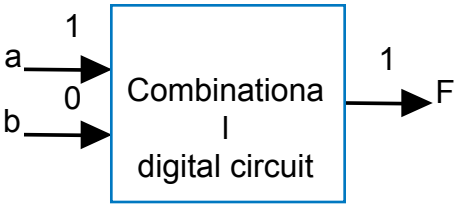
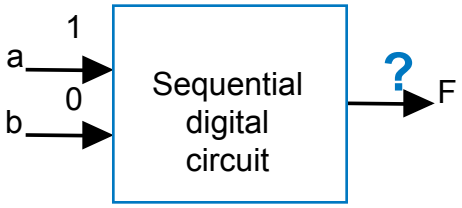
Sequential Logic Design

Introduction

- Sequential circuit
  - Output depends not just on present inputs (as in combinational circuit), but on past sequence of inputs
    - Stores bits, also known as having "state"
  - Simple example: a circuit that counts up in binary
- we will:
  - Design a new building block, a **flip-flop**, that stores one bit
  - Combine that block to build multi-bit storage
  - **register**
  - Describe the sequential behavior using a **finite state machine**
  - Convert a finite state machine to a **controller** – a sequential circuit having a register and combinational logic

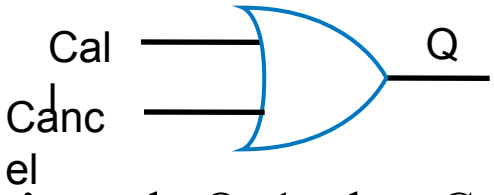




Must know sequence of past inputs to know output

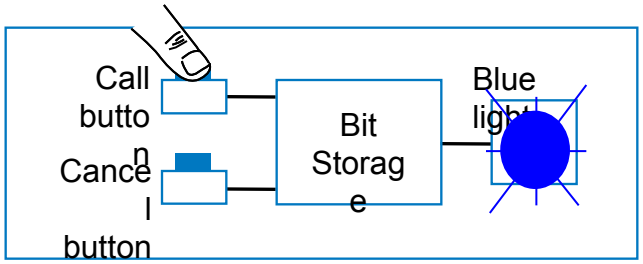
Example Needing Bit Storage

- Flight attendant call button
  - Press call: light turns on
    - ***Stays on*** after button released
  - Press cancel: light turns off
  - Logic gate circuit to implement this?

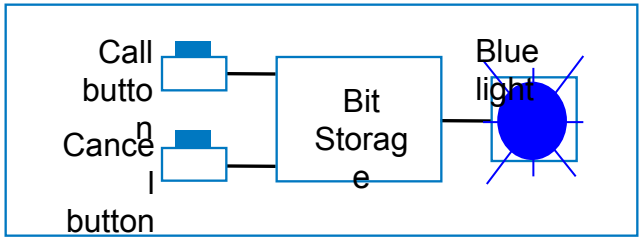


Doesn't work. Q=1 when Call=1, but doesn't stay 1 when Call returns to 0

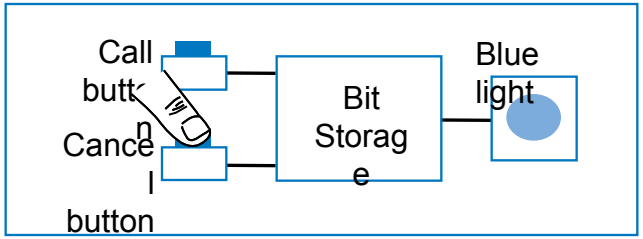
Need some form of "feedback" in the circuit



1. Call button pressed – light turns on



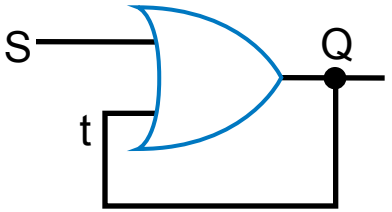
*2. Call button released – light **stays on***

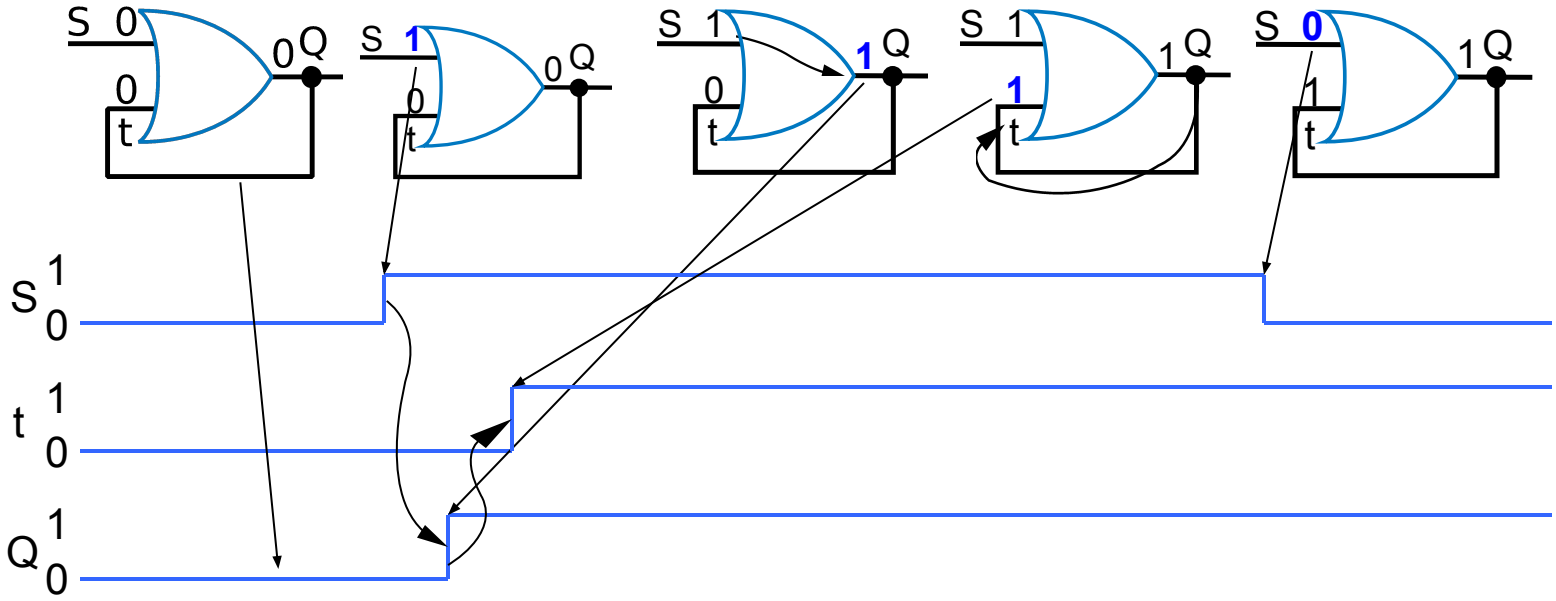


3. Cancel button pressed – light turns off

First attempt at Bit Storage

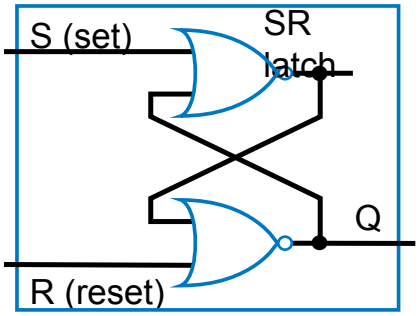
- We need some sort of feedback
  - Does circuit on the right do what we want?
    - No: Once Q becomes 1 (when S=1), Q stays 1 forever – no value of S can bring Q back to 0

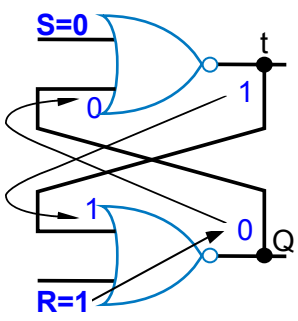


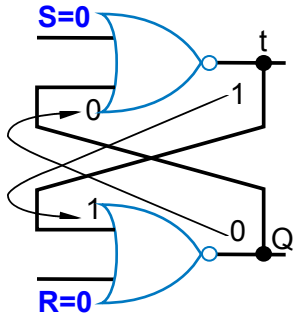


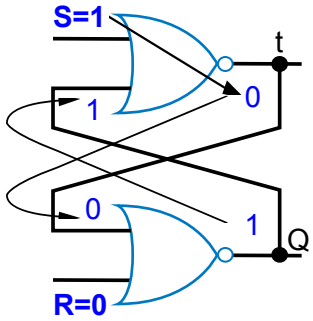
Bit Storage Using an SR Latch

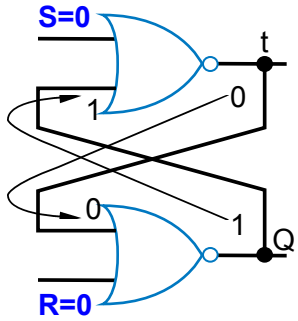
- Does the circuit to the right, with cross-coupled NOR gates, do what we want?
  - Yes! How did someone come up with that circuit? Maybe just trial and error, a bit of insight...

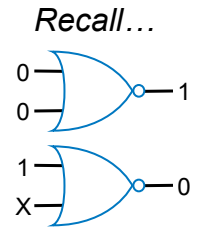


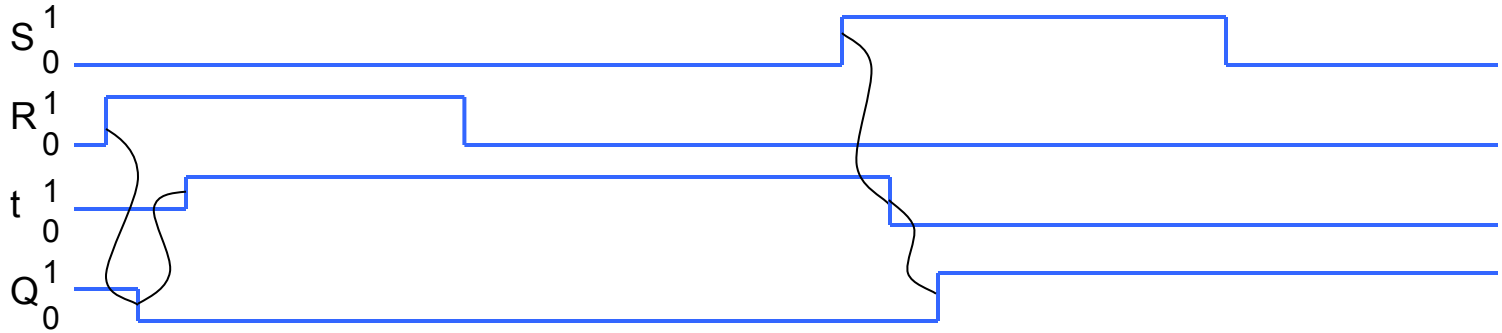










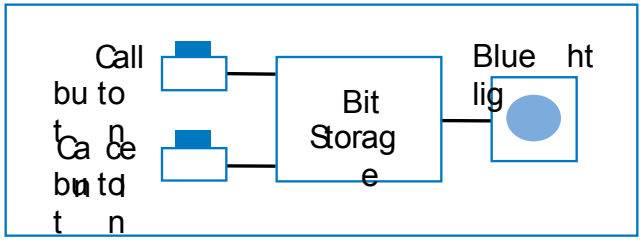


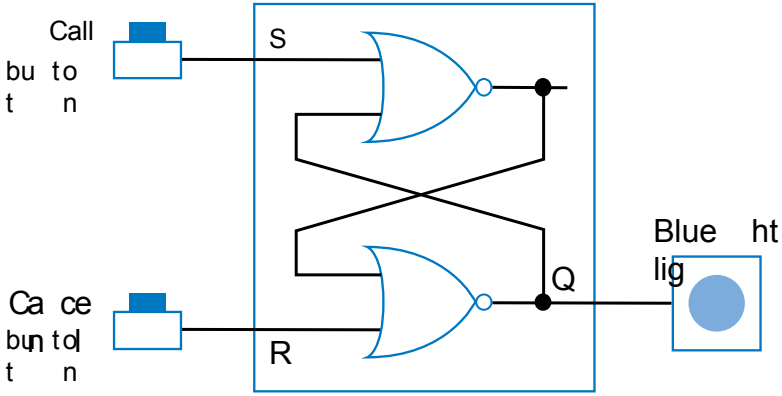
Recall ..

Example Using SR Latch for Bit Storage

- SR latch can serve as bit storage in previous example of flight-attendant call button
  - Call=1 : sets Q to 1
    - Q stays 1 even after Call=0
  - Cancel=1 : resets Q to 0

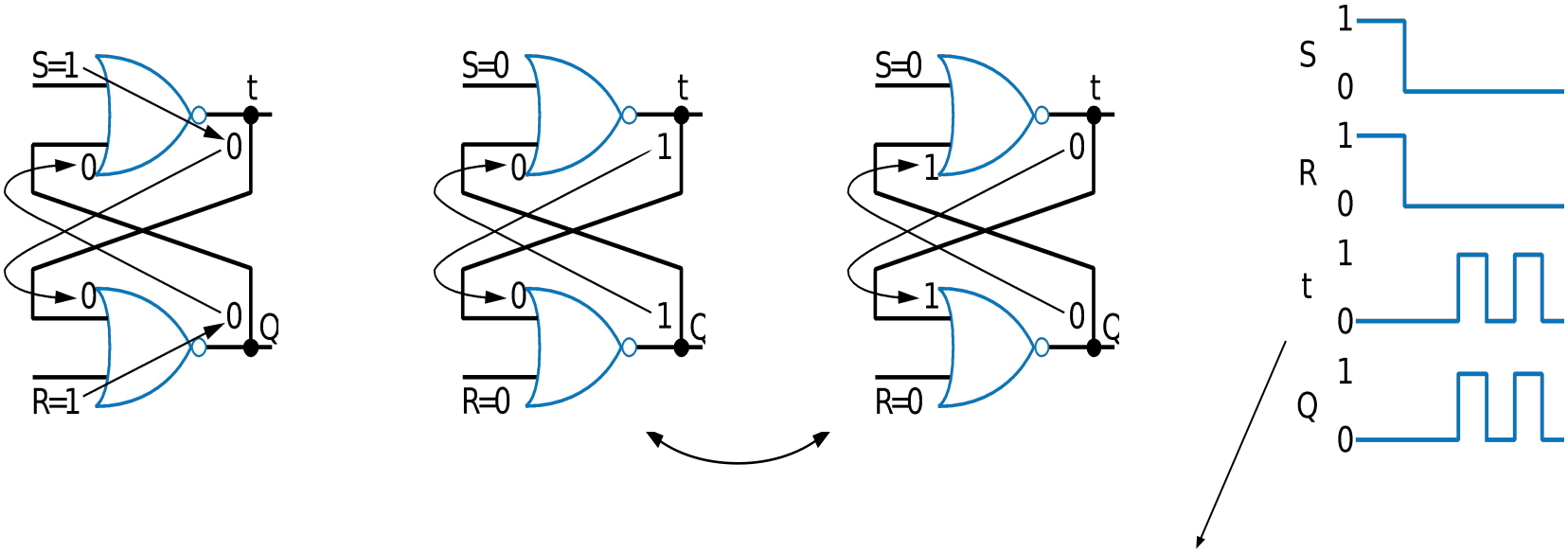
- But, there's a problem...



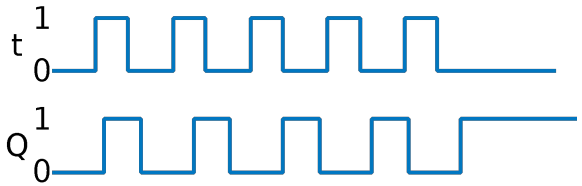


Problem with SR Latch

- Problem
  - If S=1 and R=1 simultaneously, we don't know what value Q will take

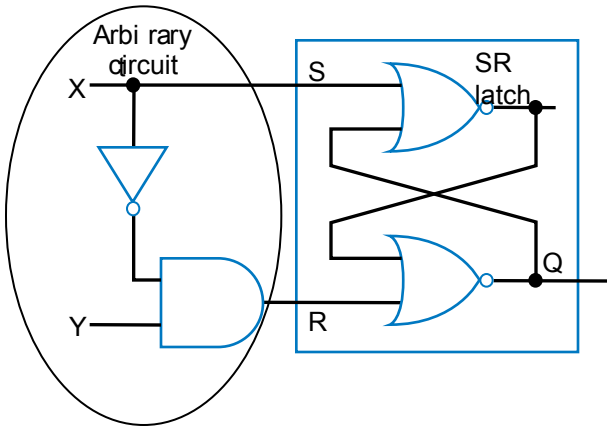


Q may oscillate. Then, because one path will be slightly longer than the other, Q will eventually settle to 1 or 0 – but we don't know which.

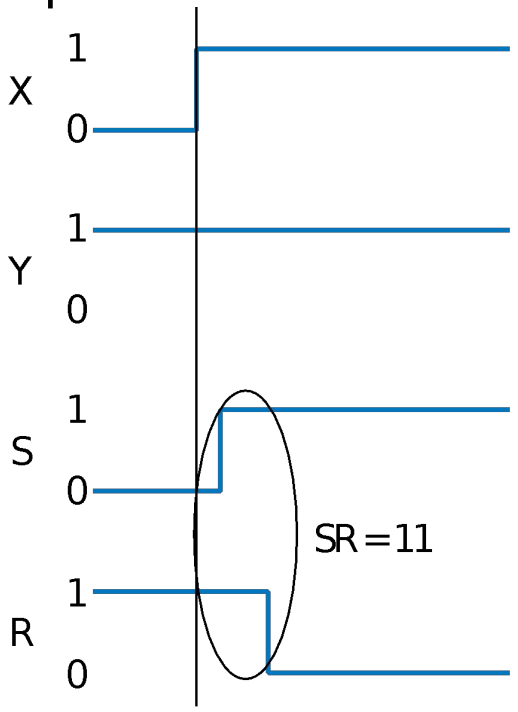


Problem with SR Latch

- Problem not just one of a user pressing two buttons at same time
- Can also occur even if SR inputs come from a circuit that supposedly never sets S=1 and R=1 at same time
  - But does, due to different delays of different paths

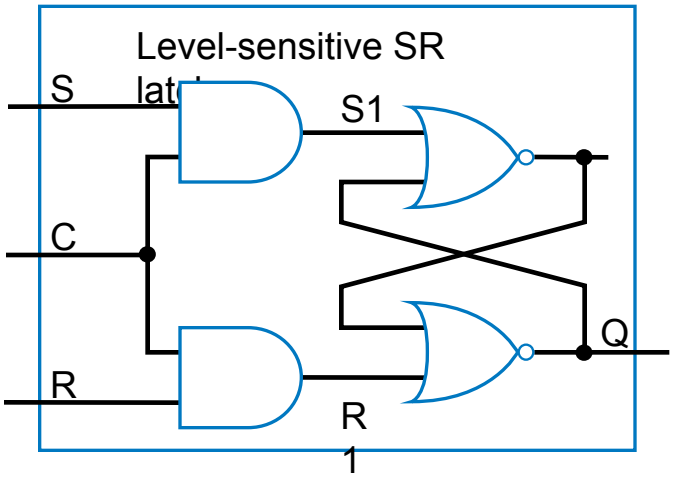


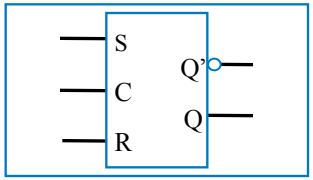
The longer path from X to R than to S causes SR=11 for short time – could be long enough to cause oscillation



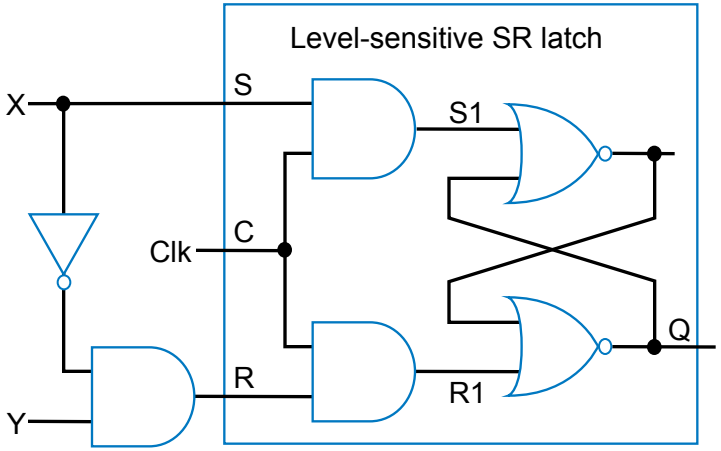
Solution: Level-Sensitive SR Latch

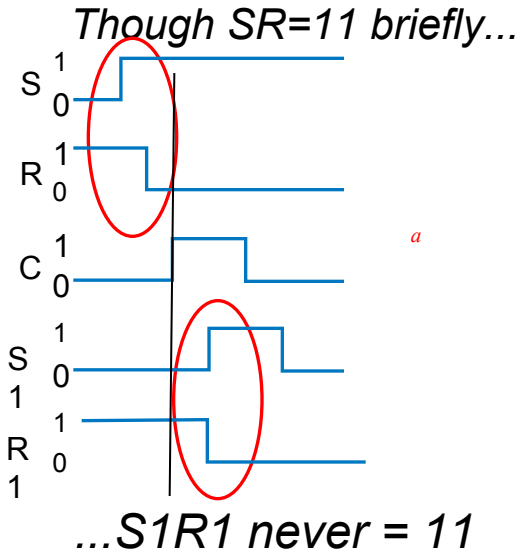
- Add enable input “C” as shown
  - Only let S and R change when C=0
    - Enure circuit in front of SR never sets SR=11, except briefly due to path delays
  - Change C to 1 only after sufficient time for S and R to be stable
  - When C becomes 1, the stable S and R value passes through the two AND gates to the SR latch’s S1 R1 inputs.



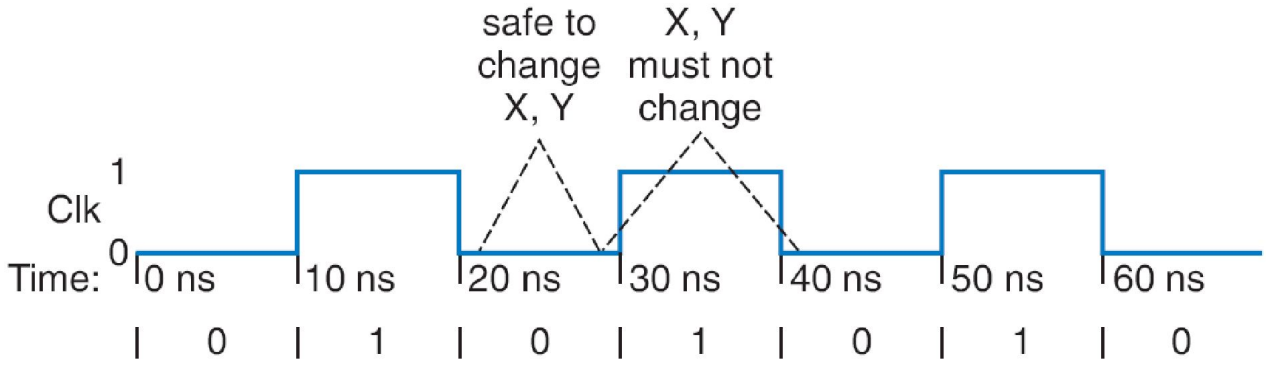


Level-sensitive SR latch symbol

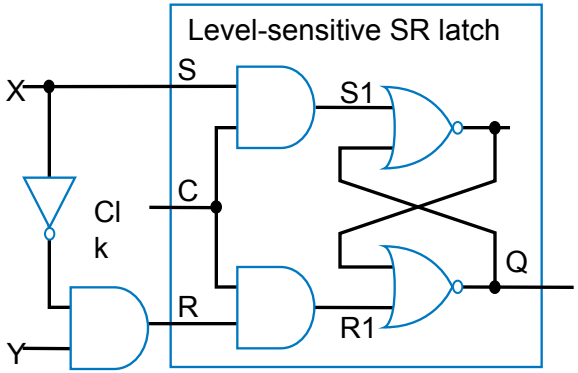




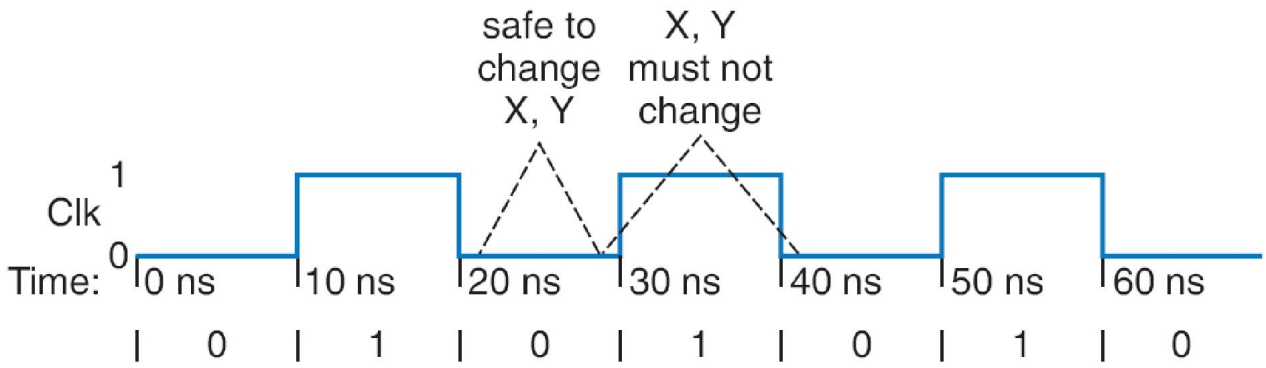
Clock Signals for a Latch



- How do we know when it's safe to set C=1?
  - Most common solution –make C pulse up/down
    - C=0: Safe to change X, Y
    - C=1: Must *not* change X, Y
    - We'll see how to ensure that later
  - ***Clock*** signal -- Pulsing signal used to enable latches
    - Because it ticks like a clock
  - Sequential circuit whose storage components all use clock signals: ***synchronous*** circuit
    - Most common type
    - Asynchronous circuits – important topic, but left for advanced course



Clocks

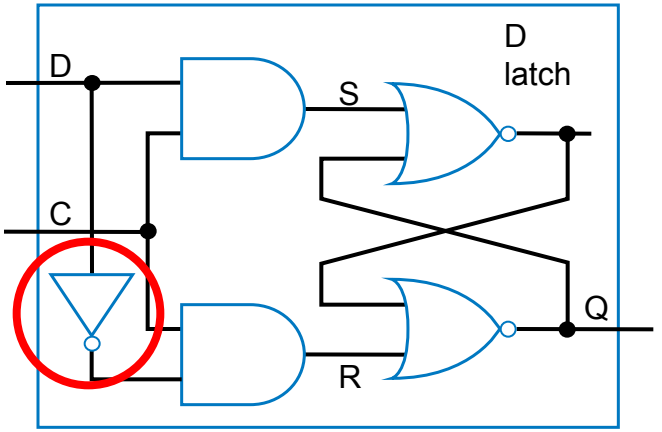


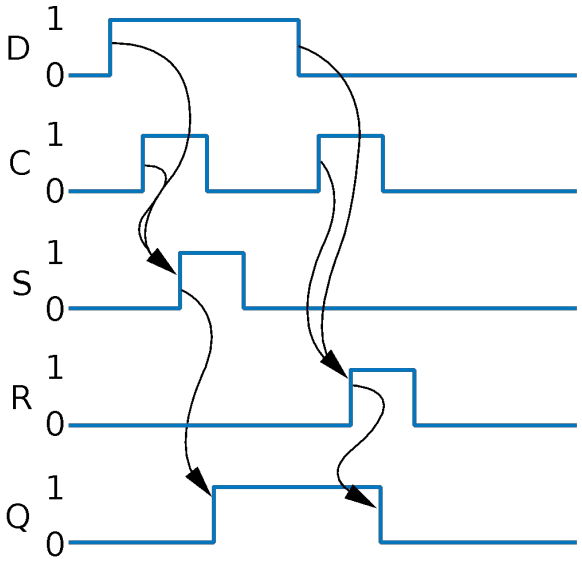
- ***Clock period***: time interval between pulses
  - Above signal: period = 20 ns
- ***Clock cycle***: one such time interval
  - Above signal shows 3.5 clock cycles
- ***Clock frequency***: 1/period
  - Above signal: frequency = 1 / 20 ns = 50 MHz
    - 1 Hz = 1/s

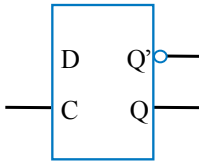
Freq	Period
100 GHz	0.01 ns
10 GHz	0.1 ns
1 GHz	1 ns
100 MHz	10 ns
10 MHz	100 ns

Level-Sensitive D Latch

- SR latch requires careful design to ensure SR=11 never occurs
- D latch relieves designer of that burden
  - Inserted inverter ensures R always opposite of S



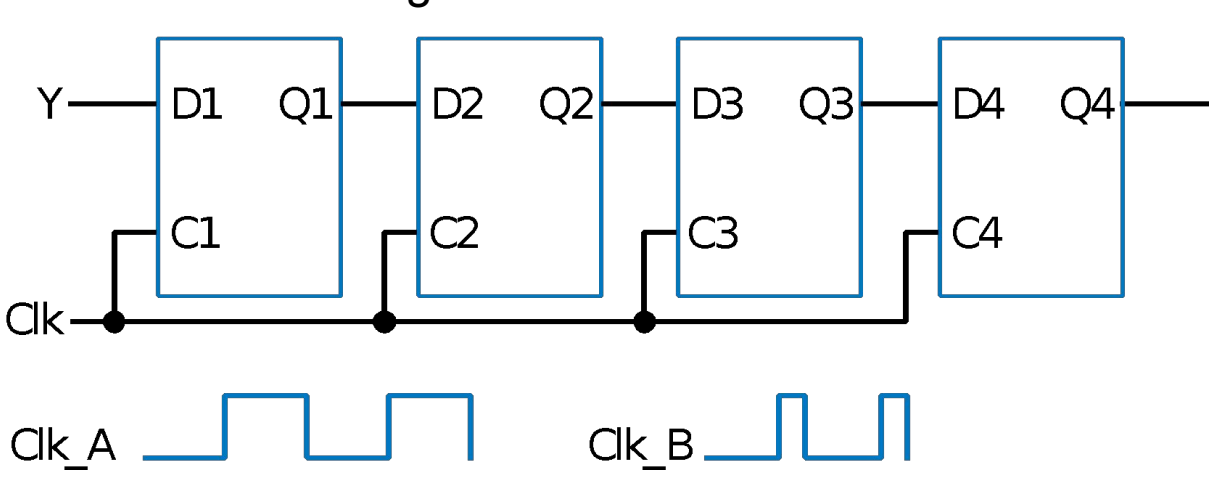


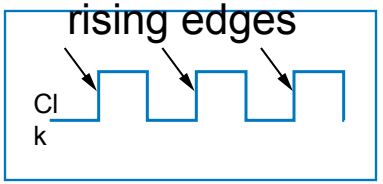


D latch symbol

Problem with Level-Sensitive D Latch

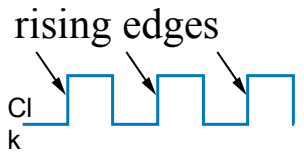
- D latch still has problem (as does SR latch)
  - When C=1, through how many latches will a signal travel?
  - Depends on for how long C=1
    - Clk_A -- signal may travel through multiple latches
    - Clk_B -- signal may travel through fewer latches
  - Hard to pick C that is just the right length
    - Can we design bit storage that only stores a value on the rising edge of a clock signal?



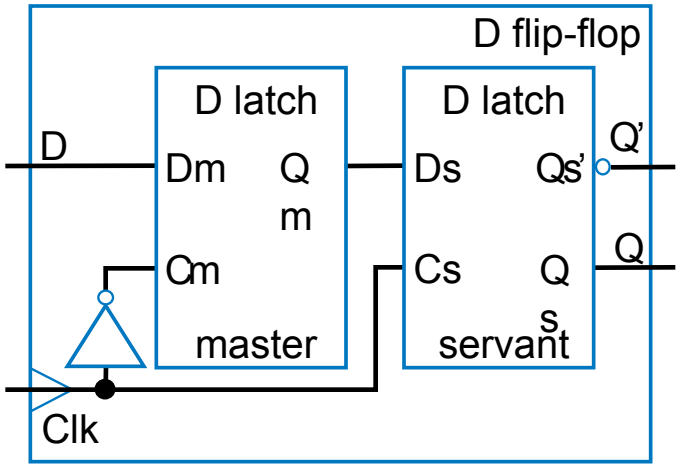


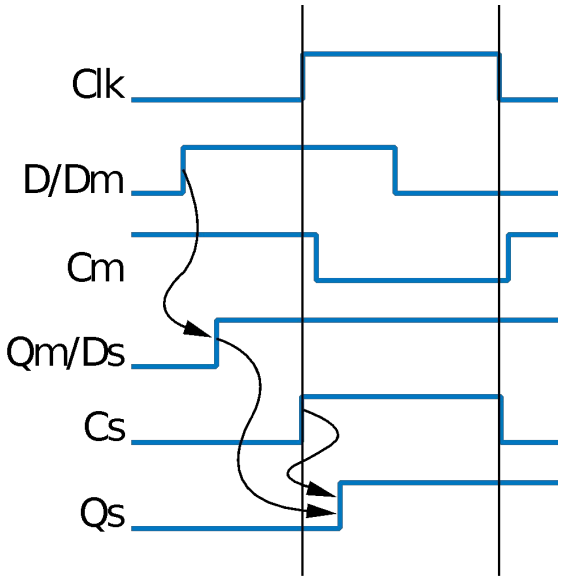
D Flip-Flop

- ***Flip-flop***: Bit storage that stores on clock edge, not level
- One design -- master-servant
  - Two latches, output of first goes to input of second, master latch has inverted clock signal
  - So master loaded when C=0, then servant when C=1
  - When C changes from 0 to 1, master disabled, servant loaded with value that was at D just before C changed -- i.e., value at D during rising edge of C



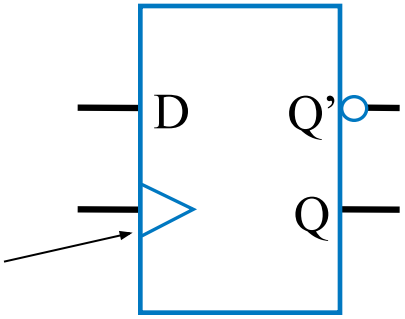
Note: Hundreds of different flip-flop designs exist



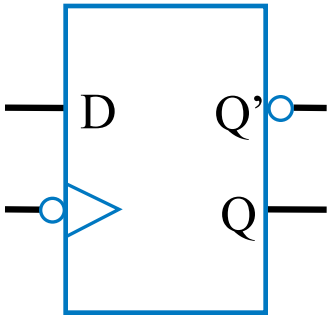


D Flip-Flop

The triangle means clock input, edge triggered

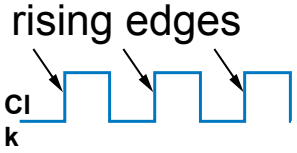


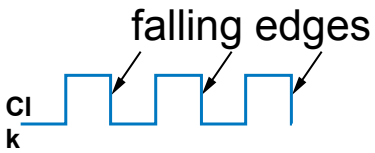
Symbol for rising-edge triggered D flip-flop



Symbol for falling-edge triggered D flip-flop

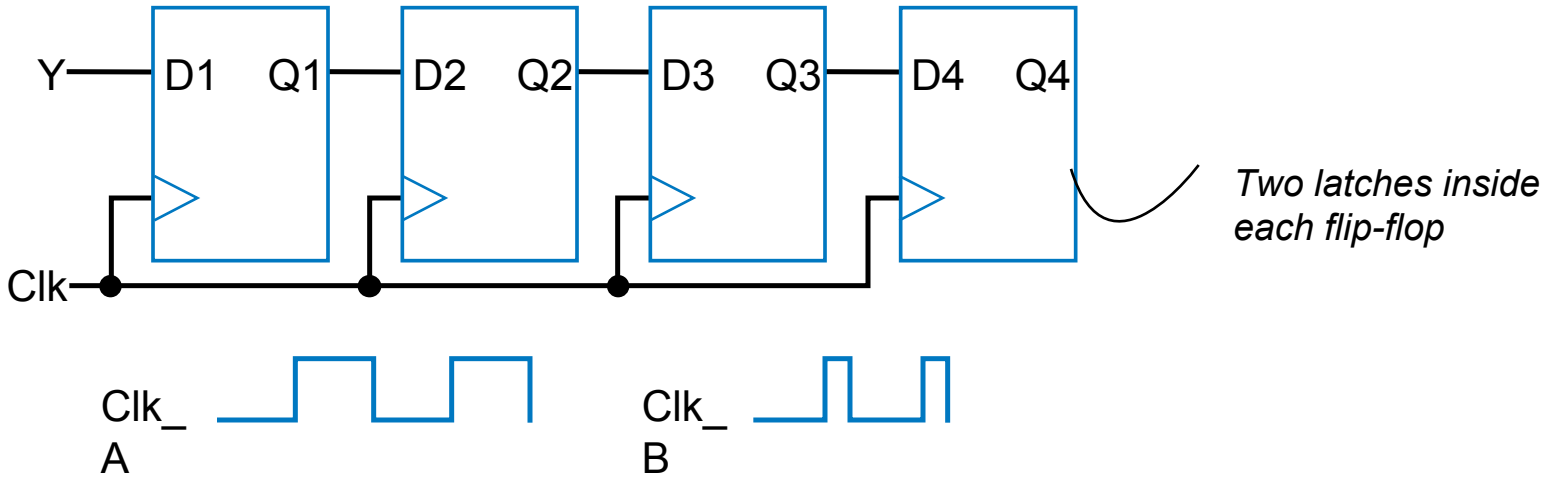
Internal design: Just invert servant clock rather than master





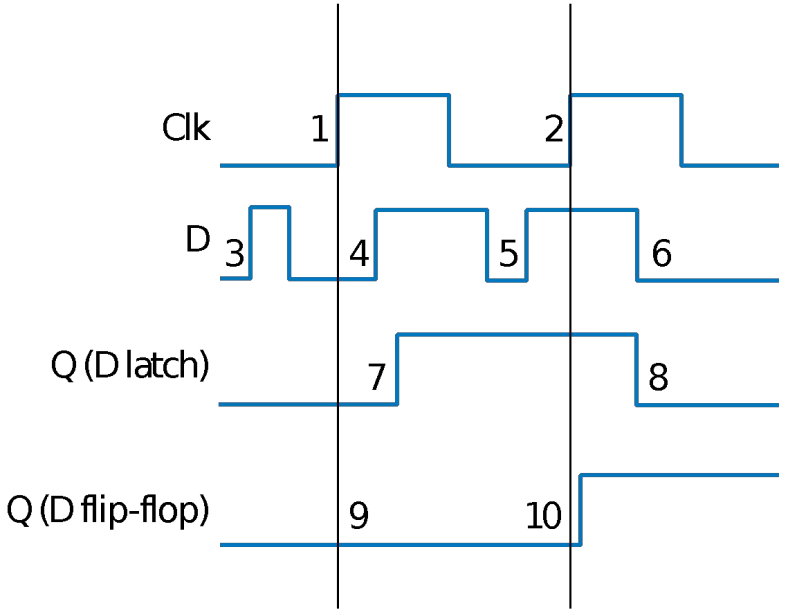
D Flip-Flop

- Solves problem of not knowing through how many latches a signal travels when C=1
  - In figure below, signal travels through exactly one flip-flop, for Clk_A or Clk_B
  - Why? Because on rising edge of Clk, all four flip-flops are loaded simultaneously -- then all four no longer pay attention to their input, until the next rising edge. Doesn't matter how long Clk is 1.

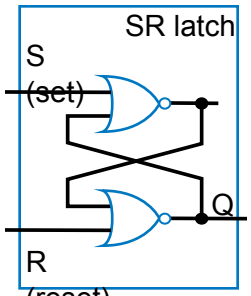


D Latch vs. D Flip-Flop

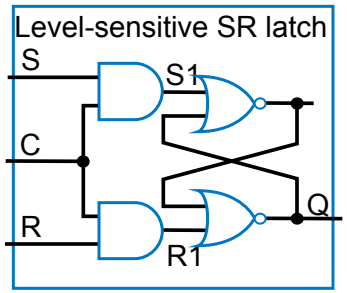
- Latch is level-sensitive: Stores D when C=1
- Flip-flop is edge triggered: Stores D when C changes from 0 to 1
  - Saying “level-sensitive latch,” or “edge-triggered flip-flop,” is redundant
  - Two types of flip-flops -- rising or falling edge triggered.
- Comparing behavior of latch and flip-flop:



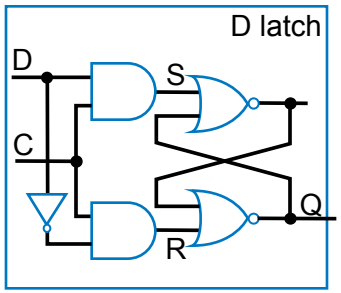
Bit Storage Summary



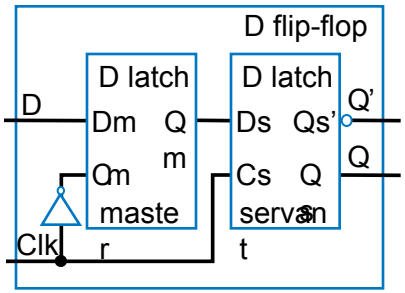
Feature: S=1 sets Q to 1, R=1 resets Q to 0. Problem: SR=11 yield undefined Q.



Feature: 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.



Feature: 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.



Feature: 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.

- We considered increasingly better bit storage until we arrived at the robust D flip-flop bit storage