Digital IC Design

Lecture 5:
Sequential Circuits

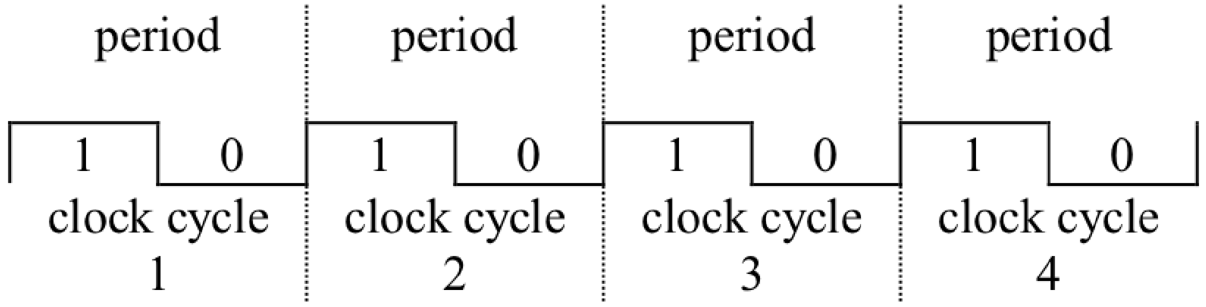
黃柏蒼 Po-Tsang (Bug) Huang
bughuang@nycu.edu.tw

International College of Semiconductor Technology
National Chiao Tung Yang Ming University

國立陽明交通大學
NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Sequencing

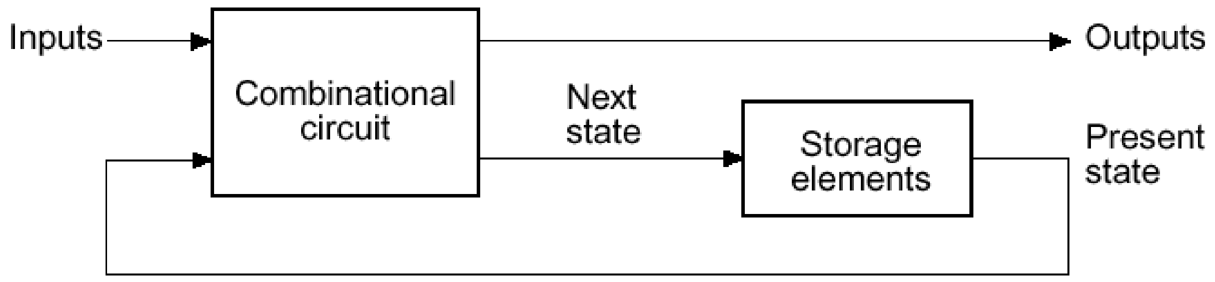
- ***Combinational logic***
  - output depends on current (present) inputs
- ***Sequential logic***
  - *output depends on* current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Circuit "remembers" previous history using
    - **Flip-Flop (F/F) or latch related with a reference clock signal**



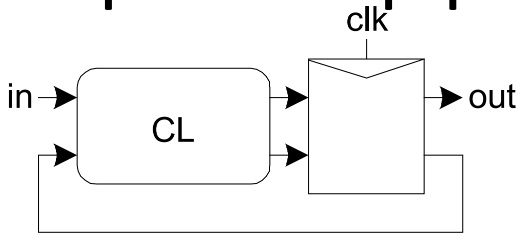
Synchronous circuits

Sequential Logic

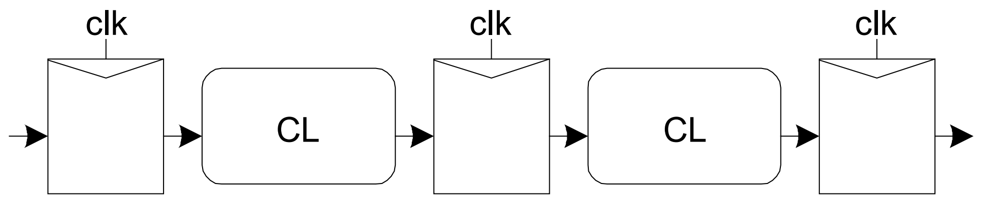
- Sequential finite state machine(FSM)
  - Output depends not only on current input but also on past input values, e.g., design a counter
  - Need some type of memory to remember the past input values



- Sequential pipeline



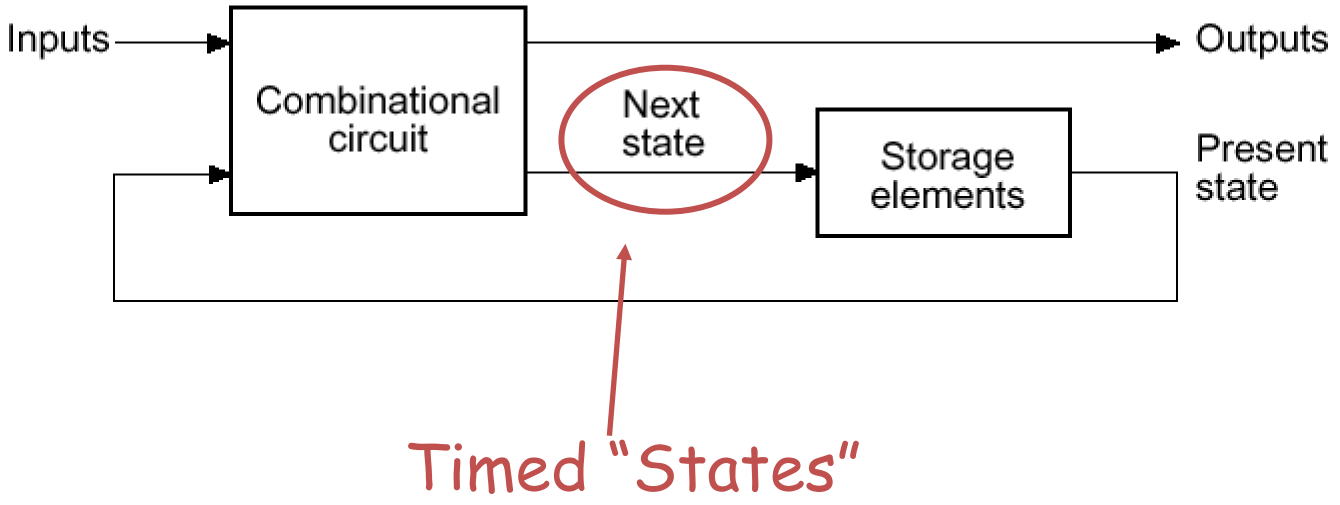
Finite State Machine



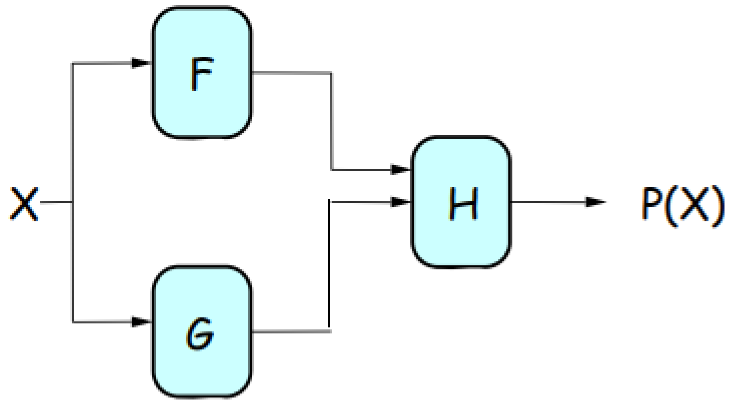
Pipeline

Sequential Logic: Concept

- Sequential Logic circuits remember past inputs and past circuit state.
- Outputs from the system are "feedback" as new inputs
  - With gate delay and wire delay
- The storage elements are circuits that are capable of storing binary information: memory.



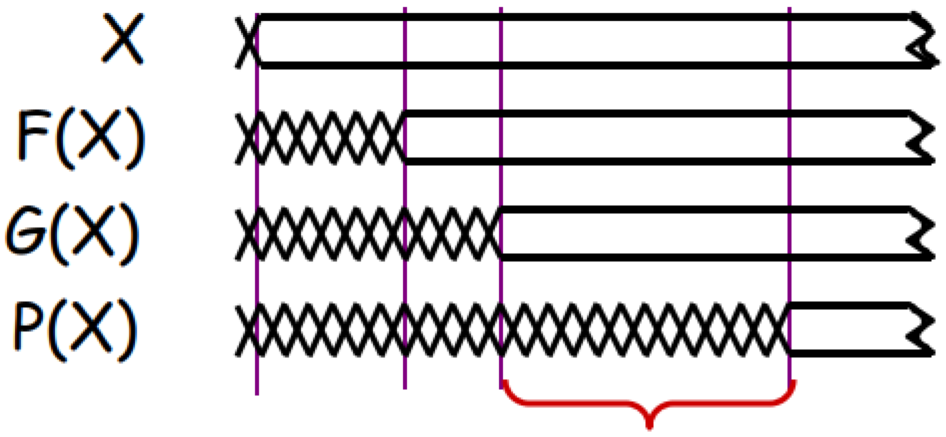
Performance of Combinational Circuits



For combinational logic:

$L = t_{PD}$,
$T = 1/t_{PD}$.

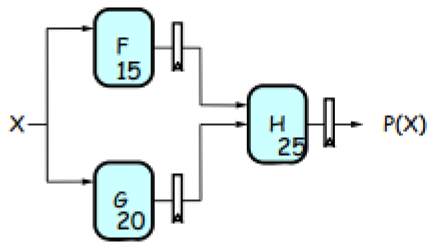
We can't get the answer faster, but are we making effective use of our hardware at all times?



F & G are "idle", just holding their outputs stable while H performs its computation

Pipeline diagrams

- The results associated with a particular set of input data moves diagonally through the diagram, progressing through one pipeline stage each clock cycle



Clock cycle →

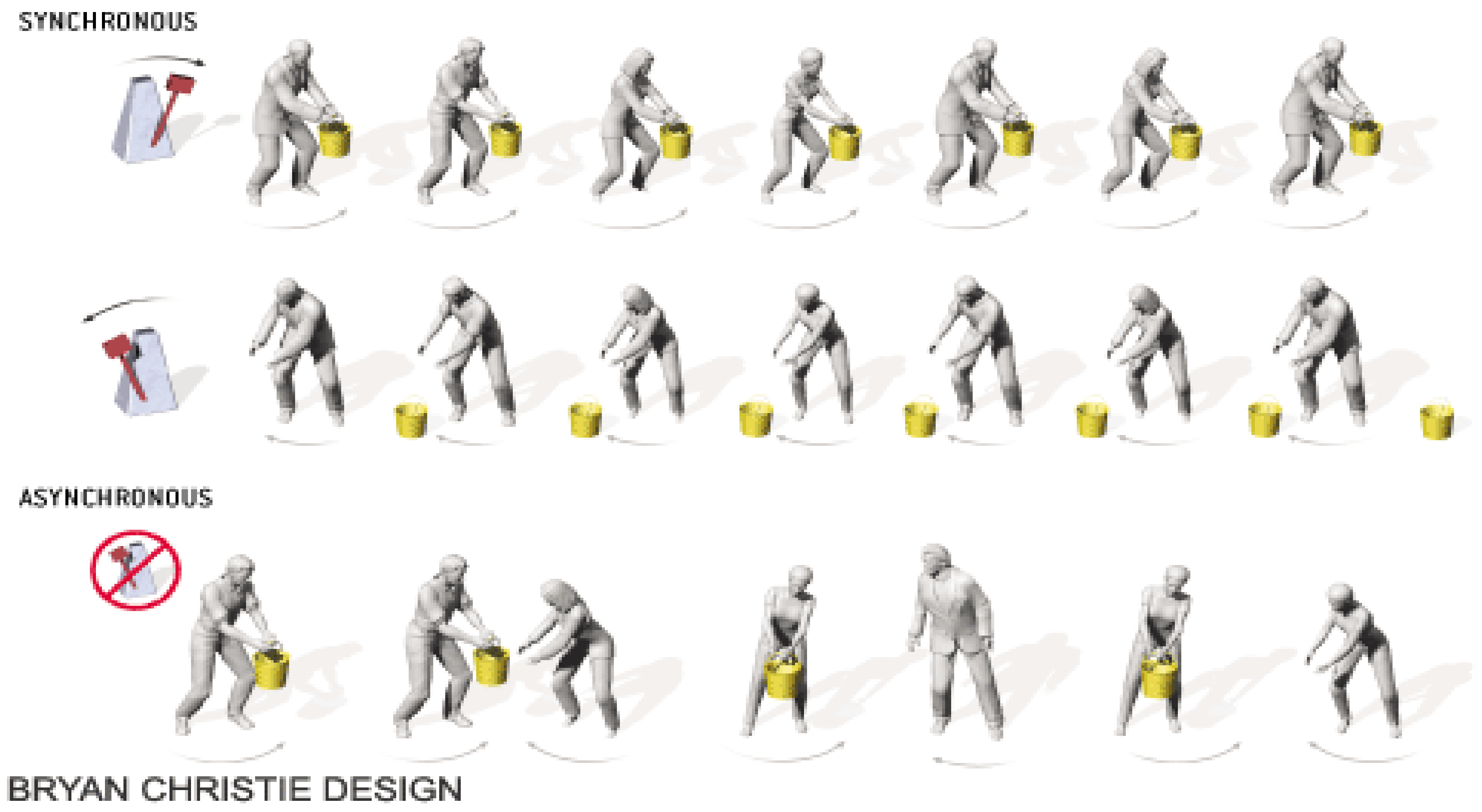
Pipeline stages ↓

	i	i+1	i+2	i+3	
Input	X_i	X_{i+1}	X_{i+2}	X_{i+3}	…
F Reg		$F(X_i)$	$F(X_{i+1})$	$F(X_{i+2})$	…
G Reg		$G(X_i)$	$G(X_{i+1})$	$G(X_{i+2})$	
H Reg			$H(X_i)$	$H(X_{i+1})$	$H(X_{i+2})$

Pipeline Conventions

- DEFINITION:
  - a K-Stage Pipeline (“K-pipeline”) is an acyclic circuit having exactly K registers on every path from an input to an output.
  - a COMBINATIONAL CIRCUIT is thus an 0-stage pipeline.

Synchronous & Asynchronous



Synchronous vs. Asynchronous

- Synchronous Circuit

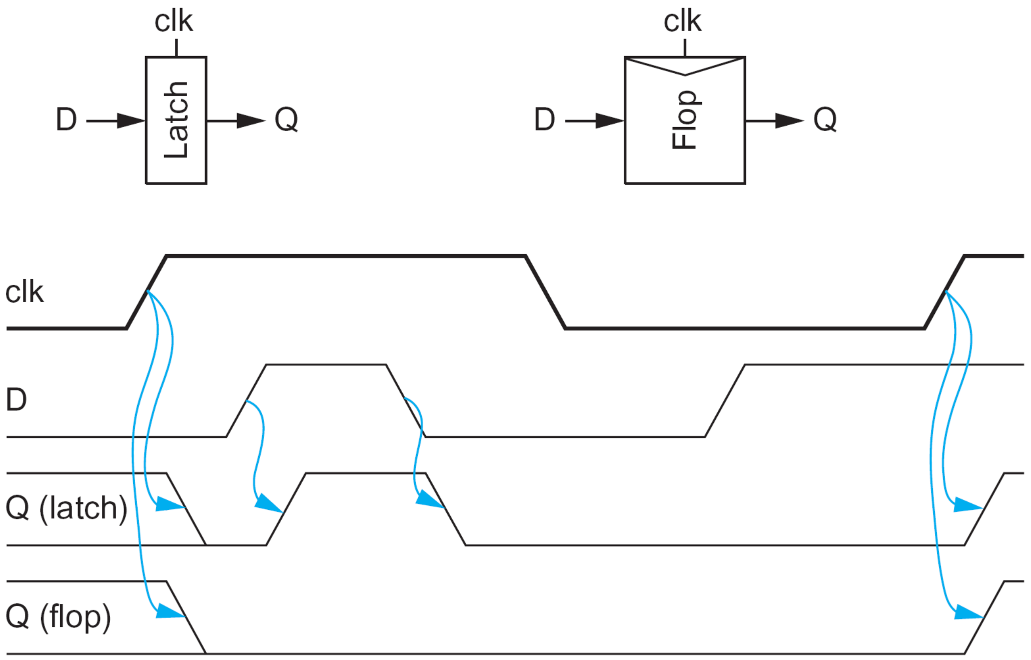
1. Use an external global **clock** to separate consecutive system states from each other
2. *Time Domain* : Assert signals at a specific time, and for a specific duration

- Asynchronous Circuit

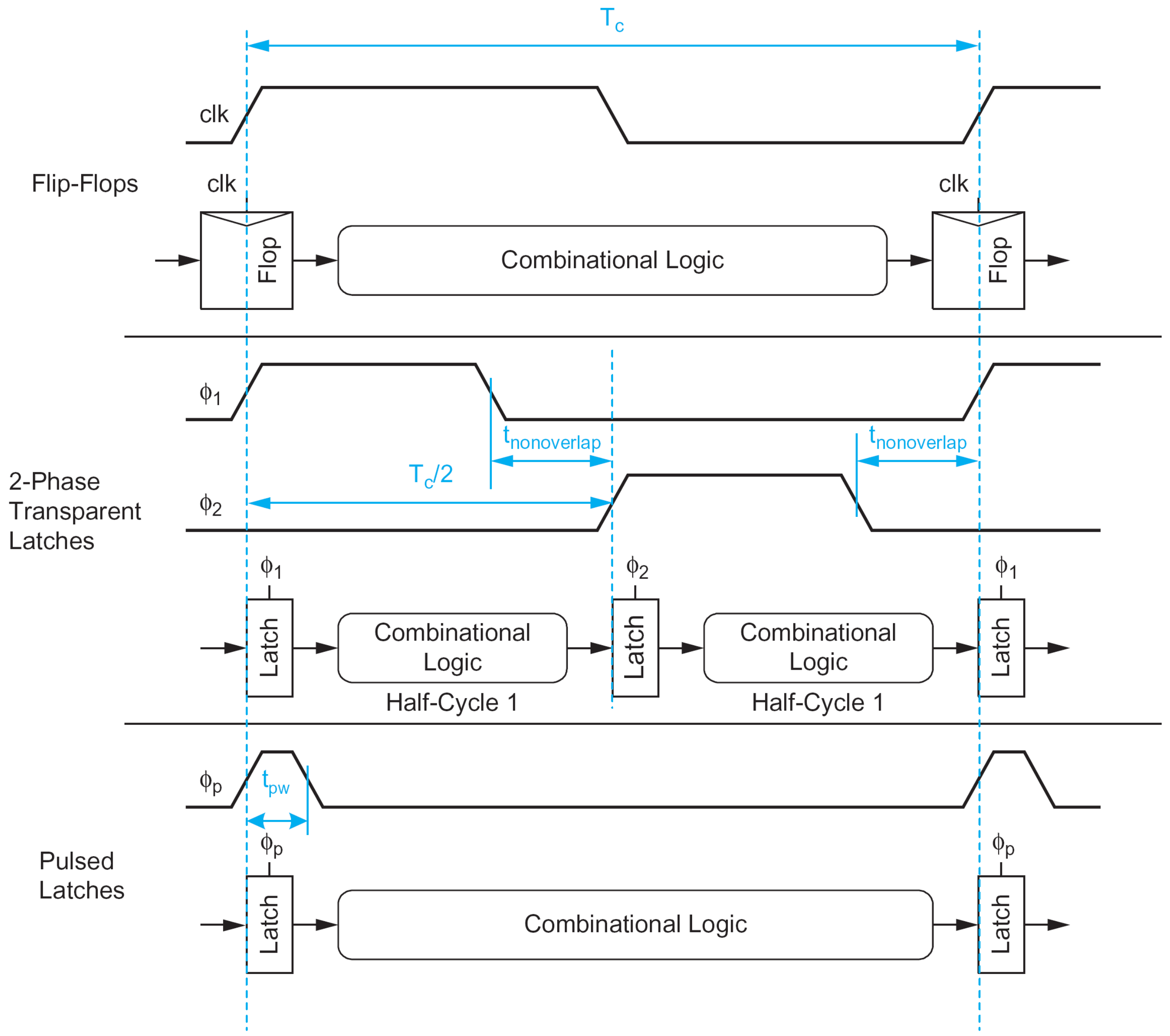
1. Define system states in terms of internal and external **events** (signal transitions)
2. Sequence Domain: Assert a signal after some event and holds the value until some other event
3. Use handshakes between system components to ensure correct sequencing of events (instead of a global clock signal)

Sequencing Elements

- **Latch**: Level sensitive (Transparent)
  - a.k.a. transparent latch, D latch
- **Flip-flop**: edge-triggered
  - master-slave flip-flop, D flip-flop, pulsed latch

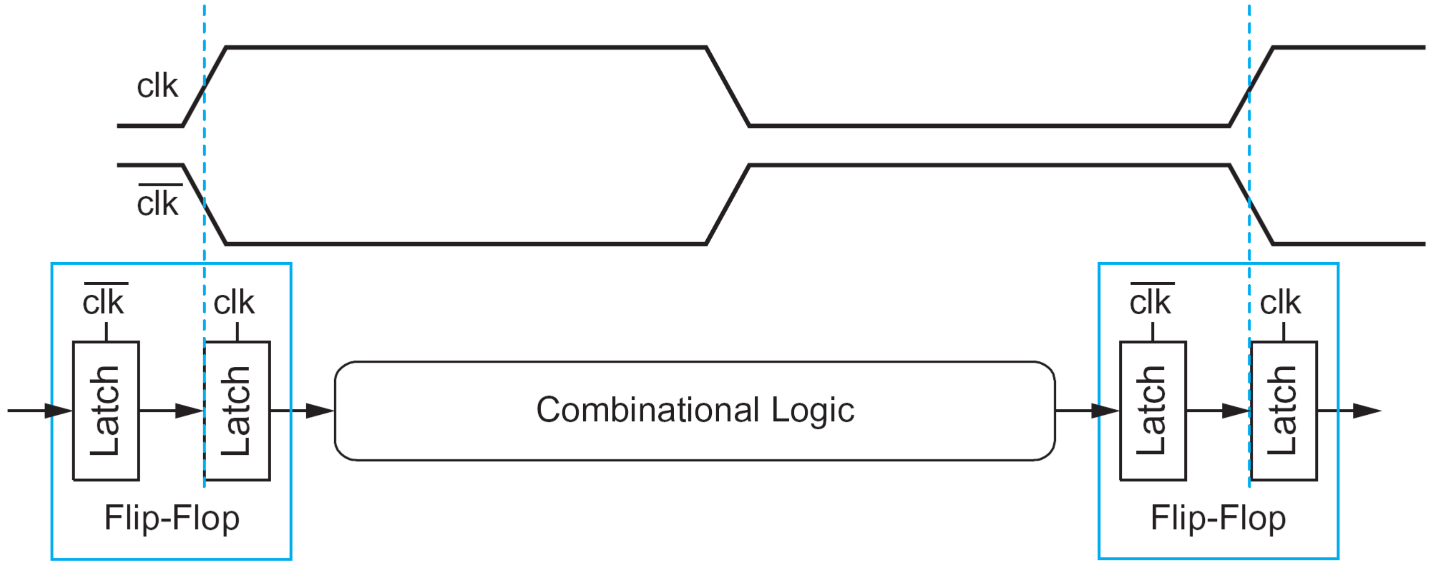


Timing of Sequencing Elements



Relation between Latch & F/F

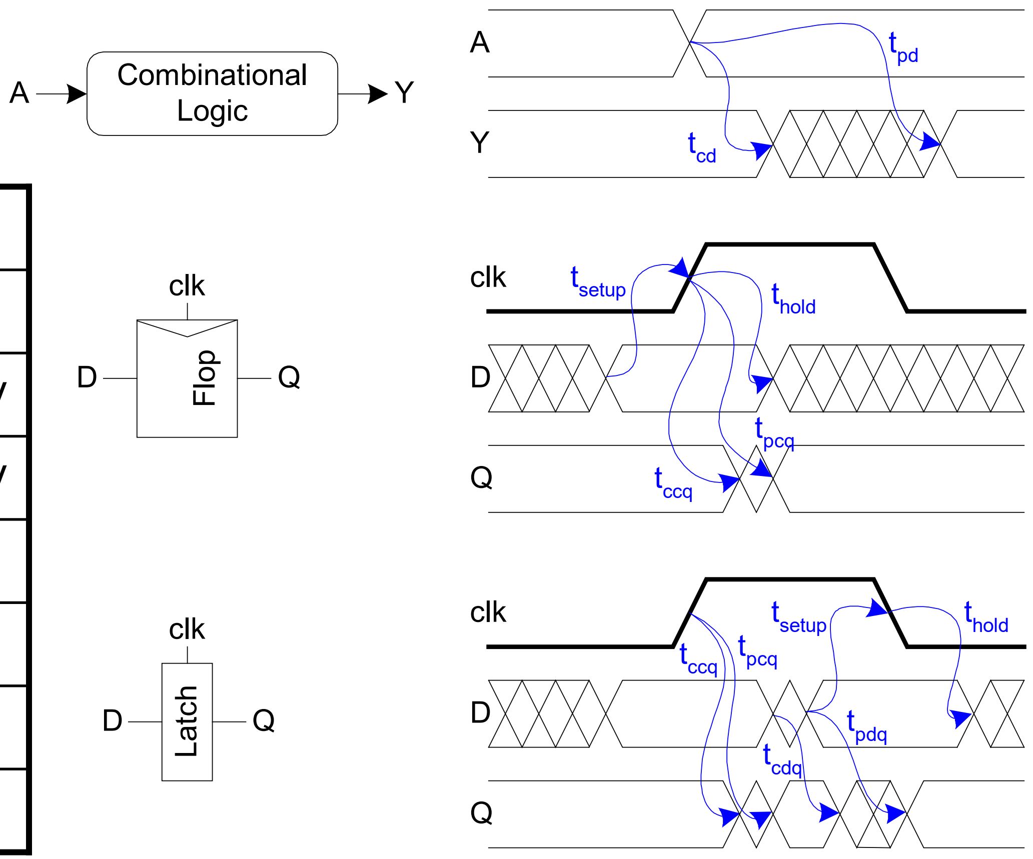
- 2 level sensitive latches = 1 edge-triggered FF



- Pulsed latch is similar to a F/F
  - Duty cycle is approached to 0% or 100%

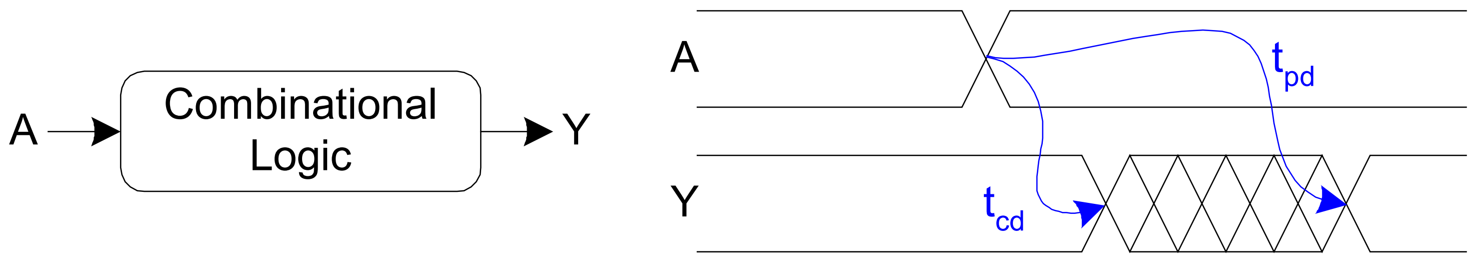
Definition of Timing issues in Sequencing

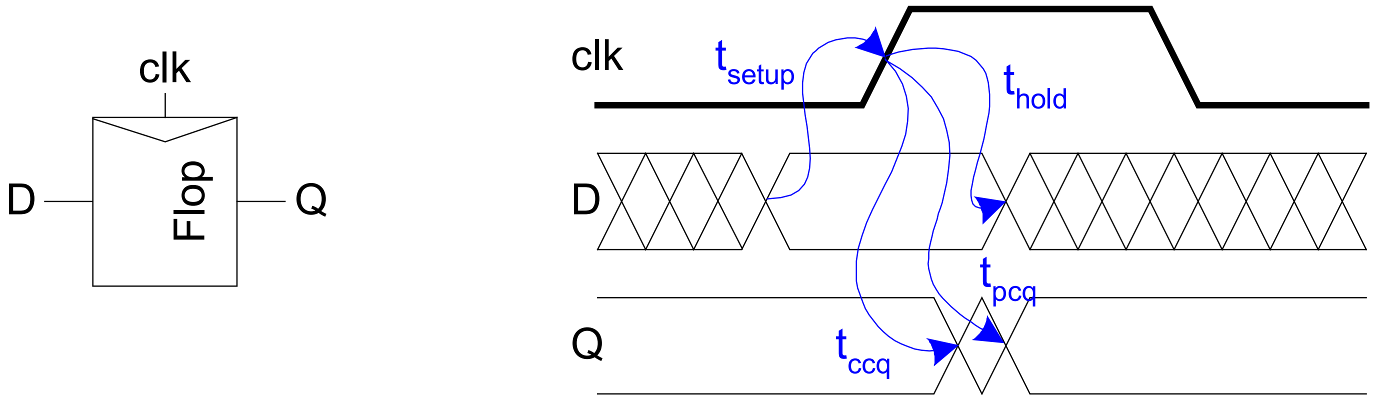
Contamination and Propagation Delays

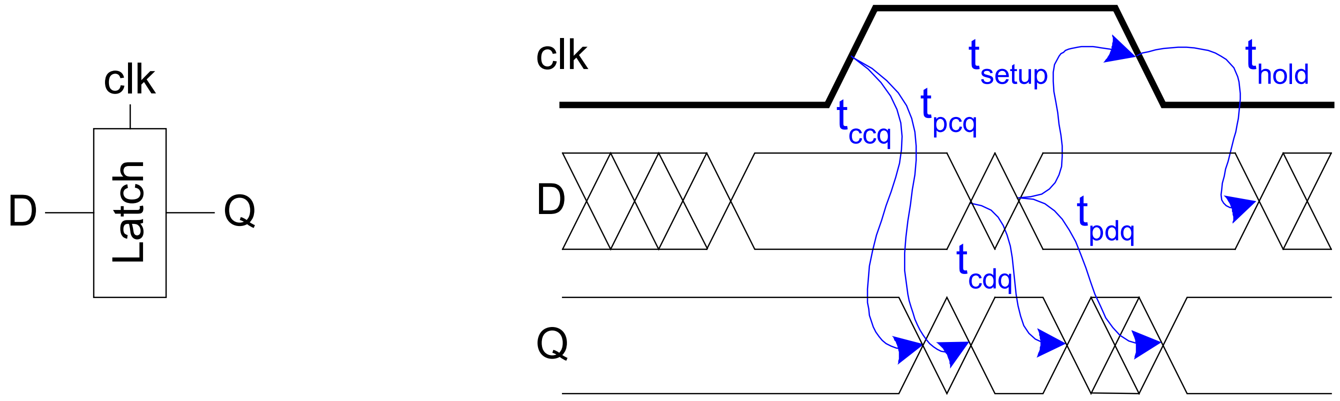


t_{pd}	Logic Prop. Delay
t_{cd}	Logic Cont. Delay
t_{pcq}	Latch/Flop Clk->Q Prop. Delay
t_{ccq}	Latch/Flop Clk->Q Cont. Delay
t_{pdq}	Latch D->Q Prop. Delay
t_{cdq}	Latch D->Q Cont. Delay
t_{setup}	Latch/Flop Setup Time
t_{hold}	Latch/Flop Hold Time

Definition of Timing issues in Sequencing

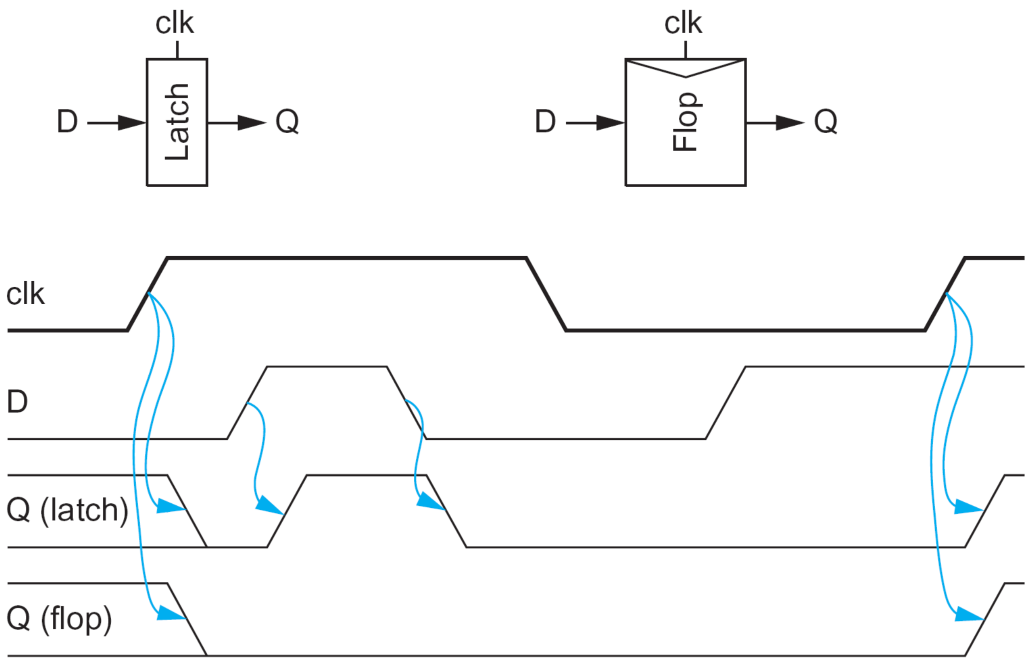






Sequencing Elements

- **Latch**: Level sensitive (Transparent)
  - a.k.a. transparent latch, D latch
- **Flip-flop**: edge-triggered
  - master-slave flip-flop, D flip-flop, pulsed latch



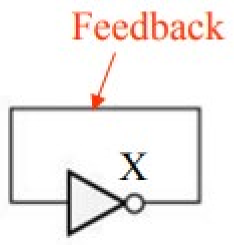
Static vs Dynamic Storage

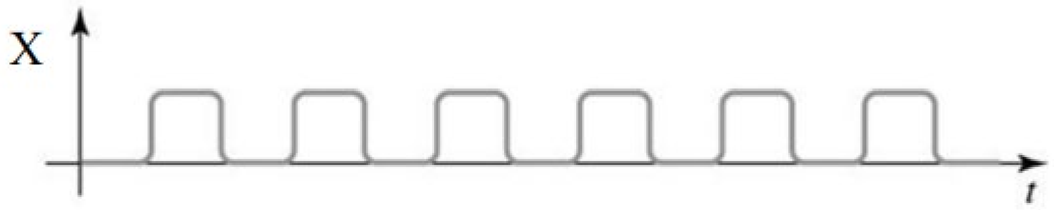
- Static storage
  - preserve state as long as the power is on
  - have positive feedback (regeneration) with an internal connection between the output and the input
  - useful when updates are infrequent (clock gating)
- Dynamic storage
  - store state on parasitic capacitors
  - only hold state for short periods of time (milliseconds)
  - require periodic refresh
  - usually simpler, so higher speed and lower power

Feedback of Odd and Even Inverters

- Timing in feedback network

One inversion (oscillatory)

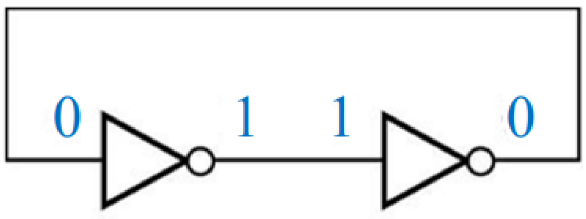




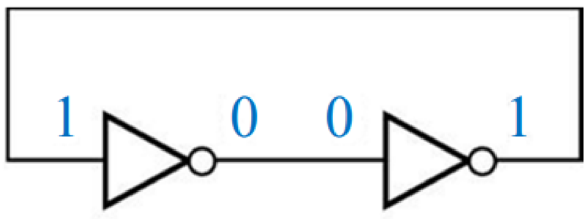
(a) Inverter with feedback

(b) Oscillation at inverter output

Two inversions (stable)

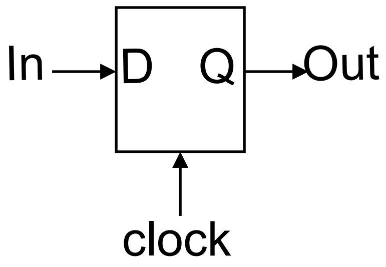


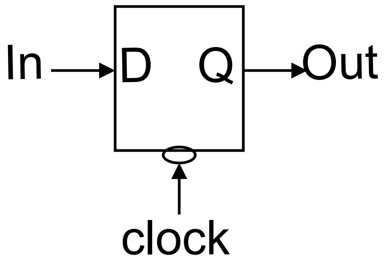
(a)

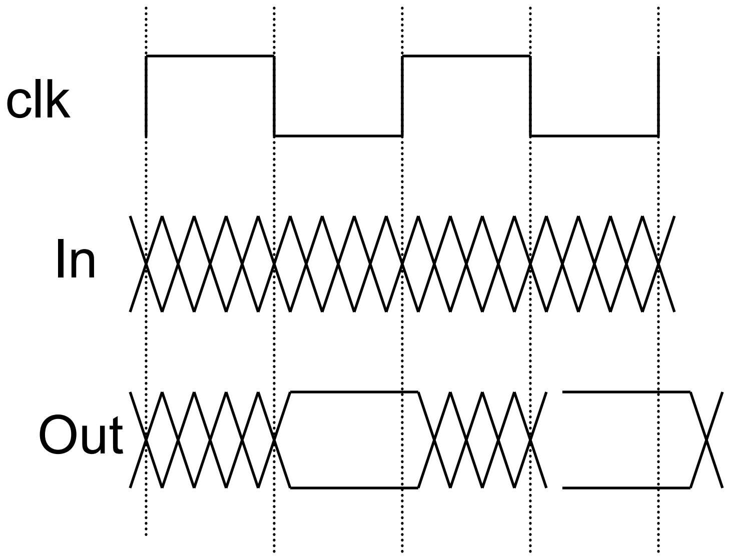


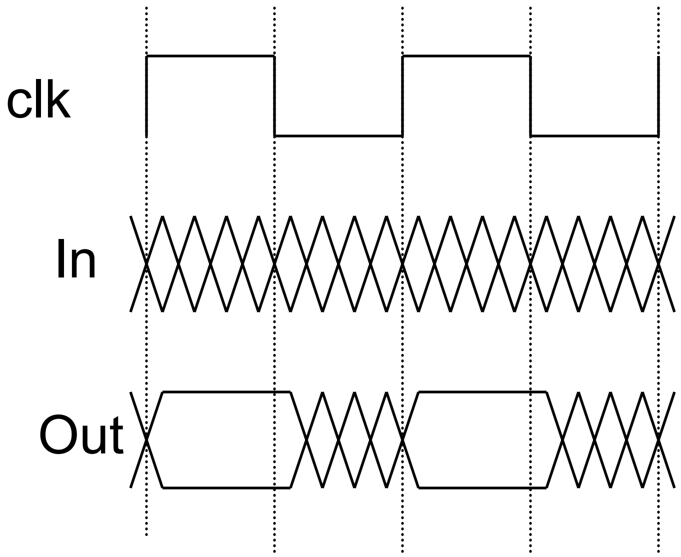
(b)

Positive and Negative Latches



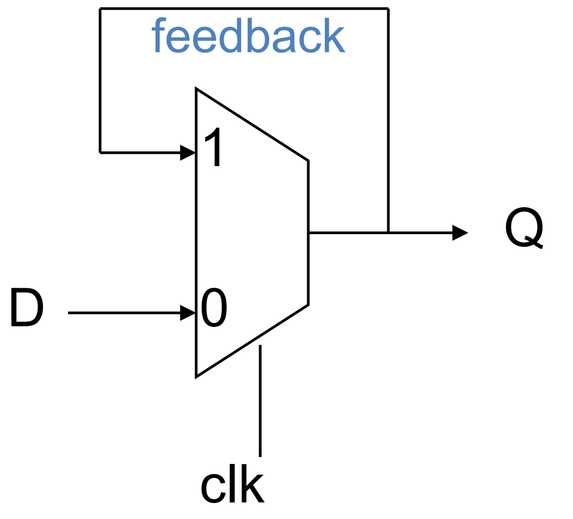






MUX Based Latches

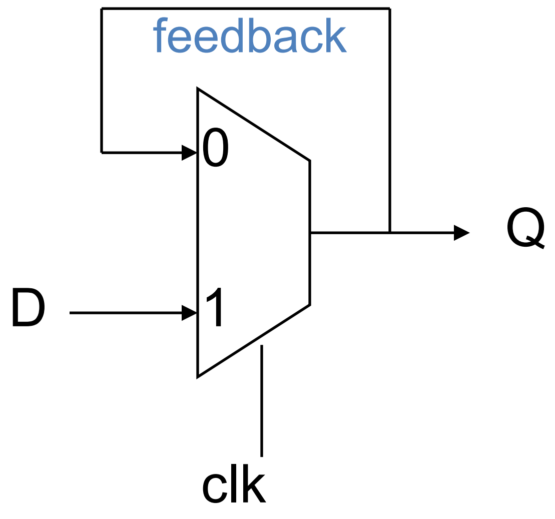
- Change the stored value by cutting the feedback loop



Negative Latch

Q = clk & Q | !clk & D

transparent when the clock is low



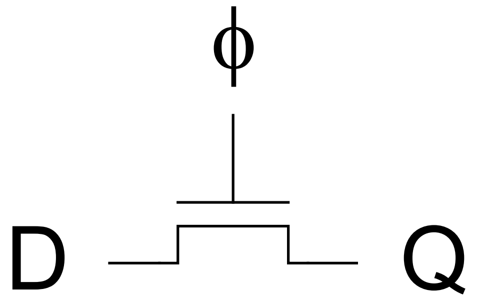
Positive Latch

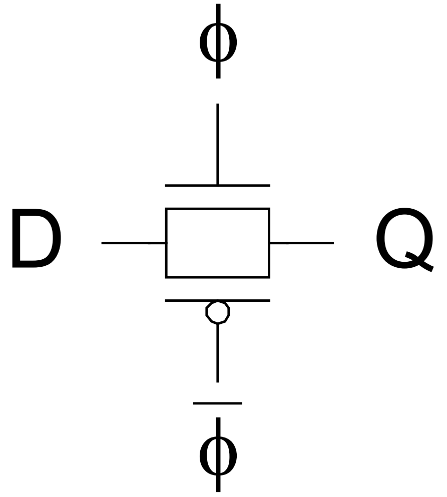
Q = !clk & Q | clk & D

transparent when the clock is high

Latch Design - Pass Transistor

- Pass Transistor Latch
- Pros
  - Tiny
  - Low clock load
- Cons
  - V_t drop
  - nonrestoring
  - backdriving
  - output noise sensitivity
  - dynamic
  - diffusion input

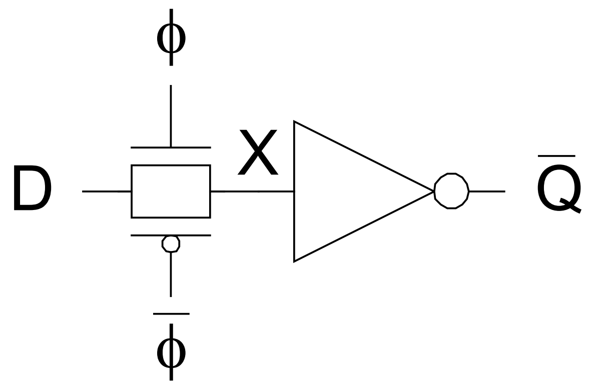


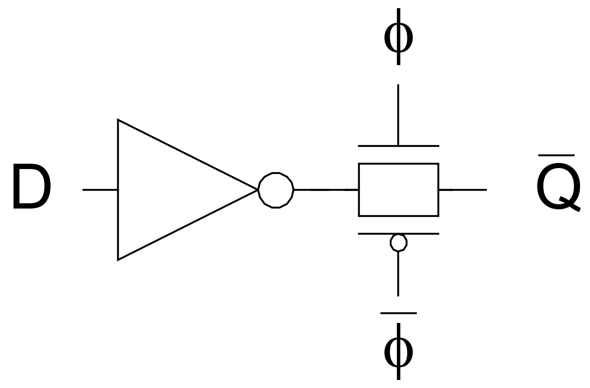


Requires inverted clock

Latch Design - Inverting buffer

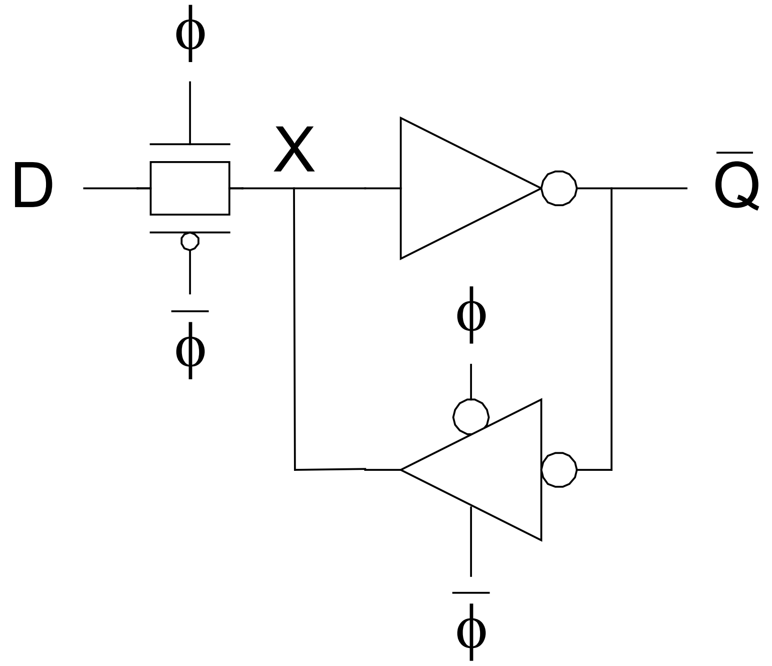
- Inverting buffer
  - + Restoring
  - + No backdriving
  - + Fixes either
    - Output noise sensitivity
    - Or diffusion input
  - ◆ Inverted output





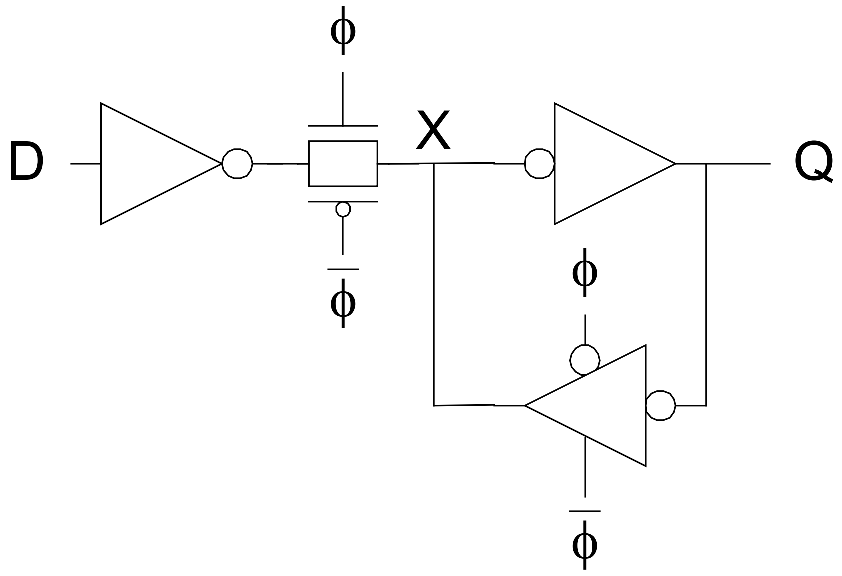
Latch Design - Tristate feedback

- Tristate feedback + Static
  - Backdriving risk
- Static latches are now essential because of leakage



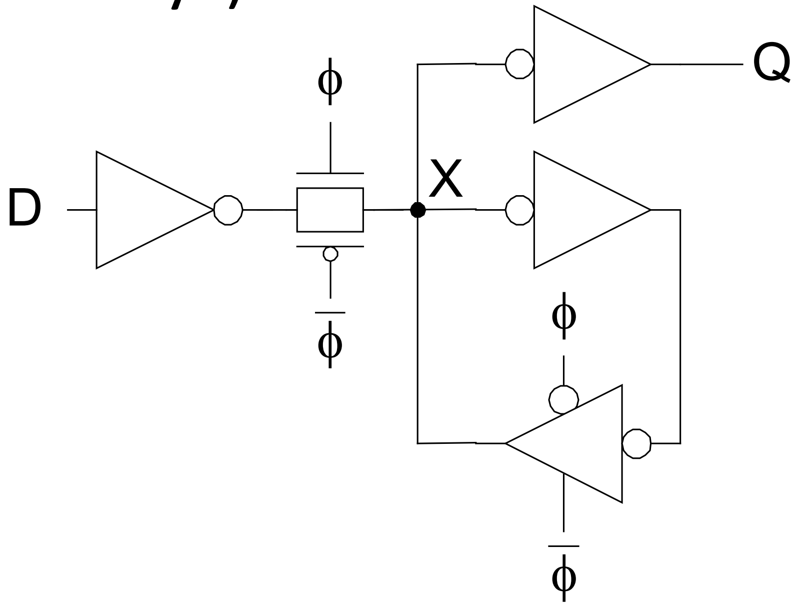
Latch Design - Buffered input

- Buffered input
  - + Fixes diffusion input
  - + Noninverting



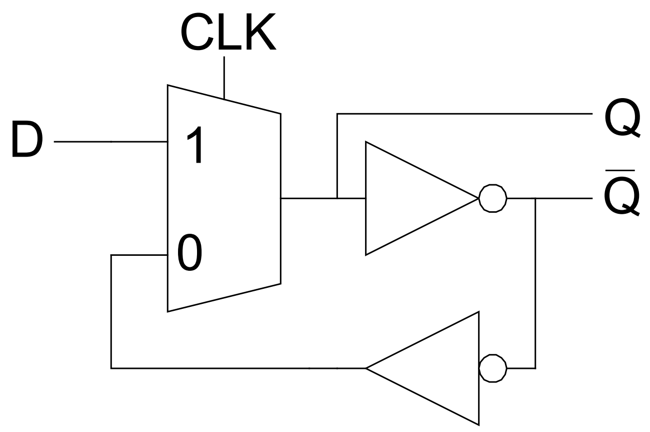
Latch Design - Buffered Output

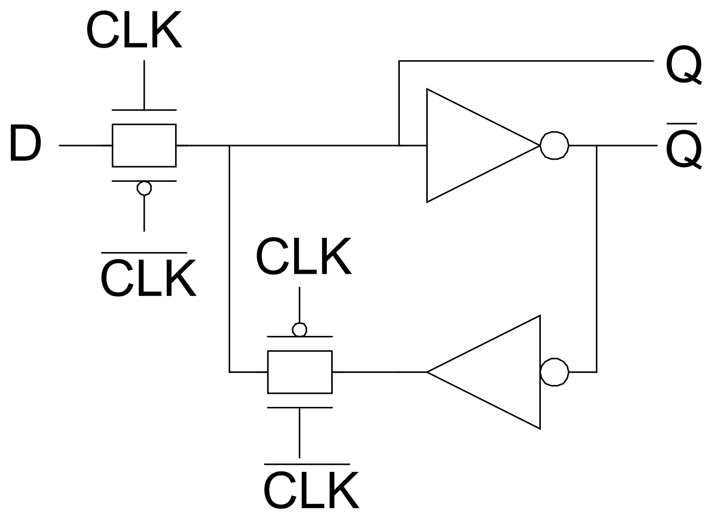
- Buffered output
  - + No backdriving
- Widely used in standard cells
  - + Very robust (most important)
  - - Rather large
  - - Rather slow (1.5 – 2 FO4 delays)
  - - High clock loading



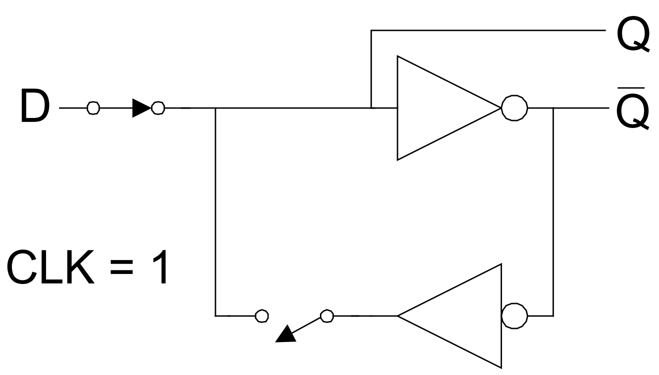
D Latch Design

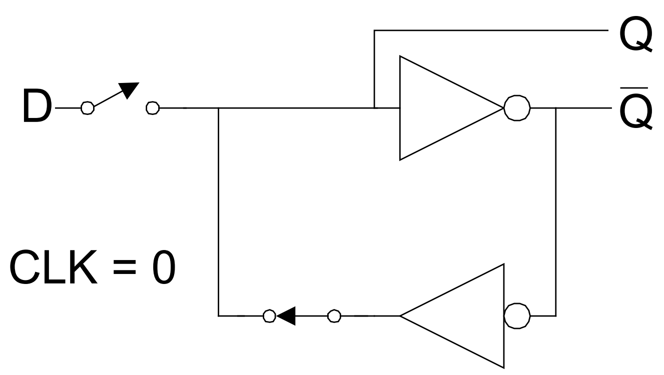
- Multiplexer chooses D or old Q

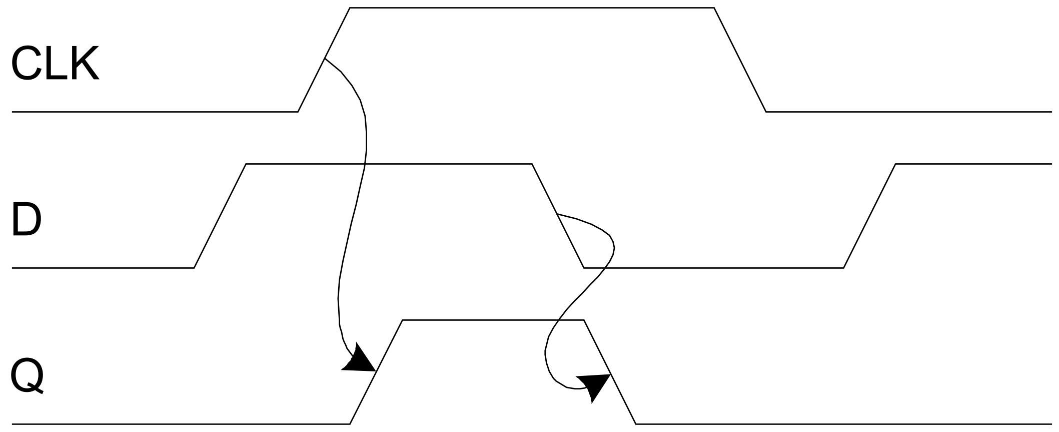




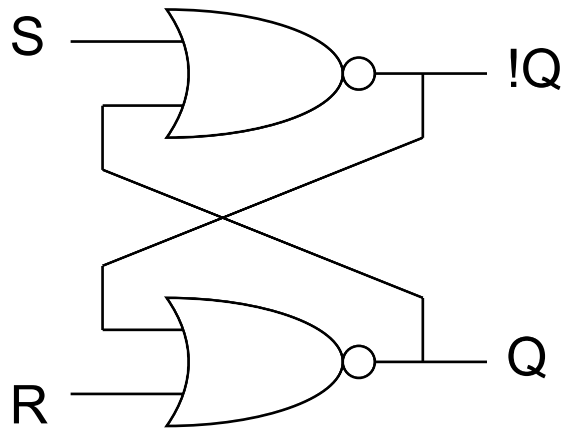
D Latch Operation





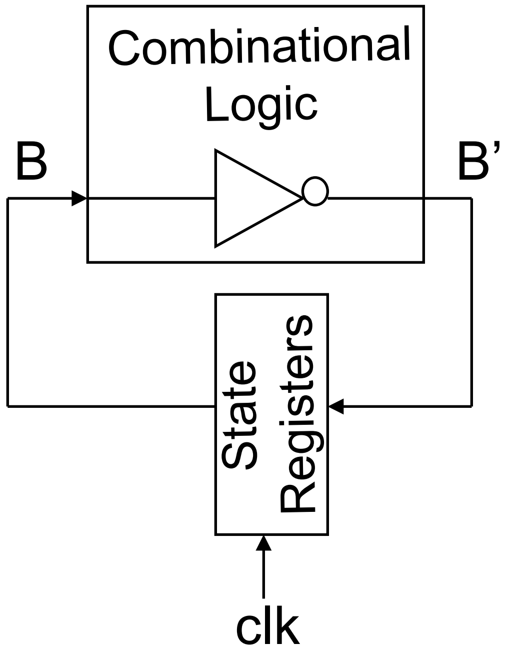


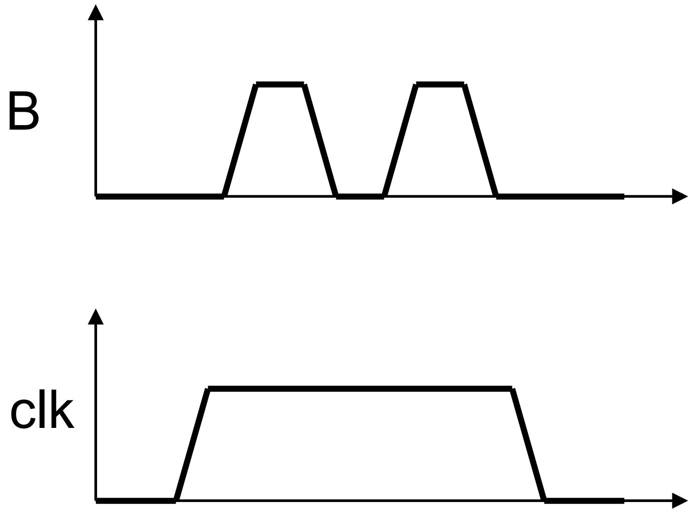
SR Latch



S	R	Q	!Q	
0	0	Q	!Q	memory
1	0	1	0	set
0	1	0	1	reset
1	1	0	0	disallowed

Latch Race Problem

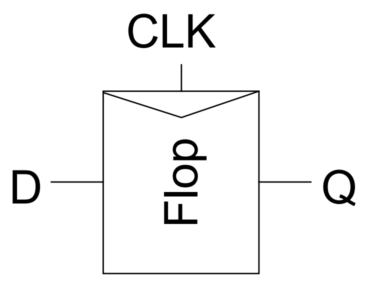


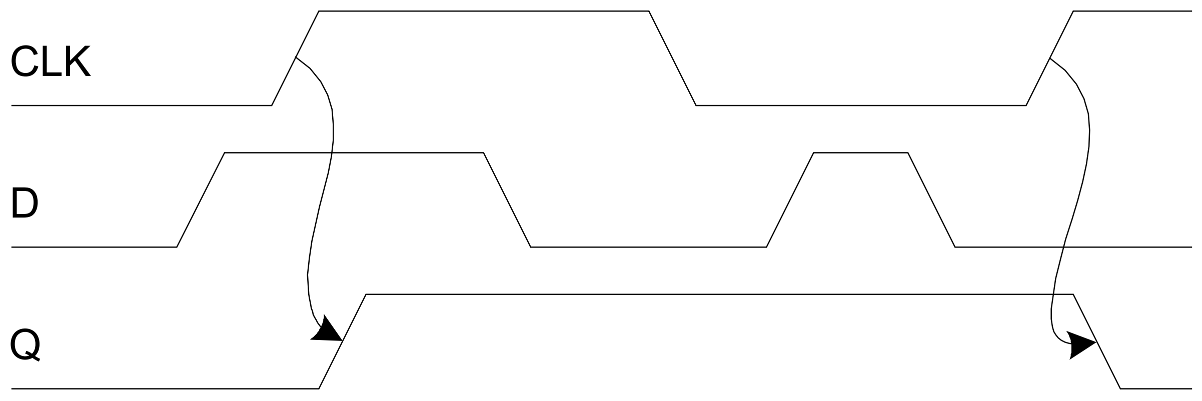


Which value of B is stored ?

D Flip-Flop

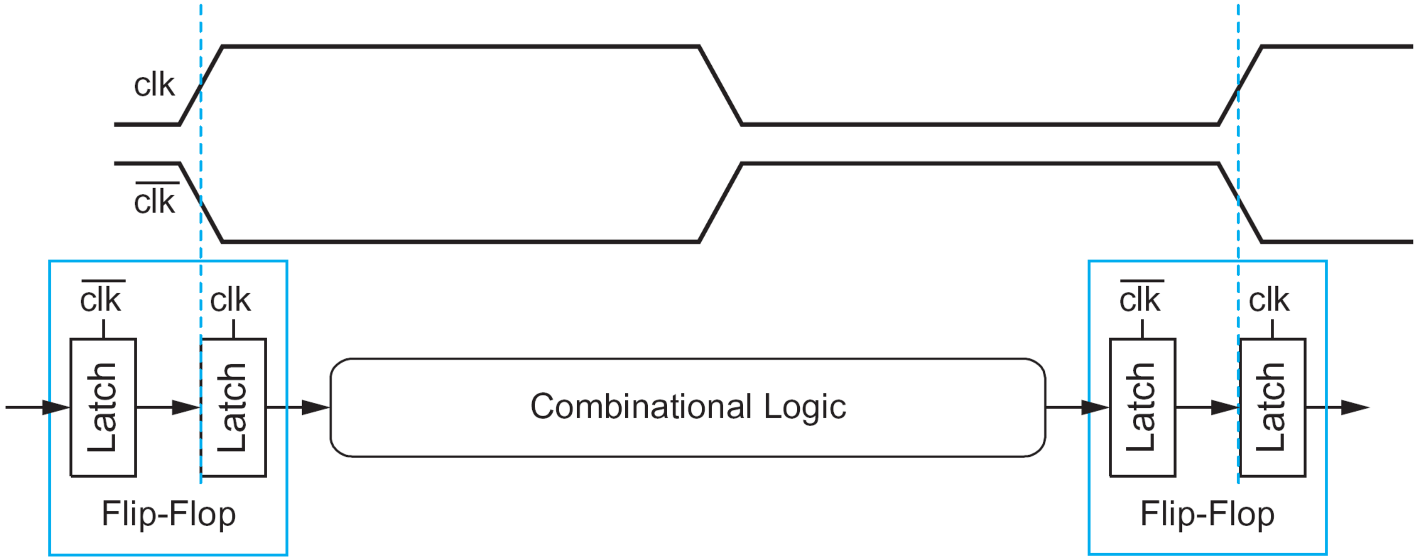
- When CLK rises, D is copied to Q
- At all other times, Q holds its value
- a.k.a. *positive edge-triggered flip-flop, master-slave flip-flop*

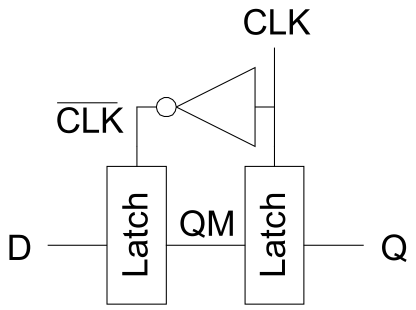


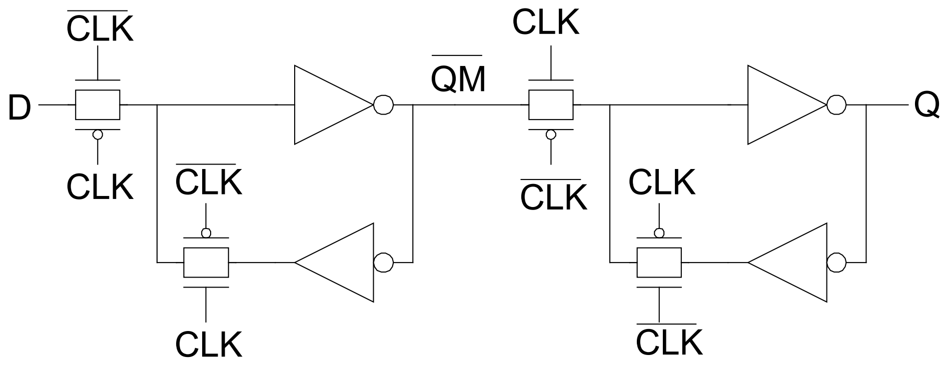


D Flip-Flop Design

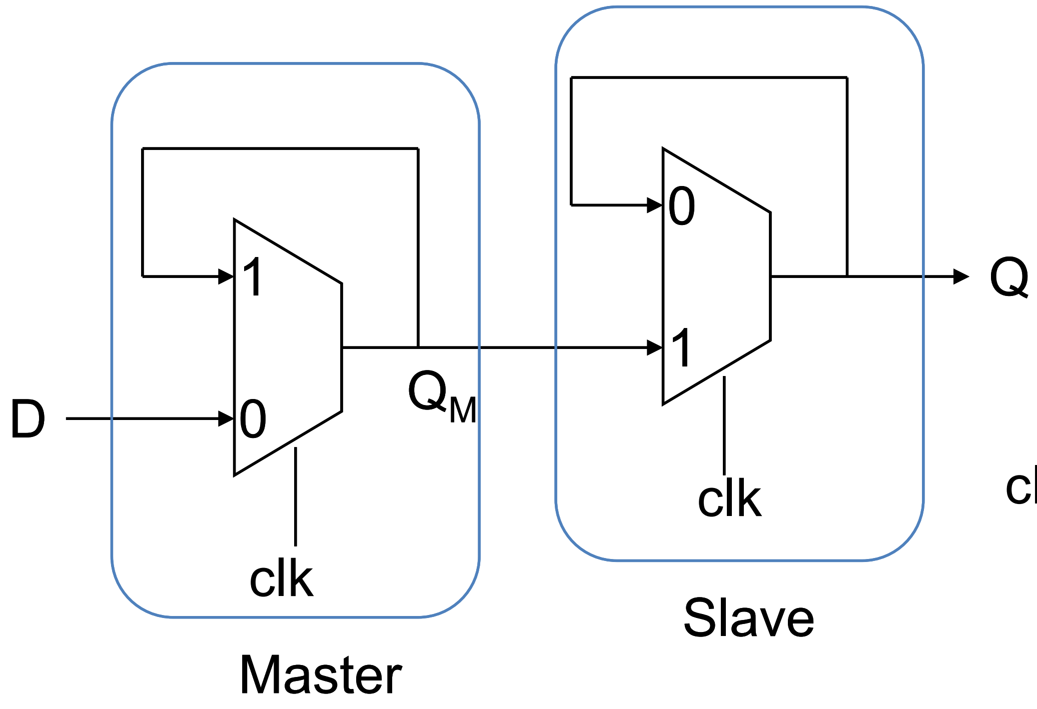
- Flip-flop is built as pair of back-to-back latches
- Built from master and slave D latches

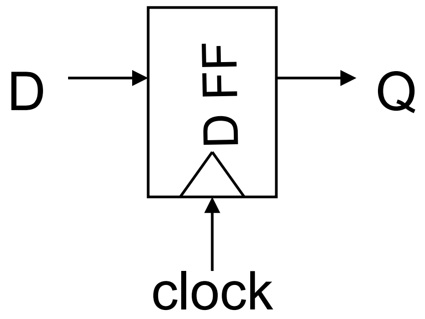


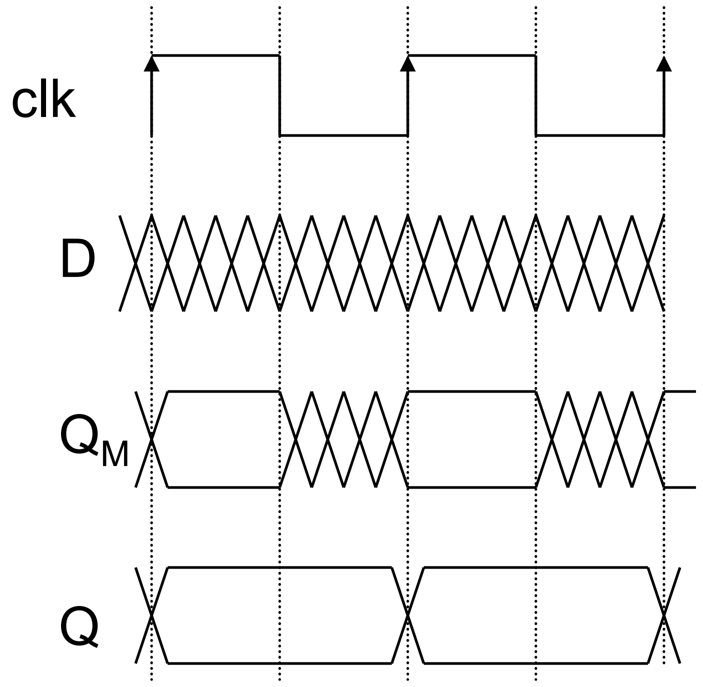




Master-Slave Edge-Triggered Flip-Flop

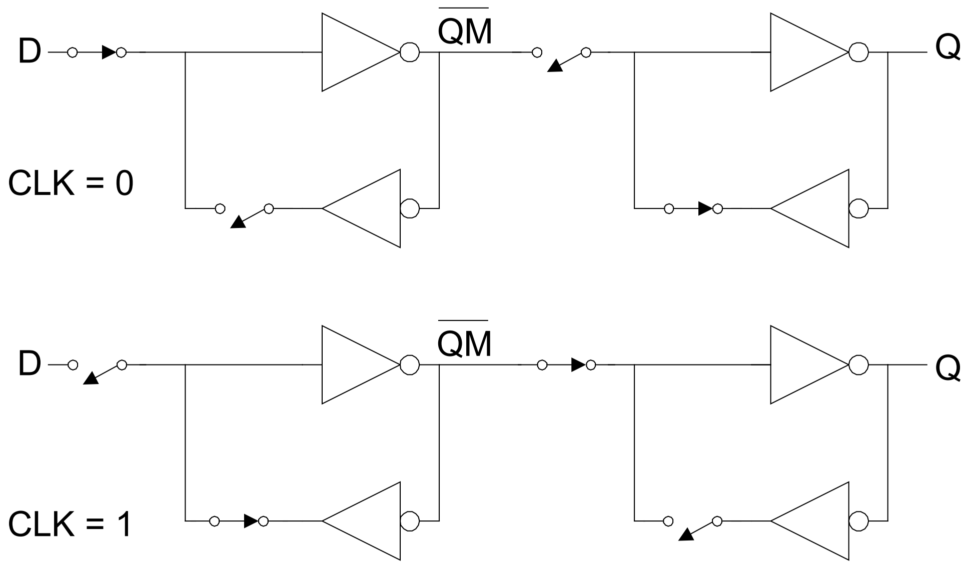


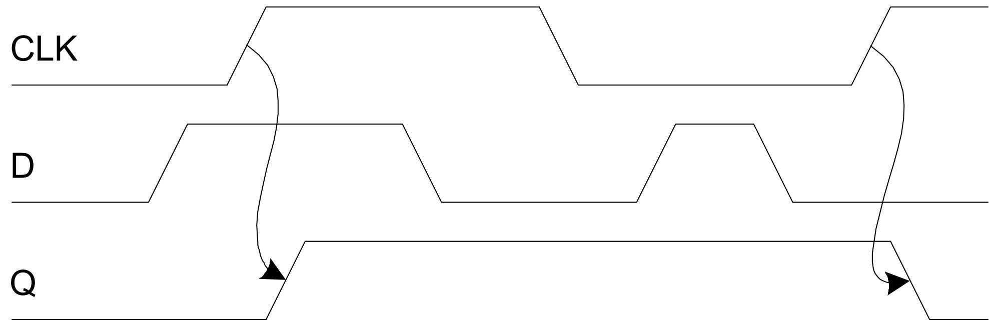




clk = 0	transparent	hold
clk = 0→1	hold	transparent

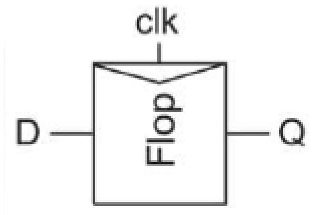
D Flip-flop Operation

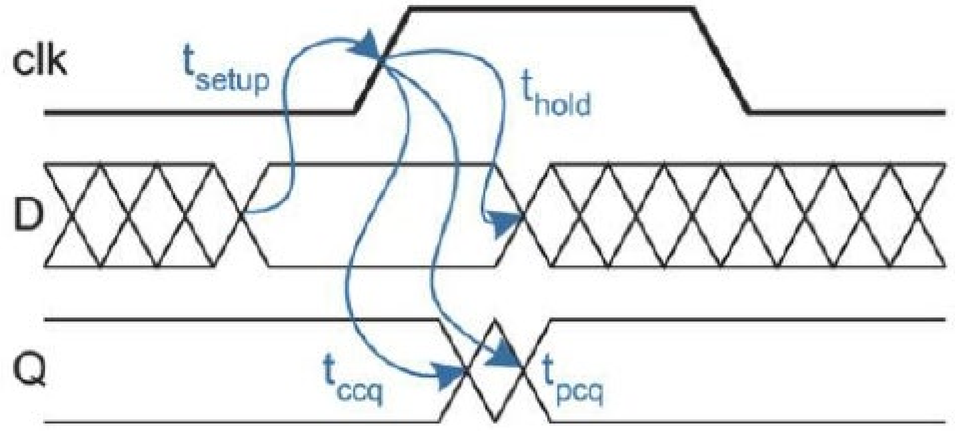




Timing Analysis-Timing of D Flip-Flop

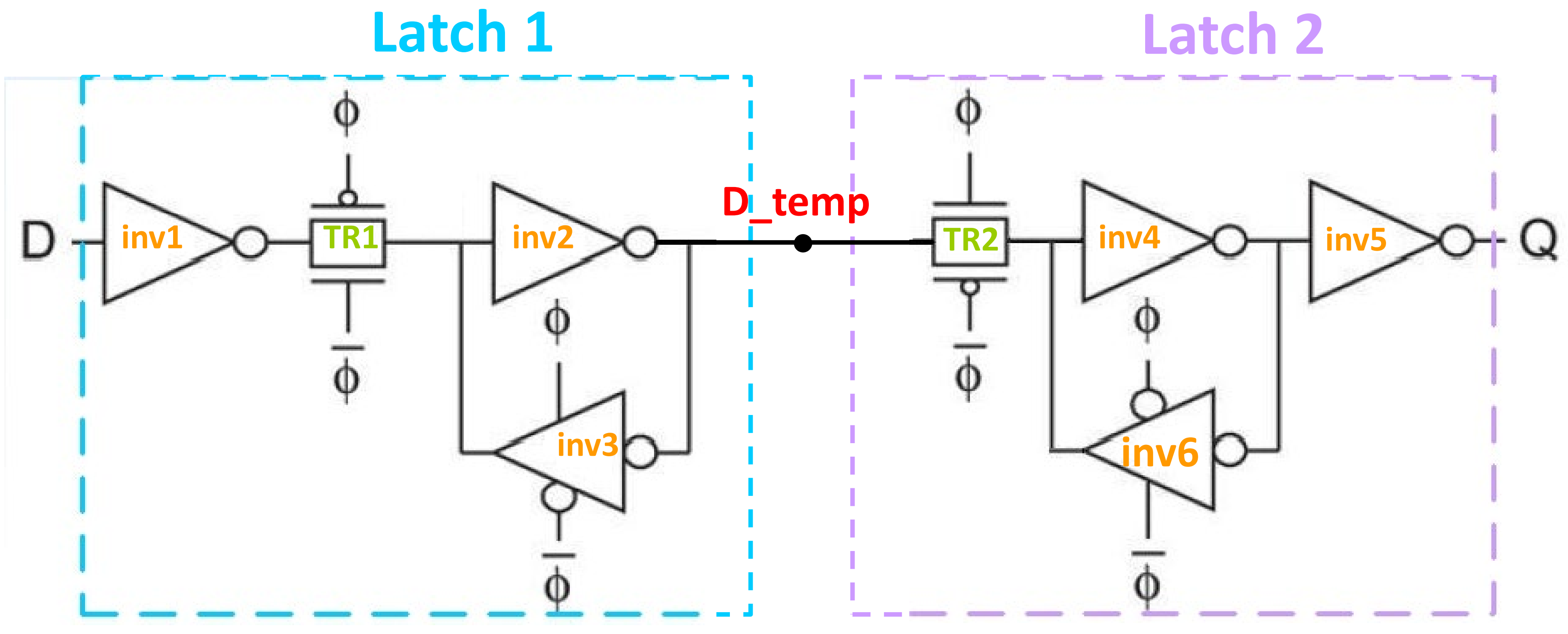
- **Setup time (Tsetup):** the time that the input signal must be stabilized before the clock edge.
- **Hold time (Thold):** the time that the input signal must be stabilized after the clock edge.
- **Clk-to-Q contamination delay (Tccq):** the contamination time that Q is first changed after the clock edge.
- **Clk-to-Q propagation delay (Tpcq):** the propagation time that Q reaches steady state after the clock edge.





Timing Analysis-Timing of D Flip-Flop

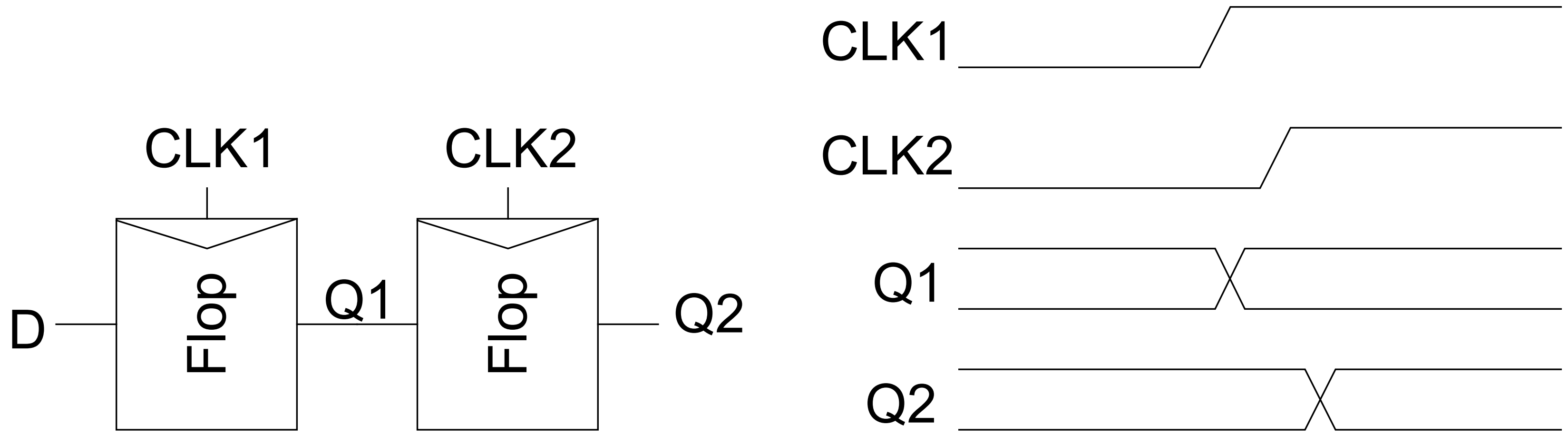
- **Static flip-flop**
  - **tsetup**: D -> inv1 -> inv2 -> inv3
  - **thold**: TR1 (ON->OFF)
  - **tpcq**: D_temp -> TR2 (OFF->ON) -> inv4 -> inv5 -> Q



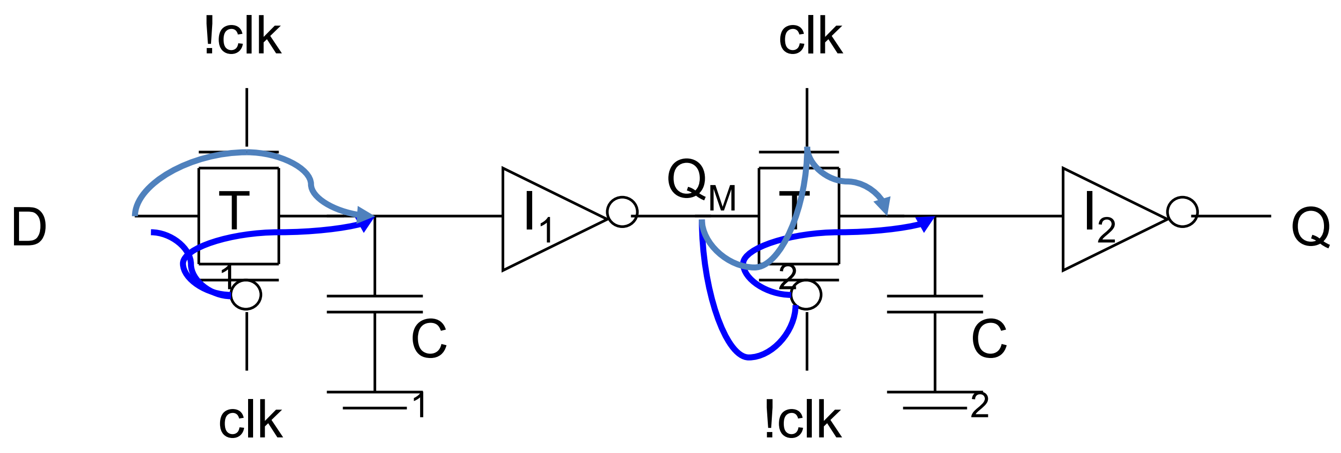
Determine the CLK-to-Q Delay

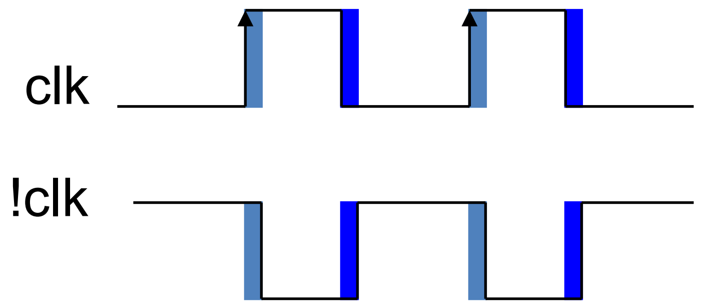
Race Condition

- Back-to-back flops can malfunction from clock skew
  - Second flip-flop fires late
  - Sees first flip-flop change and captures its result
  - Called *hold-time failure* or *race condition*



Race Conditions in Dynamic FF





0-0 overlap race condition

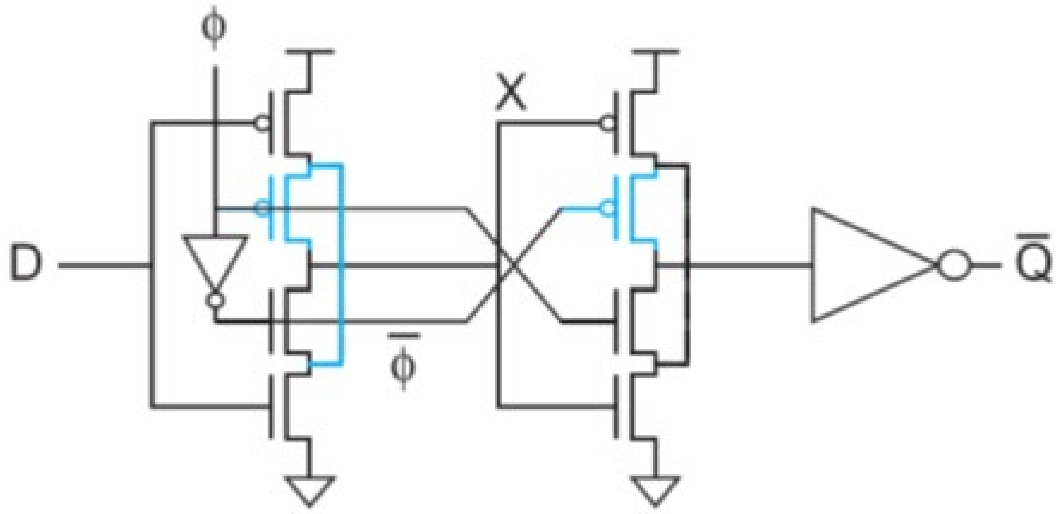
$t_{overlap0-0} < t_{T1} + t_{I1} + t_{T2}$

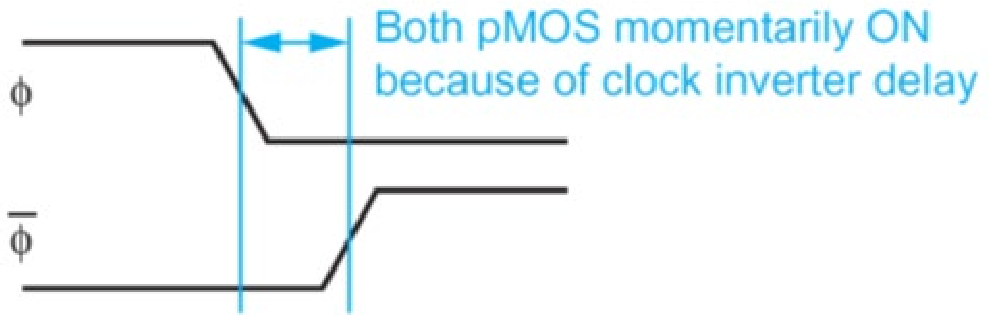
1-1 overlap race condition

$t_{overlap1-1} < t_{hold}$

NORA Dynamic FF

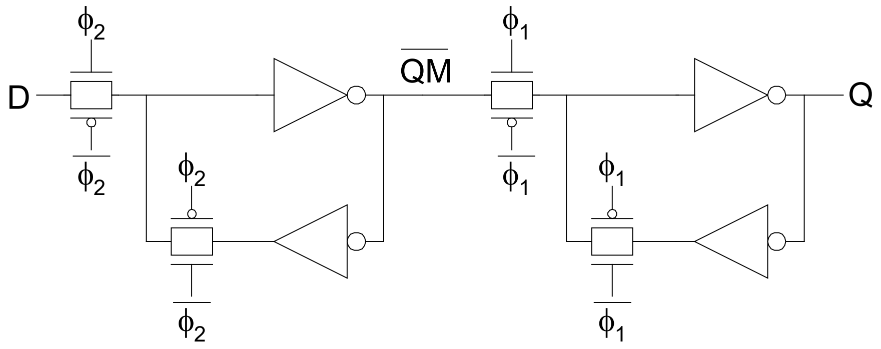
- NORA – no racing

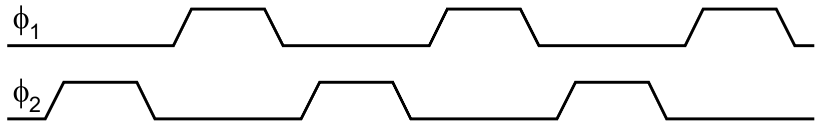




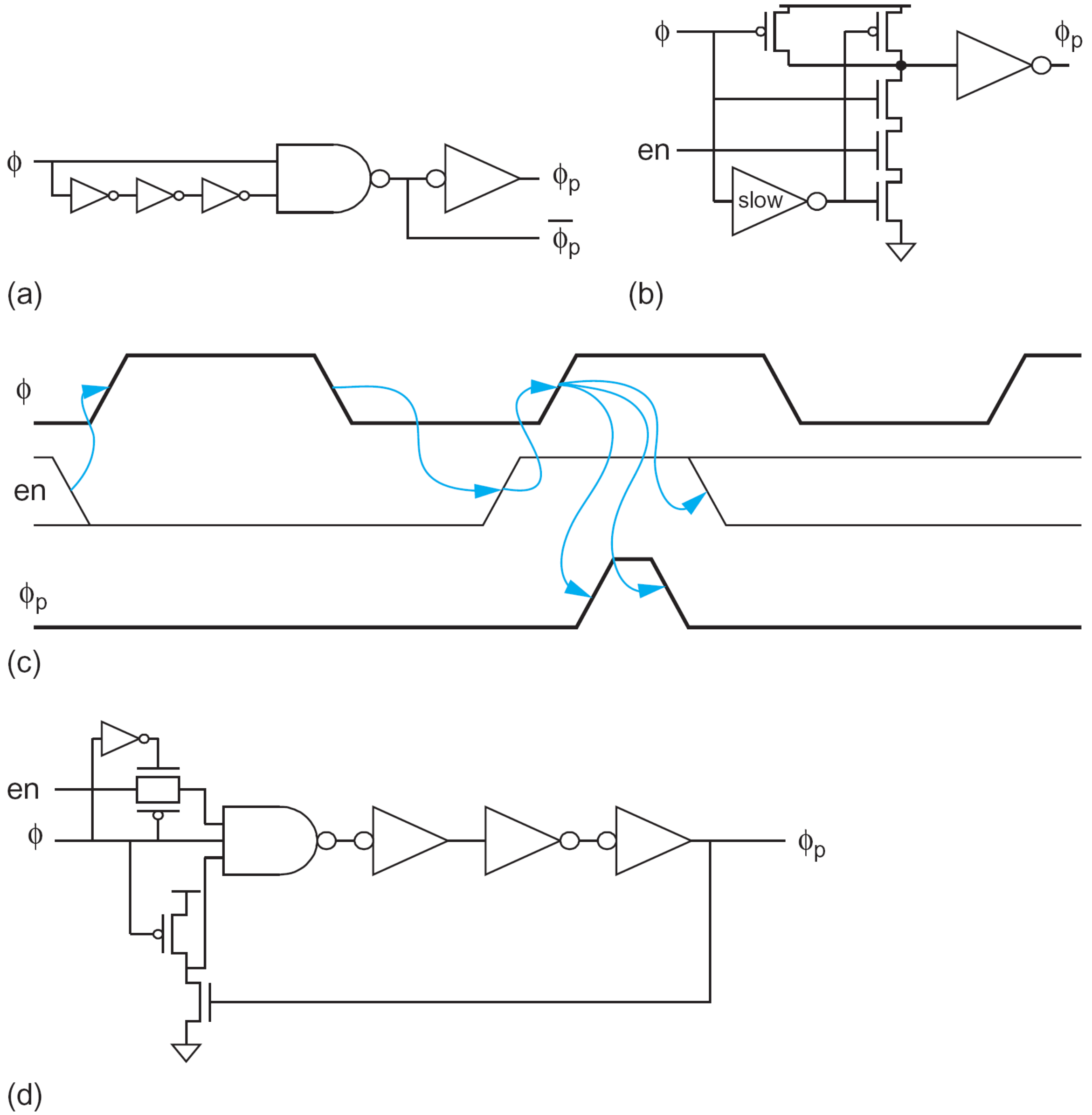
Nonoverlapping Clocks

- Nonoverlapping clocks can prevent races
  - As long as nonoverlap exceeds clock skew
- We will use them in this class for safe design
  - Industry manages skew more carefully instead



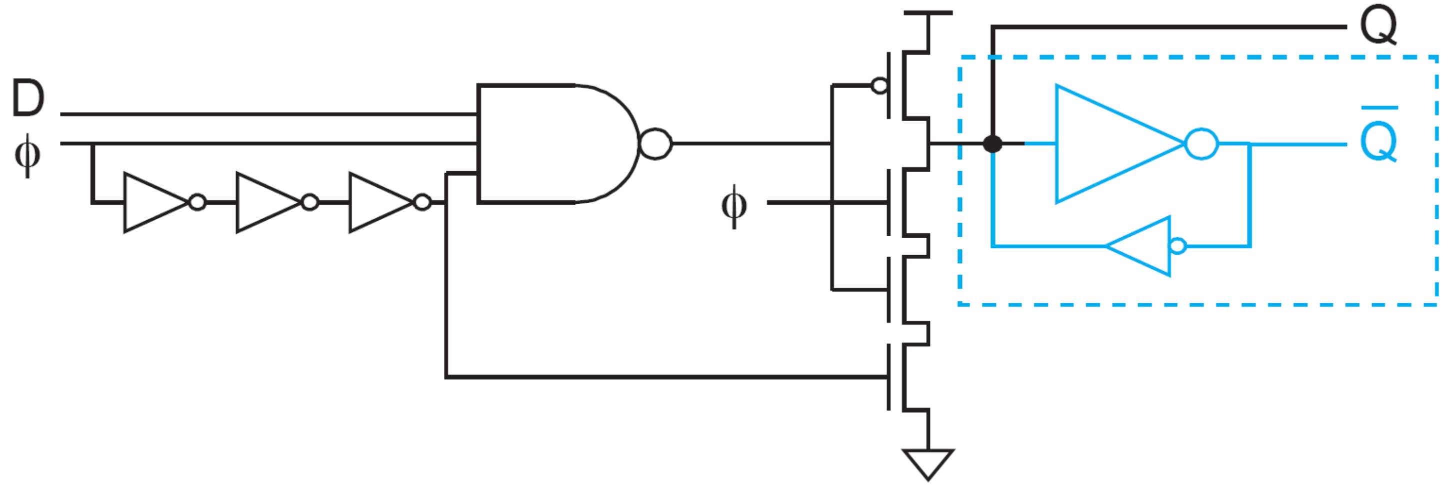


Pulse Generator + Latch (Pulsed Latch)



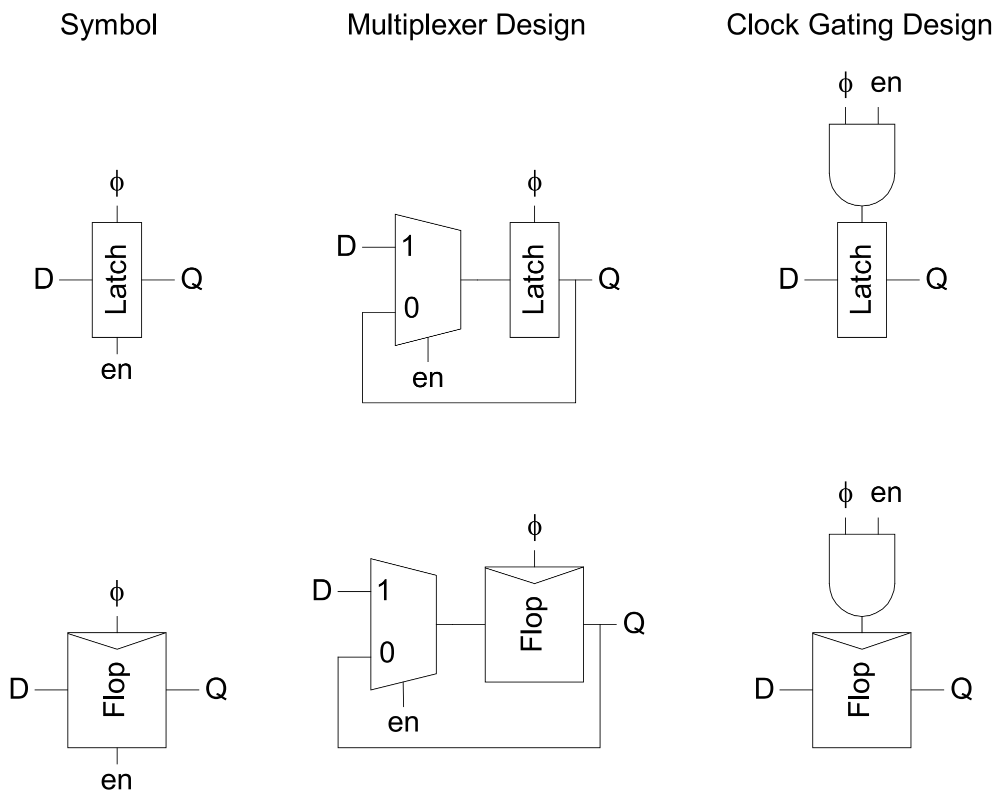
Pulsed Latch using Pulse Generators

- Pulsed generator + Latch = pulsed latch
  - Edge-sensitive



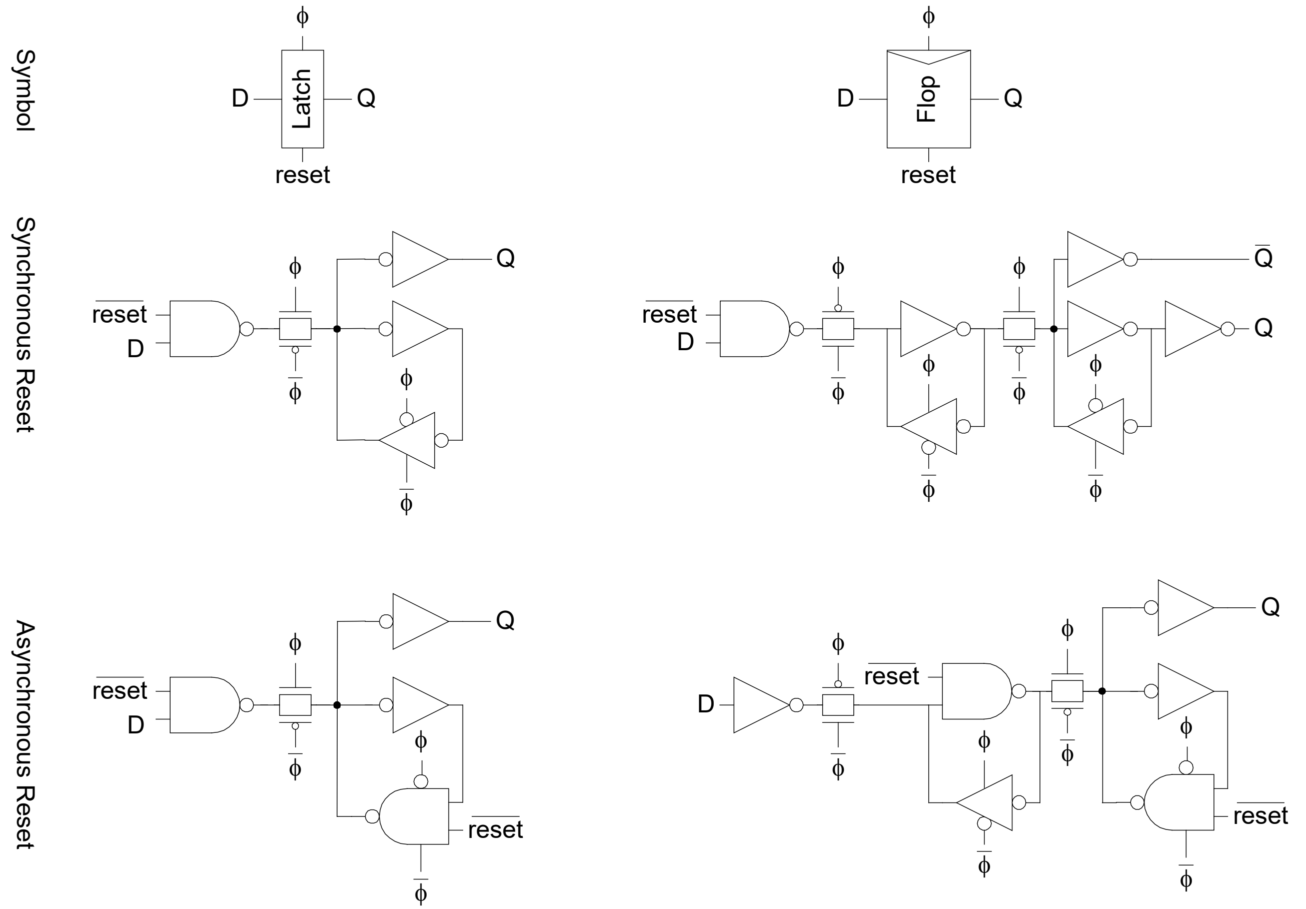
Enable

- Enable: ignore clock when en = 0
  - ◆ Mux: increase latch D-Q delay
  - ◆ Clock Gating: increase en setup time, skew



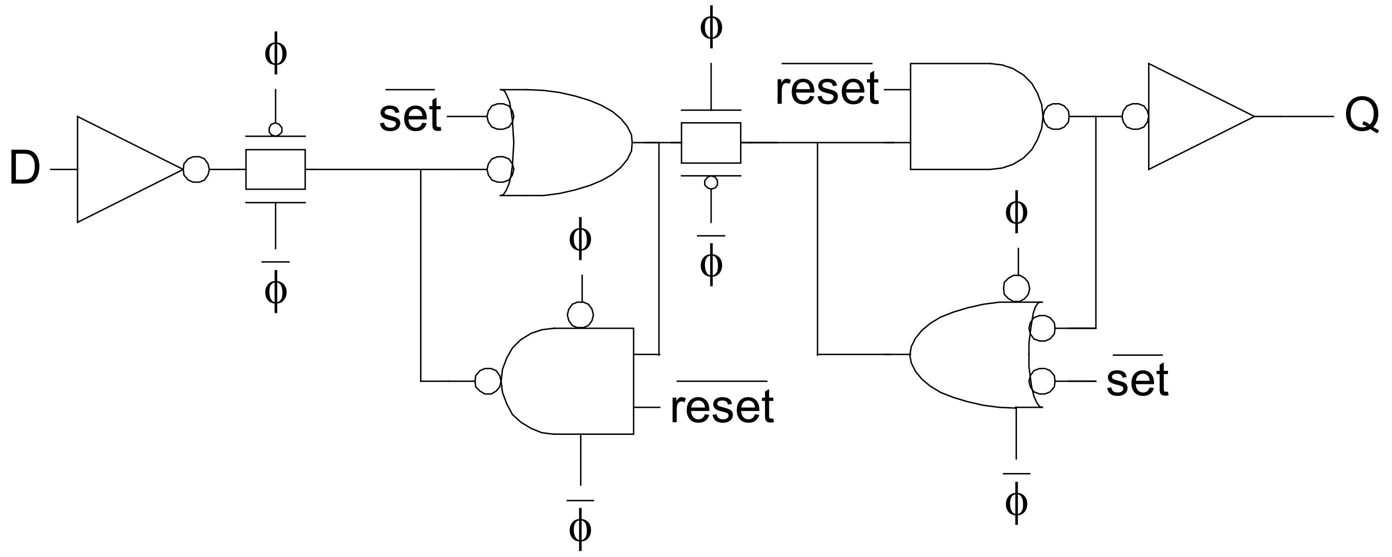
Reset

- Force output low when reset asserted
- Synchronous vs. asynchronous



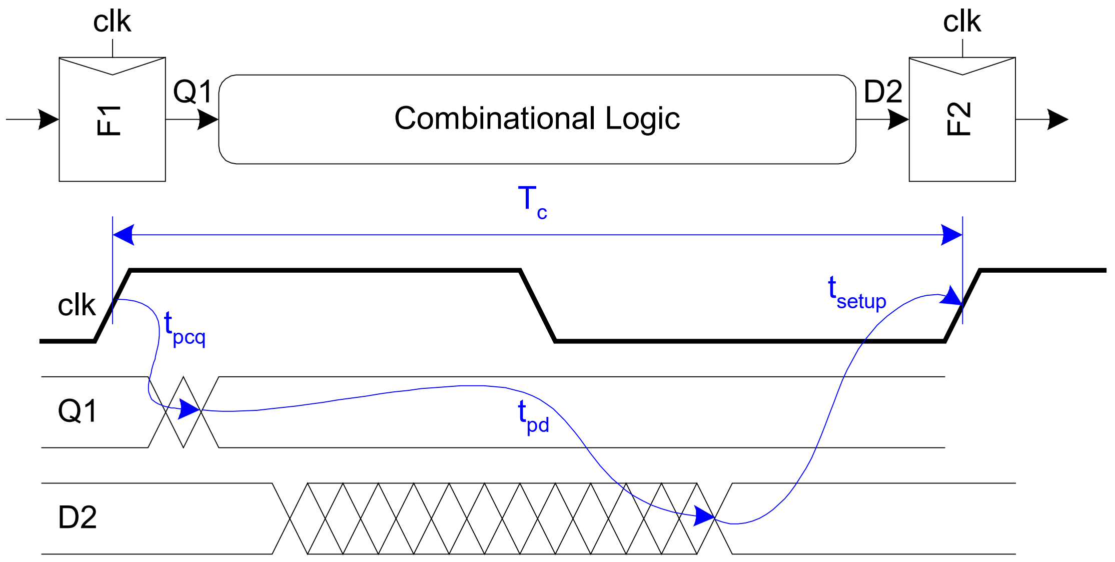
Set / Reset

- Set forces output high when enabled
- Flip-flop with asynchronous set and reset



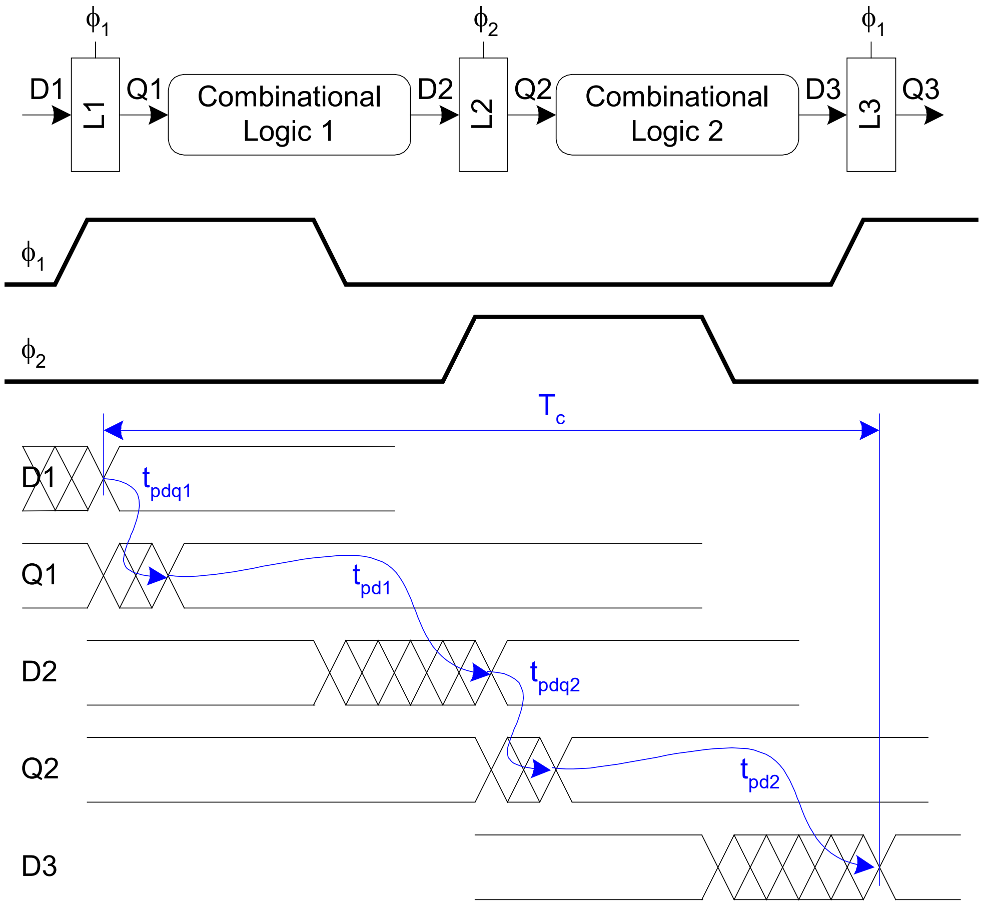
Max-Delay: Flip-Flops

- Maximum delay of the combination logic
- Avoid setup time violation



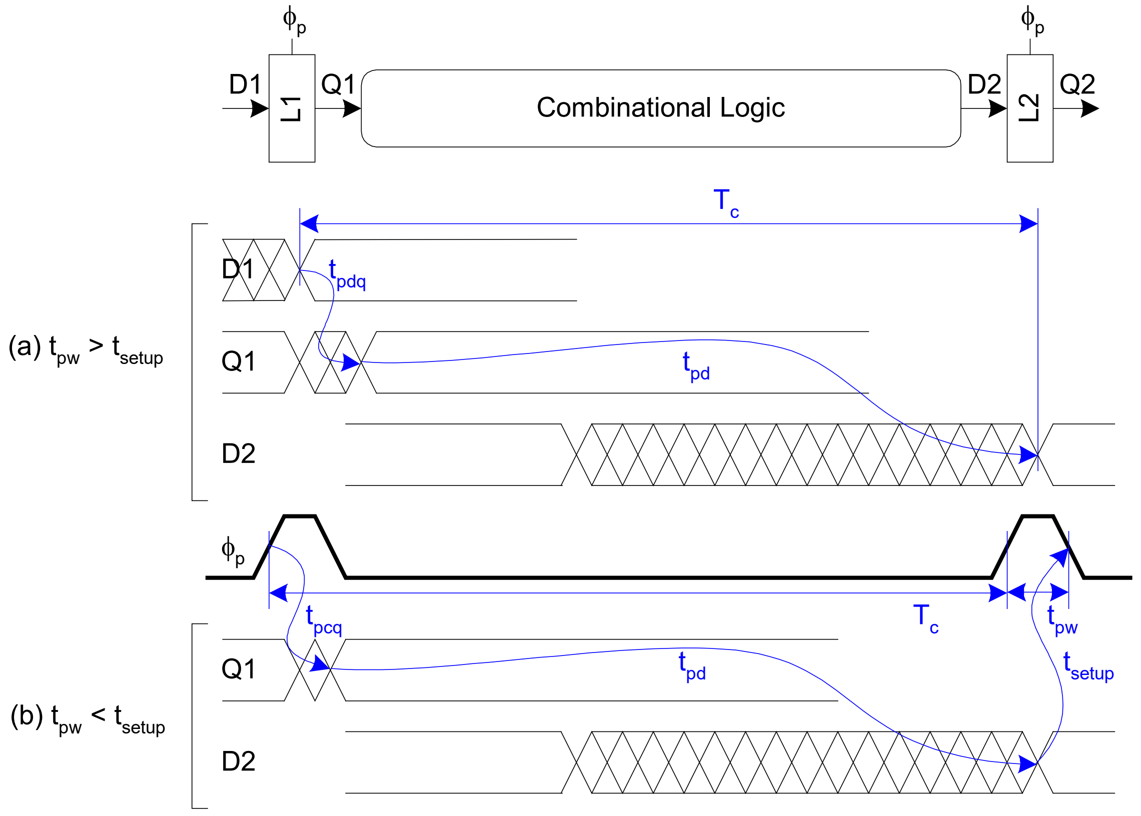
$$t_{pd} \leq T_c - \underbrace{\left(t_{setup} + t_{pcq}\right)}_{\text{sequencing overhead}}$$

Max Delay: 2-Phase Latches



$$t_{pd} = t_{pd1} + t_{pd2} \le T_c - \underbrace{\left(2t_{pdq}\right)}_{\text{sequencing overhead}}$$

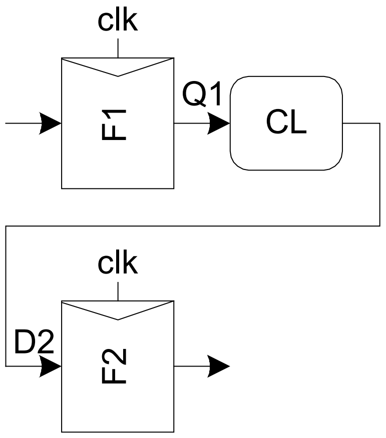
Max Delay: Pulsed Latches

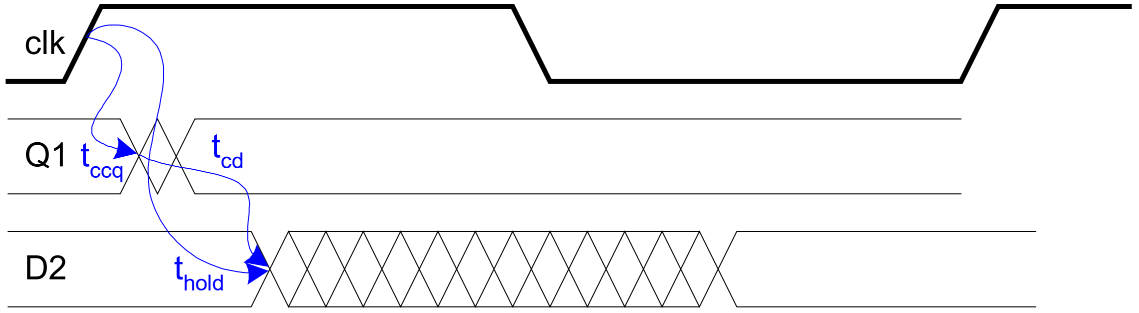


$$t_{pd} \le T_c - \underbrace{\max\left(t_{pdq}, t_{pcq} + t_{\text{setup}} - t_{pw}\right)}_{\text{sequencing overhead}}$$

Min-Delay: Flip-Flops

- Minimum delay of the combination logic
- Avoid hold-time violation

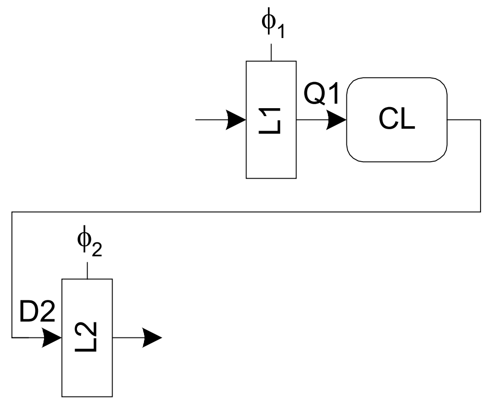




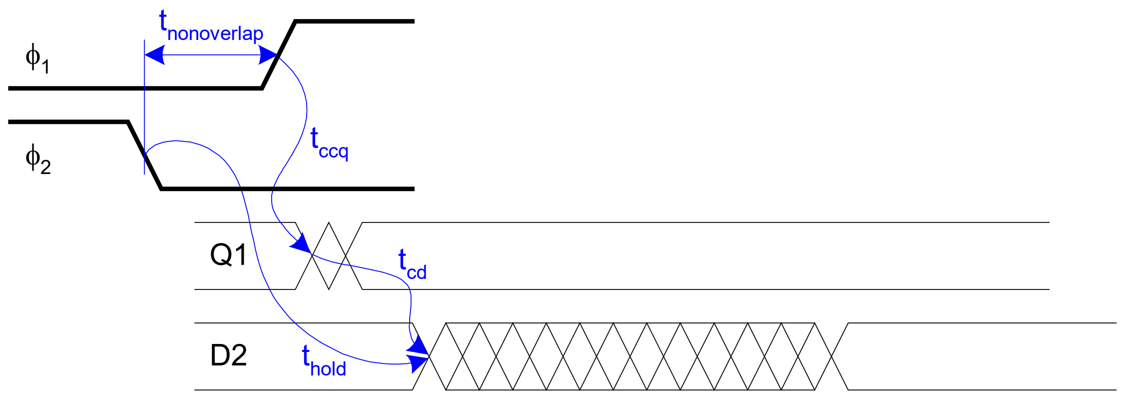
$$t_{cd} \geq t_{\text{hold}} - t_{ccq}$$

Min-Delay: 2-Phase Latches

- Hold time reduced by nonoverlap
- Paradox: hold applies twice each cycle, vs. only once for flops.
- But a flop is made of two latches!

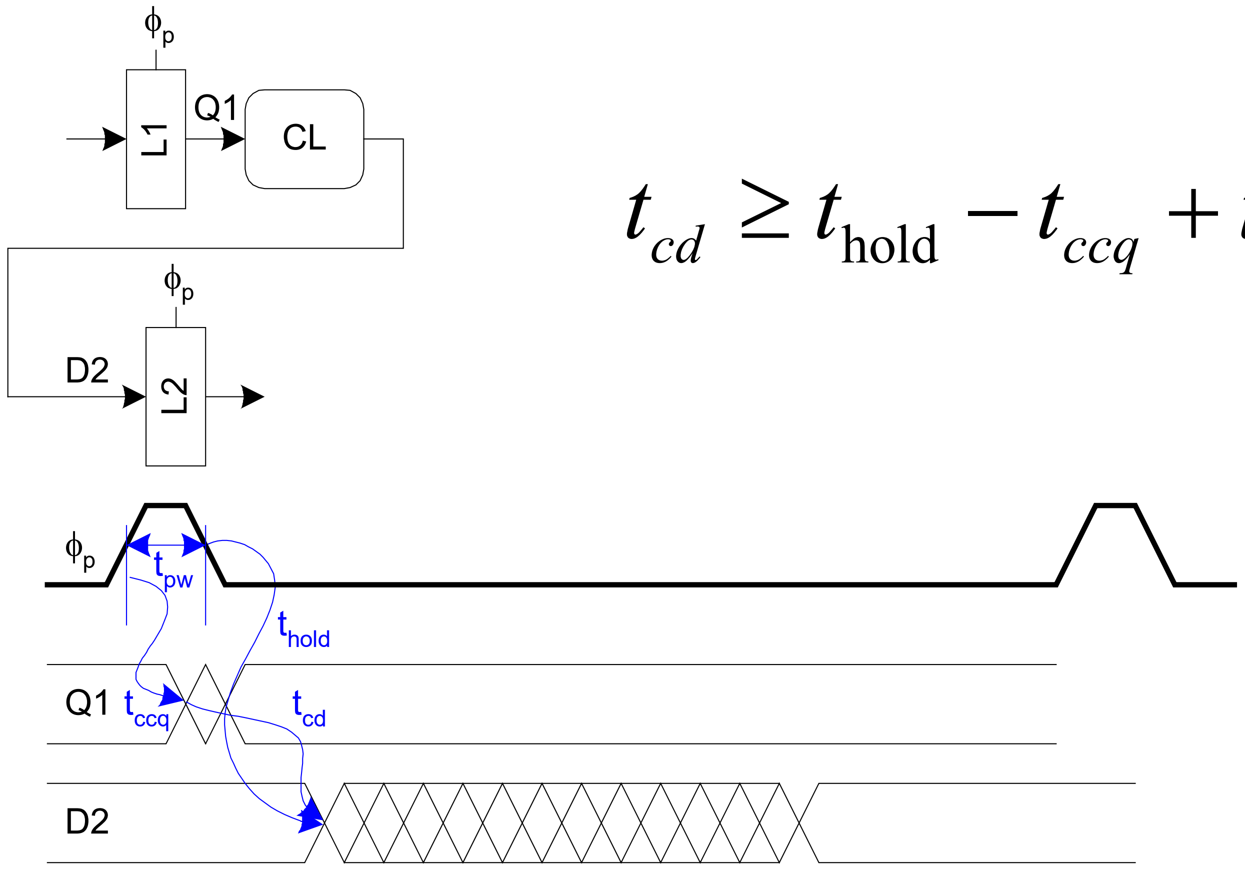


$$t_{cd1}, t_{cd2} \geq t_{\text{hold}} - t_{ccq} - t_{\text{nonoverlap}}$$



Min-Delay: Pulsed Latches

- Hold time increased by pulse width

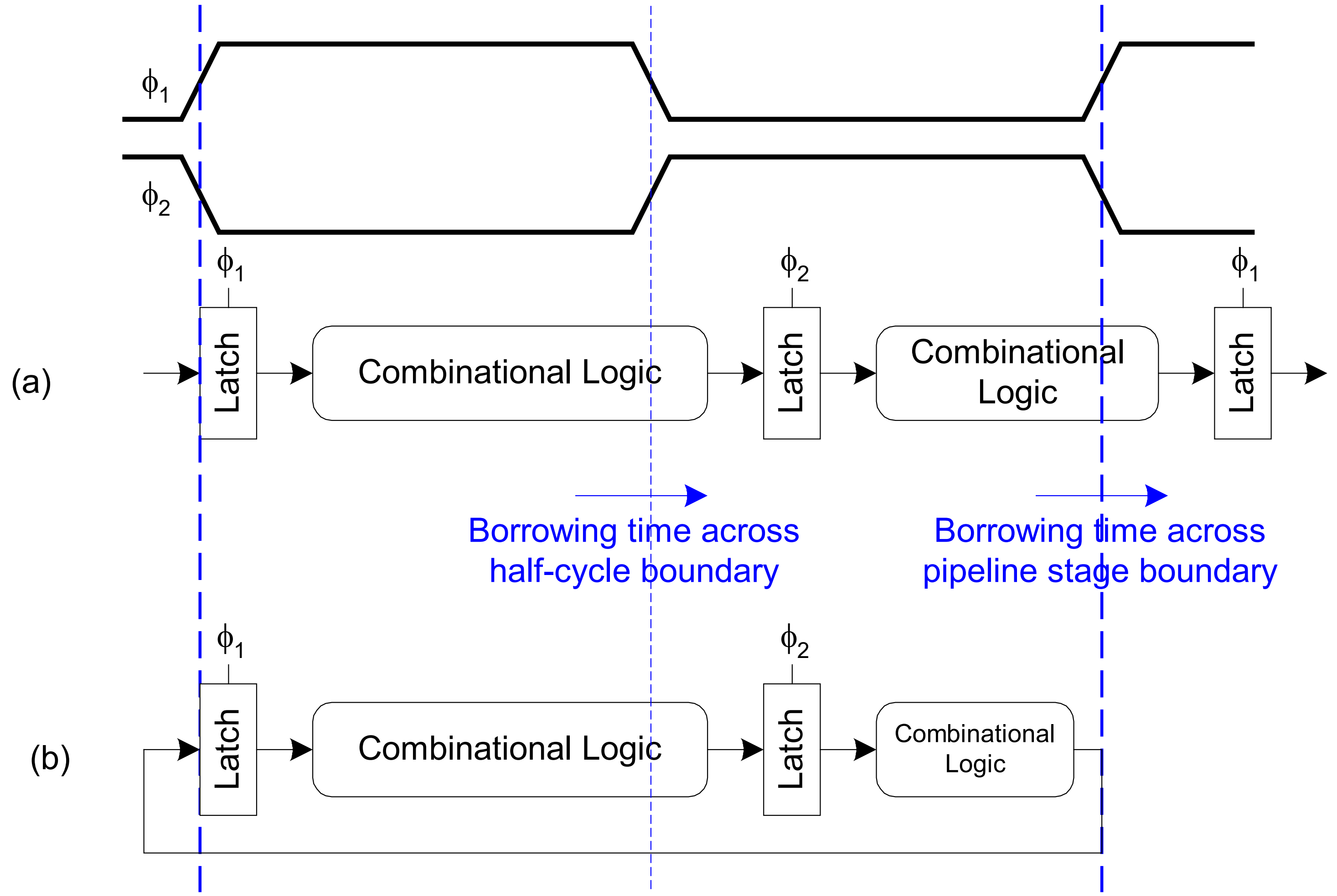


$$t_{cd} \geq t_{\text{hold}} - t_{ccq} + t_{pw}$$

Time Borrowing

- In a flop-based system:
  - Data launches on one rising edge
  - Must setup before next rising edge
  - If it arrives late, system fails
  - If it arrives early, time is wasted
  - Flops have hard edges
- In a latch-based system
  - Data can pass through latch while transparent
  - Long cycle of logic can borrow time into next
  - As long as each loop completes in one cycle

Time Borrowing Example



Loops may borrow time internally but must complete within the cycle

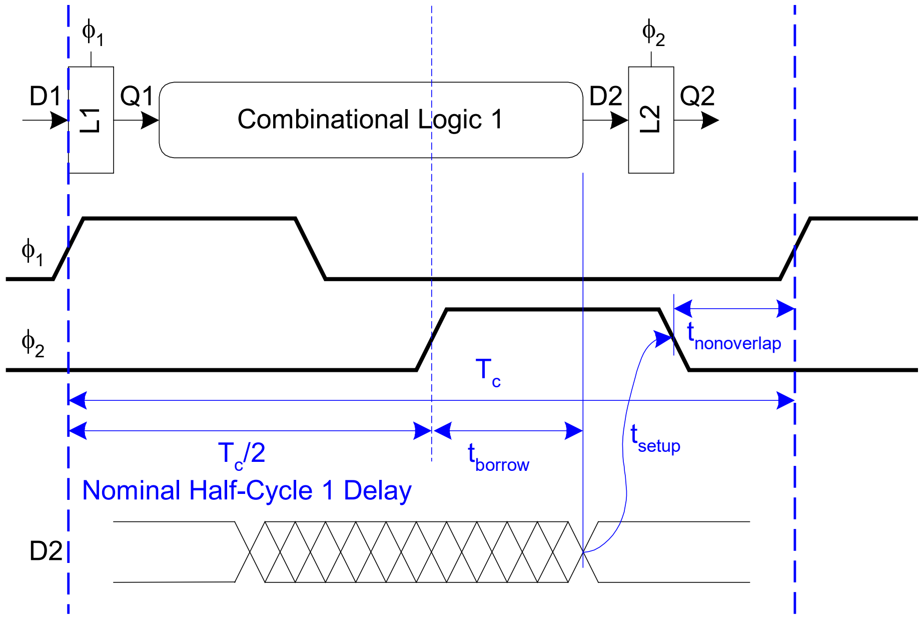
How Much Borrowing?

2-Phase Latches

$$t_{\text{borrow}} \le \frac{T_c}{2} - \left(t_{\text{setup}} + t_{\text{nonoverlap}}\right)$$

Pulsed Latches

$$t_{\text{borrow}} \le t_{pw} - t_{\text{setup}}$$



Time borrowing can be used to balance logic within a pipeline
But did not increase the amount of time available in a clock cycle

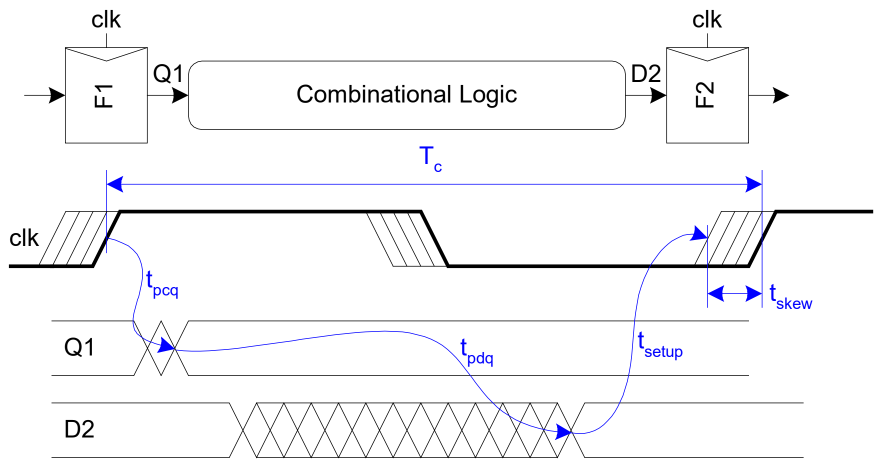
Clock Skew

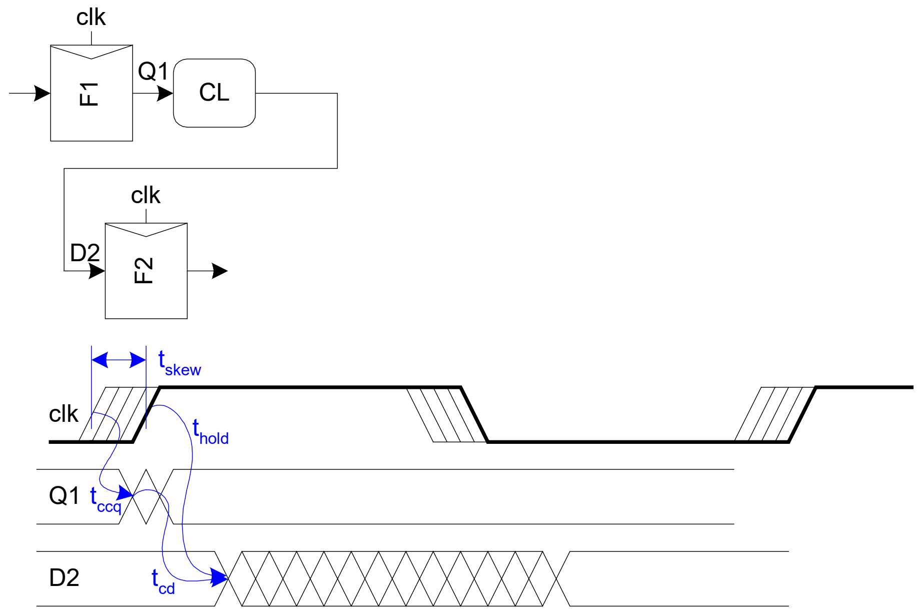
- We have assumed zero clock skew
- Clocks really have uncertainty in arrival time
  - **Decreases** maximum propagation delay
  - Increases minimum contamination delay
  - Decreases time borrowing

Skew: Flip-Flops

$$t_{pd} \le T_c - \underbrace{\left(t_{pcq} + t_{\text{setup}} + t_{\text{skew}}\right)}_{\text{sequencing overhead}}$$

$$t_{cd} \ge t_{\text{hold}} - t_{ccq} + t_{\text{skew}}$$





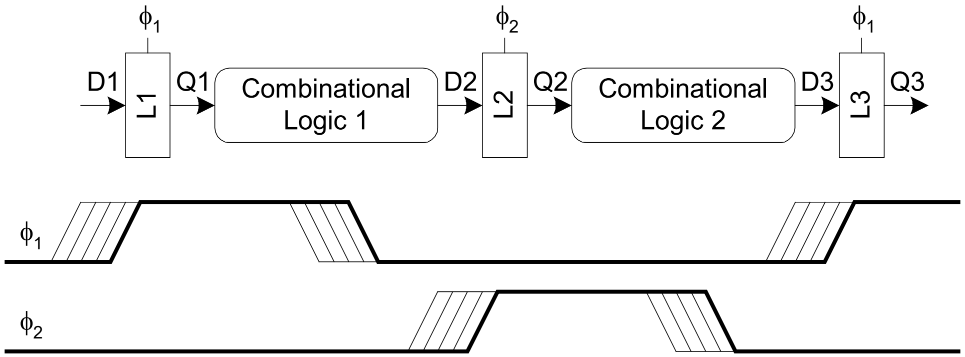
Skew: Latches

2-Phase Latches

$$t_{pd} \le T_c - \underbrace{\left(2t_{pdq}\right)}_{\text{sequencing overhead}}$$

$$t_{cd1}, t_{cd2} \ge t_{\text{hold}} - t_{ccq} - t_{\text{nonoverlap}} + t_{\text{skew}}$$

$$t_{\text{borrow}} \le \frac{T_c}{2} - \left(t_{\text{setup}} + t_{\text{nonoverlap}} + t_{\text{skew}}\right)$$



Pulsed Latches

$$t_{pd} \le T_c - \underbrace{\max\left(t_{pdq}, t_{pcq} + t_{\text{setup}} - t_{pw} + t_{\text{skew}}\right)}_{\text{sequencing overhead}}$$

$$t_{cd} \ge t_{\text{hold}} + t_{pw} - t_{ccq} + t_{\text{skew}}$$

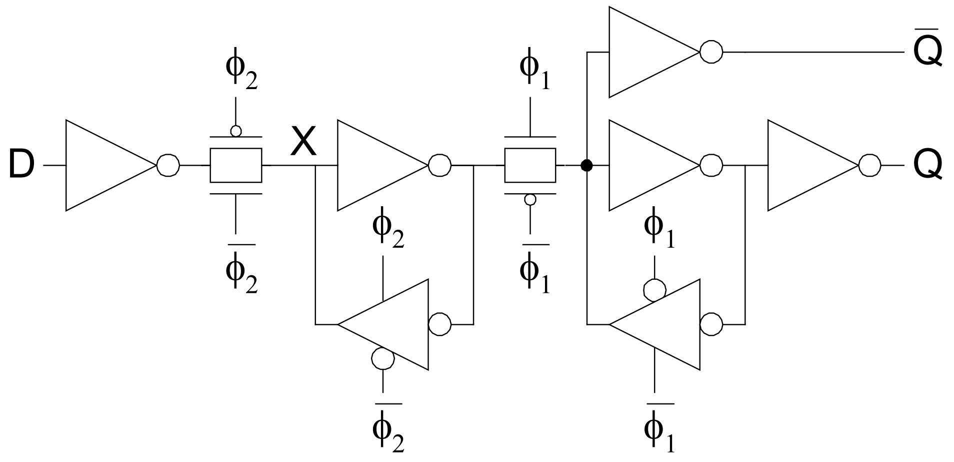
$$t_{\text{borrow}} \le t_{pw} - \left(t_{\text{setup}} + t_{\text{skew}}\right)$$

Two-Phase Clocking

- If setup times are violated, reduce clock speed
- If hold times are violated, chip fails at any speed (Hold time fix after APR)
- An easy way to guarantee hold times is to use 2-phase latches with big nonoverlap times
- Call these clocks ϕ_1, ϕ_2 (ph1, ph2)

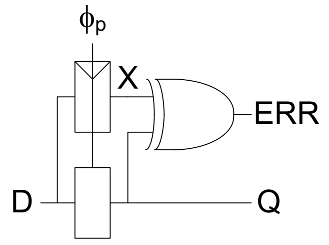
Safe Flip-Flop

- Past years used flip-flop with nonoverlapping clocks
  - Slow – nonoverlap adds to setup time
  - But no hold times
- In industry, use a better timing analyzer
  - Add buffers to slow signals if hold time is at risk

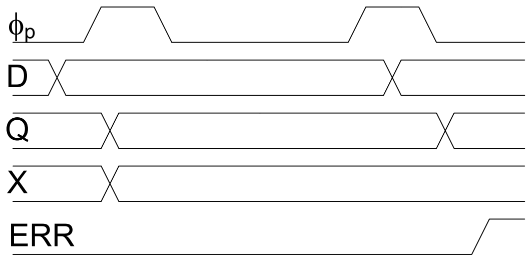


Adaptive Sequencing

- Designers include timing margin
  - Voltage
  - Temperature
  - Process variation
  - Data dependency
  - Tool inaccuracies



Double sampling



- Alternative: run faster and check for near failures
  - Idea introduced as “Razor”
    - Increase frequency until at the verge of error
    - Can reduce cycle time by ~30%

Summary

- Flip-Flops:
  - Very easy to use, supported by all tools
- 2-Phase Transparent Latches:
  - Lots of skew tolerance and time borrowing
- Pulsed Latches:
  - Fast, some skew tol & borrow, hold time risk

	Sequencing overhead ($T_c - t_{pd}$)	Minimum logic delay t_{cd}	Time borrowing t_{borrow}
Flip-Flops	$t_{pcq} + t_{\text{setup}} + t_{\text{skew}}$	$t_{\text{hold}} - t_{ccq} + t_{\text{skew}}$	0
Two-Phase Transparent Latches	$2t_{pdq}$	$t_{\text{hold}} - t_{ccq} - t_{\text{nonoverlap}} + t_{\text{skew}}$ in each half-cycle	$\frac{T_c}{2} - \left(t_{\text{setup}} + t_{\text{nonoverlap}} + t_{\text{skew}}\right)$
Pulsed Latches	$\max\left(t_{pdq}, t_{pcq} + t_{\text{setup}} - t_{pw} + t_{\text{skew}}\right)$	$t_{\text{hold}} - t_{ccq} + t_{pw} + t_{\text{skew}}$	$t_{pw} - \left(t_{\text{setup}} + t_{\text{skew}}\right)$