



# CSE140: Components and Design Techniques for Digital Systems

## Midterm Information

# Midterm Topics

In general: everything that was covered in homework 1 and 2 and related lectures, such as:

- Transistors
- Boolean algebra
- Basic gates
- Logic functions and truth tables
- Canonical forms (SOP and POS)
- Two-level logic minimization
- Kmaps
- Multiplexers (behavior and how to implement logic functions with them)
- Decoders (behavior and how to implement logic functions with them)

Midterm is in WLH2204 on Thu 08/18

# Midterm Logistics

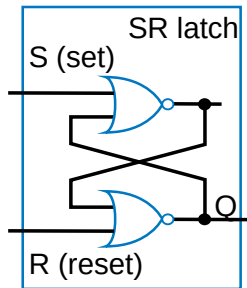
- Do not start the exam until you are told.
- Write your name and PID at the top of every page. Write the names of people on your left and right on the first page.
- Turn off and put away all your electronics.
- This is a **closedbook, closednotes exam**. You may only refer to one 8 ½ x 11” page of your **handwritten notes**.
- If you have a question, raise your hand and an exam proctor will come to you.
- You have **60 minutes** to finish the exam. When the time is finished, you must stop writing.
- Report the solutions in the spaces provided.
- Only the front pages will be graded. We will not consider anything written on the back of the pages.
- **Any student violating UCSD's Academic Dishonesty or UCSD's Student Conduct policies will earn an 'F' in the CSE140 course and will be reported to their college Dean for administrative processing.**
- When returning your exam, also show your student ID.
- Write your answers in the space provided. To get the most partial credit, clearly show all the steps of your work.
- Full credit may not be given for correct answers with no work shown.



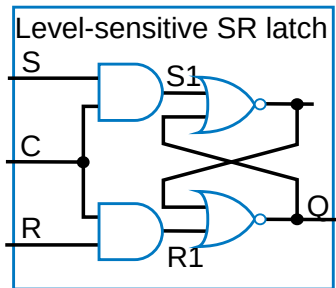
# CSE140: Components and Design Techniques for Digital Systems

## D-Flip-Flops

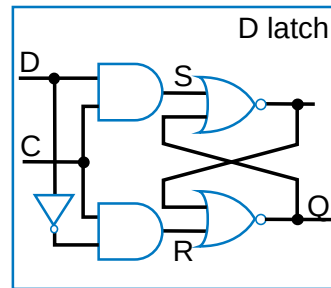
# Bit Storage Overview



S=1 sets Q to 1,  
R=1 resets Q to 0.  
Problem: SR=11  
yield undefined Q.



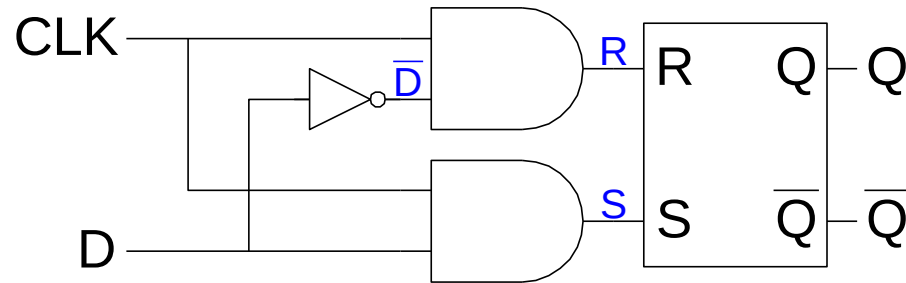
S and R only have effect  
when C=1. We can  
design outside circuit so  
SR=11 never happens  
when C=1. Problem:  
avoiding SR=11 can be a  
burden.



SR can't be 11 if D is  
stable before and while  
C=1, and will be 11 for only  
a brief glitch even if D  
changes while C=1.  
Problem: C=1 too long  
propagates new values  
through too many latches:  
too short may not enable a  
store.



# D Latch Truth Table

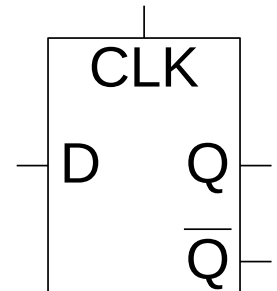


| $CLK$ | $D$ | $\overline{D}$ | $S$ | $R$ | $Q$        | $\overline{Q}$        |
|-------|-----|----------------|-----|-----|------------|-----------------------|
| 0     | X   | $\overline{X}$ | 0   | 0   | $Q_{prev}$ | $\overline{Q}_{prev}$ |
| 1     | 0   | 1              | 0   | 1   | 0          | 1                     |
| 1     | 1   | 0              | 1   | 0   | 1          | 0                     |

# D Latch Summary

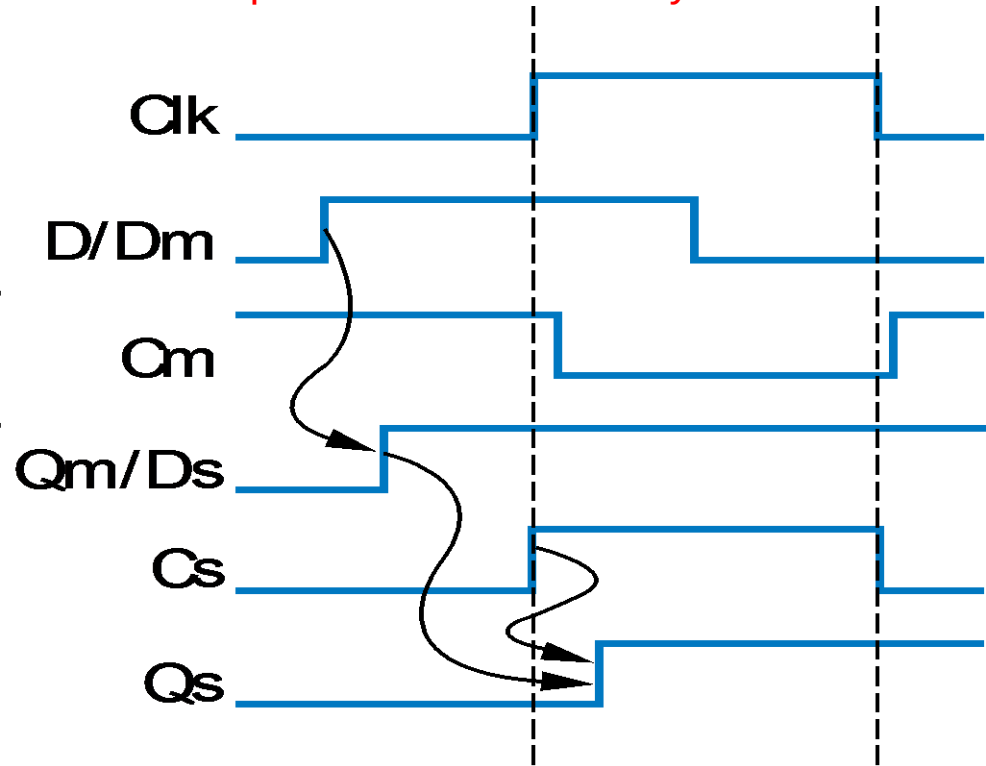
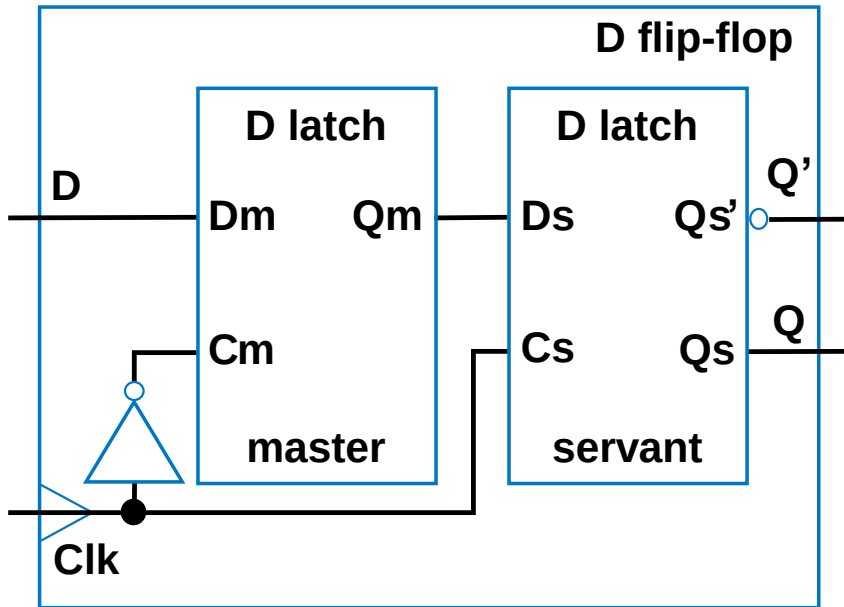
- Two inputs:  $CLK$ ,  $D$ 
  - $CLK$ : controls *when* the output changes
  - $D$  (the data input): controls *what* the output changes to
- Function
  - When  $CLK = 1$ ,  
 $D$  passes through to  $Q$  (*transparent*)
  - When  $CLK = 0$ ,  
 $Q$  holds its previous value (*opaque*)
- (Mostly) avoids invalid case  $Q = Q'$

D Latch  
Symbol



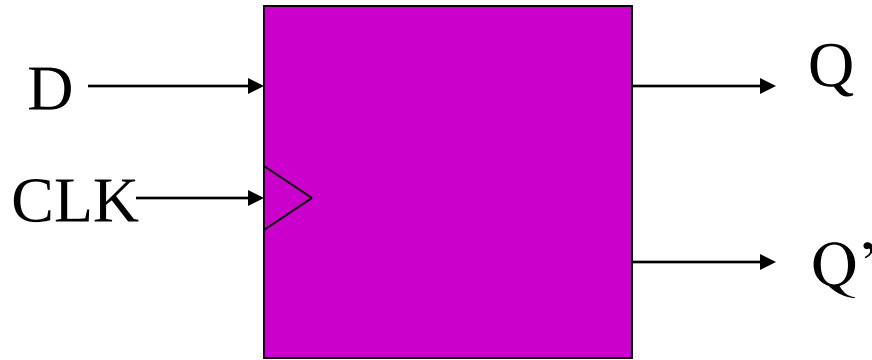
# D Flip-Flop Design & Timing Diagram

Note: this diagram assumes that each component has some delay



- **Flip-flop**: Bit storage that stores on the clock edge, not level
- Master-servant design: master loads when Clk=0, then servant when Clk=1

# D Flip-Flop: Characteristic Equation

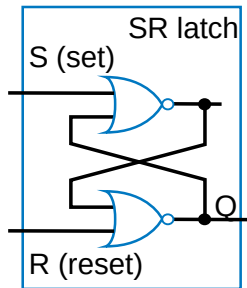


Characteristic Equation  
 $Q(t+1) = D(t)$

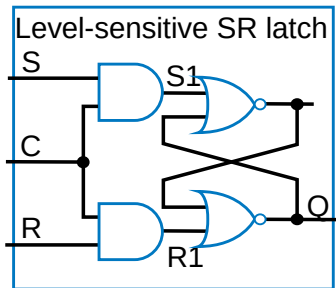
| Id | D | Q(t) | Q(t+1) |
|----|---|------|--------|
| 0  | 0 | 0    | 0      |
| 1  | 0 | 1    | 0      |
| 2  | 1 | 0    | 1      |
| 3  | 1 | 1    | 1      |

Assuming now that the value of D is observed only on the rising edge of the clock

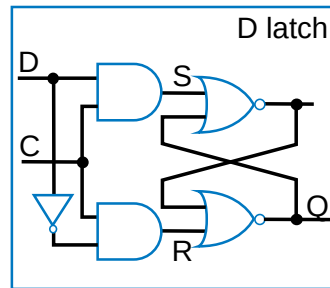
# Bit Storage Overview



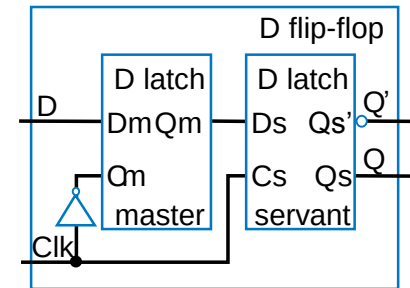
S=1 sets Q to 1,  
R=1 resets Q to 0.  
Problem: SR=11  
yield undefined Q.



S and R only have effect  
when C=1. We can  
design outside circuit so  
SR=11 never happens  
when C=1. Problem:  
avoiding SR=11 can be a  
burden.

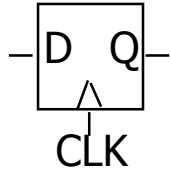


SR can't be 11 if D is  
stable before and while  
C=1, and will be 11 for only  
a brief glitch even if D  
changes while C=1.  
Problem: C=1 too long  
propagates new values  
through too many latches:  
too short may not enable a  
store.

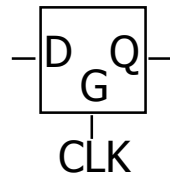


Only loads D value present at  
rising clock edge, so values  
can't propagate to other flip-  
flops during same clock cycle.  
Tradeoff: uses more gates  
internally than D latch, and  
requires more external gates  
than SR – but gate count is  
less of an issue today.

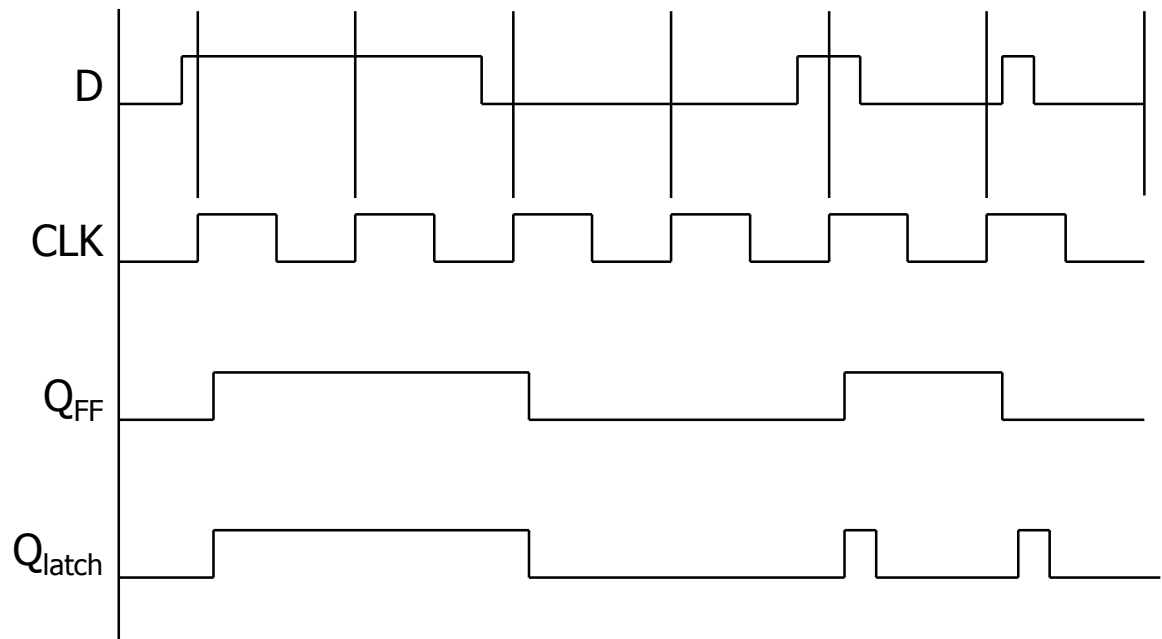
# Comparison of latches and flip-flops



positive  
edge-triggered  
flip-flop

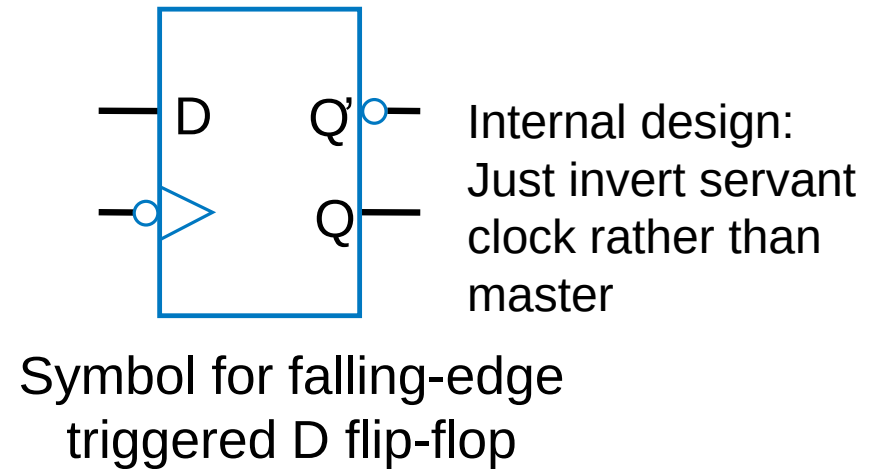
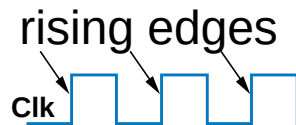
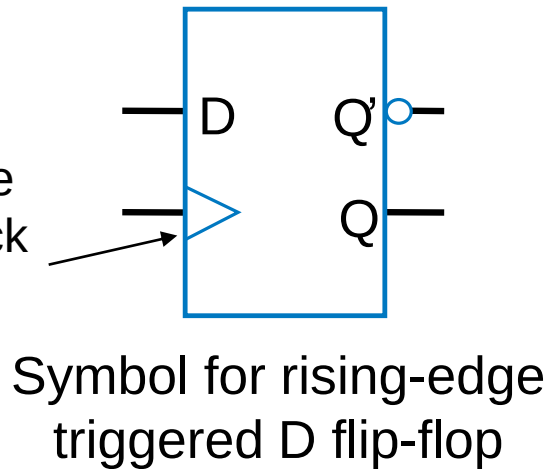


level-sensitive  
latch



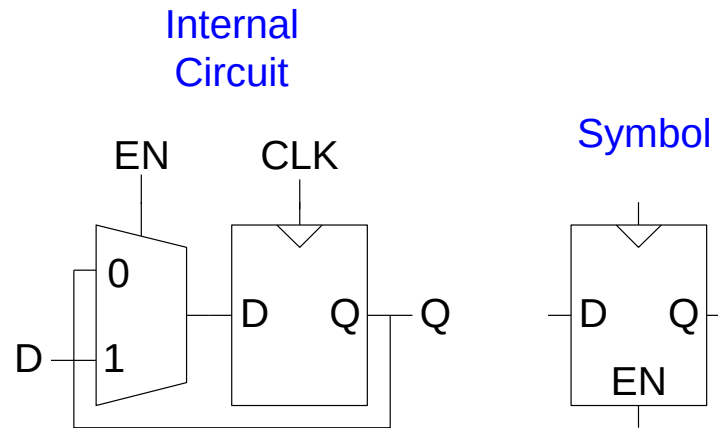
# Rising vs. Falling Edge D Flip-Flop

The triangle means clock input, edge triggered



# Enabled D-FFs

- **Inputs:**  $CLK$ ,  $D$ ,  $EN$ 
  - The enable input ( $EN$ ) controls when new data ( $D$ ) is stored
- **Function**
  - $EN = 1$ :  $D$  passes through to  $Q$  on the clock edge
  - $EN = 0$ : the flip-flop retains its previous state



# Additional D-FF Features

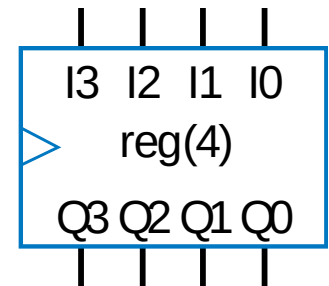
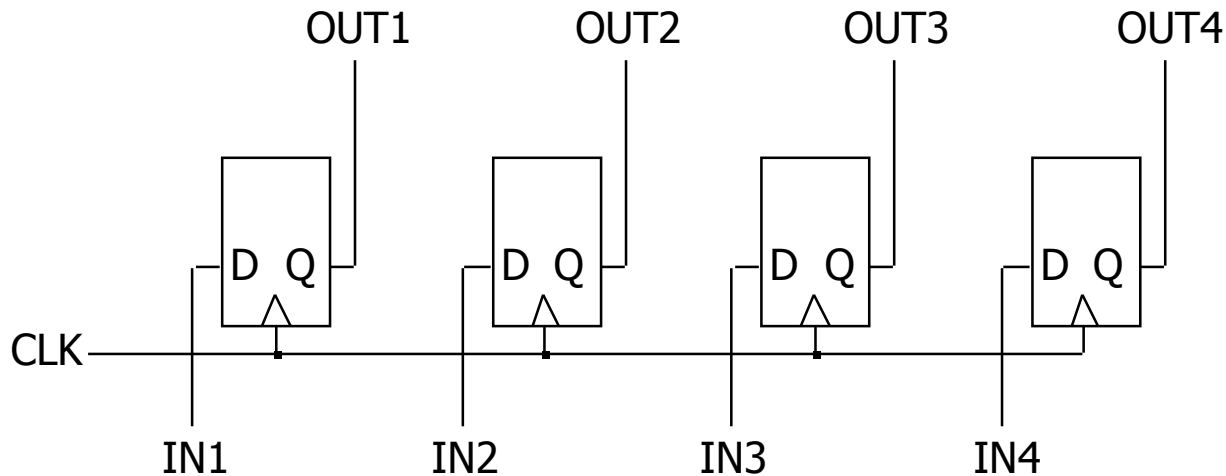
- Reset (set state to 0) – R
  - synchronous:  $D_{\text{new}} = R' \cdot D_{\text{old}}$  (when next clock edge arrives)
  - asynchronous: doesn't wait for clock
- Preset or set (set state to 1) – S (or sometimes P)
  - synchronous:  $D_{\text{new}} = D_{\text{old}} + S$  (when next clock edge arrives)
  - asynchronous: doesn't wait for clock
- Both reset and preset
  - $D_{\text{new}} = R' \cdot D_{\text{old}} + S$  (set-dominant)
  - $D_{\text{new}} = R' \cdot D_{\text{old}} + R'S$  (reset-dominant)
- Selective input capability (input enable or load) – LD or EN
  - multiplexor at input:  $D_{\text{new}} = LD' \cdot Q + LD \cdot D_{\text{old}}$
  - load may or may not override reset/set (usually R/S have priority)



# Registers and Counters

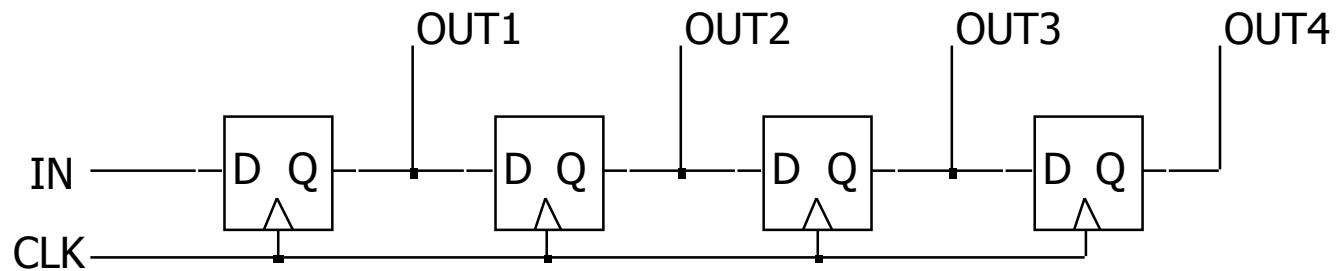
# Building blocks with FFs: Basic Register

- Register: a sequential component that can store multiple bits
- A basic register can be built simply by using multiple D-FFs



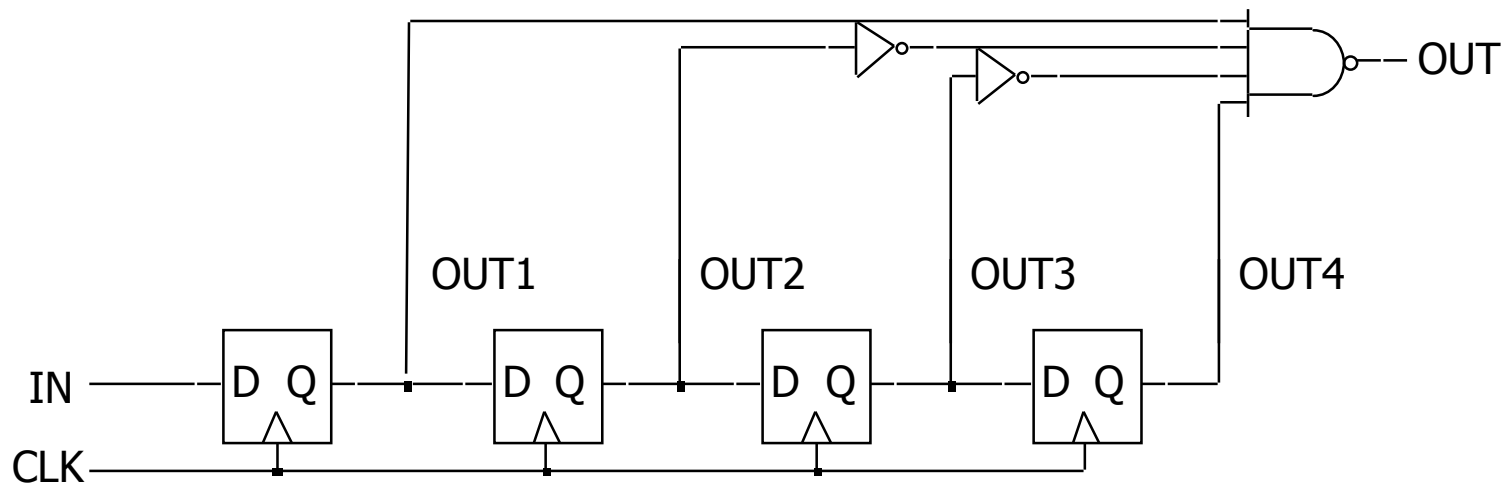
# Shift register

- Holds & shifts samples of input



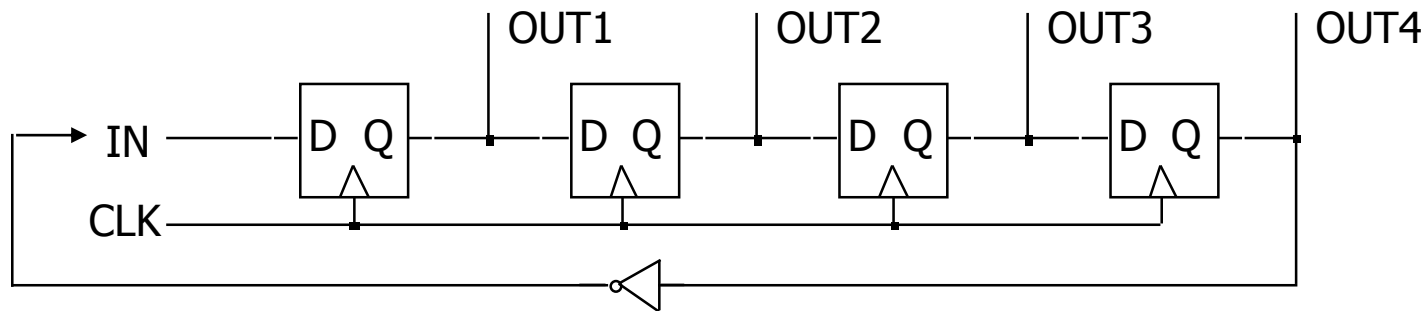
# Pattern Recognizer

- Combinational function of input samples



# Counters

- Sequences through a fixed set of patterns
- Note: definition is general
- For example, the one in the figure is a type of counter called Linear Feedback Shift Register (LFSR)



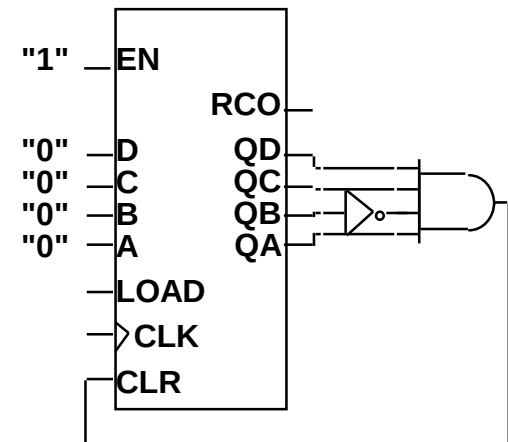
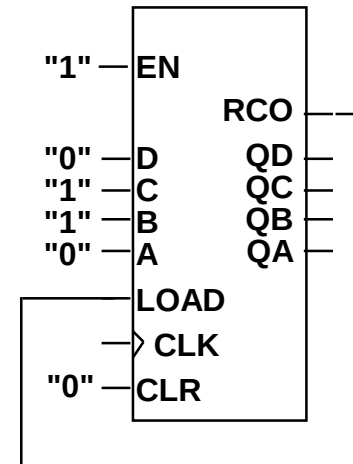
# General Counters

- **Default operation: count up**

- QA-QD counter output
- A-D parallel load data
- LOAD enables data load
- RCO ripple carry out
- CLR clears data
- EN counter enable

Similar signals can also be used for registers

You can generalize these ideas to build multifunction registers/shifters/counters with a variety of control signals (load, clear, enable, etc.)

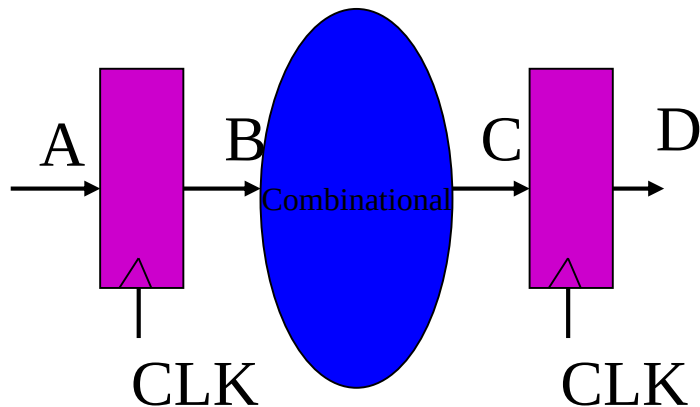




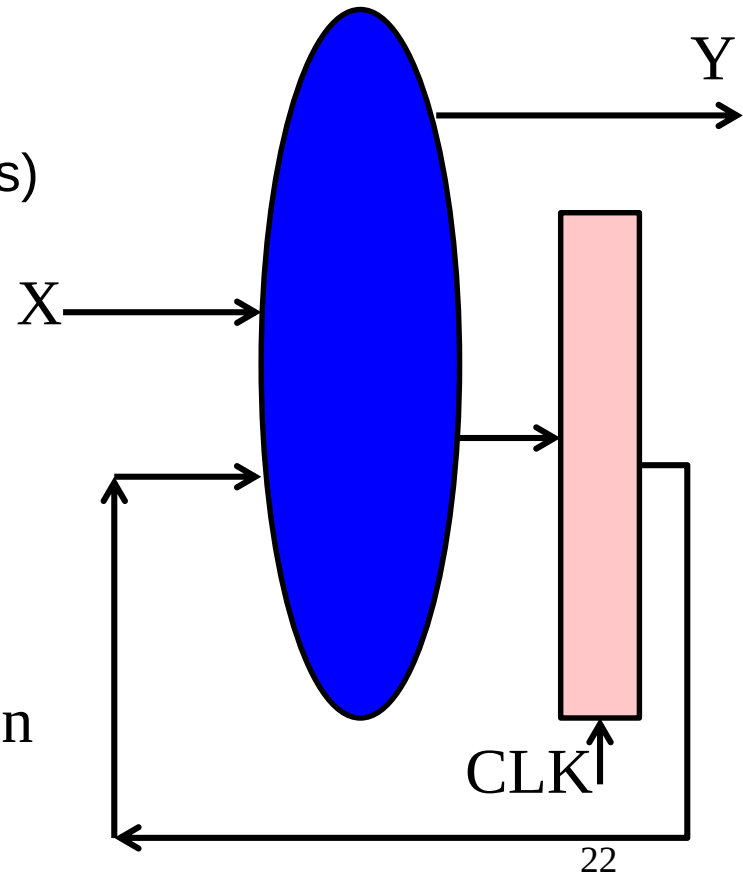
# Finite State Machines

# Circuit Specifications

- Combinational Logic
  - Truth tables, Boolean equations, logic diagrams (no feedback)
- Sequential Networks: **State Diagram (Memory)**
  - State and Excitation Tables
  - Characteristic Expression
  - Logic Diagram (FFs and feedback loops)

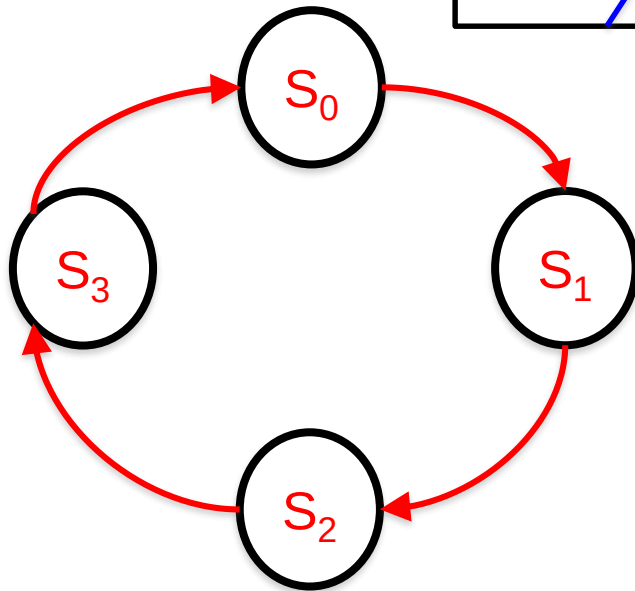
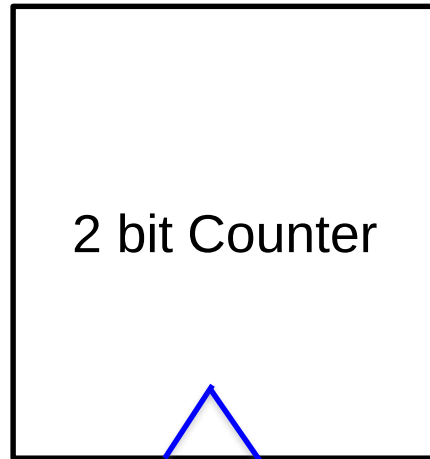


RTL: Register-Transfer Level Description



# Finite State Machines: Two Bit Counter Example

Symbol/ Circuit



State Diagram

| Current state | Next State |
|---------------|------------|
| $S_0$         | $S_1$      |
| $S_1$         | $S_2$      |
| $S_2$         | $S_3$      |
| $S_3$         | $S_0$      |

| $Q_1(t)$ | $Q_0(t)$ | $Q_1(t+1)$ | $Q_0(t+1)$ |
|----------|----------|------------|------------|
| 0        | 0        | 0          | 1          |
| 0        | 1        | 1          | 0          |
| 1        | 0        | 1          | 1          |
| 1        | 1        | 0          | 0          |

State Table

# Which is the most likely circuit realization of the two bit counter?

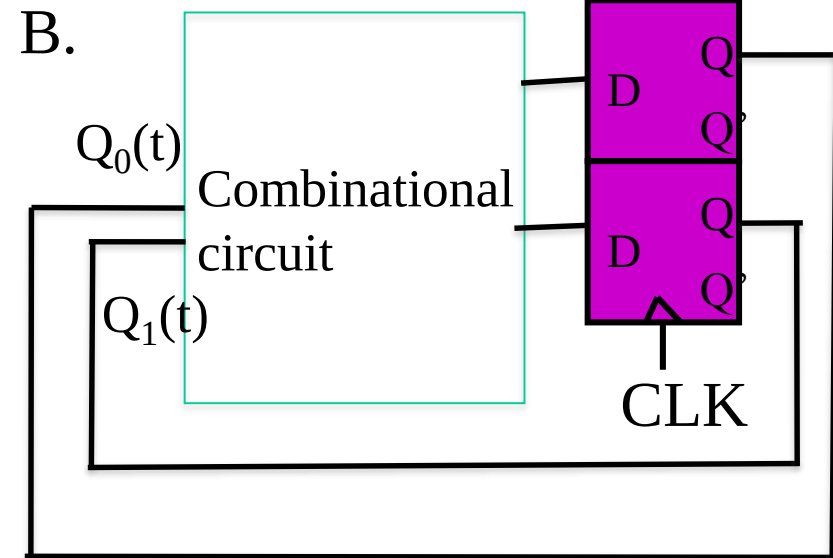
State Table

| $Q_1(t)$ | $Q_0(t)$ | $Q_1(t+1)$ | $Q_0(t+1)$ |
|----------|----------|------------|------------|
| 0        | 0        | 0          | 1          |
| 0        | 1        | 1          | 0          |
| 1        | 0        | 1          | 1          |
| 1        | 1        | 0          | 0          |

A.

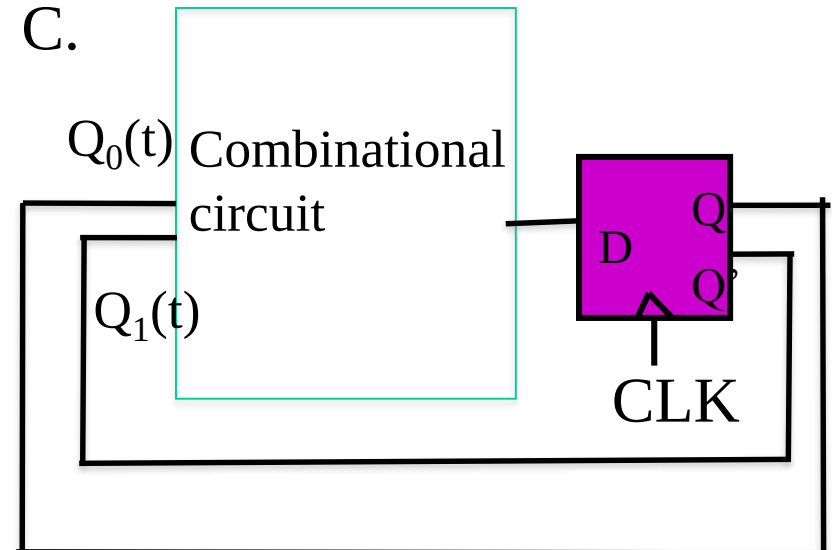


Circuit with no flip flops



Circuit with 2 flip flops

C.

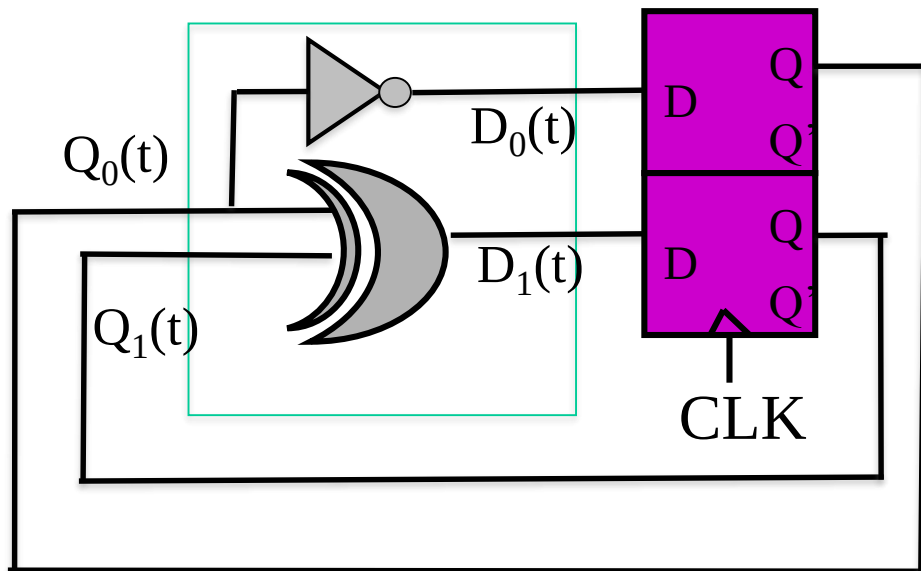


Circuit with one flip flop

# Two Bit Counter Circuit

State Table

| $Q_1(t)$ | $Q_0(t)$ | $Q_1(t+1)$ | $Q_0(t+1)$ |
|----------|----------|------------|------------|
| 0        | 0        | 0          | 1          |
| 0        | 1        | 1          | 0          |
| 1        | 0        | 1          | 1          |
| 1        | 1        | 0          | 0          |



We store the current state using D-flip flops so that:

- Inputs to the combinational circuit don't change while the next output is computed
- The transition to the next state only occurs at the rising edge of the clock

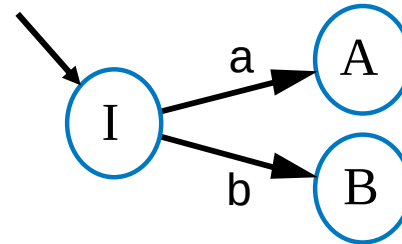
$$D_0(t) = Q_0(t)'$$

$$D_1(t) = Q_0(t) Q_1(t)' + Q_0(t)' Q_1(t)$$

Implementation of 2-bit counter

# FSM Definition

- FSM consists of
  - Set of states
  - Set of inputs, set of outputs
  - Initial state
  - Set of transitions
    - Only one can be true at a time
- FSM representations:
  - State diagram
  - State table

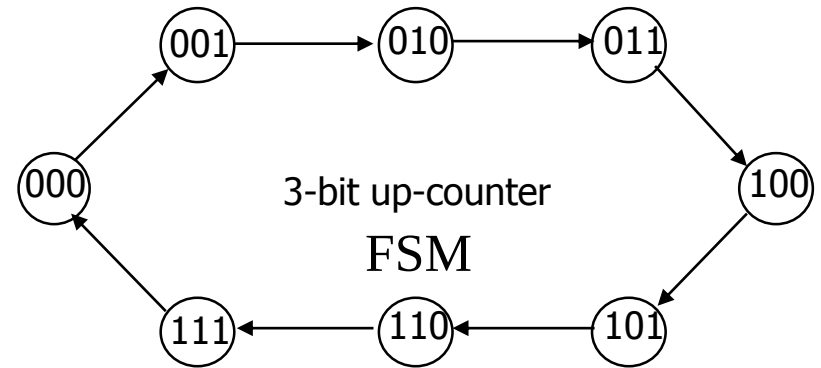


# FSM Controller Design Process with a Three Bit Counter Example

1. State Diagram
2. State Table
3. State Assignments
4. Excitation Table

(present state, inputs; next state, outputs)

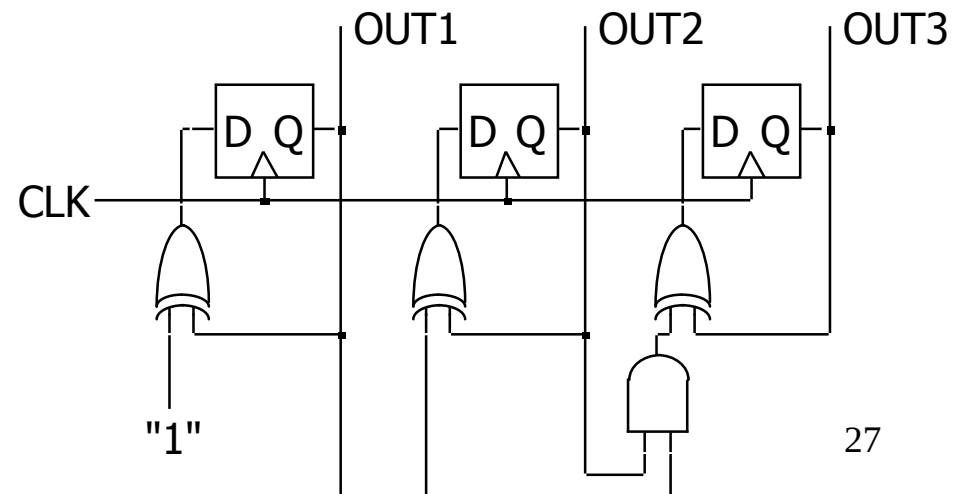
5. Circuit



State Table with Assigned State Patterns

| C3 | C2 | C1 | N3 | N2 | N1 |
|----|----|----|----|----|----|
| 0  | 0  | 0  | 0  | 0  | 1  |
| 0  | 0  | 1  | 0  | 1  | 0  |
| 0  | 1  | 0  | 0  | 1  | 1  |
| 0  | 1  | 1  | 1  | 0  | 0  |
| 1  | 0  | 0  | 1  | 0  | 1  |
| 1  | 0  | 1  | 1  | 1  | 0  |
| 1  | 1  | 0  | 1  | 1  | 1  |
| 1  | 1  | 1  | 0  | 0  | 0  |

Circuit

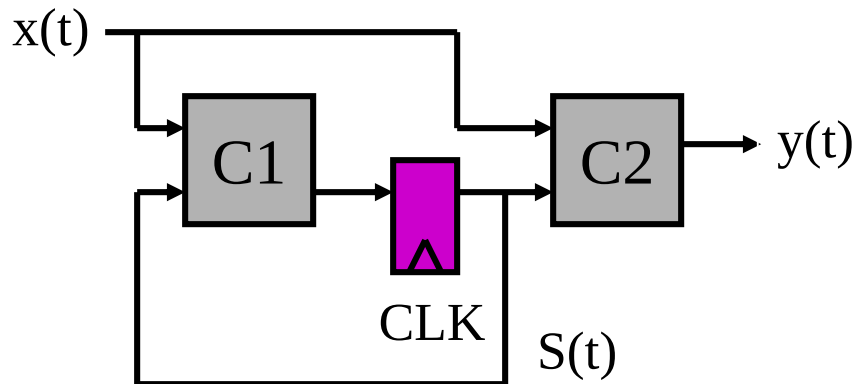


# Mealy and Moore Machines

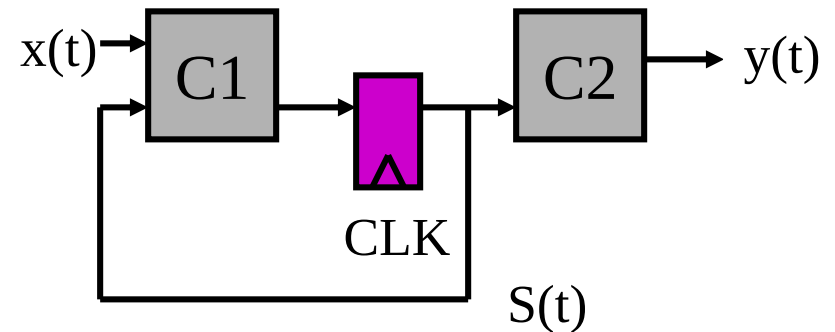
Mealy Machine:  $y_i(t) = f_i(X(t), S(t))$

Moore Machine:  $y_i(t) = f_i(S(t))$

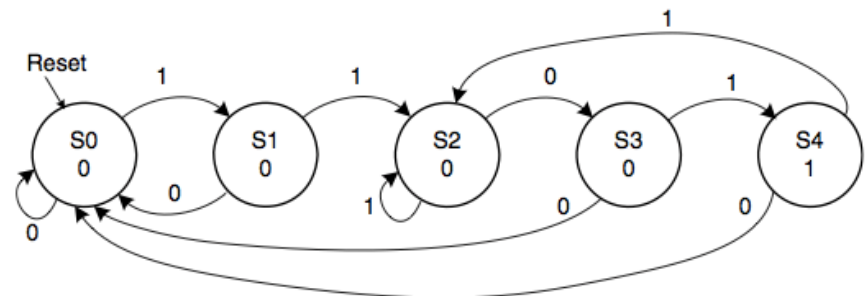
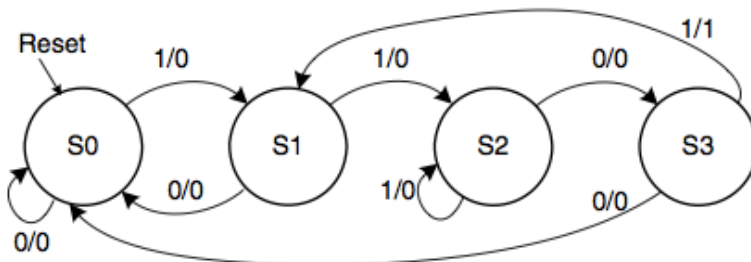
$s_i(t+1) = g_i(X(t), S(t))$



Mealy Machine

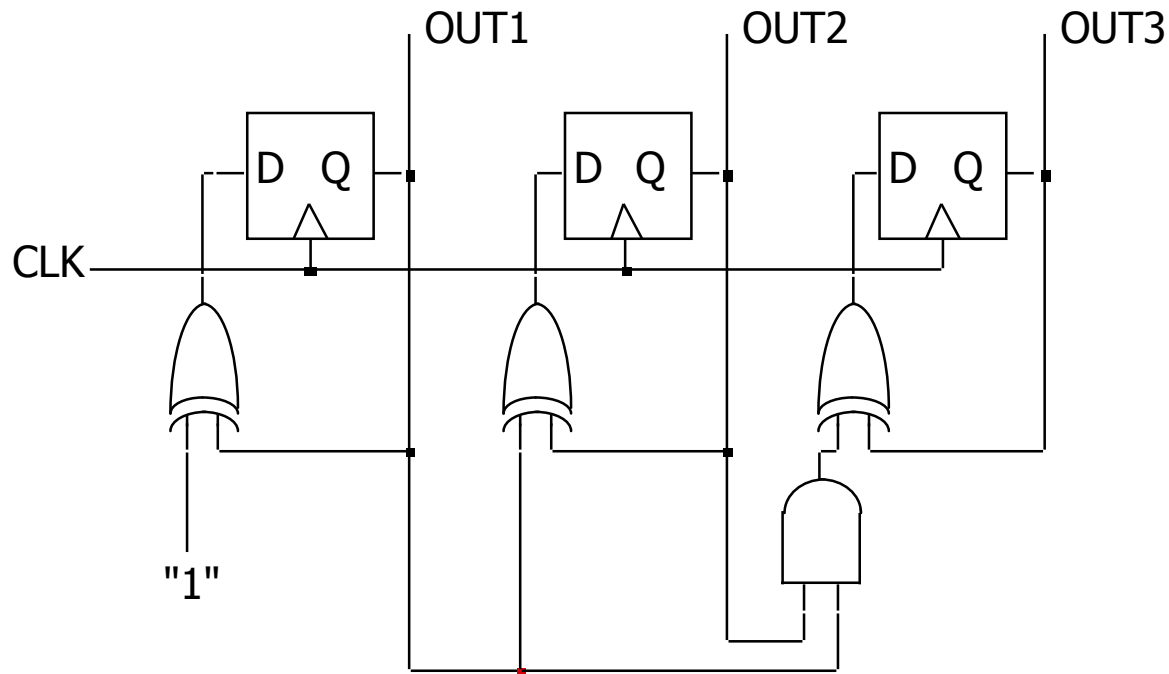


Moore Machine



# This Counter Design Is:

- A. Moore machine
- B. Mealy machine
- C. None of the above



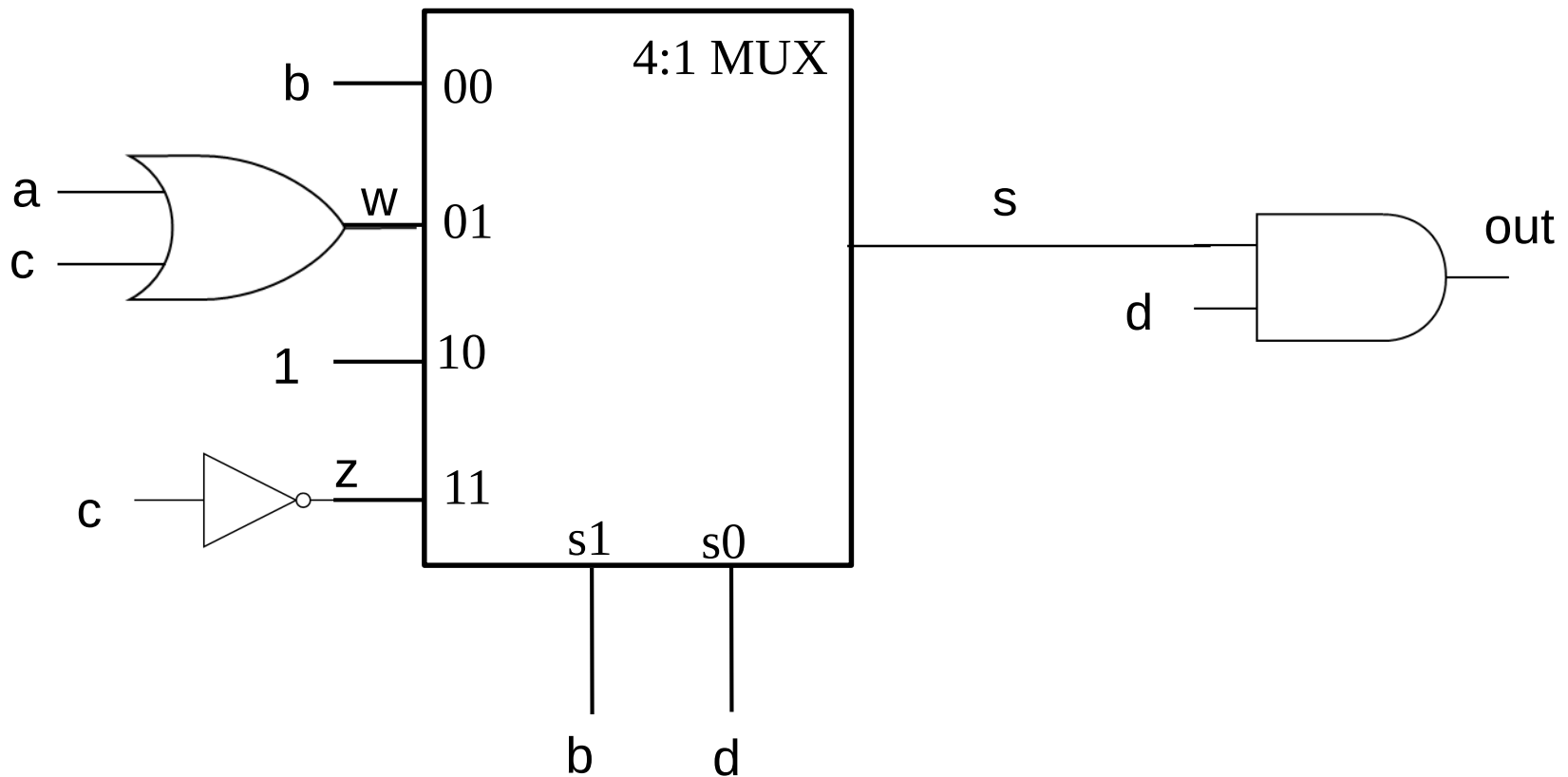


# CSE140: Components and Design Techniques for Digital Systems

## More Examples for the Midterm

# Example 1: Multiplexers

- Derive each intermediate signal as a function of a, b, c and d



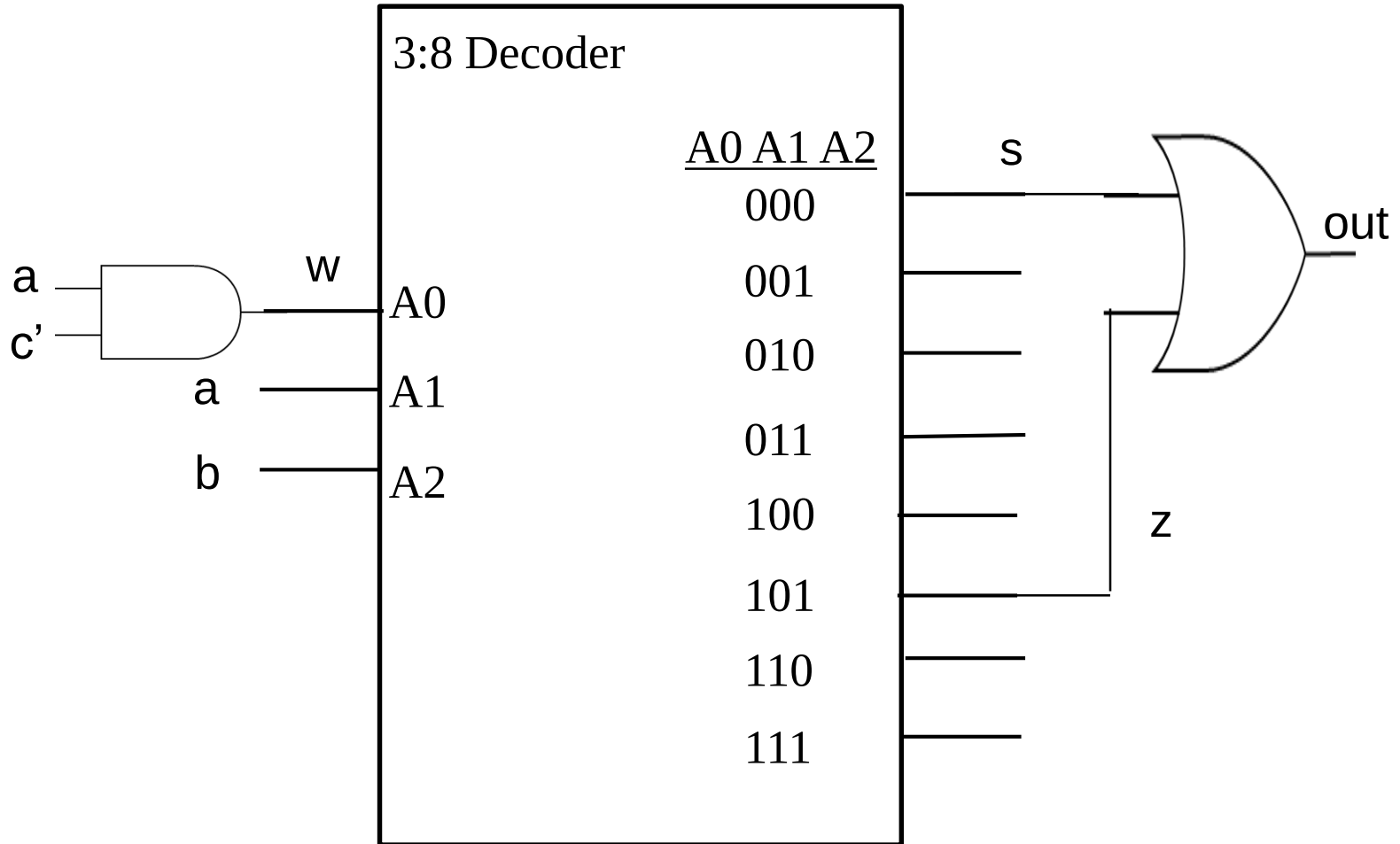
## Example 2: Multiplexers

- Implement the following function using a 4:1 MUX

$$F(a,b,c) = a'b'c + a(bc + bc')$$

## Example 3: Decoders

- Derive each intermediate signal as a function of  $a$ ,  $b$ , and  $c$



## Example 4: Decoders

- Implement the following function using a 3:8 Decoder

$$F(a,b,c) = abc + a(b'c + bc')$$