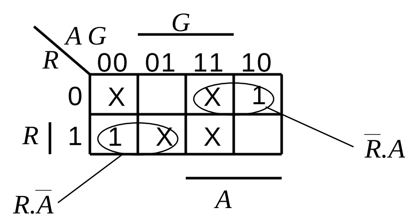
R	A	G	R'	A'	G'
1	0	0	1	1	0
1	1	0	0	0	1
0	0	1	0	1	0
0	1	0	1	0	0

Unused states, 000, 011, 101 and 111.

We now need to determine the next state combinational logic

For the R FF, we need to determine D_R

To do this we will use a K-map



$$D_R = R.\overline{A} + \overline{R}.A = R \oplus A$$

Example – Traffic Light Controller

Current state / Next state

R	A	G	R'	A'	G'
1	0	0	1	1	0
1	1	0	0	0	1
0	0	1	0	1	0
0	1	0	1	0	0

Unused states, 000, 011, 101 and 111.

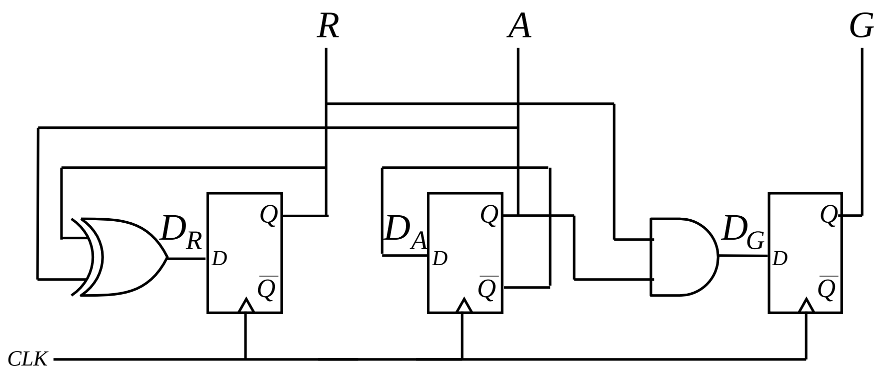
By inspection we can also see:

$$D_A = \overline{A}$$

and,

$$D_G = R.A$$

Example – Traffic Light Controller



FSM Problems

- Consider what could happen on power-up
- The state of the FFs could by chance be in one of the unused states
  - – This could potentially cause the machine to become stuck in some unanticipated sequence of states which never goes back to a used state

FSM Problems

- What can be done?
  - Check to see if the FSM can eventually enter a known state from any of the unused states
  - If not, add additional logic to do this, i.e., include unused states in the state transition table along with a valid next state
  - Alternatively use asynchronous Clear and Preset FF inputs to set a known (used) state at power up

Example – Traffic Light Controller

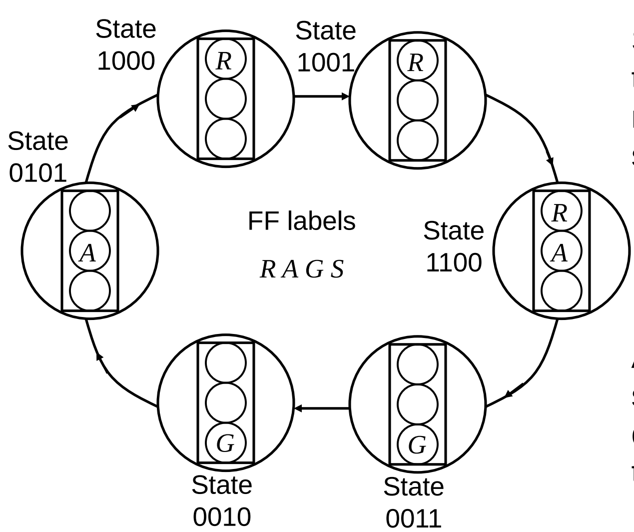
- Does the example FSM self-start?
- Check what the next state logic outputs if we begin in any of the unused states
- Turns out:

Start state	Next state logic output		
000	010	Which are all valid states	So it does self start
011	100		
101	110		
111	001		

Example 2

- We extend Example 1 so that the traffic signals spend extra time for the R and G lights
- Essentially, we need 2 additional states, i.e., 6 in total.
- In theory, the 3 FF machine gives us the potential for sufficient states
- However, to make the machine combinational logic easier, it is more convenient to add another FF (labelled S), making 4 in total

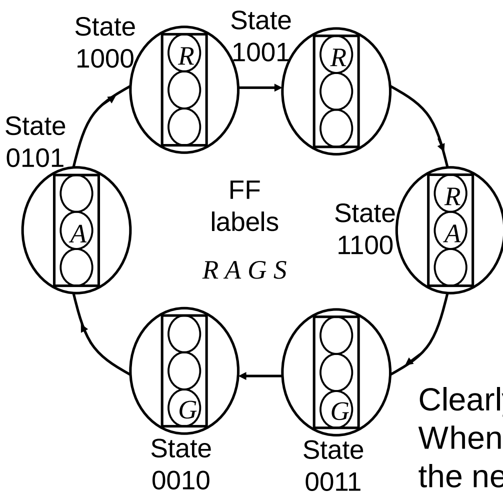
Example 2



See that new FF toggles which makes the next state logic easier

As before, the first step is to write down the state transition table

Example 2



Current state				Next state			
R	A	G	S	R'	A'	G'	S'
1	0	0	0	1	0	0	1
1	0	0	1	1	1	0	0
1	1	0	0	0	0	1	1
0	0	1	1	0	0	1	0
0	0	1	0	0	1	0	1
0	1	0	1	1	0	0	0

Clearly a lot of unused states. When plotting k-maps to determine the next state logic it is probably easier to plot 0s and 1s in the map and then mark the unused states

Example 2

Current state				Next state			
R	A	G	S	R'	A'	G'	S'
1	0	0	0	1	0	0	1
1	0	0	1	1	1	0	0
1	1	0	0	0	0	1	1
0	0	1	1	0	0	1	0
0	0	1	0	0	1	0	1
0	1	0	1	1	0	0	0

We will now use k-maps to determine the next state combinational logic

For the R FF, we need to determine D_R

RA \ GS	00	01	11	10
00	X	X	0	0
01	X	1	X	X
11	0	X	X	X
10	1	1	X	X

G A R S

$\overline{R}.A$

$R.\overline{A}$

$$D_R = R.\overline{A} + \overline{R}.A = R \oplus A$$

Example 2

Current state				Next state			
R	A	G	S	R'	A'	G'	S'
1	0	0	0	1	0	0	1
1	0	0	1	1	1	0	0
1	1	0	0	0	0	1	1
0	0	1	1	0	0	1	0
0	0	1	0	0	1	0	1
0	1	0	1	1	0	0	0

We can plot k-maps for D_A and D_G to give:

$$D_A = R.S + G.\overline{S} \quad \text{or}$$
$$D_A = R.S + \overline{R}.\overline{S} = \overline{R \oplus S}$$

$$D_G = R.A + G.S \quad \text{or}$$
$$D_G = G.S + A.\overline{S}$$

By inspection we can also see:

$$D_S = \overline{S}$$