EE1D1: Digital Systems A

BSc. EE, year 1, 2025-2026, lecture 6

Sequential Logic

Computer Engineering Lab

Faculty of Electrical Engineering, Mathematics & Computer Science

TU Delft

Delft University of Technology

Learning Objectives

As student you should be able to:

- Explain how sequential networks work
- Interpret circuits with memory elements (i.e., latches, flip-flops)
- Explain Finite State Machines
- Design Finite State Machines (Moore and Mealy type)

Outline

EE1D1: Digital Systems A

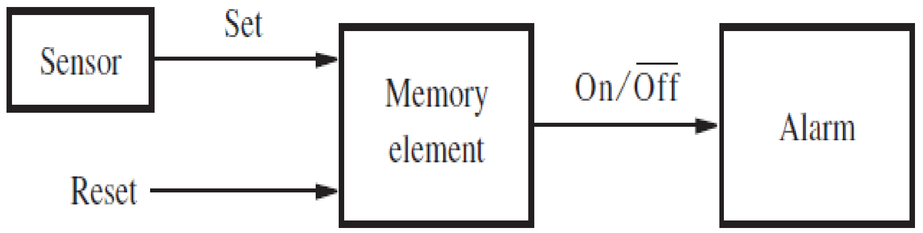
Sequential Networks

Sequential Networks – Overview

- In previous lectures we considered **combinational circuits:**
  - The value of each output depends solely on the inputs.

- Here we introduce **sequential circuits:**
  - The outputs depend on the inputs and on the past behavior of the circuit.
  - Such circuits include storage elements that store the values of logic signals.
  - The contents of the storage elements are said to represent the *state* of the circuit.

Motivation

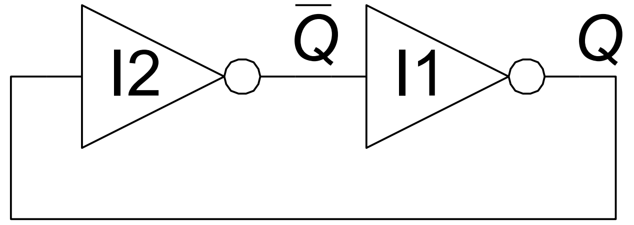


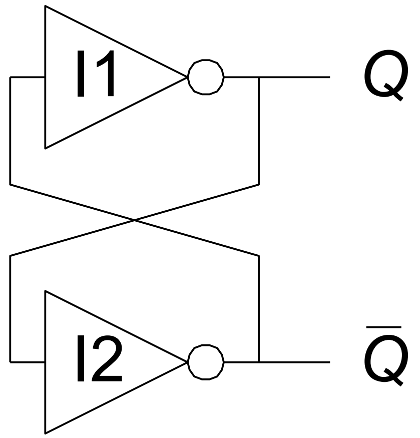
- Once the alarm is triggered by the sensor, it must remain active even if the sensor output goes back to zero.
- The alarm is turned off manually by means of a *Reset* input.
- The circuit requires a memory element to remember that the alarm has to be active until the *Reset* signal arrives.

Sequential Networks –Bistable Circuit

- Fundamental building block of other state elements
- Two outputs: Q, $\overline{Q}$
- No inputs

Same circuit!

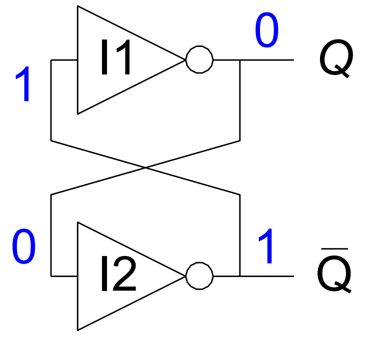




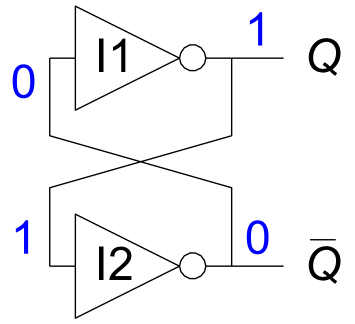
Back-to-back inverters **Cross-coupled inverters**

Bistable Circuit Analysis

- Consider the two possible cases:
  - $Q = 0$:
    then $\overline{Q} = 1$, $Q = 0$ (consistent)
  - $Q = 1$:
    then $\overline{Q} = 0$, $Q = 1$ (consistent)

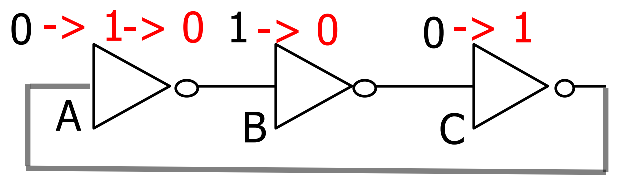


Stable



Stable

What happens with an odd number of inverters?

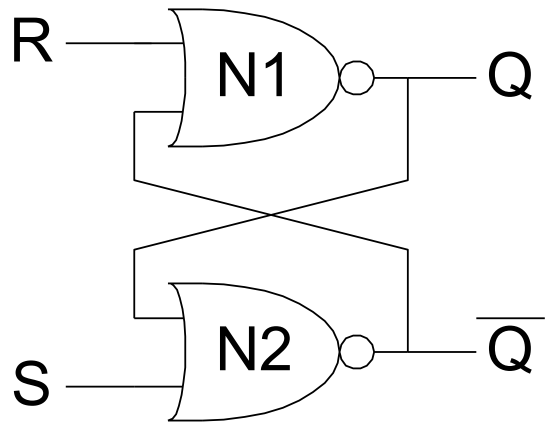


oscillation!

- Stores 1 bit of state in the state variable, Q (or $\overline{Q}$)

Sequential Networks – SR Latch Analysis

- SR Latch



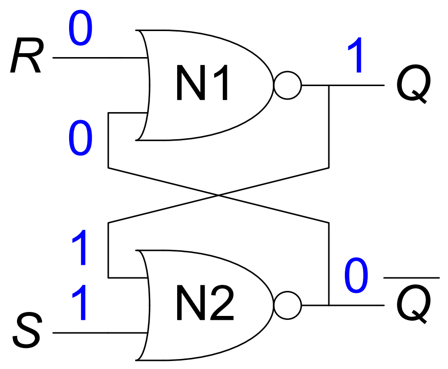
- Consider the four possible cases:
  - $S = 1, R = 0$
  - $S = 0, R = 1$
  - $S = 0, R = 0$
  - $S = 1, R = 1$

Sequential Networks – R-S Latch Analysis

– $S = 1$, $R = 0$:

then $\mathbf{Q = 1}$ and $\overline{Q} = 0$

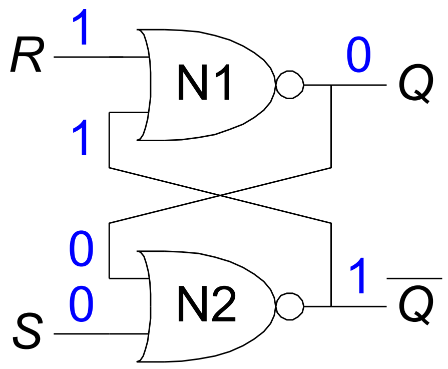
***Set* the output**



– $S = 0$, $R = 1$:

then $\mathbf{Q = 0}$ and $\overline{Q} = 1$

***Reset* the output**

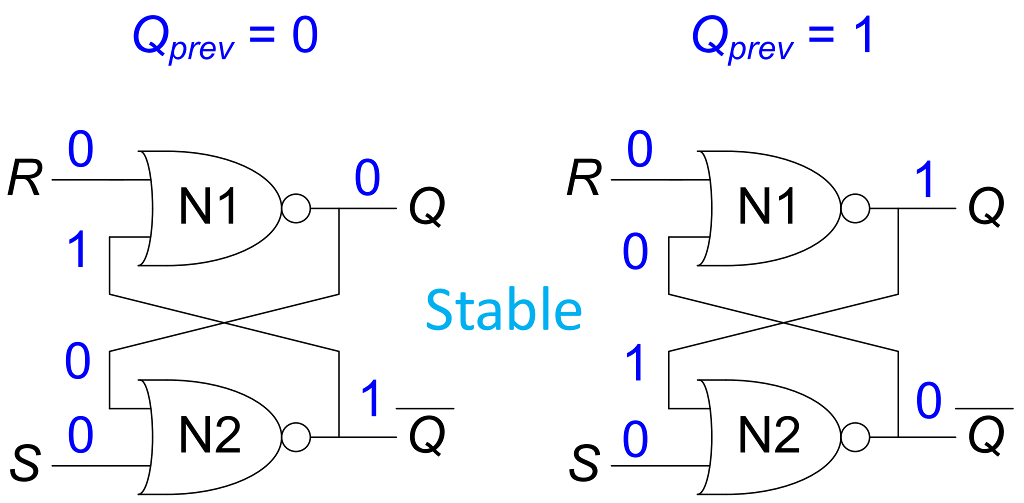


Sequential Networks – R-S Latch Analysis

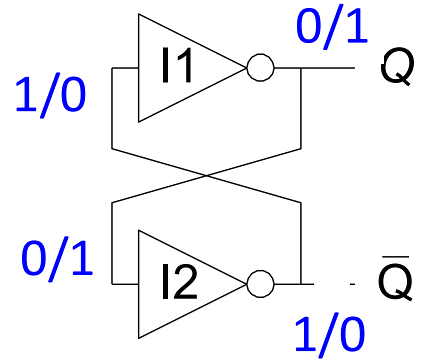
– $S = 0, R = 0$:

then $Q = Q_{prev}$

Memory!



similar to:

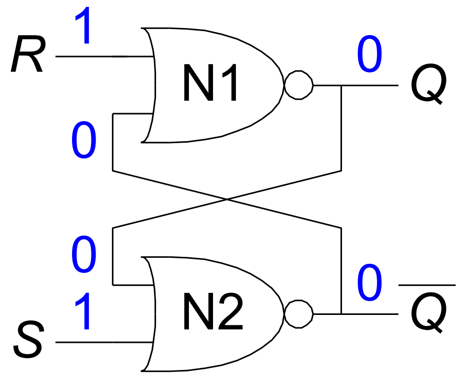


– $S = 1, R = 1$:

then $Q = 0$, $\overline{Q} = 0$

Invalid State

$\overline{Q} \neq$ NOT Q



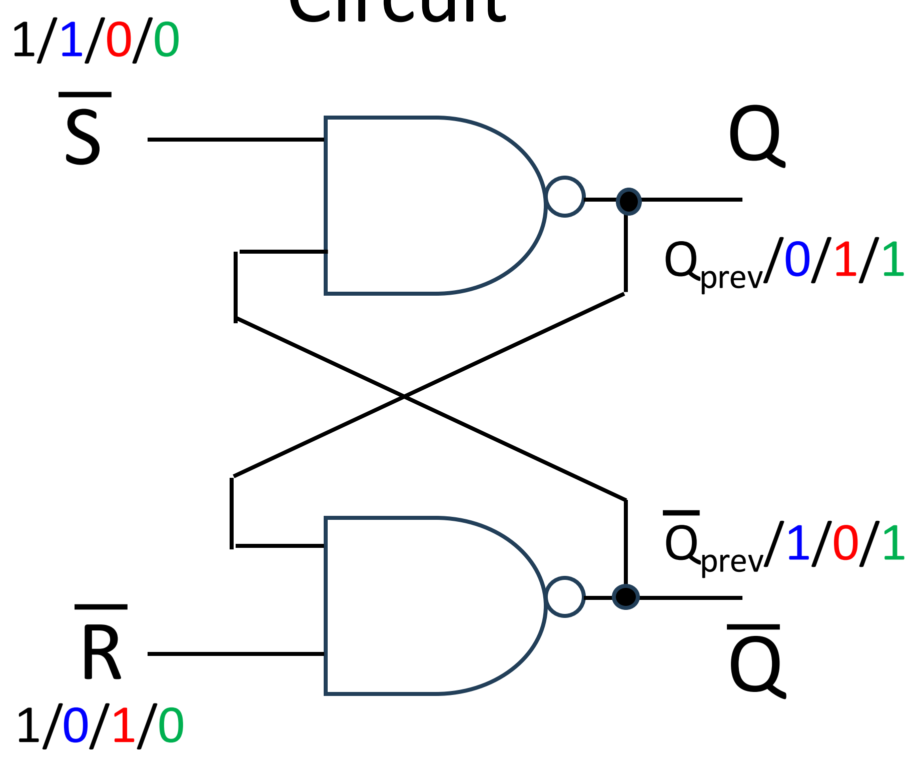
Truth Table

S	R	Q	$\overline{Q}$
0	0	Q_{prev}	$\overline{Q}_{prev}$
0	1	0	1
1	0	1	0
1	1	0	0

invalid/ forbidden

NAND SR Latch

Circuit



Truth Table

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
1	1	Q_{prev}	$\overline{Q}_{prev}$
1	0	0	1
0	1	1	0
0	0	1	1

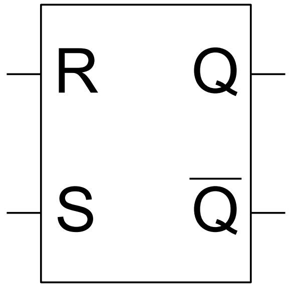
$\Rightarrow$

S	R	Q	$\overline{Q}$	
0	0	Q_{prev}	$\overline{Q}_{prev}$	hold
0	1	0	1	reset
1	0	1	0	set
1	1	1	1	invalid

Sequential Networks – SR (NOR) Latch Analysis

- SR stands for Set/Reset Latch
  - Stores one bit of state (Q)
- Control what value is being stored with S, R inputs
  - Set: Make the output 1
    $S = 1$, $R = 0$, $Q = 1$
  - Reset: Make the output 0
    $S = 0$, $R = 1$, $Q = 0$
  - Memory: Retain value
    $S = 0$, $R = 0$, $Q = Q_{prev}$
- **Must do something to avoid invalid state (when $S = R = 1$)**

SR Latch Symbol

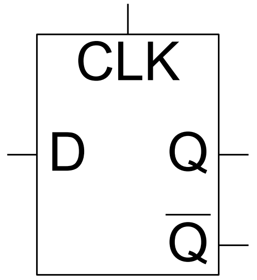


Sequential Networks – D Latch

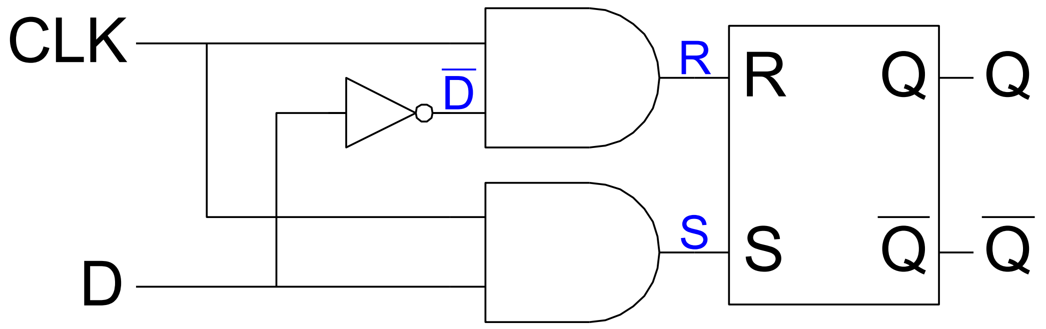
- Two inputs: *CLK*, *D*
  - *CLK*: controls *when* the output changes
  - *D* (the data input): controls *what* the output changes to
- Function
  - When *CLK* = 1,
    D passes through to *Q* (*transparent*)
  - When *CLK* = 0,
    Q holds its previous value (*memory*)
- Avoids invalid case when

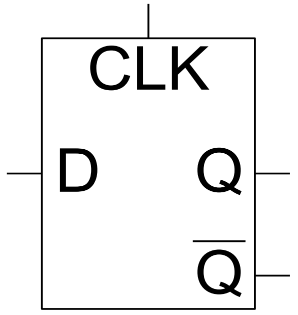
  $Q \neq \text{NOT } \overline{Q}$

D Latch Symbol



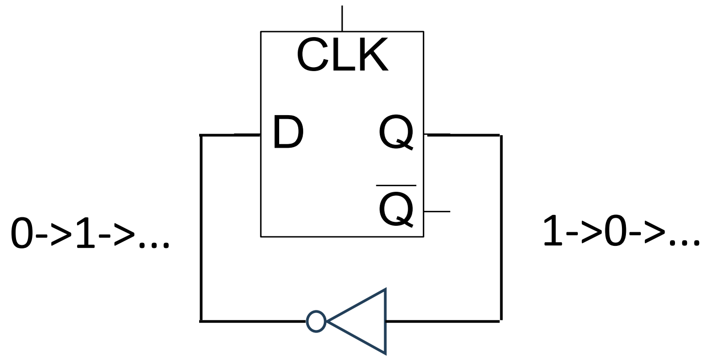
Sequential Networks – D Latch Internal Circuit





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

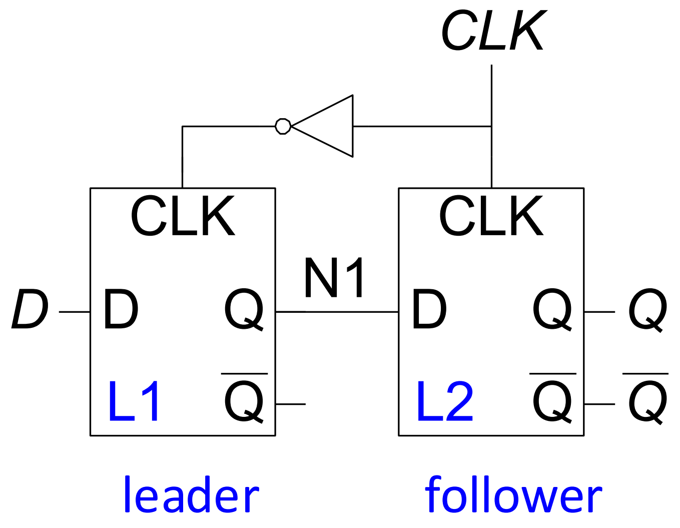
There is still a limitation:



Level-sensitive latches are transparent when clk is active. This gives problems when feedback is used.

Sequential Networks – D Flip-Flop Internal Circuit

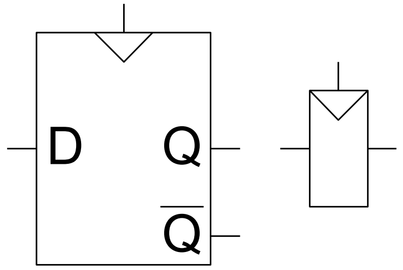
- **Two back-to-back D latches** (L1 and L2) controlled by complementary clocks
- When *CLK* = 0
  - L1 is transparent
  - L2 doesn't change
  - *D* passes through to **N1**
- When *CLK* = 1
  - L2 is transparent
  - L1 doesn't change
  - **N1** passes through to *Q*
- Thus, on the edge of the clock (when *CLK* rises from 0 → 1)
  - *D* passes through to *Q*
- Hence, flip-flop is never transparent as a whole.



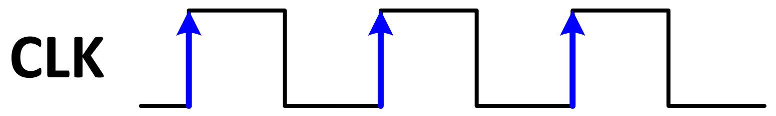
Sequential Networks – D Flip-Flop

- Inputs: *CLK*, *D*
- Function:
  - Samples *D* on **rising edge of *CLK***
    - When *CLK* rises from 0 to 1, *D* passes through to *Q*
    - Otherwise, *Q* holds its previous value
  - ***Q* changes** only on rising edge of *CLK*
- Called ***edge-triggered***
  - Activated on the *clock edge*

D Flip-Flop Symbols



A clock that updates all memory elements (flip-flops) at exactly the same moment (the active clock edge) allows us to design circuits that compute a next state from a current state in a well-defined manner!

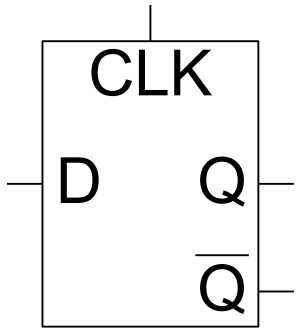


Clock edges

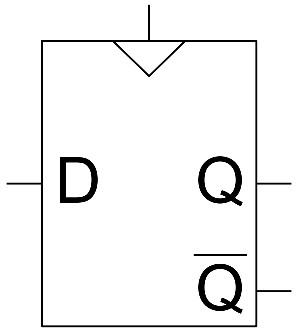
positive edge triggered

Can we also build a negative edge triggered D Flip-Flop?

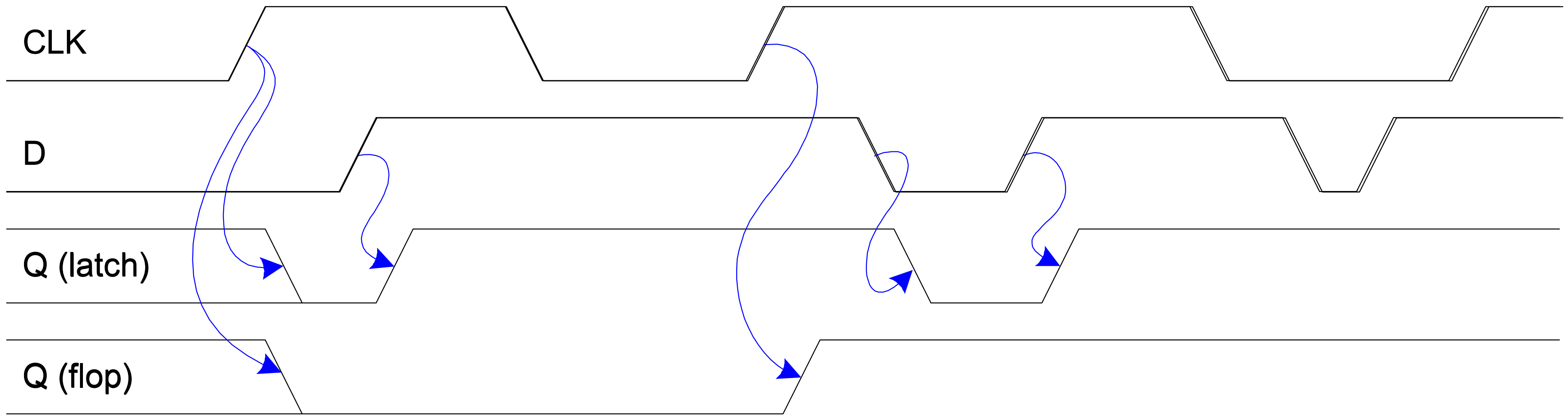
Sequential Networks – D Latch vs. D Flip-Flop



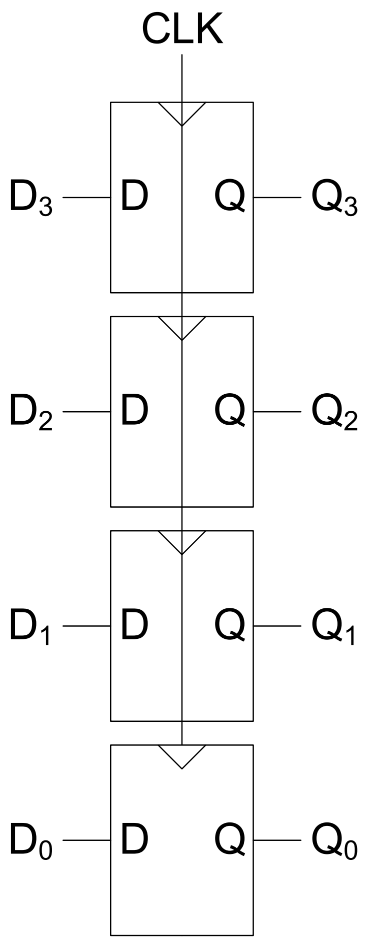
D Latch



D Flip-flop

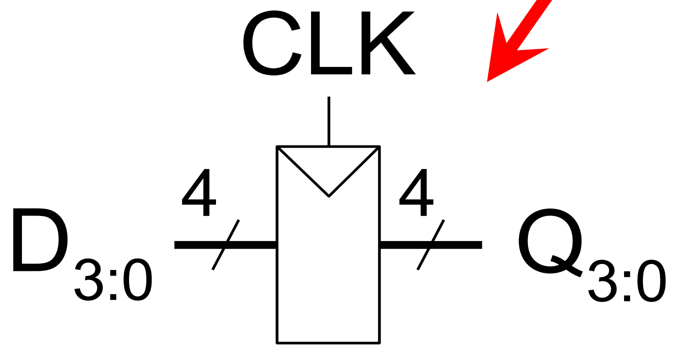


Registers: One or More Flip-flops



4-bit Register

Easier to draw!



4-bit Register

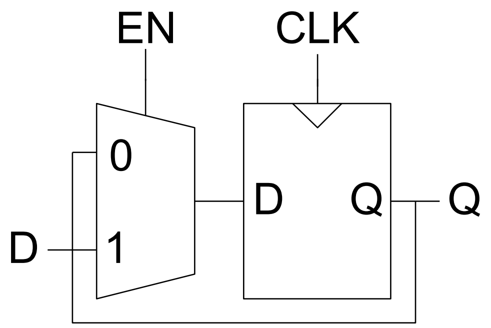
Two ways to draw a register

Enabled Flip-Flops

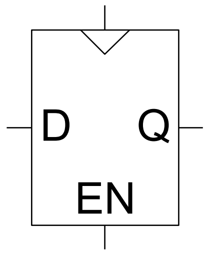
- Inputs: *CLK, D, EN*
  - The enable input (*EN*) controls when new data (*D*) is stored
- Function
  - *EN* = 1: *D* passes through to *Q* on the clock edge
  - *EN* = 0: the flip-flop retains its previous state

It's useful when we wish to load a new value into a flip-flop only sometimes, rather than on every clock edge

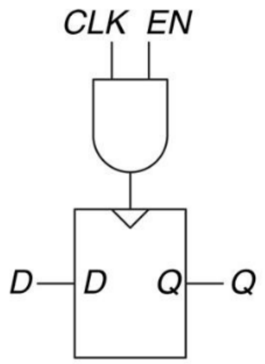
Internal Circuit



Symbol



Another way to implement it?



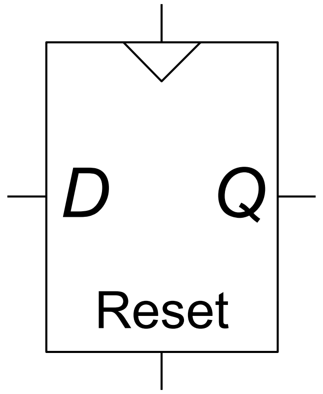
No, never put logic in the clock. Spikes may occur and cause unwanted state changes.

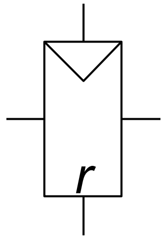
Resettable Flip-Flops

- Inputs: *CLK*, *D*, *Reset*
- Function:
  - *Reset* = 1: *Q* is forced to 0
  - *Reset* = 0: flip-flop behaves as ordinary D flip-flop

It's useful when we want to force a known state (i.e., 0) into all the flop-flops in a system when we first turn it ON.

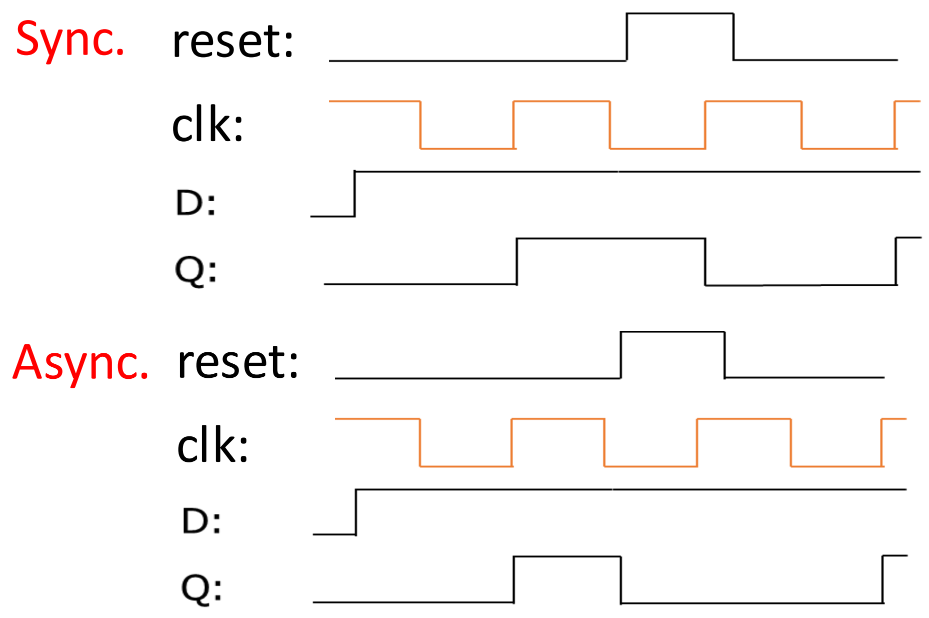
Symbols

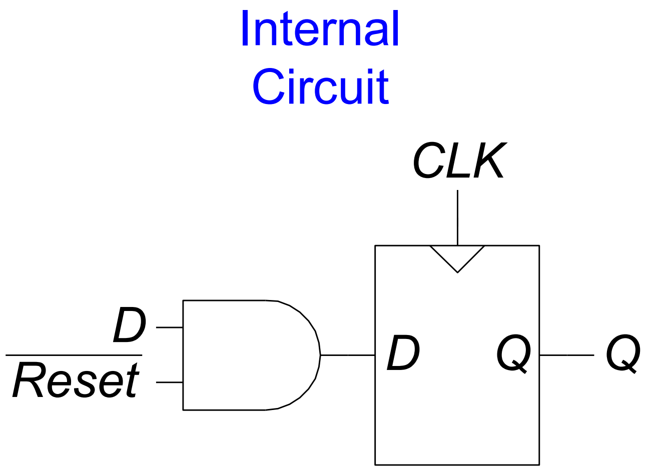




Resettable Flip-Flops

- Two types:
  - Synchronous: resets at the clock edge only
  - Asynchronous: resets immediately when *Reset* = 1
- **Asynchronously** resettable flip-flop requires changing the internal circuitry of the flip-flop
- **Synchronously** resettable flip-flop:

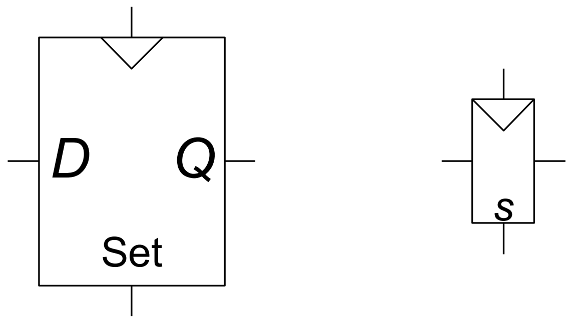




Settable Flip-Flops

- Inputs: *CLK*, *D*, *Set*
- Function:
  - *Set* = 1: *Q* is set to 1
  - *Set* = 0: the flip-flop behaves as ordinary D flip-flop

Symbols



EE1D1: Digital Systems A

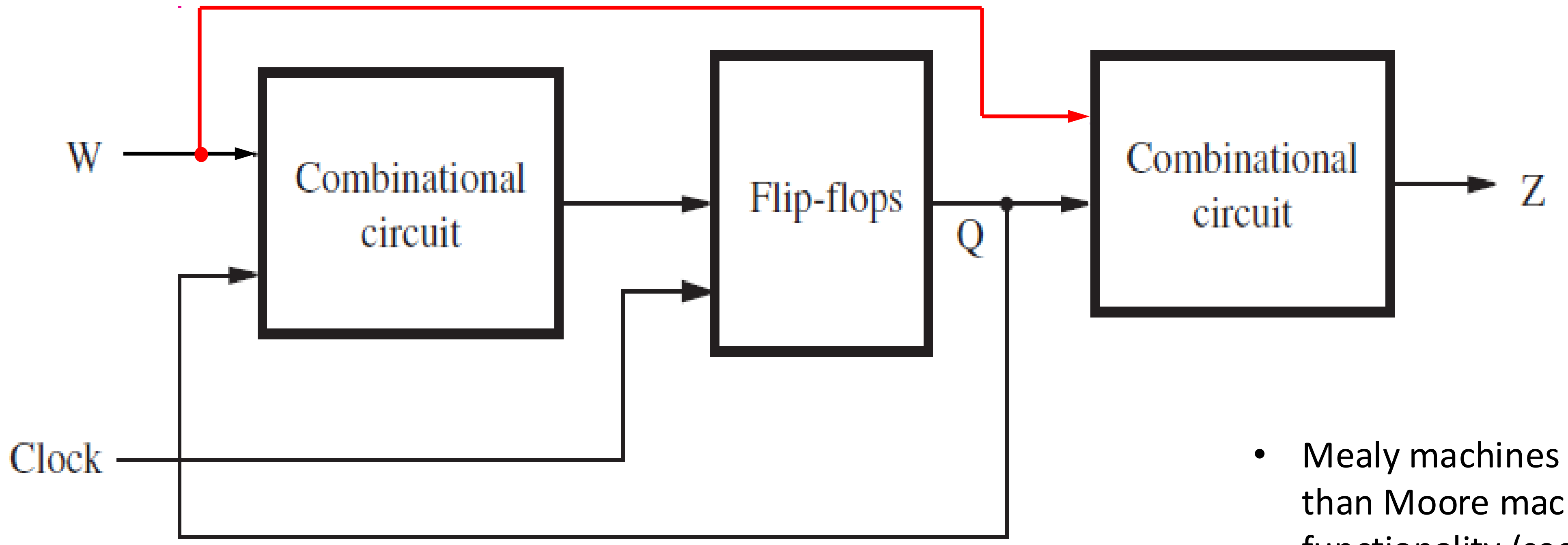
Finite State Machines (FSM)

Finite State Machine (FSM)

- FSM provides a systematic way to design synchronous sequential circuits given a functional specification
- Consists of:
  - State register
    - Stores current state
    - Loads next state at clock edge
  - Combinational logic
    - Computes the next state
    - Computes the outputs

Why we call it Finite State Machine?

Moore and Mealy FSM



•W: primary inputs
•Q: present (current) outputs of the flip-flops, i.e. state of circuit.
•Z: outputs
•**Moore machine**: outputs depend only on the state (flip-flops) of the circuit.
•**Mealy machine**: outputs depend on both the state (flip-flops) and the primary inputs

- Mealy machines can be smaller than Moore machines for same functionality (see next example).
- However, their usage may cause problems: When using feedback loops over FSMs, combinational loops may occur without a register (instability!).

FSM: Basic Design Steps

1. State Diagram

2. State Table

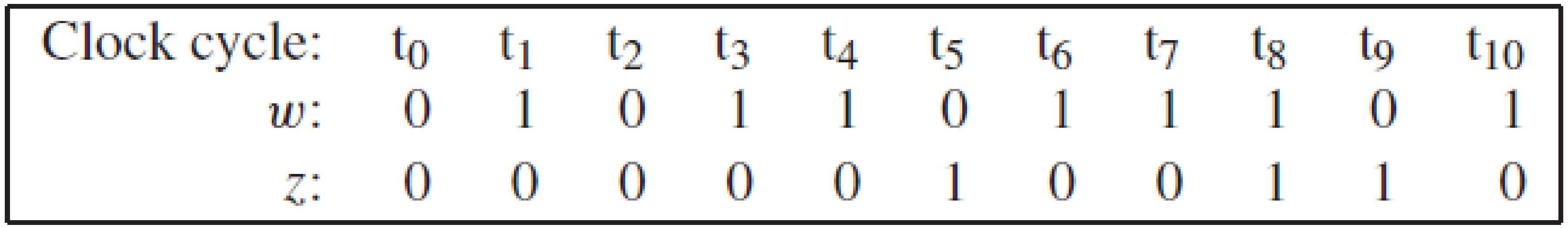
3. State Assignment

4. Choice of Flip-Flops and Derivation of Next-State and Output Expressions

5. Implementation of Next-State and Output Expressions using Logic Gates

Example: Sequence Detector using Moore Style

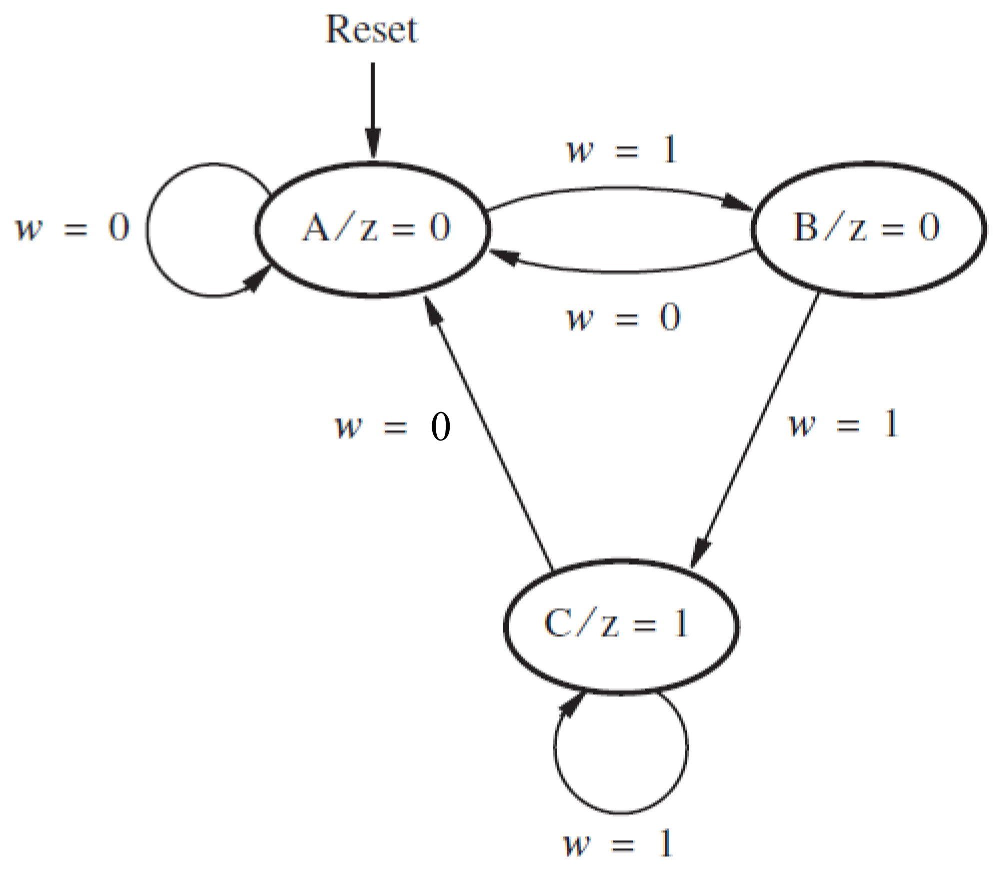
Design a synchronous FSM that has one input (w) and one output (z)
The output (z) must be equal 1 if during two subsequent clock cycles the input (w) was equal 1. Otherwise, z must be 0.



Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

Example: Sequence Detector using Moore Style

- **State Diagram**



- **State Table**

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

Example: Sequence Detector using Moore Style

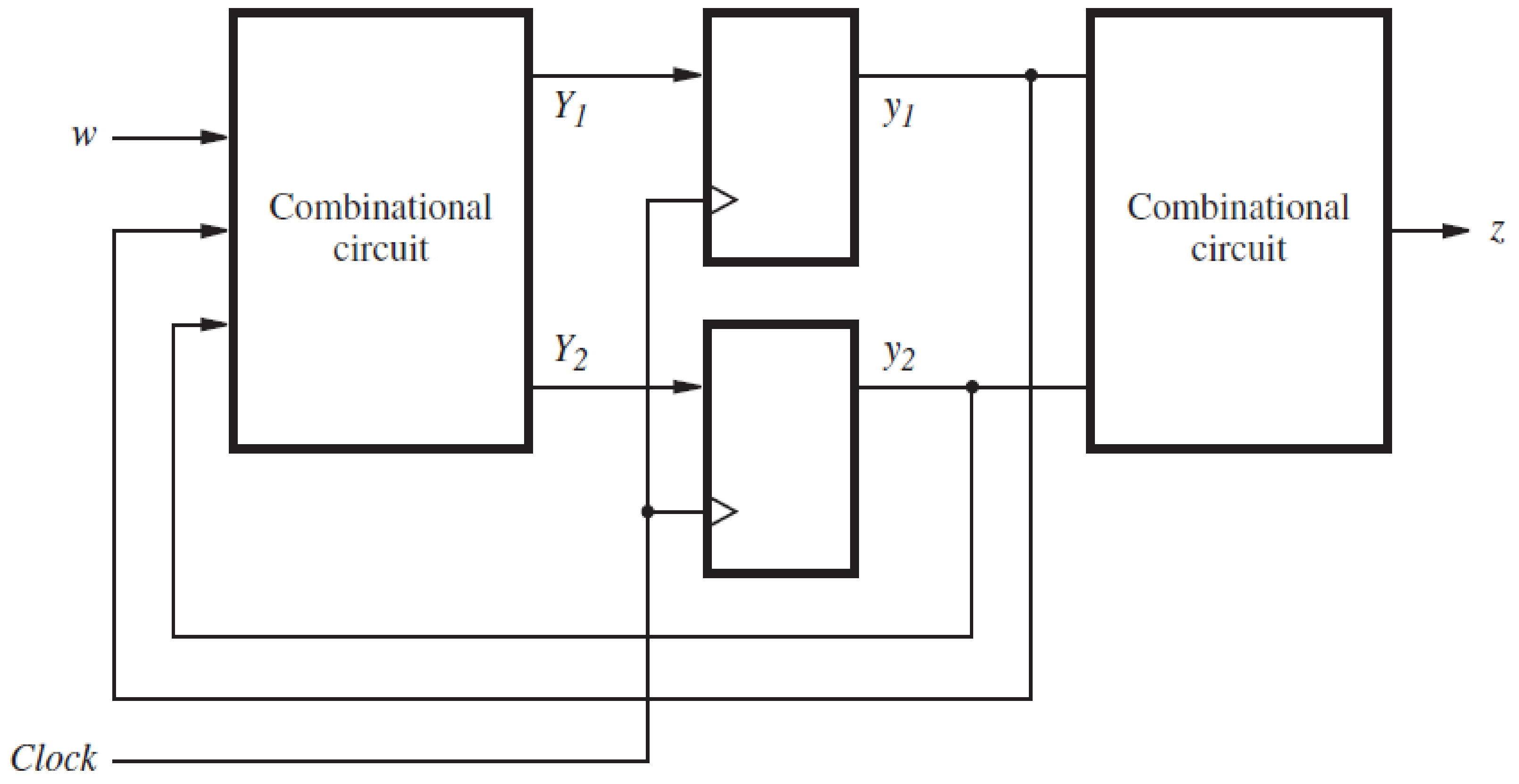
- **State Table**

Present state	Next state $w=0$	Next state $w=1$	Output z
A	A	B	0
B	A	C	0
C	A	C	1

- **State-assigned table**

	Present state $y_2 y_1$	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	dd	dd	d

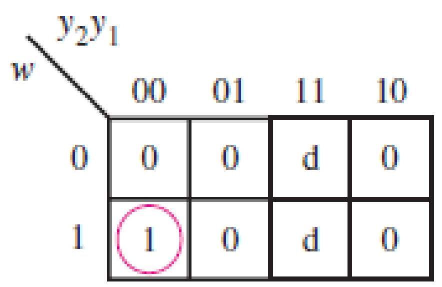
d = don't care (X)



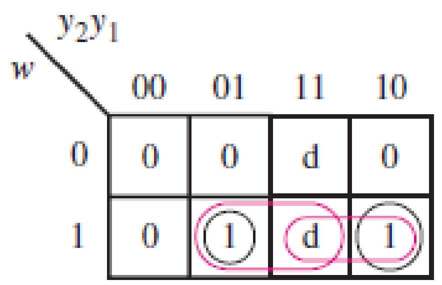
Note: for next state variables we sometimes also use the following notation: $y_2^+ \; y_1^+$

Example: Sequence Detector using Moore Style

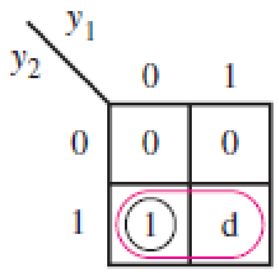
	Present state	Next state		Output
		$w = 0$	$w = 1$	
	$y_2 y_1$	$Y_2 Y_1$	$Y_2 Y_1$	z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	dd	dd	d



$w \backslash y_2 y_1$	00	01	11	10
0	0	0	d	0
1	1	0	d	0



$w \backslash y_2 y_1$	00	01	11	10
0	0	0	d	0
1	0	1	d	1



$y_2 \backslash y_1$	0	1
0	0	0
1	1	d

Ignoring don't cares

$$Y_1 = w\bar{y}_1\bar{y}_2$$

$$Y_2 = wy_1\bar{y}_2 + w\bar{y}_1y_2$$

$$z = \bar{y}_1y_2$$

Using don't cares

$$Y_1 = w\bar{y}_1\bar{y}_2$$

$$\begin{aligned} Y_2 &= wy_1 + wy_2 \\ &= w(y_1 + y_2) \end{aligned}$$

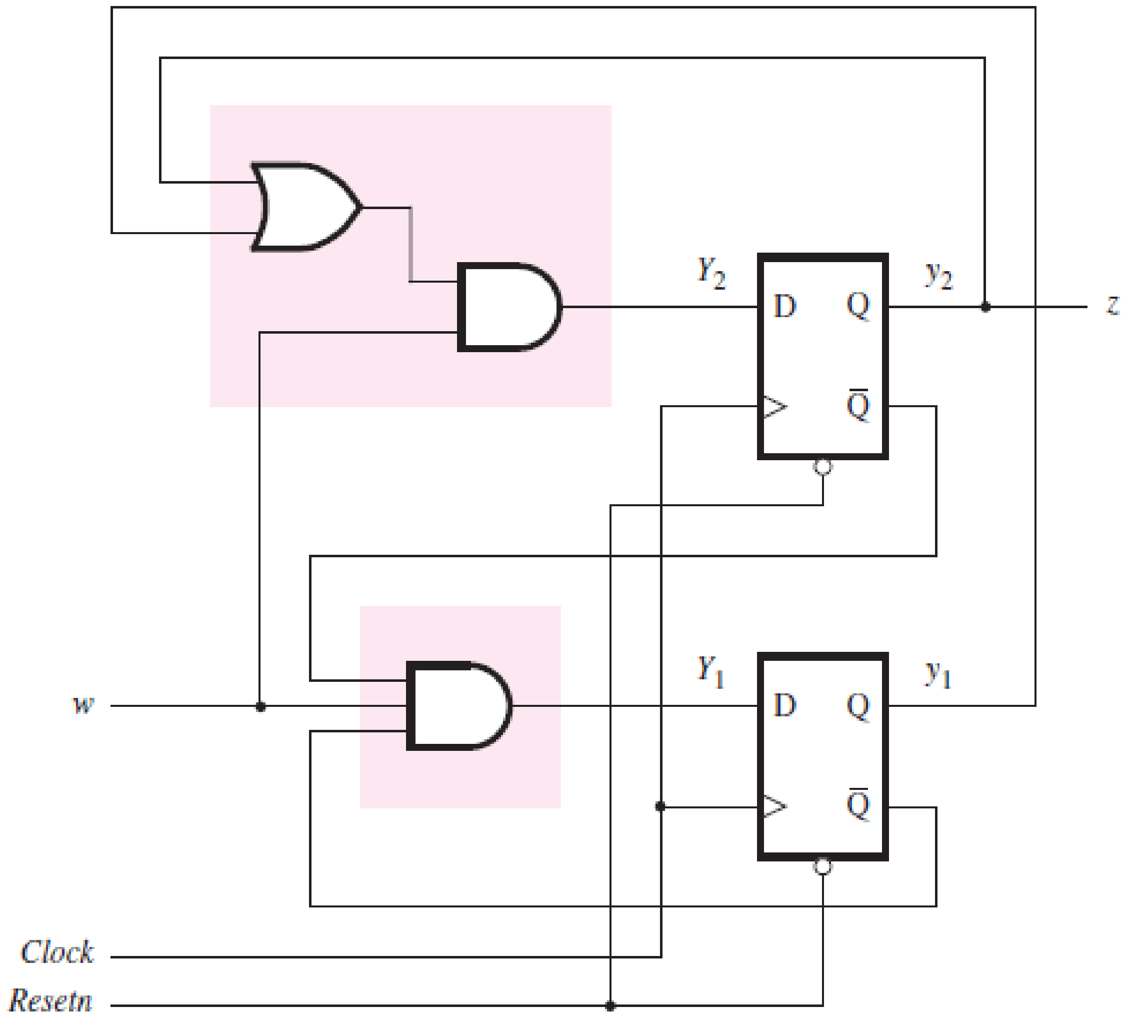
$$z = y_2$$

Example: Sequence Detector using Moore Style

$$Y_1 = w\bar{y}_1\bar{y}_2$$

$$Y_2 = w(y_1 + y_2)$$

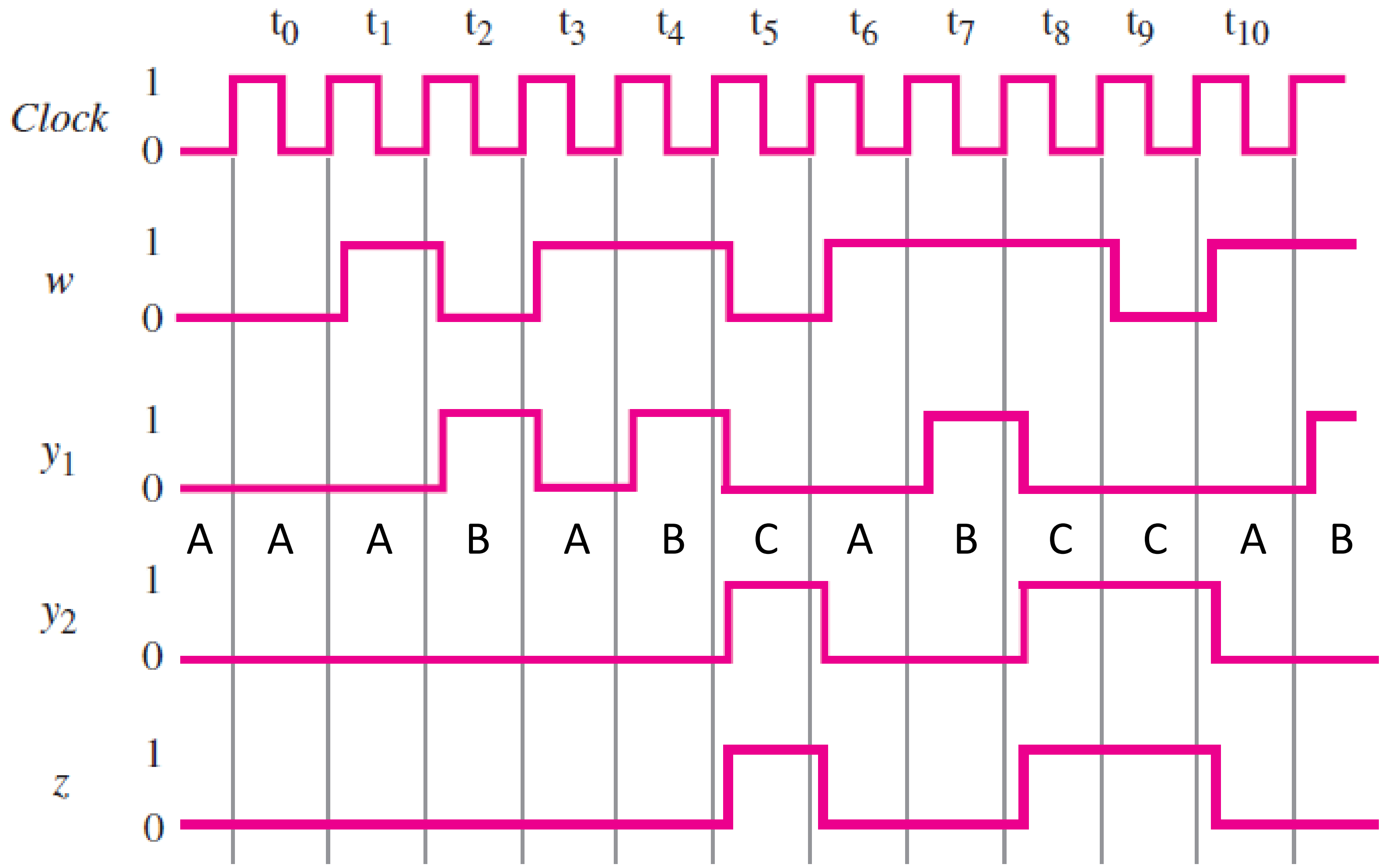
$$z = y_2$$



	Present state	Next state		Output
		$w = 0$	$w = 1$	z
	$y_2 y_1$	$Y_2 Y_1$	$Y_2 Y_1$	
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	dd	dd	d

Example: Sequence Detector using Moore Style

- **Timing diagram**

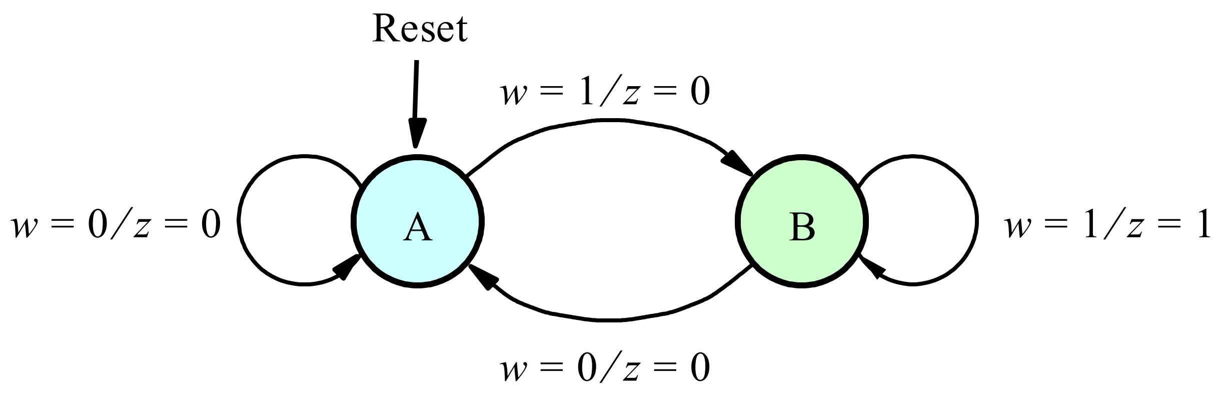


t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
Clock
w
y_1
y_2
z
1
0
A A A B A B C A B C C A B

Example: Sequence Detector using Mealy Style

Mealy Machine

Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0



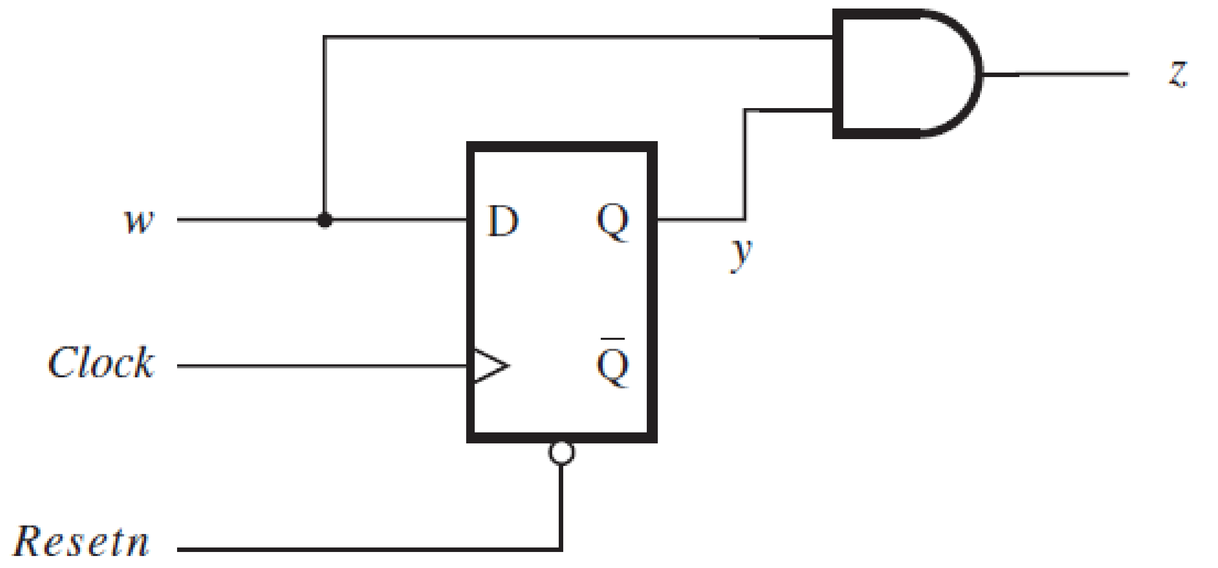
For a Mealy machine, output values are given per state transition instead of per state (Moore).

Also note: now z can become 1 in same clock cycle as cycle where w is 1 for the second time.

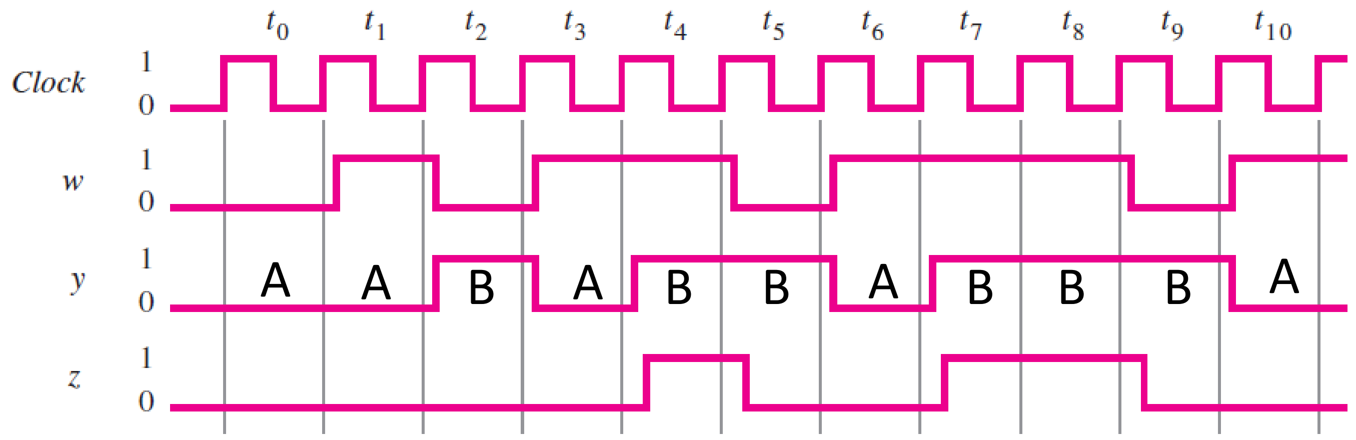
Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

Example: Sequence Detector using Mealy Style

Present state	Next state		Output *z*	
	w = 0	*w* = 1	*w* = 0	*w* = 1
A	A	B	0	0
B	A	B	0	1



	Present state	Next state		Output	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1



Summary

- Sequential networks
  - Synchronous/asynchronous sequential systems
  - Latches
  - Flip-flops: Edge/level triggered
- Finite State Machines
  - Finite State Diagram
  - Design method to go from a problem to an FSM implementation
  - Moore and Mealy Machines

To Do List

- Reading Material book "Digital Design":
  - Sections 3.1 – 3.3 (not 3.2.7)

- Assignments for this lecture:
  - Gated Practice Assignment Lecture 6

Thank you