Introduction to Digital Logic

Lecture 20:
State Machine Design

State Machine Review

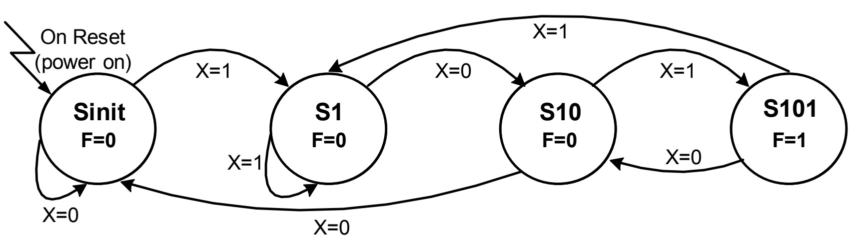
State Diagrams

1. States
2. Transition Conditions
3. Outputs

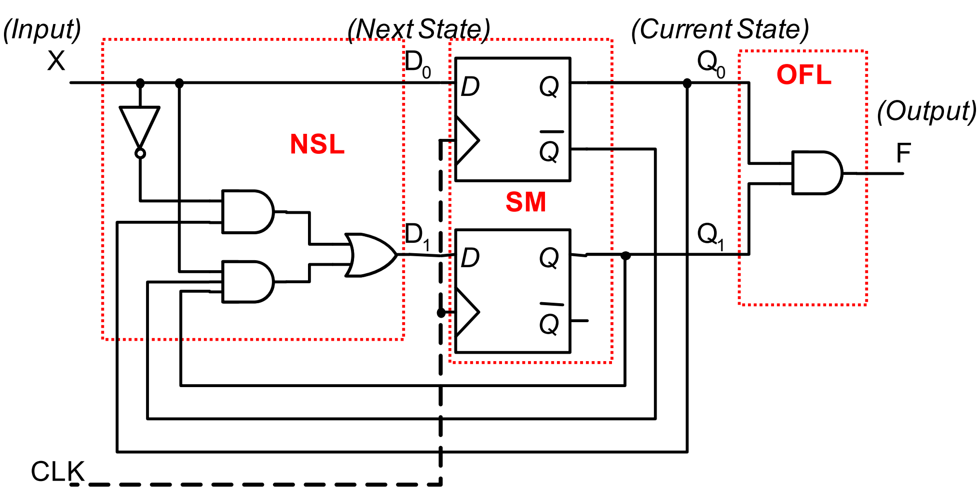
State Machine

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(state)
   – MEALY: f(state + 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 Diagram for "101" Sequence Detector



State Machine Analysis Review

- 6 Steps to analyze
  - Excitation Equations
    - Eqn's for FF inputs
  - Transition Equations ($Q_i^* = ??$)
    - Use characteristic equation of FF and substitute excitation equations for the FF inputs
  - Output Equations
  - Transition/Output Table
    - Make a table showing all combinations of current state and external inputs and then what each of the next state and output values will be for each of those combinations
  - State Name Assignment
    - Symbolic names replace binary codes
  - Draw the State Diagram

D-FF: $Q^* = D$
RS-FF: $Q^* = S + R'Q$
JK-FF: $Q^* = JQ' + K'Q$

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

Problem Description → **State Diagram** → **Circuit Implementation**

USC Viterbi School of Engineering

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

Solving Problems w/ State Diagrams

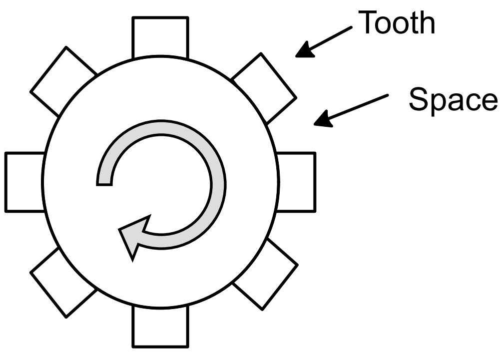
- To come up w/ a state diagram to solve a problem
  - Write out an algorithm or series of steps to solve the problem
  - Each step in your algorithm will usually be one state in your state diagram
  - Ask yourself what past inputs need to be remembered and that will usually lead to a state representation

6 Steps of State Machine Design

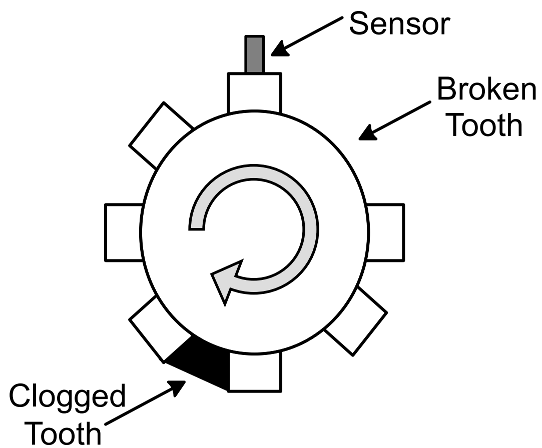
1. State Diagram
2. Transition/Output Table
3. State Assignment
   - Determine the # of FF's required
   - Assign binary codes to replace symbolic names
4. Choose FF type / Excitation Table
5. K-Maps for NSL and OFL
   - One K-Map for every FF input
   - One K-Map for every output of OFL
6. Draw out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

Fly Wheel Example

- Determine the functionality (or “health” of a fly wheel
- Healthy if teeth and spaces alternate
- Unhealthy if tooth breaks off (i.e. 2 consecutive spaces) or if a space gets clogged (i.e. 2 consecutive teeth)
- Sensor, S, outputs 1 when it sees a tooth, 0 when it sees a space



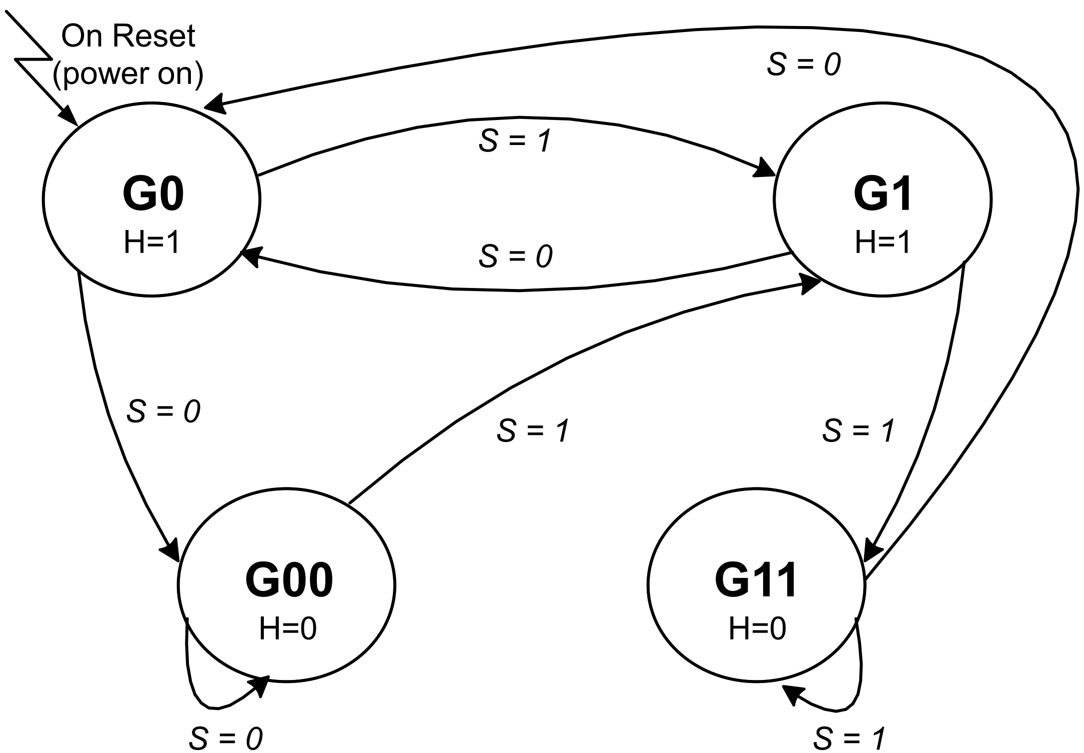
Healthy Fly Wheel



Unhealthy Fly Wheel

Fly Wheel Example

- 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)



- G1 = Got 1 consecutive '1' …healthy
- G0 = Got 1 consecutive '0'…healthy
- G11 = Got 2 "1's" in a row…unhealthy
- G00 = Got 2 "0's" in a row…unhealthy

Transition Output Table

- Convert state diagram to transition/output table

Current State			Next State						Output
			S = 0			S = 1			
State			State			State			H
G0			G00			G1			1
G11			G0			G11			0
G1			G0			G11			1
G00			G00			G1			0

State Assignment

- 4 States => 2 Flip-Flops (Q_1Q_0)
- Make up binary codes for each state

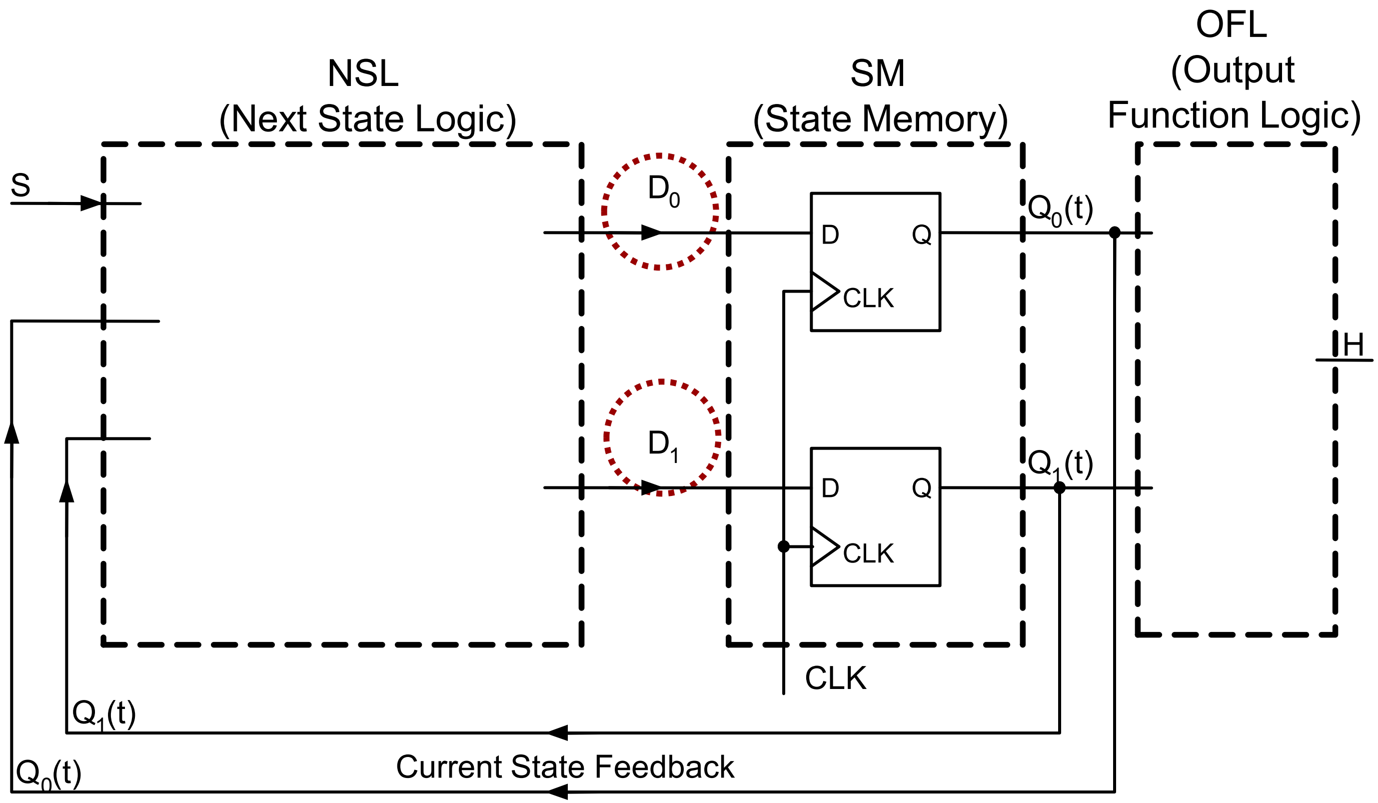
Current State			Next State						Output
			S = 0			S = 1			
State	Q_1	Q_0	State	Q_1^*	Q_0^*	State	Q_1^*	Q_0^*	H
G0	0	0	G00	1	0	G1	1	1	1
G11	0	1	G0	0	0	G11	0	1	0
G1	1	1	G0	0	0	G11	0	1	1
G00	1	0	G00	1	0	G1	1	1	0

Pick Flip-Flop Type

- We know we need 2 FF's
- Pick D-, JK-, or SR-FF's
  - D-FF's are the easiest to do design with but can sometimes yield large NSL
  - JK-FF's are good choices and usually yield small NSL but make finding the NSL more tedious

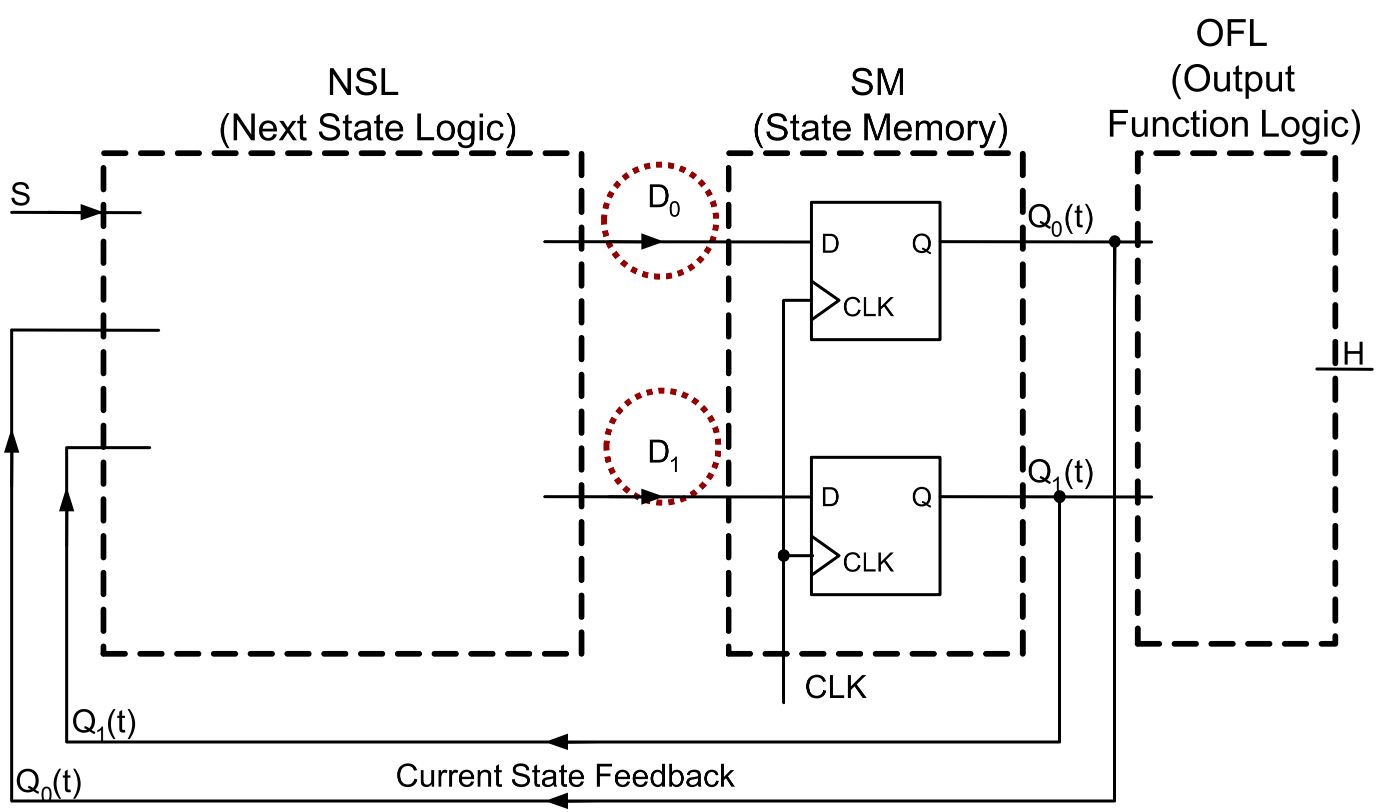
Excitation Table

- Once you've selected your FF type you have 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	H
G0	0	0	G00	1	0	G1	1	1	1
G11	0	1	G0	0	0	G11	0	1	0
G1	1	1	G0	0	0	G11	0	1	1
G00	1	0	G00	1	0	G1	1	1	0

Karnaugh Maps

- Now need to perform K-Maps for D1, D0, and H

Current State			Next State						Output
			S = 0			S = 1			
State	Q_1	Q_0	State	D_1	D_0	State	D_1	D_0	H
G0	0	0	G00	1	0	G1	1	1	1
G11	0	1	G0	0	0	G11	0	1	0
G1	1	1	G0	0	0	G11	0	1	1
G00	1	0	G00	1	0	G1	1	1	0

Q1Q0 \ S	0	1
00	1	1
01	0	0
11	0	0
10	1	1

D1 = Q0’

Karnaugh Maps

- Now need to perform K-Maps for D1, D0, and H

Current State			Next State						Output
			S = 0			S = 1			
State	Q_1	Q_0	State	D_1	D_0	State	D_1	D_0	H
G0	0	0	G00	1	0	G1	1	1	1
G11	0	1	G0	0	0	G11	0	1	0
G1	1	1	G0	0	0	G11	0	1	1
G00	1	0	G00	1	0	G1	1	1	0

Q1Q0 \ S	0	1
00	1	1
01	0	0
11	0	0
10	1	1

D1 = Q0’

Q1Q0 \ S	0	1
00	0	1
01	0	1
11	0	1
10	0	1

D0 = S

Karnaugh Maps

- Now need to perform K-Maps for D1, D0, and H

Current State			Next State						Output
			S = 0			S = 1			
State	Q_1	Q_0	State	D_1	D_0	State	D_1	D_0	H
G0	0	0	G00	1	0	G1	1	1	1
G11	0	1	G0	0	0	G11	0	1	0
G1	1	1	G0	0	0	G11	0	1	1
G00	1	0	G00	1	0	G1	1	1	0

Q1Q0 \ S	0	1
00	1	1
01	0	0
11	0	0
10	1	1

D1 = Q0'

Q1Q0 \ S	0	1
00	0	1
01	0	1
11	0	1
10	0	1

D0 = S

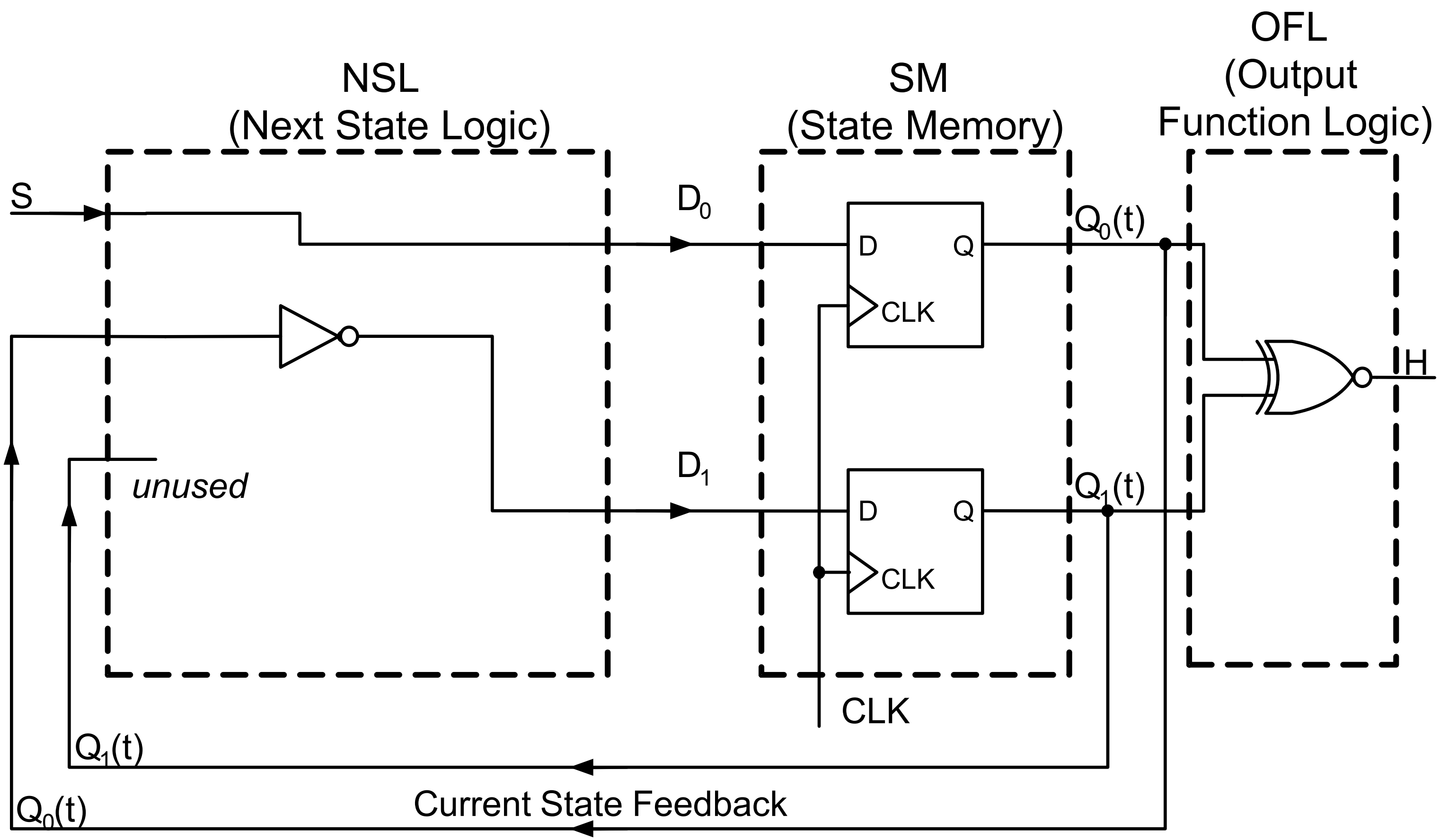
Q0 \ Q1	0	1
0	1	0
1	0	1

H = $Q_1'Q_0'$ + Q1Q0

= Q_1 XNOR Q_0

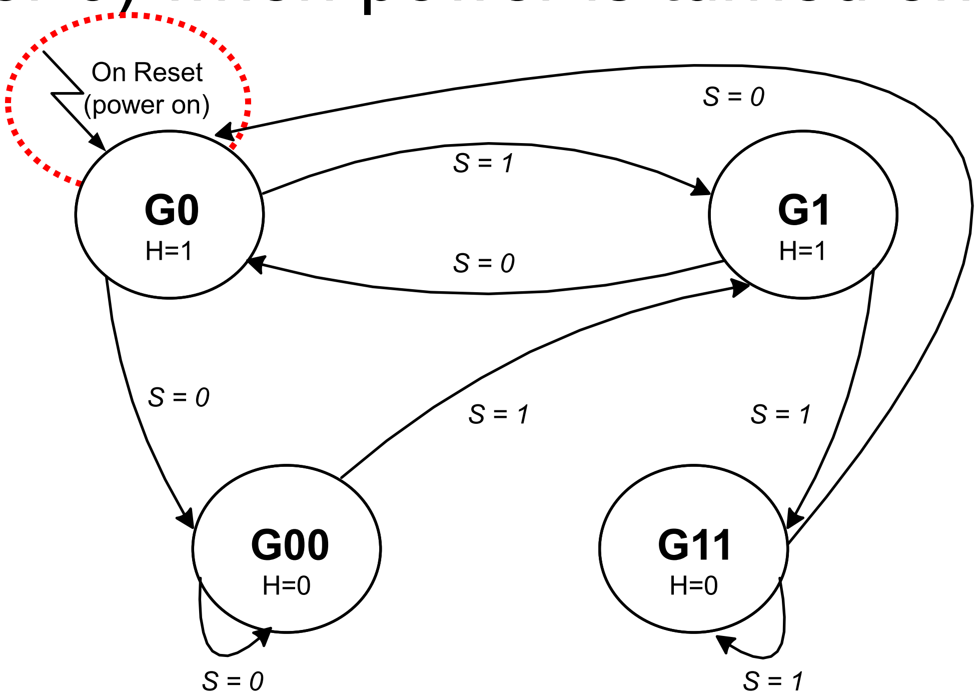
Implementing the Circuit

- Implements the fly wheel "health" monitor



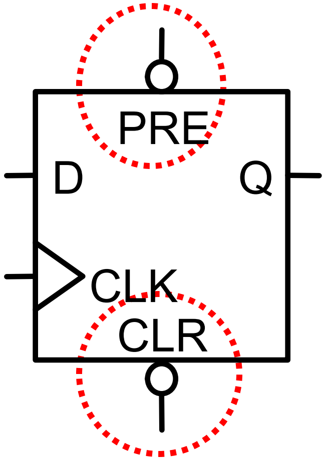
Implementing an Initial State

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



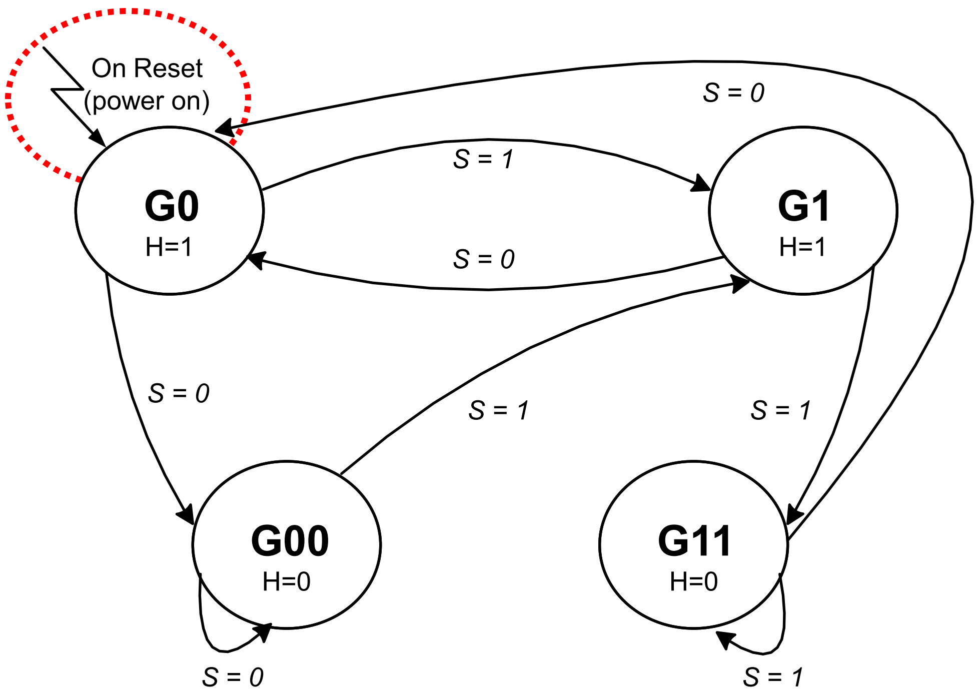
Implementing an Initial State

- 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



Implementing an Initial State

- We assigned G0 the binary code Q_1Q_0=00 so we must initialize our Flip-Flop's to 00

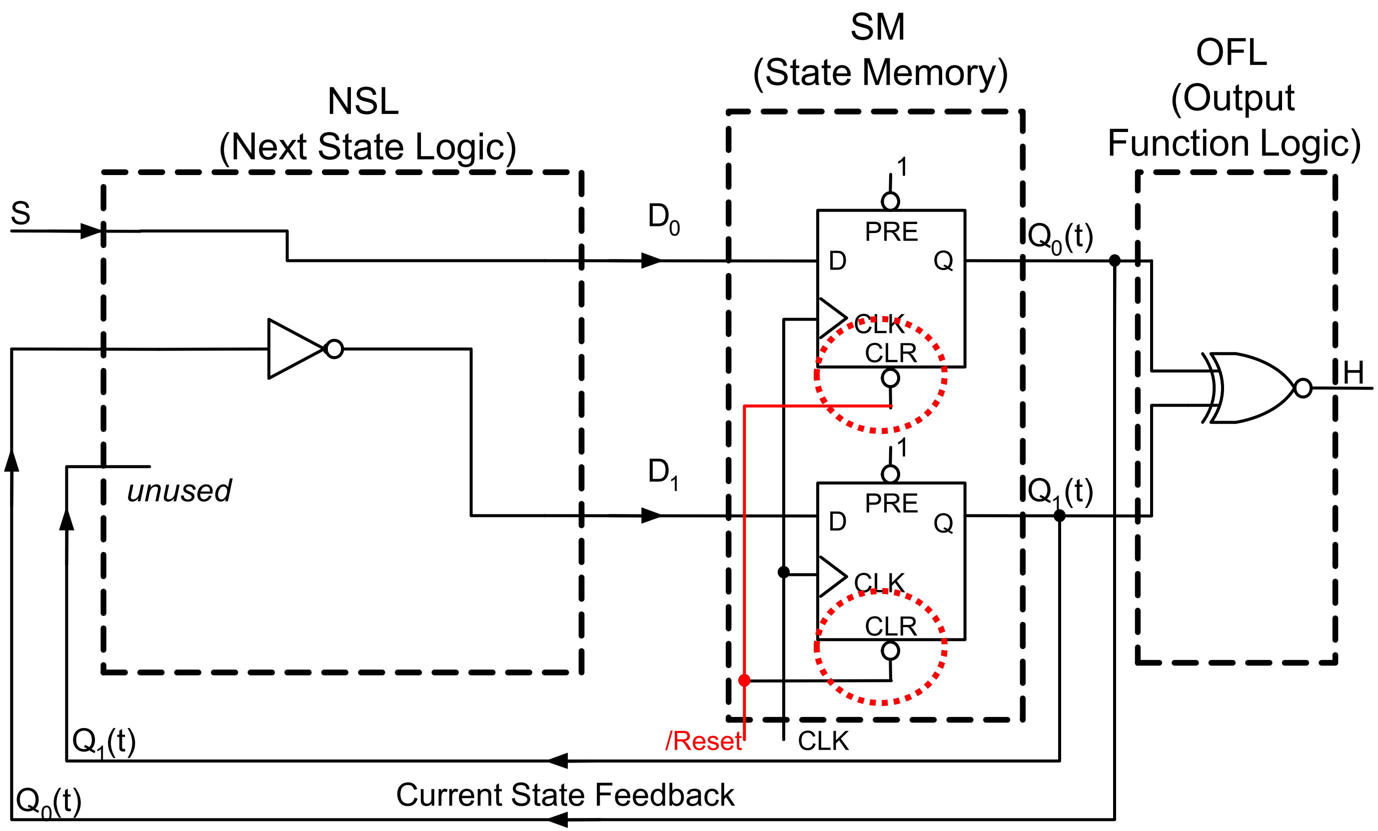


Implementing an Initial State

- To help us initialize our FF's use a RESET signal
- RESET signal (active-low) is produced for us
- It starts at *Active (0)* when power turns on and then goes to *Inactive (1)* for the rest of time
- When it's active use it to initialize the FF's and then it will go inactive for the rest of time and the FF's will work based on their inputs

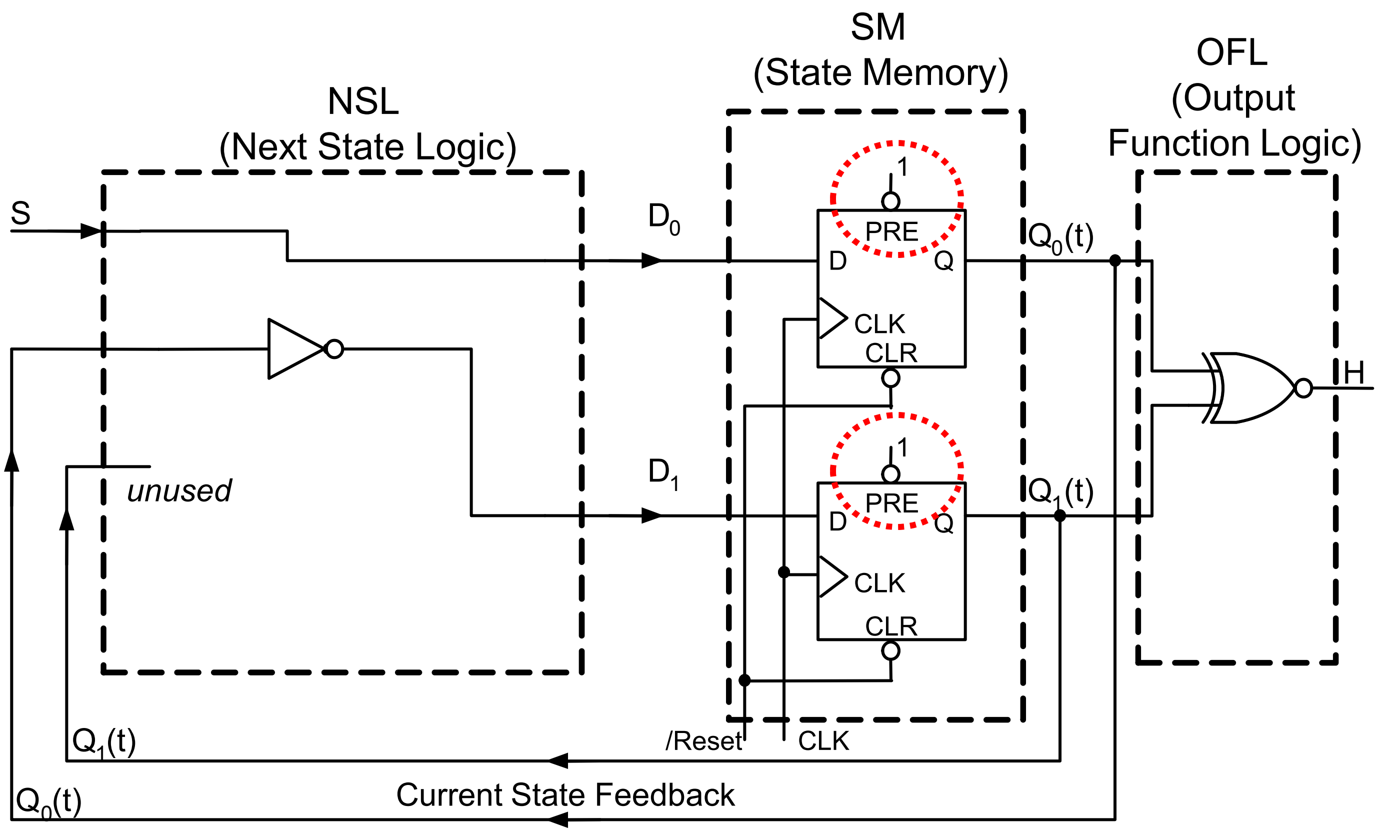
Implementing an Initial State

- Use the /CLR inputs of your FF's along with the /RESET signal to initialize them to 0's



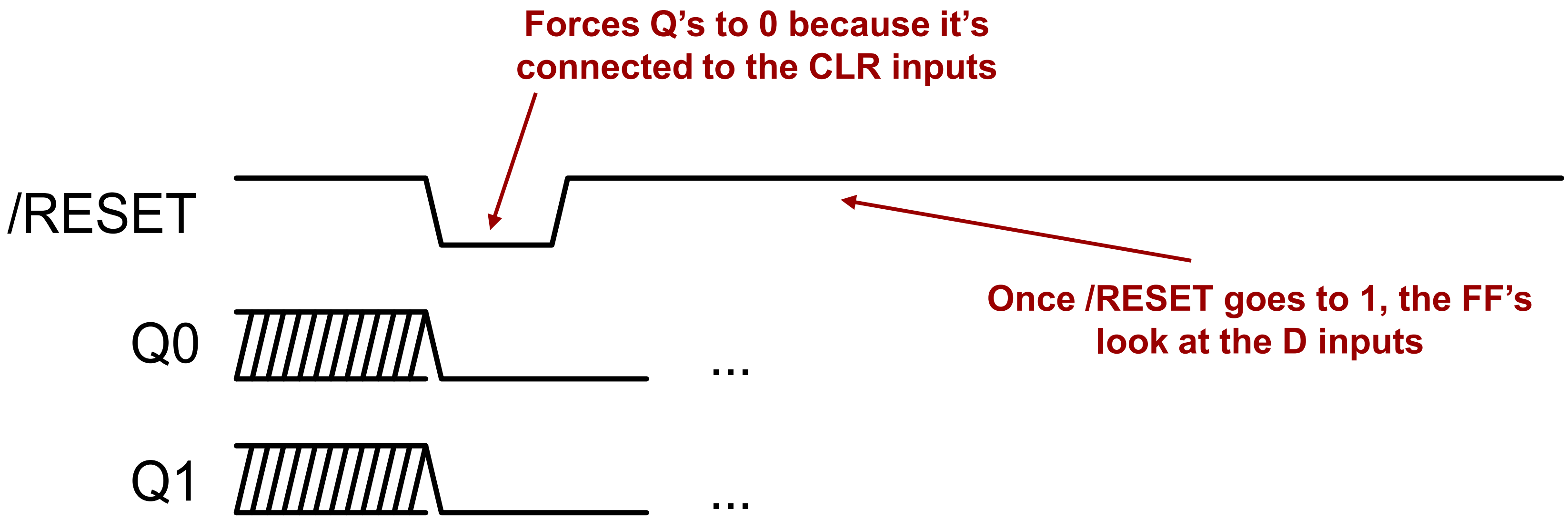
Implementing an Initial State

- We never want to initialize to 1 so tie /PRESET to inactive (1)



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



Implementation with JK FF's

- Go back to step 4
- Come up with a new excitation table for the inputs of the JK FF's

Fly Wheel Monitor using JK FF's

- Find what values of J and K are required to get the necessary transition (Q -> Q*)

Current State			Next State														Output
			S = 0							S = 1							
State	Q_1	Q_0	State	Q_1^*	Q_0^*	J_1	K_1	J_0	K_0	State	Q_1^*	Q_0^*	J_1	K_1	J_0	K_0	H
G0	0	0	G00	1	1	?	?			G1	0	1					1
G1	0	1	G0	0	0					G11	1	0					1
G00	1	1	G00	1	1					G1	0	1					0
G11	1	0	G0	0	0					G11	1	0					0

What value of J_1 and K_1 will cause Q1 to transition from 0 to 1

Fly Wheel Monitor using JK FF's

- Use the Application Table for the JK FF to fill in the values for what J and K should be

Q	Q*	J	K	
0	0	0	d	**J=0,K=0 => remember** **J=0, K=1 => Reset**
0	1	1	d	**J=1,K=0 => Set** **J=1,K=1 => Toggle**
1	0	d	1	**J=0,K=1 => Reset** **J=1,K=1 => Toggle**
1	1	d	0	**J=0,K=0 => remember** **J=1,K=0 => Set**

Fly Wheel Monitor using JK FF's

- Use the application table to find the values of J and K necessary for the desired transitions (Q -> Q*)

Current State			Next State														Output
			S = 0							S = 1							
State	Q_1	Q_0	State	Q_1^*	Q_0^*	J_1	K_1	J_0	K_0	State	Q_1^*	Q_0^*	J_1	K_1	J_0	K_0	H
G0	0	0	G00	1	1	1	d			G1	0	1					1
G1	0	1	G0	0	0					G11	1	0					1
G00	1	1	G00	1	1					G1	0	1					0
G11	1	0	G0	0	0					G11	1	0					0

To make this transition happen
$J_1 = 1, K_1 = d$

Fly Wheel Monitor using JK FF's

- Use the application table to find the values of J and K necessary for the desired transitions (Q -> Q*)

Current State			Next State														Output
			S = 0							S = 1							
State	Q_1	Q_0	State	Q_1*	Q_0*	J_1	K_1	J_0	K_0	State	Q_1*	Q_0*	J_1	K_1	J_0	K_0	H
G0	0	0	G00	1	1	1	d	1	d	G1	0	1					1
G1	0	1	G0	0	0					G11	1	0					1
G00	1	1	G00	1	1					G1	0	1					0
G11	1	0	G0	0	0					G11	1	0					0

To make this transition happen
$J_0 = 1, K_0 = d$

Fly Wheel Monitor using JK FF's

- Use the application table to find the values of J and K necessary for the desired transitions (Q -> Q*)

Current State			Next State															Output
			S = 0								S = 1							
State	Q_1	Q_0	State	Q_1*	Q_0*	J_1	K_1	J_0	K_0	State	Q_1*	Q_0*	J_1	K_1	J_0	K_0		H
G0	0	0	G00	1	1	1	d	1	d	G1	0	1						1
G1	0	1	G0	0	0	0	d			G11	1	0						1
G00	1	1	G00	1	1					G1	0	1						0
G11	1	0	G0	0	0					G11	1	0						0

To make this transition happen
$J_1 = 0$, $K_1 = d$

Fly Wheel Monitor using JK FF's

- Fill in the rest of the table

Current State			Next State														Output
			S = 0							S = 1							
State	Q_1	Q_0	State	Q_1^*	Q_0^*	J_1	K_1	J_0	K_0	State	Q_1^*	Q_0^*	J_1	K_1	J_0	K_0	H
G0	0	0	G00	1	1	1	d	1	d	G1	0	1	0	d	1	d	1
G1	0	1	G0	0	0	0	d	d	1	G11	1	0	1	d	d	1	1
G00	1	1	G00	1	1	d	0	d	0	G1	0	1	d	1	d	0	0
G11	1	0	G0	0	0	d	1	0	d	G11	1	0	d	0	0	d	0

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^*	J_1	K_1	J_0	K_0	State	Q_1^*	Q_0^*	J_1	K_1	J_0	K_0	H
G0	0	0	G00	1	1	1	d	1	d	G1	0	1	0	d	1	d	1
G1	0	1	G0	0	0	0	d	d	1	G11	1	0	1	d	d	1	1
G00	1	1	G00	1	1	d	0	d	0	G1	0	1	d	1	d	0	0
G11	1	0	G0	0	0	d	1	0	d	G11	1	0	d	0	0	d	0

Q1Q0 \ S	0	1
00	1	0
01	0	1
11	d	d
10	d	d

$J_1 = S' \bullet Q_0' + S + Q_0$

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^*	J_1	K_1	J_0	K_0	State	Q_1^*	Q_0^*	J_1	K_1	J_0	K_0	H
G0	0	0	G00	1	1	1	d	1	d	G1	0	1	0	d	1	d	1
G1	0	1	G0	0	0	0	d	d	1	G11	1	0	1	d	d	1	1
G00	1	1	G00	1	1	d	0	d	0	G1	0	1	d	1	d	0	0
G11	1	0	G0	0	0	d	1	0	d	G11	1	0	d	0	0	d	0

Q1Q0 \ S	0	1
00	1	0
01	0	1
11	d	d
10	d	d

$J_1 = S'\bullet Q_0'+S+Q_0$

Q1Q0 \ S	0	1
00	d	d
01	d	d
11	0	1
10	1	0

$K_1 = S'\bullet Q_0'+S+Q_0$

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^*	J_1	K_1	J_0	K_0	State	Q_1^*	Q_0^*	J_1	K_1	J_0	K_0	H
G0	0	0	G00	1	1	1	d	1	d	G1	0	1	0	d	1	d	1
G1	0	1	G0	0	0	0	d	d	1	G11	1	0	1	d	d	1	1
G00	1	1	G00	1	1	d	0	d	0	G1	0	1	d	1	d	0	0
G11	1	0	G0	0	0	d	1	0	d	G11	1	0	d	0	0	d	0

Q1Q0 \ S	0	1
00	1	0
01	0	1
11	d	d
10	d	d

$J_1 = S' \bullet Q_0' + S \bullet Q_0$

Q1Q0 \ S	0	1
00	d	d
01	d	d
11	0	1
10	1	0

$K_1 = S' \bullet Q_0' + S \bullet Q_0$

Q1Q0 \ S	0	1
00	1	1
01	d	d
11	d	d
10	0	0

$J_0 = Q_1'$

Q1Q0 \ S	0	1
00	d	d
01	1	1
11	0	0
10	d	d

$K_0 = Q_1'$

Fly Wheel JK-FF Implementation