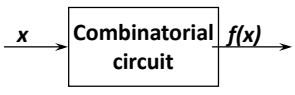
# Digital systems: from bits to microcontrollers

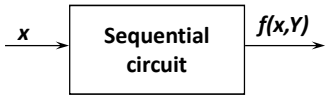
- Introduction to digital systems fundamentals
- Combinatorial digital systems
- Sequential digital systems
- Introduction to microcontrollers
- Introduction to advanced implementation platforms



# Introducing state variables (internal state)



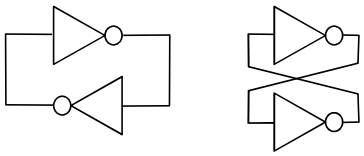

Output depends on current inputs

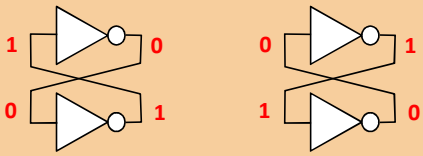



Output depends on current inputs and internal state (which in turn depends on history of inputs)

# Introducing feedback in logic elements…

Connecting output to input



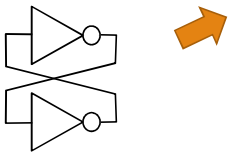


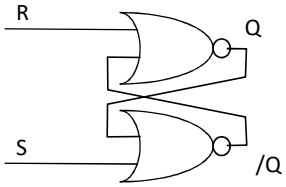

Two stable states can be observed

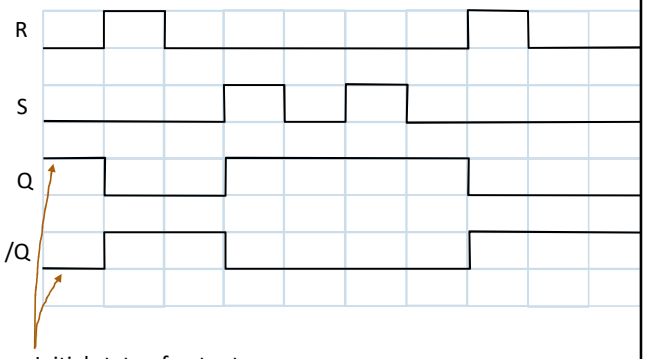
Problem: how to impose a specific state?



# Introducing inputs

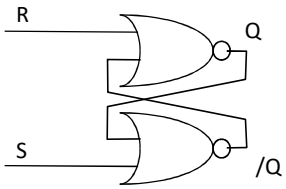


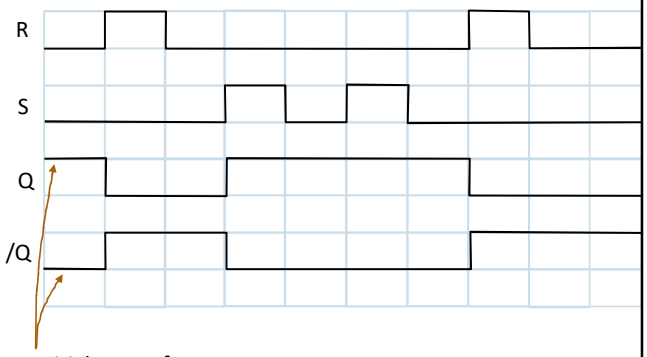



Initial state of outputs

# Introducing inputs

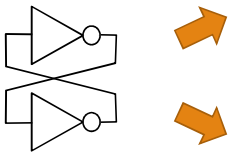


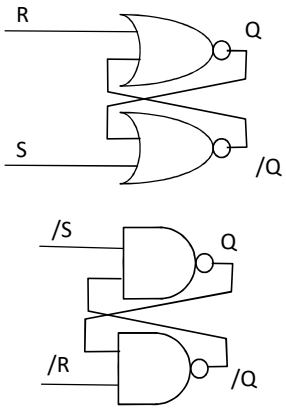



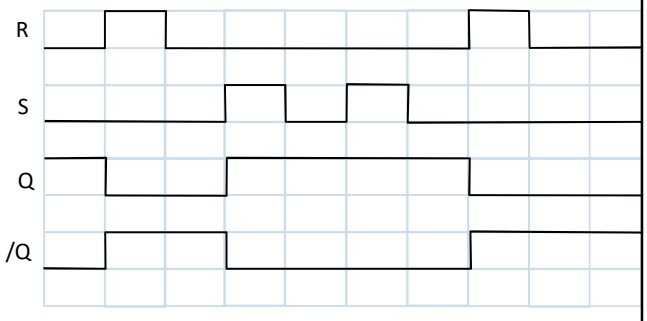

Input R (reset) forces output Q to go 0;
Input S (set) forces output Q to go 1;
Outputs Q and /Q present complementary values



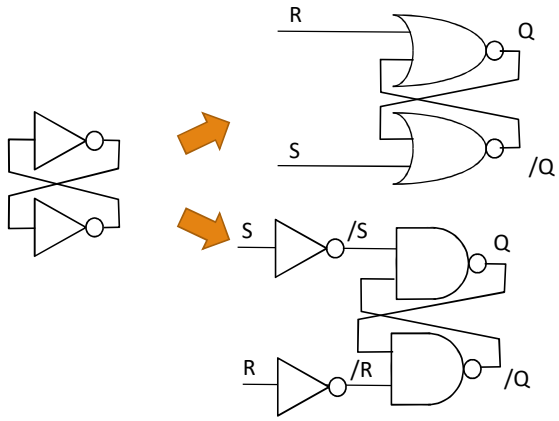
# Introducing inputs

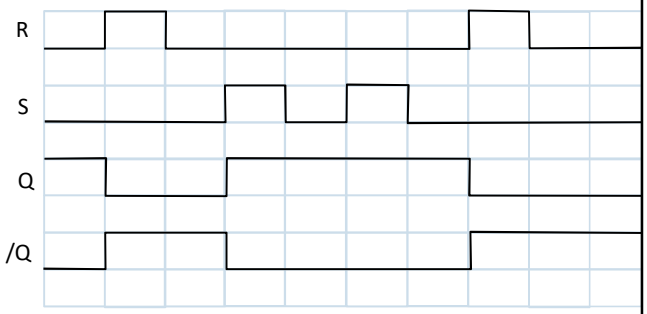




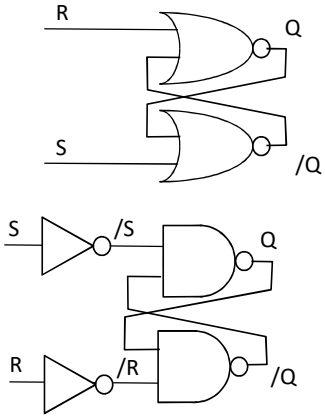
# Introducing inputs



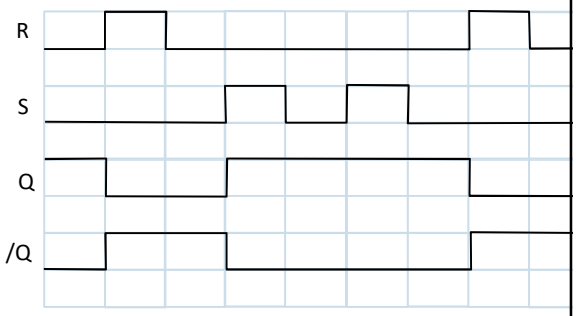






# SR Latch



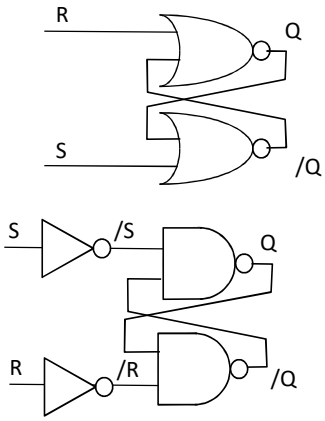

| S | R | $Q_t$ | $Q_{t+\delta}$ | $/Q_{t+\delta}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | x | x |
| 1 | 1 | 1 | x | x |



# SR Latch

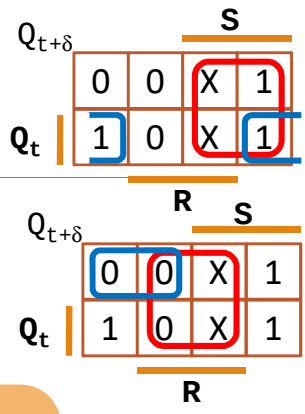
$Q_{t+\delta} = S + /R.Q_t = /(/S \ . \ /(/R.Q_t))$

$Q_{t+\delta} = /R \ . \ (S+Q_t) = /(R + /(S+Q_t))$



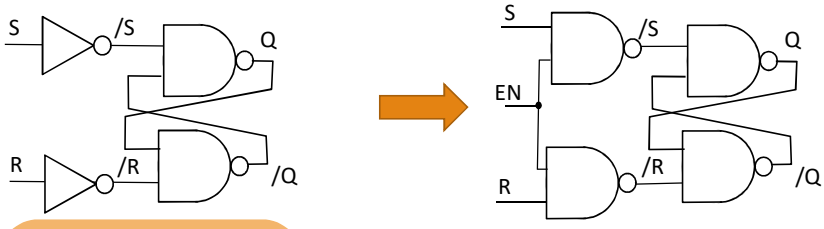

| S | R | $Q_t$ | $Q_{t+\delta}$ | $/Q_{t+\delta}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | x | x |
| 1 | 1 | 1 | x | x |

| S | R | $Q_{t+\delta}$ | $/Q_{t+\delta}$ |
|---|---|---|---|
| 0 | 0 | $Q_t$ | $/Q_t$ |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | x | x |





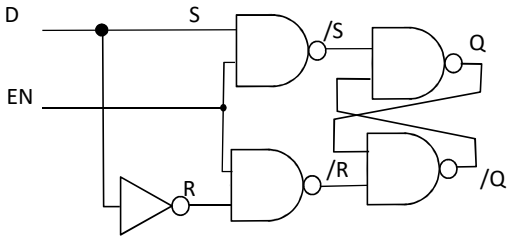

# SR Latch with enable (gated SR latch)




| S | R | $Q_{t+\delta}$ | $/Q_{t+\delta}$ |
|---|---|---|---|
| 0 | 0 | $Q_t$ | $/Q_t$ |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | x | x |

| EN | S | R | $Q_{t+\delta}$ | $/Q_{t+\delta}$ |
|---|---|---|---|---|
| 0 | - | - | $Q_t$ | $/Q_t$ |
| 1 | 0 | 0 | $Q_t$ | $/Q_t$ |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | x | x |

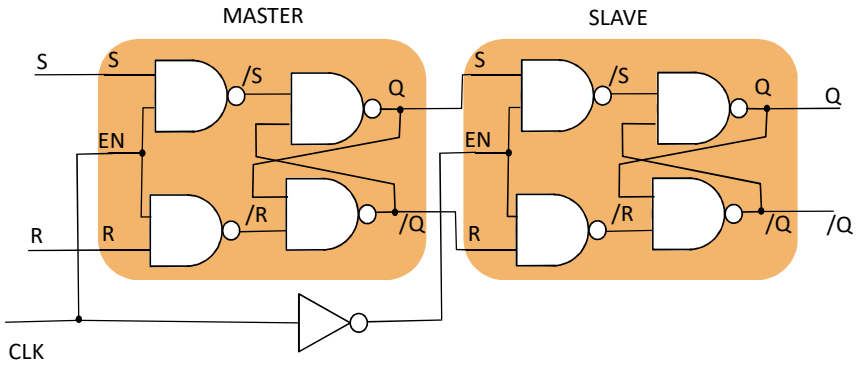
# Delay (D) Latch with enable

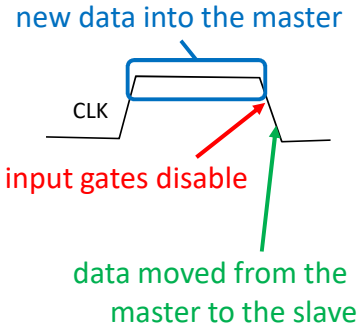



| EN | D | $Q_{t+\delta}$ | $/Q_{t+\delta}$ |
|---|---|---|---|
| 0 | - | $Q_t$ | $/Q_t$ |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

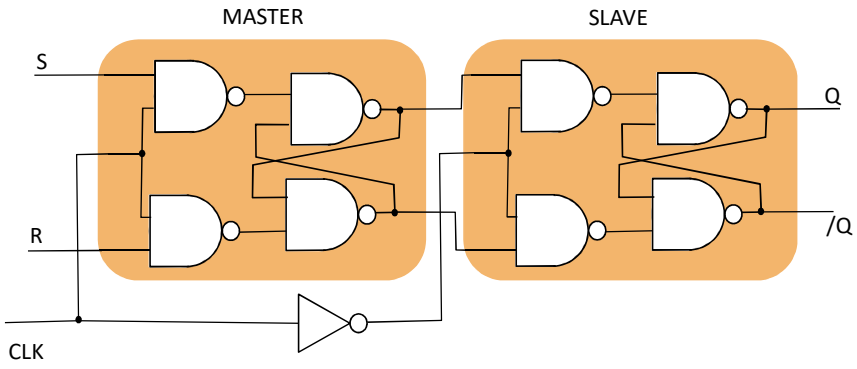


# Introducing master-slave flip-flop



# Introducing master-slave flip-flop

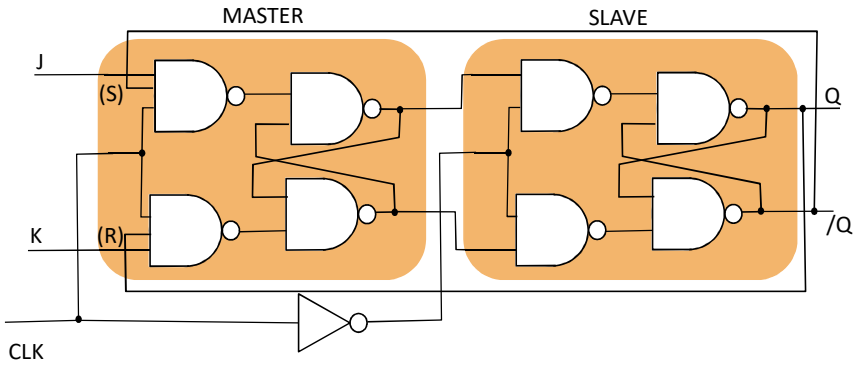



SR truth table

| S | R | $Q_{n+1}$ | $/Q_{n+1}$ |
|---|---|---|---|
| 0 | 0 | $Q_n$ | $/Q_n$ |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | x | x |



# Introducing master-slave JK flip-flop




JK truth table

| J | K | $Q_{n+1}$ | $/Q_{n+1}$ |
|---|---|---|---|
| 0 | 0 | $Q_n$ | $/Q_n$ |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | $/Q_n$ | $Q_n$ |

# Master-slave versus edge-trigerred flip-flops

While:

- master-slave flip-flop are receptive to inputs during one level of the clock and updates output in one edge of the clock,

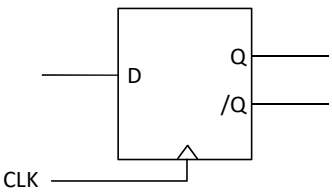
- edge-trigerred flip-flops are receptive to inputs in the same clock edge where output are updated.

In this sense, the edge-trigerred flip-flop truth table applies considering the inputs at the moment immediatly before the clock edge.

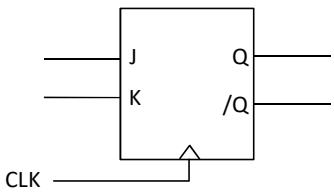
Edge-trigerred flip-flops are the ones normally used.



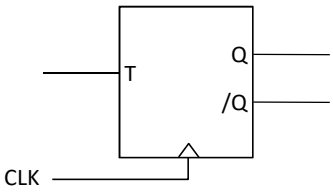
# Most common flip-flops




| D | $Q_{n+1}$ |
|---|---|
| 0 | 0 |
| 1 | 1 |




| J | K | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $/Q_n$ |

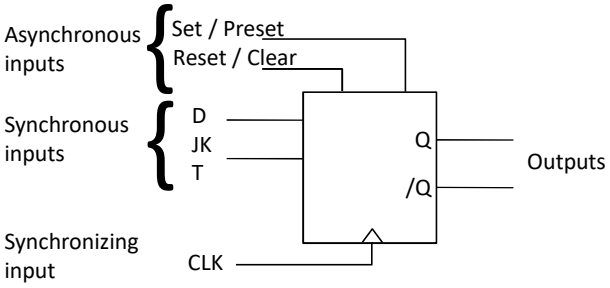



| T | $Q_{n+1}$ |
|---|---|
| 0 | $Q_n$ |
| 1 | $/Q_n$ |

# Asynchronous versus synchronous inputs

Synchronous inputs: whenever the CLOCK event occurs, the flip-flop output is governed by the truth table associated with synchronous inputs.

Asynchronous inputs: they will affect the flip-flop output independently of the CLOCK event.

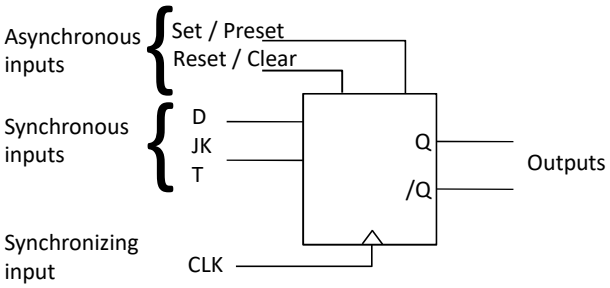




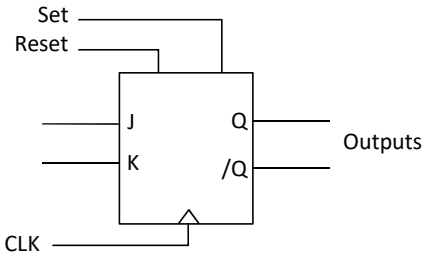

# Asynchronous versus synchronous inputs

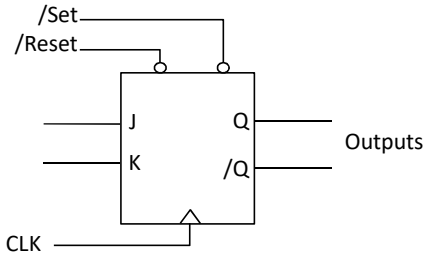
Synchronous inputs: normally used to impose a desired behavior over time.

Asynchronous inputs: normally used to force initial value.



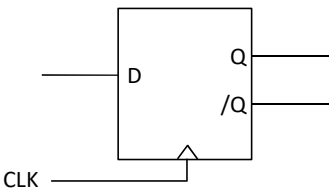
# Asynchronous inputs: active values



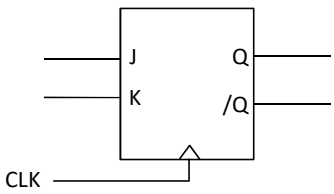





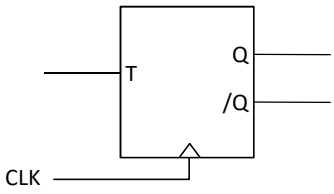

# May I use one flip-flop to produce a different one?




| D | $Q_{n+1}$ |
|---|---|
| 0 | 0 |
| 1 | 1 |

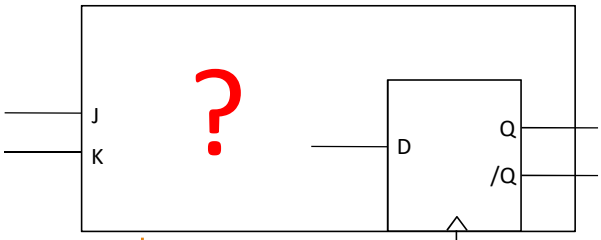



| J | K | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $/Q_n$ |




| T | $Q_{n+1}$ |
|---|---|
| 0 | $Q_n$ |
| 1 | $/Q_n$ |

# May I use one D flip-flop to produce a JK flip-flop?



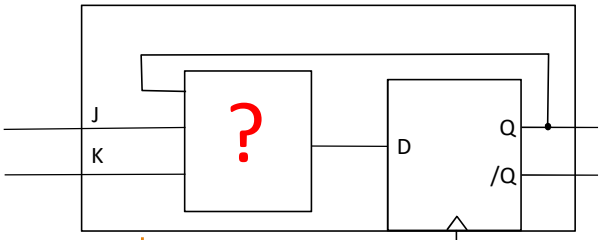

| J | K | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $/Q_n$ |

| D | $Q_{n+1}$ |
|---|---|
| 0 | 0 |
| 1 | 1 |



# May I use one D flip-flop to produce a JK flip-flop?

| J | K | $Q_n$ | $Q_{n+1}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

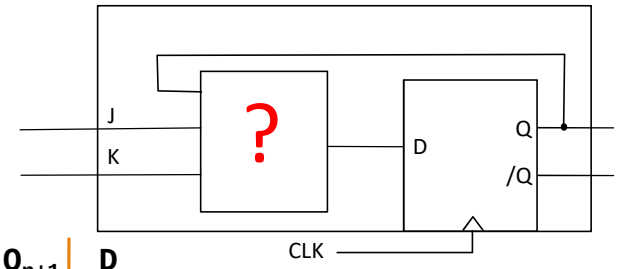



| J | K | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $/Q_n$ |

| D | $Q_{n+1}$ |
|---|---|
| 0 | 0 |
| 1 | 1 |

# May I use one D flip-flop to produce a JK flip-flop?

| J | K | $Q_n$ | $Q_{n+1}$ | D |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |



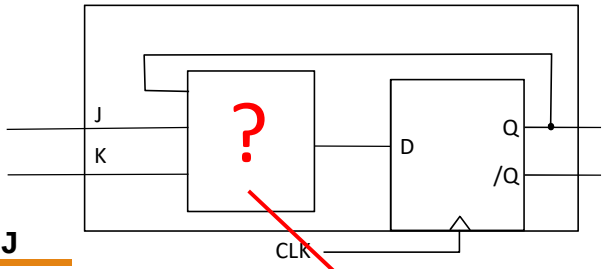

| $Q_n$ → | $Q_{n+1}$ | D |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| D | $Q_{n+1}$ |
|---|---|
| 0 | 0 |
| 1 | 1 |



# May I use one D flip-flop to produce a JK flip-flop?

| J | K | $Q_n$ | $Q_{n+1}$ | D |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |

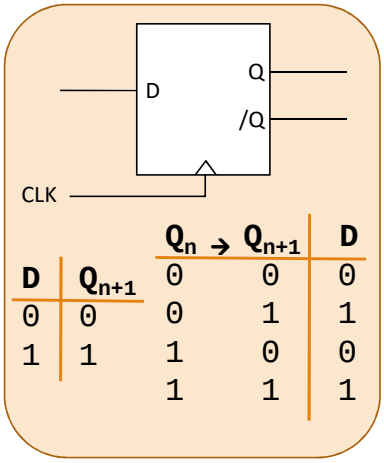



| $Q_n$ \ J | 0 | 0 | 1 | 1 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |

K

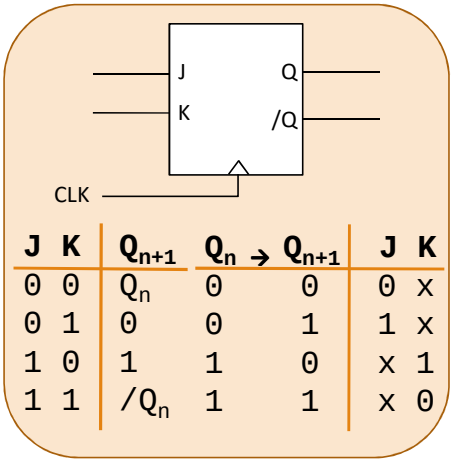
$$D = J./Q_n + /K.Q_n$$

# Flip-flop truth tables and transition tables



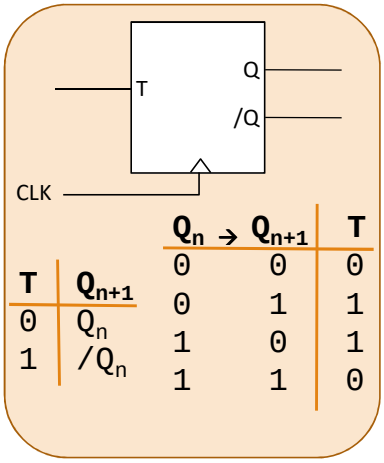

| D | $Q_{n+1}$ |
|---|---|
| 0 | 0 |
| 1 | 1 |

| $Q_n$ | → | $Q_{n+1}$ | D |
|---|---|---|---|
| 0 | | 0 | 0 |
| 0 | | 1 | 1 |
| 1 | | 0 | 0 |
| 1 | | 1 | 1 |




| J | K | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $/Q_n$ |

| $Q_n$ | → | $Q_{n+1}$ | J | K |
|---|---|---|---|---|
| 0 | | 0 | 0 | x |
| 0 | | 1 | 1 | x |
| 1 | | 0 | x | 1 |
| 1 | | 1 | x | 0 |

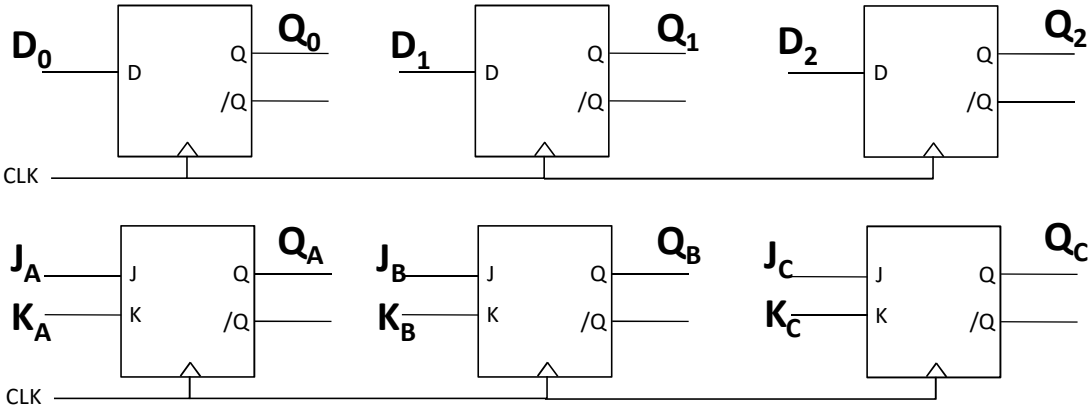



| T | $Q_{n+1}$ |
|---|---|
| 0 | $Q_n$ |
| 1 | $/Q_n$ |

| $Q_n$ | → | $Q_{n+1}$ | T |
|---|---|---|---|
| 0 | | 0 | 0 |
| 0 | | 1 | 1 |
| 1 | | 0 | 1 |
| 1 | | 1 | 0 |

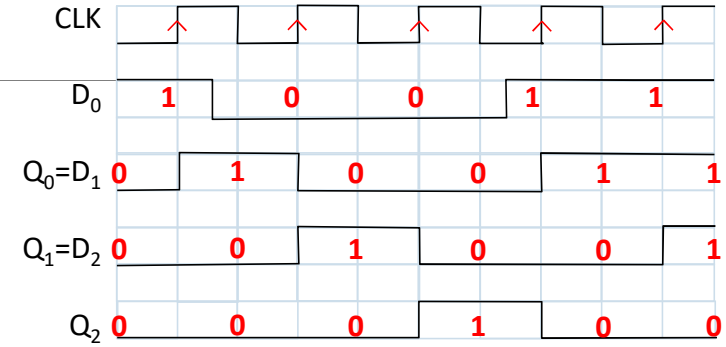
# Connecting flip-flops into registers

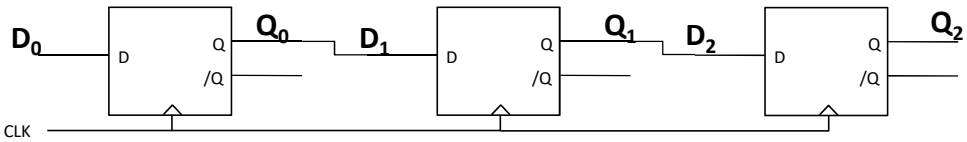
A group of flip-flops using the same synchronizing signal is a register.



# A shift-register

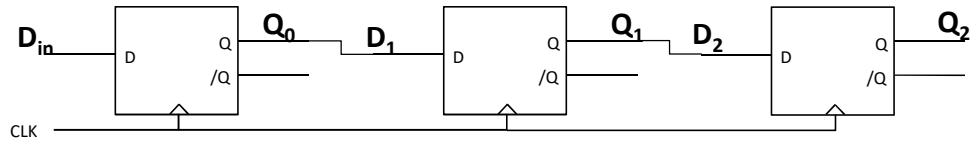
| D | $Q_{n+1}$ |
|---|---|
| 0 | 0 |
| 1 | 1 |



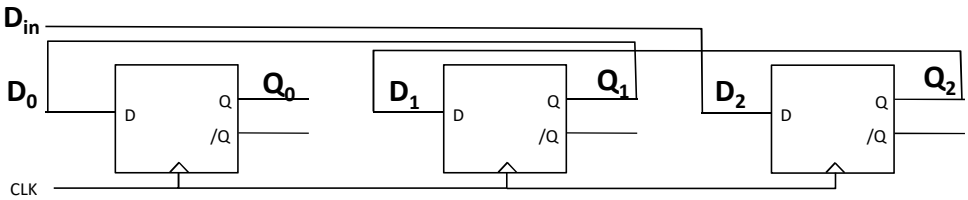




# Left and right shift-registers

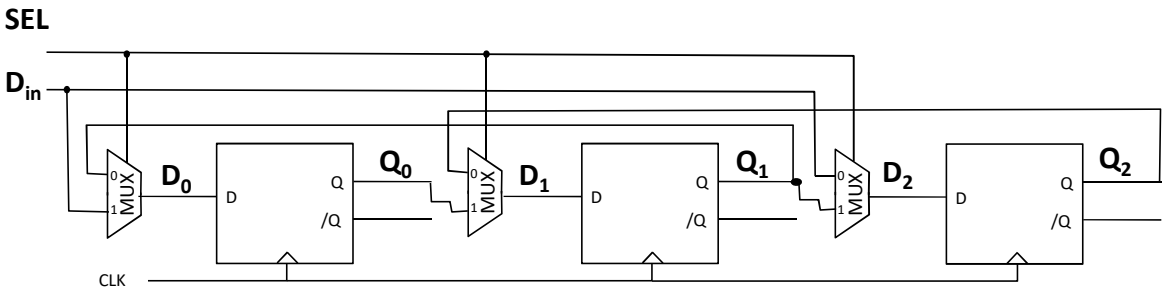
**Right**




**Left**

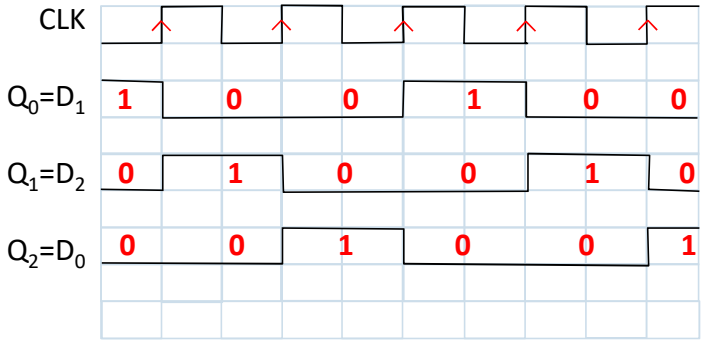


# Left-right shift-register

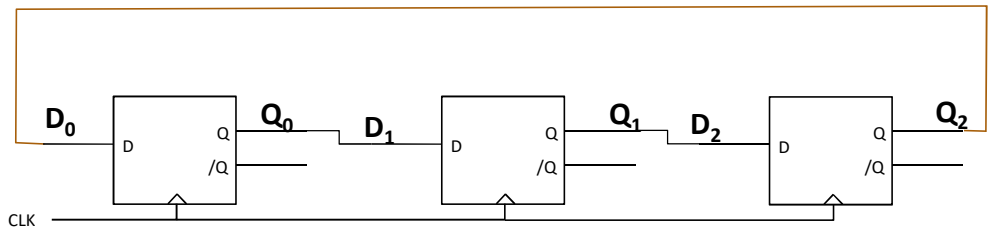



If (SEL = 0) then shift left
else shift right

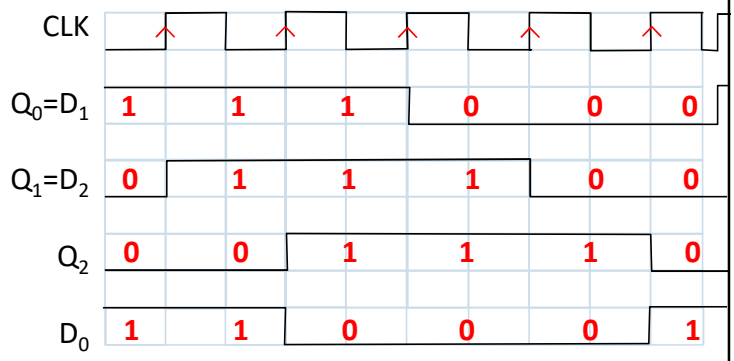
# Shift-registers as counters: ring counter



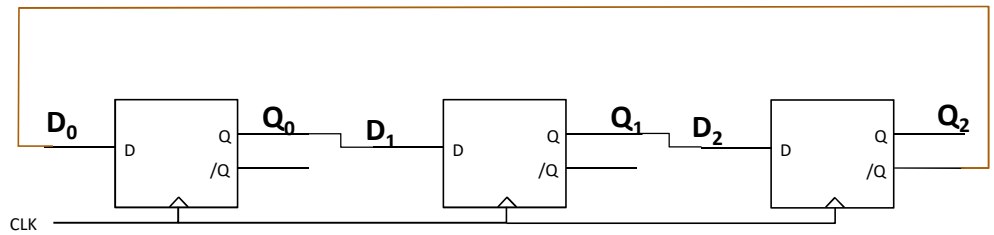

N flips => N counting states



# Shift-registers as counters: twisted-ring counter (Johnson counter)

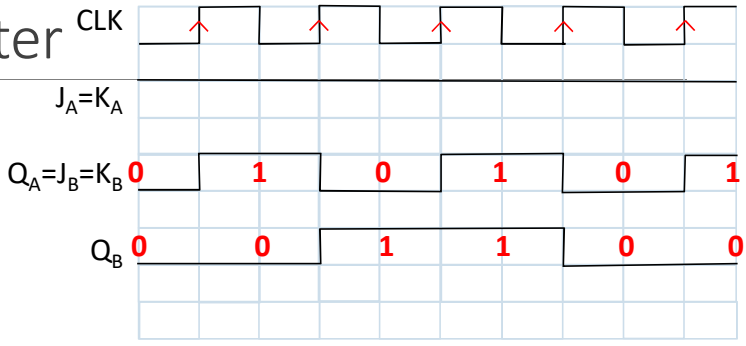


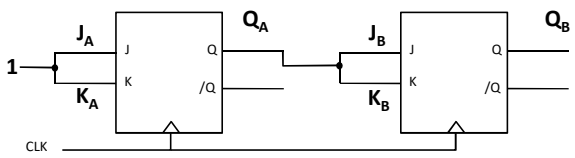

N flips => 2*N counting states




# A simple counter

| J | K | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $/Q_n$ |



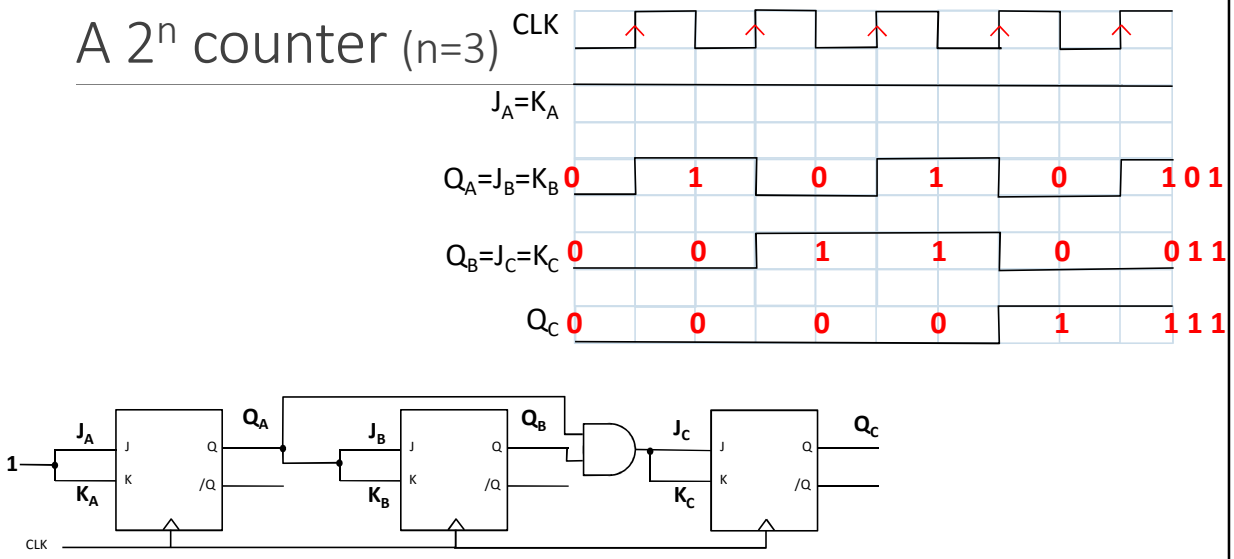




$(Q_BQ_A)$ 00 01 10 11 00 01
0 1 2 3 0 1

N flips => $2^N$ counting states

2-bit counter (modulo 4)

# A $2^n$ counter (n=3)

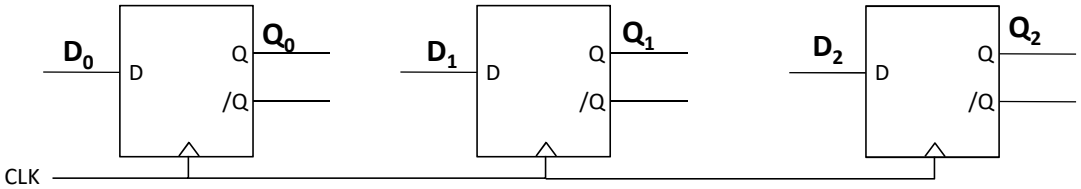





# Revisiting registers

Sometimes is necessary that one register performs different functions along the time.

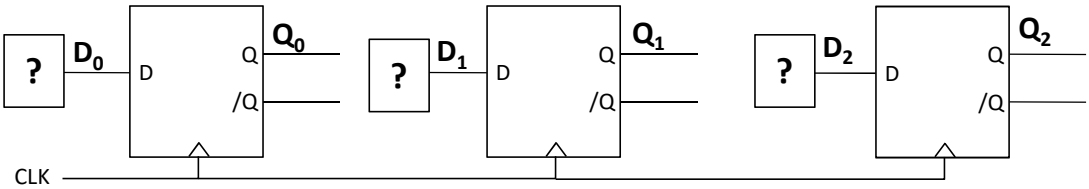
As the flip-flop structure is the same for all functions, the different functions are implemented through different flip-flop input functions.



# Revisiting registers

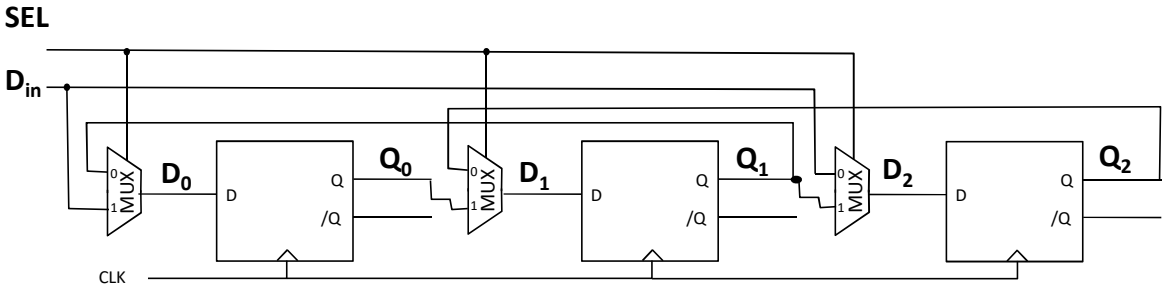
Sometimes is necessary that one register performs different functions along the time.

As the flip-flop structure is the same for all functions, the different functions are implemented through different flip-flop input functions.



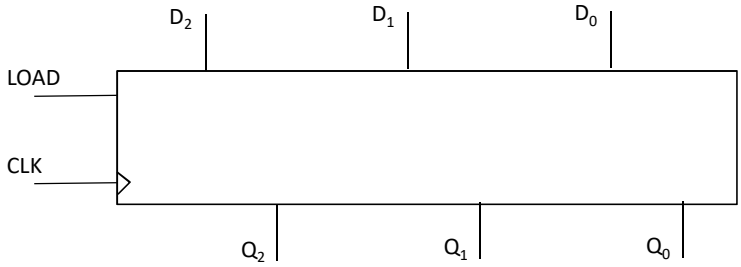



# Left-right shift-register




If (SEL = 0) then shift left
else shift right

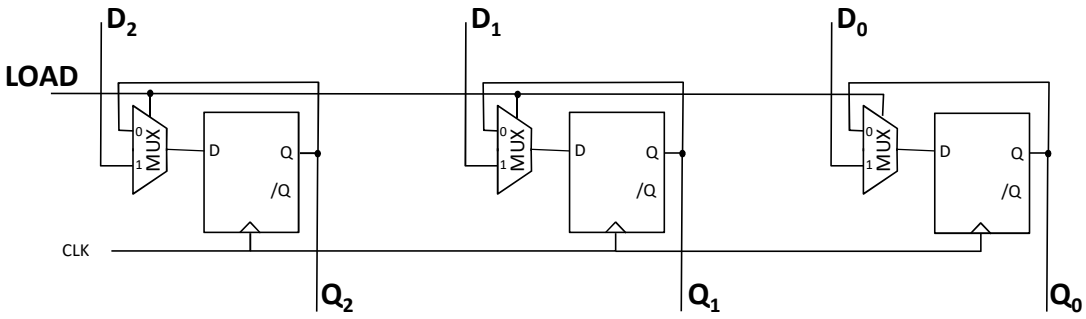
# Register with parallel load control




If (LOAD = 1) then load data inputs
else freeze outputs

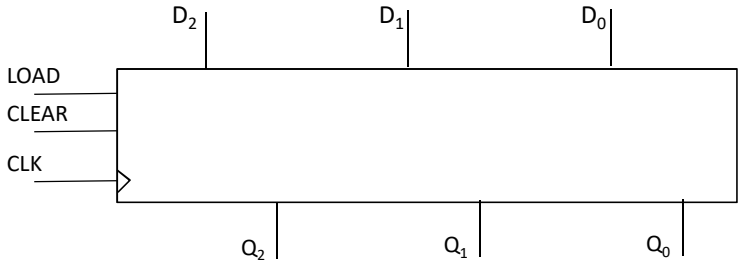


# Register with parallel load control



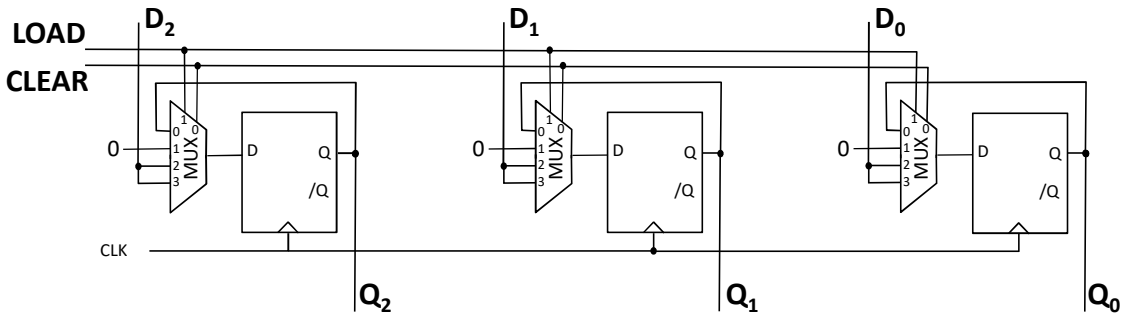

If (LOAD = 1) then load data inputs
else freeze outputs

# Register with parallel load control and clear



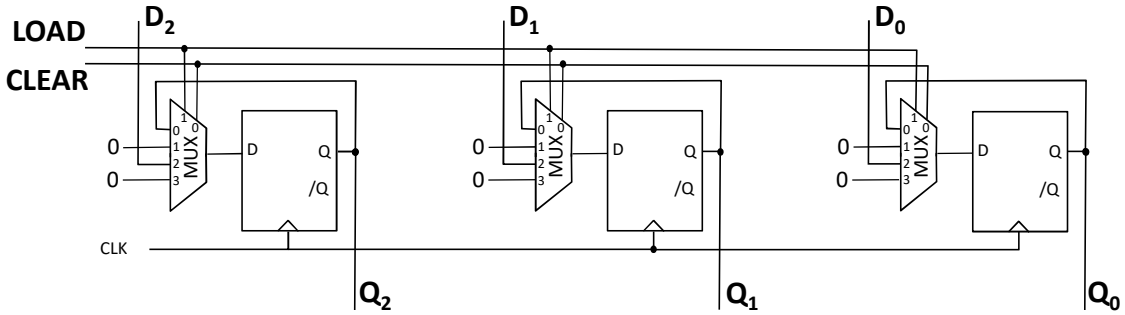

If (LOAD = 1) then load data inputs
else if (CLEAR = 1) then clear outputs
else freeze outputs

# Register with parallel load control and clear




If (LOAD = 1) then load data inputs
else if (CLEAR = 1) then clear outputs
else freeze outputs

# Register with parallel load control and clear




If (CLEAR = 1) then clear outputs
else if (LOAD = 1) then load data inputs
else freeze outputs

# An arbitrary modulus counter

**0 → 1 → 2 → 0 → 1 → …**

For example: modulus 3 counter

# An arbitrary modulus counter

**0 → 1 → 2 → 0 → 1 → …**

| Current state $S_i = Q_1Q_0$ | Next state |
|---|---|
| $S_0 = 00$ | $S_1 = 01$ |
| $S_1 = 01$ | $S_2 = 10$ |
| $S_2 = 10$ | $S_0 = 00$ |

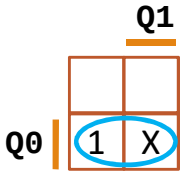


# An arbitrary modulus counter (modulus 3)

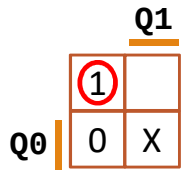
**0 → 1 → 2 → 0 → 1 → …**

$S_i = Q_1Q_0$

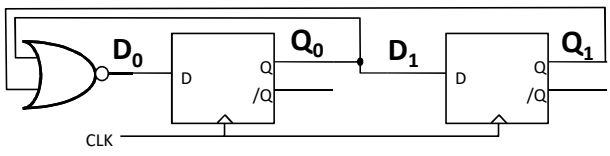
| Current state $(Q_1Q_0)_n$ | Next state $(Q_1Q_0)_{n+1}$ | $D_1D_0$ |
|---|---|---|
| $(S_0)$ 0 0 | 0 1 | 0 1 |
| $(S_1)$ 0 1 | 1 0 | 1 0 |
| $(S_2)$ 1 0 | 0 0 | 0 0 |



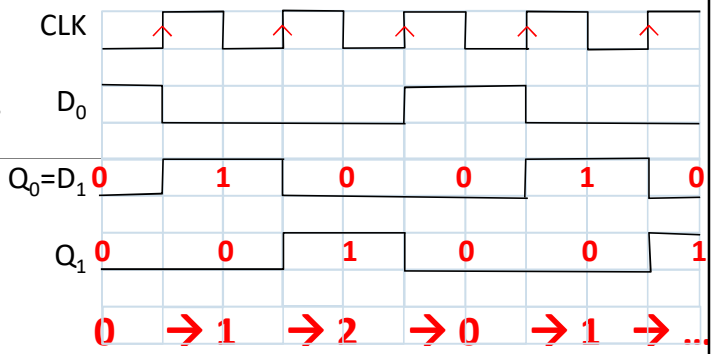

**D1 = Q0**




**D0 = /Q1./Q0 =**
**= /(Q1+Q0)**



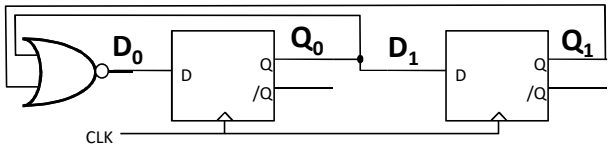
# A modulus 3 counter




$S_i = Q_1Q_0$

| Current state $(Q_1Q_0)_n$ | Next state $(Q_1Q_0)_{n+1}$ | $D_1D_0$ |
|---|---|---|
| $(S_0)$ 0 0 | 0 1 | 0 1 |
| $(S_1)$ 0 1 | 1 0 | 1 0 |
| $(S_2)$ 1 0 | 0 0 | 0 0 |

**D1 = Q0**  **D0 = /Q1./Q0 = = /(Q1+Q0)**

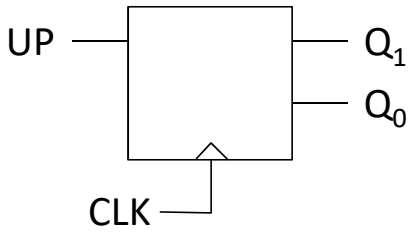





# Up-down counter

For example: modulus 3 up/down counter

if (UP = 1) then counts up
else counts down



# Modulus 3 up/down counter

if (UP = 1) then counts up
else counts down

$S_i = Q_1Q_0$

| Current state $(Q_1Q_0)_n$ | Next state UP=0 $(Q_1Q_0)_{n+1}$ | Next state UP=1 $(Q_1Q_0)_{n+1}$ |
|---|---|---|
| $(S_0)$ 0 0 | 1 0 | 0 1 |
| $(S_1)$ 0 1 | 0 0 | 1 0 |
| $(S_2)$ 1 0 | 0 1 | 0 0 |

| | Current state $(Q_1Q_0)_n$ | | Next state $(Q_1Q_0)_{n+1}$ | |
|---|---|---|---|---|
| UP | | | | |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | X | X |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | X | X |



# Modulus 3 up/down counter

if (UP = 1) then counts up
else counts down

$S_i = Q_1Q_0$

| Current state $(Q_1Q_0)_n$ | Next state UP=0 $(Q_1Q_0)_{n+1}$ | Next state UP=1 $(Q_1Q_0)_{n+1}$ | FF inputs UP=0 $D_1D_0$ | FF inputs UP=1 $D_1D_0$ |
|---|---|---|---|---|
| $(S_0)$ 0 0 | 1 0 | 0 1 | 1 0 | 0 1 |
| $(S_1)$ 0 1 | 0 0 | 1 0 | 0 0 | 1 0 |
| $(S_2)$ 1 0 | 0 1 | 0 0 | 0 1 | 0 0 |
| 1 1 | X X | X X | X X | X X |

$D_1$

| $Q0_n$ \ UP | 1 | 0 | 0 |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| X | X | X | 1 |

$Q1_n$

$D_0$

| $Q0_n$ \ UP | 1 | 0 | 1 |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| X | X | X | 0 |

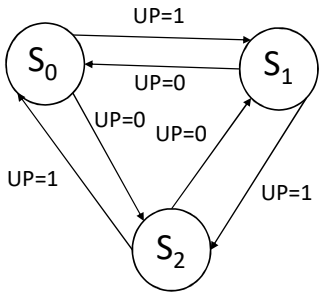
$Q1_n$

# Modulus 3 up/down counter

if (UP = 1) then counts up
else counts down

$S_i = Q_1Q_0$

| Current state $(Q_1Q_0)_n$ | Next state UP=0 $(Q_1Q_0)_{n+1}$ | Next state UP=1 $(Q_1Q_0)_{n+1}$ |
|---|---|---|
| $(S_0)$ 0 0 | 1 0 | 0 1 |
| $(S_1)$ 0 1 | 0 0 | 1 0 |
| $(S_2)$ 1 0 | 0 1 | 0 0 |

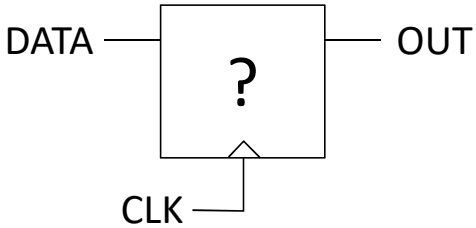




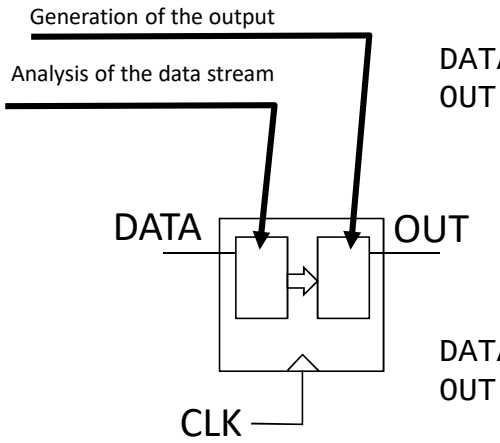

# Sequence detectors

Consider transmitting binary information through a communication channel. The data is transmitted synchronously with a clock signal, at a specific rate.

We intend to produce a system, which will analyze the data transmitted and detects whenever a specific sequence occurs.



# Sequence detector for 100



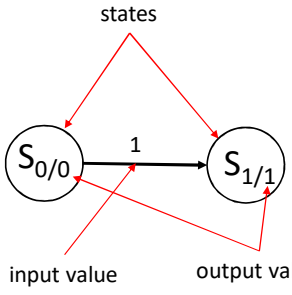

DATA → 0 1 1 0 1 0 1 1 1 0 0 0 1 0 0 0
OUT → 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0

Output active as data comes

Output active during next clock period after data comes

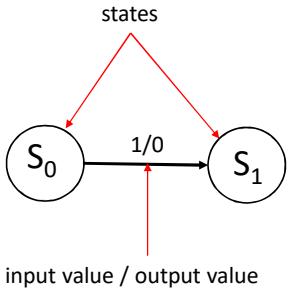
DATA → 0 1 1 0 1 0 1 1 1 0 0 0 1 0 0 0 0
OUT → 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0

# Brief comment on notation




**Moore state machine**

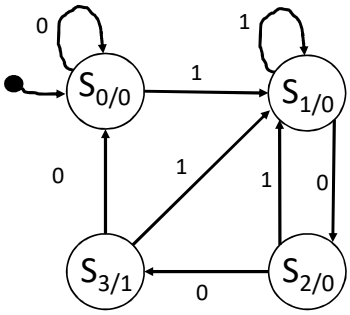
Outputs associated with states




**Mealy state machine**

Outputs associated with transitions

# Sequence detector for 100: using Moore state machine



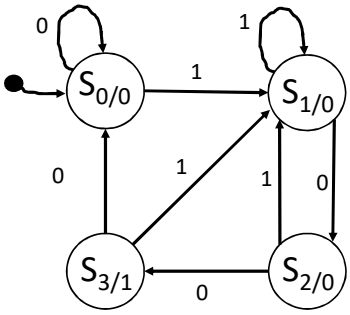

$S_0$ – out of sequence

$S_1$ – detected first bit of the sequence

$S_2$ – detected two first bits of the sequence

$S_3$ – sequence detected

# Sequence detector for 100: using Moore state machine




1 – State diagram

| Current state | Next state DATA=0 | Next state DATA=1 | OUT |
|---|---|---|---|
| $S_0$ | $S_0$ | $S_1$ | 0 |
| $S_1$ | $S_2$ | $S_1$ | 0 |
| $S_2$ | $S_3$ | $S_1$ | 0 |
| $S_3$ | $S_0$ | $S_1$ | 1 |

2 – State transition table and outputs

# Sequence detector for 100: using Moore state machine

| Current state | $Q_1Q_0$ |
|---|---|
| $S_0$ | 0 0 |
| $S_1$ | 0 1 |
| $S_2$ | 1 0 |
| $S_3$ | 1 1 |

3 – State encoding

| Current state | Next state | | |
|---|---|---|---|
| | DATA=0 | DATA=1 | |
| $(Q_1Q_0)_n$ | $(Q_1Q_0)_{n+1}$ | $(Q_1Q_0)_{n+1}$ | OUT |
| $(S_0)$ 00 | 00 | 01 | 0 |
| $(S_1)$ 01 | 10 | 01 | 0 |
| $(S_2)$ 10 | 11 | 01 | 0 |
| $(S_3)$ 11 | 00 | 01 | 1 |

4 – Encoded state transition table and outputs



# Sequence detector for 100: using Moore state machine

$Q_1$ → flip-flop D
$Q_0$ → flip-flop D
(or any others)

5 – Selection of flip-flops

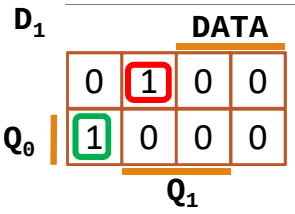
| D | $Q_{n+1}$ |
|---|---|
| 0 | 0 |
| 1 | 1 |

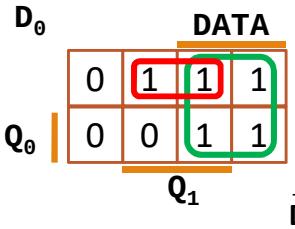
| $Q_n$ → | $Q_{n+1}$ | D |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| Current state | Next state | | Flip-flop inputs | | Outputs |
|---|---|---|---|---|---|
| | DATA=0 | DATA=1 | DATA=0 | DATA=1 | |
| $(Q_1Q_0)_n$ | $(Q_1Q_0)_{n+1}$ | $(Q_1Q_0)_{n+1}$ | $D_1D_0$ | $D_1D_0$ | OUT |
| $(S_0)$ 00 | 00 | 01 | 00 | 01 | 0 |
| $(S_1)$ 01 | 10 | 01 | 10 | 01 | 0 |
| $(S_2)$ 10 | 11 | 01 | 11 | 01 | 0 |
| $(S_3)$ 11 | 00 | 01 | 00 | 01 | 1 |

6 – Determining flip-flops input functions and output functions

# Sequence detector for 100: using Moore state machine




$D_1 = /DATA.Q_1./Q_0 + /DATA./Q_1.Q_0$

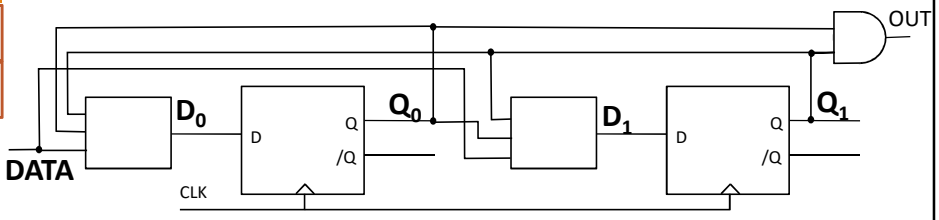
$D_0 = Q_1./Q_0 + DATA$

$OUT = Q_1.Q_0$

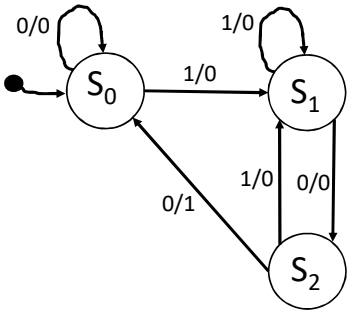
7 – Simplification of functions (namely using Karnaugh maps)

8 – Schematic

9 – Simulation, implementation and tests




# Sequence detector for 100: using Mealy state machine

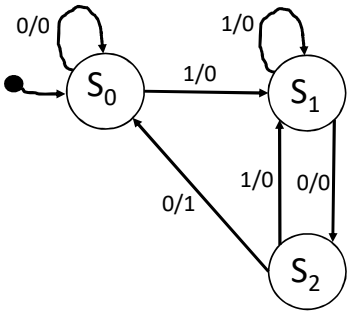



$S_0$ – out of sequence

$S_1$ – detected first bit of the sequence

$S_2$ – detected two first bits of the sequence

# Sequence detector for 100: using Mealy state machine



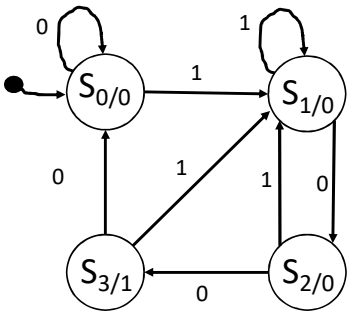

| Current state | Next state | | OUT | |
|---|---|---|---|---|
| | DATA=0 | DATA=1 | DATA=0 | DATA=1 |
| $S_0$ | $S_0$ | $S_1$ | 0 | 0 |
| $S_1$ | $S_2$ | $S_1$ | 0 | 0 |
| $S_2$ | $S_0$ | $S_1$ | 1 | 0 |

1 – State diagram

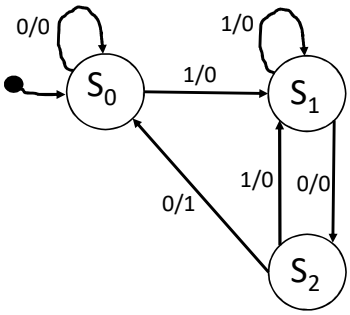
2 – State transition table and outputs

# Sequence detector for 100: comparing Moore and Mealy state machines

Moore specification




Mealy specification



# Development methodology for synchronous state machines

1 – Draw the state diagram describing desired behavior

2 – Draw state transition table and outputs

3 – State encoding

4 – Encoded state transition table and outputs

5 – Selection of flip-flops

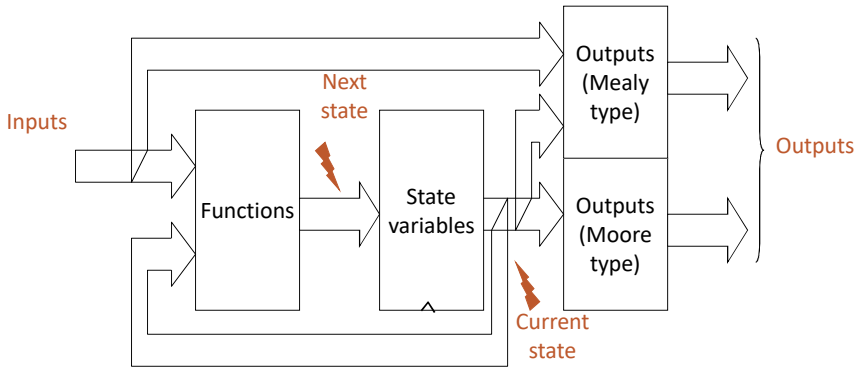
6 – Determining flip-flops input functions and output functions

7 – Simplification of functions (namely using Karnaugh maps)

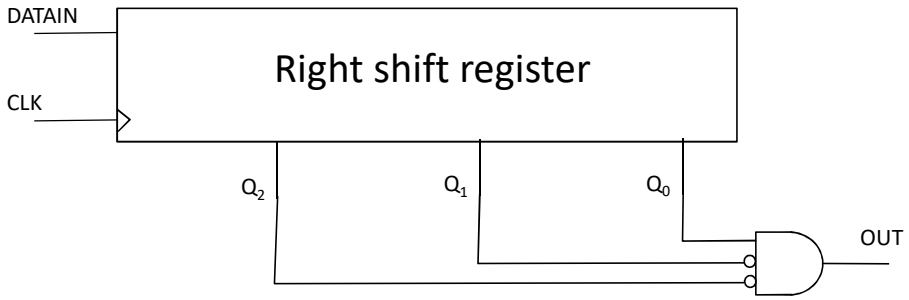
8 – Schematic

9 – Simulation, implementation and tests

# Reference architecture for execution/implementation



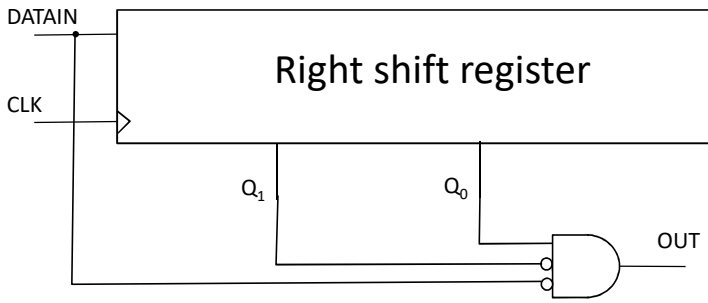
# Sequence detector for 100 using shift register




Moore output

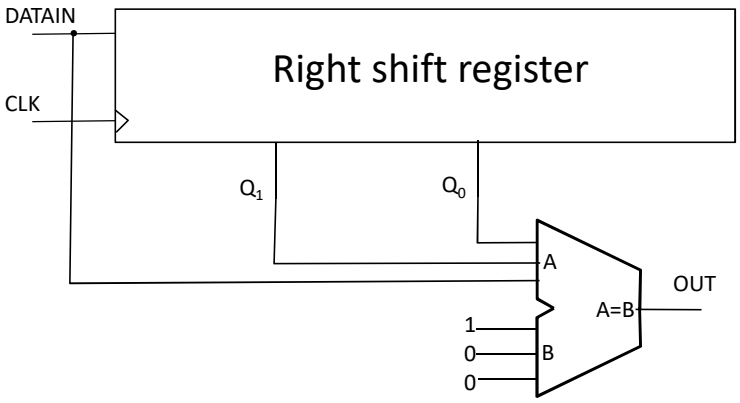


# Sequence detector for 100 using shift register




Mealy output

# Sequence detector for 100 using shift register




Mealy output