

Spiral 1 / Unit 7

State Machine Design

© Mark Redekopp

Outcomes

- I know the difference between combinational and sequential logic and can name examples of each.
- I understand latency, throughput, and at least 1 technique to improve throughput
- I can identify when I need state vs. a purely combinational function
 - I can convert a simple word problem to a logic function (TT or canonical form) or state diagram
- I can use Karnaugh maps to synthesize combinational functions with several outputs
- I understand how a register with an enable functions & is built
- I can design a working state machine given a state diagram
- I can implement small logic functions with complex CMOS gates

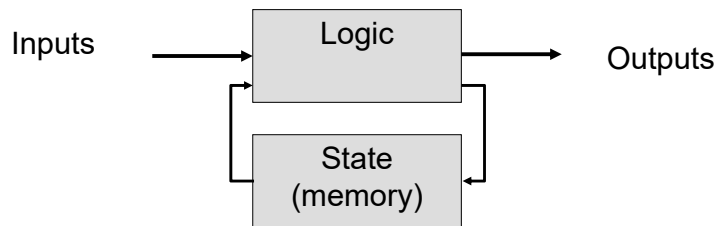
STATE MACHINES OVERVIEW

What is state?

- Decisions are generally influenced by not only what is happening _____, but based on the sum of _____ experiences
 - The sum of all previous experiences is what is known as **state**
- In a human, 'state' refers to the sum of everything that has happened that has led you to where you are now and influences your interpretation of your senses & thoughts
- In a circuit, 'state' refers to all the bits being remembered (_____ or memory)
- In software, 'state' refers to all the _____ values that are being used

State Machine Block Diagram

- A system that utilizes state is often referred to as a state machine (or finite state machine [FSM])
- Most state machines can be embodied in the following form
 - Logic examines what's happening NOW (inputs) & from the PAST (state) to produce outputs and update the state (which will be used in the future to change the decision)
- Inputs will go away or change, so state needs to summarize/capture anything that might be useful for the future

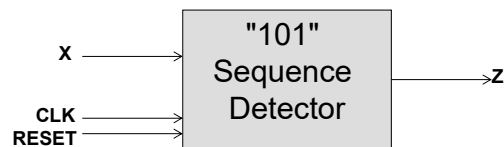


State Machines

- Provide the “brains” or control for electronic and electro-mechanical systems
- Implement a set of steps (or algorithm) to control or solve a problem
- **Goal is to generate output values at _____**
- Combine Sequential and Combinational logic elements
 - Sequential Logic to remember what step (state) we’re in
 - Encodes everything that has happened in the past
 - Combinational Logic to produce outputs and find what state to go to next
 - Generates outputs based on what state we’re in and the input values
- Use _____ (a.k.a. flowcharts) to specify the operation of the corresponding state machine

State Machine Example

- Design a sequence detector to check for the combination "101"
- Input, X, provides 1-bit per clock
- Check the sequence of X for "101" in successive clocks
- If "101" detected, output Z=1 (Z=0 all other times)



Another State Diagram Example

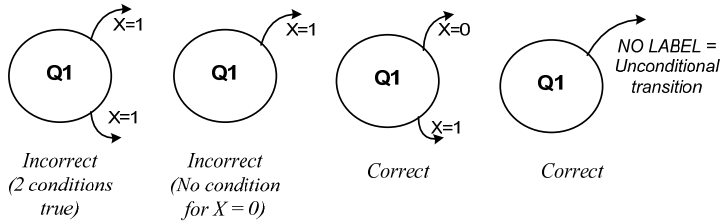
- “101” Sequence Detector should output F=1 when the sequence 101 is found in consecutive order



State Diagram for “101”
Sequence Detector

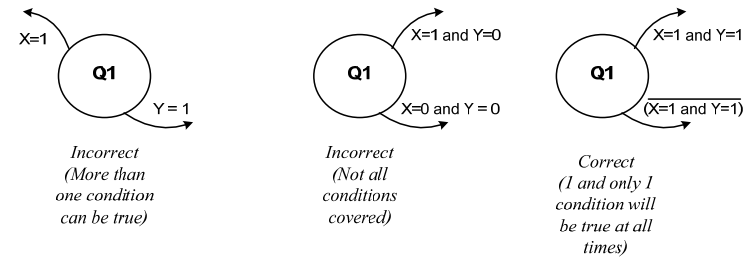
Correct Specification of State Diagrams

- _____ transition from a state may be true at a time
 - Not 2, not 0, exactly 1
 - Make sure the conditions you associate with the arrows coming out of a state are _____ (< 2 true) but _____ (> 0 true)



Correct Specification of State Diagrams 2

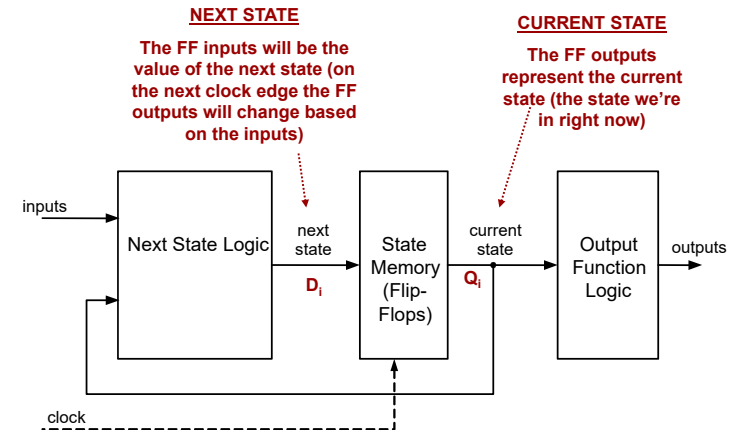
- Exactly one transition from a state may be true at a time



State Machines

- State Machines can be broken into 3 sections of logic
 - State Memory (SM)
 - Just FF's to remember the *current state*
 - Next State Logic (NSL)
 - Combo logic to determine the next state
 - Essentially implements the transition conditions
 - Output Function Logic (OFL)
 - Combo logic to produce the outputs

State Machine



Important: State is always represented and stored by the _____ in the system

State Machine Outputs

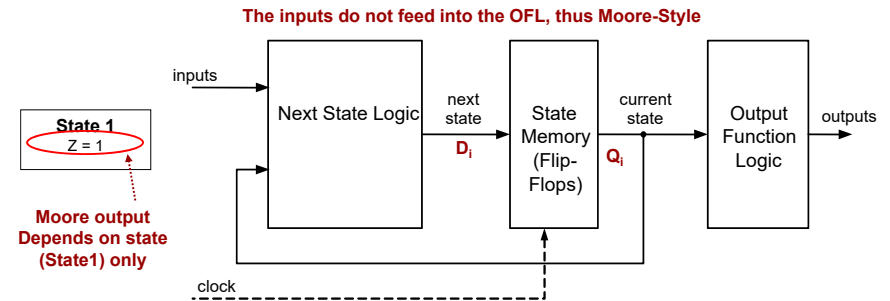
- State Machine outputs can be classified according to how the outputs are produced

– If $Outputs = f(\text{_____})$...
_____ -Style

– If $Outputs = f(\text{_____})$...
_____ -Style

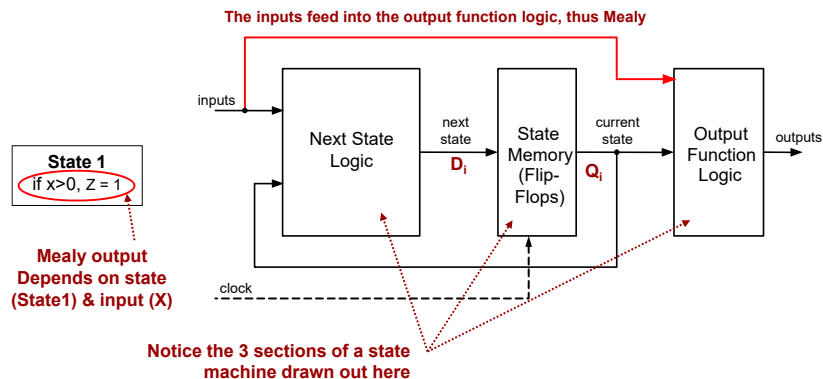
Moore-Style Outputs

- Moore-style outputs only depend on the current state
- Thus, they are valid _____ in the clock cycle and _____ nearly the entire
- Often requires extra states compared to Mealy-style implementations



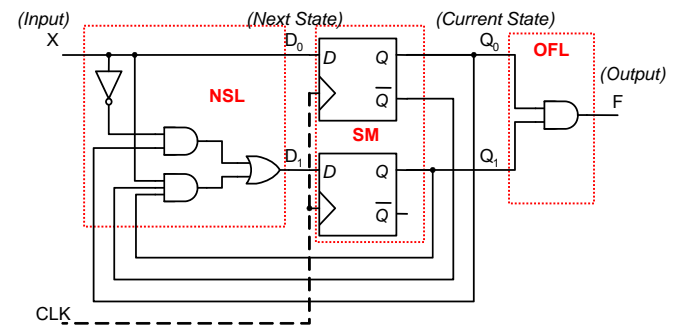
Mealy-Style Outputs

- Mealy-style outputs depend not only on the current state but the external inputs
- Thus, they may not be valid until _____ in the clock cycle and _____ during the cycle if the inputs change



State Machines

- Below is a circuit implementing a state machine, notice how it breaks into the 3 sections



STATE MACHINE DESIGN

State Machine Review

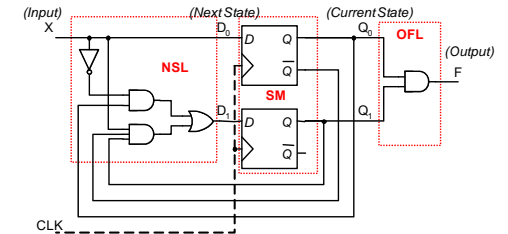
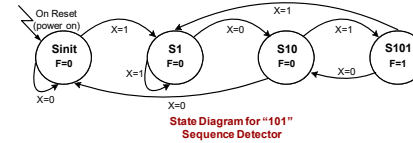
State Diagrams

1. States
2. Transition Conditions
3. Outputs

State Machine

1. State Memory => FF's
 - n-FF's => 2^n states
2. Next State Logic (NSL)
 - combinational logic
 - logic for FF inputs
3. Output Function Logic (OFL)
 - MOORE: $f(\text{state})$
 - MEALY: $f(\text{state} + \text{inputs})$

State Machines require sequential logic to remember the current state (w/ just combo logic we could only look at the current value of X, but now we can take 4 separate actions when X=0)



State Machine Design

- State machine design involves taking a problem description and coming up with a state diagram and then designing a circuit to implement that operation



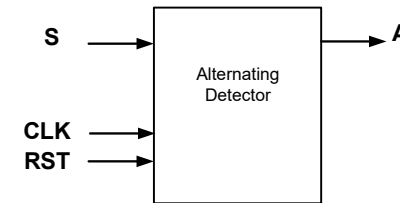
State Machine Design

- Coming up with a state diagram is non-trivial
- Requires creative solutions
- Designing the circuit from the state diagram is done according to a simple set of steps
- To come up w/ a state diagram to solve a problem
 - Write out an algorithm or _____ to solve the problem
 - Each step in your algorithm will usually be _____ in your state diagram
 - Ask yourself what past inputs need to be remembered and that will usually lead to a state representation

EXAMPLE 1

Alternating Detector

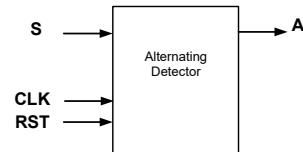
- Given bits coming in from a sensor, design a system that outputs true if sequential bits alternate or false if the same bit value is detected in that past two clock cycles



Alternating Detector Example

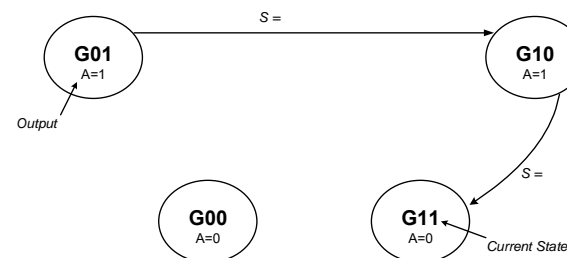
- Can take a Mealy or Moore approach

•Mealy Approach
(usually requires less states):



Alternating Detector

- Design a state machine to check if sensor produces two 0's in a row (i.e. 2 consecutive spaces) or two 1's in a row (i.e. 2 consecutive teeth)



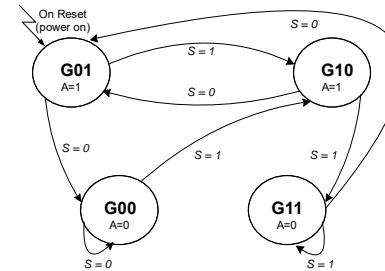
- G10 = Last cycle we got 1, two cycles ago we got 0
- G01 = Last cycle we got 0, two cycles ago we got 1
- G11 = Got 2 consecutive 1's
- G00 = Got 2 consecutive 0's

6 Steps of State Machine Design

1. State Diagram
2. Transition/Output Table
3. State Assignment
 - Determine the # of FF's required
 - Assign binary codes to replace symbolic names
4. Excitation Table (Rename Q* to D)
5. K-Maps for NSL and OFL
 - One K-Map for every FF input
 - One K-Map for every output of OFL
6. Draw out the circuit

Transition Output Table

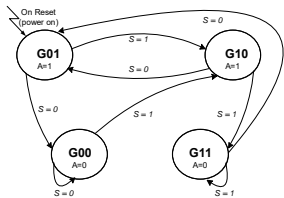
- Convert state diagram to transition/output table
 - Show Next State & Output as a function of Current State and Input



Current State	Input (S)	Next State	Output (A)
G01	0		1
G01	1		1
G11	0		0
G11	1		0
G00	0		0
G00	1		0
G10	0		1
G10	1		1

Transition Output Table

- Now assign binary codes to represent states



State Assignment Mapping

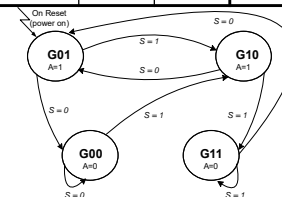
State	Q ₁	Q ₀
G01	0	0
G11	0	1
G00	1	0
G10	1	1

Current State		Input	Next State		Output
Q ₁	Q ₀	S	Q ₁ *	Q ₀ *	A
0	0	0	1	0	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	0	1	1

Transition Output Table

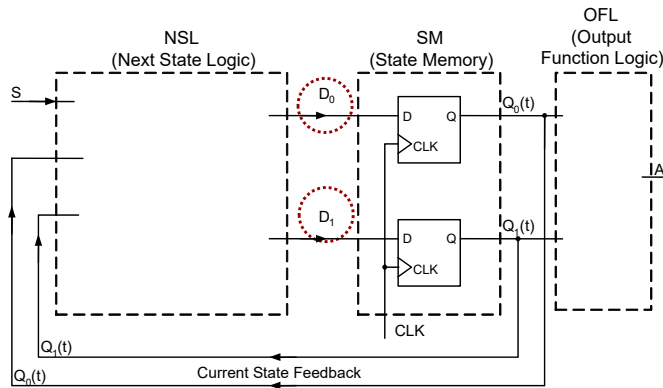
- Convert state diagram to transition/output table

Current State			Next State				Output
			S = 0		S = 1		
State	Q ₁	Q ₀	State		State		A
G01	0	0	G00		G10		1
G11	0	1	G01		G11		0
G10	1	1	G01		G11		1
G00	1	0	G00		G10		0



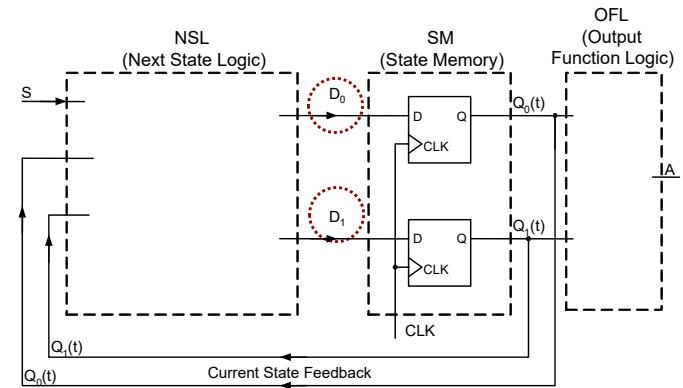
Excitation Table

- The goal is to produce logic for the inputs to the FF's (D_1, D_0)...these are the excitation equations



Excitation Table

- Using your transition table you know what you want Q^* to be, but how can you make that happen?
- For D-FF's Q^* will be whatever D is at the edge



Excitation Table

- In a D-FF Q^* will be whatever D is, so if we know what we want Q^* to be just make sure that's what the D input is

Current State			Next State						Output
			S = 0			S = 1			
State	Q_1	Q_0	State	D_1	D_0	State	D_1	D_0	A
G01	0	0	G00	1	0	G10	1	1	1
G11	0	1	G01	0	0	G11	0	1	0
G10	1	1	G01	0	0	G11	0	1	1
G00	1	0	G00	1	0	G10	1	1	0

Karnaugh Maps

- Now need to perform K-Maps for D_1 , D_0 , and A

Current State			Next State						Output
			S = 0			S = 1			
State	Q_1	Q_0	State	D_1	D_0	State	D_1	D_0	A
G01	0	0	G00	1	0	G10	1	1	1
G11	0	1	G01	0	0	G11	0	1	0
G10	1	1	G01	0	0	G11	0	1	1
G00	1	0	G00	1	0	G10	1	1	0

K-Map for D_1 :

	S	
Q_1Q_0	0	1
00	1	1
01	0	0
11	0	0
10	1	1

K-Map for D_0 :

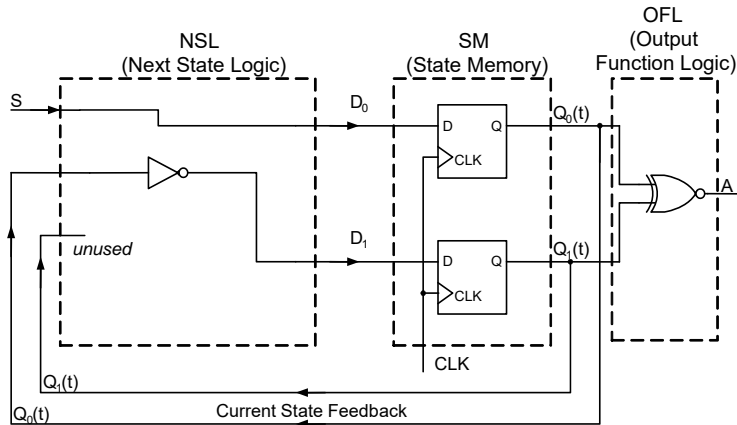
	S	
Q_1Q_0	0	1
00	0	1
01	0	1
11	0	1
10	0	1

K-Map for A:

	Q_1	
Q_0	0	1
0	1	0
1	0	1

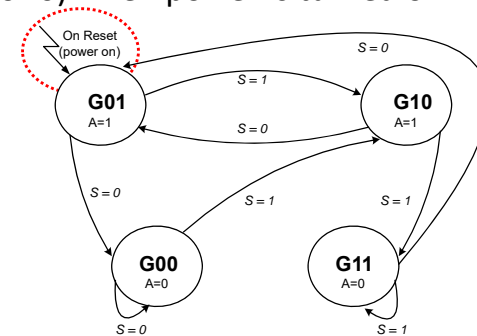
Implementing the Circuit

- Implements the alternating detector



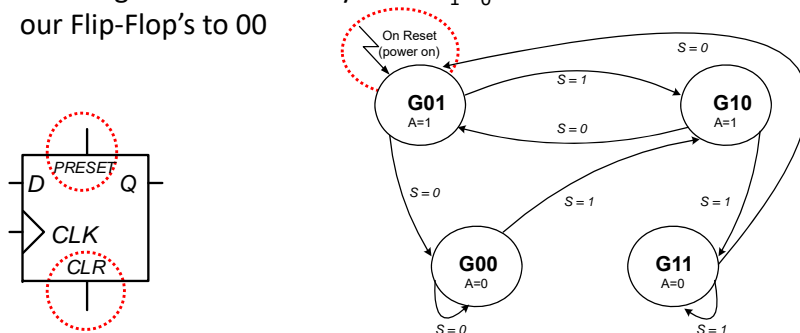
Implementing an Initial State

- How can we make the machine start in G0 on reset (or power on?)
- Flip-flops by themselves will initialize to a random state (1 or 0) when power is turned on



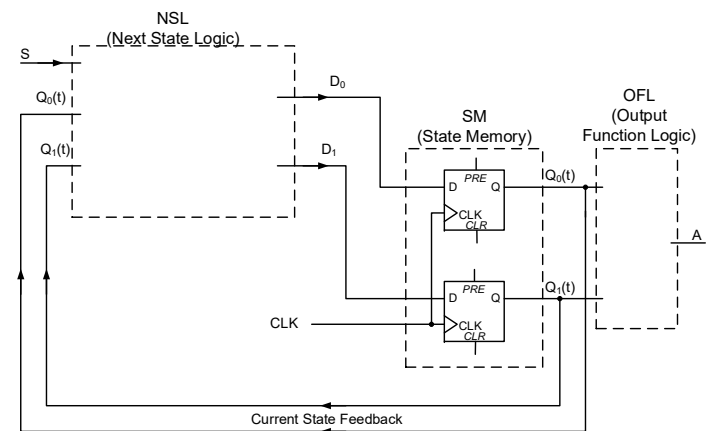
Implementing an Initial State

- Use the CLEAR and PRESET inputs on our flip-flops in the state memory
 - When CLEAR is active the FF initializes $Q=0$
 - When PRESET is active the FF initializes $Q=1$
- We assigned G0 the binary code $Q_1Q_0=00$ so we must initialize our Flip-Flop's to 00



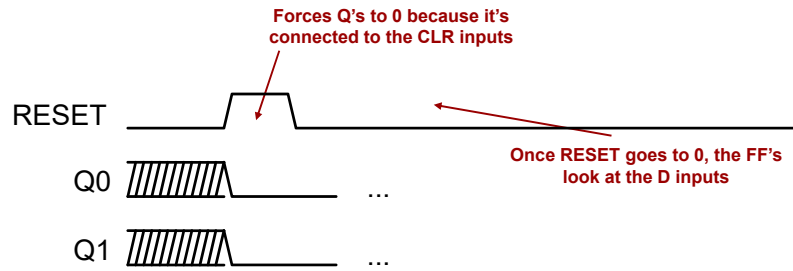
Implementing an Initial State

- Use the CLR inputs of your FF's along with the RESET signal to initialize them to 0's and tie off the PRE inputs



Implementing an Initial State

- When RESET is activated Q's initialize to 0 and then when it goes back to 1 the Q's look at the D inputs



Alternate State Assignment

- Important Fact:** The _____ we assign to our states can have a big impact on the size of the NSL and OFL
- Let us work again with a different set of assignments

State	Q ₁	Q ₀
G01	0	0
G11	0	1
G10	1	1
G00	1	0

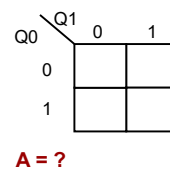
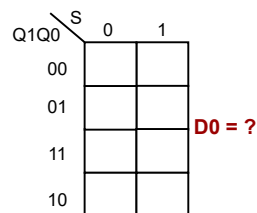
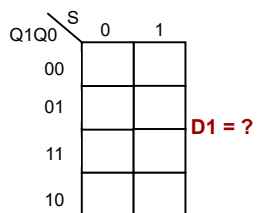
Old Assignments

Current State			Next State				Output
			S = 0		S = 1		
State	Q ₁	Q ₀	State		State		A
G01	0	0	G00		G10		1
G10	0	1	G01		G11		1
G00	1	1	G00		G10		0
G11	1	0	G01		G11		0

New Assignments

Alternate State Assignment

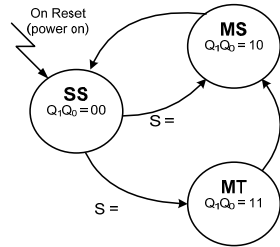
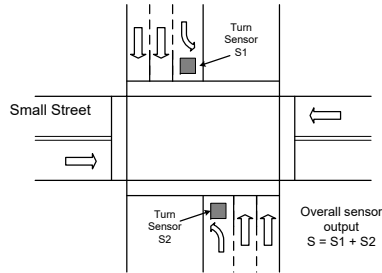
Current State			Next State						Output
			S = 0			S = 1			
State	Q ₁	Q ₀	State	Q ₁ *= D1	Q ₀ *= D0	State	Q ₁ *= =D1	Q ₀ *= =D0	A
G01	0	0	G00	1	1	G10	0	1	1
G10	0	1	G01	0	0	G11	1	0	1
G00	1	1	G00	1	1	G10	0	1	0
G11	1	0	G01	0	0	G11	1	0	0



EXAMPLE 2

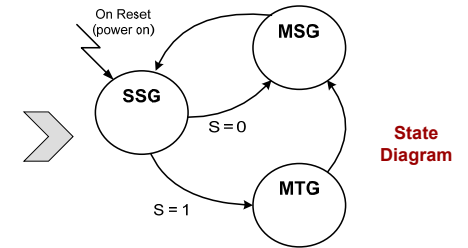
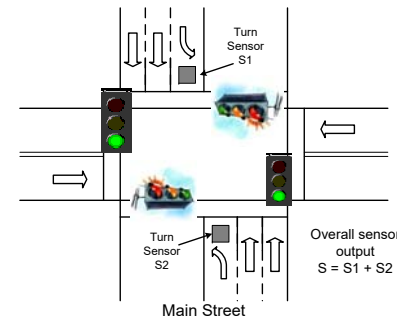
Traffic Light Controller

- Design the controller for a traffic light at an intersection
 - Main street has a protected turn while small street does not
 - Sensors embedded in the street to detect cars waiting to turn
 - Let $S = \text{_____}$ to check if any car is waiting
 - Simplify and only have Green and Red lights (no yellow)



State Assignment

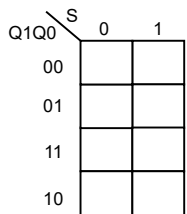
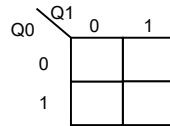
- Design of the traffic light controller with main turn arrow
- Represent states with some binary code
 - Codes: 3 States => 2 bit code: 00=SSG, 10=MSG, 11=MTG



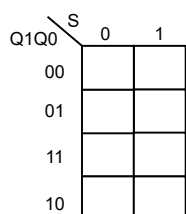
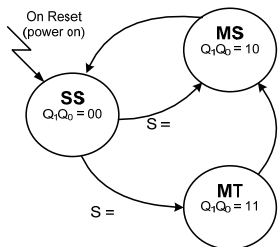
K-Maps

- Find logic for each FF input by using K-Maps

Current State	Next State						Output				
	S = 0			S = 1							
State	Q ₁	Q ₀	State	Q ₁ *	Q ₀ *	State	Q ₁ *	Q ₀ *	SSG	MTG	MSG
SS	0	0									
N/A	0	1									
MT	1	1									
MS	1	0									



$D_1 =$

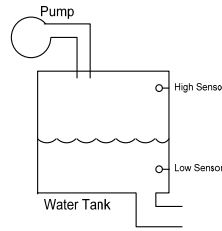
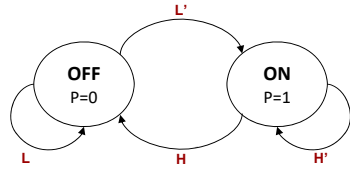


$D_0 =$

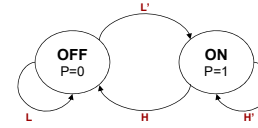
EXAMPLE 3

Water Pump

- Implement the water pump controller using the High and Low sensors as inputs



Transition Table



Current State	Q	Next State							
		H L = 0 0		H L = 0 1		H L = 1 1		H L = 1 0	
Symbol	Q	Sym.	Q*	Sym.	Q*	Sym.	Q*	Sym.	Q*
OFF	0								
ON	1								

Note: The State Value, Q forms the Pump output (i.e. 1 when we want the pump to be on and 0 otherwise)

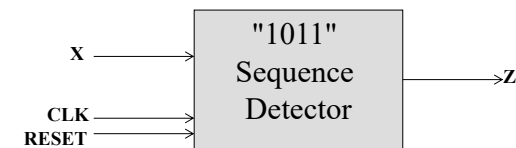
Q	0	1
00		
01		
11		
10		

D =

EXAMPLE 4

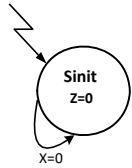
State Machine Example

- Design a sequence detector to check for the combination "1011"
- Input, X, provides 1-bit per clock
- Check the sequence of X for "1011" in successive clocks
- If "1011" detected, output Z=1 (Z=0 all other times)



State Diagram

- Be sure to handle overlapping sequences



Transition Output Table

- Translate the state diagram into the transition output table

Current State	Next State										Output
	X = 0				X = 1				Z		
	State*	Q2*	Q1*	Q0*	State*	Q2*	Q1*	Q0*			
Sinit	0	0	0								0
S10	0	0	1								0
S1	0	1	1								0
S101	0	1	0								0
S1011	1	1	0								1

NSL & OFL

Current State	Next State										Output
	X = 0				X = 1				Z		
	State*	D2	D1	D0	State*	D2	D1	D0			
Sinit	0	0	0		S1	0	1	1	0		0
S10	0	0	1		S101	0	1	0	0		0
S1	0	1	1		S1	0	1	1	0		0
S101	0	1	0		S1011	1	1	0	0		0
S1011	1	1	0		S1	0	1	1	1		1

Q1Q0	Q2	
	0	1
00	0	d
01	0	d
11	0	d
10	0	1

Z = Q2

Q1Q0	Q2			
	00	01	11	10
00	0	d	d	0
01	0	d	d	0
11	0	d	d	0
10	0	0	0	1

$D_2 = X \cdot Q_2' \cdot Q_1 \cdot Q_0'$

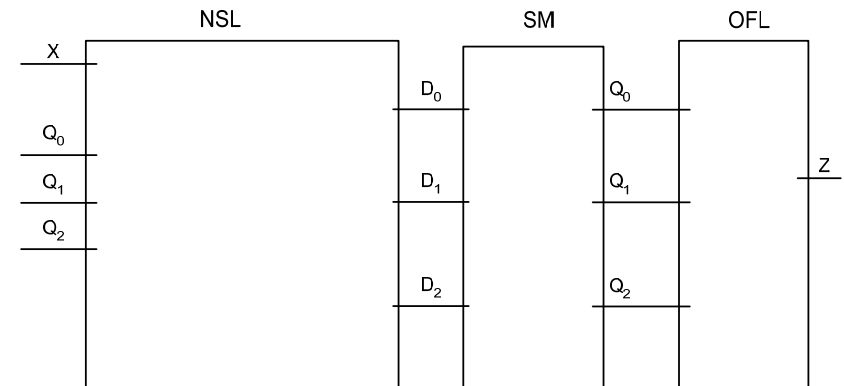
Q1Q0	Q2			
	00	01	11	10
00	0	d	d	1
01	0	d	d	1
11	0	d	d	1
10	0	0	1	1

$D_1 = X$

Q1Q0	Q2			
	00	01	11	10
00	0	d	d	1
01	0	d	d	0
11	1	d	d	1
10	1	1	1	0

$D_0 = Q_2 + Q_1 Q_0 + X' Q_1 + X Q_1' Q_0'$

Drawing the Circuit



Waveform for 1011 Detector

