Unit 14

State Machine Design

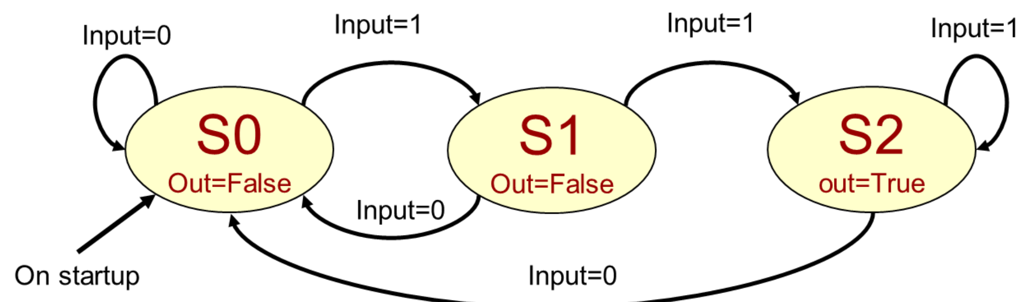
Outcomes

- I can create a state diagram to solve a sequential problem
- I can implement a working state machine given a state diagram

STATE MACHINES OVERVIEW

Review of State Machines

- We've implemented state machines in software, now let's see how we can build them in hardware
- State machines are described with state diagrams that show various states, transition arrows between them, and outputs to be generated based on the current state
  - We use the state to help us know which step of an algorithm we are currently at

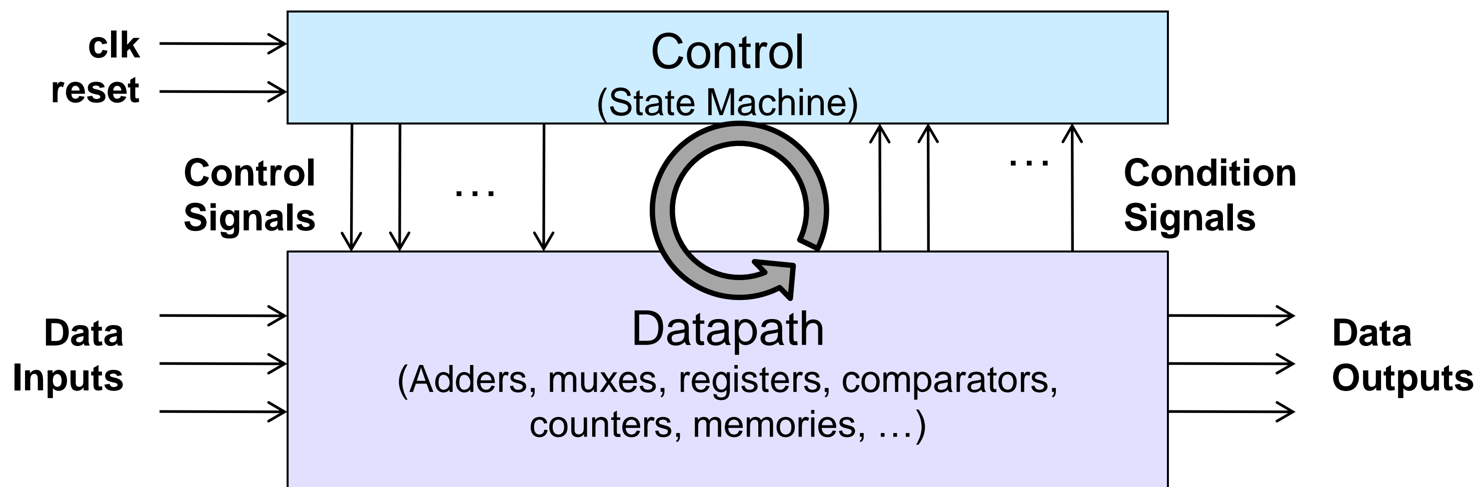


Hardware State Machines

- Hardware (finite) state machines (aka FSMs) provide the “brains” or control for electronic and electro-mechanical systems
  - Many custom hardware designs use a hardware-based FSM to control their operation
- **FSMs are required to generate output values at specific times (i.e. when you need time-dependent hardware outputs)**
  - **Example 1: Traffic light. The system must automatically transition from green to yellow to red without any external input stimulus**
  - **Example 2: Sequence detection. Open a lock only if a certain code is entered over time (e.g. number lock).**
- FSMs require ____________ and __________________ 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

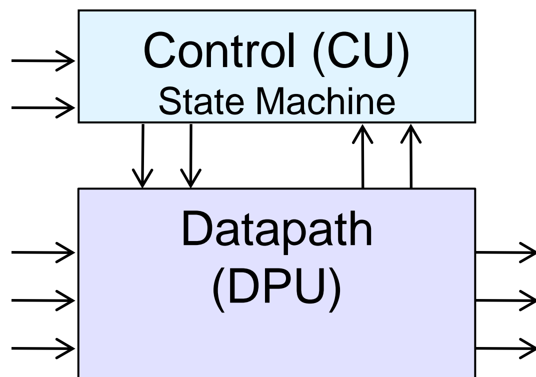
Digital System Design

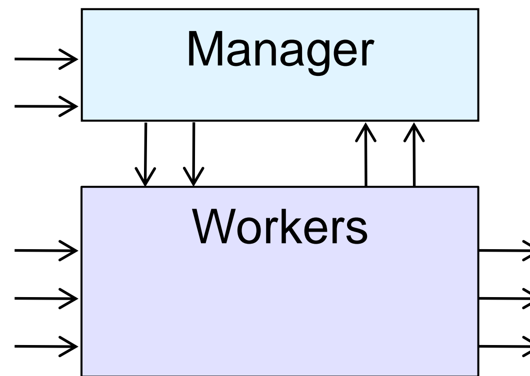
- Most non-trivial digital circuits can be decomposed into two parts: Control **(CU)** and Datapath Unit **(DPU)** paradigm
  - Separate logic into datapath elements that operate on data and control elements that generate control signals for datapath elements
  - Datapath: ________, ___________, comparators, __________, registers (w/ enables), memories, FIFO's
  - Control Unit: **State machines**



Datapath + Control

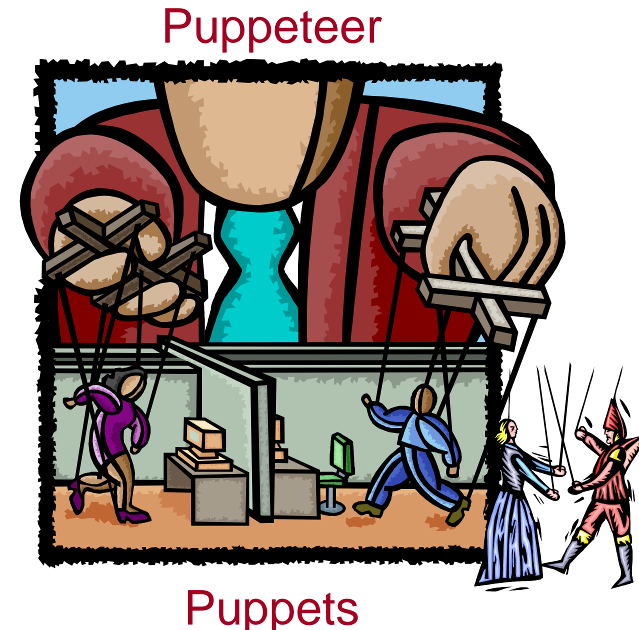
- The control unit acts as scheduler and "__________" while the datapath "does" the work
  - Control signals include: mux selects, load enables, count enables, output enables, add/subtract, etc.
  - Control unit (state machine) turns on the correct control signals at the appropriate times





Construction Company

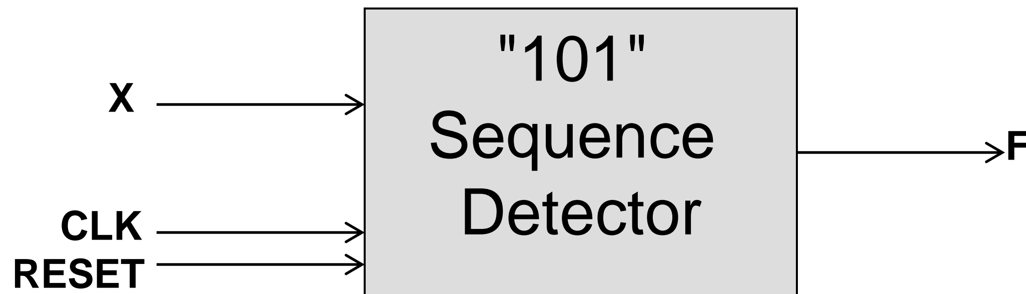
Puppeteer



Puppets

Warm Up Example

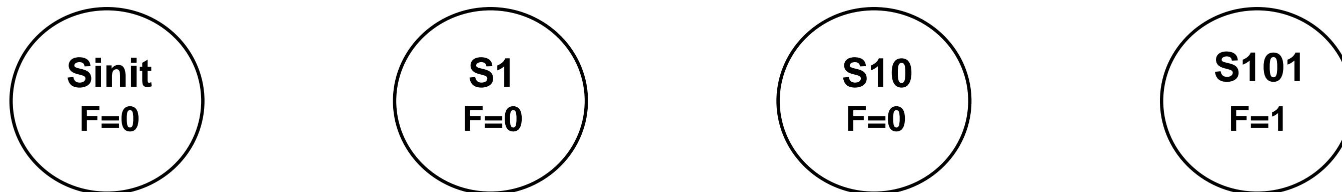
- Design the state diagram for a state machine to detect the single-bit input sequence "101"
- Input, X, provides 1-bit per clock
- Check the sequence of X for "101" in successive clocks
- If "101" detected, output F=1 (F=0 all other times)



Warm Up Example State Diagram

- "101" Sequence Detector should output F=1 when the sequence 101 is found in consecutive order
  - Add transitions for each possible input value of X

State Diagram for "101" Sequence Detector



But how can we implement this in a circuit ??

(i.e. HW state machine implementation)

See the end of this slide set for more detailed solutions and explanations.

Hardware vs. Software FSM Comparison

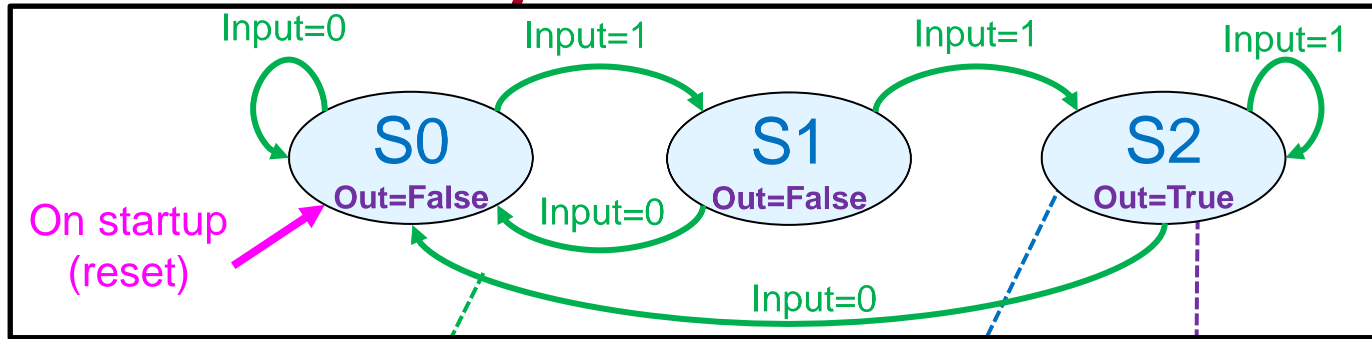
Hardware FSMs

- Can change state (make a transition) **every ___________.**
- Uses **flip-flops (i.e. a register)** to store the ______________
- Designer can choose state 'codes' **arbitrarily,** but the choice can greatly affect the ______ of the circuit or ease of implementation
- Uses **logic gates** (found from a truth table/K-map or other means) to implement the **state transition arrows** (aka **Next State Logic – NSL**)
- Must implement the **initial state** value using the ________ signal

Software FSMs

- Can change state (make a transition) when software **polls** the inputs (which could be very low frequency)
- Uses a **variable** to store the current state
- Programmer can choose state 'codes' **arbitrarily** with little implication
- Uses **'if' statements** to implement the **state transition arrows**
- Must implement the **initial value** of the state variable

HW Anatomy of a State Machine



NEXT STATE

On the next clock edge the FF outputs will change based on these inputs. Thus, the logic feeding the D inputs determines the next state and corresponds to the transition arrows in the diagram

CURRENT STATE

The binary code given by the FF outputs indicates the current state (the state we're in right now)

OUTPUTS

The outputs are generated based on the current state (i.e. the State Memory / FF outputs)

inputs

Logic

next state

D_i

(Flip-Flops)

current state

Q_i

Output Function Logic

outputs

CLOCK

The clock causes the current state (flip-flops) to update once per cycle (on the edge)

clock

reset

INITIAL STATE

The reset signal initializes the flip-flops (current state) to a known/desired value when power is turned on

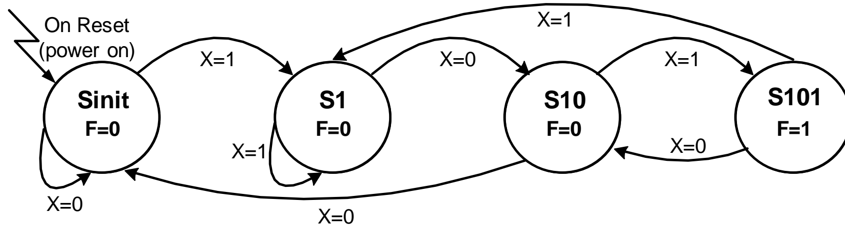
Important: State is always represented and stored by the flip-flop outputs in the system

State Diagram vs. State Machine

State Diagrams

1. States
2. Transition Conditions
3. Outputs

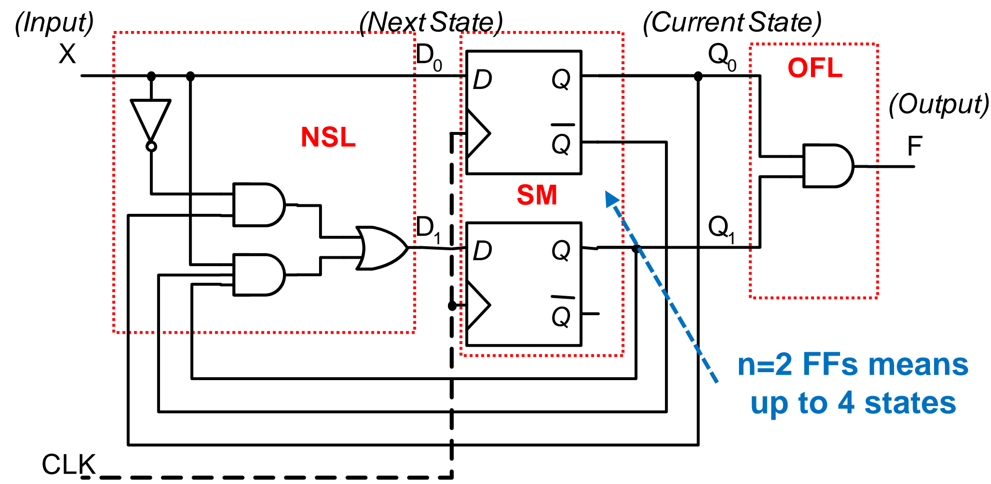
State Machines require sequential logic to remember the current state (w/ just combo logic we could only look at the current value of X, but now we can take 4 separate actions when X=0)



State Diagram for "101" Sequence Detector

State Machine

1. State Memory => Flip-Flops (FFs)
   - Each state assigned a binary code
   - n-FF's => up to ____ states
2. Next State Logic (NSL)
   - combinational logic
   - logic for D-inputs of flip-flops
3. Output Function Logic (OFL)
   - combinational logic



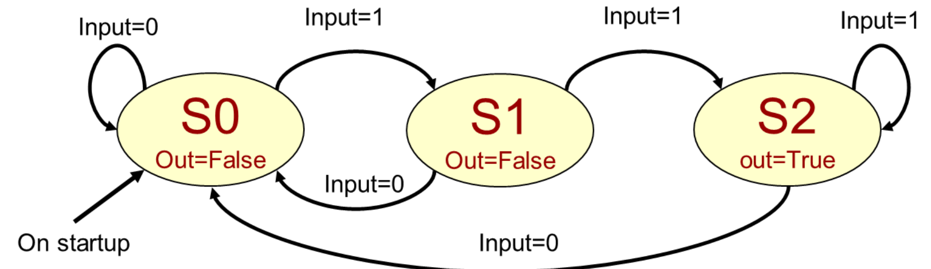
USC Viterbi School of Engineering

Comparison: FSM in SW and HW

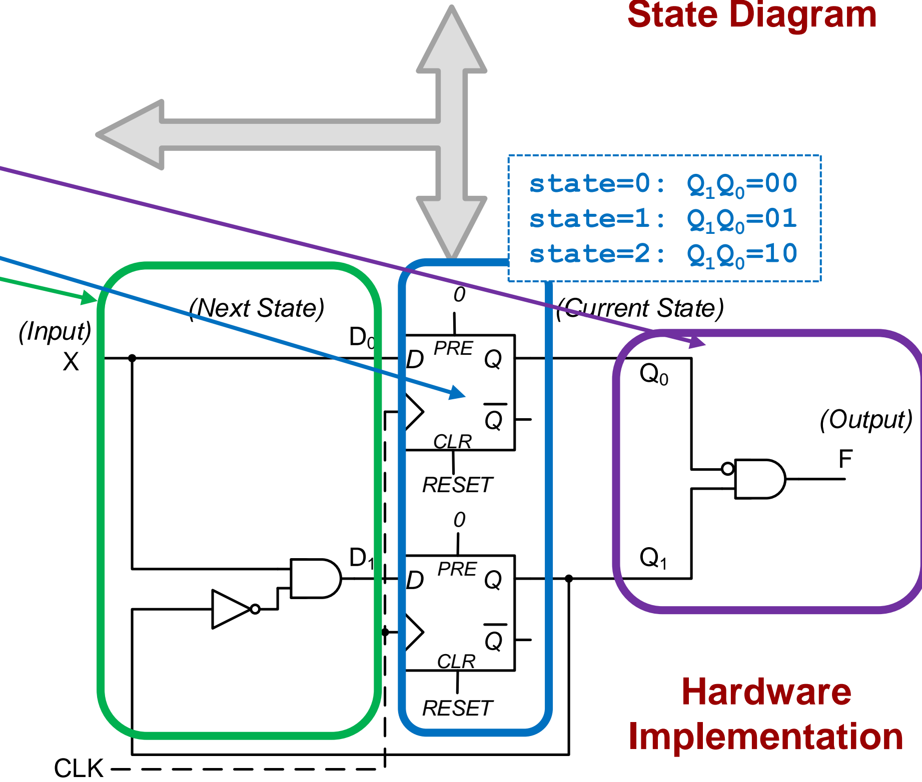
```
int main()
{
  unsigned char state=0;  // init state
  unsigned char input, output;
  while(1)
  {
    _delay_ms(10); // choose appropriate delay
    input = PIND & (1 << PD0);

    if(state == 0){
      PORTD &= ~(1 << PD7); // output off
      if( input ){
         state = 1; /* transition */
      }
      else {
         state = 2; /* transition */
      }
    }
    else if(state == 1){
      PORTD &= ~(1 << PD7); // output on
      if( input ){ state = 2; }
      else { state = 0; }
    }
    else if(state == 2) {
      PORTD |= (1 << PD7);  // output on
      if( !input ) { state = 0; }
    }
  }
  return 0;
}
```

Software Implementation



State Diagram



Hardware Implementation

A General Approach (For any possible state coding)

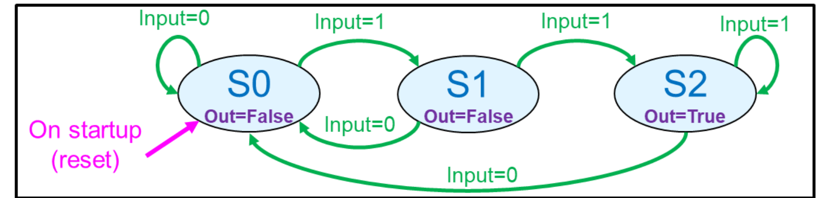
STATE MACHINE IMPLEMENTATION

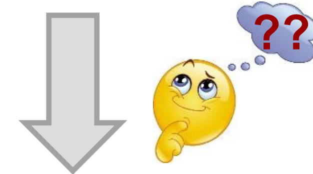
State Machine Design

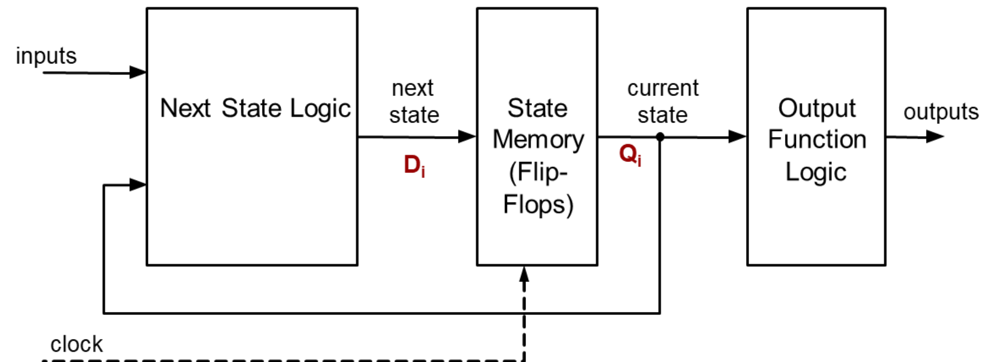
- State machine design involves taking a problem description and coming up with a state diagram and then designing a circuit to implement that operation

HW FSM Implementation (1)

- How do we find a circuit to implement the given state diagram?
- Easy!
  Fill in the 3 parts: SM, NSL, OFL
- Okay… but how do you do that?
- **Step 1**: Start with the __________ (which is just flip-flops) and **determine ____________ flip-flops you need.**

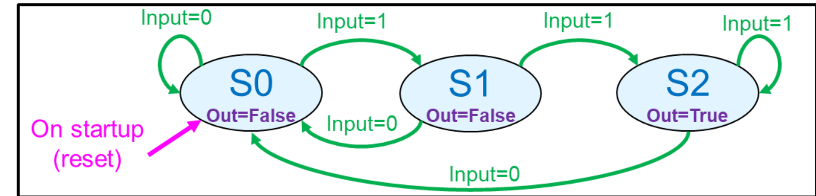






This Photo by Unknown Author is licensed under CC BY-SA-NC

HW FSM Implementation (1)



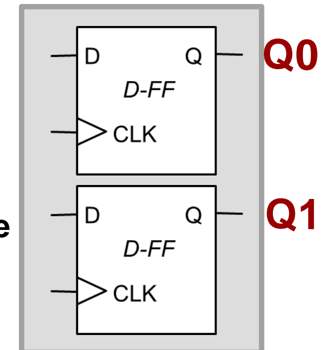
- **Step 1a**: **Assign binary codes to each state.**
- Given **n** states…
  - Codes can use the **minimum** number $\lceil \log_2(n) \rceil$ of bits: { S0=00, S1=01, S2=10 }
  - Codes could use more bits such as "_______" codes (1 bit per state): { S0 = 100, S1 = 010, S2 = 001 }
  - Different codes can work but lead to different size implementations
  - We'll often choose "1-hot" codes as we progress
- **Step 1b**: Determine number of FFs **k bit codes => ___ flip-flops**
  - 2 FFs if a minimal coding is used
  - 3 FFs if a 1-hot coding is used

Minimal Coding

State	Code
S0	
S1	
S2	

2-bit state code => 2 FFs

State Memory

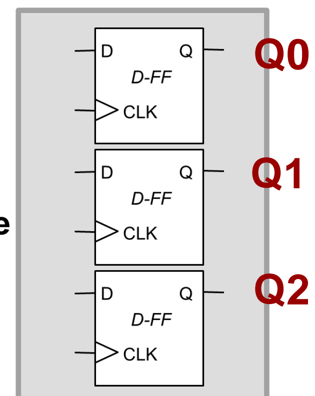


1-Hot Coding

State	Code
S0	100
S1	010
S2	001

3-bit state code => 3 FFs

State Memory



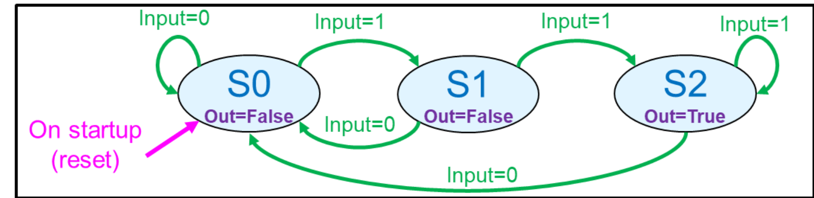
HW FSM Implementation (2)

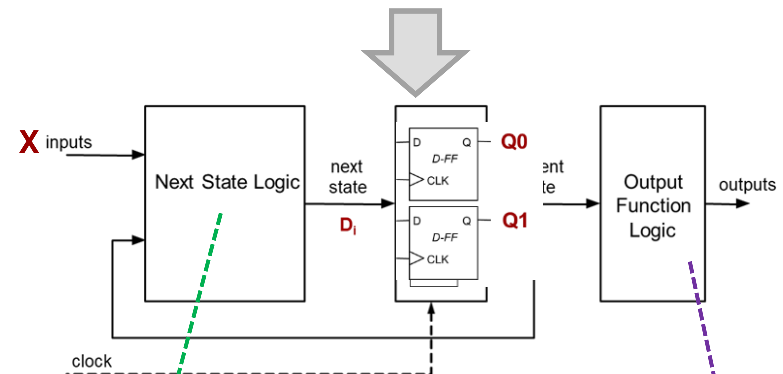
- Now that we have our state memory flip-flops, we can solve for the other two parts: Next State Logic (NSL) and Output Function Logic (OFL)
- **Step 2a**: Make a (next) state transition table that shows the next state for each state, input combination:

  { State x Inputs } -> Next State

- **Step 2b**: Make a state output table that show the output for each state

  { State } -> Outputs





	Input (X)	
State (Q1Q0)	X=0 (Q_1*Q_0*)	X=1 (Q_1*Q_0*)
S0		
S1		
S2		

(Next) State Transition Table

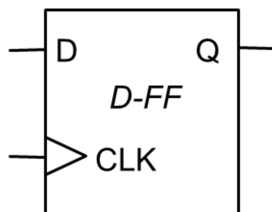
State	Outputs
S0	0
S1	0
S2	1

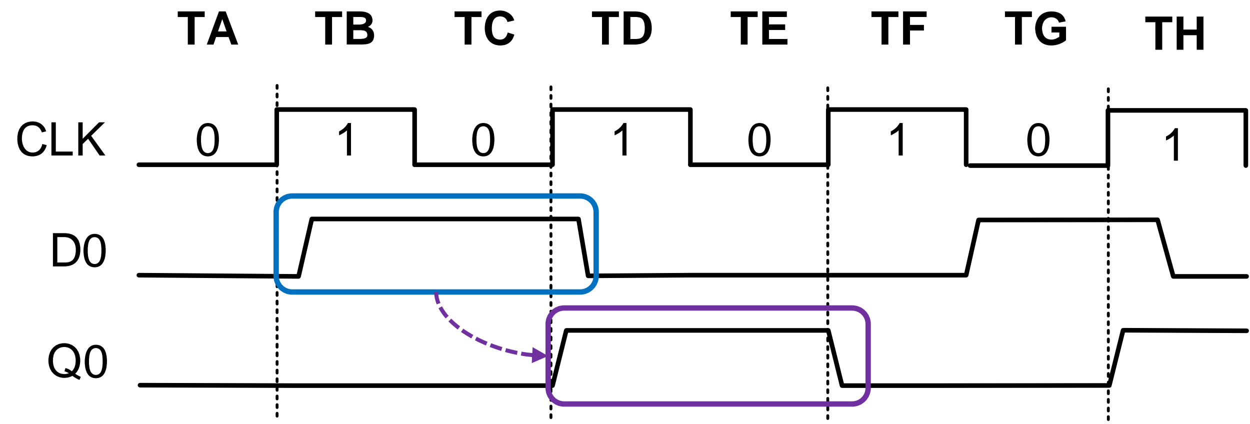
State Output Table

Key Idea

- So we know the current state value and the desired next state value for our state flip-flops, but how do we "make" the desired next value
- **Key**: The __________ value of a FF at the end of the **current** clock will be the value of Q for the full **next** clock cycle
  - Analogy: How you prepare and study **today** will determine your performance on the exam **tomorrow**!
- **Conclusion**: To make Q* = 1 (or 0) next cycle, make __ = 1 (or 0) now! (________)

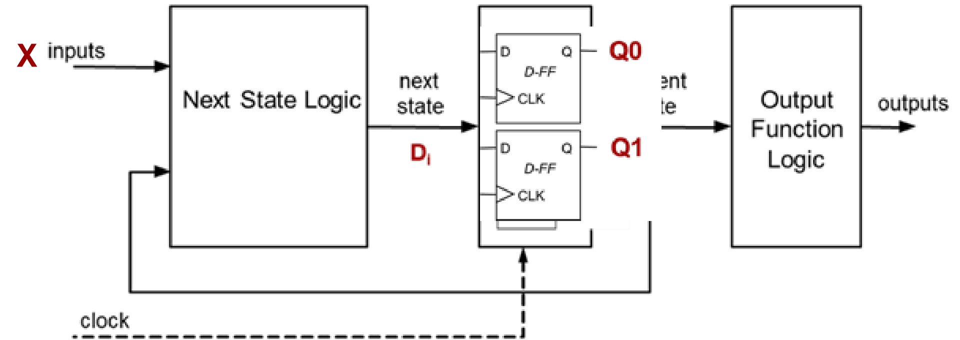
The D-input of a FF on one clock cycle becomes the Q value on the next.





HW FSM Implementation (3)

- **Step 3a,b**: Replace the state names with the binary codes you've assigned in each table
- You now have truth tables that you can use to find combinational logic equations:
- For the next state logic, the inputs are **current state** (i.e. FF outputs **Q1,Q0**) and the **input**(s). The table shows the desired next state (**D-inputs**)
- For the output function logic the inputs are the **current state** (i.e. FF outputs **Q1,Q0**)



	Input (X)	
State (Q1Q0)	**X=0 (D1D0)**	**X=1 (D1D0)**
S0=00	S0=00	S1 = 01
S1=01	S0=00	S2 = 10
S2=10	S0=00	S2 = 10
NA=11	- = dd	- = dd

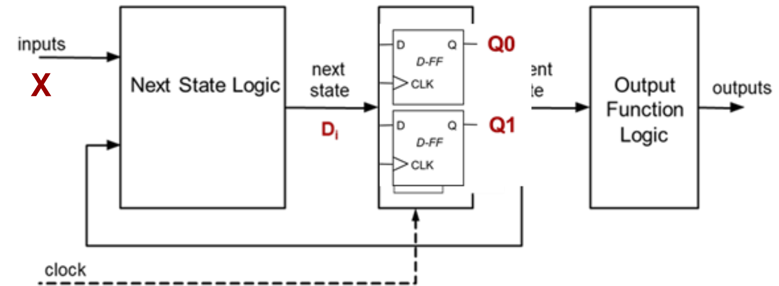
(Next) State Transition Table

State (Q1Q0)	Outputs
S0=00	0
S1=01	0
S2=10	1
NA=11	d

State Output Table

HW FSM Implementation (4)

- **Step 4**: Use the truth tables to find logic for the D inputs (next state logic) and output



	Input (X)	
State (Q1Q0)	X=0 (D1,D0)	X=1 (D1,D0)
S0=00	S0=00	S1 = 01
S1=01	S0=00	S2 = 10
S2=10	S0=00	S2 = 10
NA=11	- = dd	- = dd

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

D1 = X•Q1+X•Q0

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

D0 = X•Q1'•Q0'

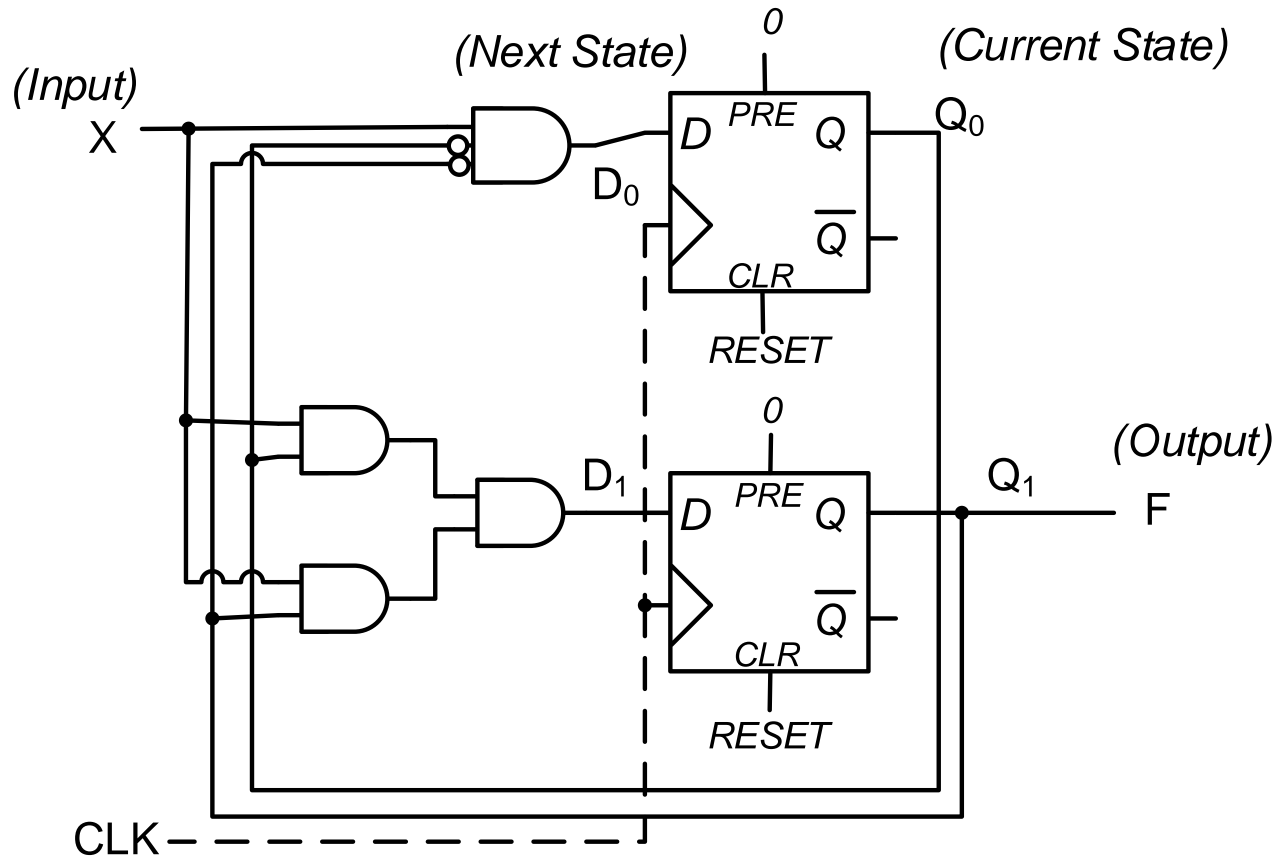
State (Q1Q0)	Outputs
S0=00	0
S1=01	0
S2=10	1
NA=11	d

Q0 \ Q1	0	1
0	0	1
1	0	d

output = Q1 (just a wire)

Implementing the Circuit

- Implements the consecutive 1s detector



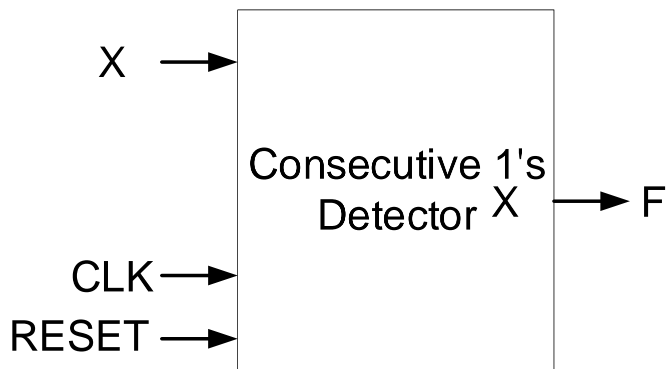
Simplified design process when using 1-hot coding of states

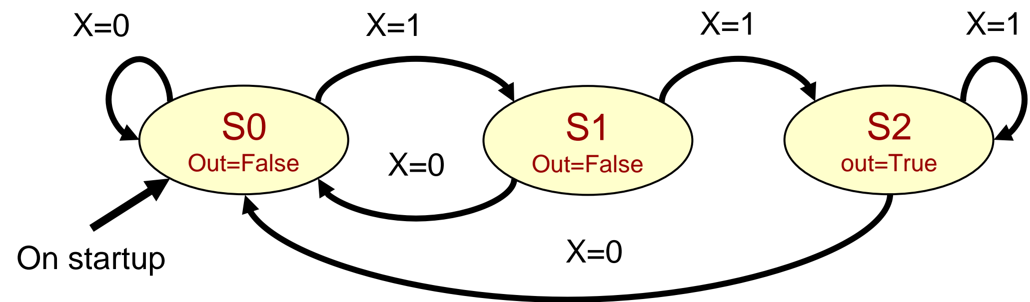
1-HOT STATE MACHINE DESIGN

EXAMPLE 1

Consecutive 1 Detector

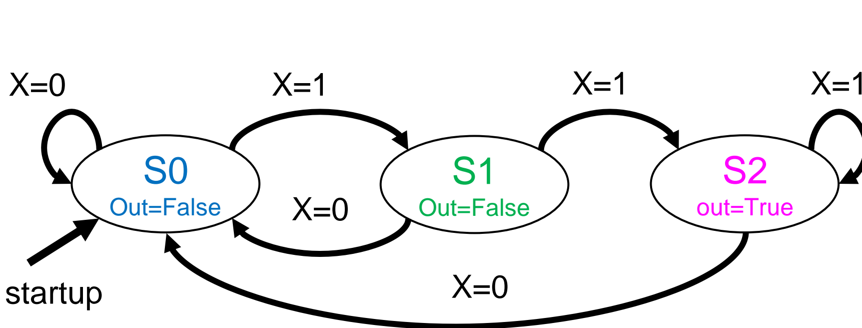
- Given a single-bit input, X, set the output to 1 if the last 2 values of X have been 1

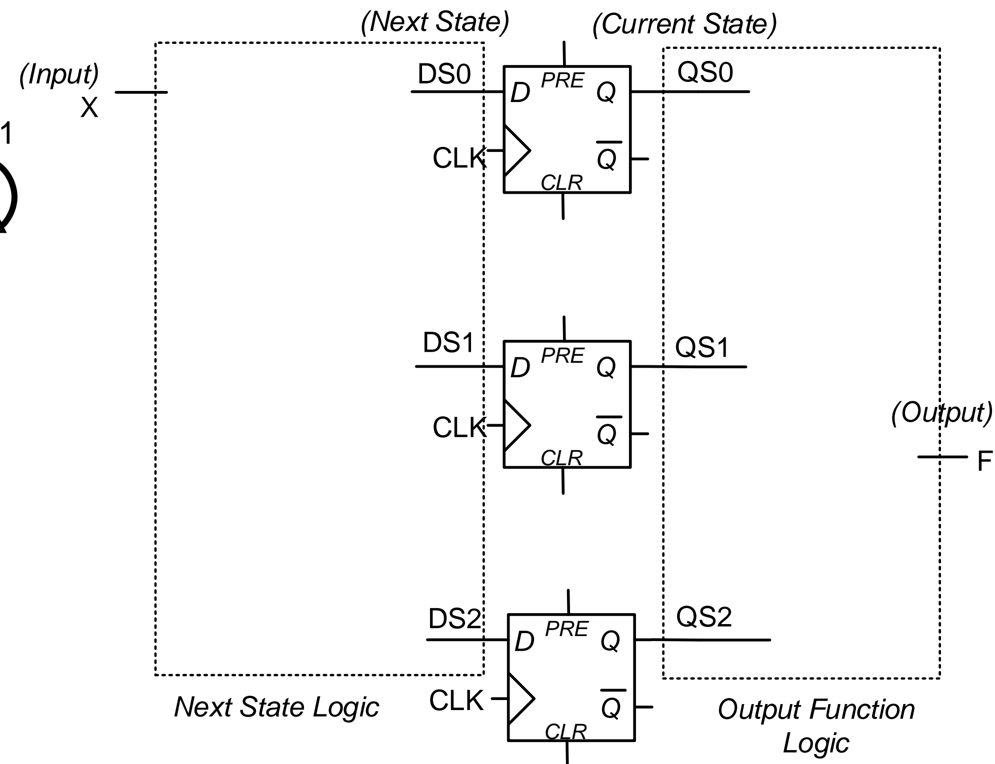




Consecutive 1 Detector - SM

- Given n-states in your state diagram, use __-FFs (___ per state) and assign 1-hot codes
- If a flip-flop output is ___, the FSM is in that state

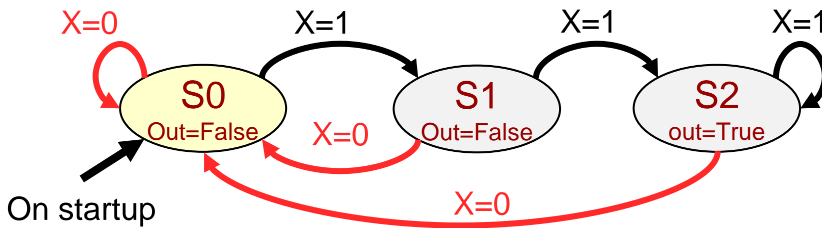


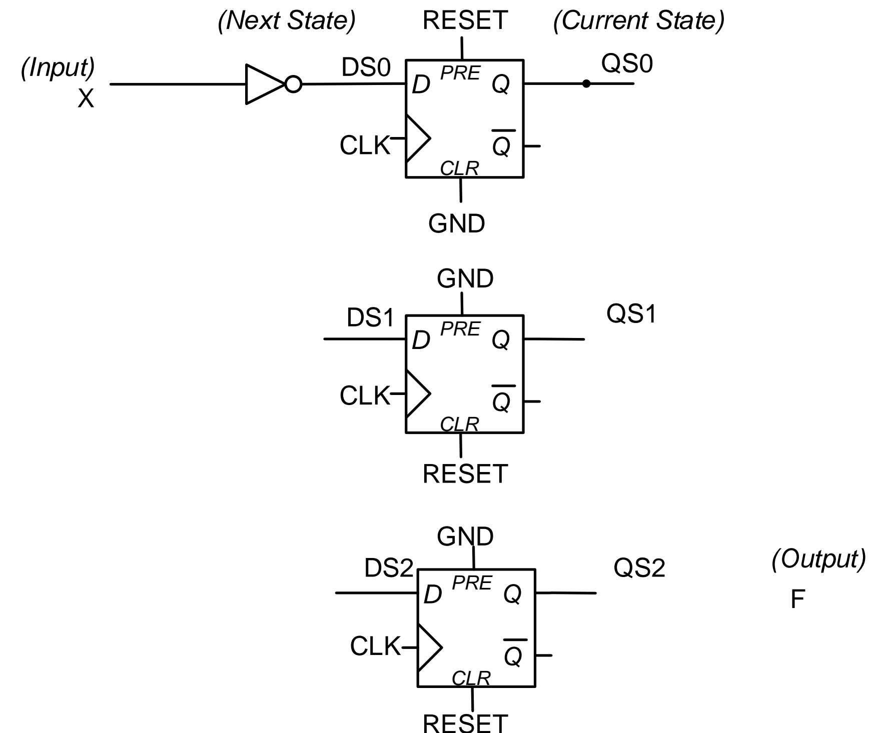


State	QS2 QS1 QS0
S0	0 0 1
S1	0 1 0
S2	1 0 0

Consecutive 1 Detector – NSL (a)

- To find the D-input logic for each FF **look at the arrows pointing _______ the corresponding state**
  - For each arrow, ______ the corresponding originating state FF output with the input condition associated with the transition
  - OR together each transition arrow's AND gate (simplifying as necessary)





DS0 = ________________________

(If in state S0 and X=0 OR
in state S1 and X=0, OR
in state S2 and X=0)

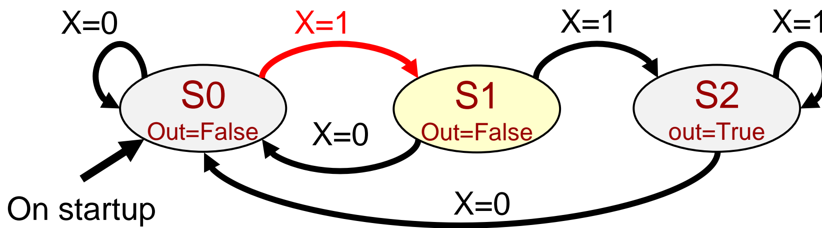
= X'•(QS0 + QS1 + QS2)

= X'•1 = X'

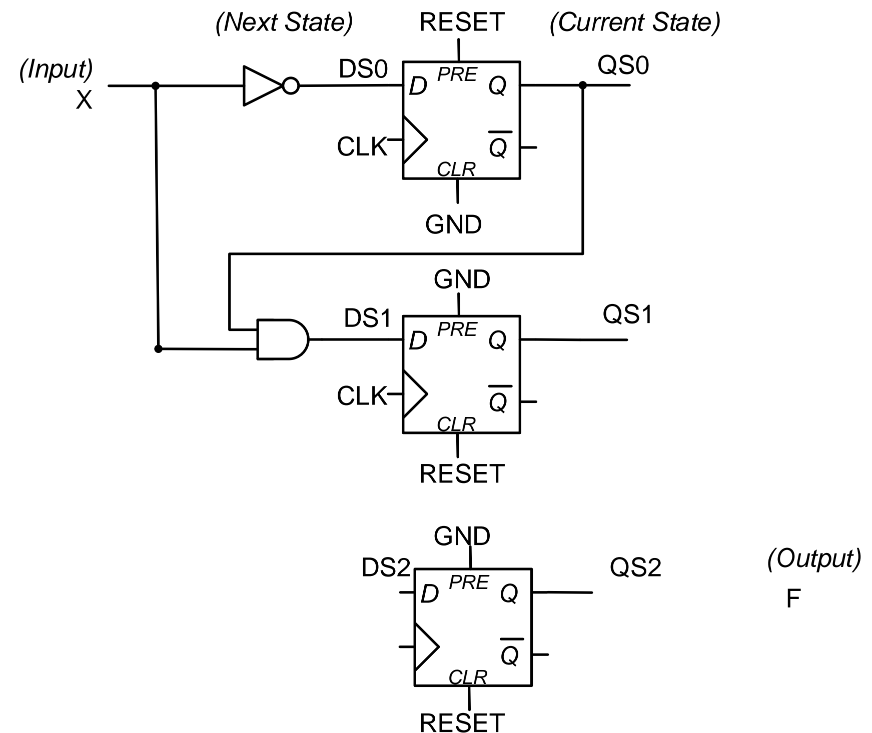
USC Viterbi School of Engineering

Consecutive 1 Detector – NSL (b)

- To find the D-input logic for each FF Look at the arrows pointing INTO the corresponding state
  - For each arrow, AND the corresponding originating state FF output with the input condition associated with the transition
  - OR together each transition arrow's AND gate (simplifying as necessary)

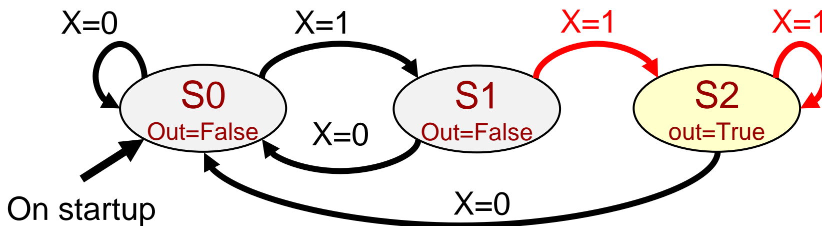


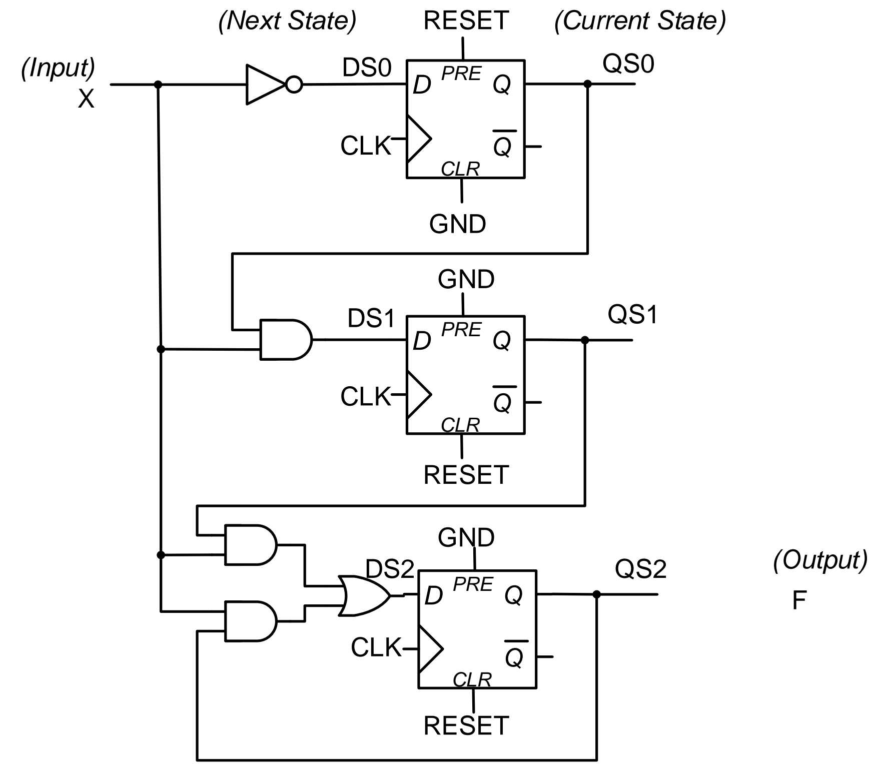
DS1 = ________



Consecutive 1 Detector – NSL (b)

- To find the D-input logic for each FF Look at the arrows pointing INTO the corresponding state
  - For each arrow, AND the corresponding originating state FF output with the input condition associated with the transition
  - OR together each transition arrow's AND gate (simplifying as necessary)

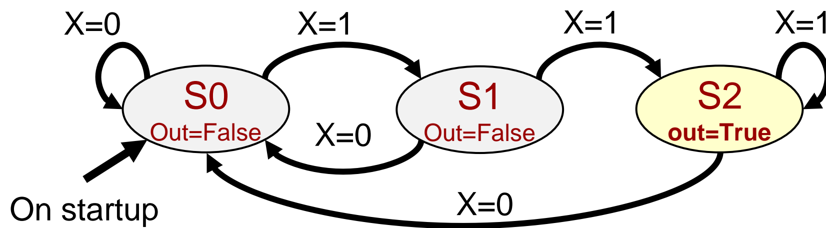


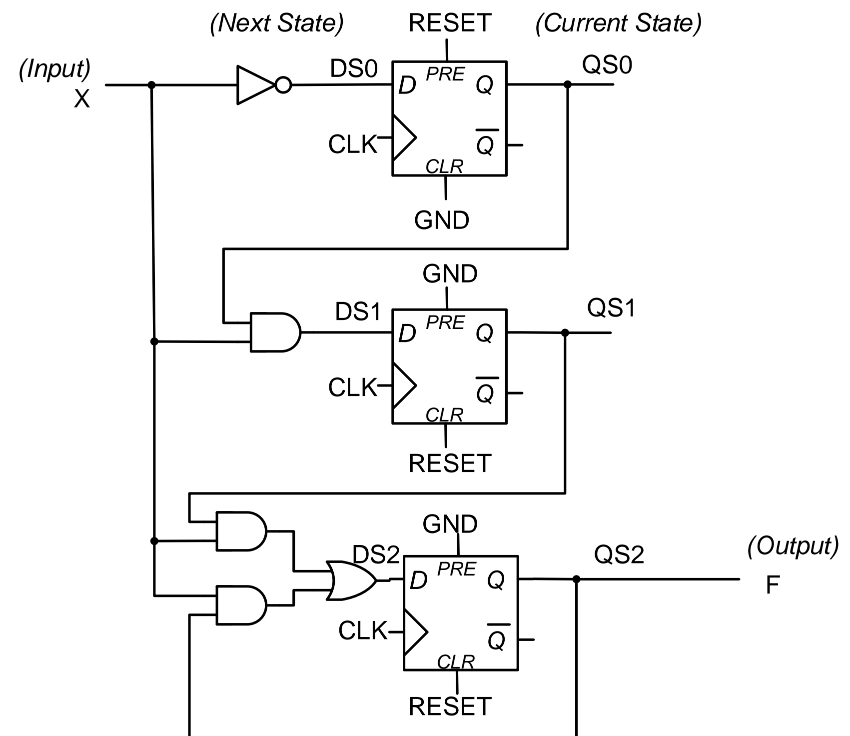


DS2 = _______________

Consecutive 1 Detector - OFL

- To find the output logic
  - OR together the FF output of each state where the output is true (if the output is true in only one state, then that FF output serves as the desired output of the state machine)

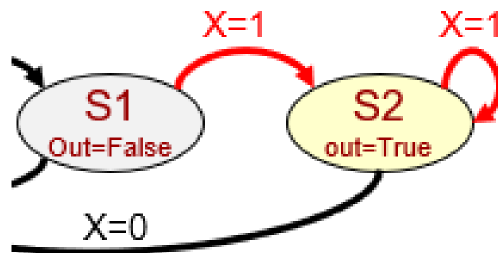


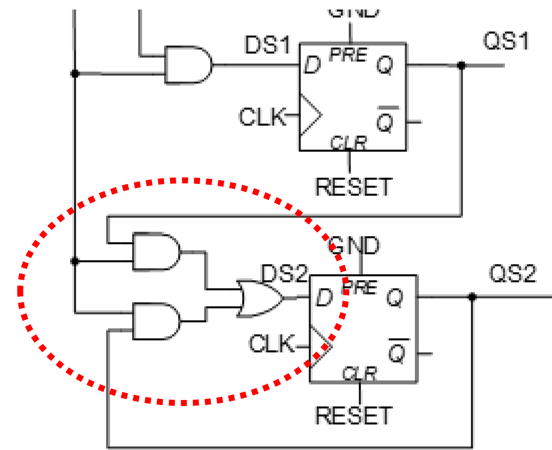


F = ______

Next State Summary and Why

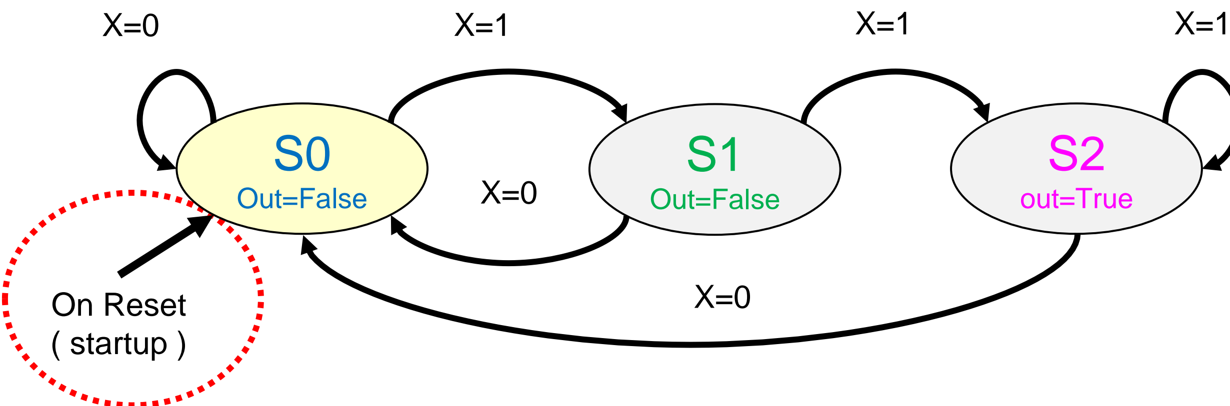
- The D-input of a flip-flop determines its next value
  - The value of D on the edge becomes the value of Q for the full next cycle
- In a state diagram, the arrows pointing TO (not from) a state indicate how that state can become the ______ state
- Each arrow pointing to a state is a _____________________ leading to that state becoming the next state
  - Each arrow can be converted to logic by checking we are in the current state where the arrow originates, and the inputs match the condition associated with that arrow (i.e. QS1=1 AND X=1)
- We then OR together each condition because if ANY (logical OR) is true, we want to transition to that state





Implementing an Initial State

- Flip-flops by themselves will initialize to a ____________ state (1 or 0) when power is turned on
- How can we make the machine start in S0 on reset (or power on?)
- Need QS2 to initialize to 1 while QS1 and QS0 initialize to 0

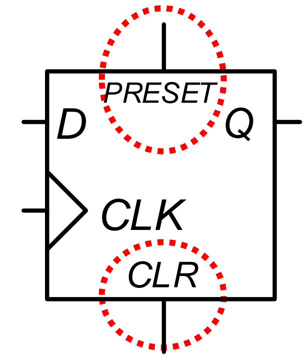


State	QS2 QS1 QS0
S0	0 0 1
S1	0 1 0
S2	1 0 0

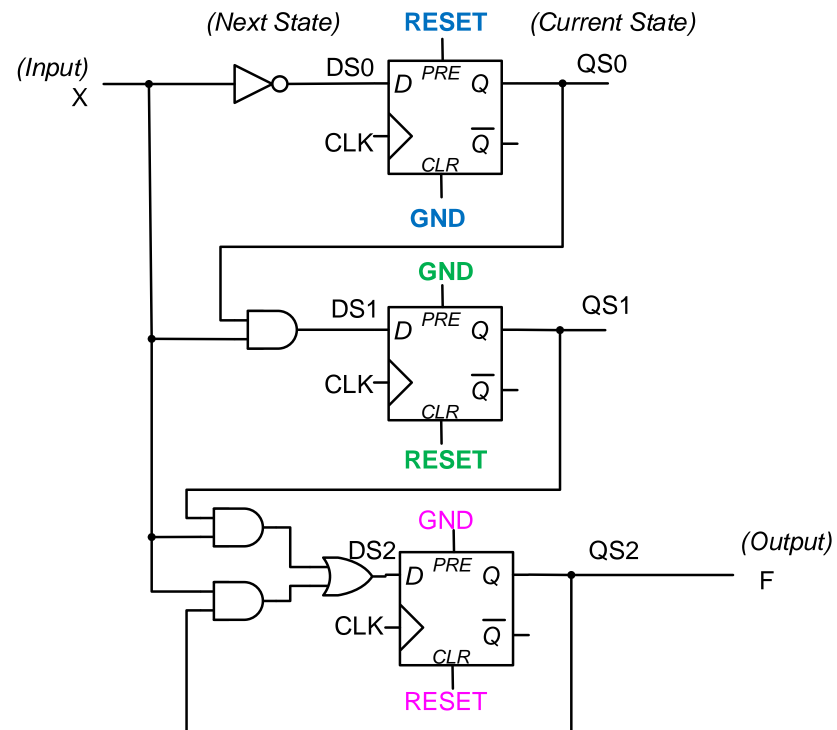
USC Viterbi School of Engineering

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=__
  - When PRESET is active the FF initializes Q=__

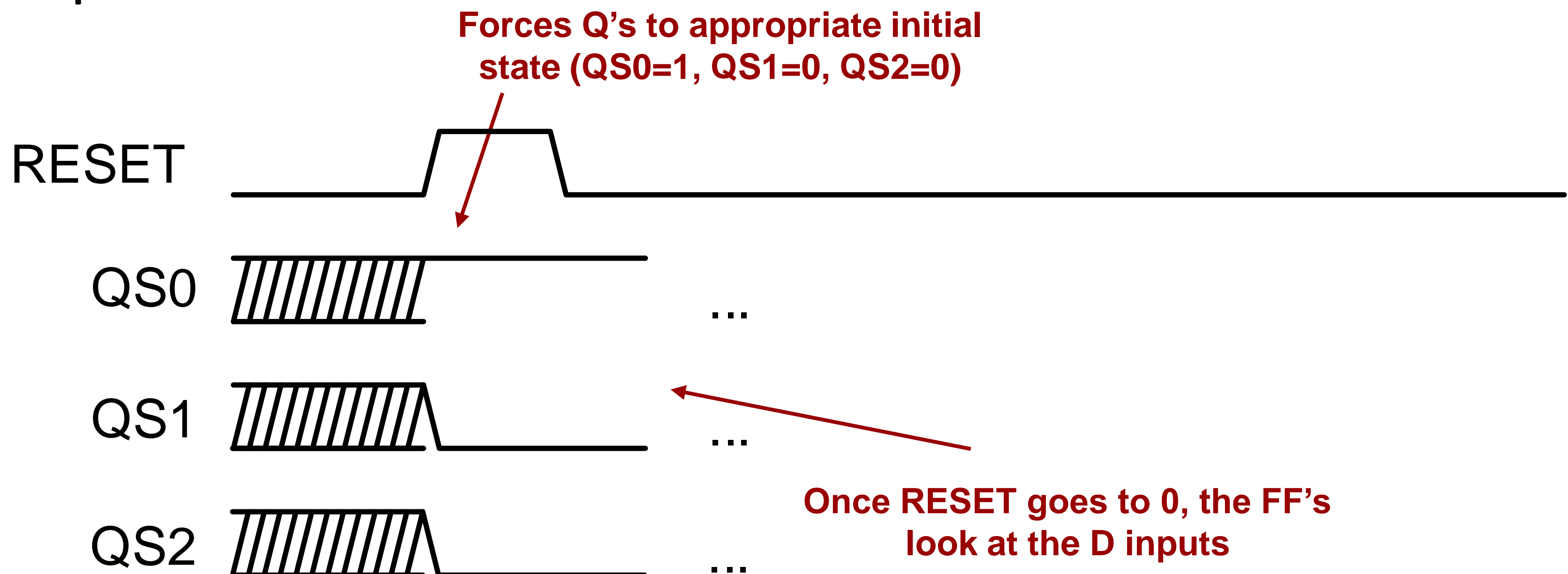


State	QS2 QS1 QS0
S0	0 0 1
S1	0 1 0
S2	1 0 0



Implementing an Initial State

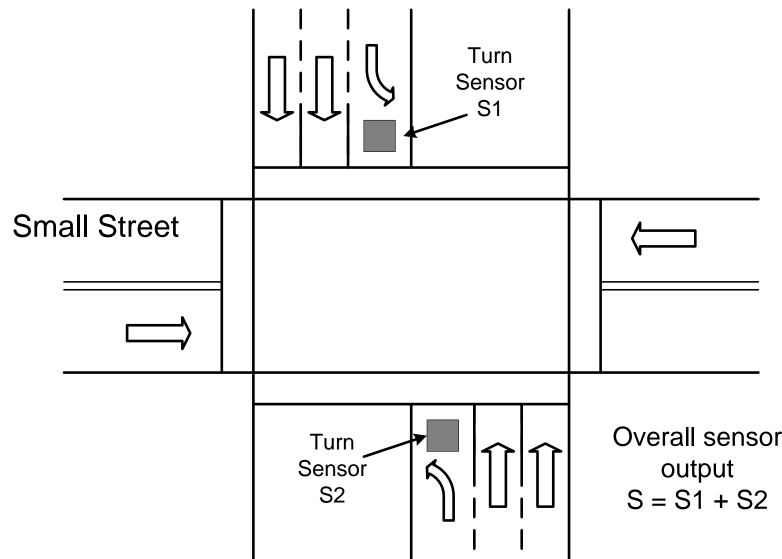
- When RESET is activated: Q’s initialize to starting state
- When RESET is deactivated: Q’s respond to the D inputs

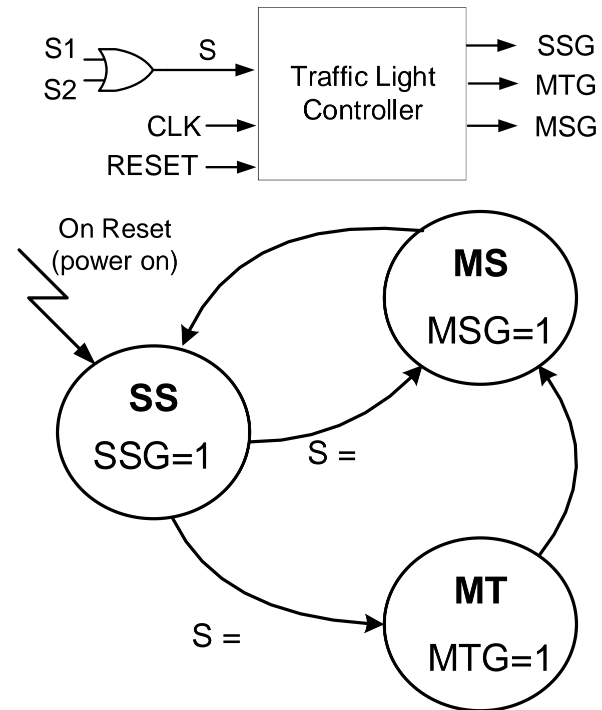


EXAMPLE 2

Traffic Light Controller

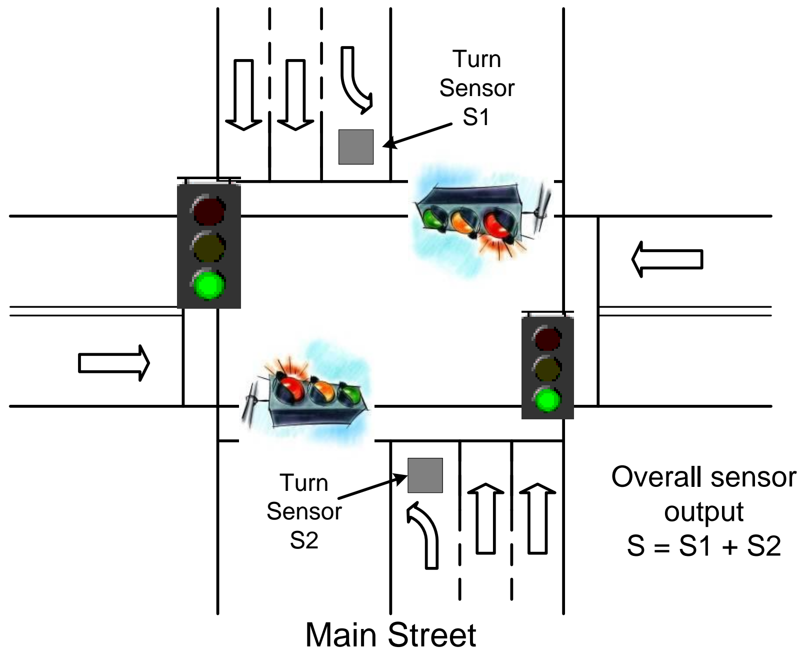
- Design the controller for a traffic light at an intersection
  - Main street has a protected turn while small street does not
    - Sensors embedded in the street to detect cars waiting to turn
    - Let S = S1 OR S2 to check if any car is waiting
  - Simplify and only have Green and Red lights (no yellow)

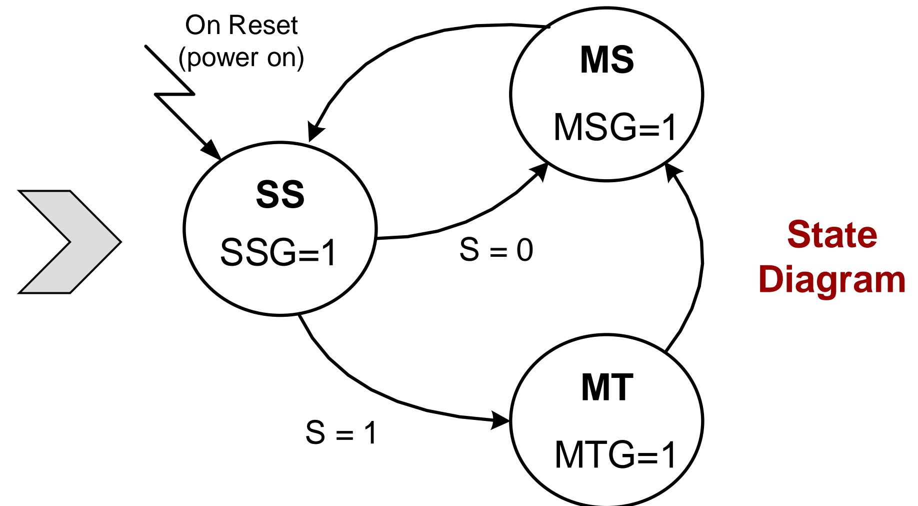




Traffic Light State Assignment

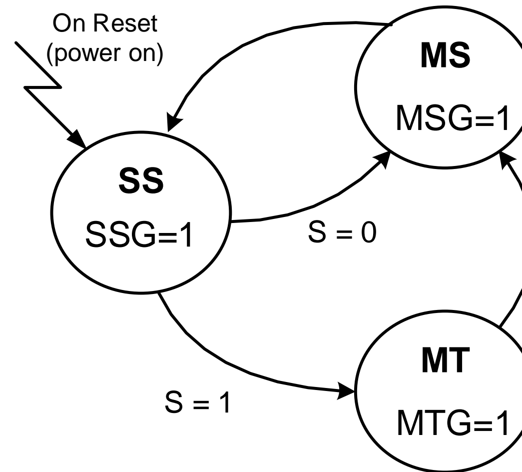
- Design of the traffic light controller with main turn arrow
- Represent states with binary codes
  - One-hot: Separate FF per state: 100=SS, 010=MS, 001=MT





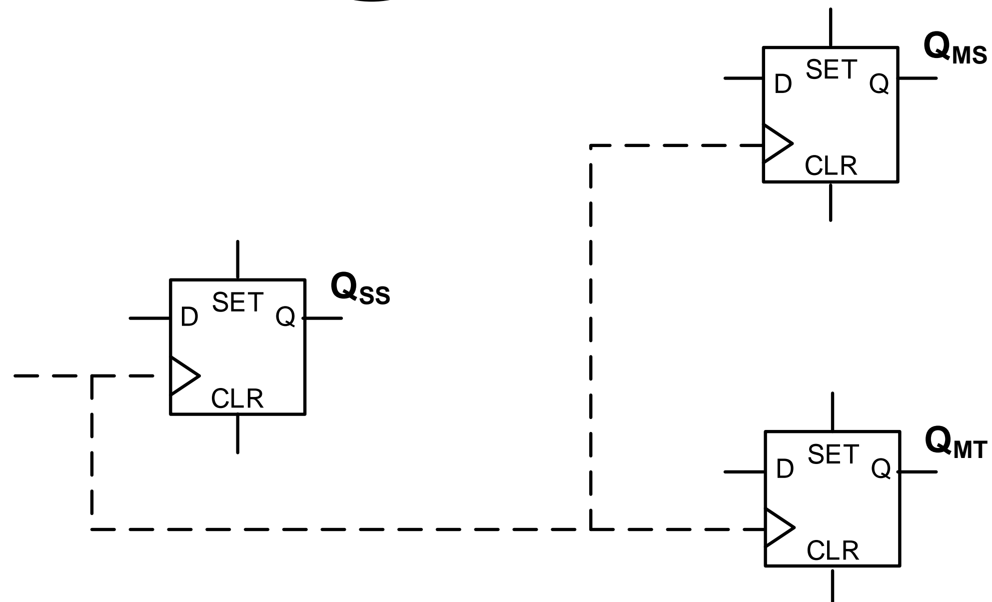
Traffic Light NSL

- In one-hot assignment, NSL is designed by simple observation
- For each state, examine each incoming transition
  - Each incoming arrow will be one case in our logic
  - We can just OR each condition together
- Describe each transition as a combination of what state it originates from & any associated conditions
- Ex. Two arrows converge on MS: "Q_{MS} should be '1' on the next clock when…
  - Current state is MT ***…OR…***
  - Current stat is SS **AND** S=0



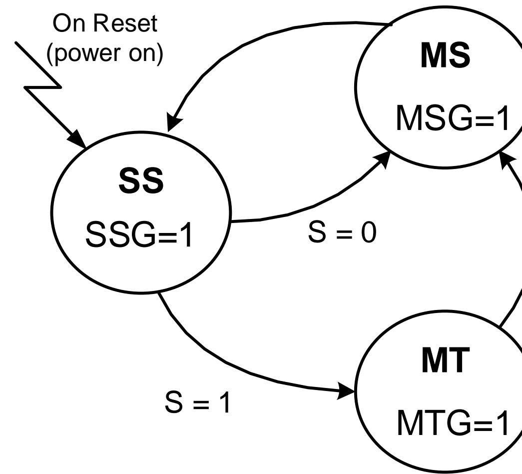
	Q_{SS}	Q_{MT}	Q_{MS}
SS	1	0	0
MT	0	1	0
MS	0	0	1

One-hot State Assignment



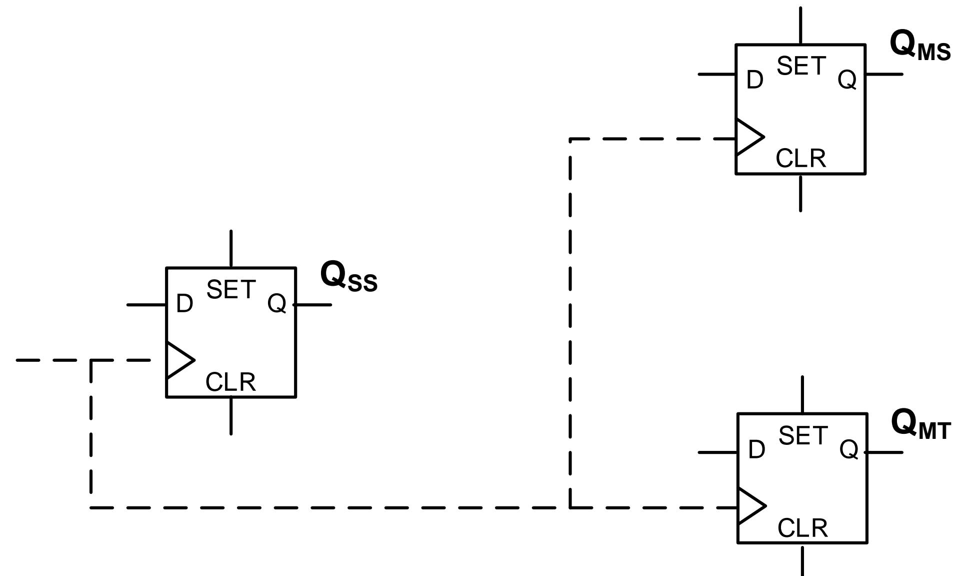
NSL Implementation in 1-Hot Method

- Two arrows converge on MS: "Q_{MS} should be '1' on the next clock when…
  - Current state is MT *…OR…*
  - Current stat is SS **AND** S=0
- **$Q^*_{MS} = D_{MS}$ = ______________**
- **$Q^*_{MT} = D_{MT}$ = ______________**
- **$Q^*_{SS} = D_{SS}$ =______________**
- **Outputs:**
  - **MSGreen = Q_{MS}**
  - **MTGreen = Q_{MT}**
  - **SSGreen = Q_{SS}**
  - **Red lights are inverse of greens**
- **What about initial state? Preset the appropriate flip flop, and clear otherwise**



	Q_{SS}	Q_{MT}	Q_{MS}
SS	1	0	0
MT	0	1	0
MS	0	0	1

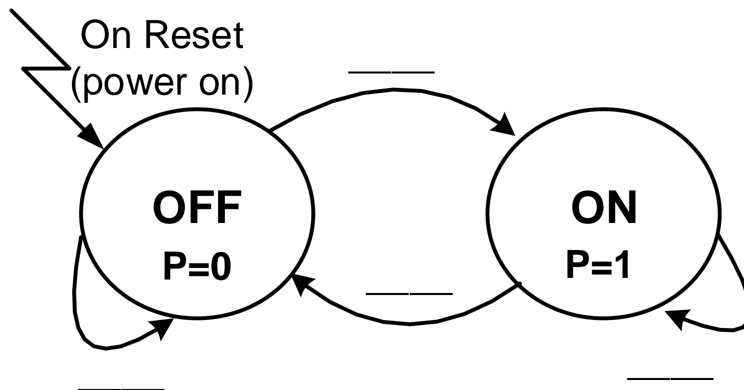
One-hot State Assignment



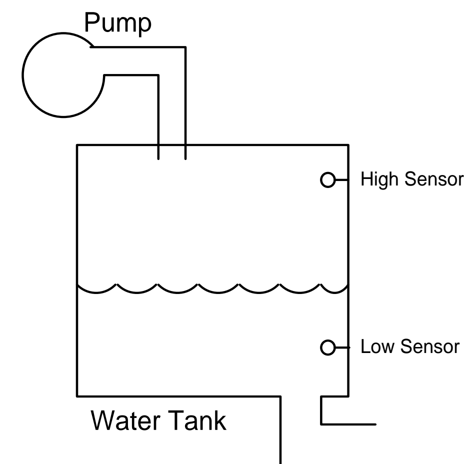
EXAMPLE 3

Water Pump

- Implement the water pump controller using the High and Low sensors as inputs
  - Recall the H and L sensor produce 1 when water is covering them and 0 otherwise

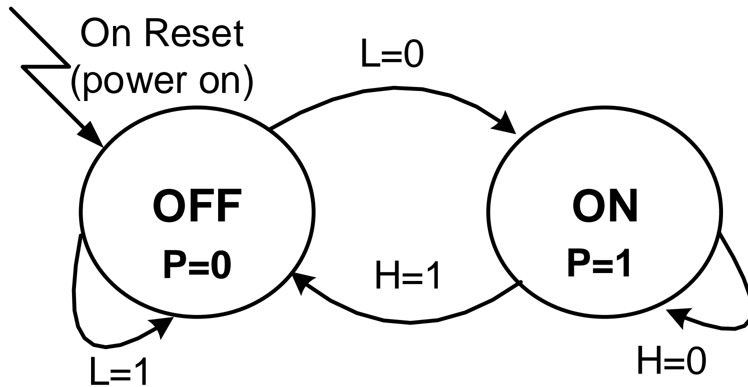


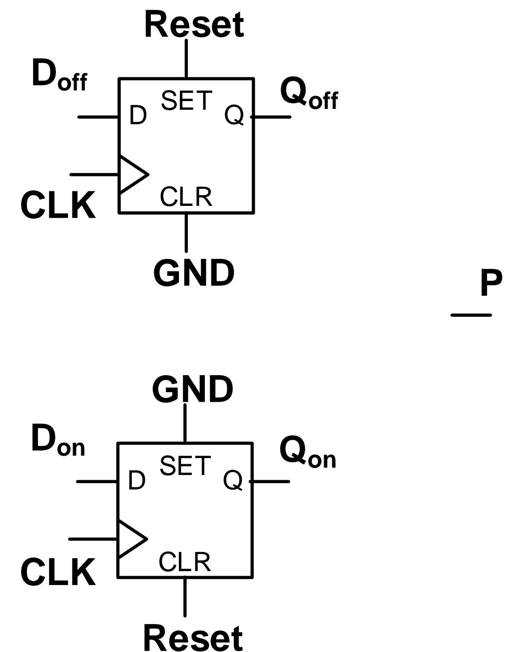
Notice that in each state, only 1 input matters.
For example, we stay in the OFF state until L=0 regardless of H.
We could write the transition from OFF to ON as:
L=0 and (H=1 OR H=0)
but (H=1 OR H=0) is always True and L=0 and True => L=0, so we simply drop H's value



Water Pump

- Implement the water pump controller using the High and Low sensors as inputs
  - Recall the H and L sensor produce 1 when water is covering them and 0 otherwise



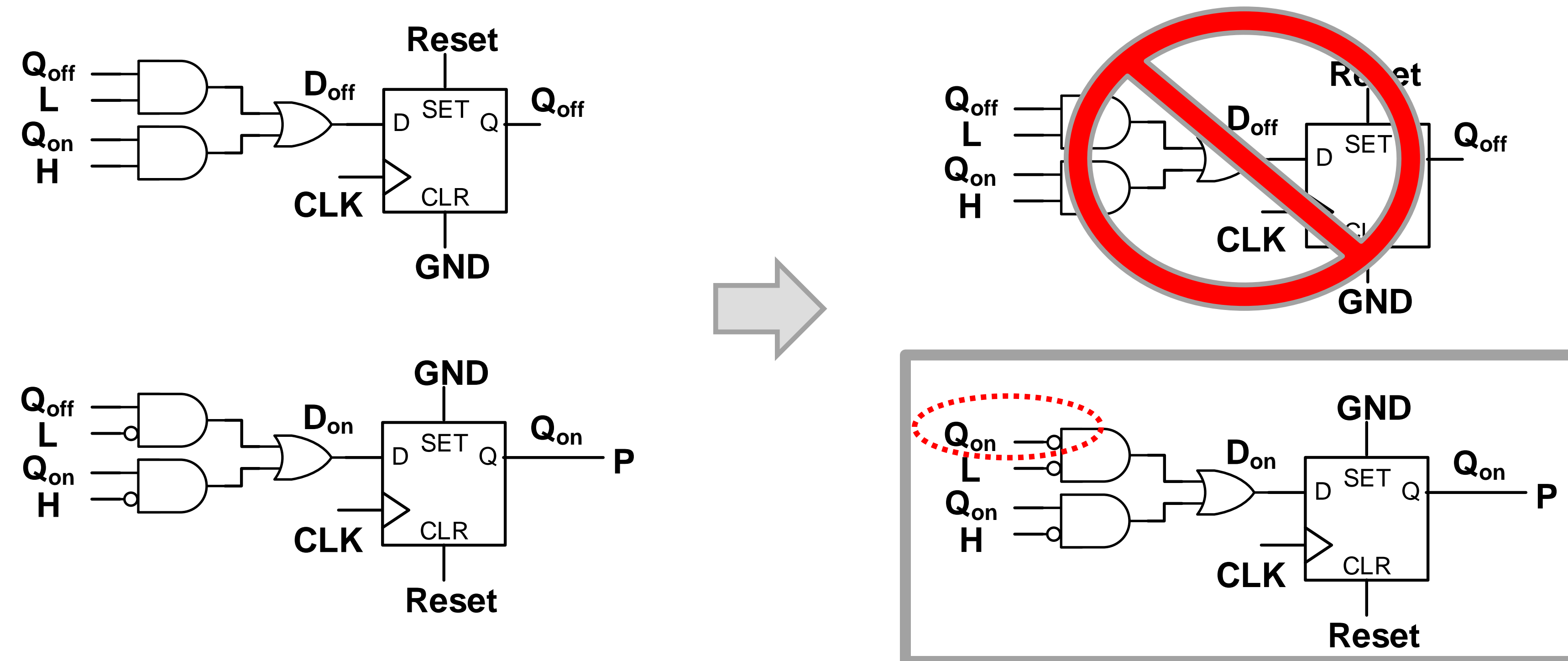


Doff = ________________

Don = ________________

Simplifying

- For 2 states, 1-hot coding implies the states are the inverse of each other (i.e. Qon = ~Qoff) and can usually be simplified using that relationship

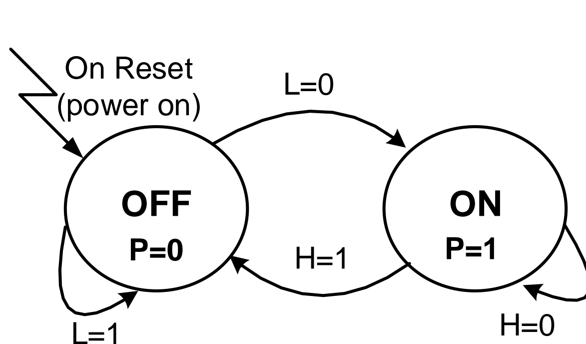


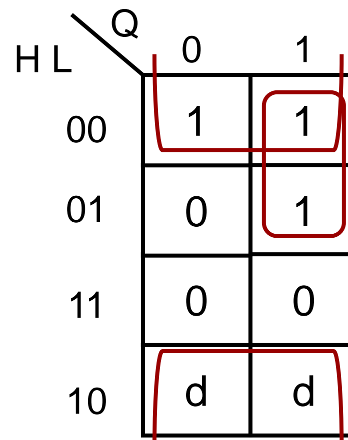
Transition Table

- Other codes may help us simplify
  - Using a 1-bit state code and a K-map we can simplify…

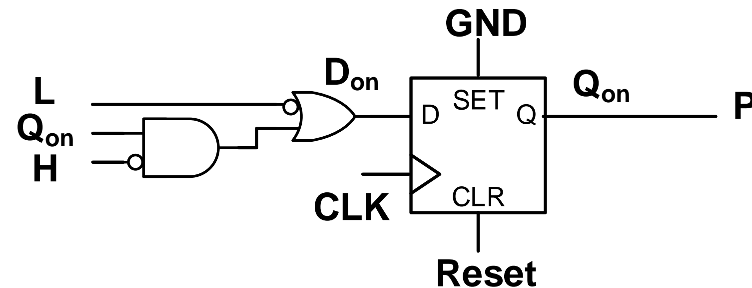
Current State		Next State							
		H L = 0 0		H L = 0 1		H L = 1 1		**H L = 1 0**	
Symbol	Q	Sym.	Q*	Sym.	Q*	Sym.	Q*	Sym.	**Q***
OFF	0	ON	1	OFF	0	OFF	0	X	**d**
ON	1	ON	1	ON	1	OFF	0	X	d

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





D = L' + H'Q

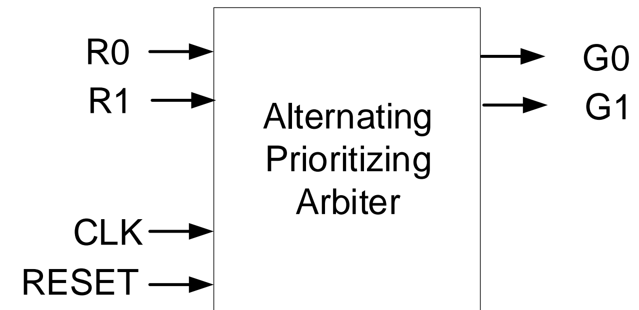


Alternating Priority Arbiter

EXAMPLE 4

Problem Description

- Two digital devices (Device 0 and Device 1) can request to use a shared resource via individual request signals: R0 (from Dev0) and R1 (from Dev1)
- An arbiter will examine the requests and issue a grant signal to the appropriate device (G0 to Dev0 and G1 to Dev1).
- Requests are examined during 1 cycle and a grant will be generated on the next, and active for one cycle
- If only one device makes a request during a cycle, it should receive the grant on the next.
- If both devices request on the same cycle, the grant should be given to the device who hasn't received a grant in the longest time.



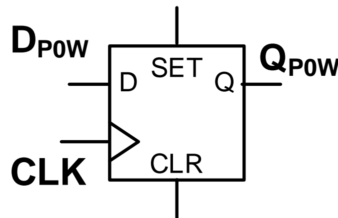
Cycle	R0	R1	G0	G1
1	1	0	-	-
2	1	1	1	0
3	1	1	0	1
4	0	0	1	0
5	1	0	0	0
6	1	1	1	0
7	-	-	0	1

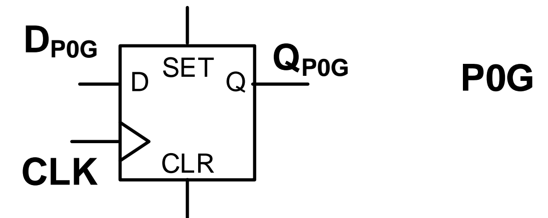
Ex 4: State Diagram Design

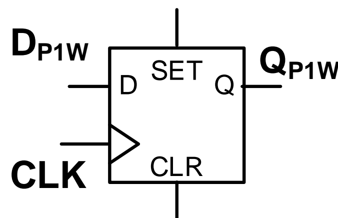
- **Exercise**: Design a state diagram to solve the alternating priority arbiter
  - Consider how many states you need and what each one helps you remember or achieve

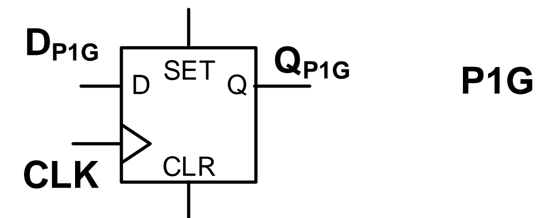
Ex 4: Implementation

- **Exercise**: Implement the state machine you designed on the previous page





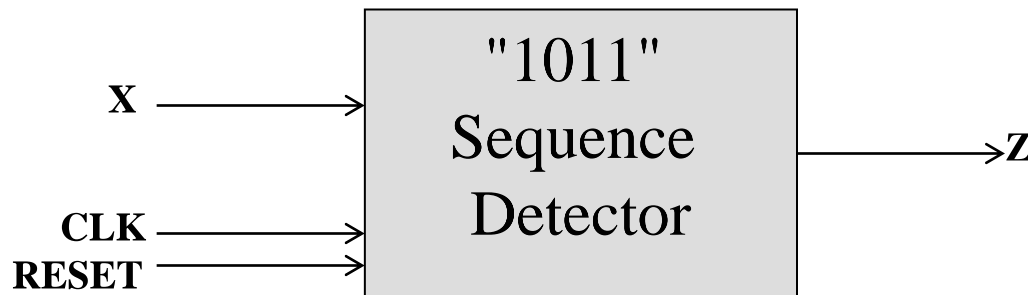




EXAMPLE 5

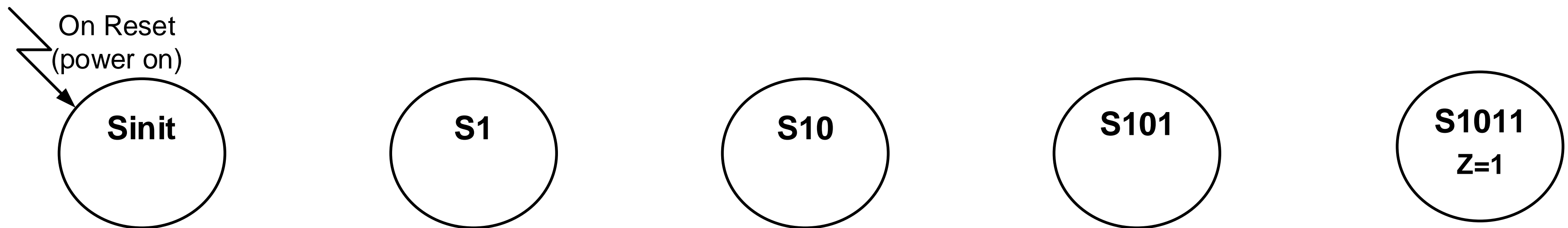
State Machine Example

- 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)

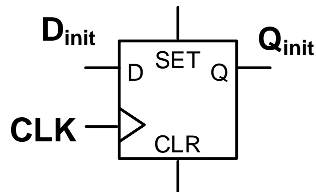


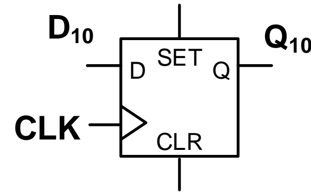
Ex 5: State Diagram Design

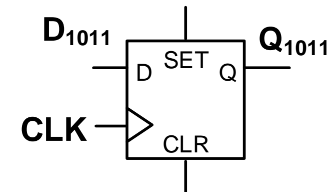
- **Exercise**: Design a state diagram to solve the "1011" sequence detector
  - Be sure to handle overlapping sequences and use 1-hot state coding



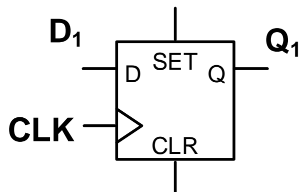
Ex. 5: Implementation

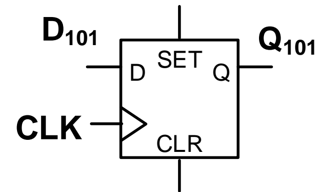




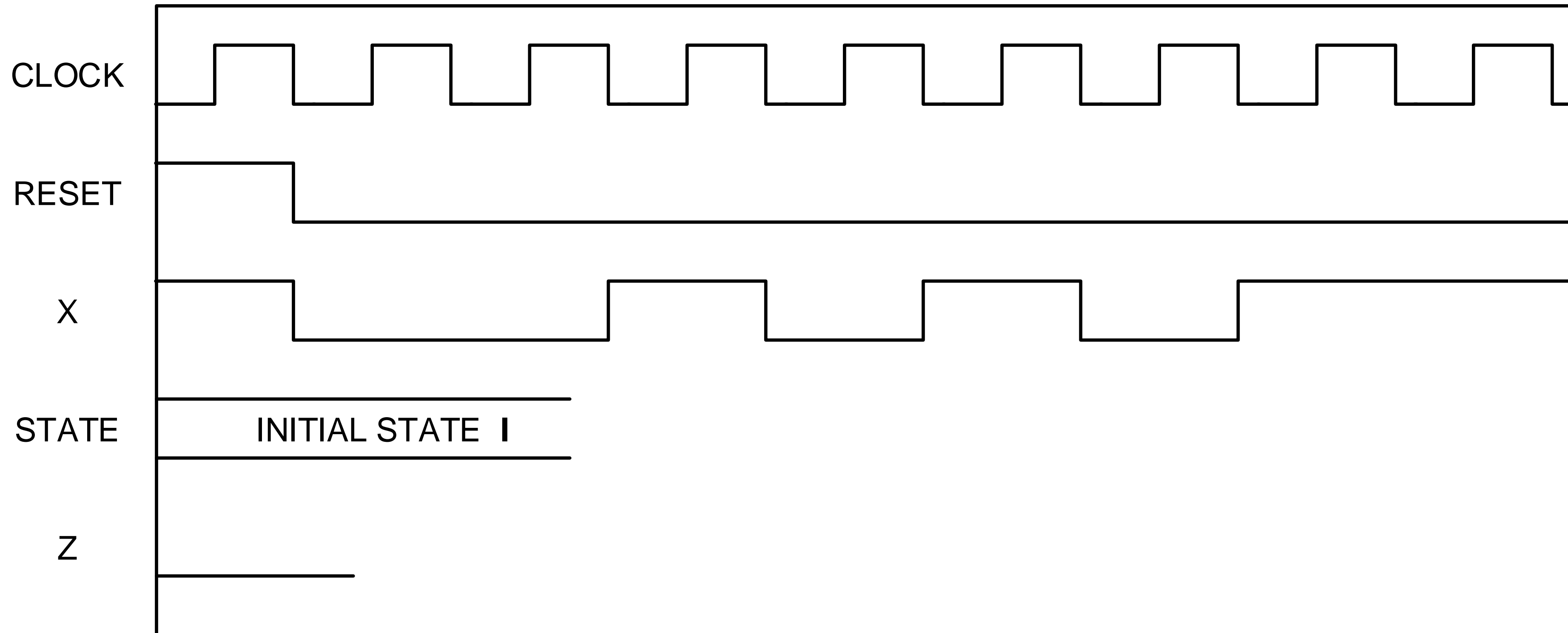


Z





Waveform for 1011 Detector



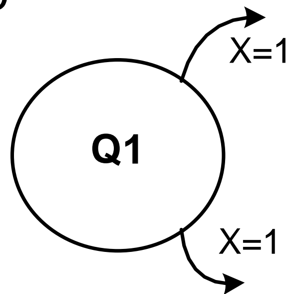
CORRECTLY SPECIFYING STATE DIAGRAMS

State Machine Design

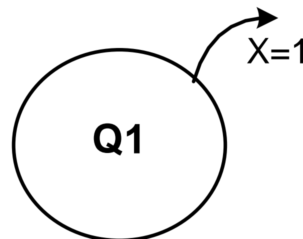
- Coming up with a state diagram is non-trivial
- Requires creative solutions
- Designing the circuit from the state diagram is done according to a simple set of steps
- To come up w/ a state diagram to solve a problem
  - Write out an algorithm or series of steps to solve the problem
  - Each step in 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

Correct Specification of State Diagrams

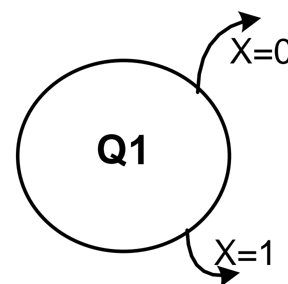
- For HW especially, it is critical that **exactly _____ transition from a state may be true at a time**
  - We can't go 2 places at once and if we don't tell it explicitly where to go next, it may go to any random state
  - If you want to stay in a state, include an explicit __________ arrow
    - In SW, the state variable will retain its value, **but in HW we must be explicit**
    - On the 2nd example if you want to stay in Q1, include a loopback labeled X=0



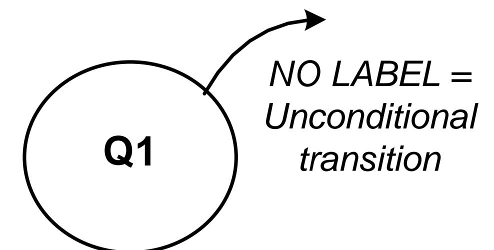
Incorrect (2 conditions true)



Incorrect (No condition for X = 0)



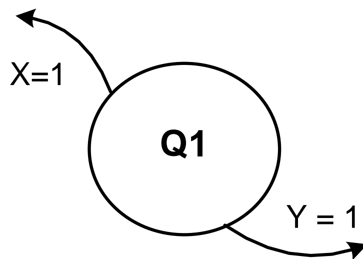
Correct



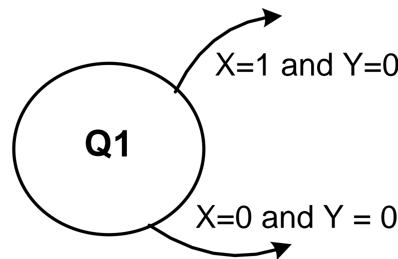
Correct

Correct Specification of State Diagrams 2

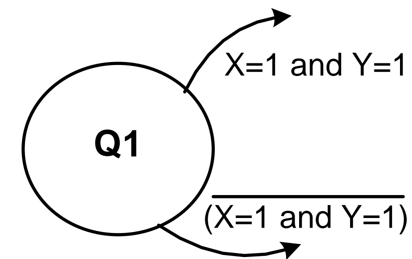
- **Exactly one transition from a state may be true at a time**
  - Make sure the conditions you associate with the arrows coming out of a state are **mutually exclusive** (< 2 true) and also **all inclusive** (> 0 true)



Incorrect (More than one condition can be true)



Incorrect (Not all conditions covered)

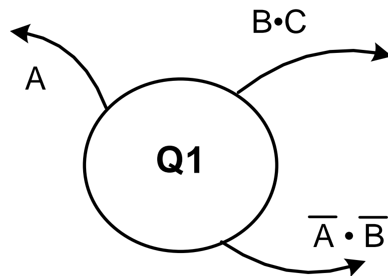


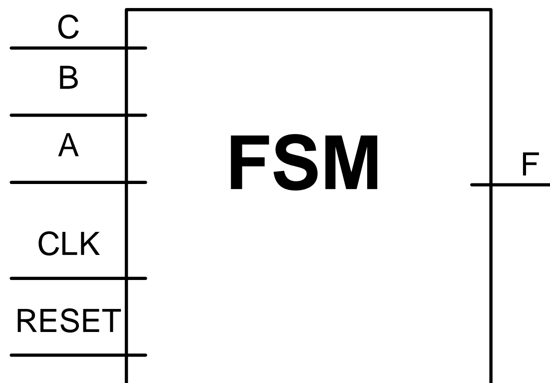
Correct (1 and only 1 condition will be true at all times)

ALWAYS double check your transitions to ensure they are mutually exclusive.

Relationship of Input Transitions

- A state machine with **k** inputs can have ___ transitions originating from each state
  - But many combinations can be combined with a simplified condition
- Verify if these transitions are valid (exactly 1 can be true)

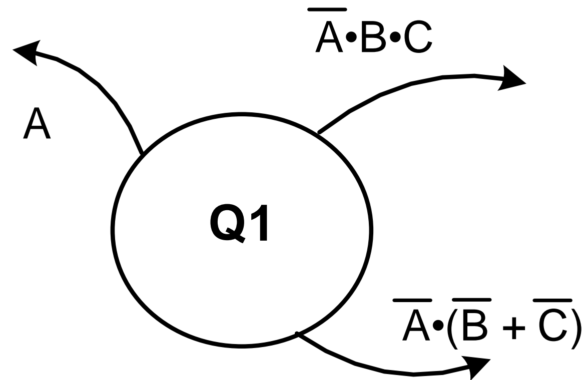


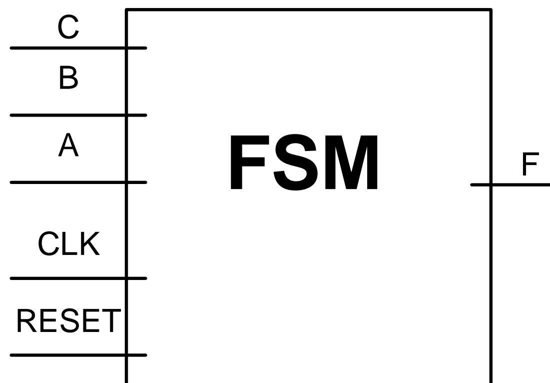


A	B	C	A	B•C	A'•B'
0	0	0	0		1
0	0	1	0		1
0	1	0	0		0
0	1	1	0		0
1	0	0	1		0
1	0	1	1		0
1	1	0	1		0
1	1	1	1		0

Updated Input Transitions

- Update the transitions so exactly 1 can be true





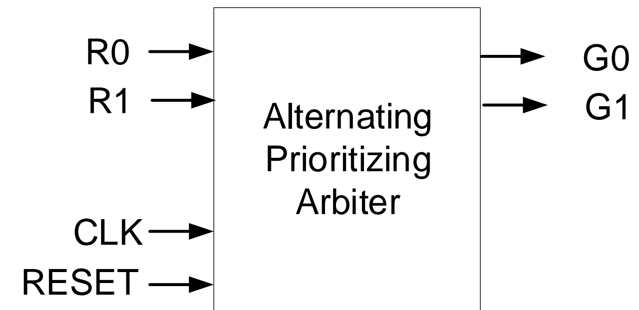
A	B	C	A	A'•B•C	A'•(B'+C')
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	1	0

Implementation of our state diagram approach

EXAMPLE 4 IMPLEMENTATION

Problem Description

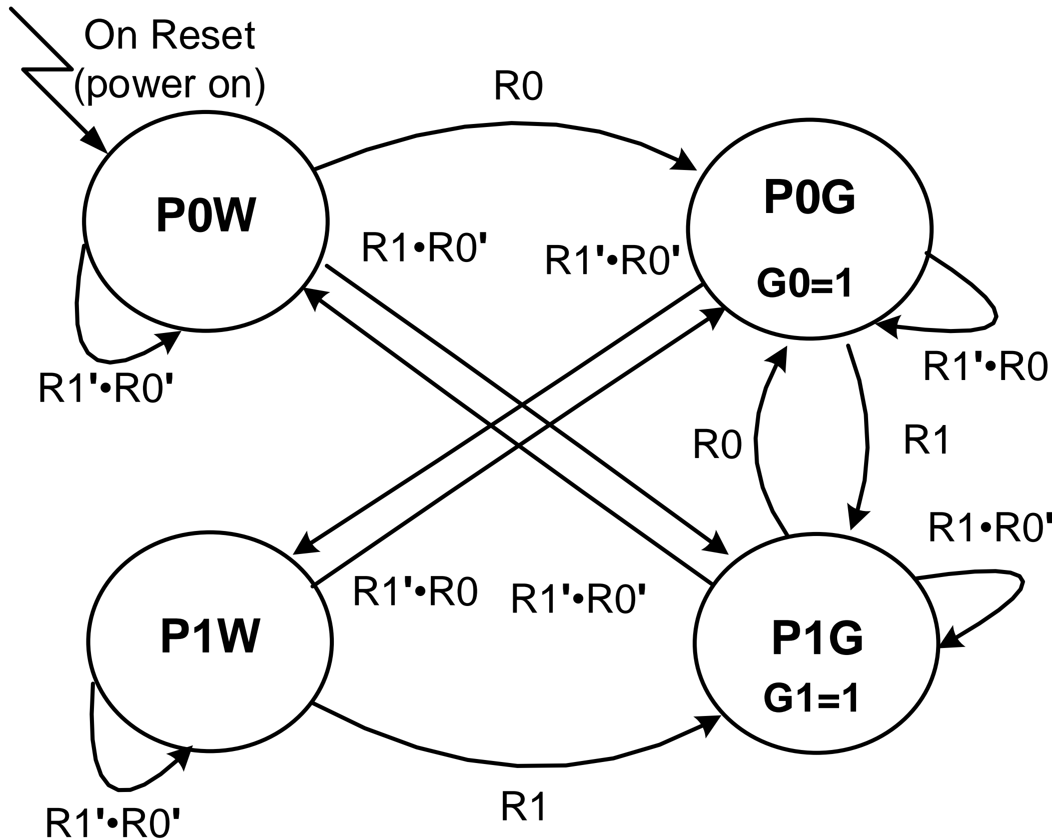
- Two digital devices (Device 0 and Device 1) can request to use a shared resource via individual request signals: R0 (from Dev0) and R1 (from Dev1)
- An arbiter will examine the requests and issue a grant signal to the appropriate device (G0 to Dev0 and G1 to Dev1).
- Requests are examined during 1 cycle and a grant will be generated on the next, and active for one cycle
- If only one device makes a request during a cycle, it should receive the grant on the next.
- If both devices request on the same cycle, the grant should be given to the device who hasn't received a grant in the longest time.



Cycle	R0	R1	G0	G1
1	1	0	-	-
2	1	1	1	0
3	1	1	0	1
4	0	0	1	0
5	1	0	0	0
6	1	1	1	0
7	-	-	0	1

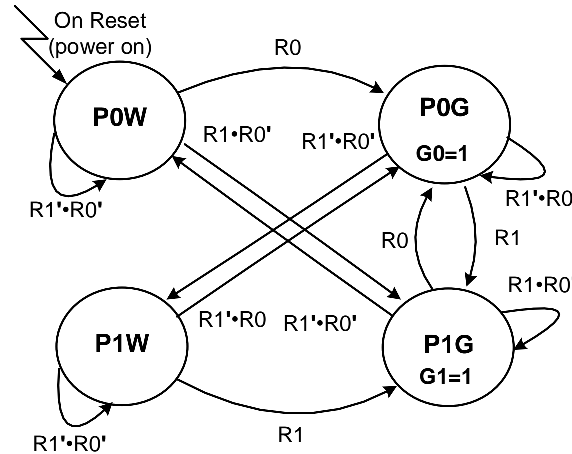
State Diagram

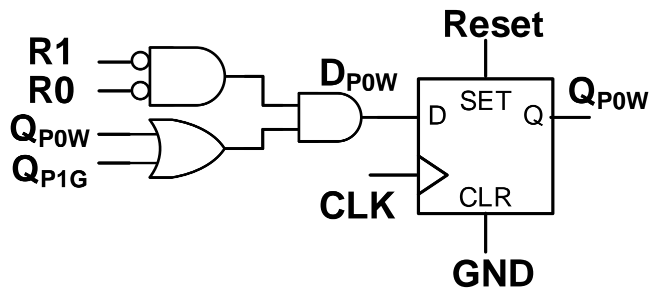
- Complete the state diagram

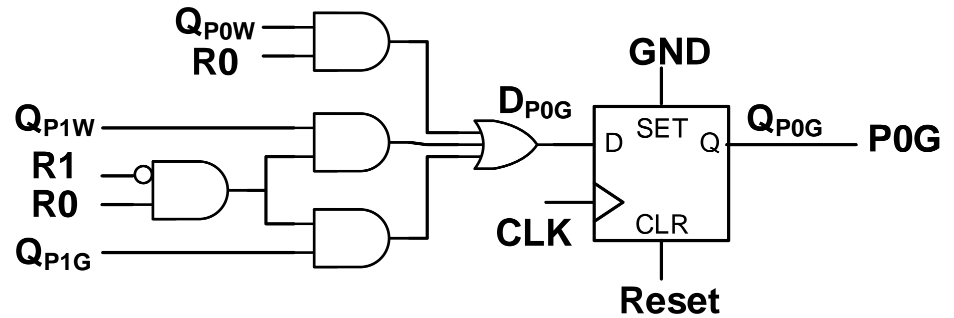


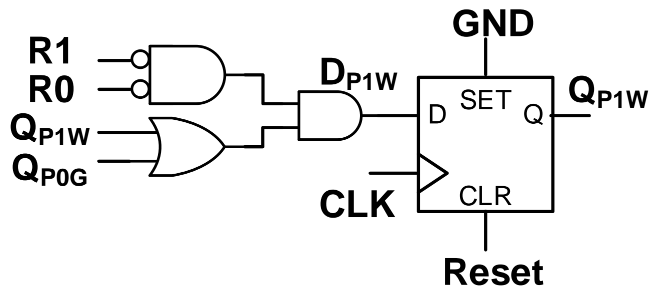
Cycle	R0	R1	St.	G0	G1
1	1	0	P0W	-	-
2	1	1	P0G	1	0
3	1	1	P1G	0	1
4	0	0	P0G	1	0
5	1	0	P1W	0	0
6	1	1	P0G	1	0
7	-	-	P1G	0	1

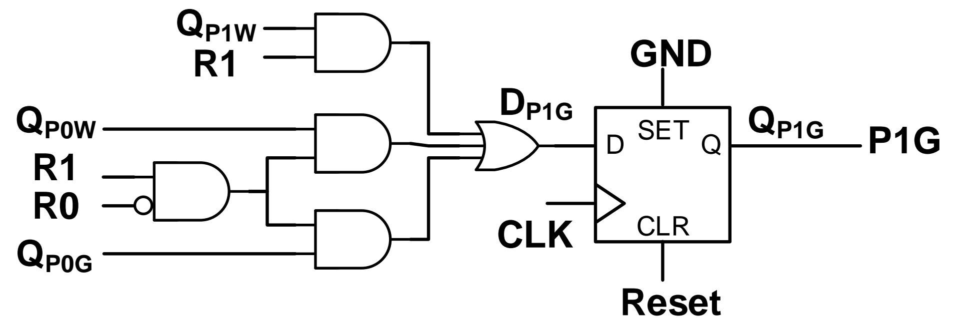
Final Circuit







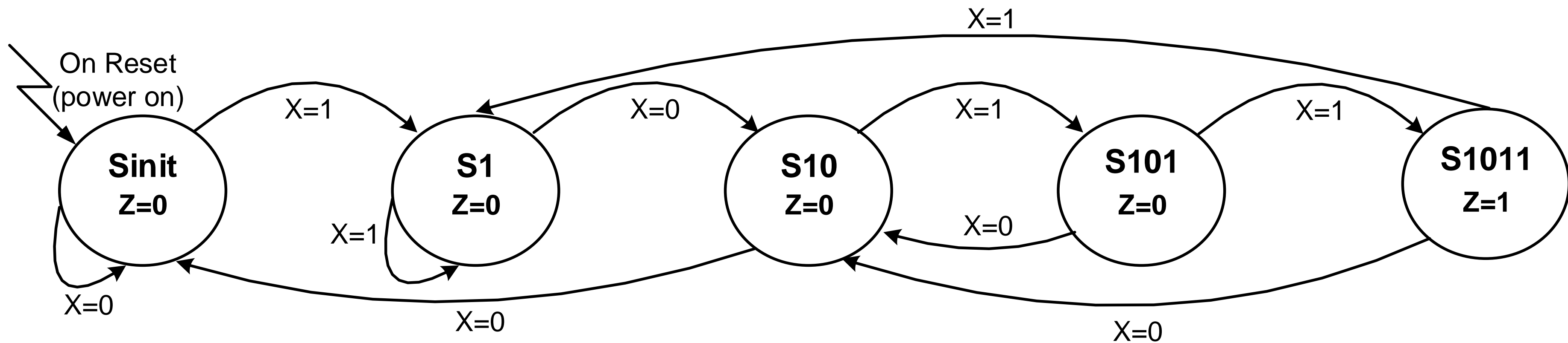




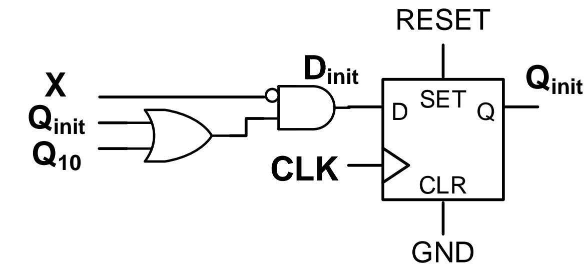
EXAMPLE 5 IMPLEMENTATION

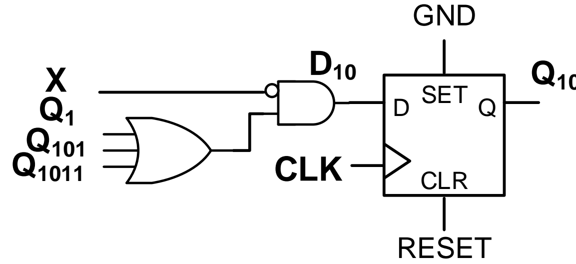
State Diagram

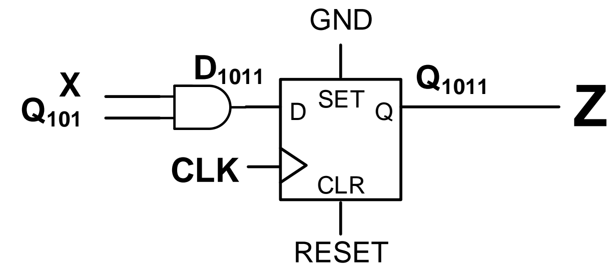
- Be sure to handle overlapping sequences

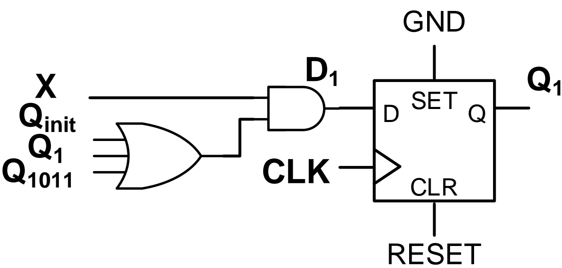


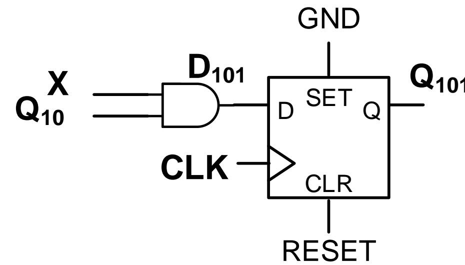
1-Hot Implementation



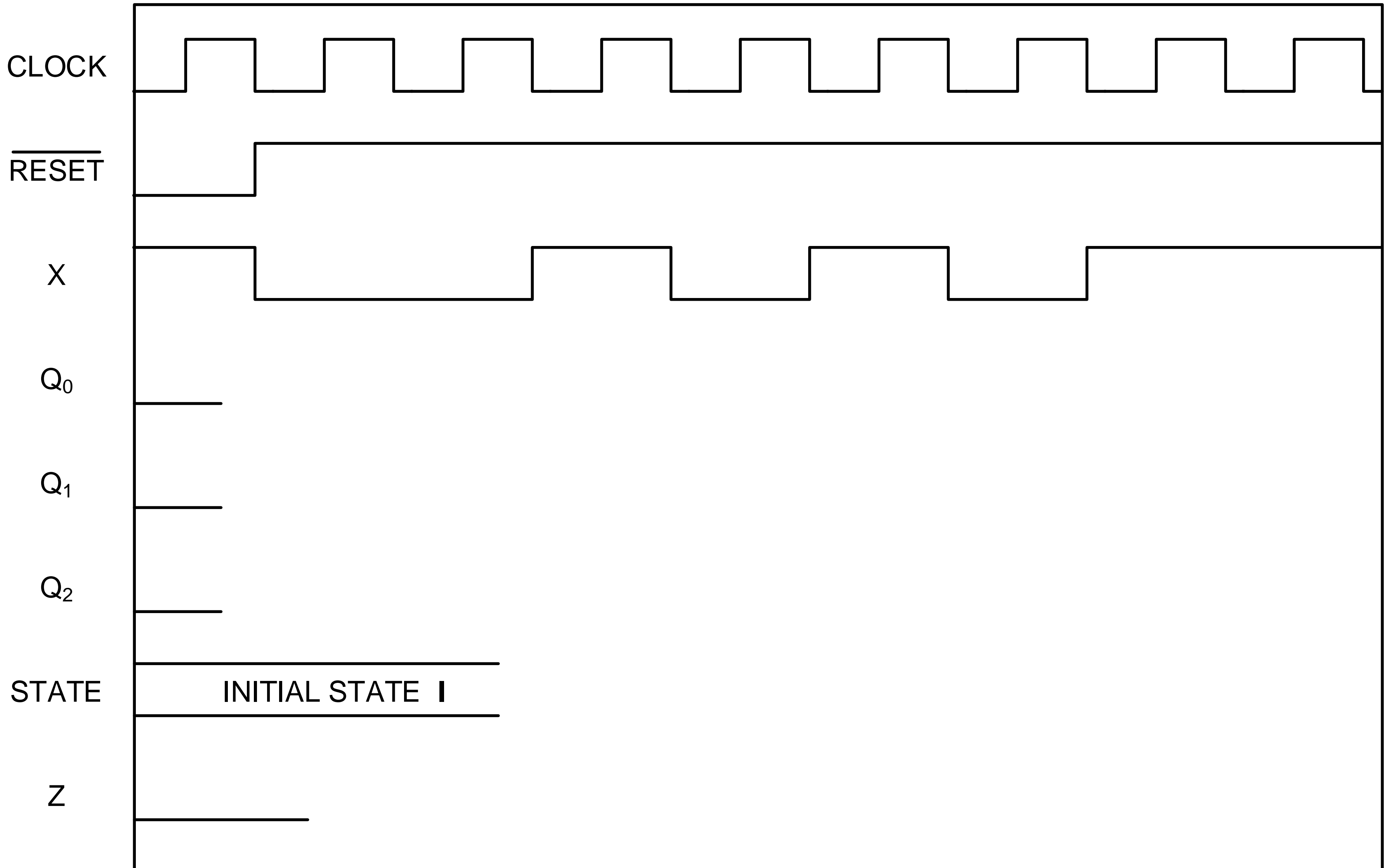








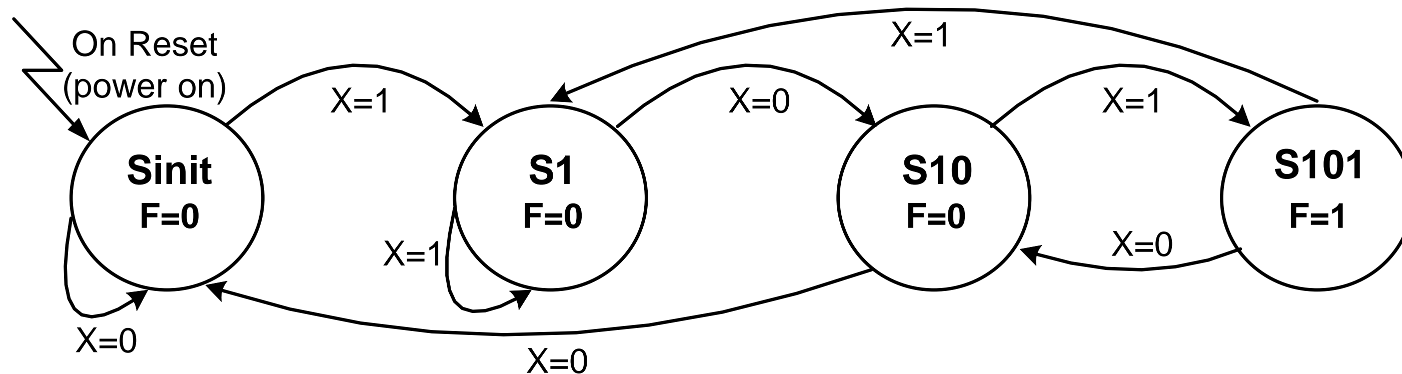
Waveform for 1011 Detector



SELECTED SOLUTIONS

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

Another State Diagram Example

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