CHAPTER 7: SEQUENTIAL CIRCUITS – FLIP-FLOPS, REGISTERS, AND COUNTERS

What will we learn?

- Logic circuits that can store information
  - Latches, which store *a single bit*
  - Flip-Flops, which store *a single bit*
  - Registers, which store *multiple bits*
- Shift registers
- Counters
- Design Examples

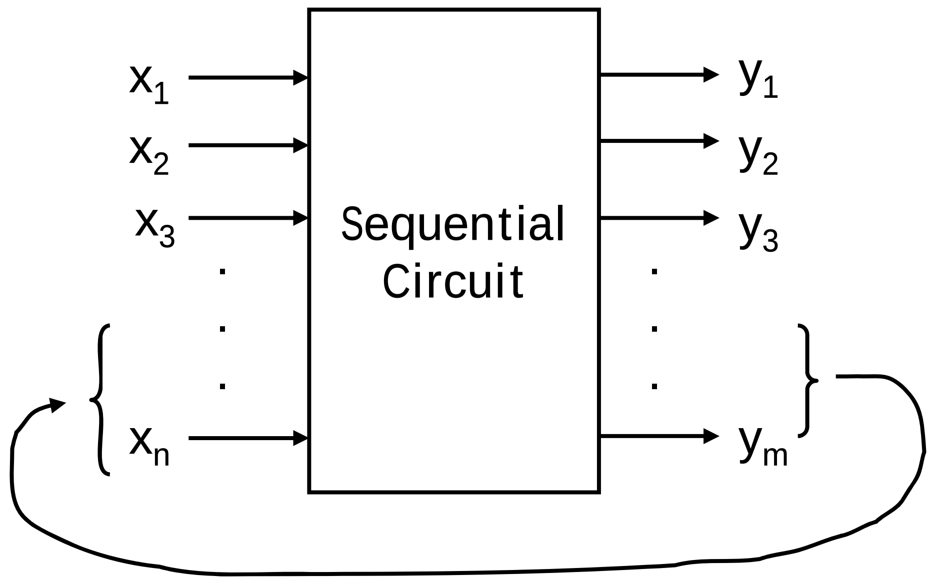
Sequential Circuits

- Combinational Circuits
  - circuits without feedback
  - output = f (current inputs)

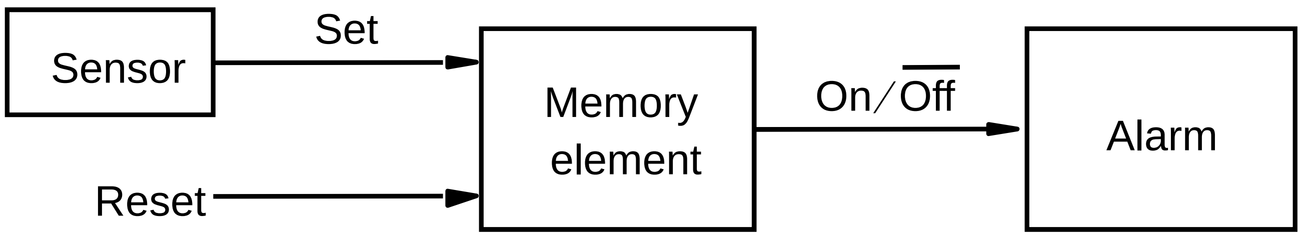
- Sequential Circuits
  - circuits with feedback
  - output = f (current inputs, past inputs, past outputs)
  - how can we feed the past inputs and outputs into the circuits?
    - basis for building "memory" into logic circuits

Circuits with feedback

- How to control feedback?
  - what stops values from cycling around endlessly



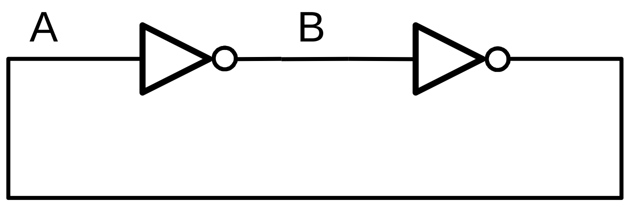
Control of an alarm system



- the simplest case of a sequential circuit
  - Alarm is on when the sensor generates the "Set" signal in response to some undesirable events
  - Once the alarm is on, it can only be turned off manually through a reset button
- Memory is needed to remember that the alarm has to be active until the reset signal arrives

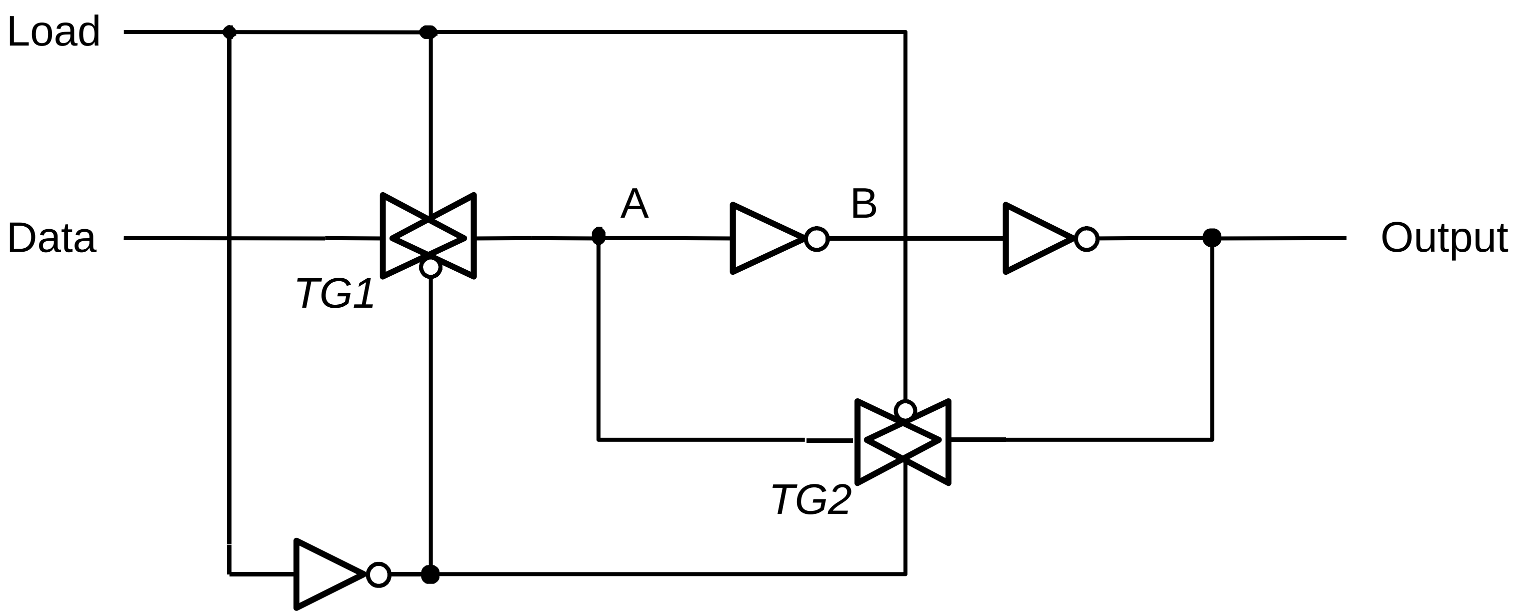
A simple memory element

- The most rudimentary memory element
  - Two inverters form a static memory cell
    - Assume A=0 and B=1, then the below circuit will maintain these values indefinitely (as long as it has power applied)
    - The state is defined by the value of the memory cell
      - Two states



- How to get a new value into the memory cell?
  - selectively break feedback path
  - load new value into cell

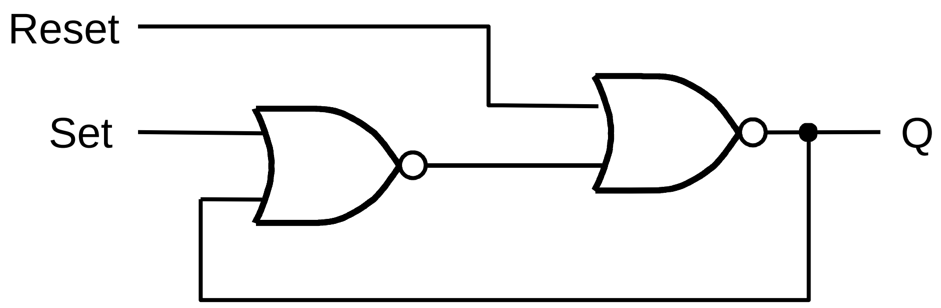
A controlled memory element



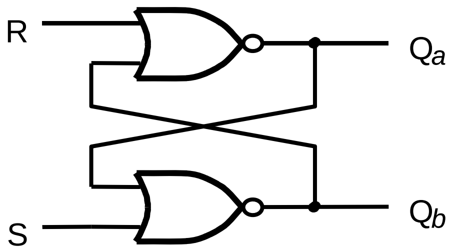
A memory element with NOR gates

- Construct a memory cell using ordinary logic gates
  - Two NOR gates are connected in cross-coupled style
  - Basic Latch

- Two inputs
  - Set
  - Reset



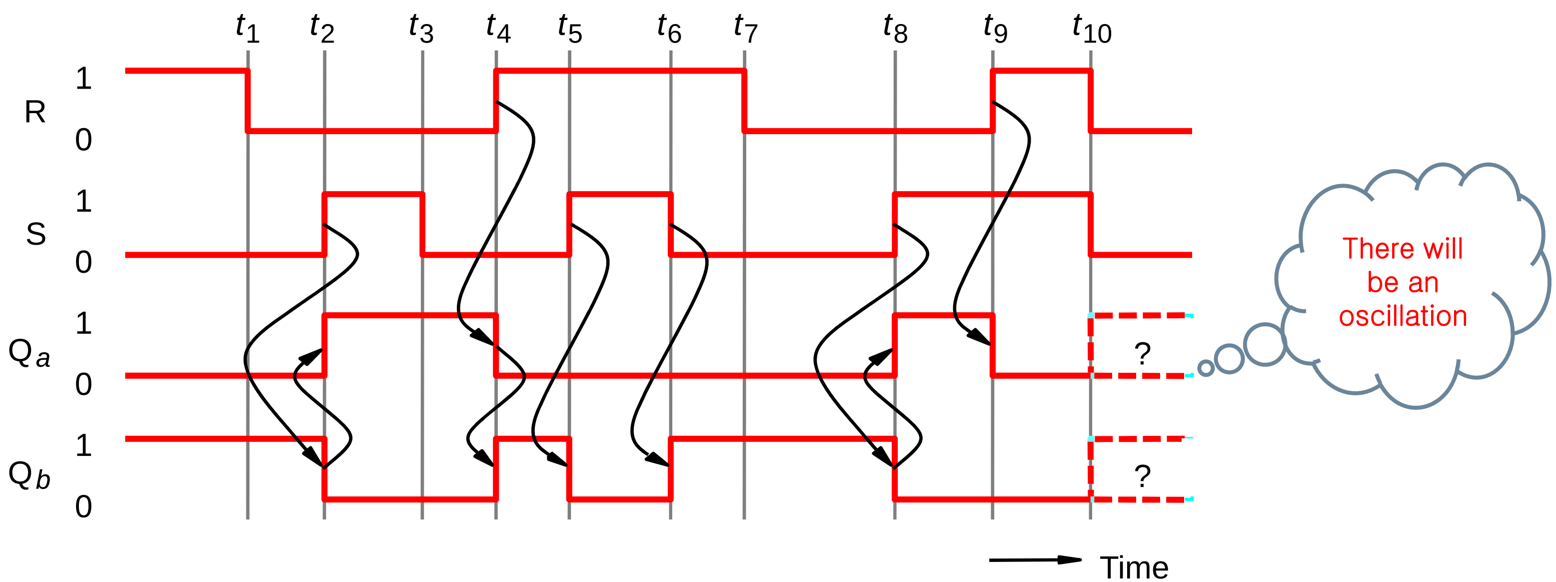
A basic latch built with NOR gates



(a) Circuit

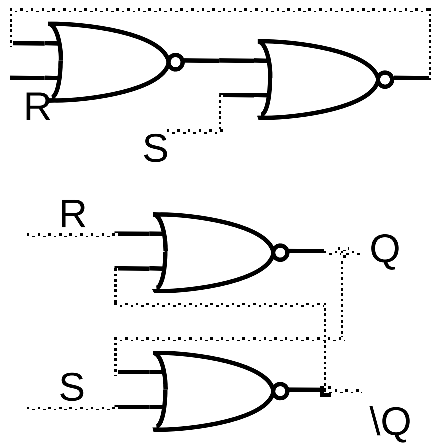
S	R	Q_a	Q_b	
0	0	0/1	1/0	(no change)
0	1	0	1	
1	0	1	0	
1	1	0	0	

(b) Truth table or *characteristic table*

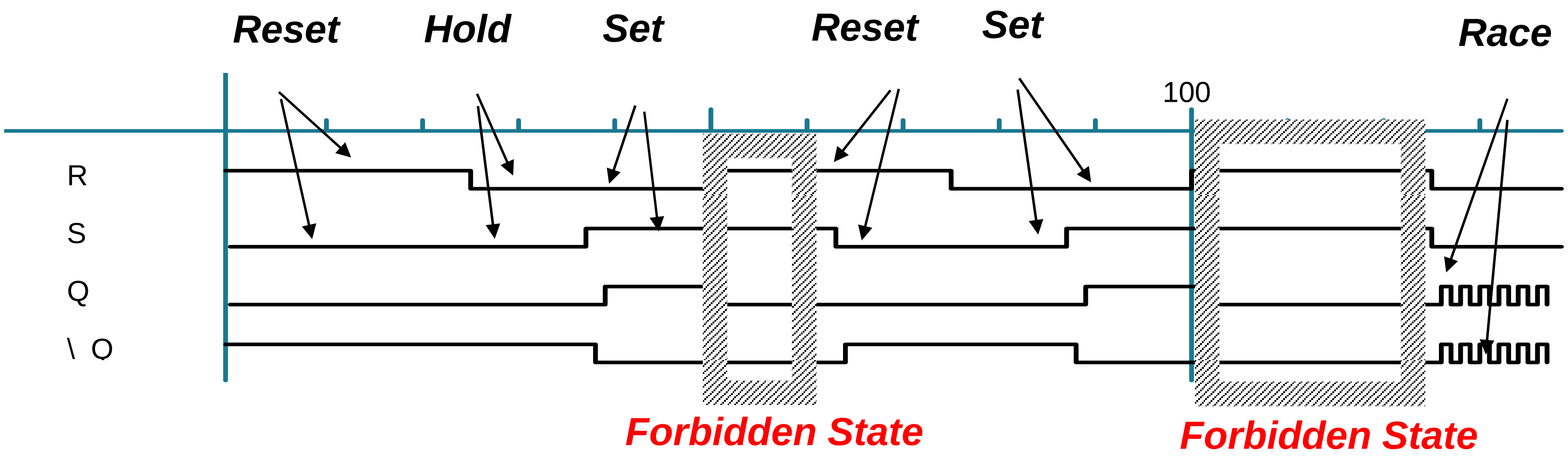


(c) Timing diagram

Timing Waveform



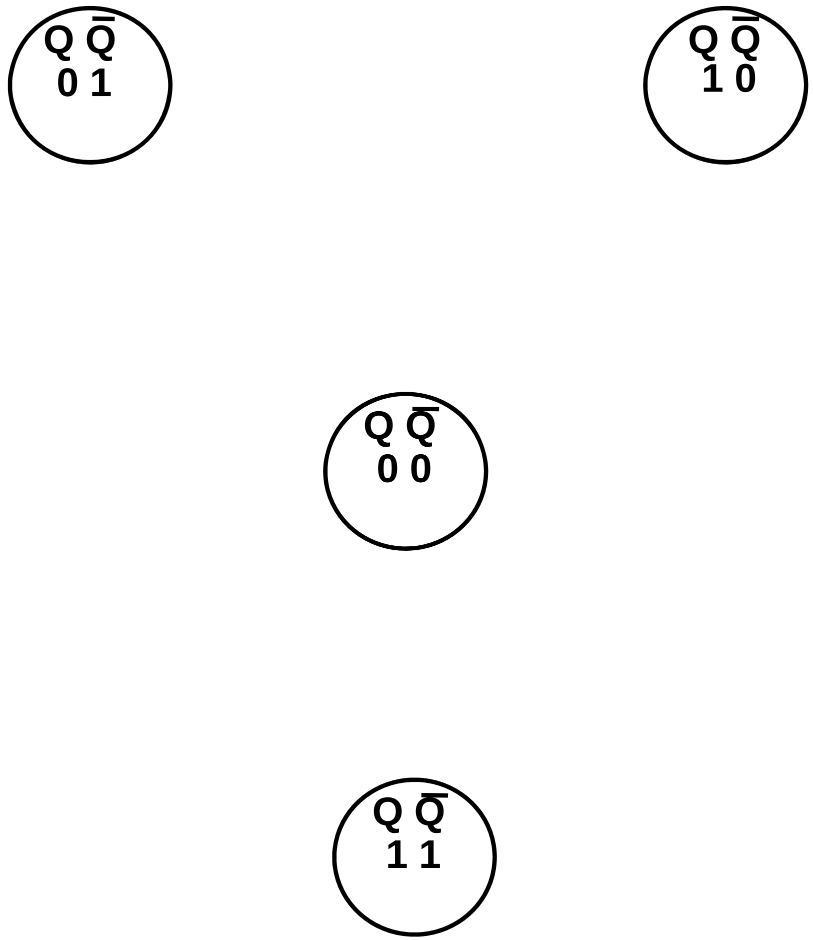
Timing Waveform



State Behavior of R-S Latch

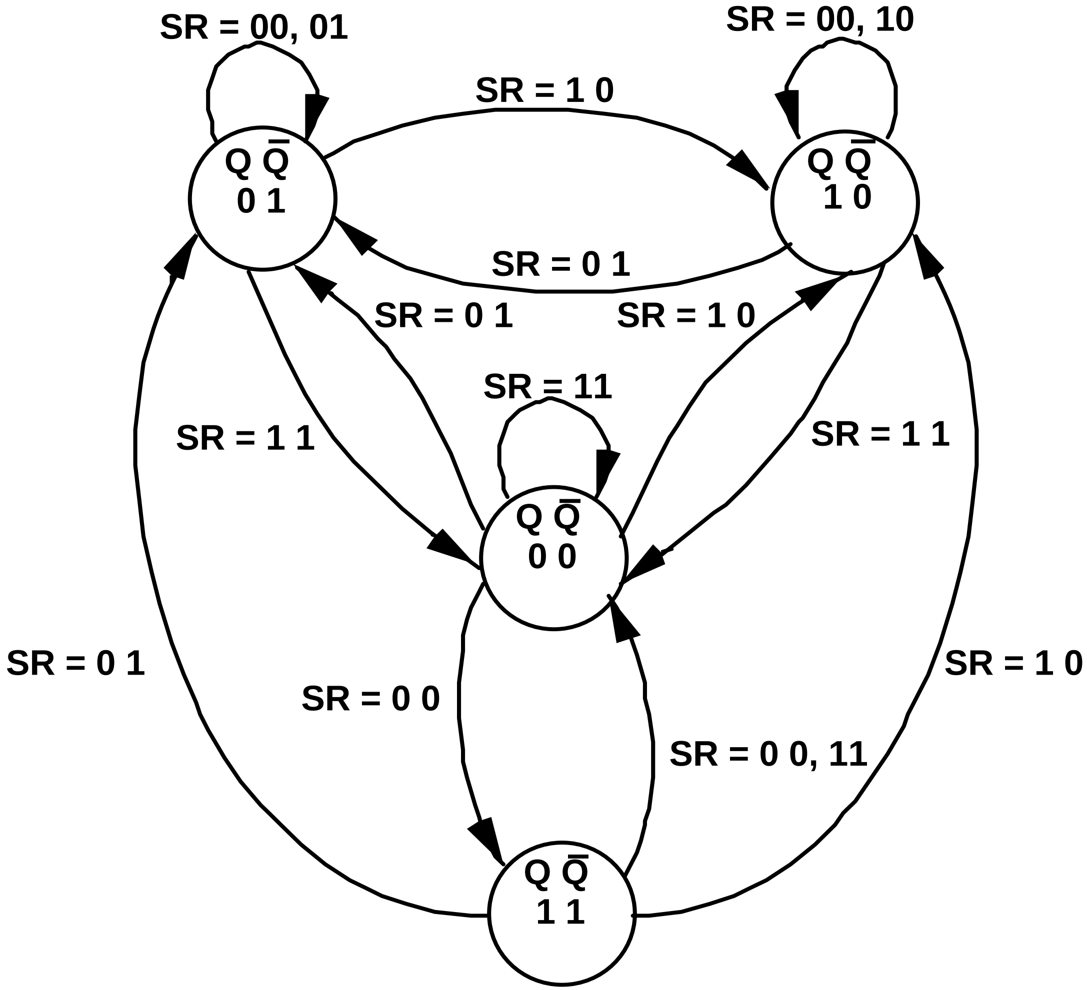
S	R	Q
0	0	hold
0	1	0
1	0	1
1	1	unstable

Truth Table Summary of R-S Latch Behavior

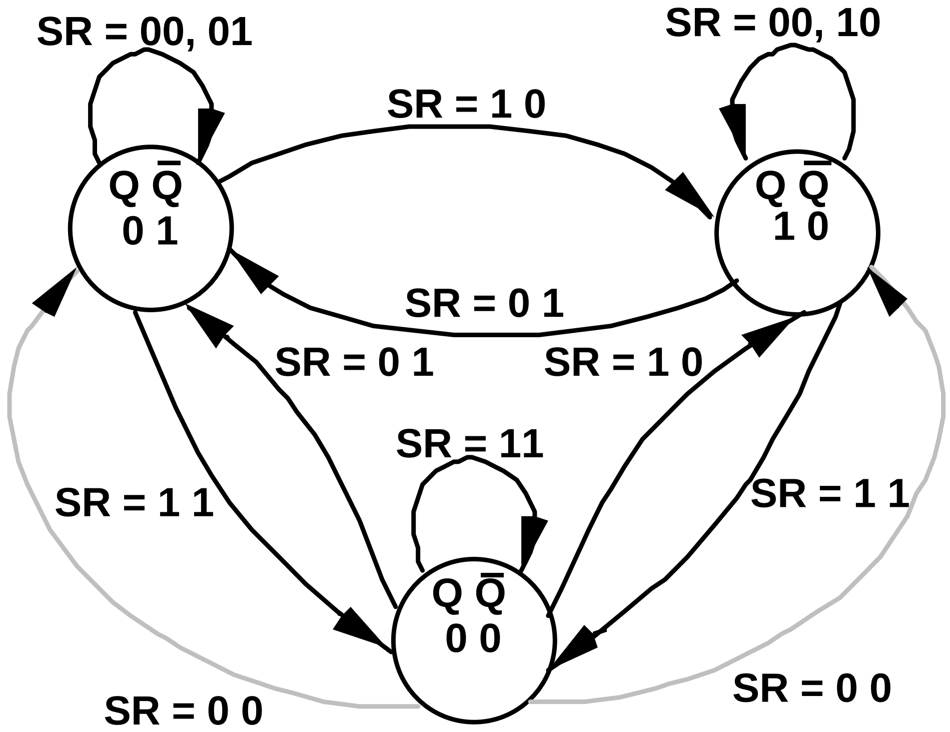


Theoretical R-S Latch State Diagram

- State Diagram
  - state: possible values
  - transitions: changes based on inputs



Observed R-S Latch Behavior



Very difficult to observe R-S Latch in the 1-1 state

Ambiguously returns to state 0-1 or 1-0

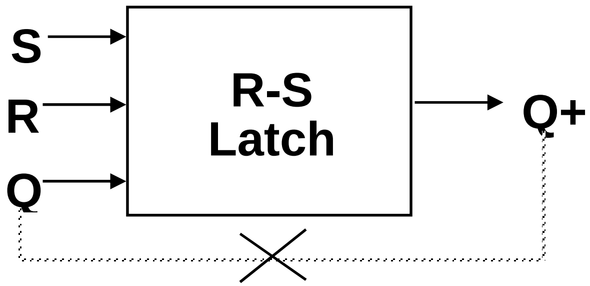
A so-called "race condition"

R–S Latch Analysis

Truth Table:
Next State = F(S, R, Current State)

R-S Latch Revisited

S	R	Q_t	Q+	
0	0	0	0	hold
0	0	1	1	
0	1	0	0	reset 0
0	1	1	0	
1	0	0	1	set 1
1	0	1	1	
1	1	0	x	not allowed
1	1	1	x	



Derived K-Map:

S (columns 11, 10); R (columns 01, 11)

Q(t) \ SR	00	01	11	10
0	0	0	X	1
1	1	0	X	1

Characteristic Equation:

$$Q+ = S + \overline{R}\, Q_t$$

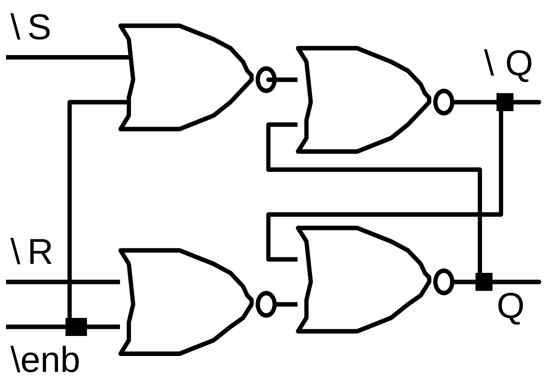
Problems of R-S Latch

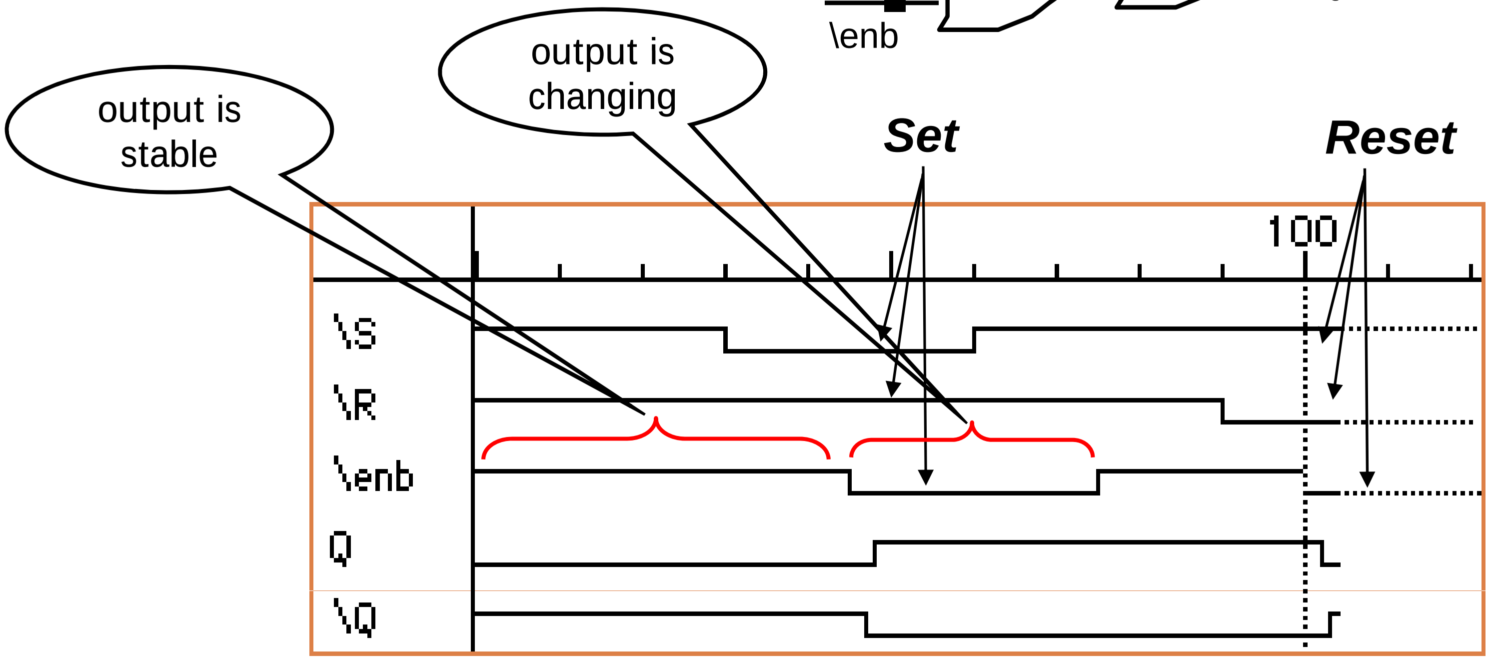
- *The slightest glitch on R or S* could cause change in value stored
  - R-S Latch has transparent outputs
    - Transparent outputs : when the memory element's outputs immediately change in response to input changes

- Want to control *when R and S inputs have effect on value stored*
  - Enable Signal (or clock signal)
    - R and S inputs are active *only when Enable = 1*
  - Gated Latches or Level sensitive latches

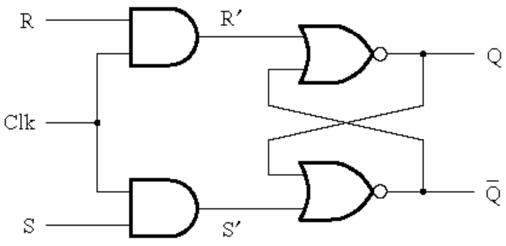
Gated SR latch

- *Control when R and S inputs matters*
  - the latch can be modified to respond to the input signal S and R only when Enable =1





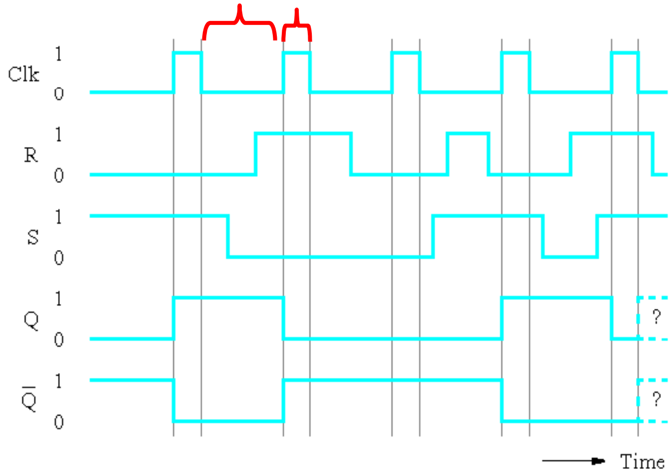
Gated SR Latch



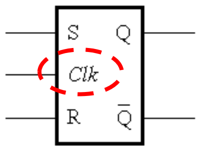
(a) Circuit

Clk	S	R	Q(t + 1)
0	x	x	Q(t) (no change)
1	0	0	Q(t) (no change)
1	0	1	0
1	1	0	1
1	1	1	x

(b) Characteristic table

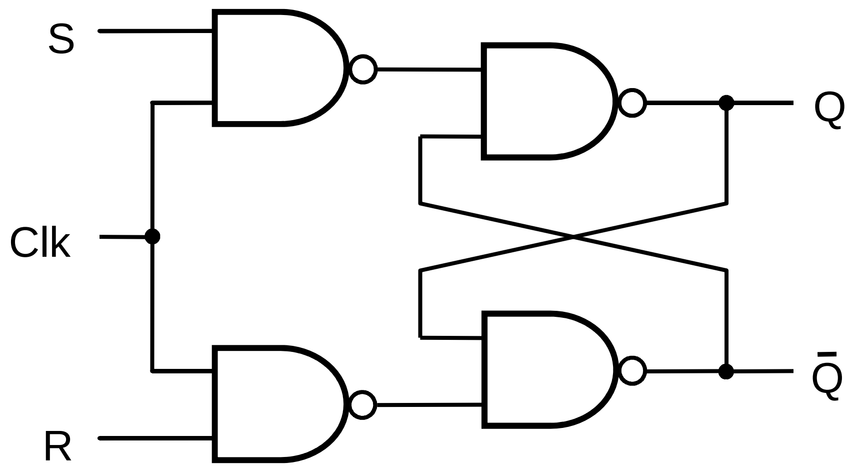


(c) Timing diagram



(d) Graphical symbol

Gated SR latch with NAND gates



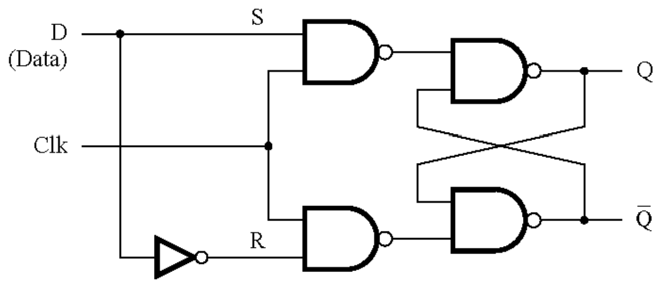
Problems of the Gated S/R Latches

- 1. Forbidden State and Race condition
  - How to eliminate the forbidden state and race condition
  - When S=R=1, $Q = \overline{Q} = 0$ (forbidden state)
  - Oscillation (Race condition)
    - D-type Latch
    - JK-Latch (toggling)
      - The output toggles forever when J=K=1

- 2. When cascading level-sensitive Latches
    - Master/Slave F/F's
    - Edge-triggered F/F's

1. How to eliminate the forbidden state?

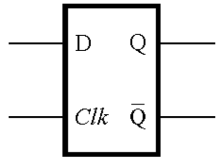
- Gated D-latch
  - eliminate the troublesome situation where $S=R=1$



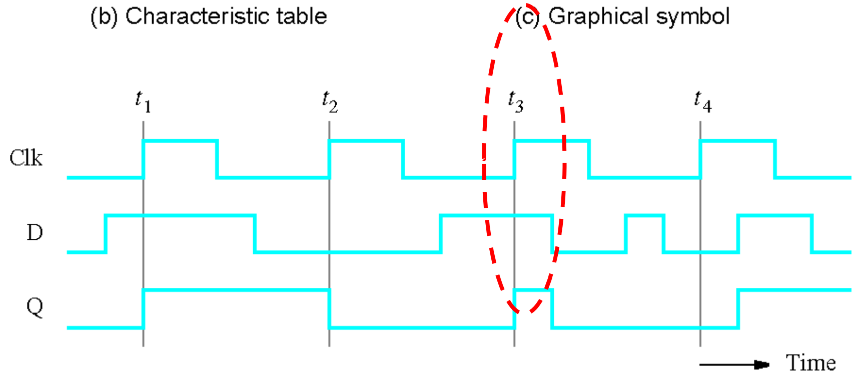
(a) Circuit

Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

(b) Characteristic table



(c) Graphical symbol

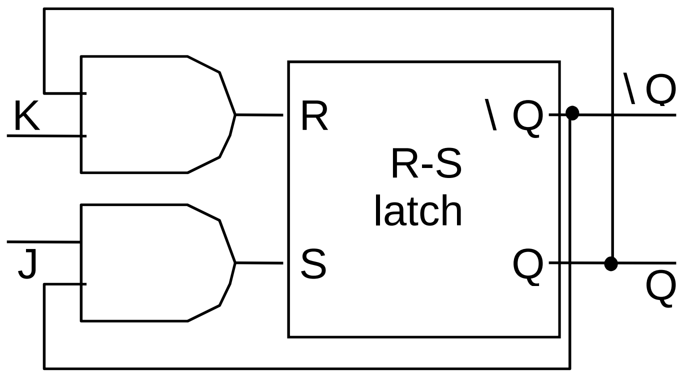


(d) Timing diagram

How to eliminate the forbidden state? cont'd

Idea: use output feedback to guarantee that R and S are never both one

J, K both one yields toggle



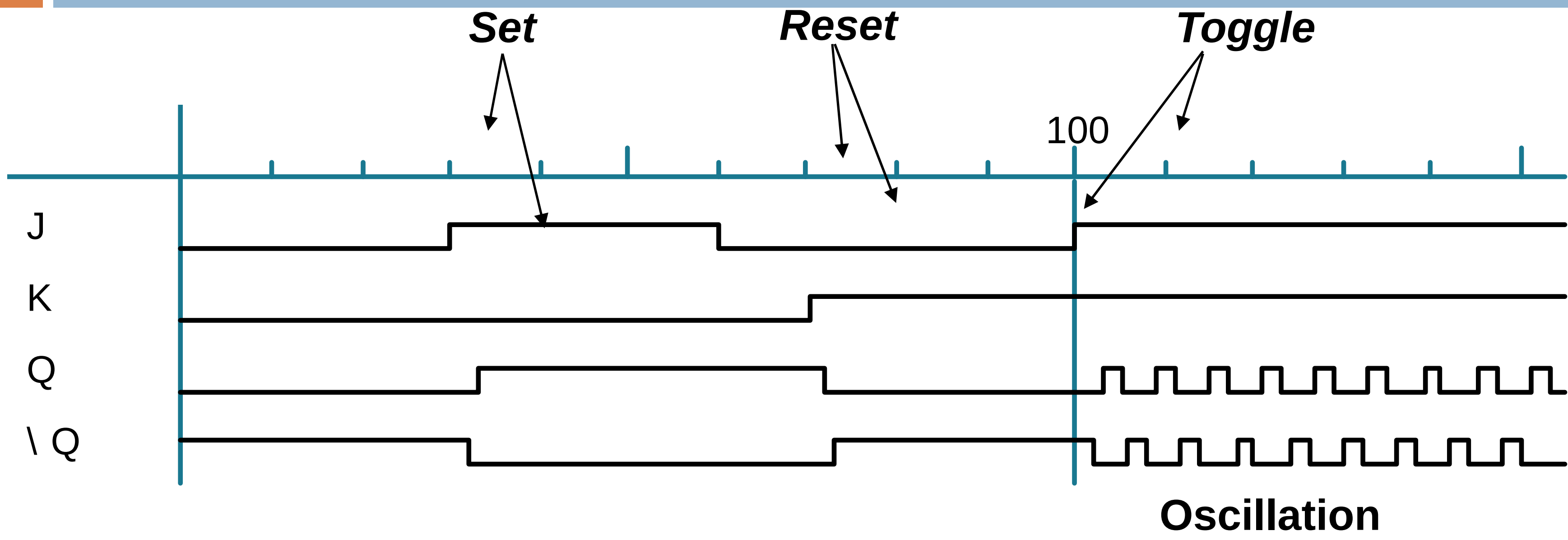
J-K Latch

J(t)	K(t)	Q(t)	Q+		
0	0	0	0	hold	
0	0	1	1		
0	1	0	0	reset	0
0	1	1	0		
1	0	0	1	set	1
1	0	1	1		
1	1	0	1	toggle	
1	1	1	0		

Characteristic Equation:

$$Q+ = Q\,\overline{K} + \overline{Q}\,J$$

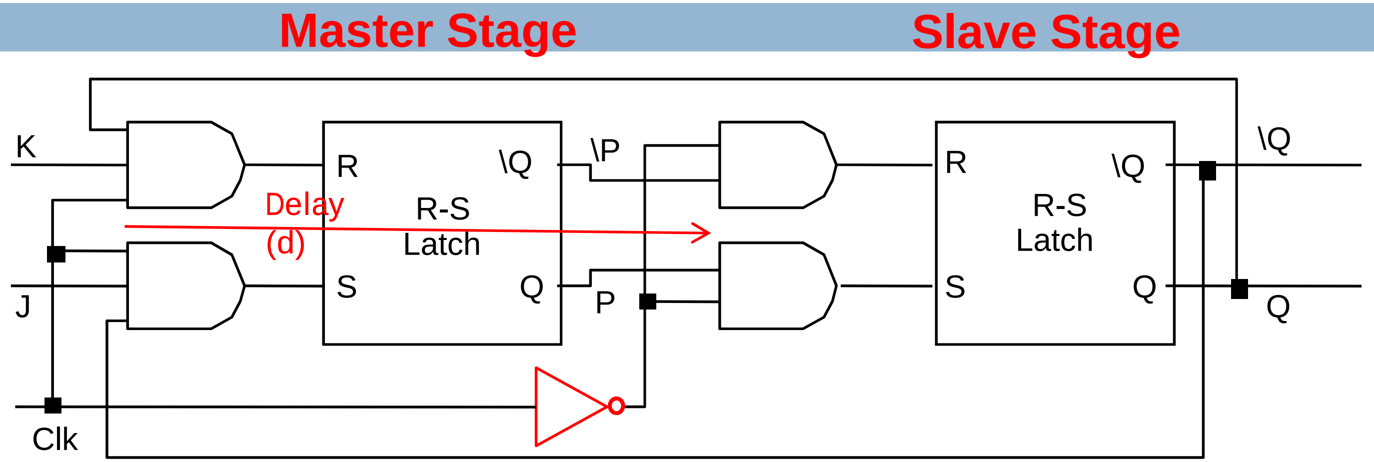
J-K Latch: Toggles forever in the toggle mode



Toggle Correctness: Single State change per clocking event

Solution: Master/Slave Flipflop

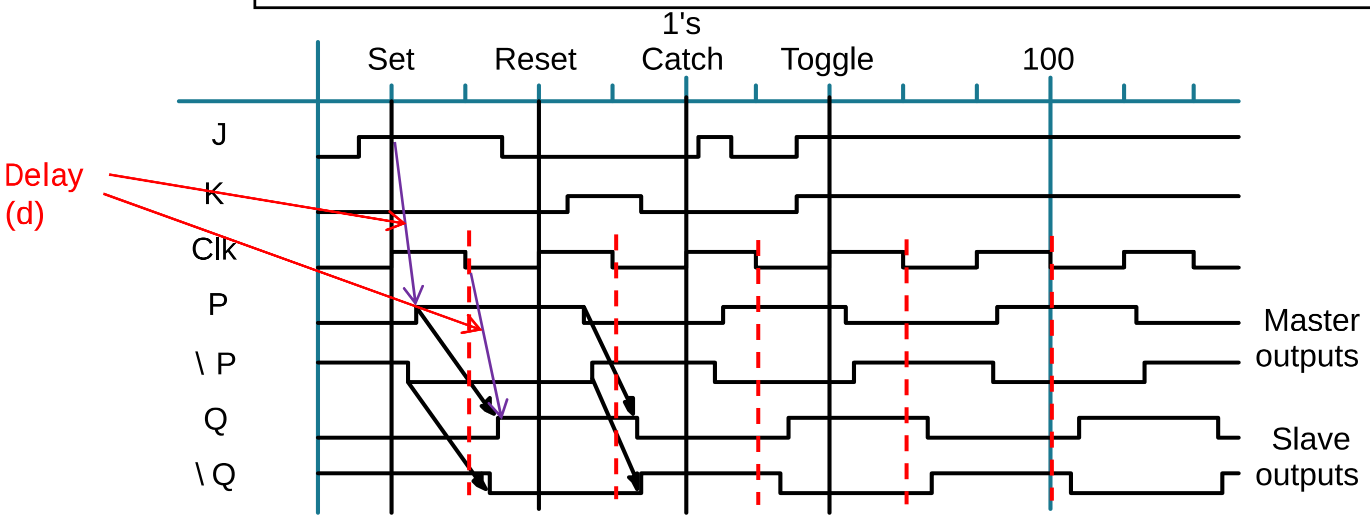
Master/Slave J-K Flipflop



Sample inputs while clock high

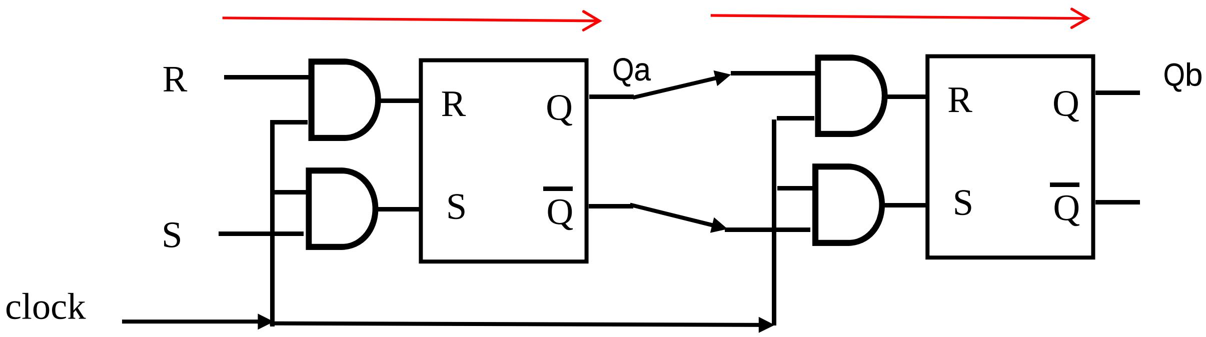
Sample inputs while clock low

Uses time to break feedback path from outputs to inputs!



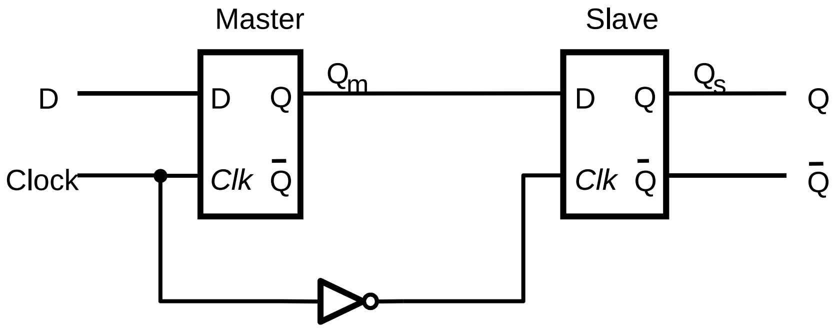
Correct Toggle Operation

2. When cascading Latches

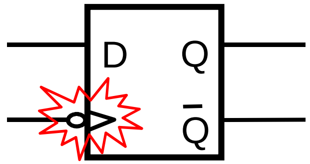


- How to stop changes from racing through chain?
  - need to be able to control flow of data from one latch to the next
  - move one latch per clock period
  - have to worry about logic between latches that is too fast

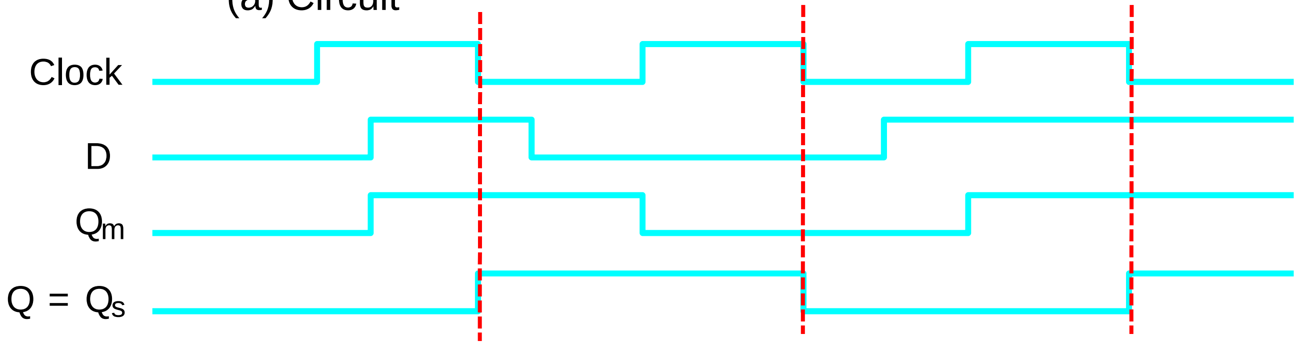
Master/Slave D Flip-Flop



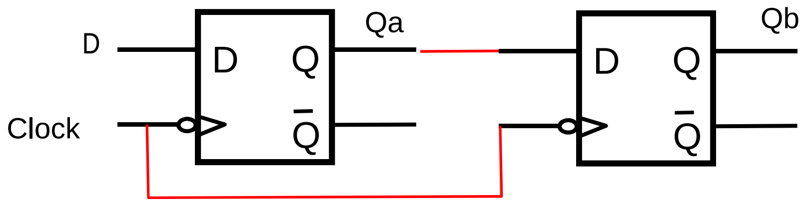
(a) Circuit



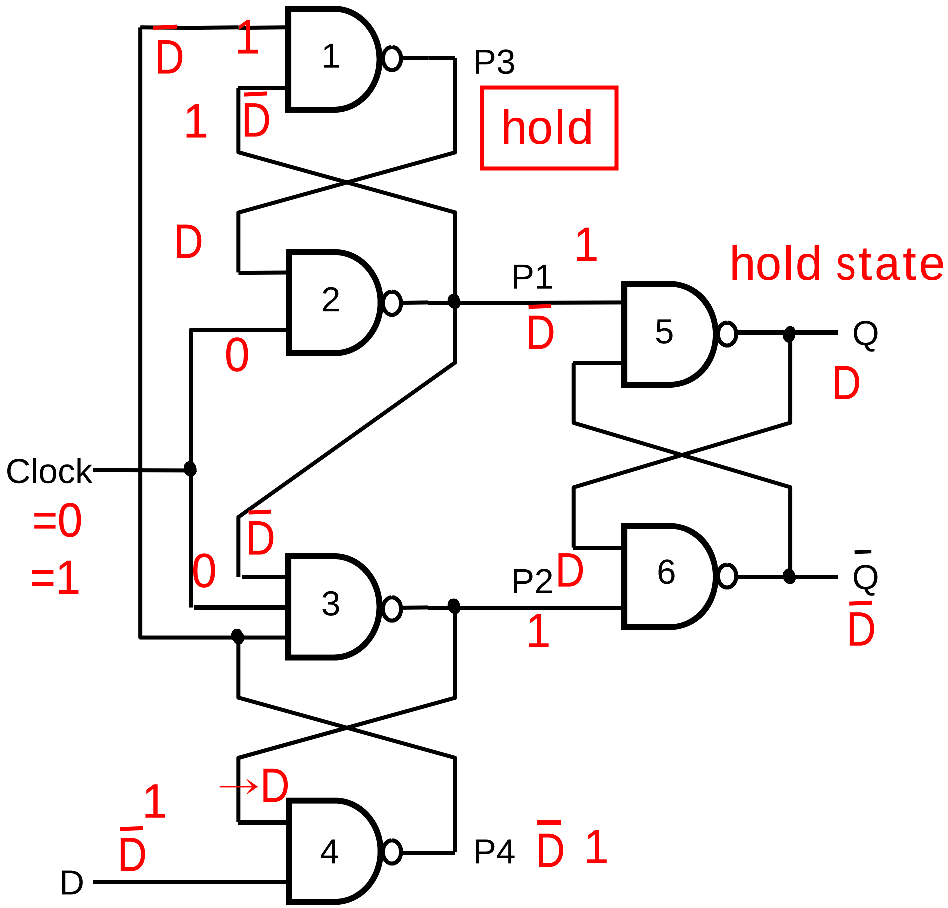
(c) Graphical symbol



(b) Timing diagram

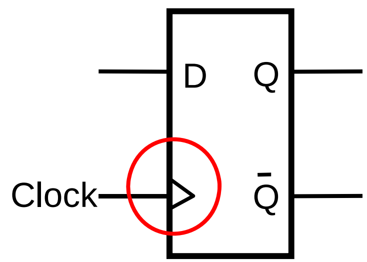


Positive-edge-triggered D flip-flop



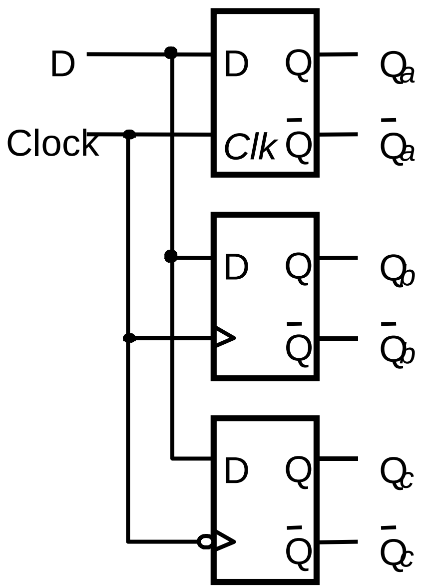
(a) Circuit

- Clock = 0
  - Output of gate 2 and 3 are high -> P1,P2 high
  - Output is maintained
- Clock = 1
  - P3 and P4 are transmitted through gate 2 and 3 to cause P1 = D' and P2 = D.
  - This sets Q = D and Q = D'
- P3 and P4 must be stable when the clock goes from 0 to 1.
- After that, the changes in D have no effect.

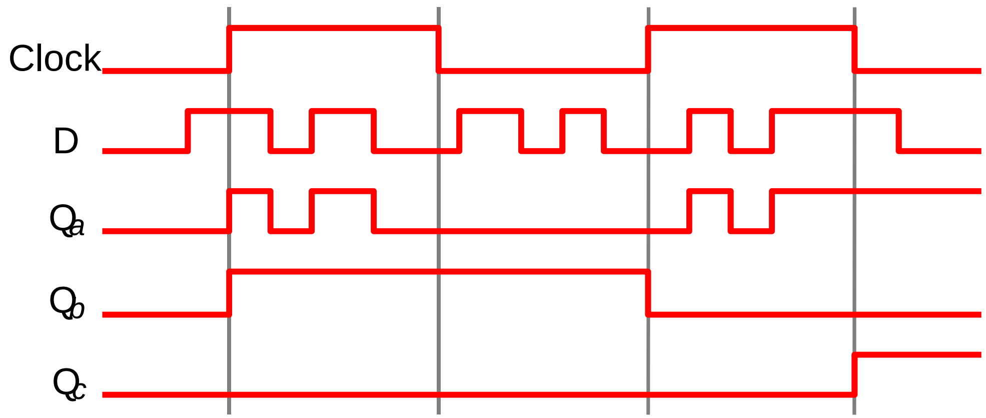


(b) Graphical symbol

Comparison of level-sensitive and edge-triggered D storage elements

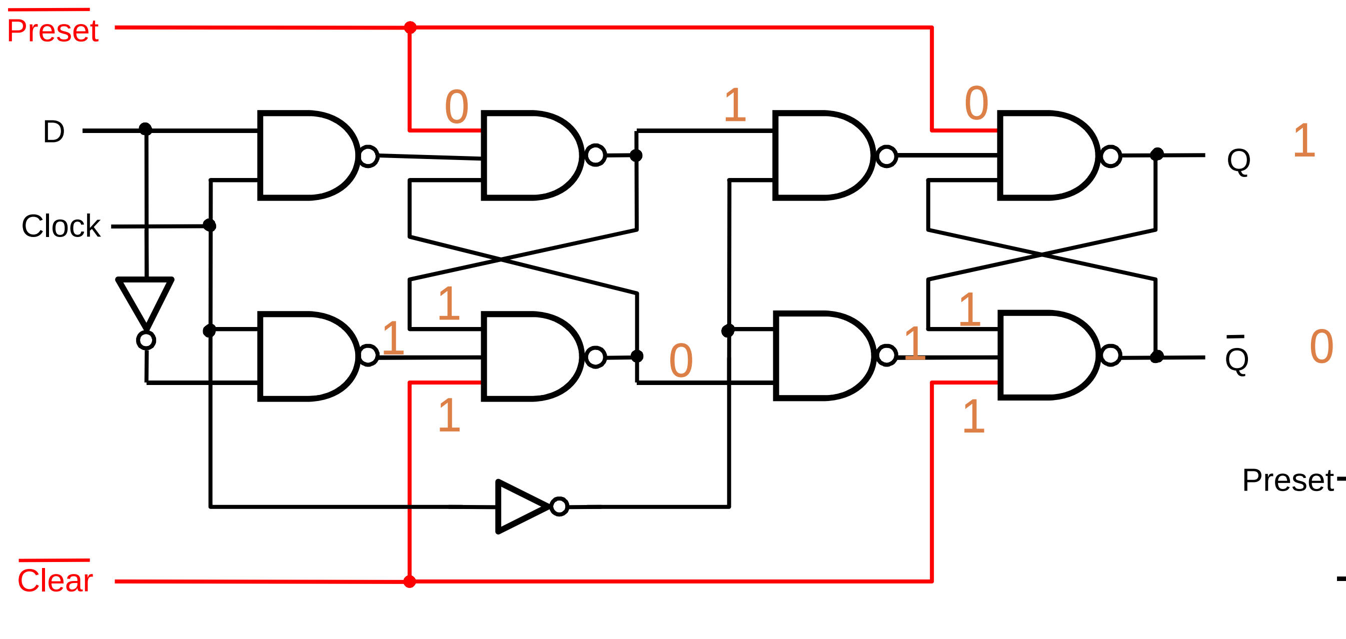


(a) Circuit

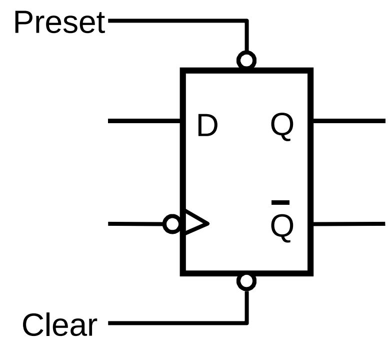


(b) Timing diagram

Master-slave D flip-flop with Clear and Preset

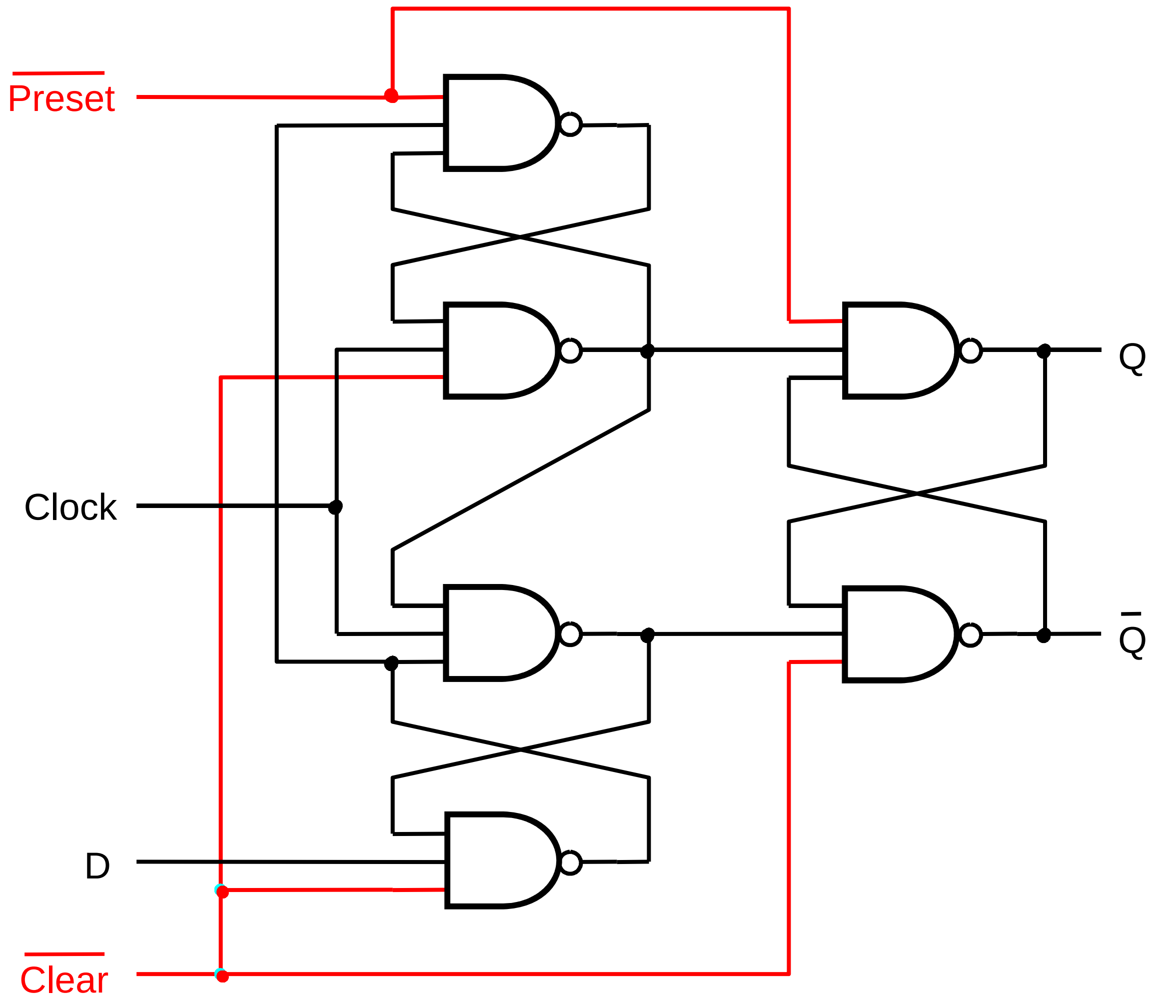


(a) Circuit

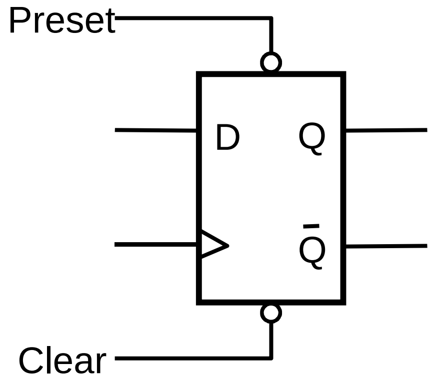


(b) Graphical symbol

Positive-edge-triggered D flip-flop with Clear and Preset

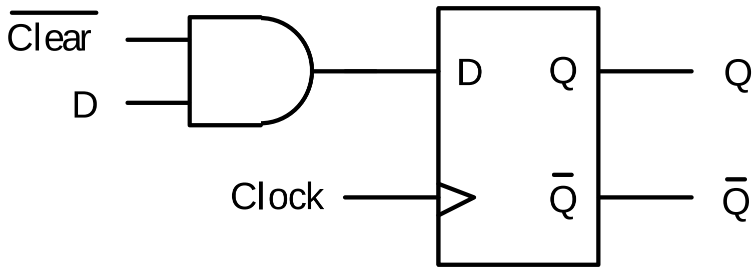


(a) Circuit

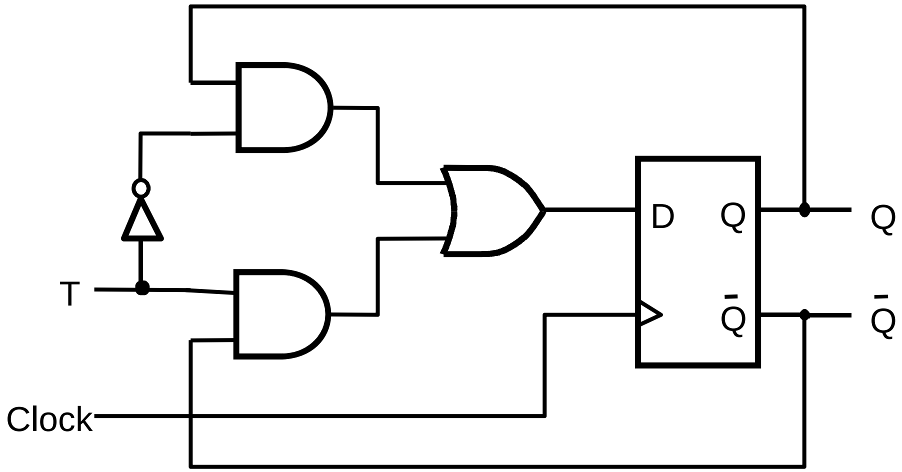


(b) Graphical symbol

Synchronous reset for a D flip-flop



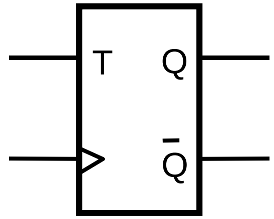
T Flip-Flop



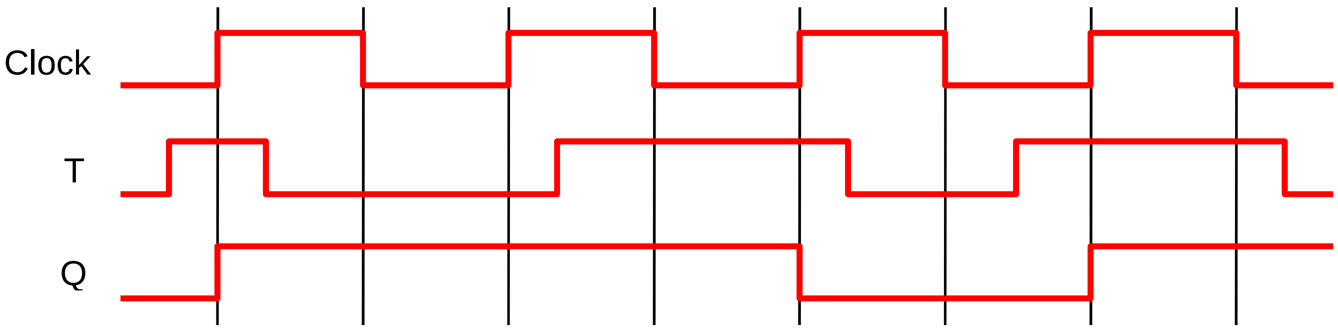
(a) Circuit

T	Q(t + 1)
0	Q(t)
1	$\overline{Q}(t)$

(b) Truth table

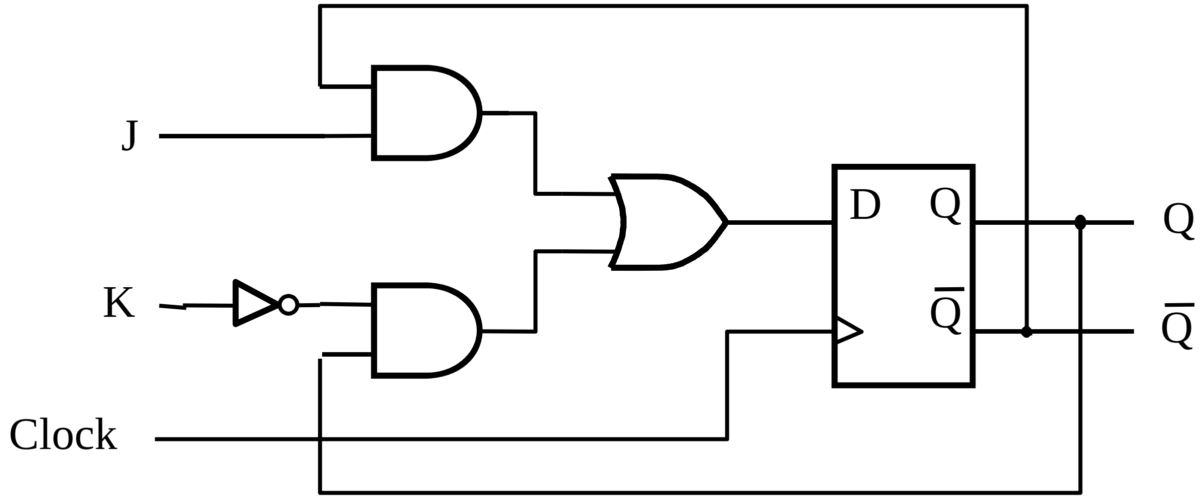


(c) Graphical symbol



(d) Timing diagram

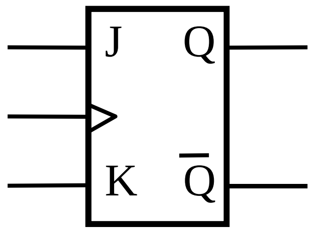
Realizing JK flip-flop with D flip-flop



(a) Circuit

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}(t)$

(b) Truth table



(c) Graphical symbol

Last time

- Memory Cell
  - SR Latch
- Problems of SR Latches
  - Glitch problems
    - Transparent output – the memory element's outputs immediately change in response to input changes

    → Gated SR Latches *(Enable signal or clock)*
  - Another problems
    - Forbidden state and racing problem → D-latches, JK-latches
    - When cascading latches
      - How to stop changes from racing through chain?

    → Mater slave F/Fs and Edge triggered F/Fs *(clock signal)*

    → Memory elements change their states in response to a clock signal

    → We call these *Synchronous systems*

Today

- Timing Methodologies
  - To guarantee the correct operation when cascading the Memory blocks
- Comparison of Latches and F/Fs
- Registers – store multiple bits
  - Storage registers
  - Shift registers
- Counters – count events
  - Asynchronous counters
  - Synchronous counters

Timing Methodologies

- Set of rules for interconnecting components and clocks
  - When followed, guarantee proper operation of system

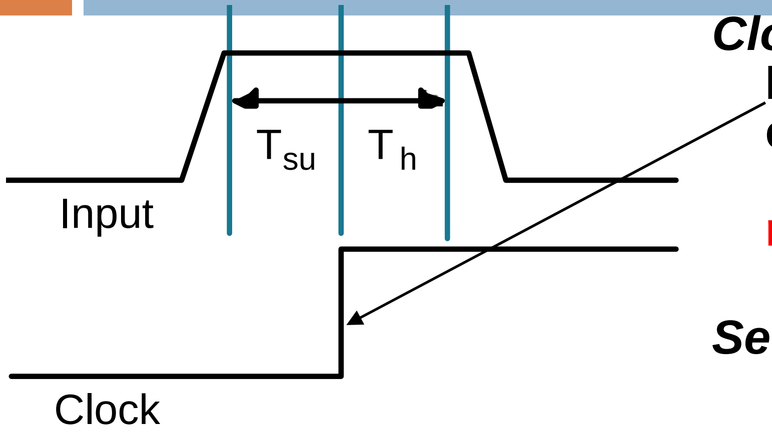
- Proper operation:

  (1) The correct inputs, with respect to time, are provided to the FFs

  (2) no FF changes more than once per clocking event

- Approach depends on building blocks used for memory elements
  - For systems with latches:
    - Narrow Width Clocking
    - Multiphase Clocking (e.g., Two Phase Non-Overlapping)
  - For systems with edge-triggered flip-flops:
    - Single Phase Clocking

Definition of Terms



Clock:

Periodic Event, causes state of memory element to change

rising *edge*, falling *edge*, high *level*, low *level*

Setup Time (Tsu)

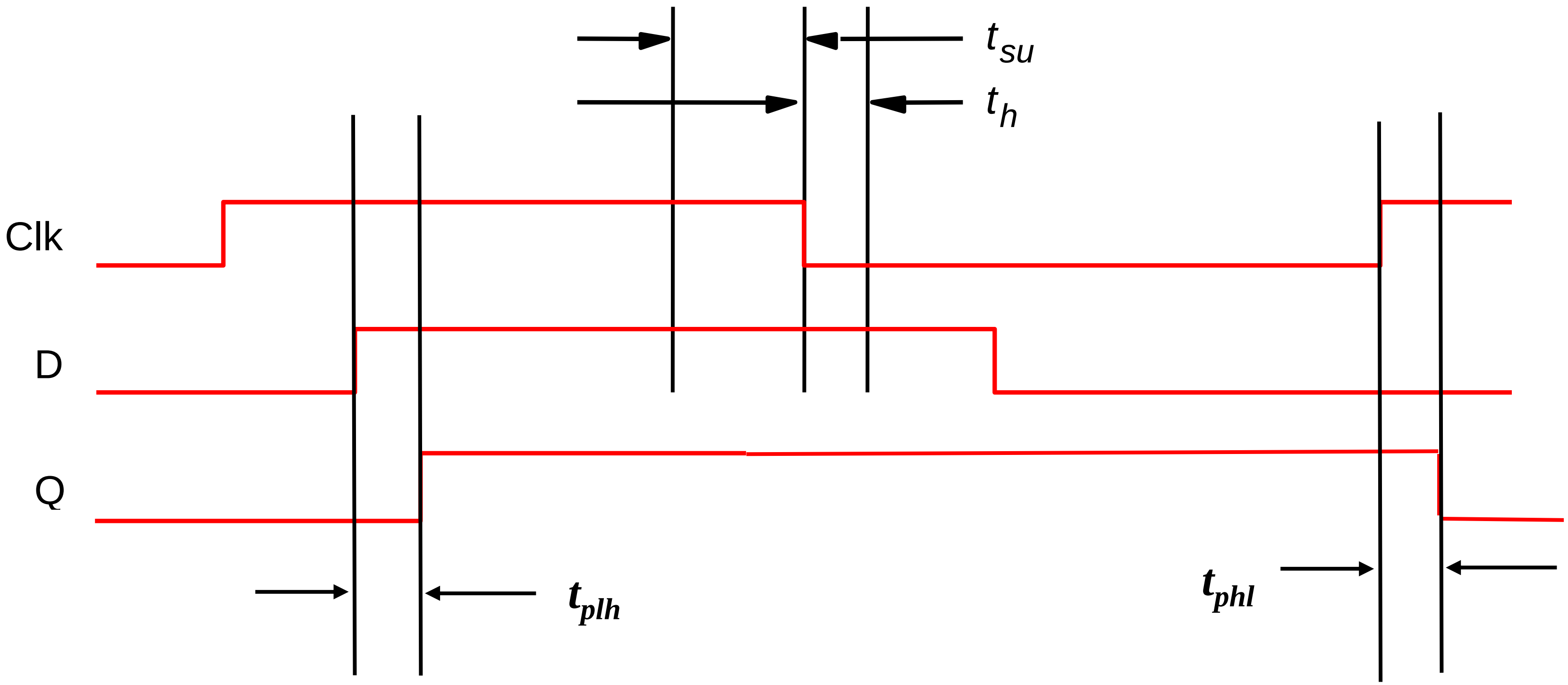
Minimum time before the clocking event by which the input must be stable

Hold Time (Th)

Minimum time after the clocking event during which the input must remain stable

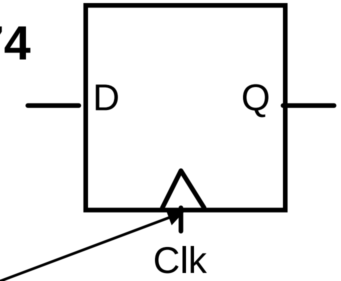
There is a timing "window" around the clocking event during which the input must remain stable and unchanged in order to be recognized

Setup and Hold times for Latches



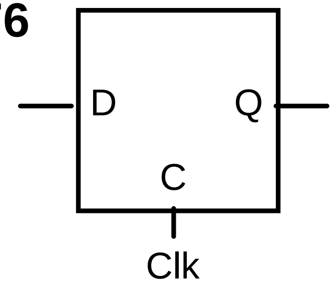
Comparison of latch and F/F

7474



Positive edge-triggered flip-flop

7476



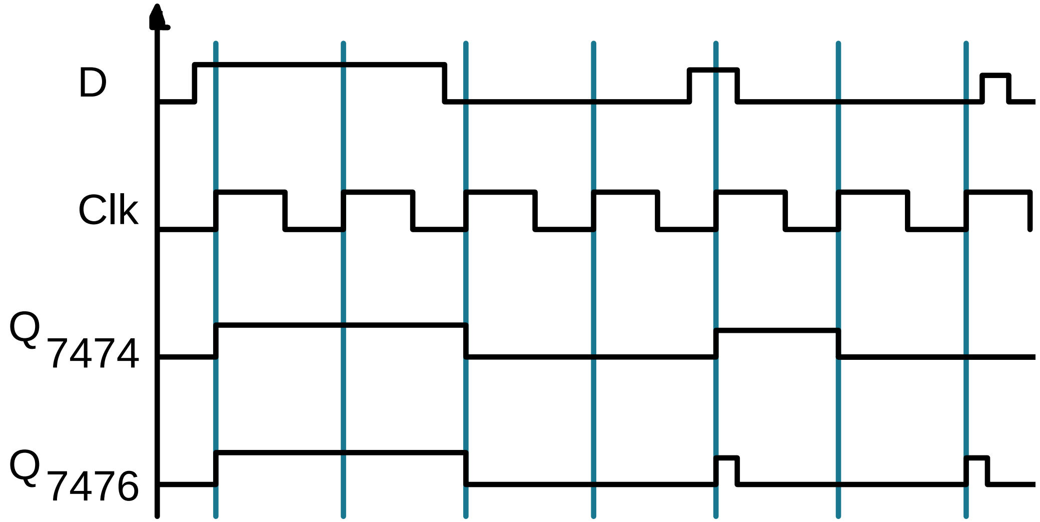
Level-sensitive latch

Bubble here for negative edge triggered device

Edge triggered **device sample inputs on the event edge**

Transparent latches **sample inputs as long as the clock is asserted**

Timing Diagram:



Behavior the same unless input changes while the clock is high

Comparison of latches and F/Fs

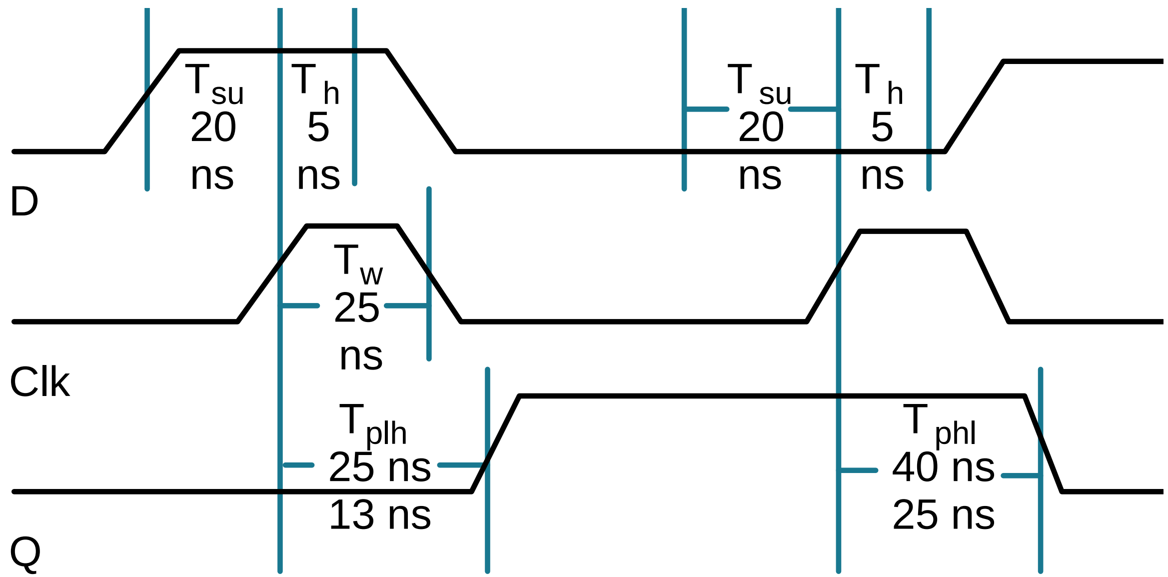
Input/Output Behavior of Latches and Flipflops

Type	When Inputs are Sampled	When Outputs are Valid
unclocked latch	always	propagation delay from input change
level sensitive latch	clock high (Tsu, Th around falling clock edge)	propagation delay from input change
positive edge flipflop	clock lo-to-hi transition (Tsu, Th around rising clock edge)	propagation delay from rising edge of clock
negative edge flipflop	clock hi-to-lo transition (Tsu, Th around falling clock edge)	propagation delay from falling edge of clock
master/slave flipflop	clock hi-to-lo transition (Tsu, Th around falling clock edge)	propagation delay from falling edge of clock

Typical Timing Specifications: Flipflops vs. Latches

74LS74 Positive Edge Triggered D Flipflop

- **Setup time**
- **Hold time**
- **Minimum clock width**
- **Propagation delays (low to high, high to low, max and typical)**

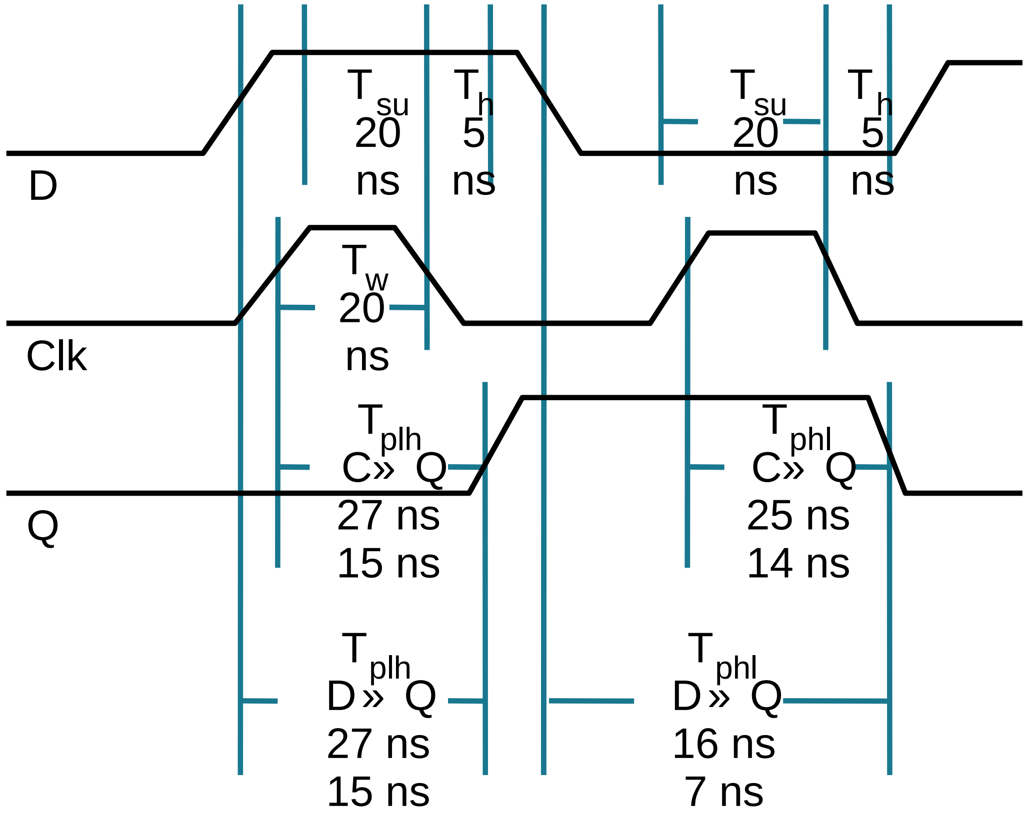


All measurements are made from the clocking event that is, the *rising edge* of the clock

Typical Timing Specifications: Flipflops vs. Latches

74LS76 Transparent Latch

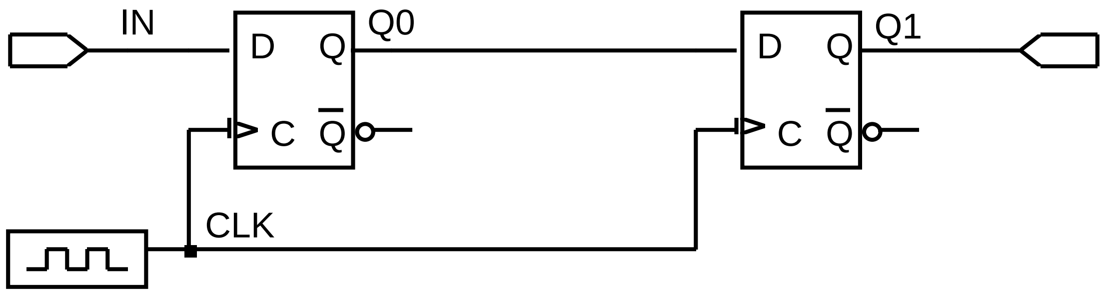
- **Setup time**
- **Hold time**
- **Minimum Clock Width**
- **Propagation Delays:**
  high to low, low to high,
  maximum, typical
  data to output
  clock to output



Measurements from falling clock edge or rising or falling data edge

Timing Methodologies

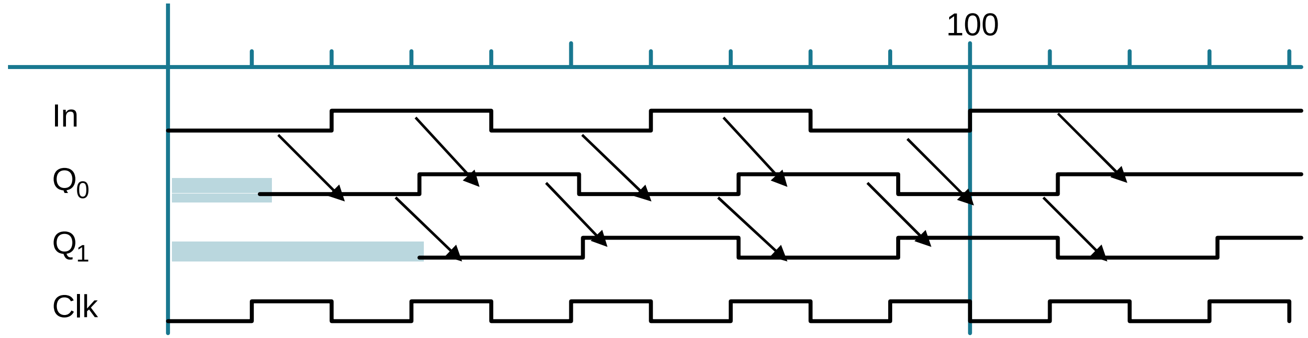
Two F/Fs are cascaded



New value to first stage while second stage obtains current value of first stage → Shift Register

Cascaded Flipflops and Setup/Hold/Propagation Delays

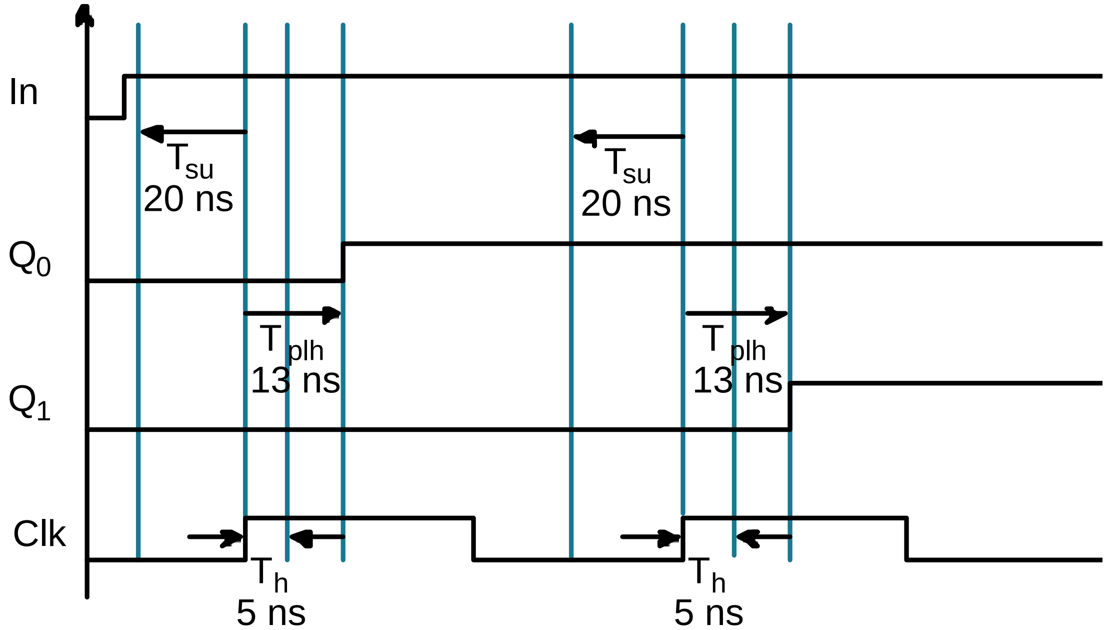
Correct Operation, assuming positive edge triggered FF



Cascaded Flipflops and Setup/Hold/Propagation Delays

Why this works:

- **Propagation delays far exceed hold times; Clock width constraint exceeds setup time**
- **This guarantees following stage will latch current value before it is replaced by new value**
- **Assumes infinitely fast distribution of the clock**



Timing constraints guarantee proper operation of cascaded components

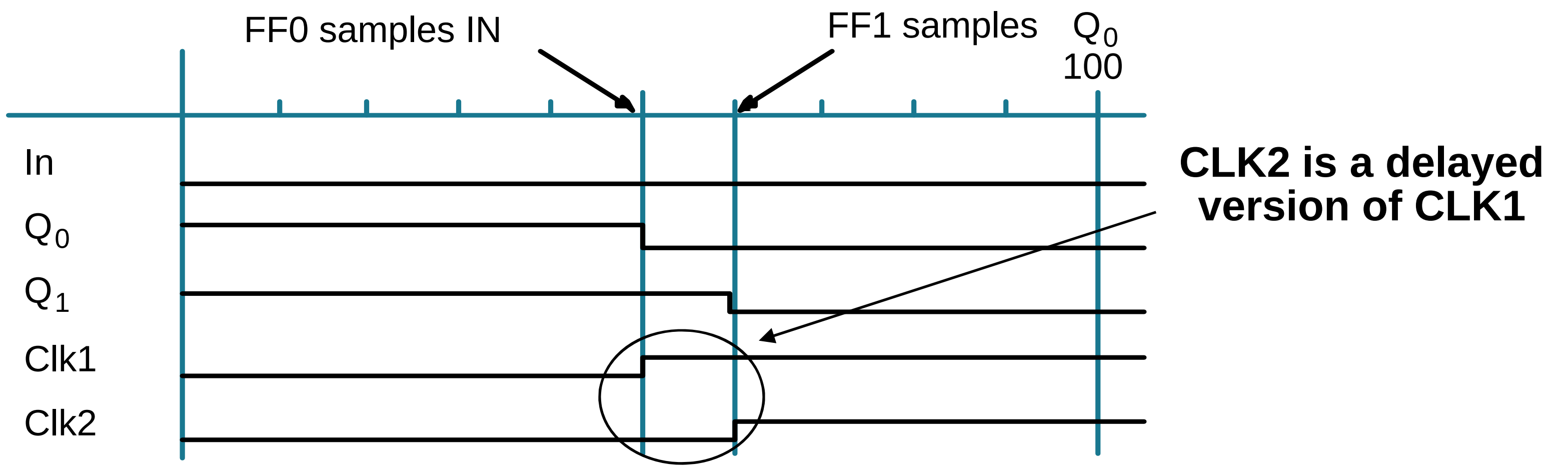
The Problem of Clock Skew

Correct behavior assumes next state of all storage elements determined by all storage elements *at the same time*

Not possible in real systems!

- **logical clock driven from more than one physical circuit with timing behavior**
- **different wire delay to different points in the circuit**

Effect of Skew on Cascaded Flipflops:



Original State: Q0 = 1, Q1 = 1, In = 0
Because of skew, next state becomes: Q0 = 0, Q1 = 0, not Q0 = 0, Q1 = 1

Design Strategies for Minimizing Clock Skew

Typical propagation delays for LS FFs: 13 ns

Need substantial clock delay (on the order of 13 ns) for skew to be a problem in this relatively slow technology

Nevertheless, the following are good design practices:

- ✓ **distribute clock signals in general direction of data flows**
- ✓ **wire carrying the clock between two communicating components should be as short as possible**
- ✓ **for multiphase clocked systems, distribute all clocks in similar wire paths; this minimizes the possibility of overlap**
- ✓ **for the non-overlap clock generate, use the phase feedback signals from the furthest point in the circuit to which the clock is distributed; this guarantees that the phase is seen as low everywhere before it allows the next phase to go high**

Choosing a Flipflop

R-S Clocked Latch:

- used as storage element in narrow width clocked systems
- its use is not recommended!
- however, fundamental building block of other flipflop types

J-K Flipflop: (historically popular, *but now not used*)

- versatile building block
- can be used to implement D and T FFs
- usually requires least amount of logic to implement
- but has two inputs with increased wiring complexity
- because of 1's catching, never use master/slave J-K FFs
- edge-triggered varieties exist

D Flipflop:

- minimizes wires, much preferred in VLSI technologies
- simplest design technique
- best choice for storage registers

T Flipflops:

- don't really exist, constructed from J-K FFs
- usually best choice for implementing counters

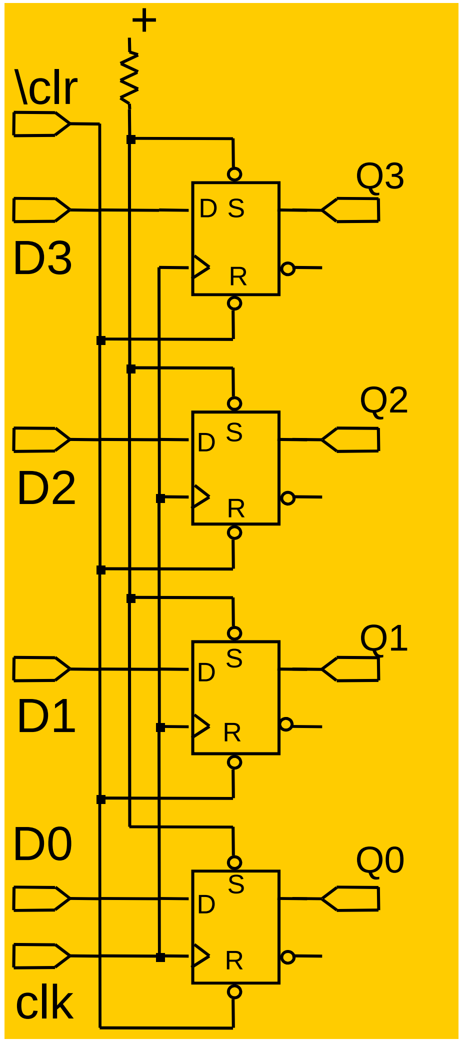
Preset and Clear inputs highly desirable!!

Registers

- Collection of Flip-Flops with similar controls and logic
  - stored values somehow related
  - share clocks, reset, and set lines
  - similar logic at each stage

- Examples
  - storage registers
  - shift registers
  - counters

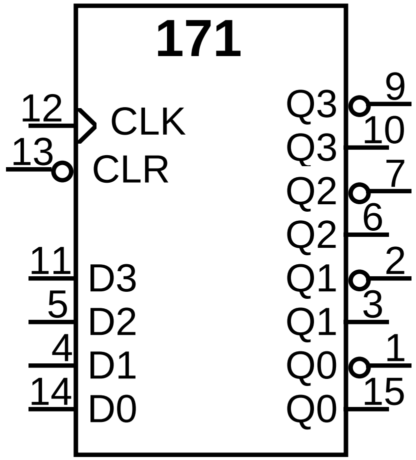
Storage Register



Group of storage elements read/written as a unit

4-bit register constructed from 4 D FFs
Shared clock and clear lines

Schematic Shape

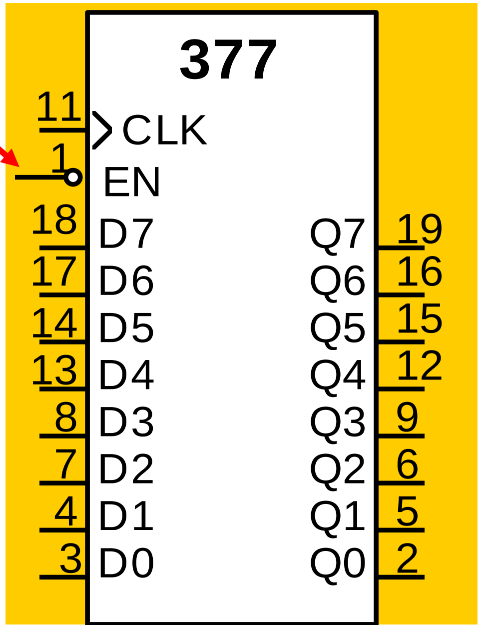


TTL 74171 Quad D-type FF with Clear
(Small numbers represent pin #s on package)

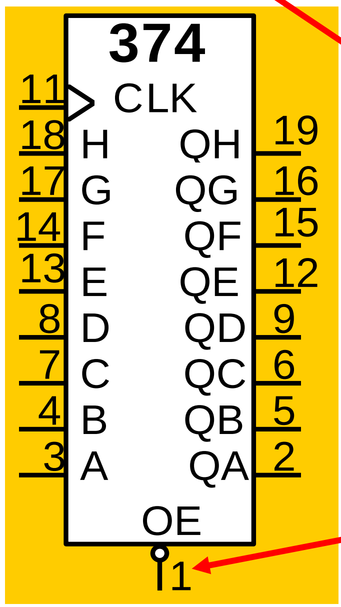
Kinds of Registers

Input/Output Variations

Selective Load Capability
Tri-state or Open Collector Outputs
True and Complementary Outputs



74377 Octal D-type FFs
with input enable

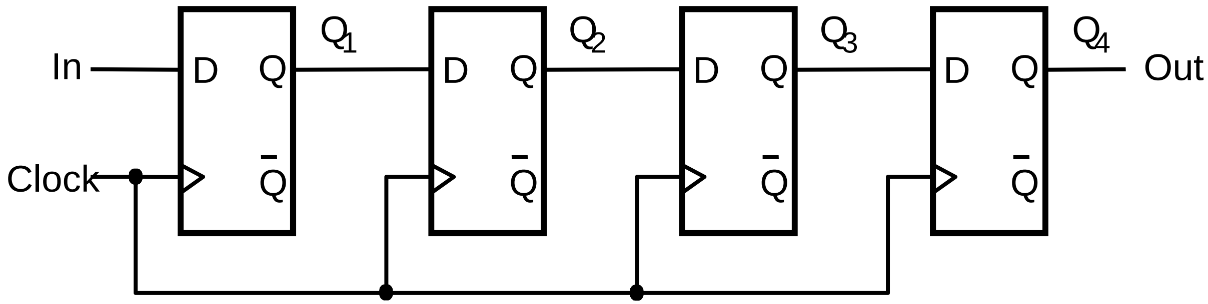


74374 Octal D-type FFs
with output enable

EN enabled low and lo-to-hi clock transition to load new data into register

OE asserted low presents FF state to output pins; otherwise high impedence

A simple shift register

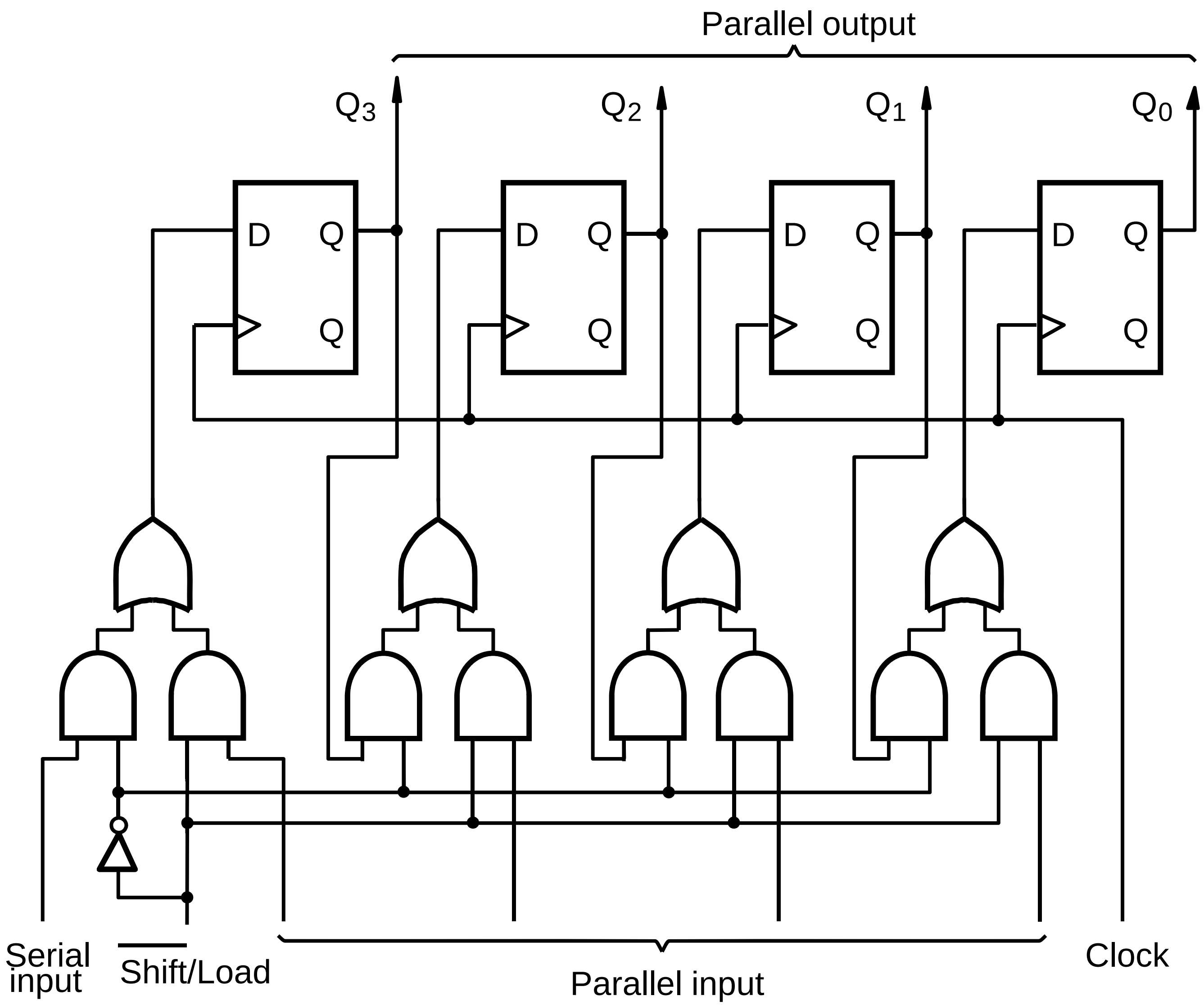


(a) Circuit

	In	Q_1	Q_2	Q_3	Q_4 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

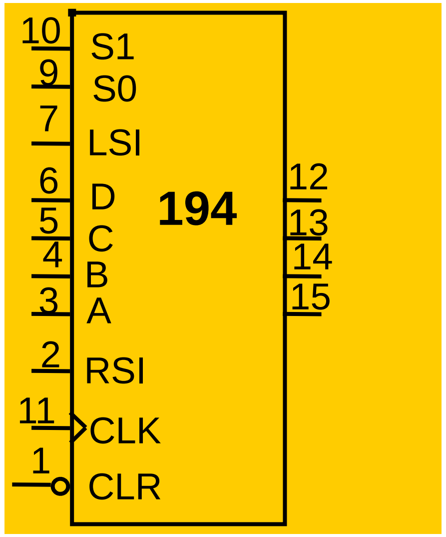
(b) A sample sequence

Parallel-access shift register



Shift Register I/O

Serial vs. Parallel Inputs
Serial vs. Parallel Outputs
Shift Direction: Left vs. Right



74194 4-bit Universal Shift Register

Serial Inputs: LSI, RSI
Parallel Inputs: D, C, B, A
Parallel Outputs: QD, QC, QB, QA
Clear Signal
Positive Edge Triggered Devices

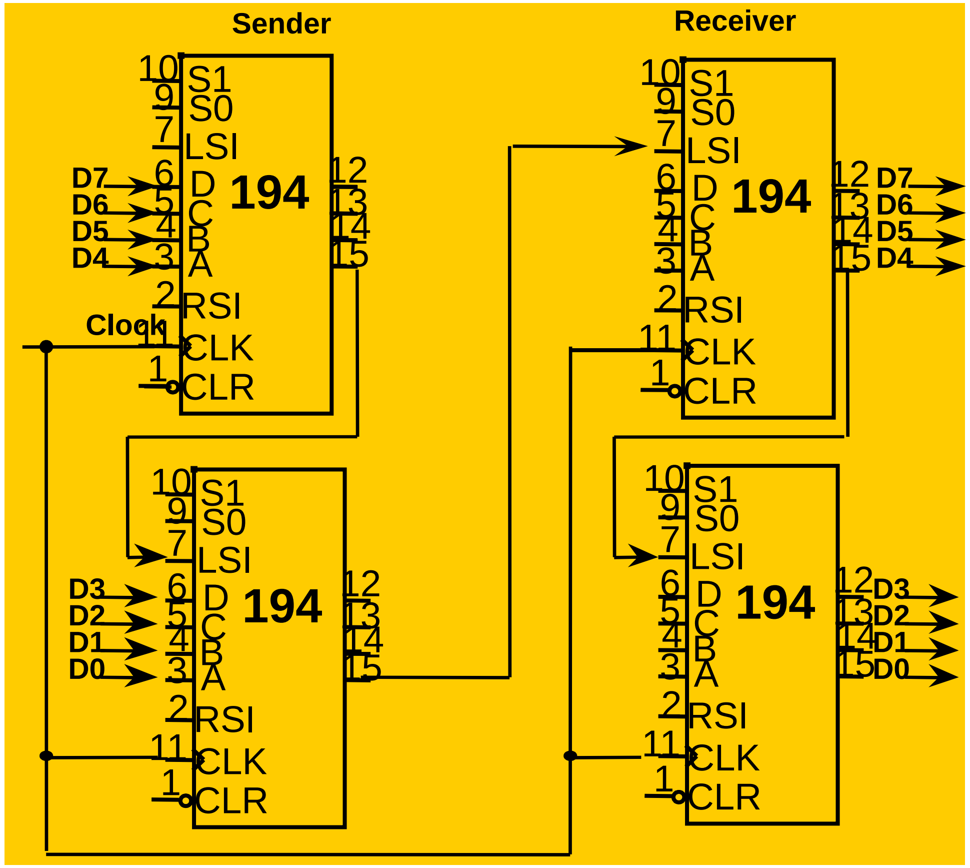
S1,S0 determine the shift function
S1 = 1, S0 = 1: Load on rising clk edge synchronous load
S1 = 1, S0 = 0: shift left on rising clk edge LSI replaces element D
S1 = 0, S0 = 1: shift right on rising clk edge RSI replaces element A
S1 = 0, S0 = 0: hold state

Multiplexing logic on input to each FF!

Shifters well suited for serial-to-parallel conversions, such as terminal to computer communications

Shift Register Application: Parallel to Serial Conversion

Parallel Inputs



Parallel Outputs

Serial transmission

Counters

Counters

Proceed through a well-defined sequence of states in response to count signal

3 Bit Up-counter: 000, 001, 010, 011, 100, 101, 110, 111, 000, ...

3 Bit Down-counter: 111, 110, 101, 100, 011, 010, 001, 000, 111, ...

Binary vs. BCD vs. Gray Code Counters

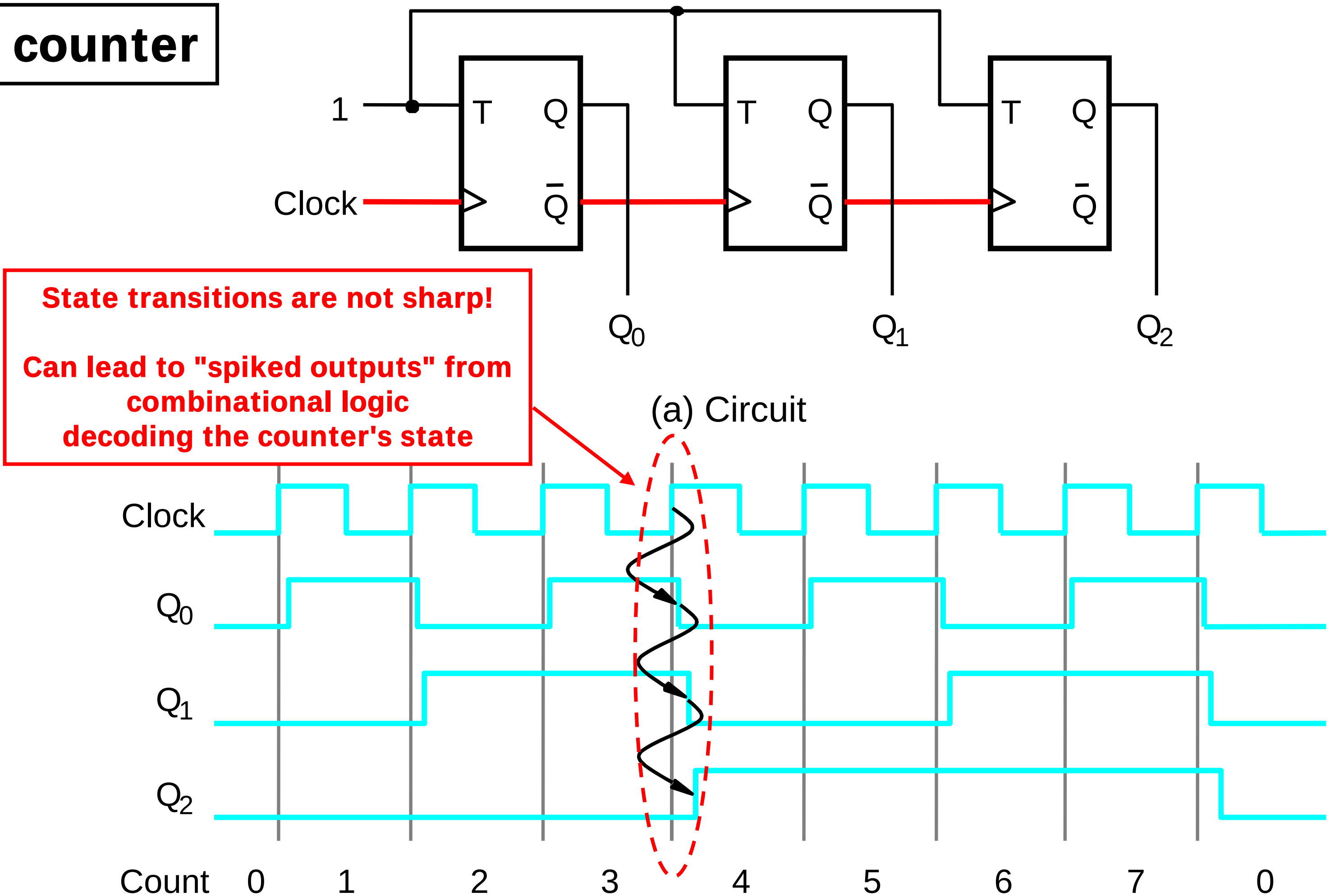
A counter is a "degenerate" finite state machine/sequential circuit where the state *is* the only output

Types of counters

Asynchronous vs. Synchronous Counters

Asynchronous counters

Ripple counter

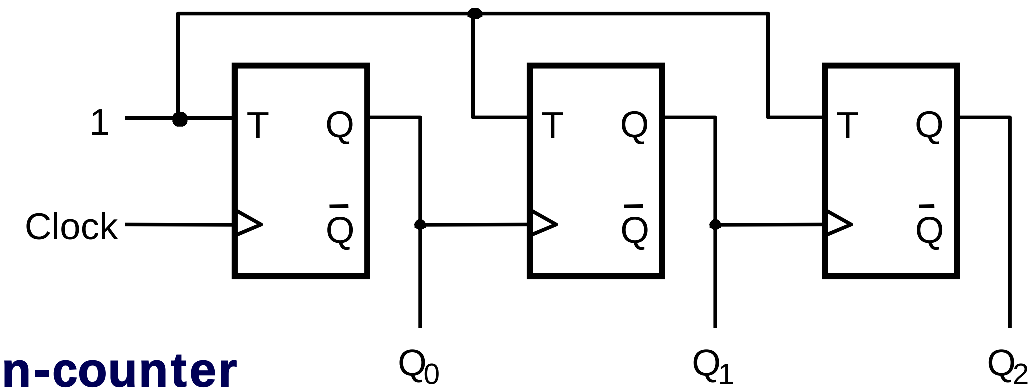


(a) Circuit

(b) Timing diagram

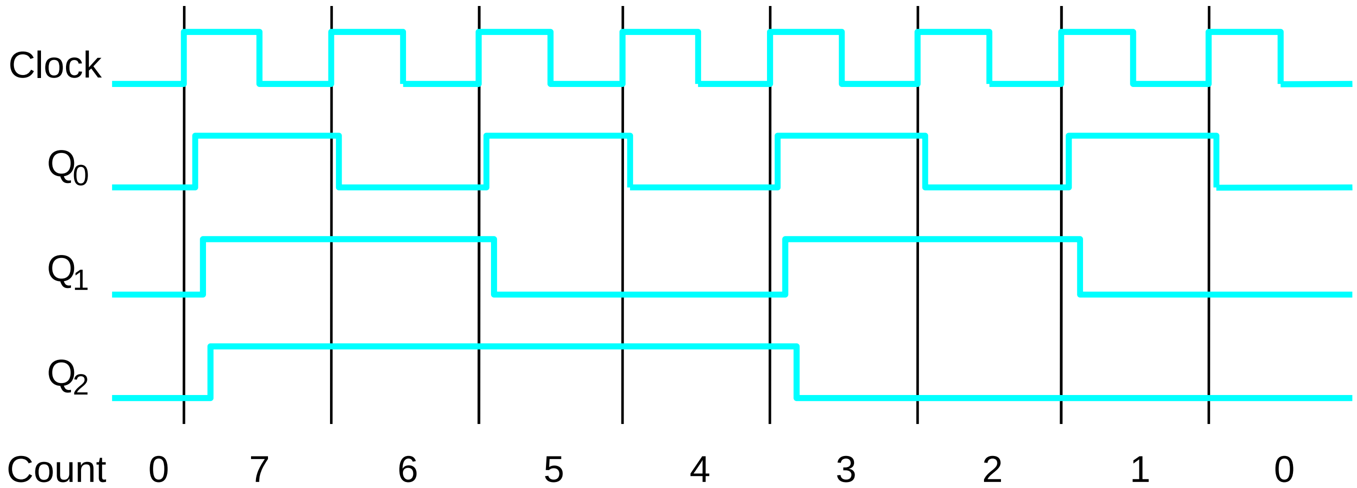
A three-bit up-counter

Asynchronous counters, cont'd



A three-bit down-counter

(a) Circuit



(b) Timing diagram

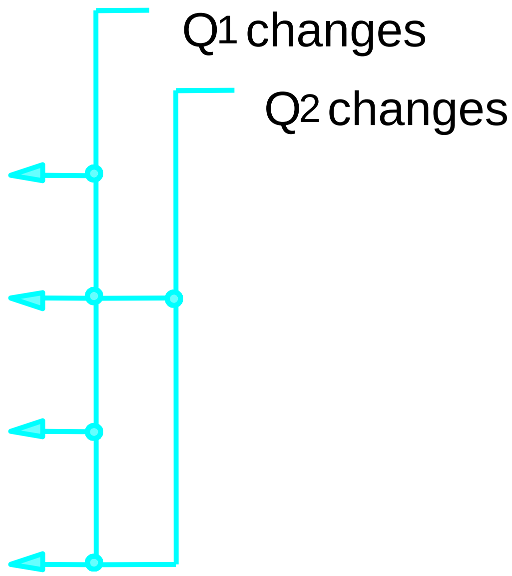
Synchronous counter

- Asynchronous counters
  - simple, but not very fast
  - can build faster counters by clocking all FFs at the same time →"synchronous counter"

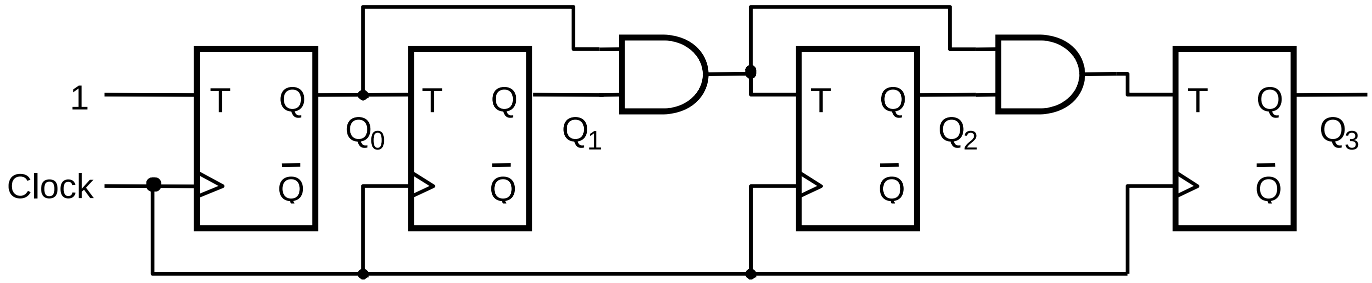
Synchronous counters with T F/F

$T_0=1$
$T_1=Q_0$
$T_2=Q_0Q_1$

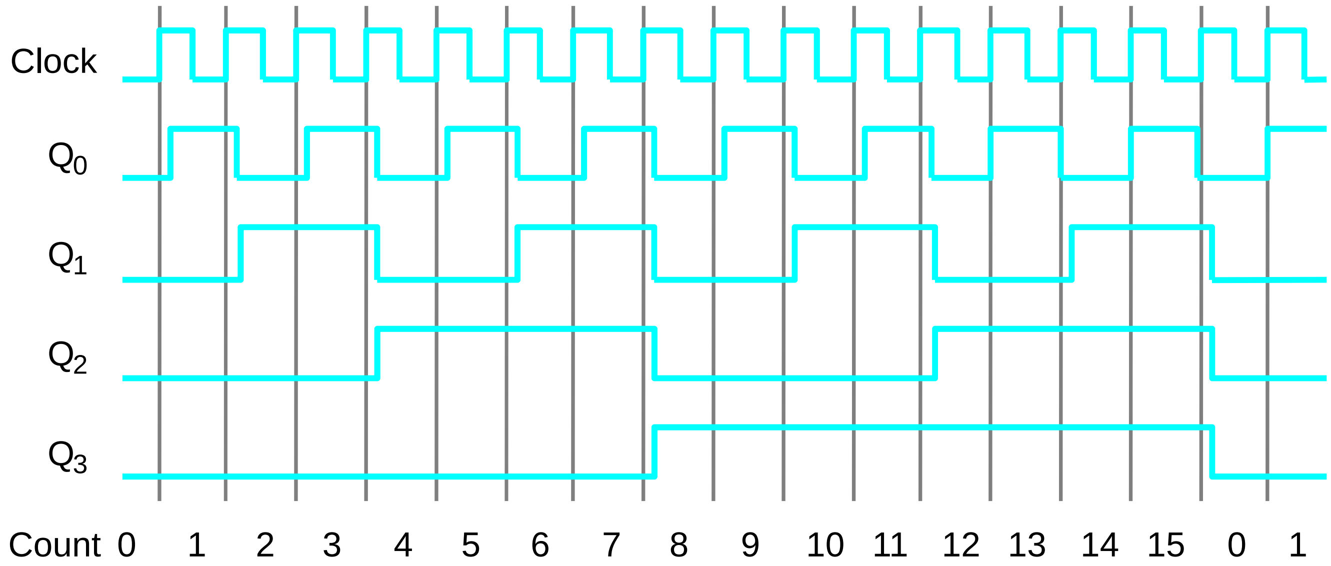
Clock cycle	Q2	Q1	Q0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0



A four-bit synchronous up-counter

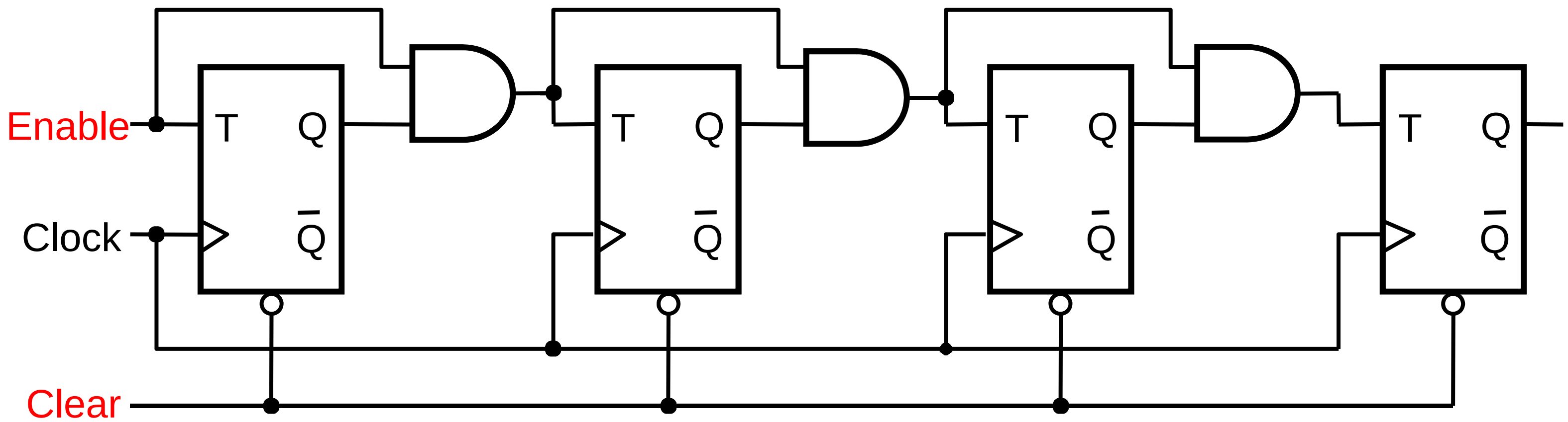


(a) Circuit

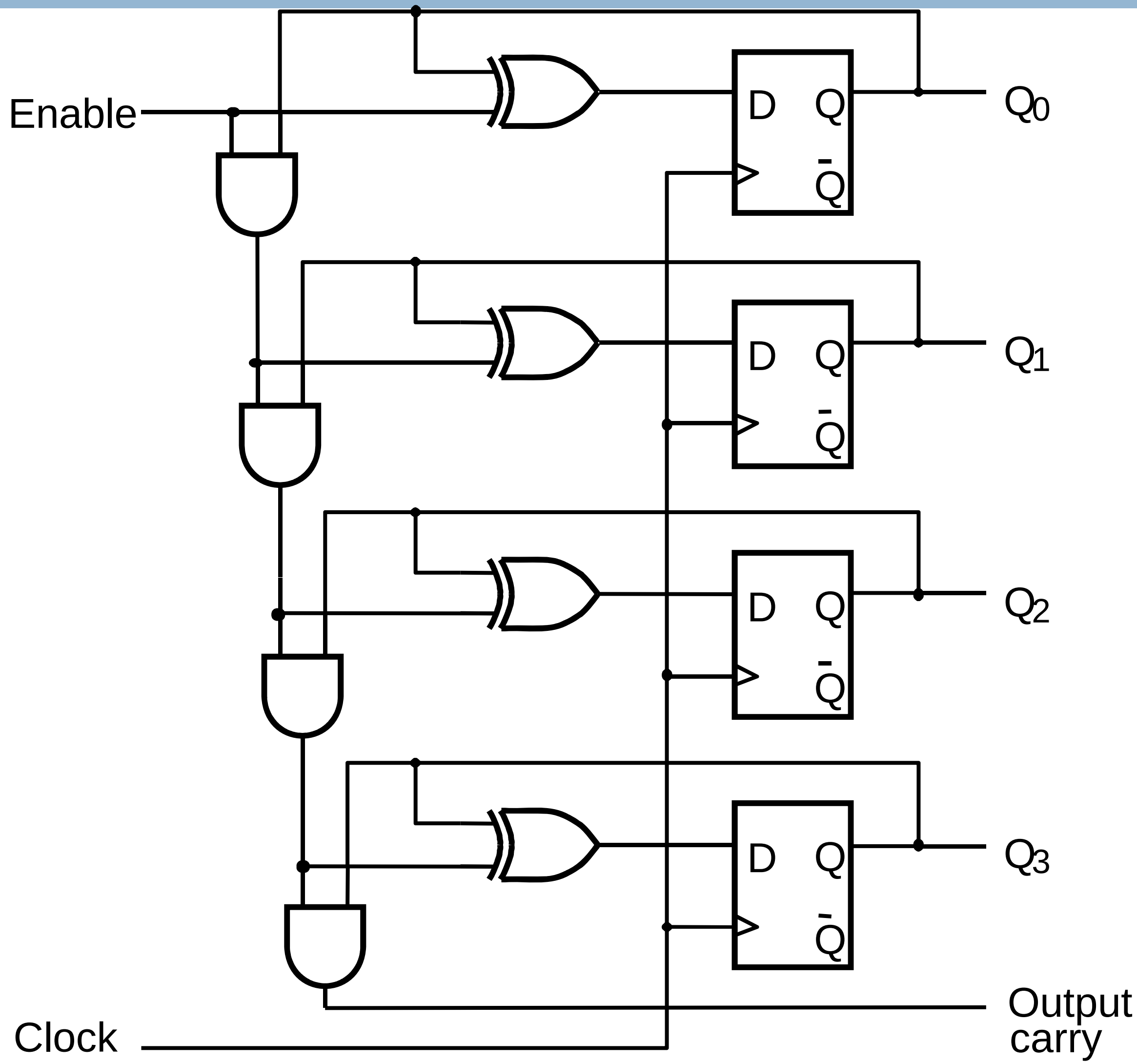


(b) Timing diagram

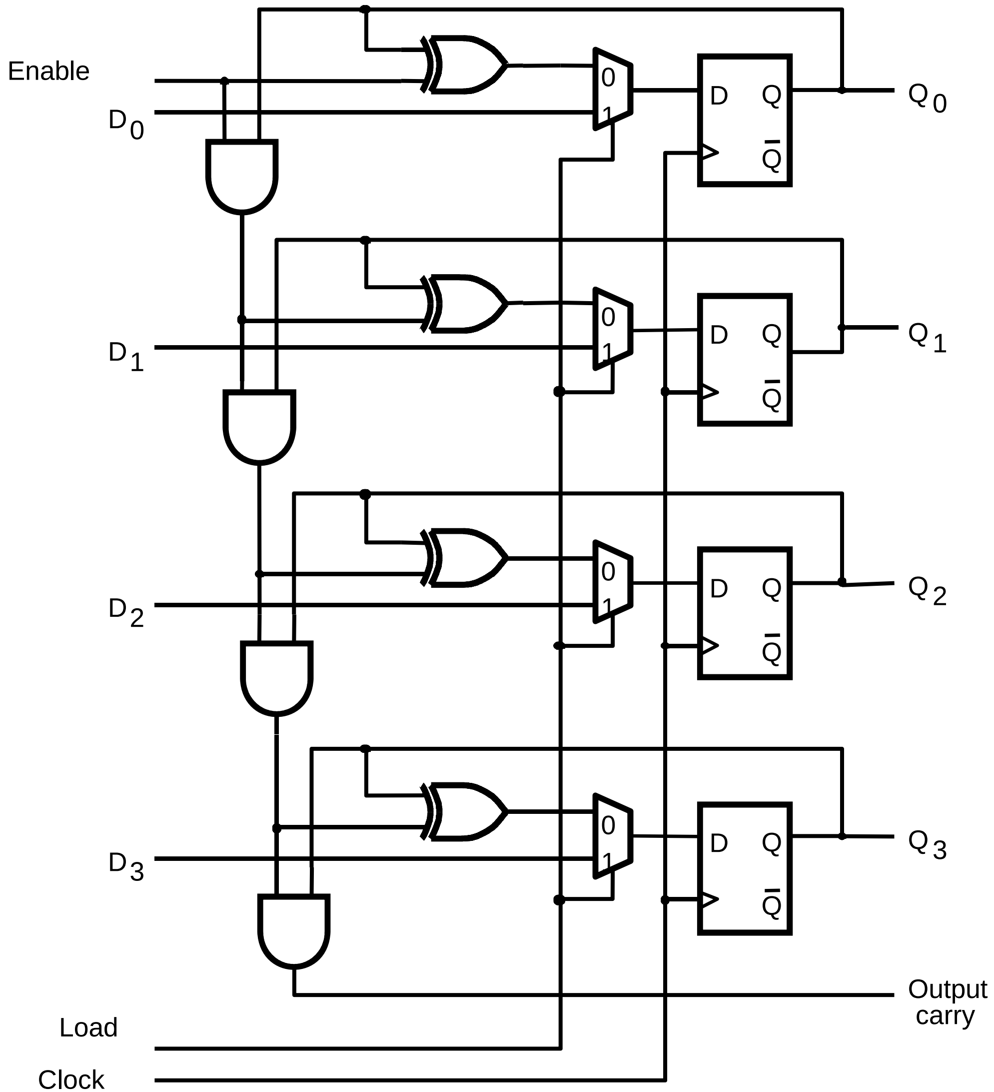
Enable and Clear capability



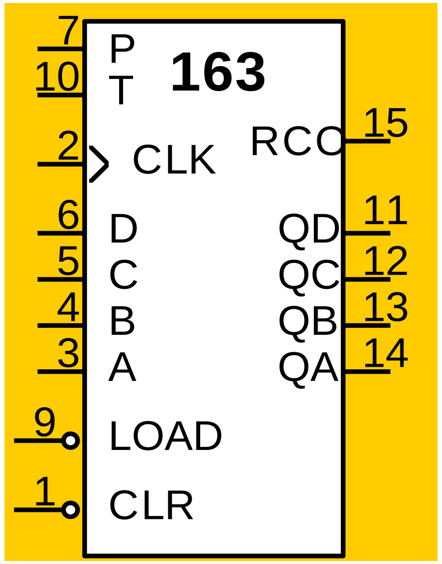
A four-bit counter with D FFs



A counter with parallel-load capability



Catalog Counter



74163 Synchronous 4-Bit Upcounter

Synchronous Load and Clear Inputs

Positive Edge Triggered FFs

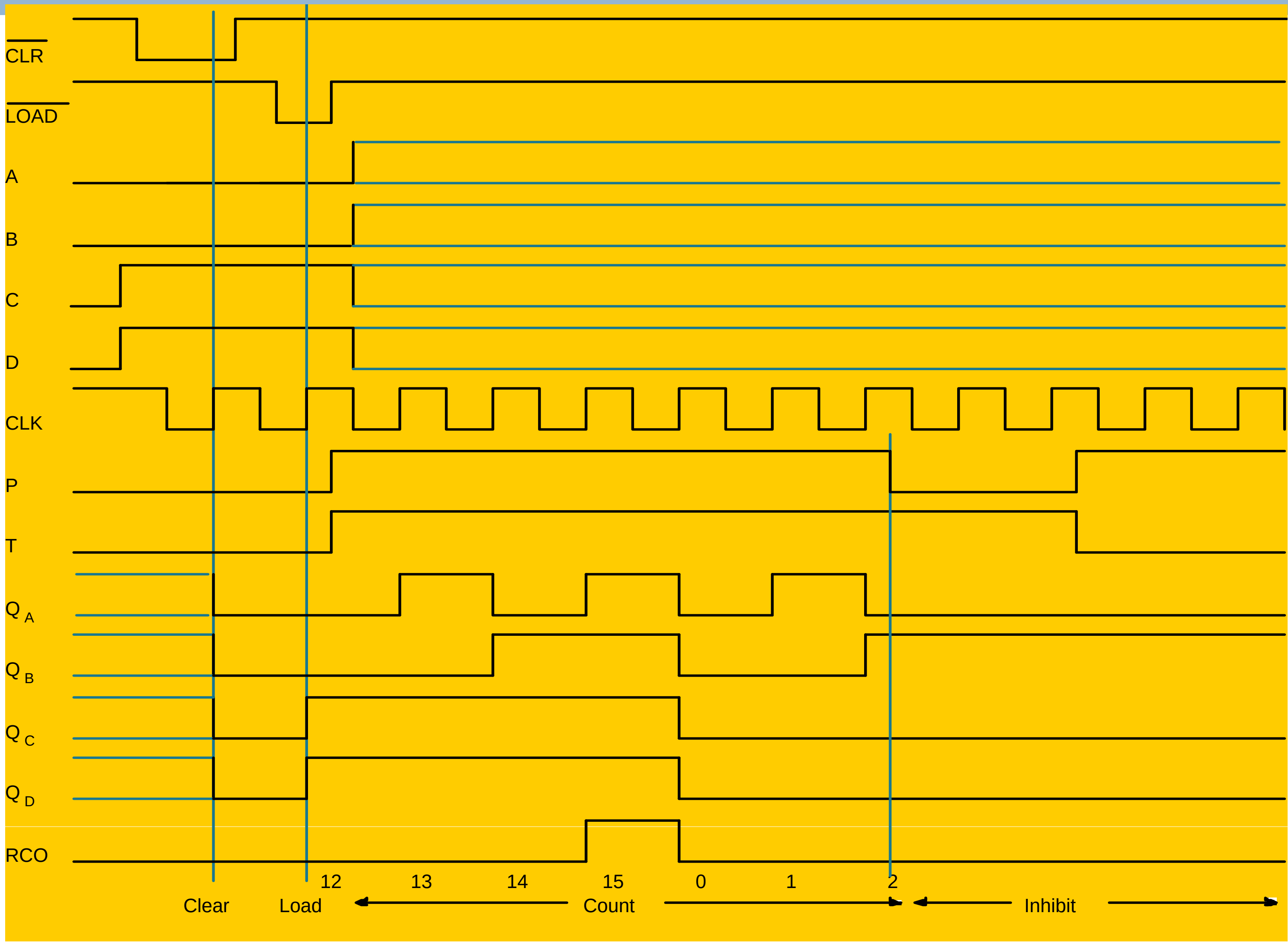
Parallel Load Data from D, C, B, A

P, T Enable Inputs: both must be asserted to enable counting

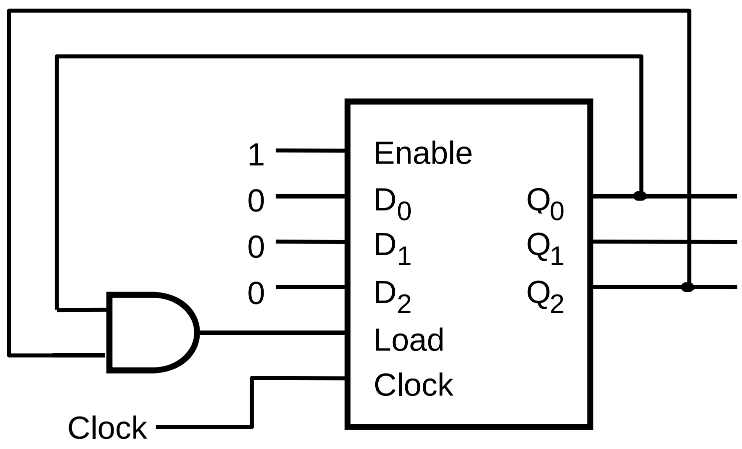
RCO: asserted when counter enters its highest state 1111, used for cascading counters *"Ripple Carry Output"*

74161: similar in function, asynchronous load and reset

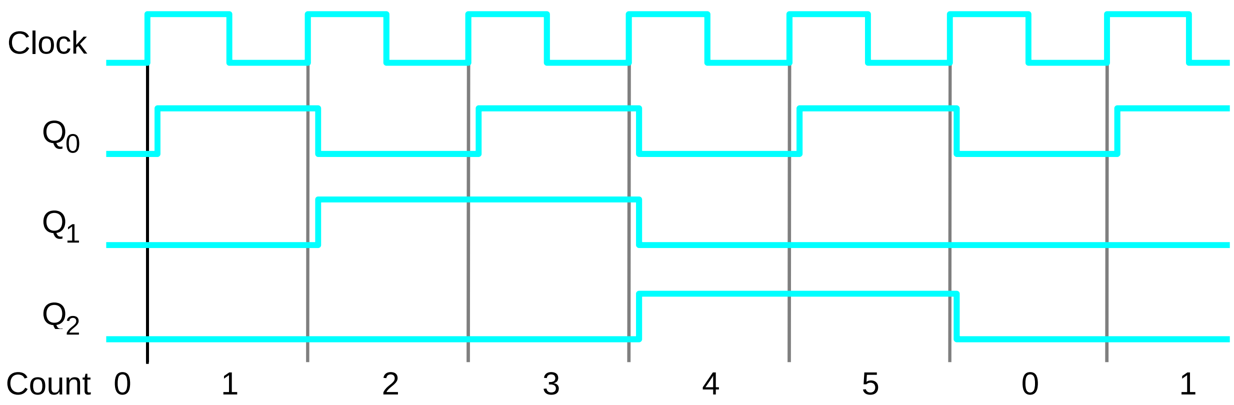
74163 Detailed Timing Diagram



A modulo-6 counter with synchronous reset

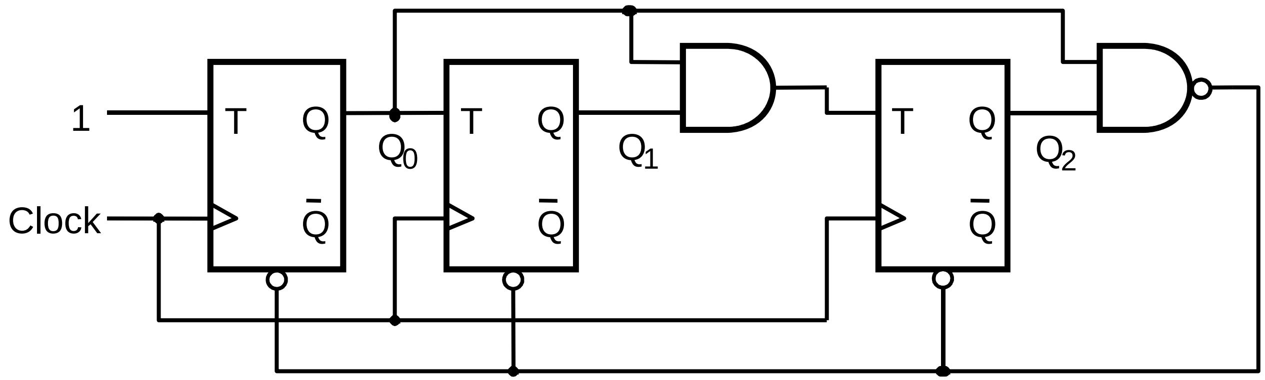


(a) Circuit

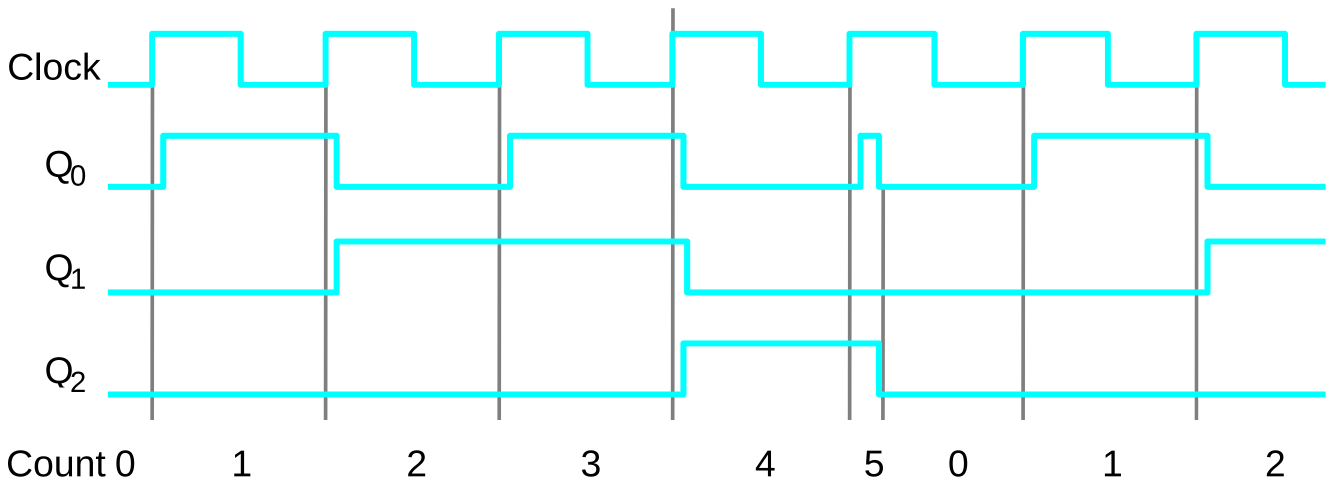


(b) Timing diagram

A modulo-6 counter with asynchronous reset



(a) Circuit

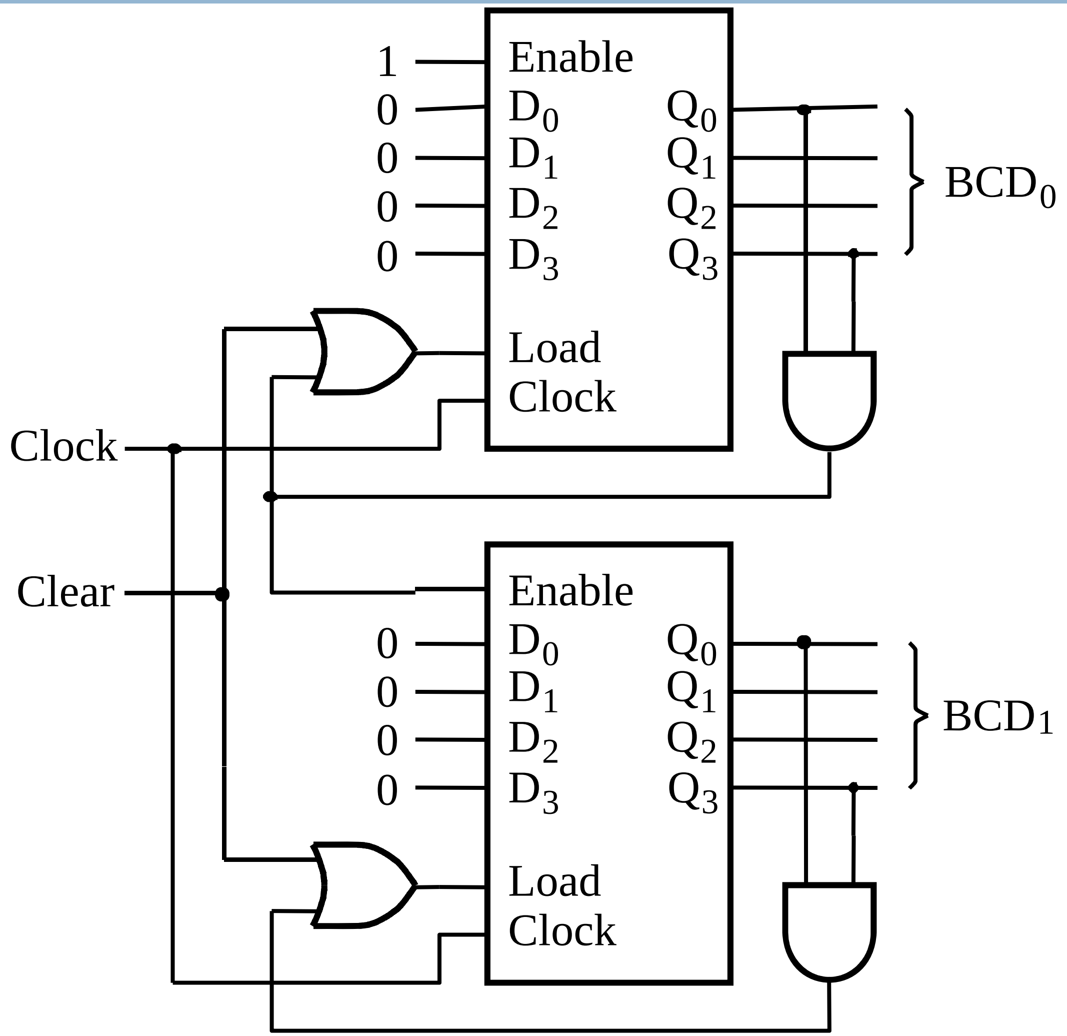


(b) Timing diagram

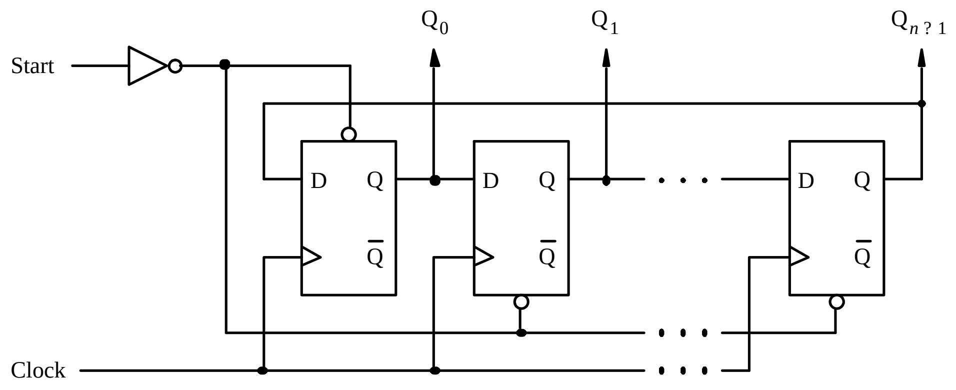
Other types of counters

- Two-digit BCD counters
  - Two modulo-10 counters, one for each digit
  - Reset when the counter reaches 9
- Ring counters
  - One bit is one while other bits are 0
    - one hot encoding
- Johnson counter
  - 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000, …

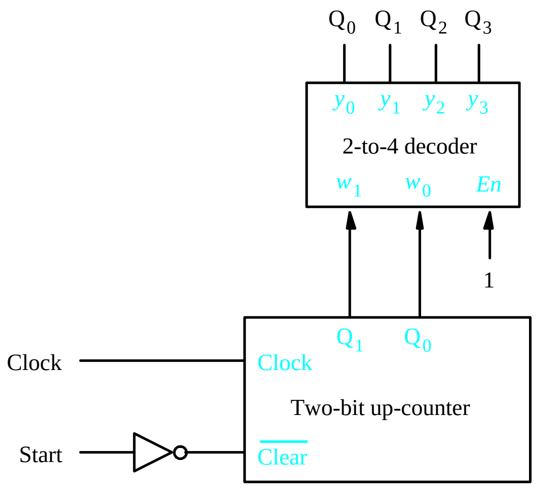
A two-digit BCD counter



Ring Counter



(a) An *n*-bit ring counter



(b) A four-bit ring counter

Johnson counter