Introduction to Memory - Part#1

PART
1) 기본적인 메모리 내용
2) 다양한 메모리들이 어떻게 컴퓨터 시스템에 활용되는가 (+체계)

Yoonjin Kim

Full Professor

Division of Computer Science
Sookmyung Women's University

(컴퓨터 구조)

Outline

컴퓨터 구성하는 CPU와 근접해있는 register에 대해 이야기한 것.

- **From latch to flip-flop for one-bit storage** → 확장 → 동기화, CLK
  - latch ~ flip-flop 까지 1bit가 어떻게 저장되는가
    - register의 1bit를 구성함.
- **Register for multi-bit storage**
  - 1bit → multi-bit 저장하는 Register.
    - 확장

(확장과정)

flip-flop → CLK → 동기화회로 → 디지털시스템설계

(1bit storage)

Memory hierarchy (메모리 계층)

DRAM ≠ 2차메모리

(≠ 용량 16GB, ≠ Register 처럼 휘발성있는 메모리)

>>>> SSD, 하드디스크 (보조기억장치)

= CPU에서 가장 먼 메모리.

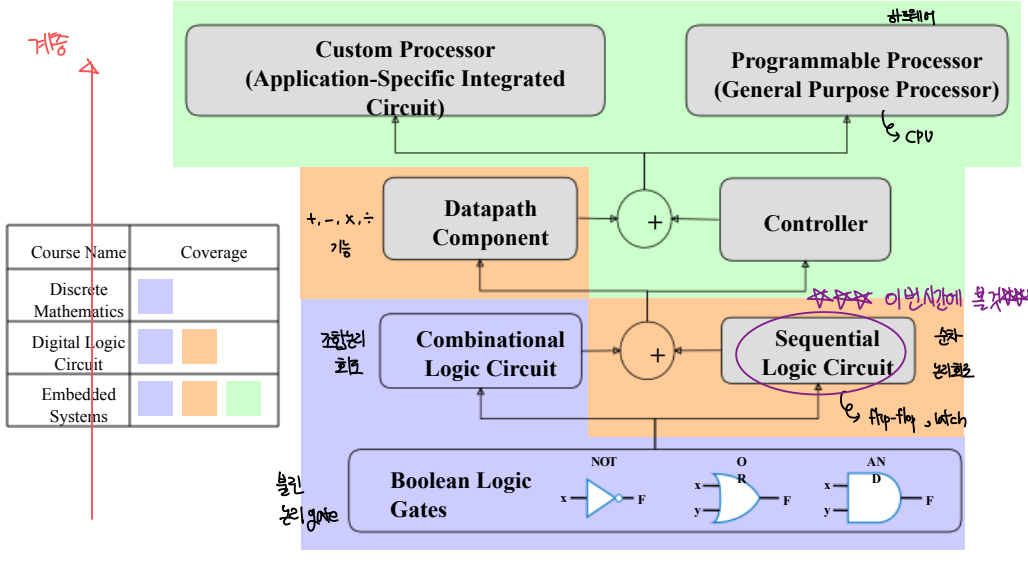
↳ 오늘은 컴퓨터 프로그램이 가야할 길을 방향.

From Latch to Flip-Flop for One-bit Storage

1. Basic principle how to store a bit data based on logic gates
   ↳ AND, OR, NOT gate.
2. Necessity of synchronization (Storage → 동기화)
   : Concept of 'Clock'
   ↳ 논리적·물리적 요소를 파악하기 위한 것. (논리회로에서)
3. Necessity of more elaborated synchronization (더 정교한 동기화)
   : Concept of rising edge of 'Clock'

⇒ CLK에 의해 디지털 회로가 구성됨.
↳ Storage를 위한 개념.

Digital Circuit Design Hierarchy



Course Name	Coverage
Discrete Mathematics	
Digital Logic Circuit	
Embedded Systems	

Introduction

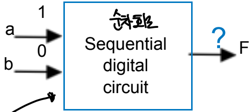
Sequential circuit

Output depends not just on present inputs (as in combinational circuit), but on past sequence of inputs.

Stores bits, also known as having "state"

In this section, we will:

Design a new building block, **a flip-flop**, that stores one bit.



Must know sequence of past inputs to know output

1) flip-flop이 어떻게 만들어지는가 (설명가능하도록)

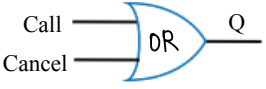
2) latch vs flip-flop 차이점이 무엇인가

이때 나와야하는 개념 Clock

latch → flip-flop

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?



⇒ don't work

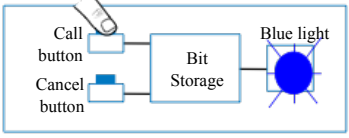
Doesn't work. Q=1 when Call=1, but doesn't stay 1 when Call returns to 0.

Need some form of "feedback" in the circuit

In addition, Q=1when Cancel=1.

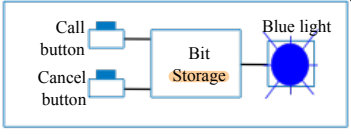
– Malfunction: Q should be 0 when Cancel= 1.

Bit storage를 만들어보자!

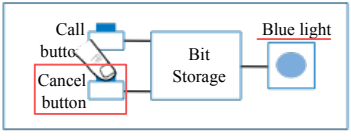


1. Call button pressed – light turns on

유지



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



3. Cancel button pressed – light turns off

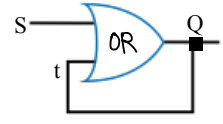
First attempt at Bit Storage

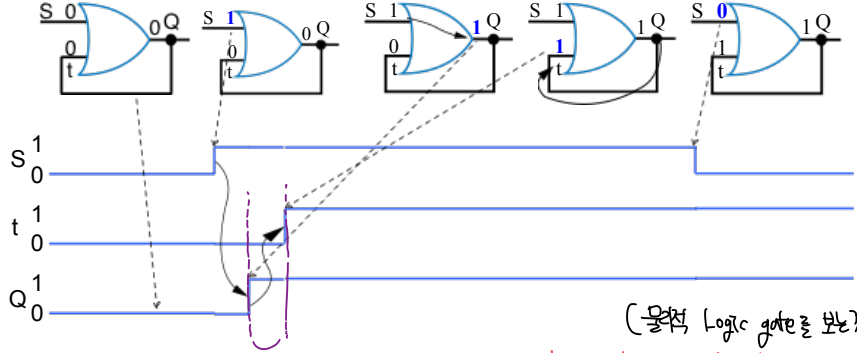
OR Gate

S	t	결과값Q
0	0	0
0	1	1
1	0	1
1	1	1

Q가 영원히 1일 ⇒ 우리가 원하는 결과

- 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.





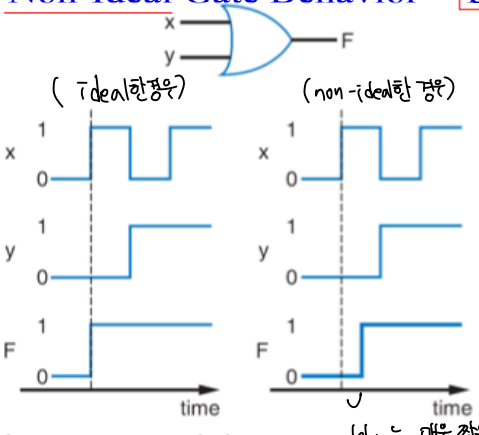
Q가 변화할 때 생긴 delay는 거의 0 가 됨.

(물적 Logic gate를 보는것)

값이 hold 되는 것까지만 가능.

Additional Considerations

Non-Ideal Gate Behavior -- Delay



delay는 매우 짧은 시간임.

- Real gates have some delay
  - Outputs don't change immediately after inputs change

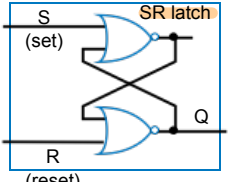
↳ delay가 설계 수 밖에 없음.

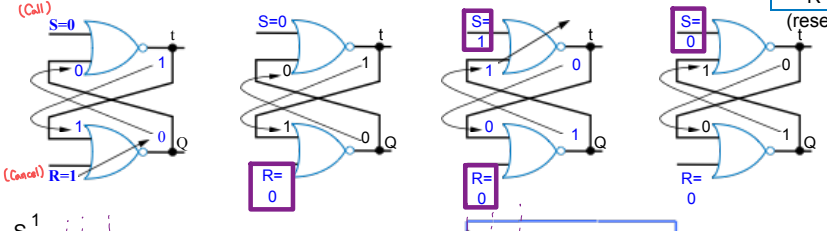
Bit Storage Using an SR Latch

논리적 비약

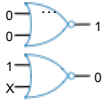
(set · reset)

- 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



NOR gate

A	B	출력
0	0	1
0	1	0
1	0	0
1	1	0

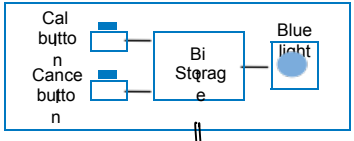
결과 NOR gate 2개로 값을 저장하는 storage 만듦

0

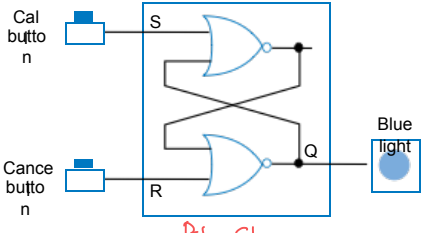
Example Using SR Latch for Bit Storage

- 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...
  ↳ latch의 문제가 있음



SR latch

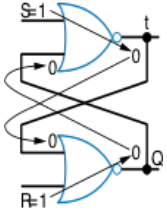


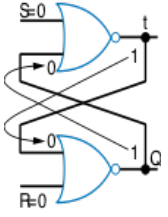
Bit Storage

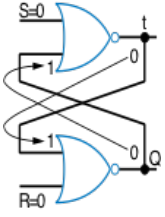
latch
(사전적의미) 걸쇠 , 잠그다 참고다
⇒ bit를 잡아두다
⇒ Storage (저장)

Problem with SR Latch

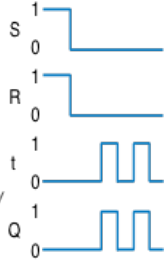
- Problem ⇒ 정상에 S=1 , R=1 을 누르고 , 때면 문제 발생 ⇒ blue light ⇒ 무한 반복으로 깜빡거림.
  - If S=1 and R=1 simultaneously, we don't know what value Q will take



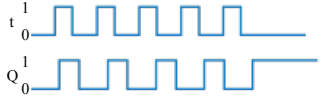




무한 반복.

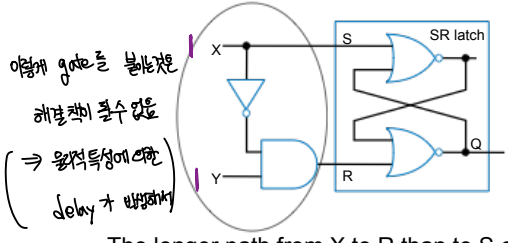


Leads to oscillation!

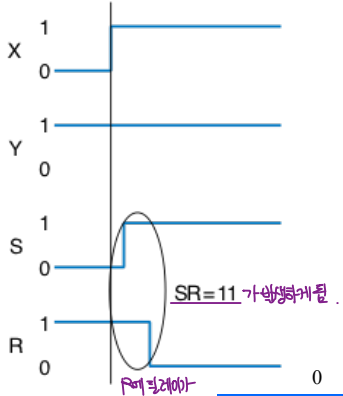


Problem with SR Latch

- 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

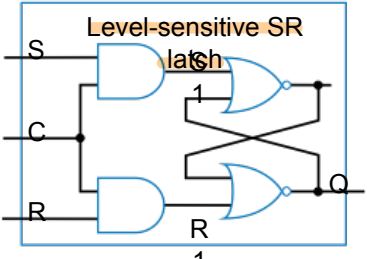


The longer path from X to R than to S causes SR=11 for short time – could be long enough to cause oscillation.

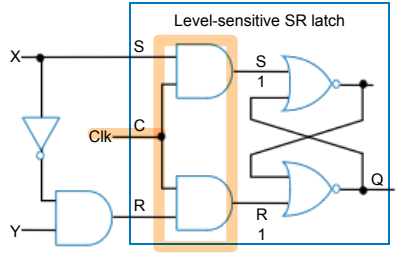


Solution: Level-Sensitive SR Latch

- 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
  - 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.

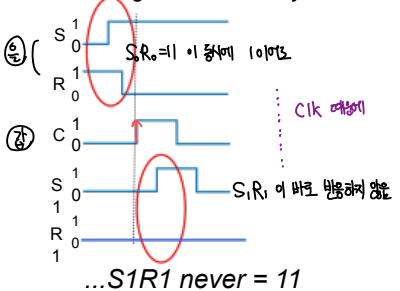


↳ 동시에 1이 되는 경우가 없어짐. ↳ oscillation되는 경우를 막아줌.

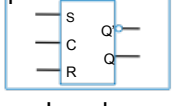


↳ Clk이 허용한 때만 , S와 R에 원하는 값을 넣을 수 있음.

Though SR=11 briefly...



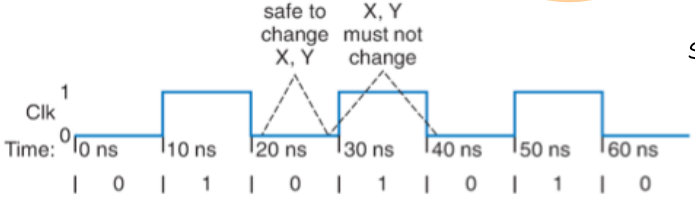
...S1R1 never = 11



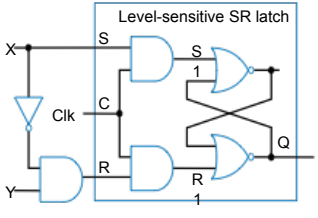
Level-sensitive SR latch symbol

Clock Signals for a Latch

CLK은 일정하게 값이 들어감.
(fixed)
⇓
SR값은 CLK을 보고 판비사람.



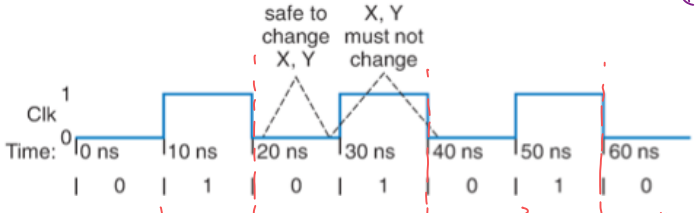
- How do we know when it's safe to set C=1?
  - Most common solution –make C pulse up/down
    - C=0: Safe to change X, Y
    - C=1: Must *not* change X, Y
  - ***Clock*** signal -- Pulsing signal used to enable latches
    - Because it ticks like a clock
  - Sequential circuit whose storage components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



CLK signal을 이용한 Sequential Circuit 11 Synchronous circuit (동기화회로)

Clocks

전체시간 = (CLK period) X (CLK cycle)



개념

20 ns X 3.5 = 70ns

⇒ 전체 시간을 알수있게됨

Clock period: time interval between pulses

Above signal: period = 20 ns

Clock cycle: one such time interval

Above signal shows 3.5 clock cycles

⇒ clk 주기가 몇번 반복되는가

Clock frequency: 1/period

Above signal: frequency = 1 / 20 ns = 50 MHz

1 Hz = 1/1sec – period is 1sec.

1 GHz = 103 MHz = 109 Hz = 1/ (10-9 sec) = 1/1ns – period is 1ns.

↳ 1ns의 주기를 가진 clk의 주파수 Hz = 1 GHz

10^{-9} 10^{9}

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

More Complex Bit Storage System

- Previous example - Flight attendant call button
  - Level-Sensitive SR-Latch is enough to design the button system.
  - Because the system is very simple!

이전에 배운 예제

- More complex bit storage system = 더 복잡한 bit storage system.
  - For example,
    - Some storages transfer bit-data to other storages. ⇒ storage 끼리의 data 이동
    - Several storages exchange bit-data together. ⇒ storage 끼리의 data 교환
  - Level-Sensitive Latch can perfectly support such systems?

↳ 이러한 경우에서 flip-flop이 꼭 필요한지 생각할.

Timeline : 1시간

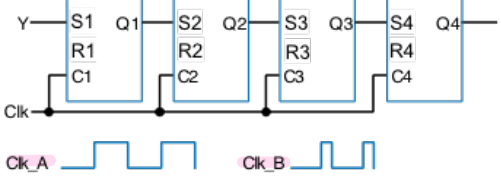
More Complex Bit Storage System

↳ SR Latch는 제어에 어려움이 있음.

- Difficulties controlling level-sensitive SR latch ⇒ 너무많은권한을 부여함.
  - We allow level-sensitive SR latch to have too much power.
    - Timeless storing capability based on two inputs (S and R) style
      - If R is always zero, the latch can permanently keep a value of S when Clk=1. ⇒ Clk=1일때, 항상 R=0 이면 S의 값을 영구적으로 저장 가능
      - Therefore, the system may keep the error values caused by troublousness of cotrolling two inputs (S and R). ⇒ 두개 input을 제어하는 것에 대한 번거로움.
    - It depends on for how long Clk=1.
      - When Clk=1, through how many latches will a signal travel?
        - Clk_A -- signal may travel through multiple latches.
        - Clk_B -- signal may travel through fewer latches.
      - Hard to pick Clk that is just the right length. ⇒ 어떤 상황에서 clk length가 적절한지 알수가 없음.

두개의 input의 시불연관 저장 가능성

Clk=1인 시간에 의존



time-line: 1분 16초

↳ 더 정확한 증폭 ⇒ 더복잡한 storage 설계 가능

More Elaborate Synchronization

⇒ 제어를 편하게 하기 위해서 latch의 권한을 줄이자.

- Let's reduce the power of the latch for simplifying control.

We allow level-sensitive SR latch to have too much power.

- ~~Timeless storing capability based on two inputs (S and R) style~~
  - If R is always zero, the latch can permanently keep a value of S when Clk=1.
  - Therefore, the system may keep the error values caused by troublousness of cotrolling two inputs (S and R).

input은 1개로 storage capability는 1-cycle로 줄여버리자

Only 1-cycle capability based on one-input style

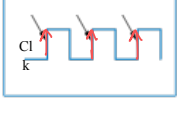
· It does not matter because the storing capability can be extended with feedback circuits.

↳ 계속 그 값을 저장하고 싶으면 feedback회로로 계속 넣어주면 됨. (개발자가 편성가능)

- ~~It depends on for how long Clk=1.~~
  - When Clk=1, through how many latches will a signal travel?
    - » Clk_A -- signal may travel through multiple latches.
    - » Clk_B -- signal may travel through fewer latches.
  - Hard to pick Clk that is just the right length.

CLK이 rising edge일 순간만 output 값이 바뀜

It only depends on the rising edge of a clock signal.



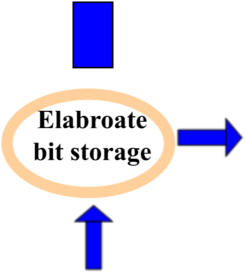
0, 1만 쓰는 이유 ⇑

More Elaborate Synchronization

More Elaborate Synchronization 이 가진 큰 특징

1) **Only 1-cycle capability based on one-input style**

> 더 정교한 Storage를 만들기 위해 component의 권한을 빼앗아옴
> ⇒ 개발자 권한이 높아짐.
> ⇒ 통제하기 좋음. (단순화)



It means that the bit storage only store a value from the 'current rising edge' to the brink of the 'next rising edge'.

↳ 현재 rising edge ~ 다음 rising edge까지 값 저장.

2) **It only depends on the rising edge of a clock signal.**

(∵ rising edge 값이 유효하며 1-cycle 이므로)

More Elaborate Synchronization

– Design strategy for the elaborate bit storage

- **Step#1**: First of all, let's make level-sensitve SR latch to have 'only 1-cycle storing capability with only one-input.

확장 ⇓

- **Step#2**: Then, we try to design the bit storage that only stores a value on the rising edge of a clock signal based on Step#1.

⇓

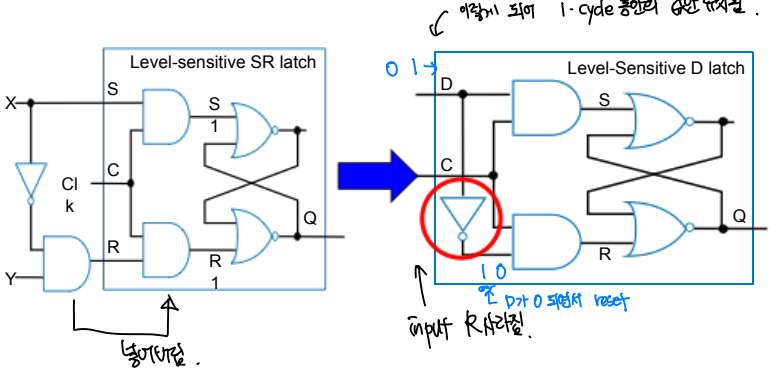
결과 : Flip-flop

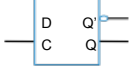
Design strategy for the elaborate bit storage : Step#1

Step#1: First of all, let's make level-sensitve SR latch to have 'only 1-cycle storing capability with only one-input.

↑ SR latch 에서 R 제거

- Level-sensitive D latch
  - If we consider only one input 'D'
    - Inserted inverter ensures R always opposite of S.

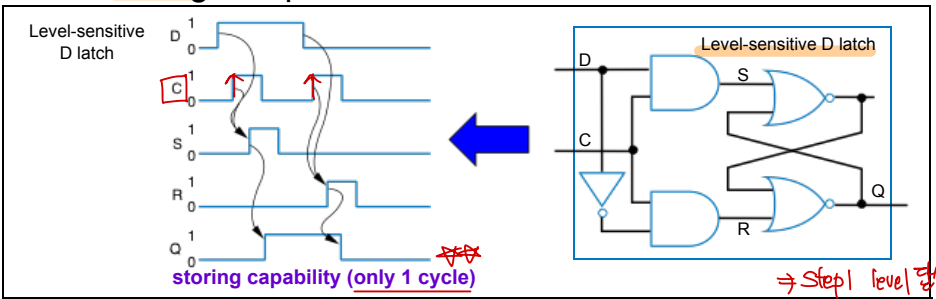




Level-Sensitive D latch symbol

Design strategy for the elaborate bit storage : Step#1

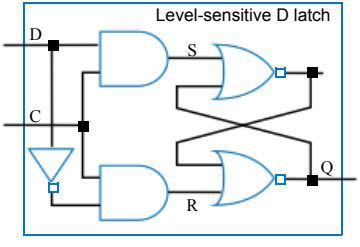
- Timing comparison between SR latch and D latch



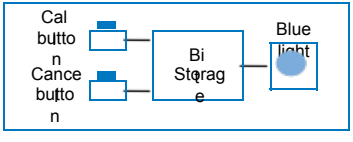
Design strategy for the elaborate bit storage : Step#1

- Previous example - Flight attendant call button
  - Is level-sensitive D-Latch enough to design the button system?

불충분.



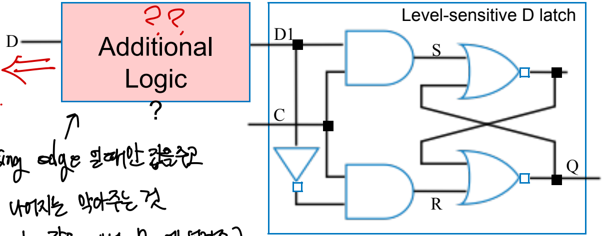
?



Step 1 ⇒ 1-cycle (storing capability)
1-input

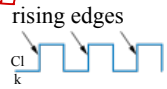
Design strategy for the elaborate bit storage : Step#2

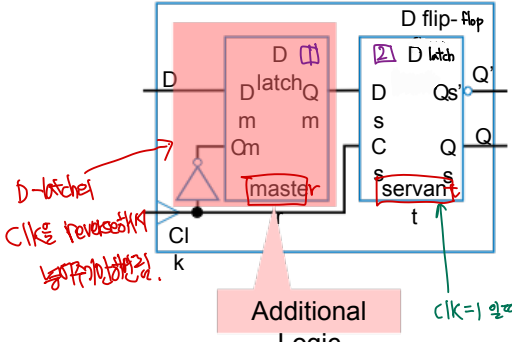
- Step#2: We try to design the bit storage that only stores a value on the rising edge of a clock signal based on Step#1
  - Level sensitive D latch
- The elaborate bit storage
  - 'only storing on the rising edge' means that any change of the input value right after the rising edge must not affect the output even though Clk =1.
  - This is impossible on the level sensitive D latch. However, it may be achieved if an additional logic stores the input value 'D' on the rising edge and feed it to input 'D1' of the latch regardless of the change of 'D' as below figure.

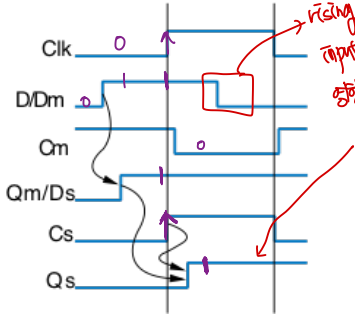


Design strategy for the elaborate bit storage : Step#2

- **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 Clk=0, then servant when Clk=1.
  - When Clk changes from 0 to 1, master disabled, servant loaded with value that was at D just before Clk changed -- i.e., value at D during rising edge of Clk.



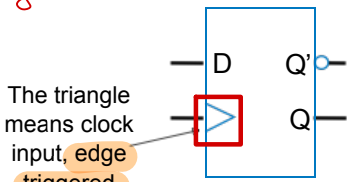




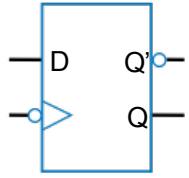
D Flip-Flop

대부분 rising을 씀 rising edge flip-flop

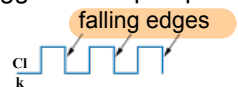
falling edge flip-flop



Symbol for rising-edge triggered D flip-flop



Symbol for falling-edge triggered D flip-flop



flip-flop은 latch 개념

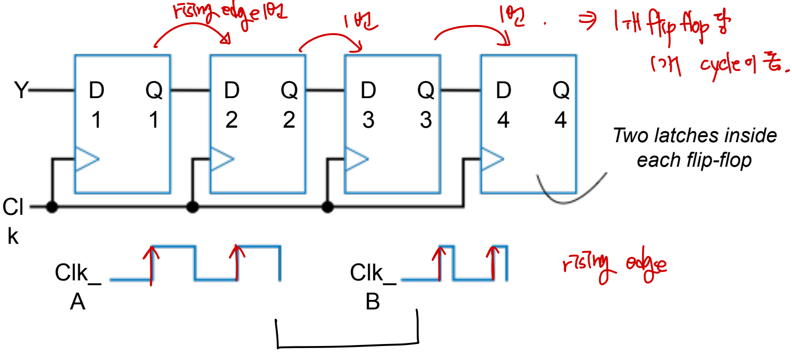
⇒ ▷ 가있다

↳ edge triggered

D Flip-Flop

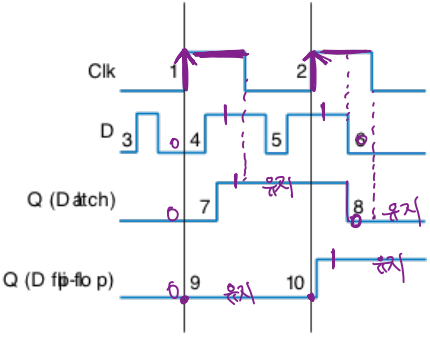
17 페이지 SR latch와 비교

- Solves problem of not knowing through how many latches a signal travels when Clk=1 as shown in page #17
  - 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 Clk=1
- Flip-flop is edge triggered: Stores D when Clk changes from 0 to 1
  - Saying "level-sensitive latch," or "edge-triggered flip-flop," is redundant : Just saying latch or flip-flop is ok!
  - Two types of flip-flops -- rising or falling edge triggered.
- Comparing behavior of latch and flip-flop:

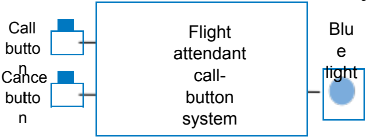


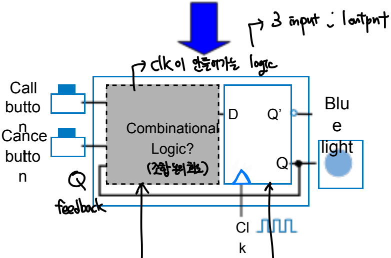
→ D flip flop으로 앞에 기능 구현해보기

Flight-Attendant Call Button Using D Flip-Flop

↳ rising edge일 때 D→Q 넘어감.

- Combinational Logic + D flip-flop
- Combinational Logic
  - Inputs are Call, Cancel, and present output value of D flip-flop, Q
  - Output is input value of D flip-flop, D.
  - Design process
    - 1. Capture the function: truth table shown below
    - 2. Convert to equation: ?
    - 3. Implement as a gate-based circuit : ?





(Truth table) → 8가지 조합.

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

Flight-Attendant Call Button Using D Flip-Flop

- Combinational Logic + D flip-flop
  - Combinational Logic
    - Design process

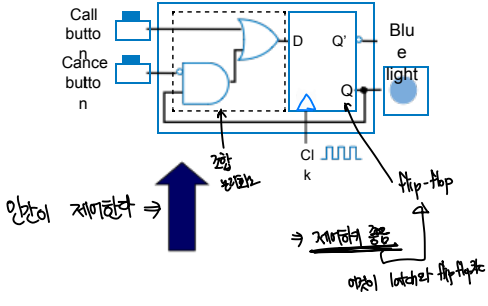
1. Capture the function

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

2. Convert to equation: D = Call + Q · Cancel'

Q \ Call Canc	0	0	1	1
0			1	1
1	1		1	1

3. Implement as a gate-based circuit



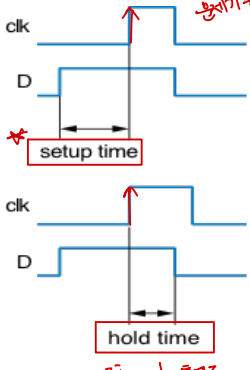
Non-Ideal Flip-Flop Behavior

Can't change flip-flop input too close to clock edge

- **Setup time**: time that D must be stable *before* edge
  - Else, output value temporarily shows the previous value.
- **Hold time**: time that D must be held stable *after* edge
  - Else, output value changes.

Violating setup/hold time can lead to unstable situation

- Therefore, we must make sure setup/hold time.
  - A flip flop typically comes with a datasheet describing setup times and hold times.



(→ 추가설명임)

이러한 오버 현상이 일어나는 원인이 다름

(2시간 34분)

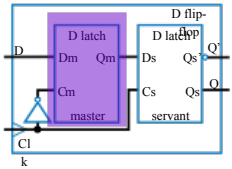
Non-Ideal Flip-Flop Behavior

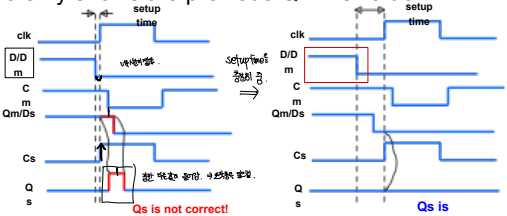
⇒ master D latch

Setup Time Violation – caused by internal delay of master D latch

Setup time is very short . So correct Qm/Ds Value is not yet prepared on rising edge of Clk.

Therefore, output Qs temporarily shows the previous Qm/Ds value.

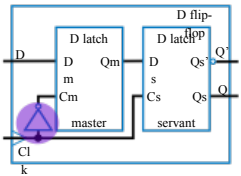


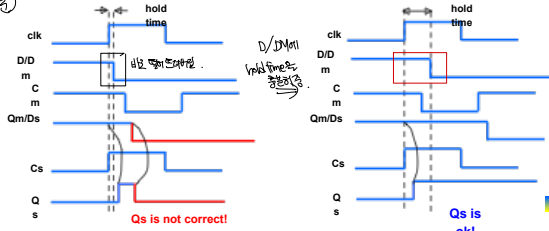


- **Hold Time Violation – caused by inverter delay coupled with master D latch**
  - Hold time is very short. So D/Dm value changes before Cm becomes 0.
    - Therefore, Qm/Ds Value also changes and affects output Qs.

⇒ inverter clk

(2시간 4분)

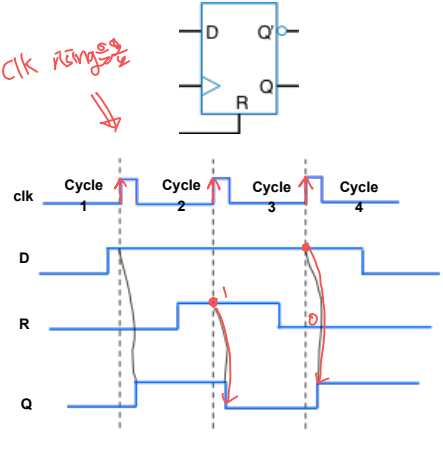


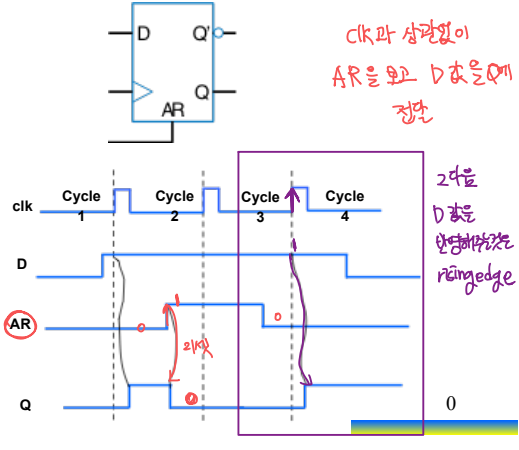


hold time이 짧은 inverter delay로 인해 발생함.

Flip-Flop Reset Inputs

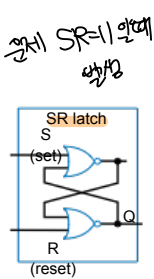
- Some flip-flops have additional inputs
  - ① Synchronous reset: clears Q to 0 on rising edge of clock.
  - ③ Asynchronous reset: clear Q to 0 immediately. (not dependent on clock edge)



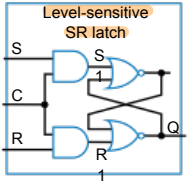


정리

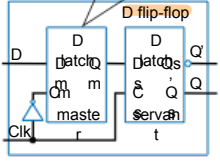
Bit Storage Summary



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: level-sensitive sync' method is not feasible to more complex bit storage systems.



Feature: Only loads D value present at rising clock edge, so values can't propagate to other flip-flops during same clock cycle – more elaborate sync' method. Tradeoff: uses more gates internally than D latch 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.

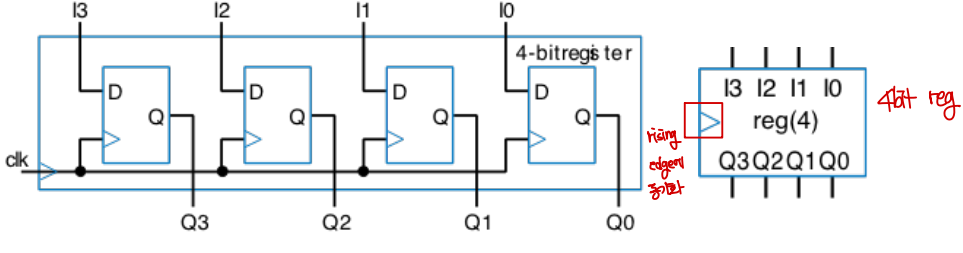
Register for Multi-bit Storage

Basic Register → D flip-flop의 여러개 결합 ver.

4bit register ⇒ 4 D flip-flop

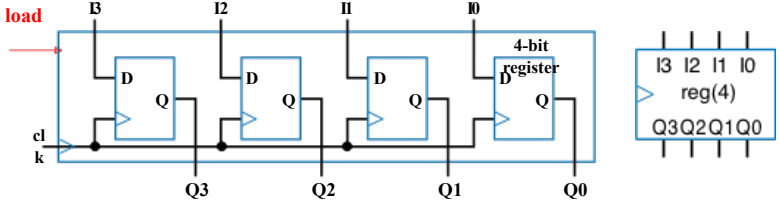
- Typically, we store multi-bit items.
  - e.g., storing a 4-bit binary number
- ***Register***: multiple flip-flops sharing clock signal.
  - From this point, we'll use registers for bit storage.
    - No need to think of latches or flip-flops.
    - But now you know what's inside a register.

여러개의 flip-flop이 clock signal을 공유하는 storage



Register with Parallel Load

- Basic register: Loaded every cycle
  - For other uses, may want to load only on certain cycles



Basic register loads on every clock cycle

How extend to only load on certain cycles?

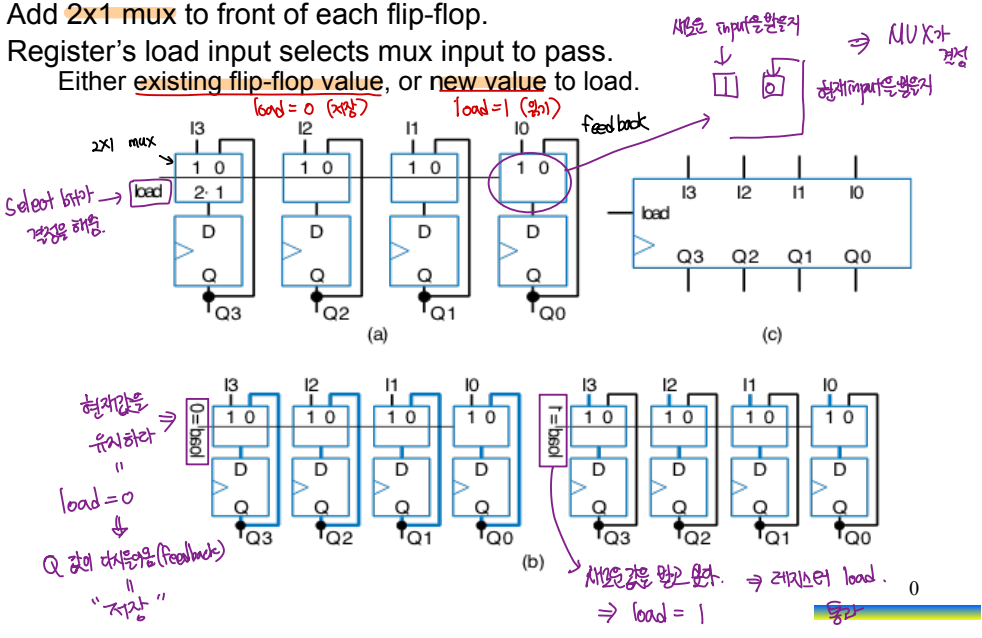
새로운 값이 들어오는 것.

↳ 특정한 시간 (cycle) 만큼 값을 저장하고 싶으면 어떻게 해야 하는가?

Register with Parallel Load

MUX
⇒ 여러개 값이 들어올때 하나의 값을 선택하는것

- Add 2x1 mux to front of each flip-flop.
- Register's load input selects mux input to pass.
  - Either existing flip-flop value, or new value to load.

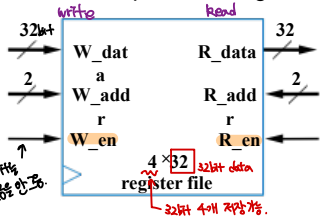


Register File

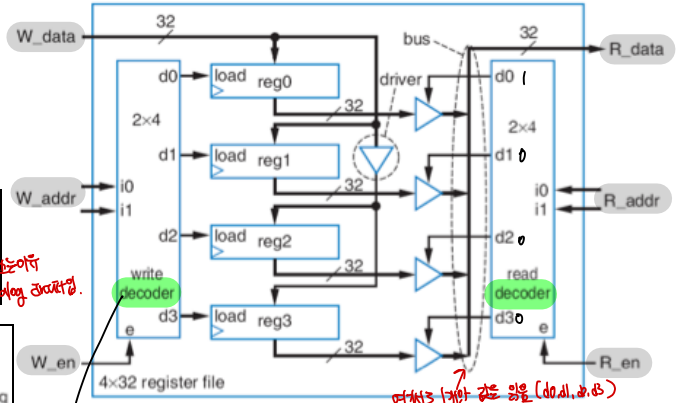
MxN register file component provides efficient access to M cases of N-bit-wide registers.

Register file has one data input and one data output, and allows us to specify which internal register to write and which to read.

For example, 4x32 register file



W_en
R_en

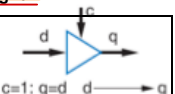


Buffer

: output equals its input but the output is as stronger ~~(higher current) signal.~~

Tri-state buffer

: same to general buffer but supports high impedence ('Z') like no connection.



Register File Timing Diagram

- Synchronous write operation
- Asynchronous read operation
- Simultaneous read & write operation possible
  - may be same register.