



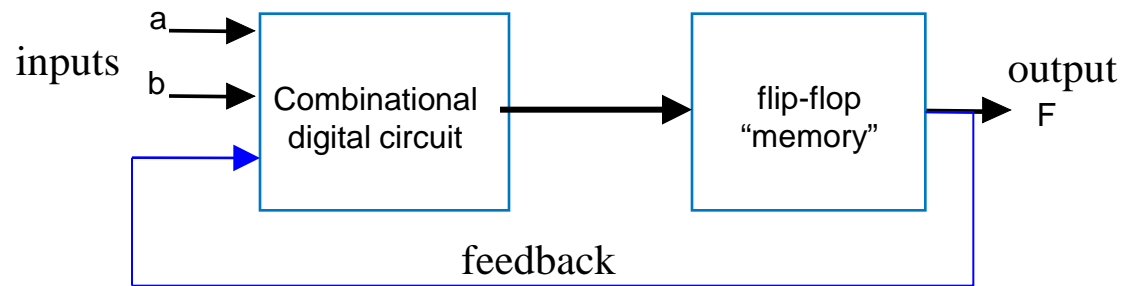
Slides to accompany the textbook *Digital Design*, First Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2007.
<http://www.ddvahid.com>

Instructors of courses requiring Vahid's Digital Design textbook (published by John Wiley and Sons) have permission to modify and use these slides for customary course-related activities, subject to keeping this copyright notice in place and unmodified. These slides may be posted as unanimated pdf versions on publicly-accessible course websites. PowerPoint source (or pdf with animations) may not be posted to publicly-accessible websites, but may be posted for students on internal protected sites or distributed directly to students by other electronic means. Instructors may make printouts of the slides available to students for a reasonable photocopying charge, without incurring royalties. Any other use requires explicit permission. Instructors may obtain PowerPoint source or obtain special use permissions from Wiley – see <http://www.ddvahid.com> for information.

Introduction

- Sequential circuit
 - Output depends not just on present inputs (as in combinational circuit), but also on the present and previous values of the output.
 - The output defines “the state” of the circuit.
- Examples:
 - A digital clock: the next new time value depends on the present time.
 - A timer: the next new time value depends on the present time AND whether the start button or stop button is pressed.
 - A garage door opener: the direction of the motor depends on the previous direction as well as the control inputs.
 - An up/down counter: the next value of the counter depends on the present count value and the value of the direction signal.

Sequential Circuit Model



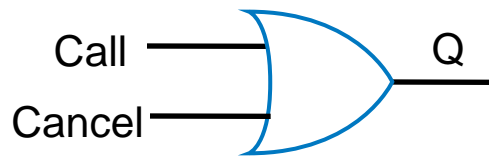
Chapter Overview

- In this chapter, we will:
 - Design a new building block, a **flip-flop**, that stores one bit
 - Combine that block to build multi-bit storage – a **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



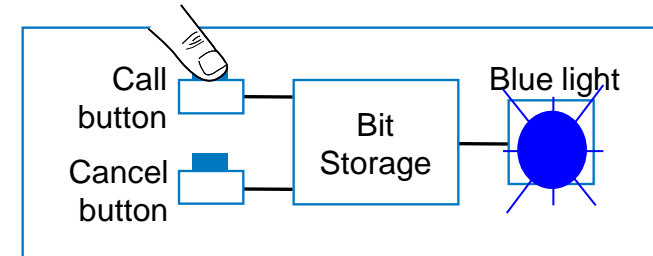
Example Needing Bit Storage

- 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?

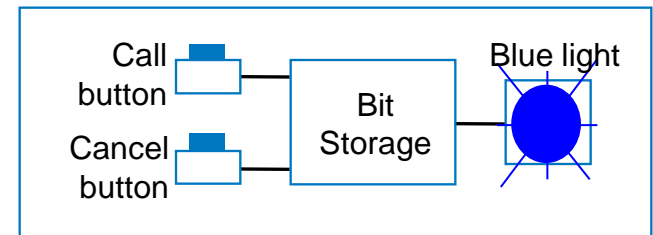


Doesn't work. $Q=1$ when $Call=1$, but doesn't stay 1 when $Call$ returns to 0

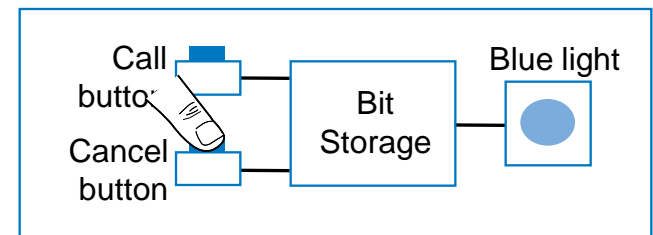
Need some form of "feedback" in the circuit



1. Call button pressed – light turns on



2. Call button released – light stays on

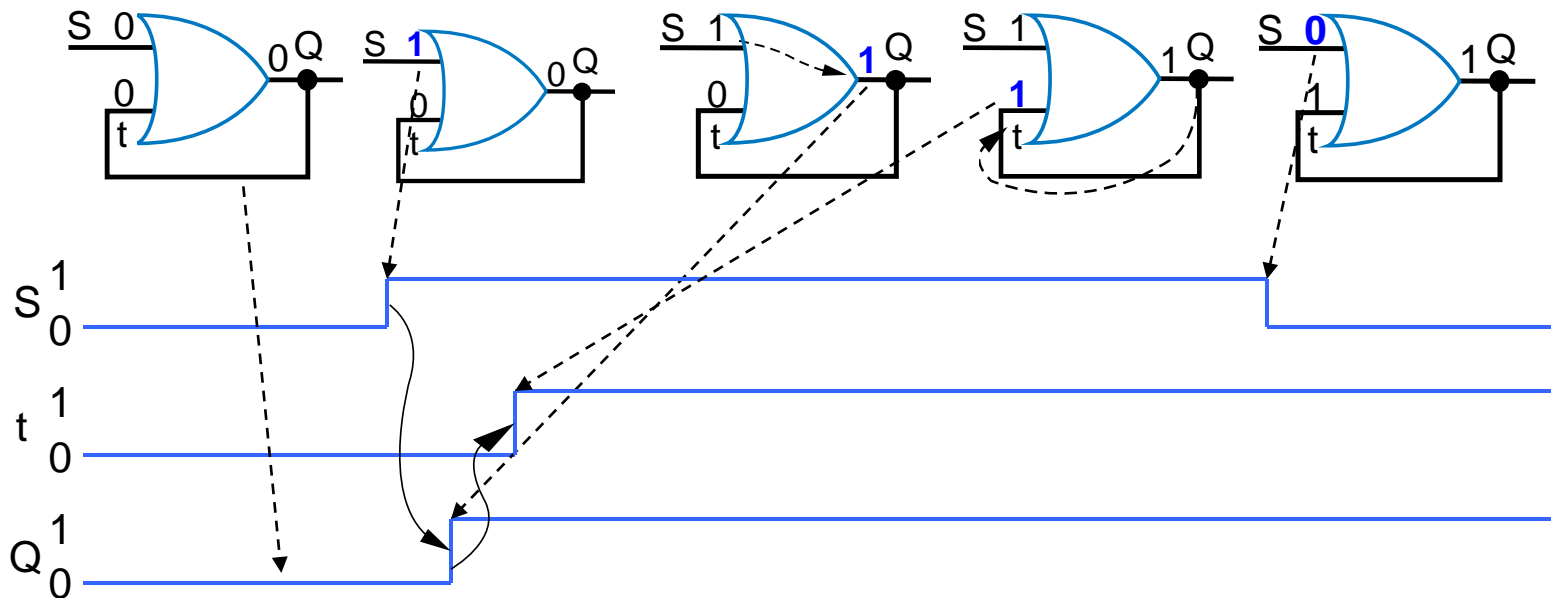
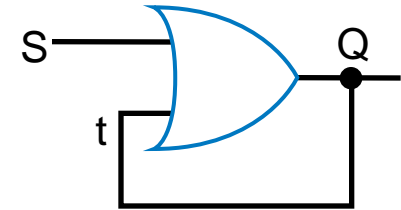


3. Cancel button pressed – light turns off



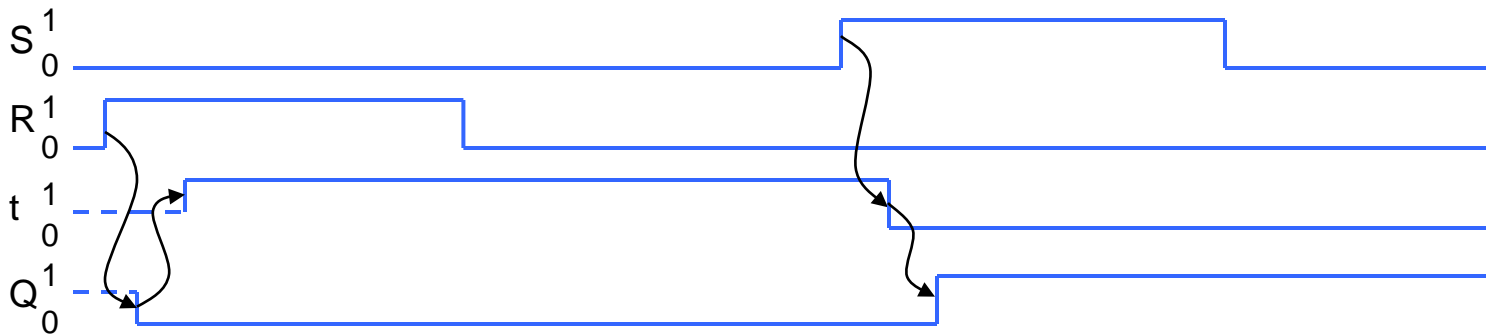
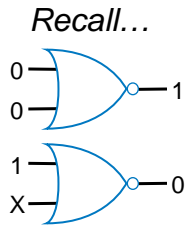
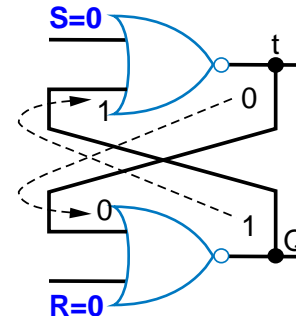
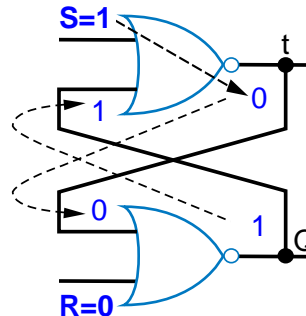
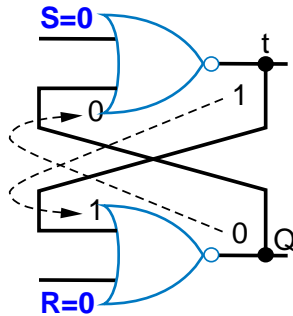
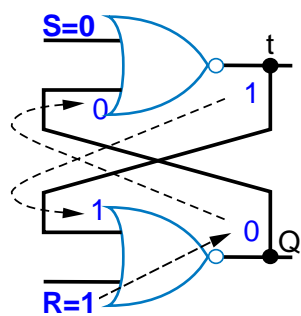
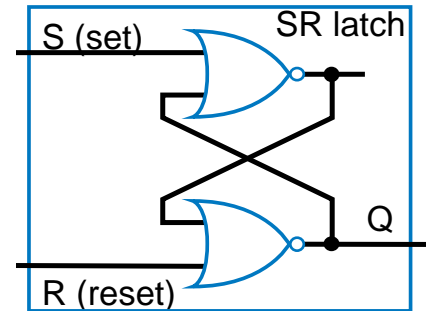
First attempt at Bit Storage

- 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



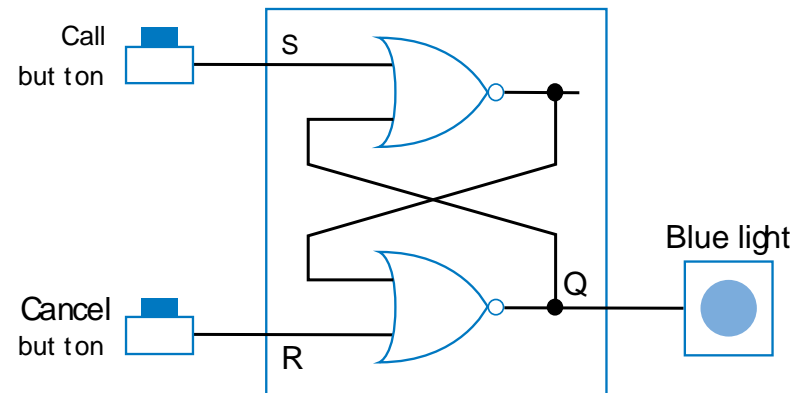
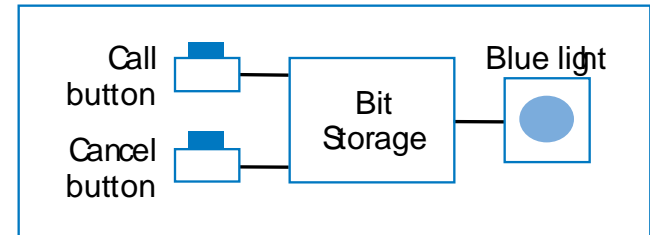
Bit Storage Using an SR Latch

- Does the circuit to the right, with cross-coupled NOR gates, do what we want?
 - SR stands for SET (1) – RESET (0)



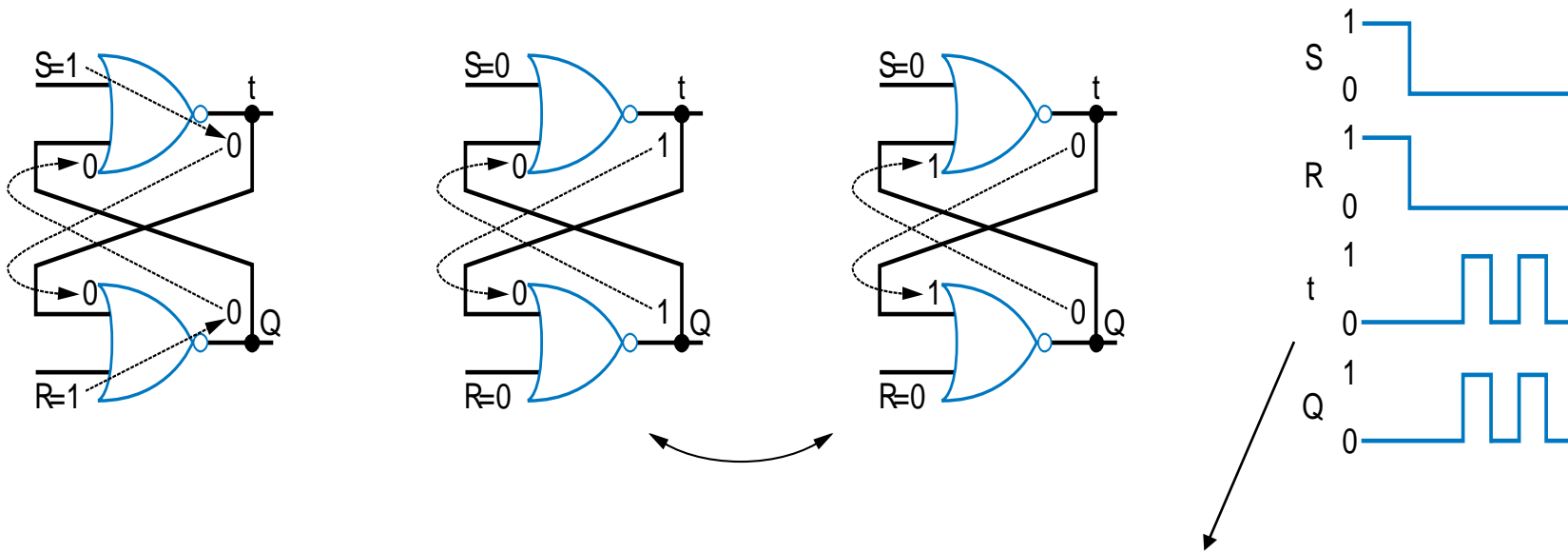
Example Using SR Latch for Bit Storage

- 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...

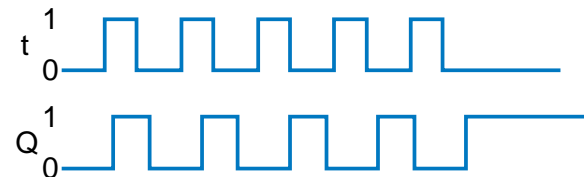


Problem with SR Latch

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

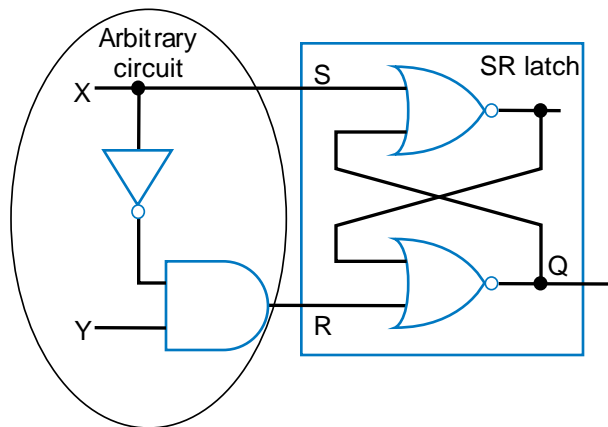


Q may oscillate. Then, because one path will be slightly longer than the other, Q will eventually settle to 1 or 0 – but we don't know which.

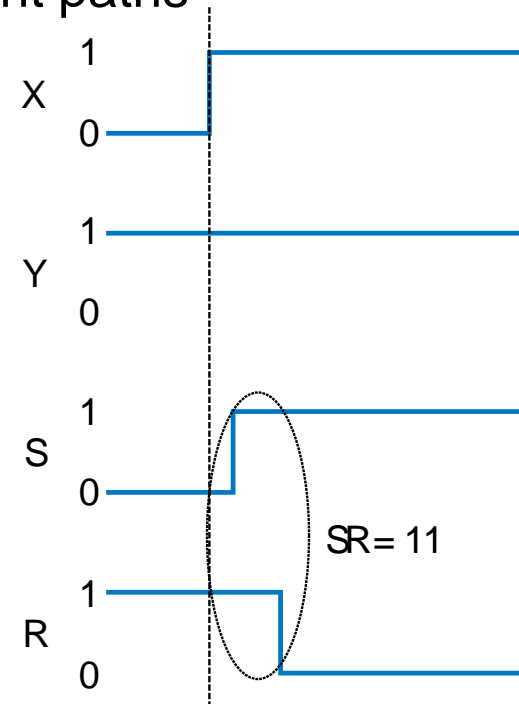


Problem with SR Latch

- 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

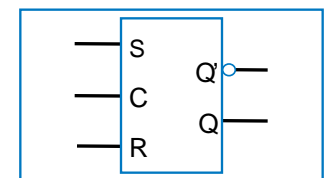
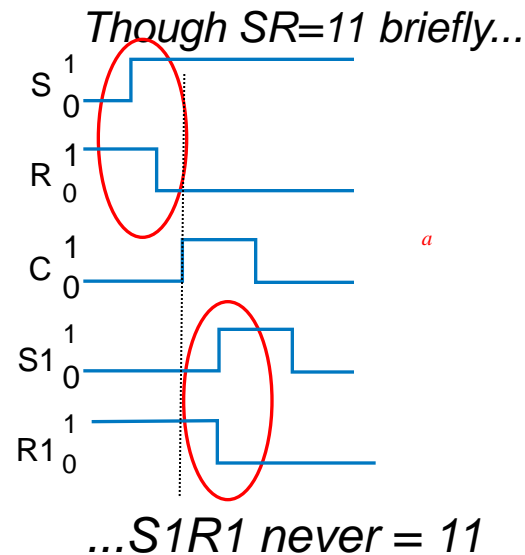
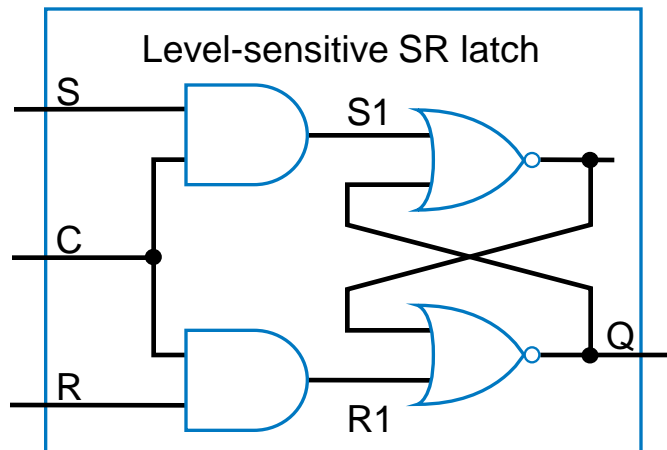


The longer path from X to R than to S causes $SR=11$ for short time – could be long enough to cause oscillation



Solution: Level-Sensitive SR Latch

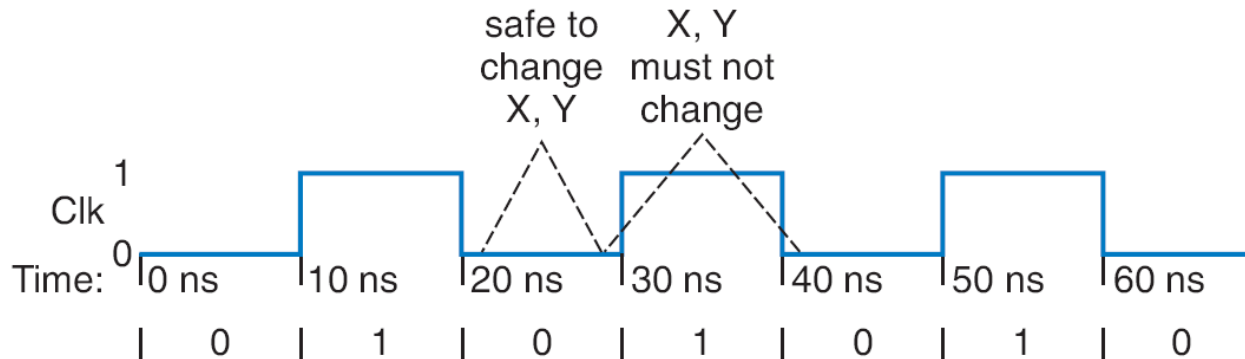
- Add enable input C (clock) as shown
 - Only let S and R change when C=0
 - Ensure circuit in front of SR never sets $SR=11$, except briefly due to path delays
 - Change C to 1 only after sufficient time for S and R to be stable
 - When C becomes 1, the stable S and R value passes through the two AND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



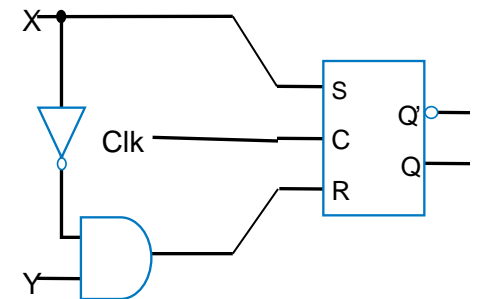
Level-sensitive SR latch symbol



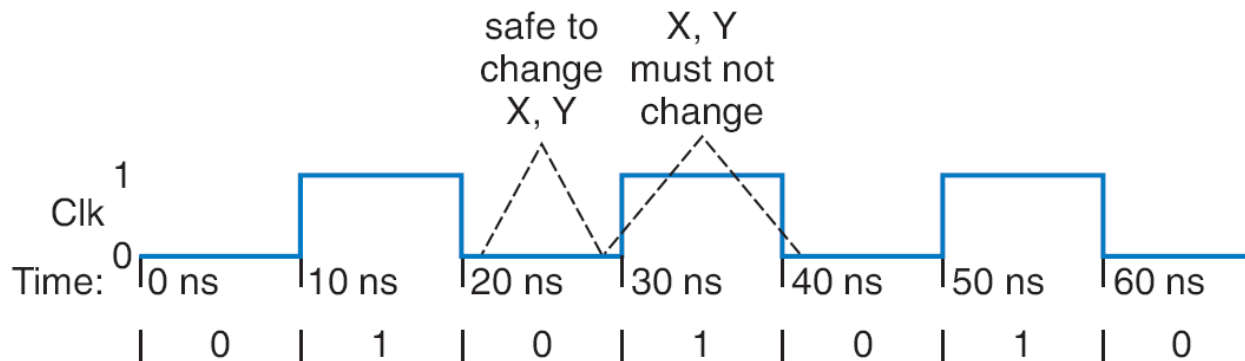
Clock Signals for a Latch



- 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, i.e., output changes are *synchronous* with the clock.



Clock Terms



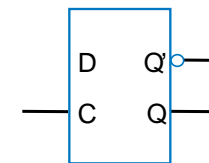
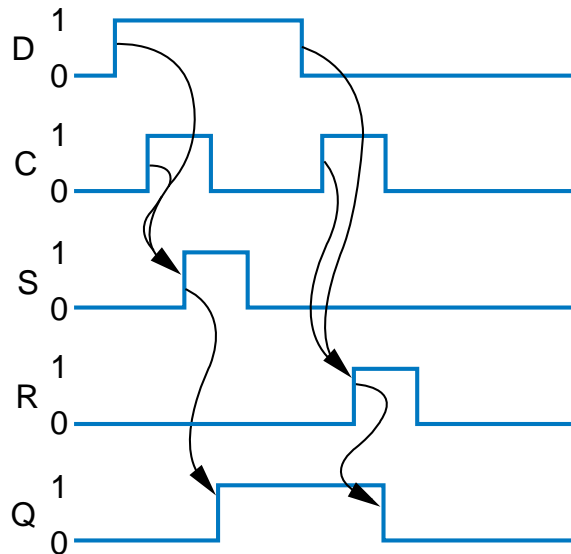
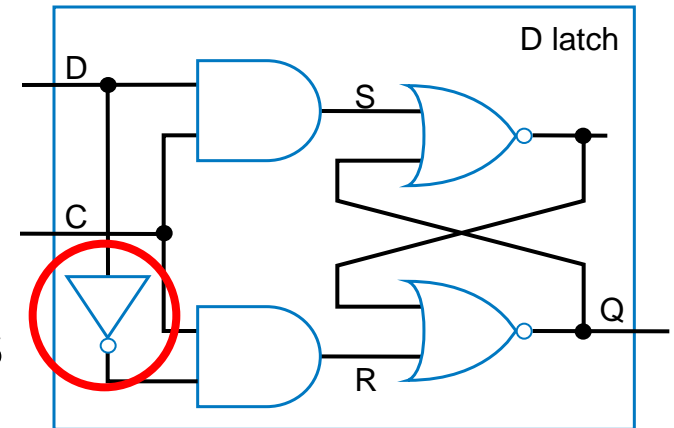
- **Clock period:** time interval between pulses
 - Above signal: period = 20 ns
- **Clock frequency:** $1/\text{period}$
 - Above signal: frequency = $1 / 20 \text{ ns} = 50 \text{ MHz}$
 - $1 \text{ Hz} = 1/\text{s}$
- **Timing Diagram:** graphical representation of inputs/outputs with respect to time

| 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 (data) latch relieves designer of that burden
 - inverter ensures R always opposite of S



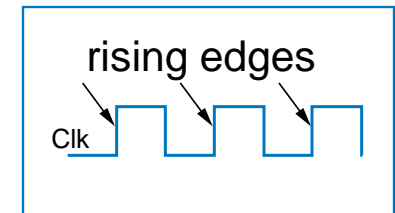
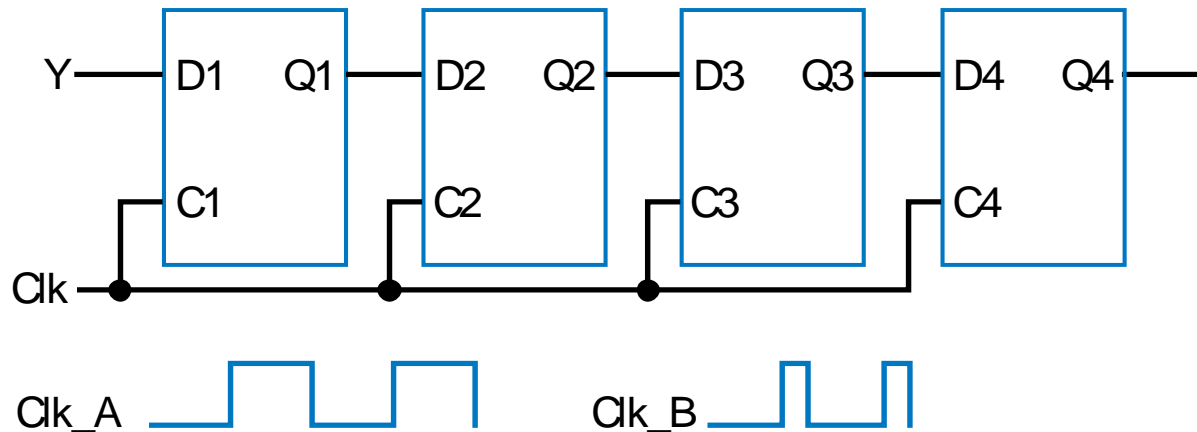
D latch symbol

D latch symbol



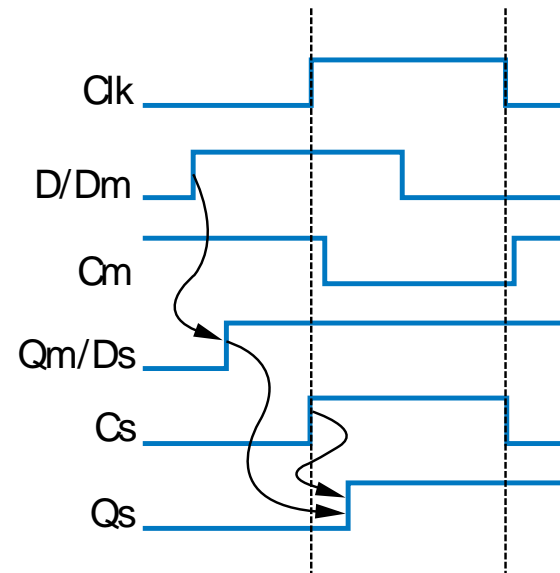
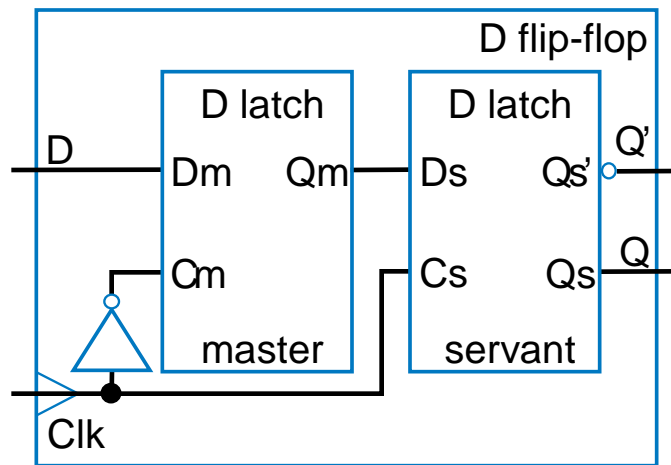
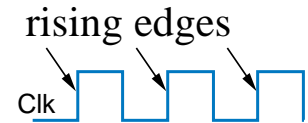
Problem with Level-Sensitive D Latch

- D latch still has problem
 - When $C=1$, through how many latches will a signal travel?
 - Depends on 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 edge of a clock signal?



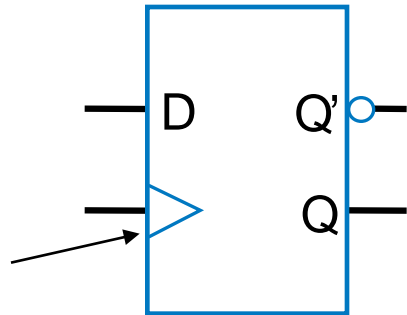
D (data) Flip-Flop

- **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

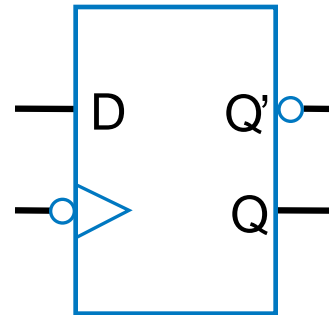
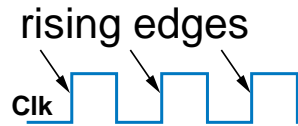


D Flip-Flop Symbol Notation

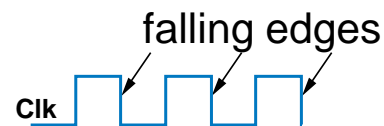
The triangle means clock input, edge triggered



Symbol for rising-edge triggered D flip-flop

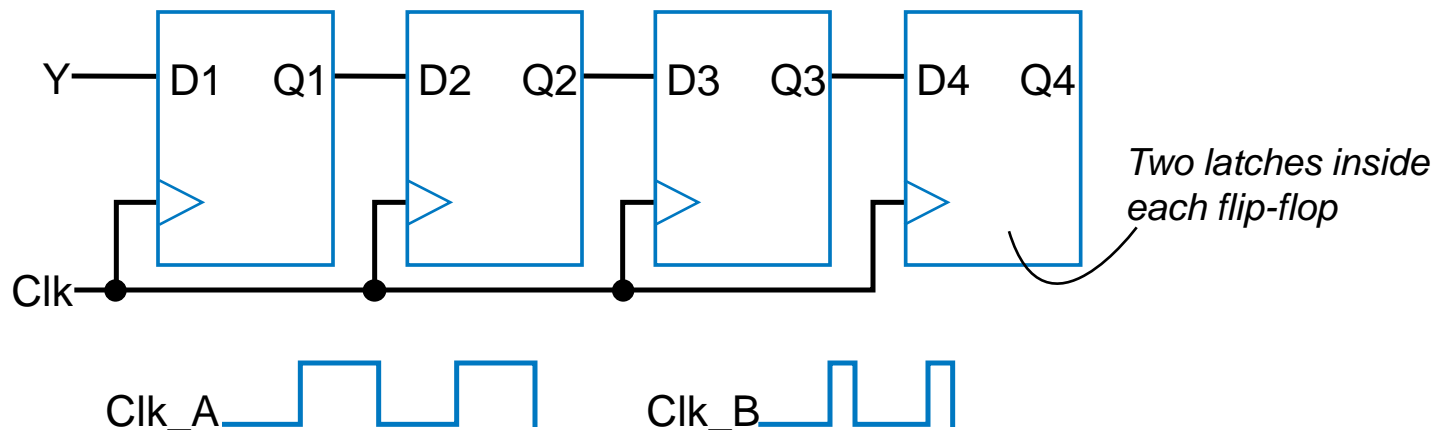


Symbol for falling-edge triggered D flip-flop



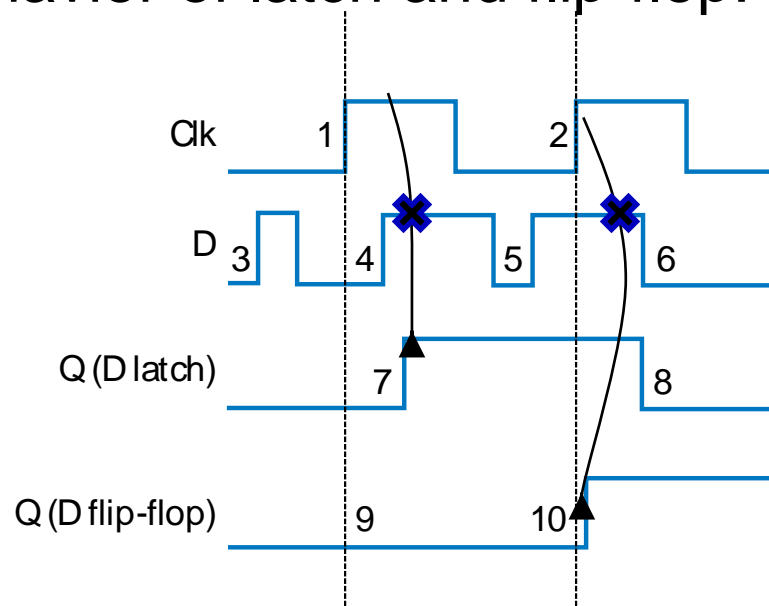
D Flip-Flop

- 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:



Flight-Attendant Call Button Using D Flip-Flop

- D flip-flop will store bit
- Inputs are Call, Cancel, and present value of D flip-flop, Q
- Truth table shown below

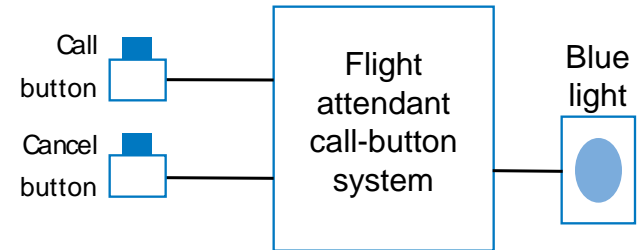
| Call | Cancel | Q | D |
|------|--------|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Preserve value: if
Q=0, make D=0; if
Q=1, make D=1

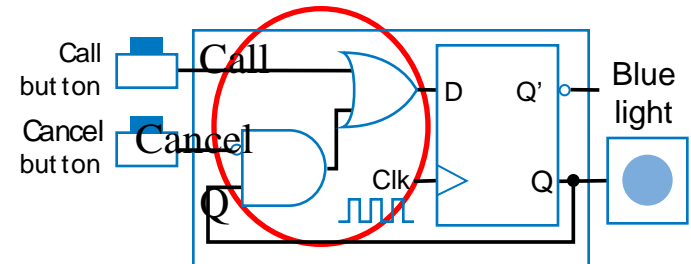
Cancel -- make
D=0

Call -- make D=1

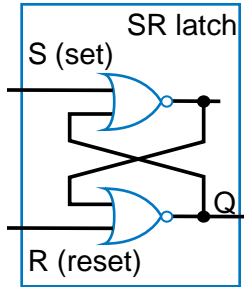
Let's give priority
to Call -- make
D=1



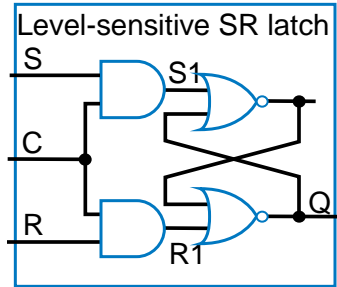
Circuit derived from truth table,
using Chapter 2 combinational
logic design process



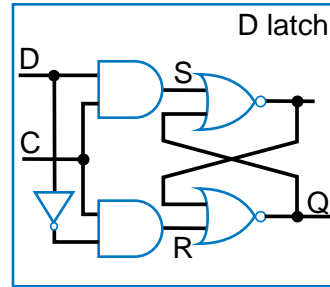
Bit Storage Summary



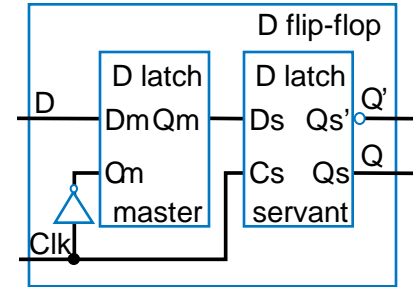
Feature: $S=1$ sets Q to 1, $R=1$ resets Q to 0. Problem: $SR=11$ yield undefined Q .



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.



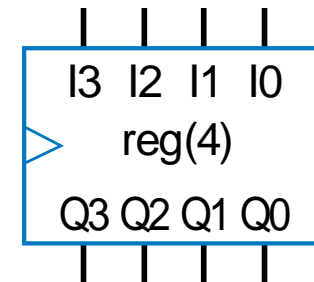
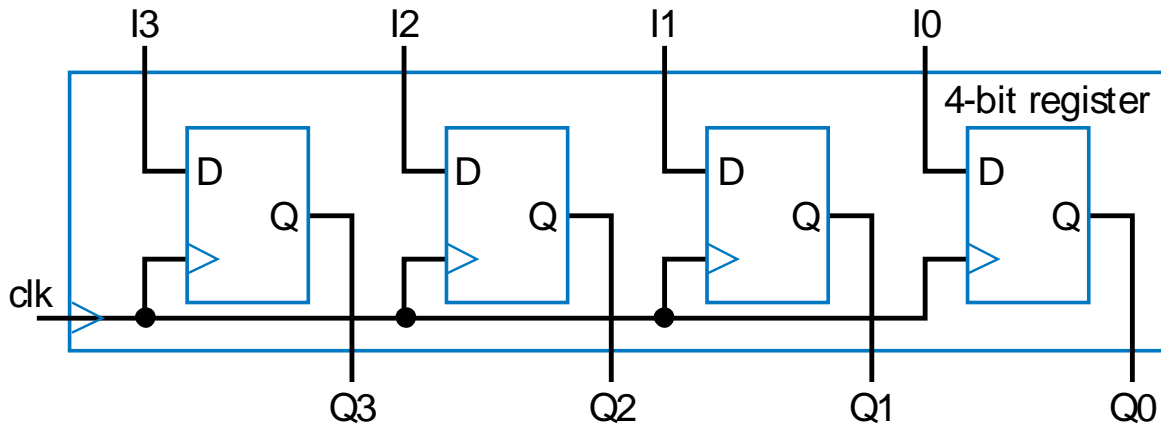
Feature: Only loads D value present at rising clock edge, so values can't propagate to other flip-flops during same clock cycle. Tradeoff: uses more gates internally than D latch, and requires more external gates than SR – but gate count is less of an issue today.

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



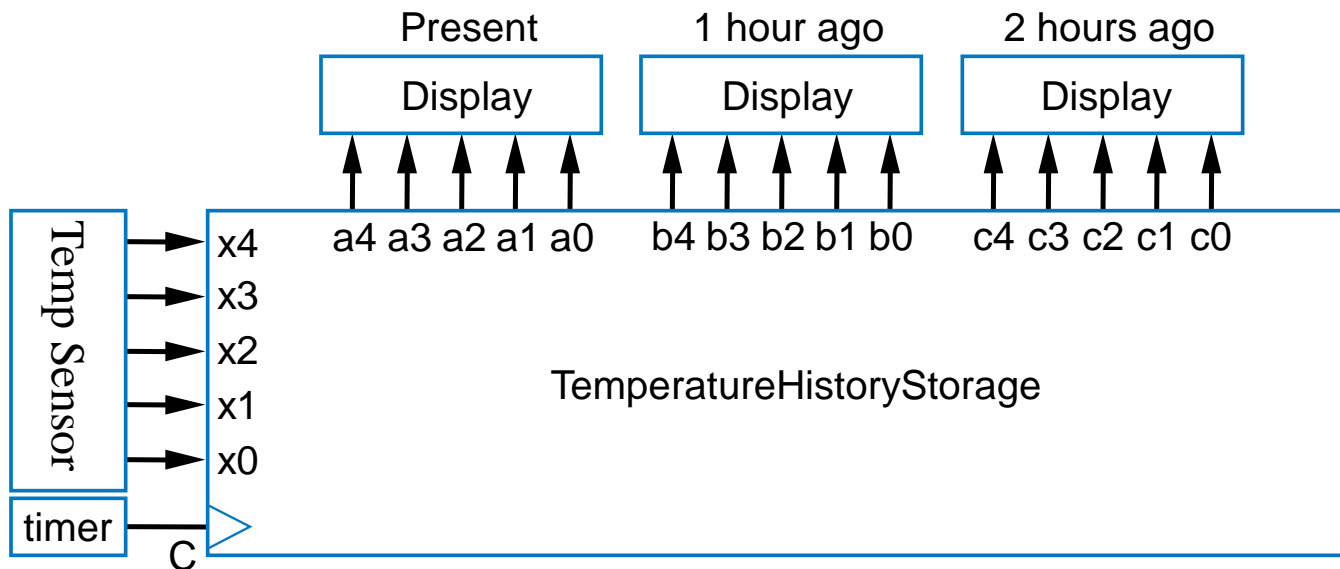
Basic Register

- Typically, we store multi-bit items
 - e.g., storing a 4-bit binary number
- **Register**: multiple flip-flops forming a single entity with the same clock signal



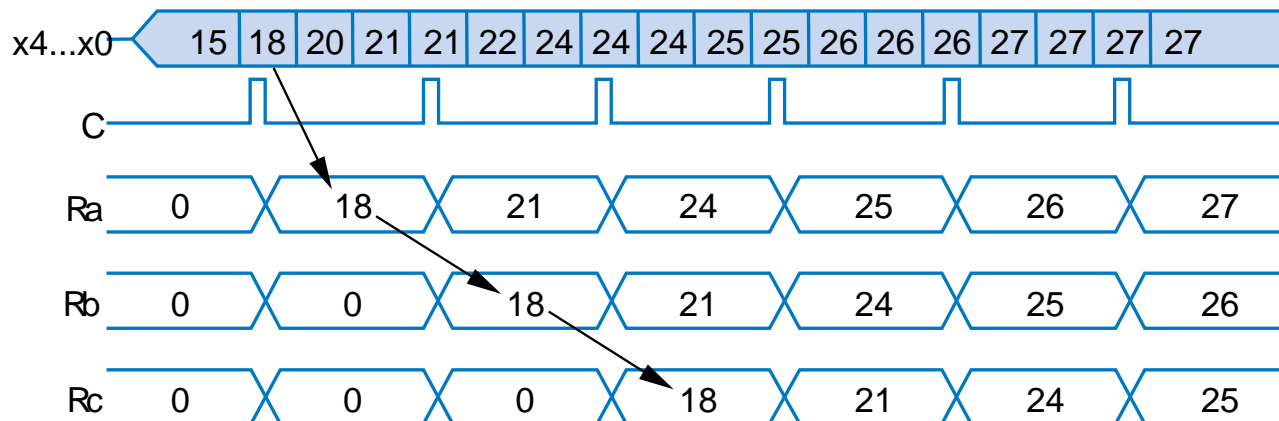
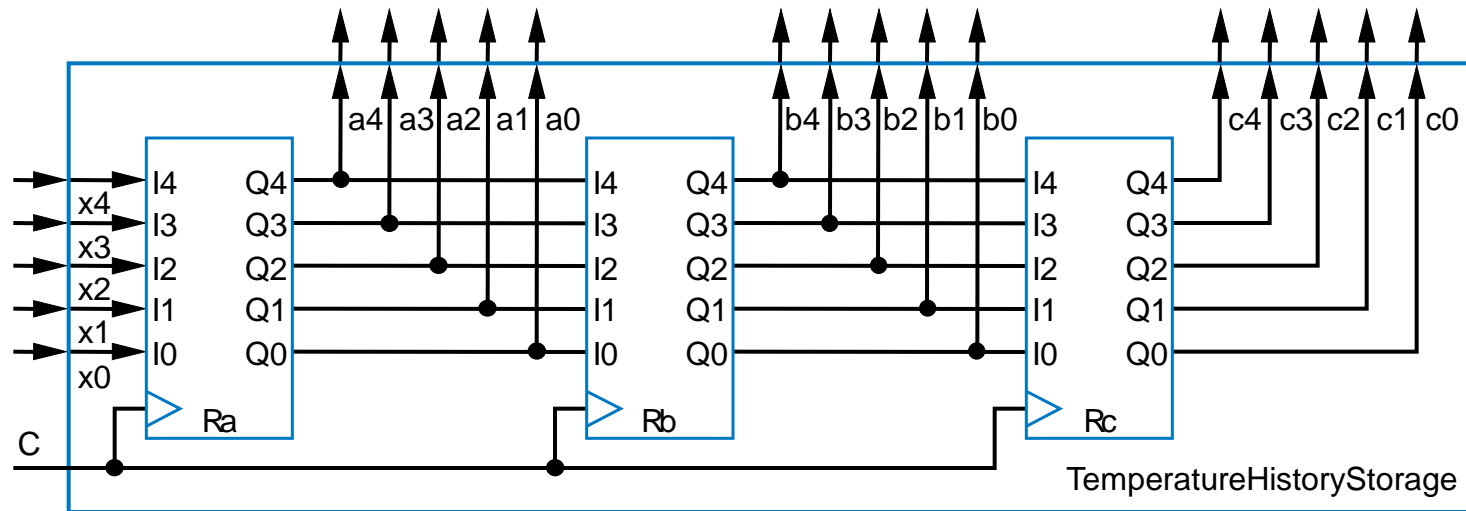
Example Using Registers: Temperature Display

- Temperature history display
 - Sensor outputs temperature as 5-bit binary number
 - Timer pulses C every hour
 - Record temperature on each pulse, display last three recorded values



Example Using Registers: Temperature Display

- Use three 5-bit registers



Other Flip-Flops

- Previously utilized to minimize logic outside flip-flop
 - Today, minimizing logic to such extent is not as important
 - D flip-flops are by far the most common



JK Flip-Flop

– JK flip-flop:

- $J=0, K=0 \rightarrow \text{HOLD}, Q_{(n+1)} = Q_{(n)}$, where $n+1$ = after the clock edge
- $J=0, K=1 \rightarrow \text{CLEAR}, Q_{(n+1)} = 0$
- $J=1, K=0 \rightarrow \text{SET}, Q_{(n+1)} = 1$
- $J=1, K=1 \rightarrow \text{TOGGLE}, Q_{(n+1)} = Q'_{(n)}$

| J | K | $Q_{(n+1)}$ |
|---|---|-------------|
| 0 | 0 | $Q_{(n)}$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $Q'_{(n)}$ |



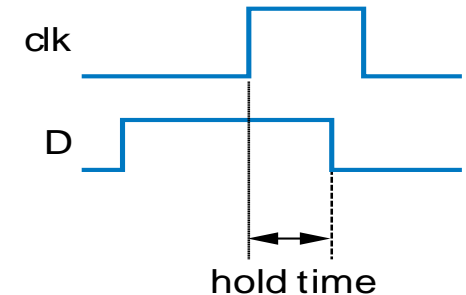
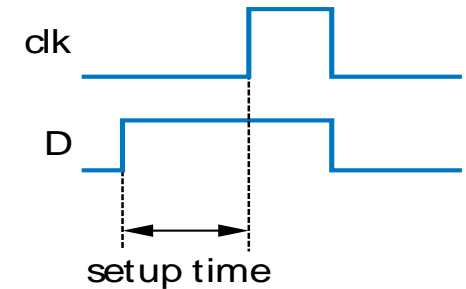
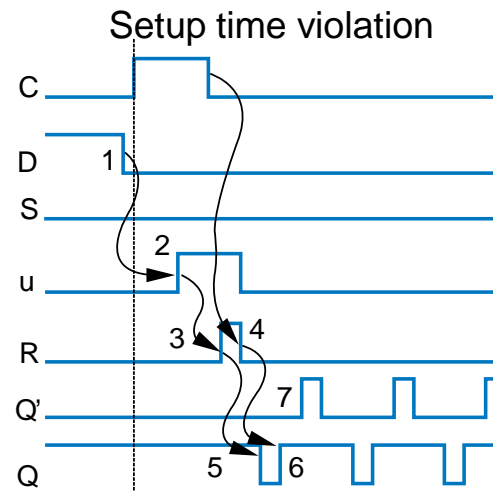
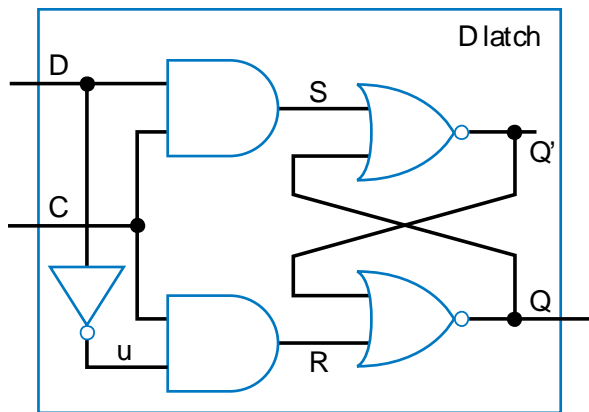
T (toggle) Flip-Flop

- T flip-flop:
 - $T=0 \rightarrow \text{HOLD}, Q_{(n+1)} = Q_{(n)}$
 - $T=1 \rightarrow \text{TOGGLE}, Q_{(n+1)} = Q'_{(n)}$
- SR flip-flop: like SR latch, but edge triggered



Non-Ideal Flip-Flop Behavior

- Can't change flip-flop input too close to clock edge
 - Setup time: time that D must be stable *before* edge
 - Else, stable value not present at internal latch
 - Hold time: time that D must be held stable *after* edge
 - Else, new value doesn't have time to loop around and stabilize in internal latch

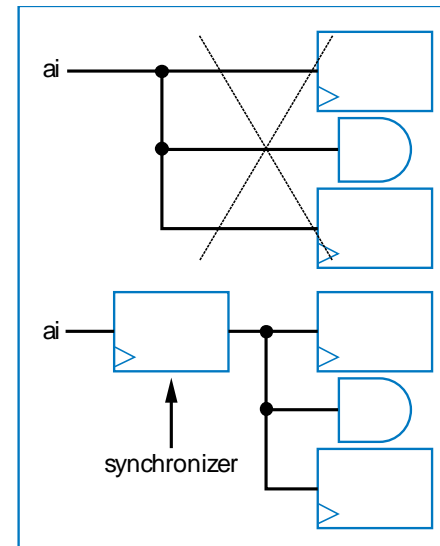
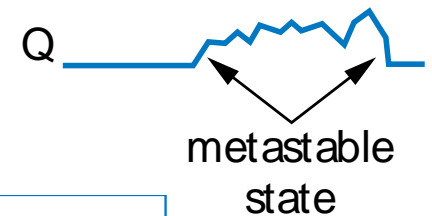
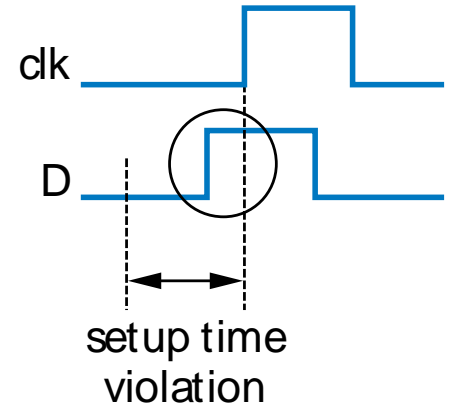


Leads to oscillation!



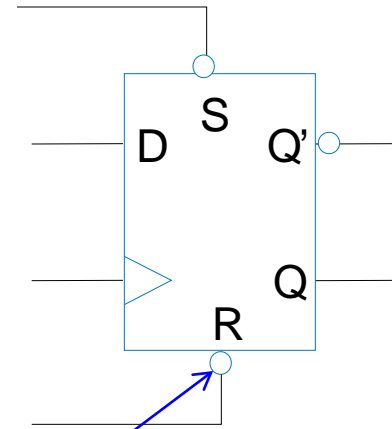
Metastability

- Violating setup/hold time can lead to bad situation known as **metastable** state
 - Metastable state: Any flip-flop state other than stable 1 or 0
 - Eventually settles to one or other, but we don't know which
 - For internal circuits, we can make sure observe setup time
 - But what if input comes from external (asynchronous) source, e.g., button press?
- Partial solution
 - Insert synchronizer flip-flop for asynchronous input
 - Special flip-flop with very small setup/hold time
 - Doesn't completely prevent metastability



Asynchronous Inputs

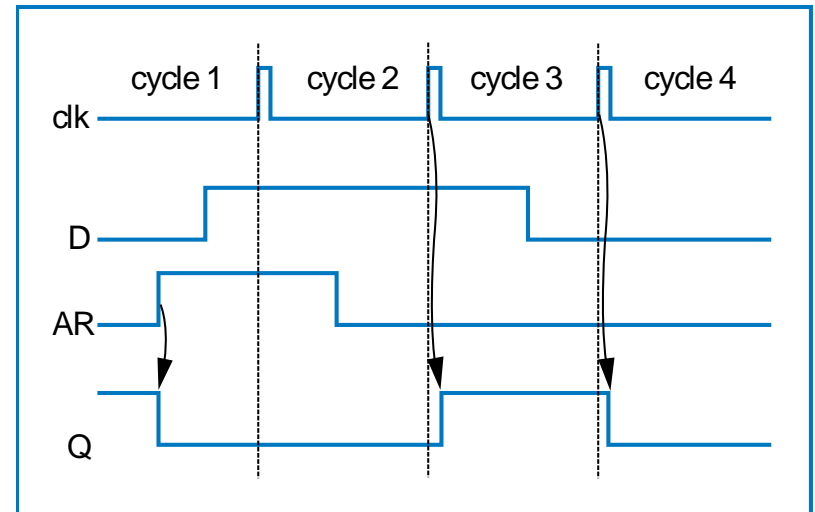
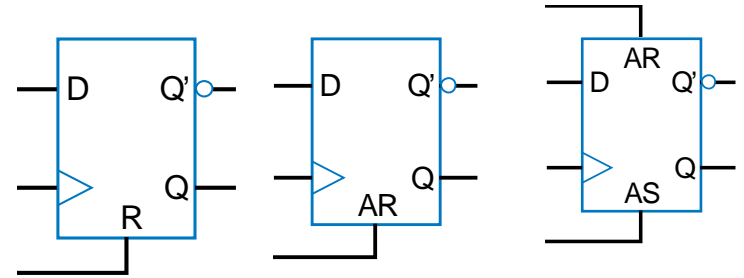
- Asynchronous: inputs that cause the output to change independent of the clock.
- Often used for reset, ie., force the output to a known state



Note low voltage assertion

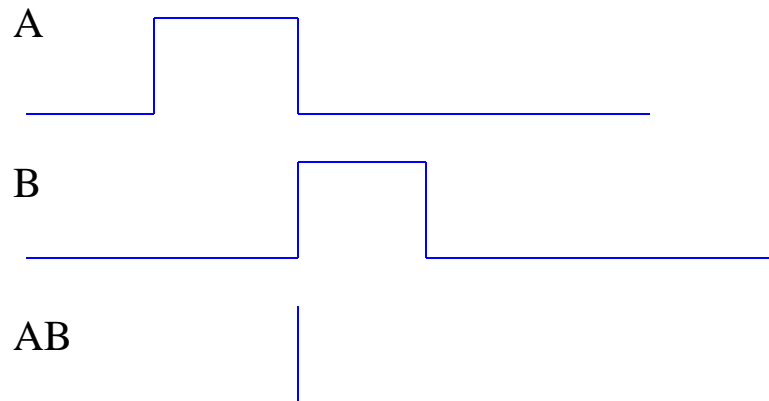
Flip-Flop Set and Reset Inputs

- Some flip-flops have additional inputs
 - Synchronous reset: clears Q to 0 on next clock edge
 - Synchronous set: sets Q to 1 on next clock edge
 - Asynchronous reset: clear Q to 0 immediately (not dependent on clock edge)
 - Example timing diagram shown
 - Asynchronous set: set Q to 1 immediately



Glitching

- Glitch: Temporary values on outputs that appear soon after input changes, before stable new output values
- Designer must determine whether glitching outputs may pose a problem
 - If so, may consider adding flip-flops to outputs
 - Delays output by one clock cycle, but may be OK



Chapter Summary

- Sequential circuits
 - Have state
- Latches: level sensitive, transparent output
- Flip-Flops: Edge sensitive, clock signal
- Created robust bit-storage device: D flip-flop
 - Put several together to build register
- Asynchronous signals
 - outputs change independent of clock signal



Homework

- Problems: chapter 3 – 8, 12, 14, 19, 22, 47, 48
- Homework is due on Thursday, February 25