Indian Institute of Technology Jodhpur, Year 2018-2019

Digital Logic and Design

(Course Code: EE222)

Lecture 25-26: Sequential Circuits Contd..

Course Instructor: Shree Prakash Tiwari

Email: sptiwari@iitj.ac.in

Webpage: http://home.iitj.ac.in/~sptiwari/

Course related documents will be uploaded on http://home.iitj.ac.in/~sptiwari/DLD/

1

State Assignment Problem

- **Some state assignments are better than others.**
- **The state assignment influences the complexity of the state machine.**
  - **The combinational logic required in the state machine design is dependent on the state assignment.**
- **Types of state assignment**
  - **Binary encoding:** 2^N **states → N Flip-Flops**
  - **Gray-code encoding:** 2^N **states → N Flip-Flops**
  - **One-hot encoding: N states → N Flip-Flops**

FSM: State Assignment

Example:

Design a FSM that detects a sequence of two or more consecutive ones on an input bit stream.

The FSM should output a 1 when the sequence is detected, and a 0 otherwise.

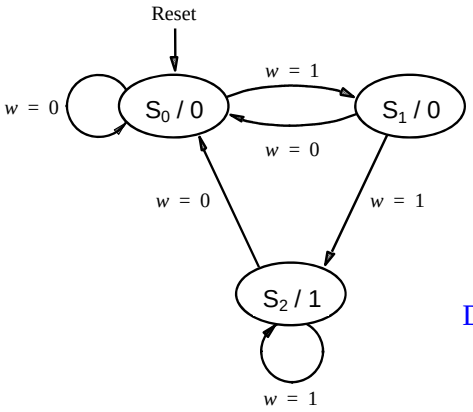
This is another example of a *sequence detector*.

FSM: State Assignment

Input: **0 1 1 1 0 1 0 1 1 0 1 1 1 0 1 …**

Output: **0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 …**

FSM: State Assignment



State Diagram

FSM: State Assignment

Present State	Next State		Output
	w = 0	w = 1	
S_0	S_0	S_1	0
S_1	S_0	S_2	0
S_2	S_0	S_2	1

State Table

FSM: State Assignment #1

State Assigned Table

Present State			Next State						Output
			w = 0			w = 1			
	Q_A	Q_B		Q_A^+	Q_B^+		Q_A^+	Q_B^+	z
S_0	0	0	S_0	0	0	S_1	0	1	0
S_1	0	1	S_0	0	0	S_2	1	0	0
S_2	1	0	S_0	0	0	S_2	1	0	1
	1	1		d	d		d	d	d

Using Binary Encoding
for the State Assignment

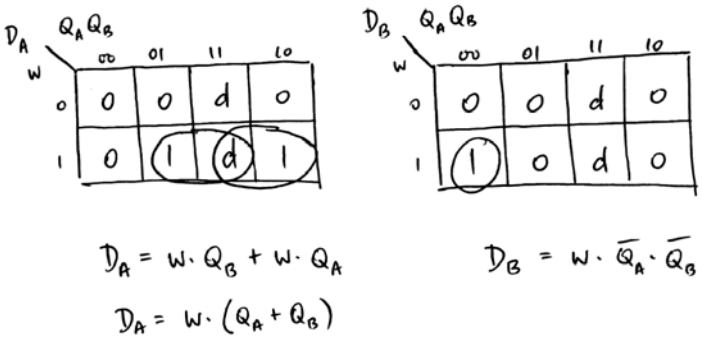
FSM: State Assignment #1

State Assigned Table

Present State			Next State				FF Inputs			
			w = 0		w = 1		w = 0		w = 1	
	Q_A	Q_B	Q_A^+	Q_B^+	Q_A^+	Q_B^+	D_A	D_B	D_A	D_B
S_0	0	0	0	0	0	1	0	0	0	1
S_1	0	1	0	0	1	0	0	0	1	0
S_2	1	0	0	0	1	0	0	0	1	0
	1	1	d	d	d	d	d	d	d	d

Characteristic Equation: $D = Q^+$

FSM: State Assignment #1

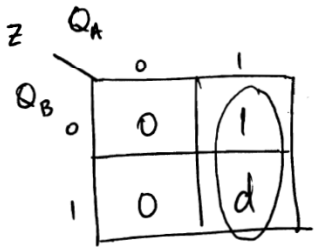


$D_A = W \cdot Q_B + W \cdot Q_A$

$D_A = W \cdot (Q_A + Q_B)$

$D_B = W \cdot \overline{Q}_A \cdot \overline{Q}_B$

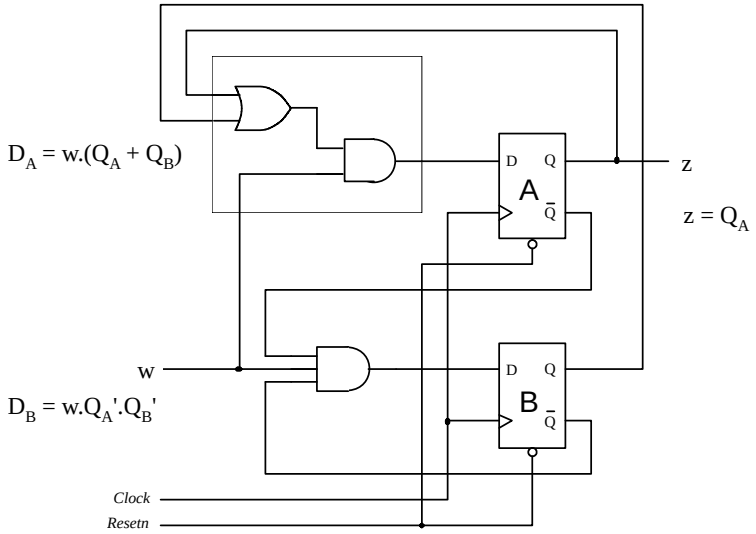
FSM: State Assignment #1



$z = Q_A$

K-Map and Boolean expression for z

FSM: State Assignment #1



FSM: State Assignment #2

State Assigned Table

Present State			Next State						Output
			w = 0			w = 1			
	Q_A	Q_B		Q_A^+	Q_B^+		Q_A^+	Q_B^+	z
S_0	0	0	S_0	0	0	S_1	0	1	0
S_1	0	1	S_0	0	0	S_2	1	1	0
S_2	1	1	S_0	0	0	S_2	1	1	1
	1	0		d	d		d	d	d

Using Gray-code Encoding for the State Assignment

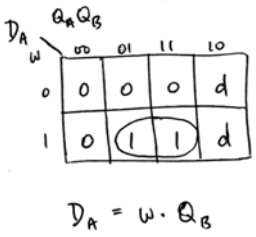
FSM: State Assignment #2

State Assigned Table

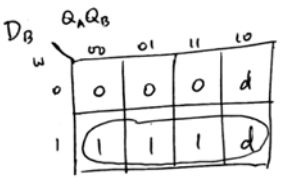
Present State			Next State				FF Inputs			
			w = 0		w = 1		w = 0		w = 1	
	Q_A	Q_B	Q_A^+	Q_B^+	Q_A^+	Q_B^+	D_A	D_B	D_A	D_B
S_0	0	0	0	0	0	1	0	0	0	1
S_1	0	1	0	0	1	1	0	0	1	1
S_2	1	1	0	0	1	1	0	0	1	1
	1	0	d	d	d	d	d	d	d	d

Characteristic Equation: $D = Q^+$

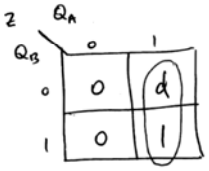
FSM: State Assignment #2



$D_A = w \cdot Q_B$

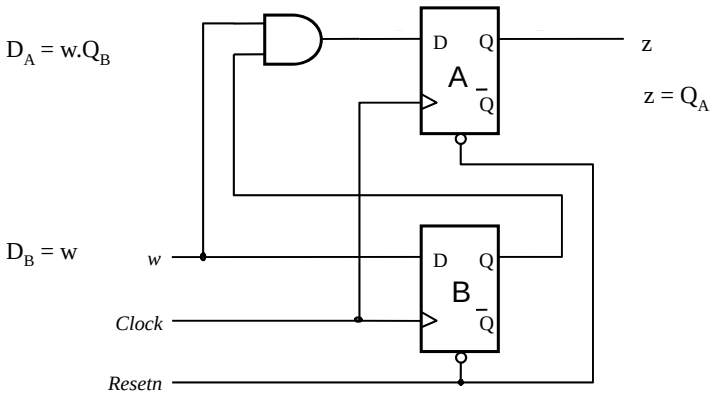


$D_B = w$



$z = Q_A$

FSM: State Assignment #2



FSM: State Assignment #3

State Assigned Table

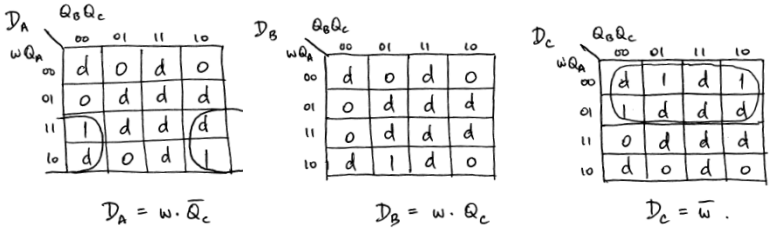
Present State				Next State							
				w = 0				w = 1			
	Q_A	Q_B	Q_C		Q_A^+	Q_B^+	Q_C^+		Q_A^+	Q_B^+	Q_C^+
S_0	0	0	1	S_0	0	0	1	S_1	0	1	0
S_1	0	1	0	S_0	0	0	1	S_2	1	0	0
S_2	1	0	0	S_0	0	0	1	S_2	1	0	0

Using One-hot Encoding for the State Assignment

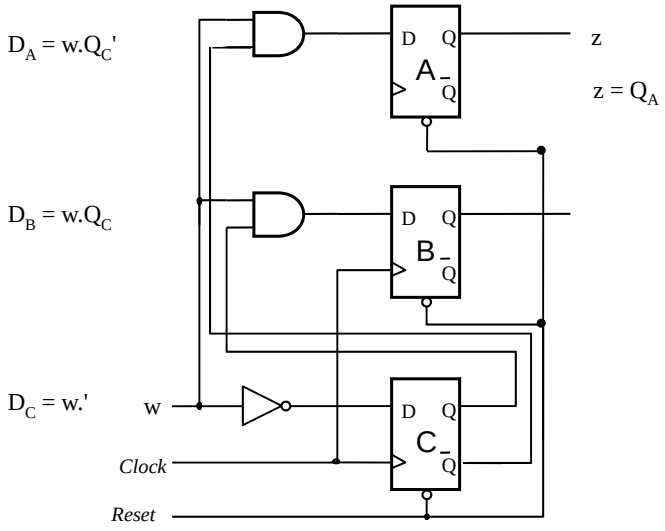
For each state only one flip-flop is set to 1.
The remaining combination of state variables are not used.

Characteristic Equation: $D = Q^+$

FSM: State Assignment #3



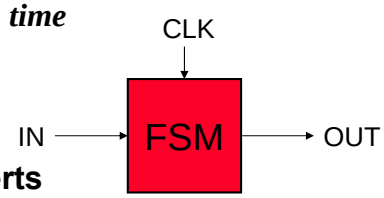
FSM: State Assignment #3



Finite State Machines: Other Examples

° Example: Edge Detector

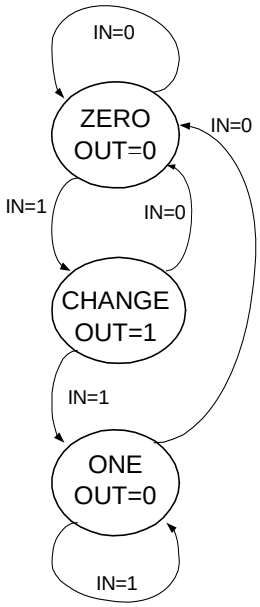
Bit are received one at a time (one per cycle), such as: 000111010 ⟶ *time*



Design a circuit that asserts its output for one cycle when the input bit stream changes from 0 to 1.

Try two different solutions.

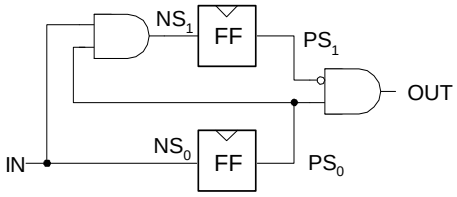
State Transition Diagram Solution A



	IN	PS	NS	OUT
ZERO	0	00	00	0
	1	00	01	0
CHANGE	0	01	00	1
	1	01	11	1
ONE	0	11	00	0
	1	11	11	0

Solution A, circuit derivation

	IN	PS	NS	OUT
ZERO	0	00	00	0
	1	00	01	0
CHANGE	0	01	00	1
	1	01	11	1
ONE	0	11	00	0
	1	11	11	0



PS

IN \ PS	00	01	11	10
0	0	0	0	-
1	0	1	1	-

$NS_1 = IN\ PS_0$

PS

IN \ PS	00	01	11	10
0	0	0	0	-
1	1	1	1	-

$NS_0 = IN$

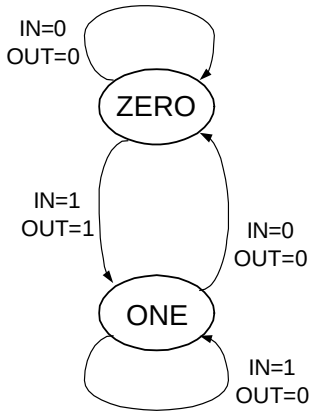
PS

IN \ PS	00	01	11	10
0	0	1	0	-
1	0	1	0	-

$OUT = \overline{PS_1}\ PS_0$

Solution B

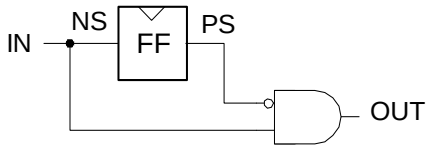
Output depends non only on PS but also on input, IN



Let ZERO=0, ONE=1

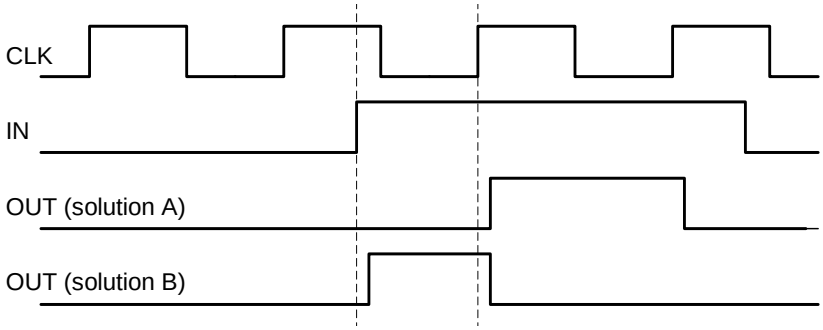
IN	PS	NS	OUT
0	0	0	0
0	1	0	0
1	0	1	1
1	1	1	0

NS = IN, OUT = IN PS'



What's the *intuition* about this solution?

Edge detector timing diagrams



- **Solution A: output follows the clock**
- **Solution B: output changes with input rising edge and is asynchronous wrt the clock.**

FSM Comparison

Solution A

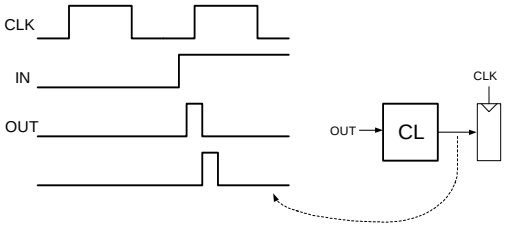
Moore Machine

- **output function only of PS**
- **maybe <u>more</u> state**
- **synchronous outputs**
  - **no glitching**
  - **one cycle "delay"**
  - **full cycle of stable output**

Solution B

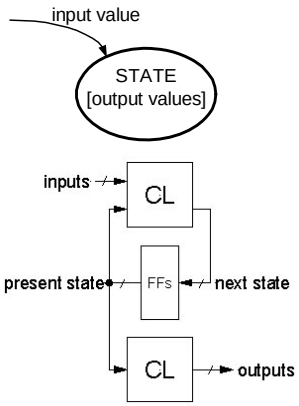
Mealy Machine

- **output function of both PS & input**
- **maybe fewer states**
- **asynchronous outputs**
  - **if input glitches, so does output**
  - **output immediately available**
  - **output may not be stable long enough to be useful:**

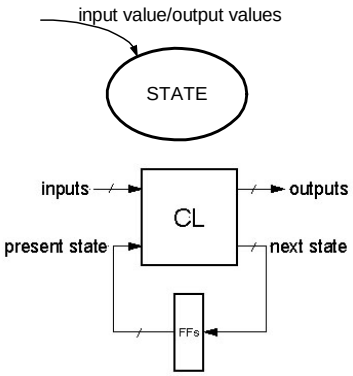


FSM Recap

Moore Machine



Mealy Machine



Both machine types allow one-hot *implementations.*

What next……

° **Sequential Circuits contd…**

Overview

° Important to minimize the size of digital circuitry

° Analysis of state machines leads to a state table (or diagram)

° In many cases reducing the number of states reduces the number of gates and flops

- This is not true 100% of the time

° We attempt state reduction by examining the state table

° Other, more advanced approaches, possible

° Reducing the number of states generally reduces complexity.

FSM Optimization

° **State Reduction:**

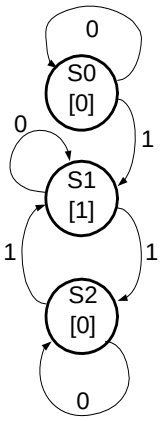
Motivation:

lower cost

- **fewer flip-flops in one-hot implementations**
- **possibly fewer flip-flops in encoded implementations**
- **more don't cares in next state logic**
- **fewer gates in next state logic**

Simpler to design with extra states then reduce later.

° **Example: Odd parity checker**



Moore machine

State Reduction

- **"Row Matching" is based on the state-transition table:**
- **If two states**
  - **have the same output *and* both transition to the same next state**
  - ***or* both transition to each other**
  - ***or* both self-loop**
  - **then they are equivalent.**
- **Combine the equivalent states into a new renamed state.**
- **Repeat until no more states are combined**

State Transition Table

	NS		output
PS	x=0	x=1	
S0	S0	S1	0
S1	S1	S2	1
S2	S2	S1	0

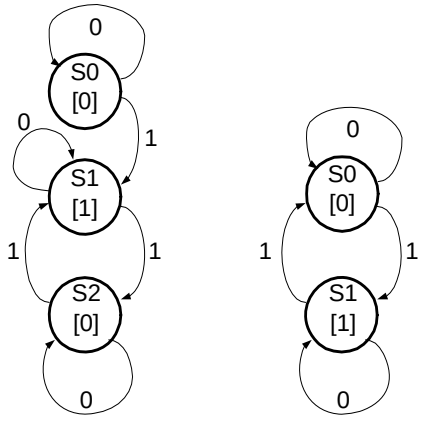
FSM Optimization

- **Merge state S2 into S0**
- **Eliminate S2**
- **New state machine shows same I/O behavior**

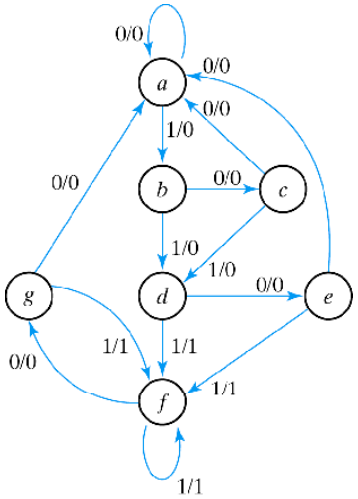
State Transition Table

	NS		output
PS	x=0	x=1	
S0	S0	S1	0
S1	S1	S0	1

- **Example: Odd parity checker.**



Row Matching Example



State Transition Table

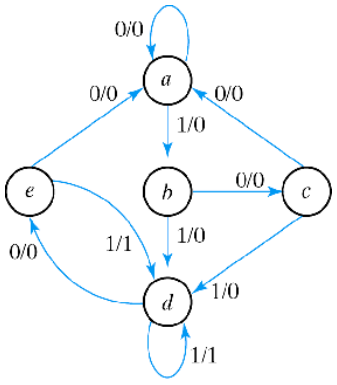
	NS		output	
PS	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

Row Matching Example

	NS		output	
PS	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

	NS		output	
PS	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

Reduced State Transition Diagram



State Reduction

- **The "row matching" method is not guaranteed to result in the optimal solution in all cases, because it only looks at pairs of states.**

- **Another method guarantees the optimal solution:**
- "Implication table" **method:**

  Read Mano, chapter 9.

Submit one page as assignment

Encoding State Variables

- **Option 1: Binary values**
  - **000, 001, 010, 011, 100 …**
- **Option 2: Gray code**
  - **000, 001, 011, 010, 110 …**
- **Option 3: One hot encoding**
  - **One bit for every state**
  - **Only one bit is a one at a given time**
  - **For a 5-state machine**
    - **00001, 00010, 00100, 01000, 10000**

Summary

- Important to create smallest possible FSMs
- This course: use visual inspection method
- Often possible to reduce logic and flip flops
- State encoding is important