University of Southern California

Viterbi School of Engineering

EE577A

VLSI System Design

Sequential Design

References: Slides and notes from Professors Pedram and Gupta, syllabus textbooks, online resources

Shahin Nazarian

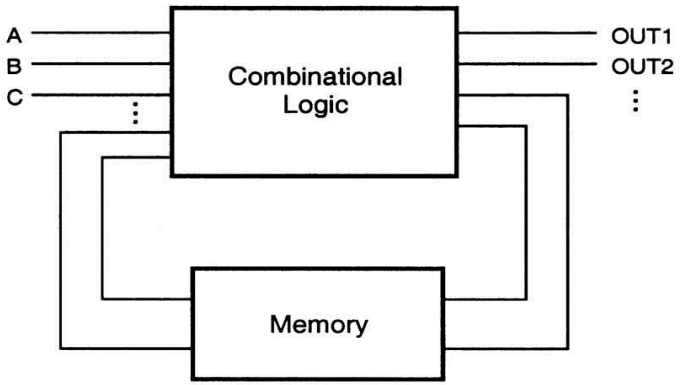
Spring 2013

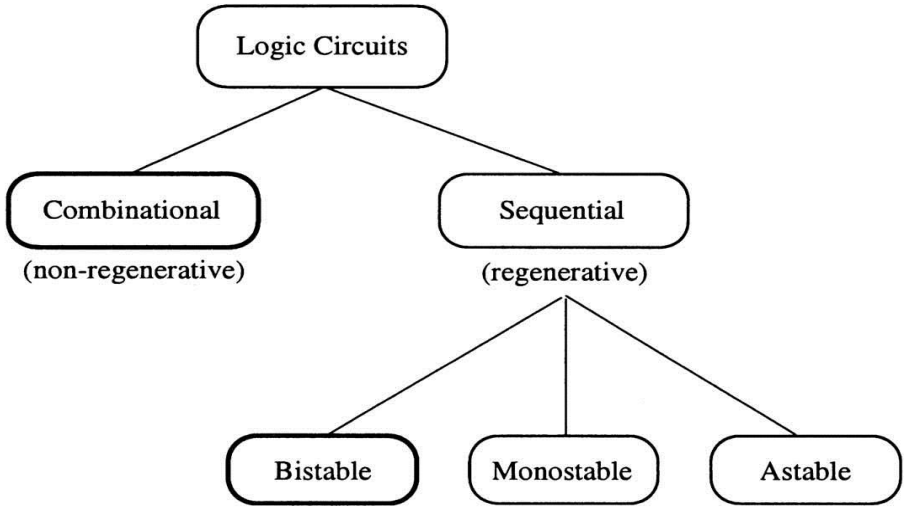
Sequential MOS Logic Circuits

- Output levels of a *combinational* logic circuit at any time are directly determined as Boolean functions of the applied input variables
  - Combinational logic circuits do not have the capability of storing any previous event or displaying an output behavior which is dependent upon the previously applied inputs.
  - Circuits of this type are also called *non-regenerative* circuits, since there is no *feed back* relationship between output and input
- The other major class of logic circuits is called *sequential* circuits, in which the output is determined by the present inputs as well as previously applied inputs

Types of Sequential Circuits

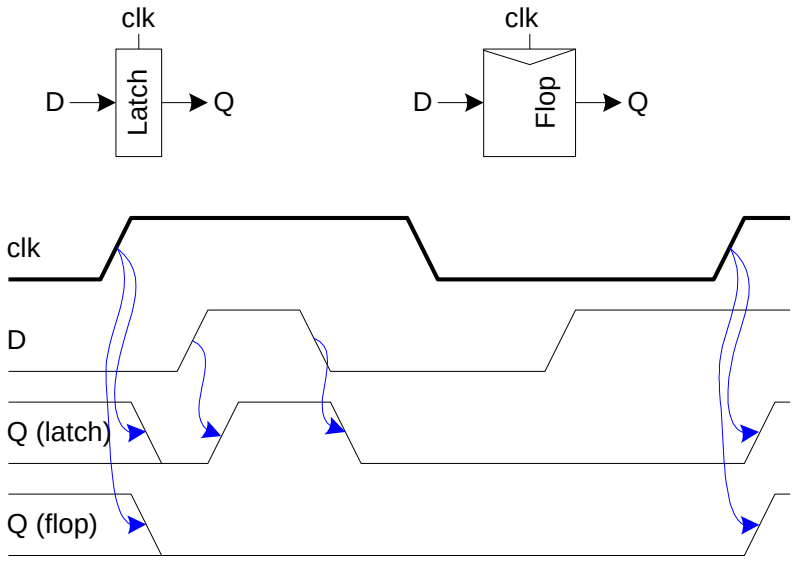
- There are three types for sequential circuits:
  - Bistable
  - Monostable
  - Astable





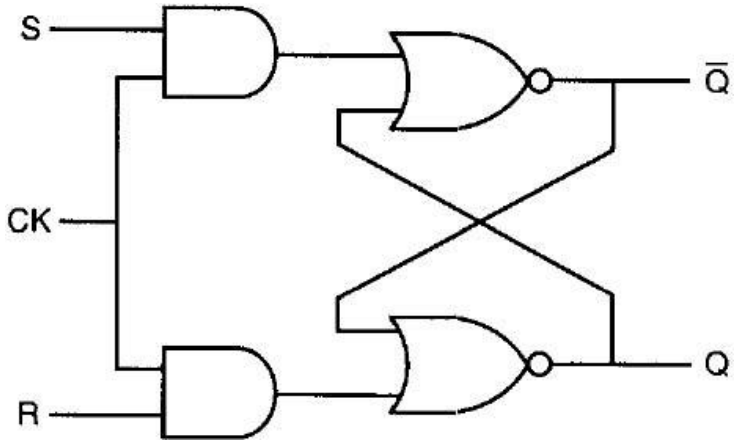
Sequencing Elements

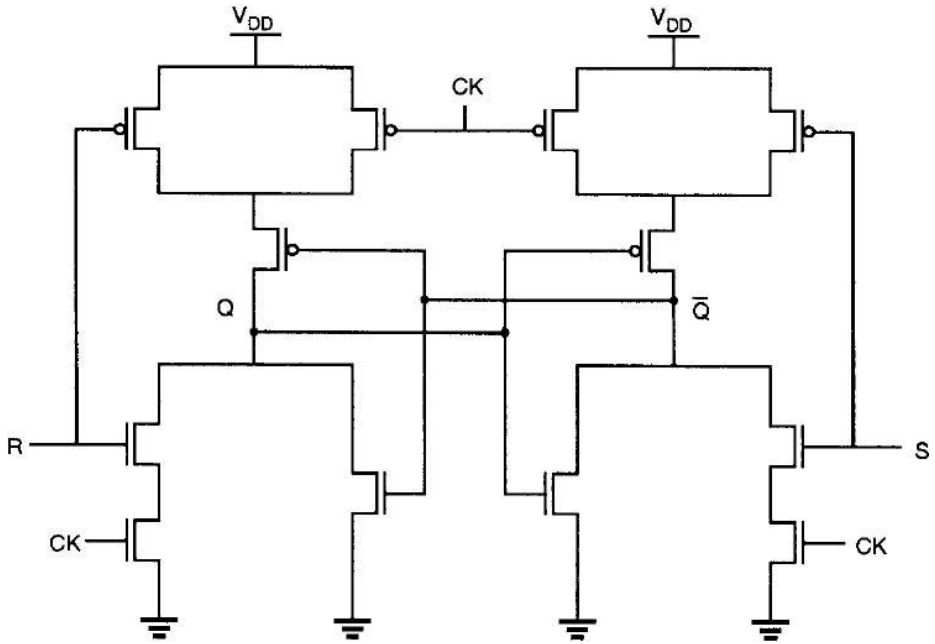
- Latch: Level sensitive
  - a.k.a. transparent latch, D latch
- Flip-flop: Edge triggered
  - a.k.a. master-slave flip-flop, D flip-flop, D register
- Timing Diagrams
  - Latch
    - Transparent
    - Opaque
  - Edge-trigger



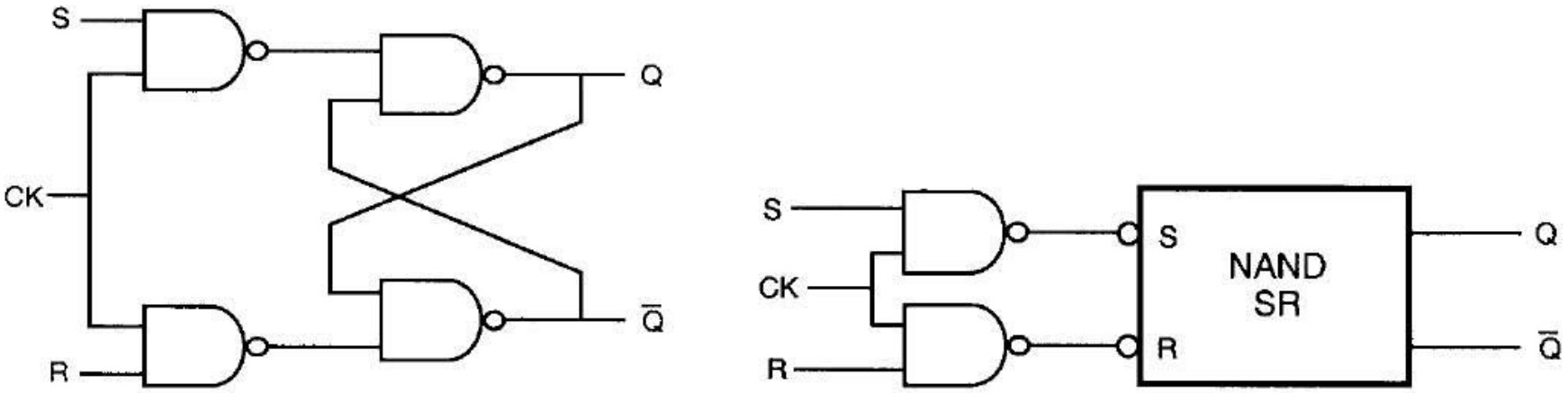
Clocked NOR-based SR Latch

- If CK="0" then inputs have no influence on the outputs, because outputs of the AND gates remain "0", which forces SR latch to hold its current state
- If CK="1" then input control signals, S and R, are permitted to reach SR latch, and possibly change its state (note that this is an active high implementation)
- As in non-clocked SR latch, the input combination S=R="1" is not allowed





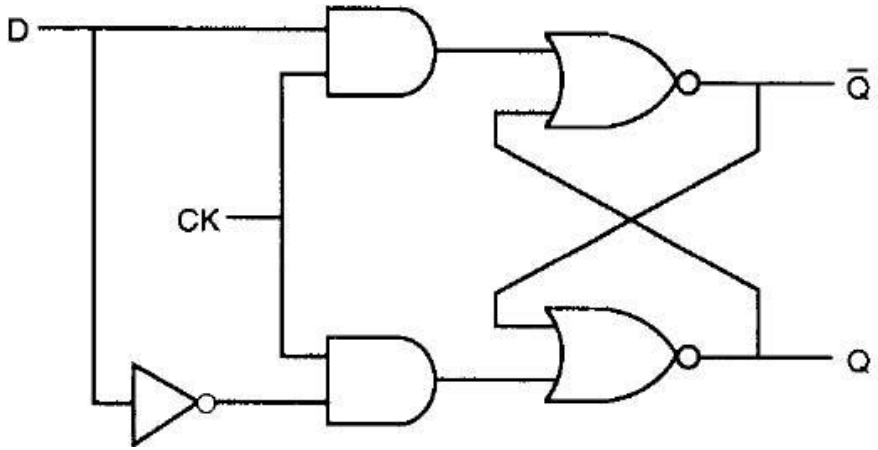
Clocked NAND-Based SR Latch

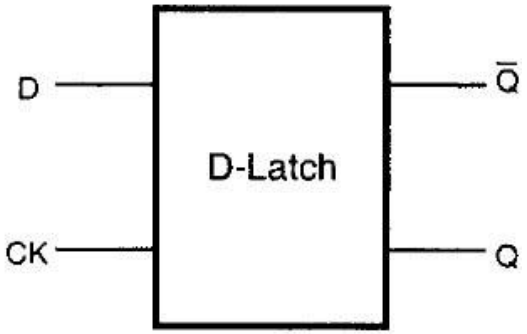


- In this case both input signals and the clock signal are *active high*
- The latch preserves its state as long as the CK signal is inactive, i.e., CK="0"

Gate-based Implementation of a D-Latch

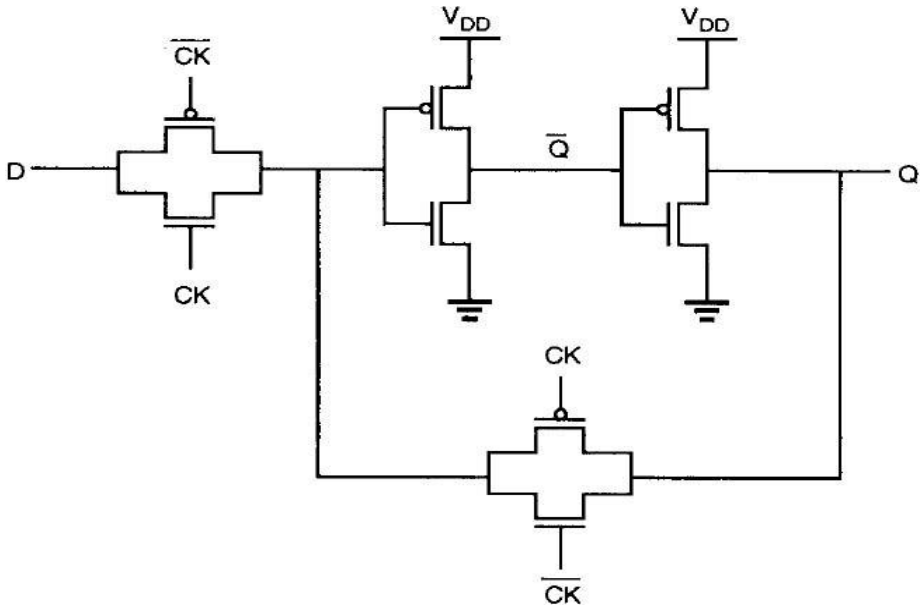
- By modifying the clocked NOR-based SR-latch, we obtain a D-latch
  - When CK=1, output Q assumes the value of the input D
  - When CK=0, the output preserves its state
  - Therefore, the CK input acts as an enable signal, which allows data to be accepted into the D-latch





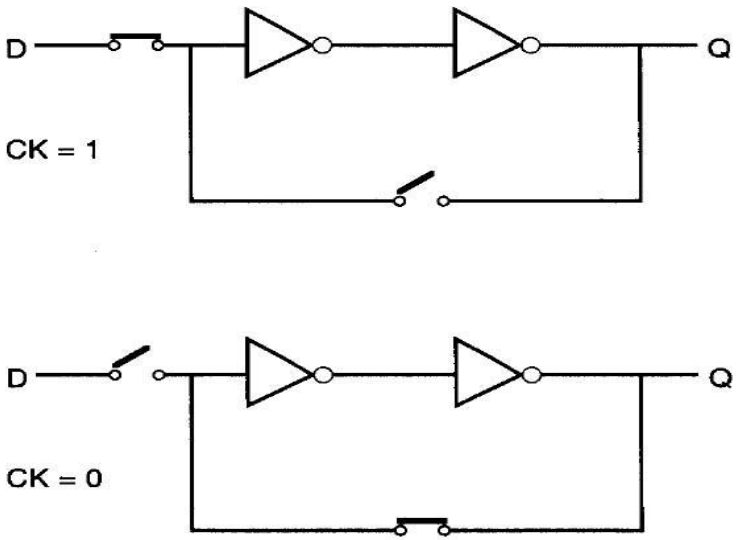
CMOS TG Implementation of the D-latch

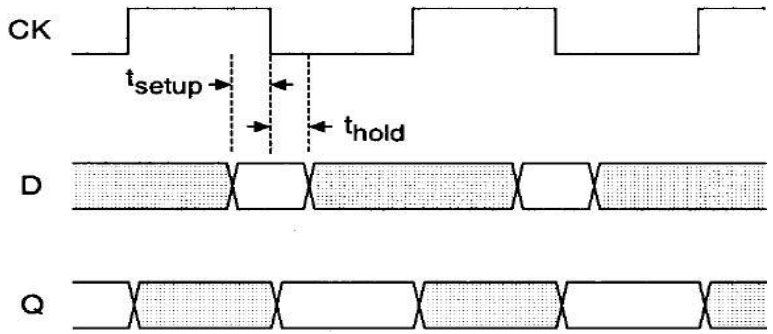
- Transmission gate (TG) at the input is activated by *CK*, and TG at the output is activated by $\overline{CK}$
- Thus, the input signal is accepted into the circuit when the clock is high, and this information is preserved as the state of the inverter loop when the clock goes low



Operation of TG-based D-Latch

- Operation of the D-latch can be understood better by replacing TG's with simple switches
- The valid D input must be stable for a short time before (*setup time*, t_{setup}) and after (*hold time*, t_{hold}) the negative clock transition
- Any violation of this condition may give rise to *metastability* problems

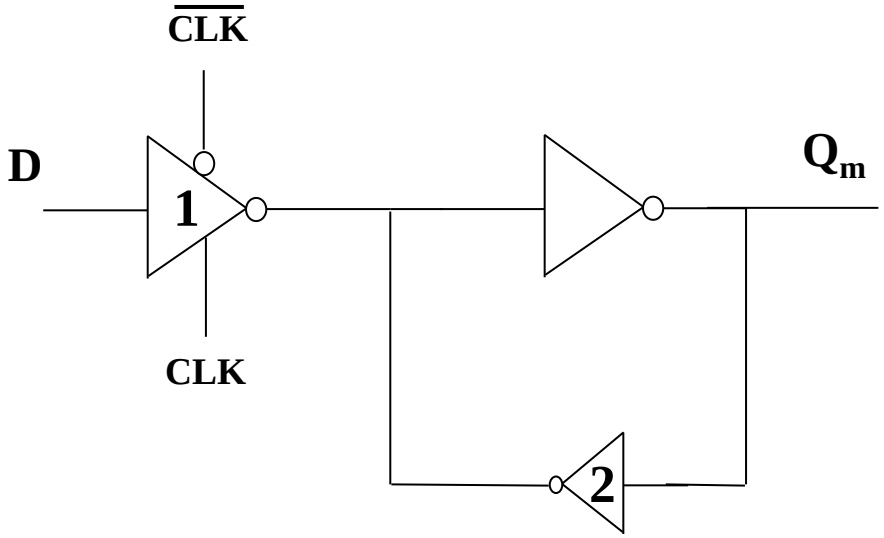




Another Implementation of a D-Latch

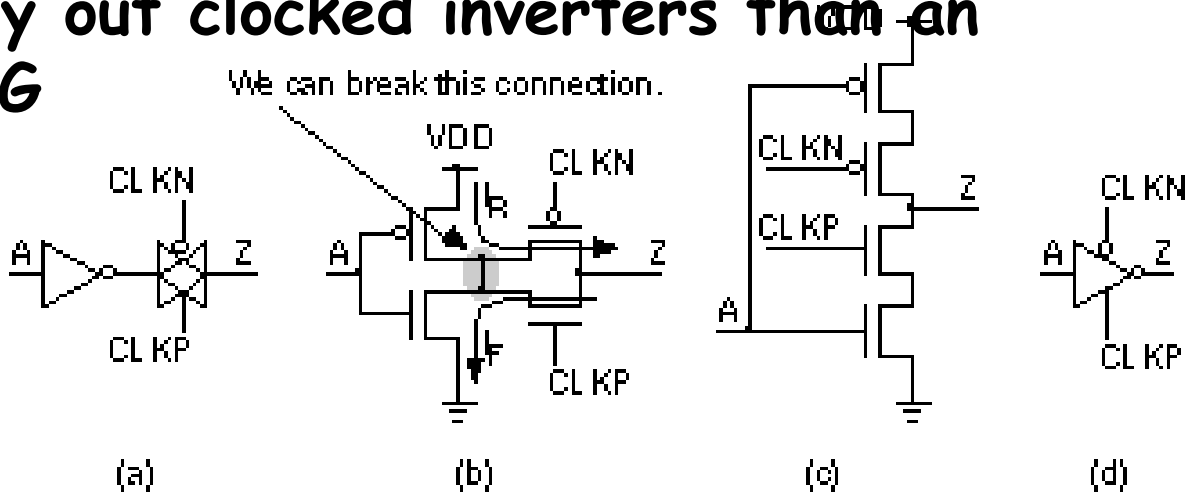
- The following design also works but requires careful <u>transistor sizing</u>:

$$\left(\frac{W}{L}\right)_1 >> \left(\frac{W}{L}\right)_2$$



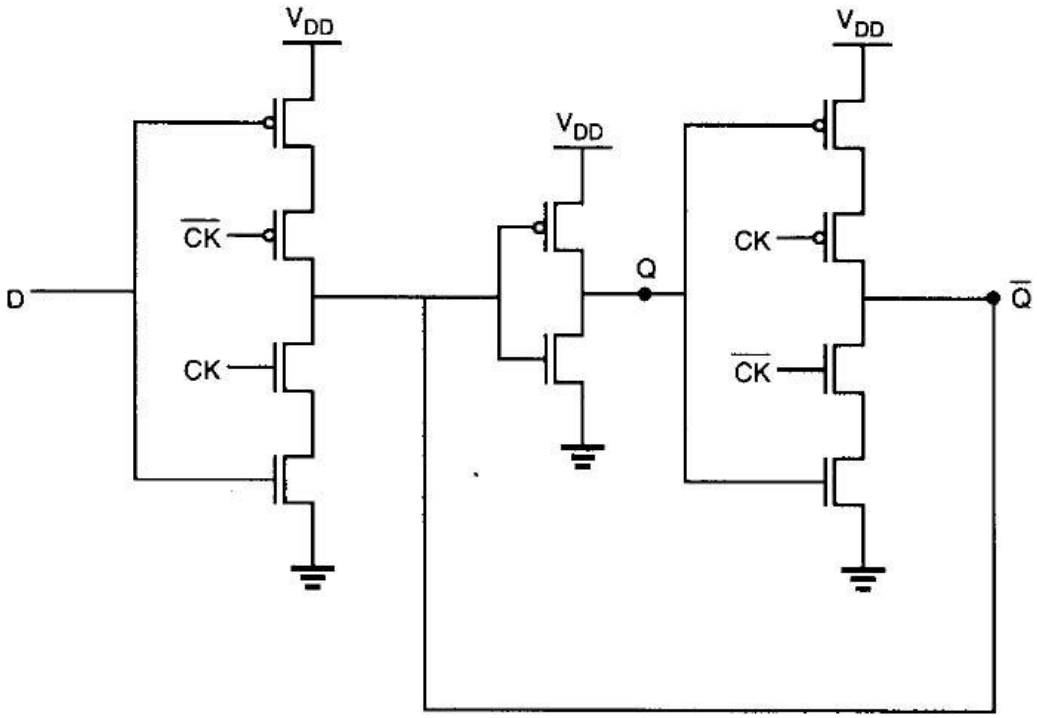
Clocked Inverter

- (a) An inverter plus transmission gate (TG)
- (b) The current flow in the inverter and TG allows us to break the connection between the transistors in the inverter
- (c) Breaking the connection forms a clocked inverter
- (d) A common symbol
- It is easier to lay out clocked inverters than an inverter plus a TG

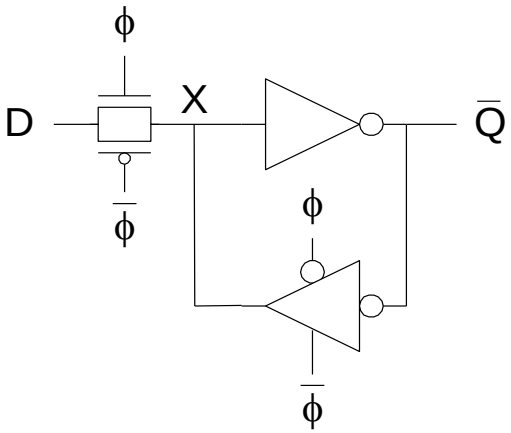


Implementation of D-Latch Using Clocked Inverters

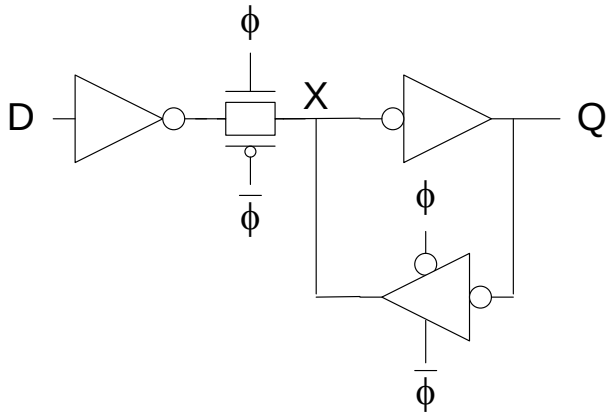
- When CK=1, the second tristate inverter is at its high-impedance state, and the output Q is following its input value D
- When CK=0, the input inverter becomes inactive, and the whole circuit, which is a two-inverter loop, preserves its state until the next clock pulse



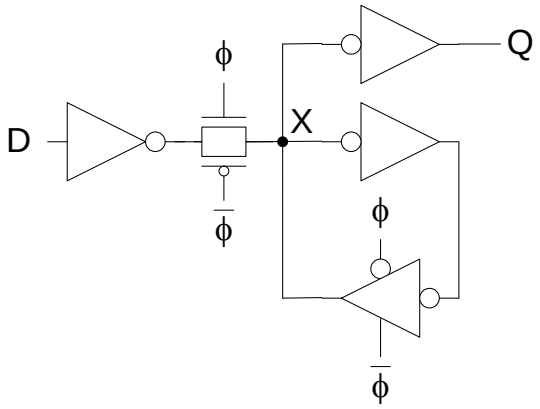
Static Latch Design



(a)



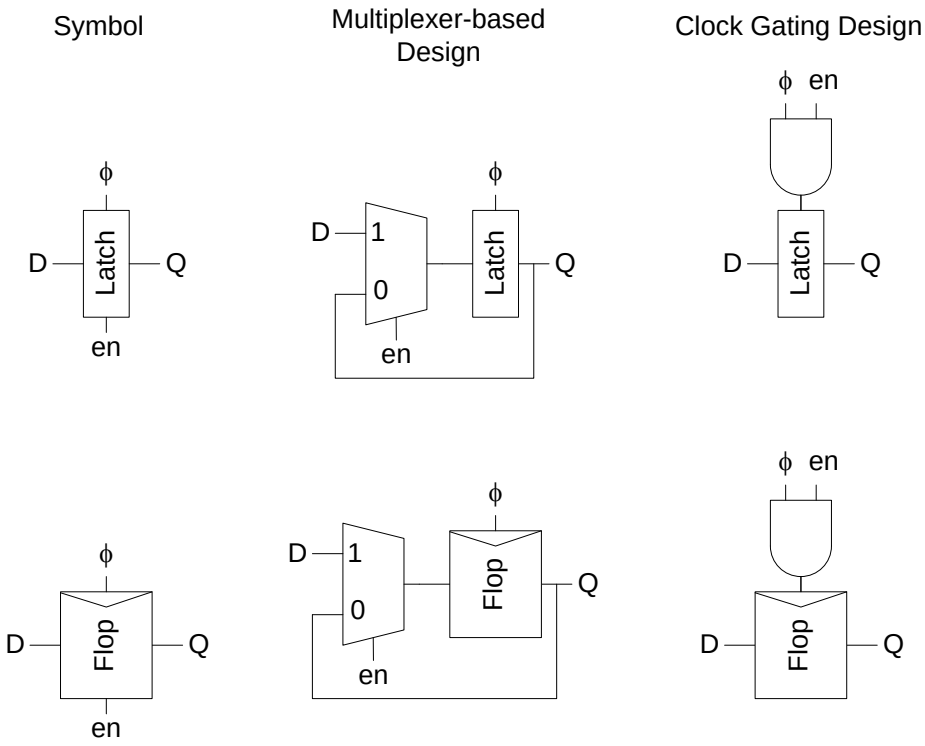
(b)



(c)

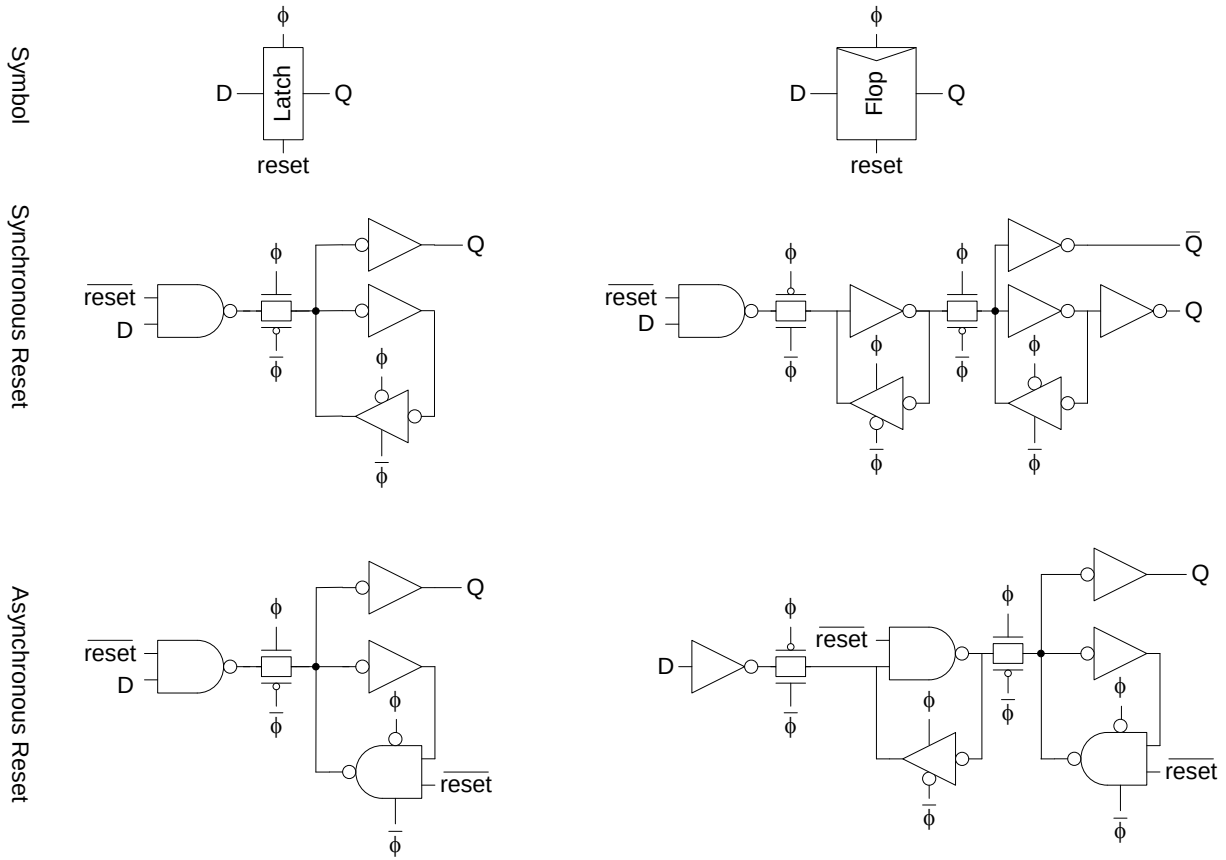
Latch with Enable

- Enable: Ignore clock when en = 0
  - Mux-based design: Increases the latch D-Q delay
  - Clock gating design: Increases the setup time with respect to *en* input. Can increase the clock skew



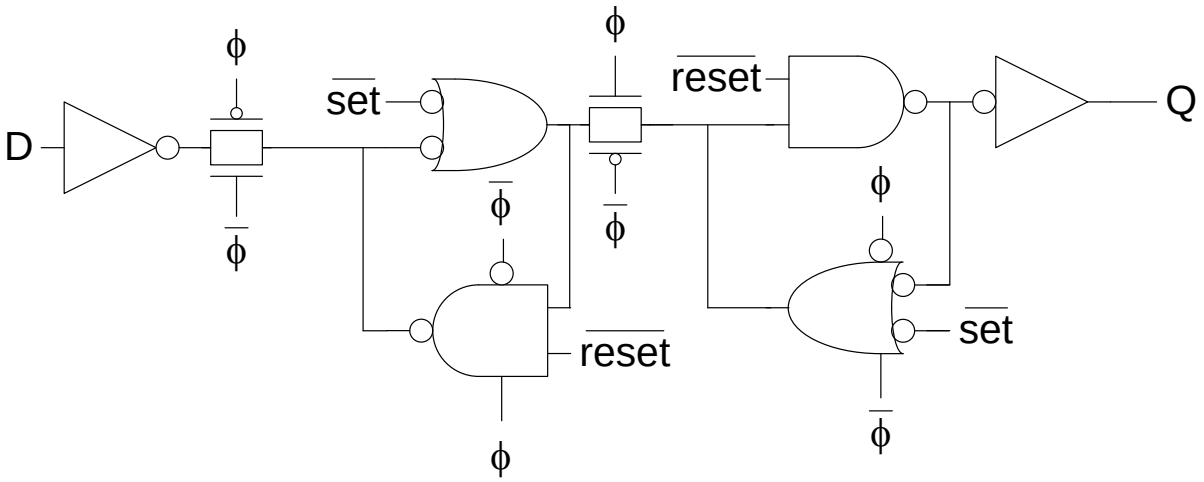
Latch and Flip Flop with Reset

- Force the output low when reset signal is asserted
- Synchronous vs. asynchronous:



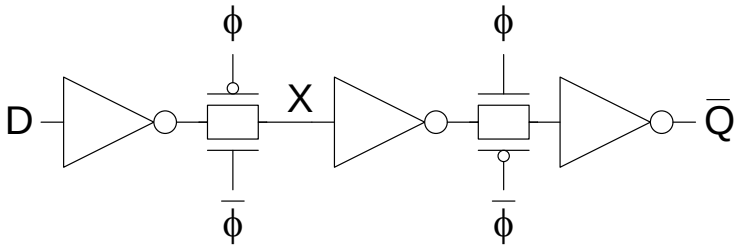
Latch with Set / Reset

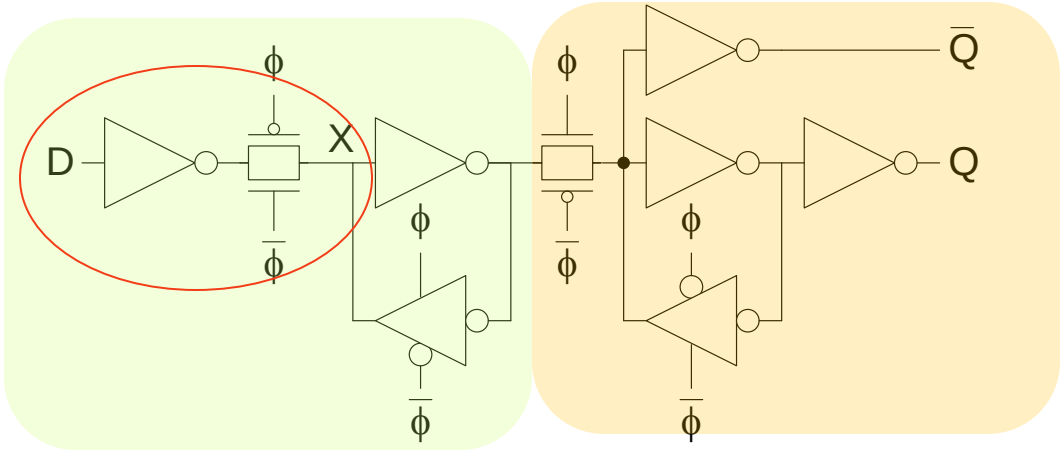
- Set forces the output high when it is asserted
- Example: Flip-flop with asynchronous set and reset



Flip-Flop Design

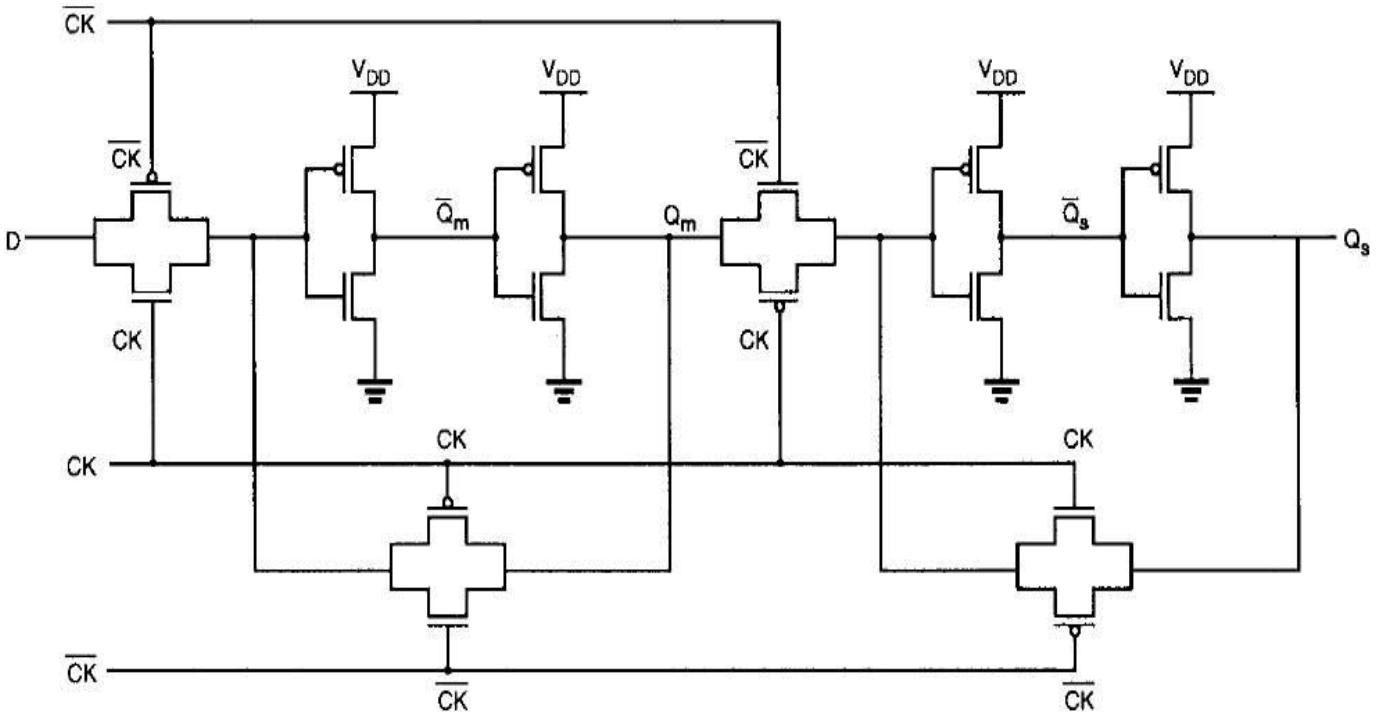
- Flip-flop is built as a pair of back-to-back latches





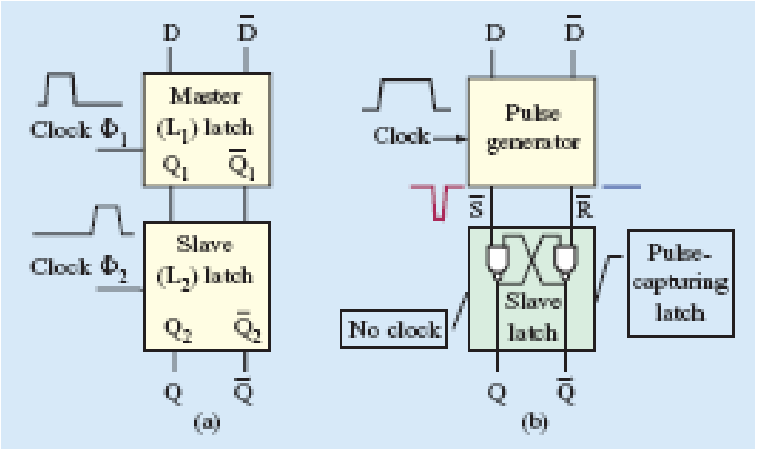
Master Latch **Slave Latch**

Negative Edge Triggered Static FF Design

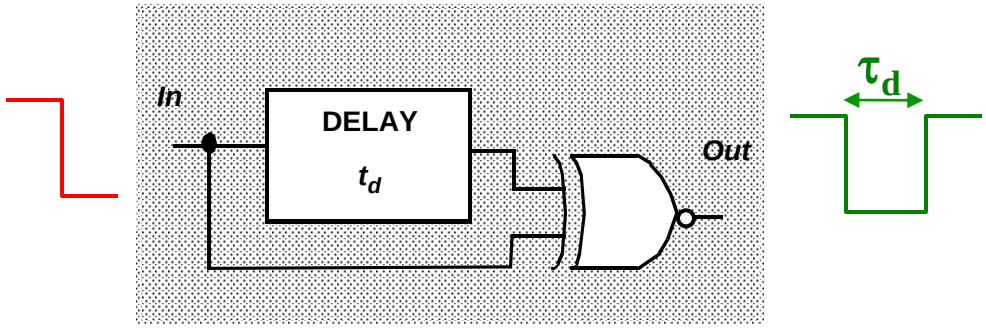


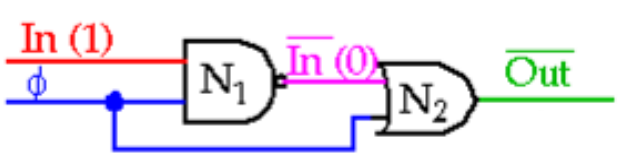
M-S D Flip-Flop vs. Edge-Triggered Flip-Flop

- The general structure of the edge-triggered flip-flop is shown in Figure (b). Notice the difference between the flip-flop and the M-S flip-flop
- An edge-triggered flip-flop consists of two stages: a *pulse generator* (PG) and a *pulse-capturing latch* (PCL)
- The PG generates a negative pulse on either ~S or ~R lines, which are normally held at logic 1 level
  - This pulse is a function of data (D) and clock signals and should be of sufficient duration to be captured in the PCL
  - The duration of the pulse can be as long as half of the clock period, or it can be as short as one inverter delay

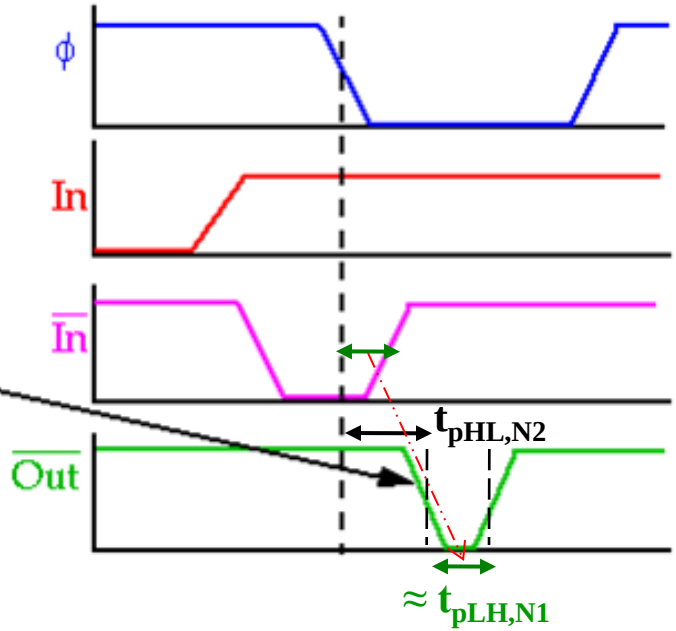


Monostable Triggers



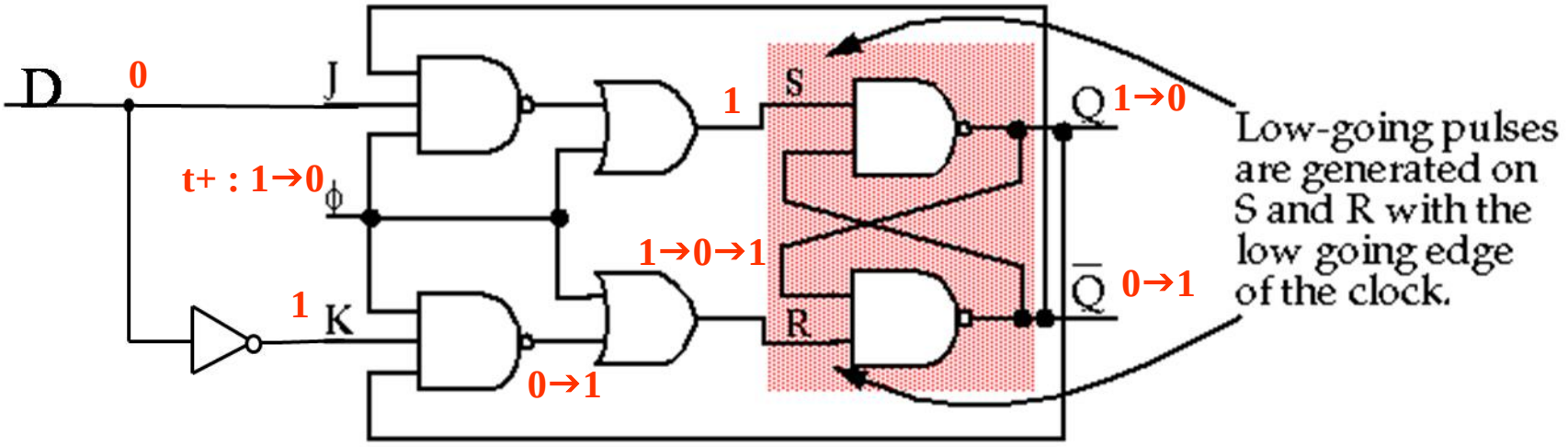


Results in a short low-going pulse at the output of N_2 with length approximately equal to the propagation delay through N_1.



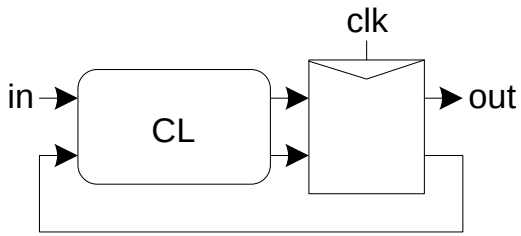
Monostable-Based (Time-Window-Based) Negative-Edge-Triggered D Flip-Flop

- **Logic transitions illustrate how the Q output is reset after the *falling edge* of the clock when D=0**

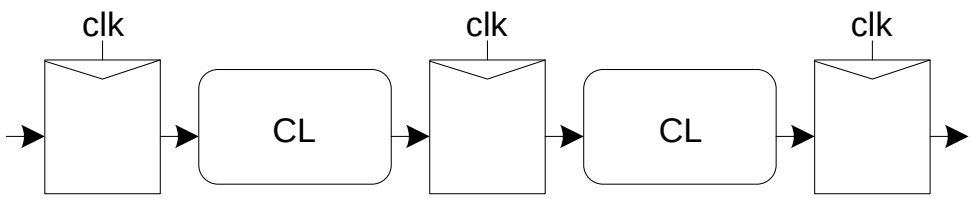


Sequencing

- Combinational logic
  - Output depends on current inputs only
- Sequential logic
  - Output depends on current and previous inputs
  - Requires separating previous, current, and future values
  - Use *states* or *tokens*
  - Ex: finite state machine, pipeline circuit



Finite State Machine



Pipeline Circuit

Sequencing (Cont.)

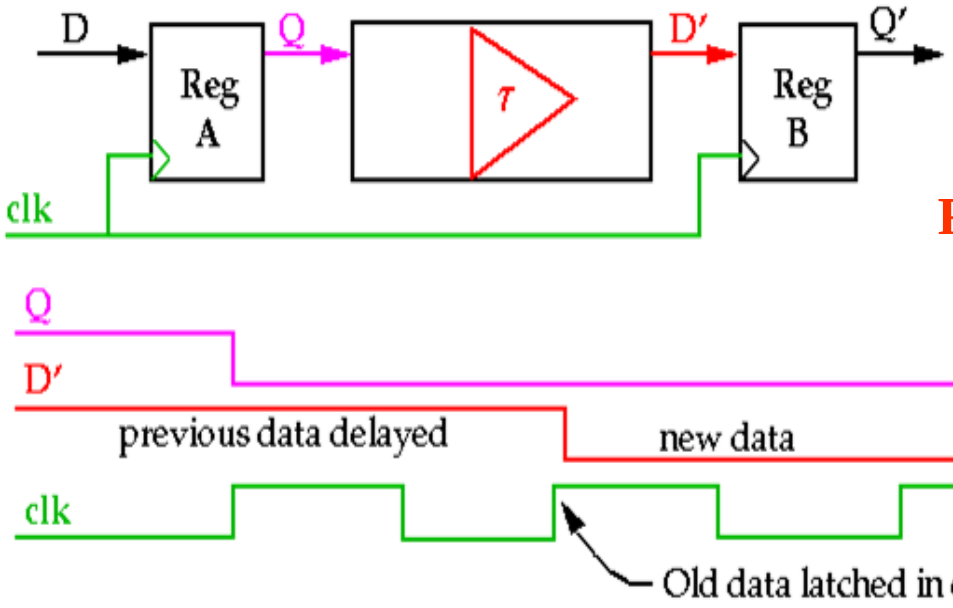
- If data (token) moved through the pipeline at constant speed, no sequencing elements would be necessary
- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before the first one reaches end of cable
  - No need for hardware to separate pulses
  - However, *wave dispersion* sets min time between pulses
- This is known as *wave pipelining* in VLSI circuits
- In most circuits, dispersion is high
  - Delay the fast data so they don't catch up with the slow ones

Sequencing Overhead

- Use flip-flops to delay fast data so they move through exactly one stage in each cycle
- This scheme inevitably adds some delay to the slow data, which makes the circuit slower than just the logic delay
  - This is called the *sequencing overhead* (sometimes called the *clocking overhead*)
  - This overhead applies to asynchronous circuits too
  - It is the unavoidable overhead of maintaining the sequencing rule

Clock Race Conditions

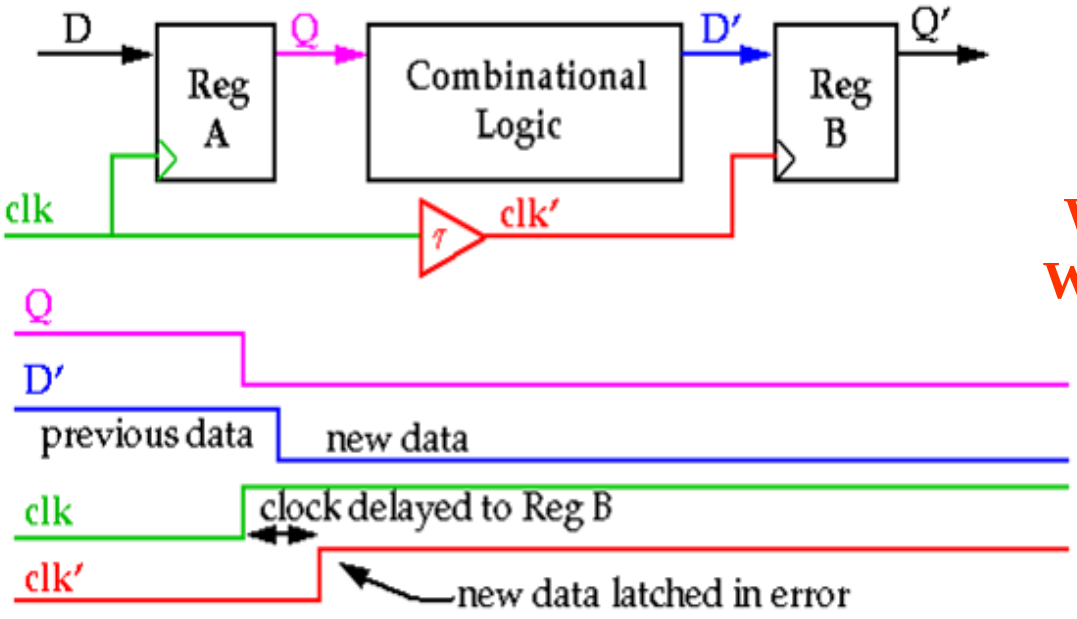
- Delays in the combinational logic that are larger than the clock cycle time (setup-time violation)
  - Data arrives late at register B, old data retained instead of latching new data



Read-before-Write Hazard

Clock Race (Cont.)

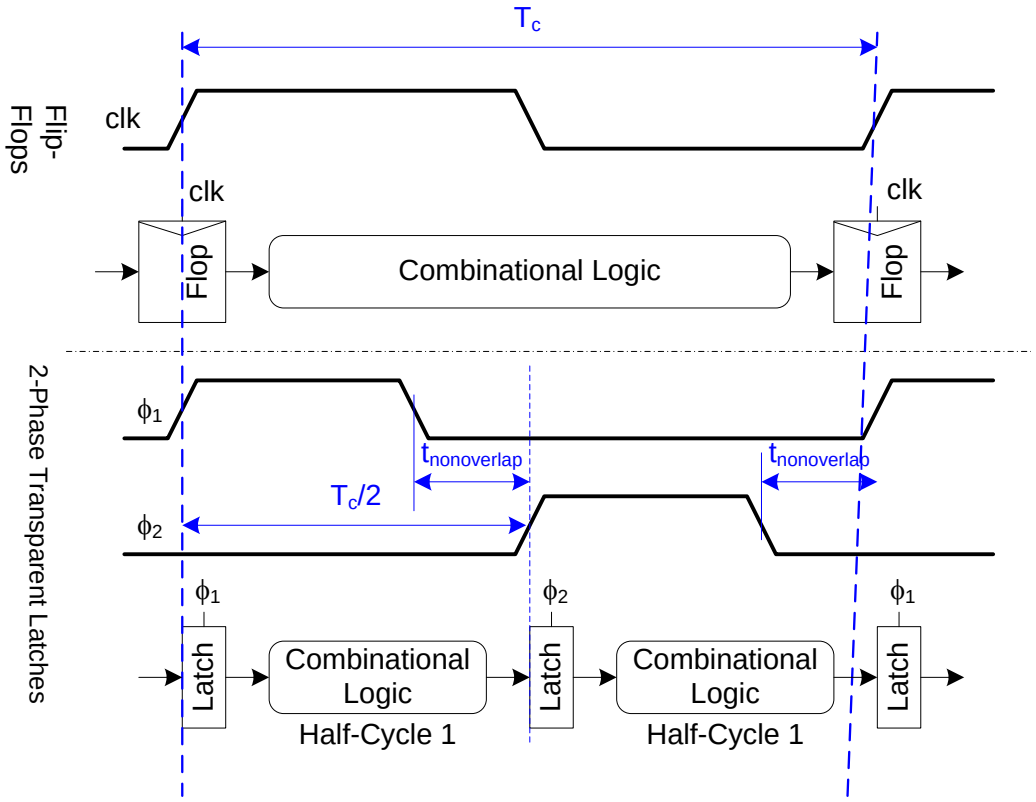
- Clock race occurs when the data input to the register does not obey the setup and hold-time constraints
  - Delays in the clock line to register B (hold-time violation)
    - New data stored instead of previous data



Write-after-Write Hazard

Sequencing Methods

- Flip-flops
- 2-Phase Latches



How to Meet Setup and Hold Time Constraints

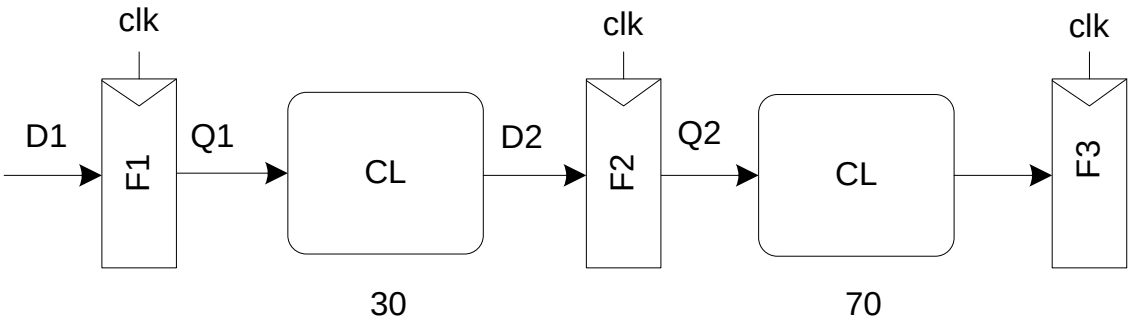
- If setup times are violated, reduce clock speed
- If hold times are violated, chip fails at any speed
- Working chips are most important
- Clock skew analysis is crucial and tools may not help
- An easy way to guarantee hold times is to use two-phase latches with a large nonoverlap time between pseudo-complementary clocks ϕ_1, ϕ_2

Time Borrowing

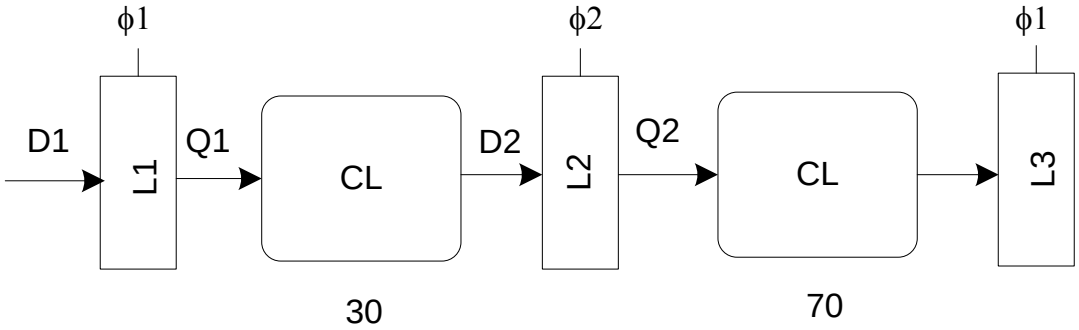
- In a flip-flop-based design:
  - Data launches on some rising edge of the clock
  - It must set up before the next rising edge
  - If it arrives late, the design will fail
  - If it arrives early, clock cycle time will be wasted
  - Flip-flops have hard edges
- In a latch-based design:
  - Data can pass through the latch while it is transparent
  - Long stage of combinational logic can borrow time from the next stage as long as each loop in the design completes in no more than one cycle

Time Borrowing

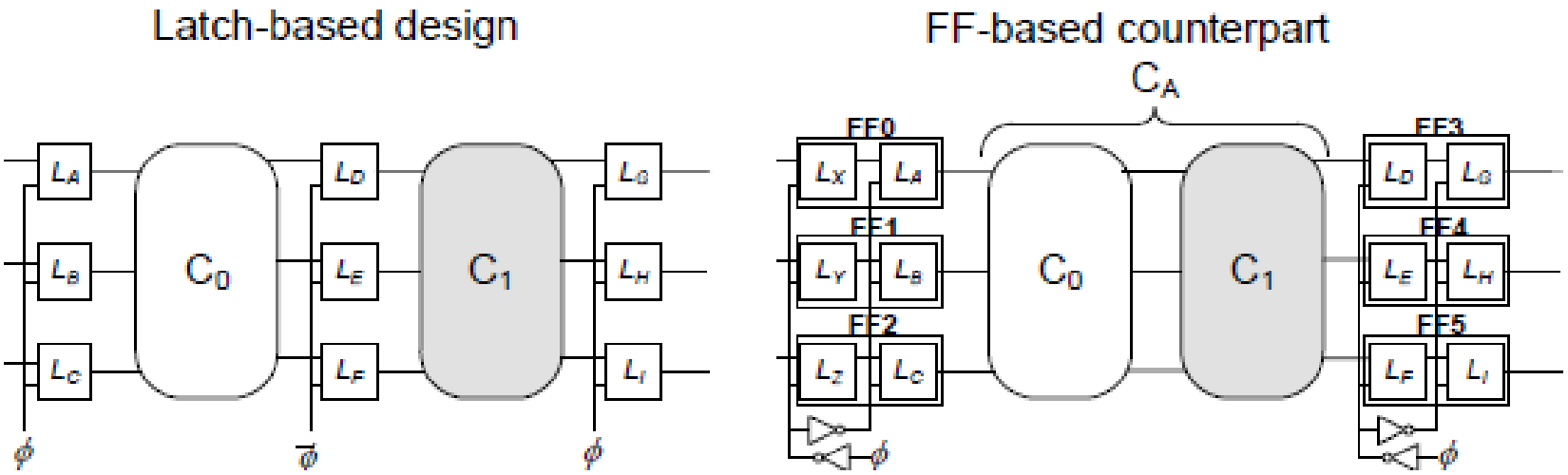
- In Flip-Flop based linear pipeline designs, clock is set by the slowest stage



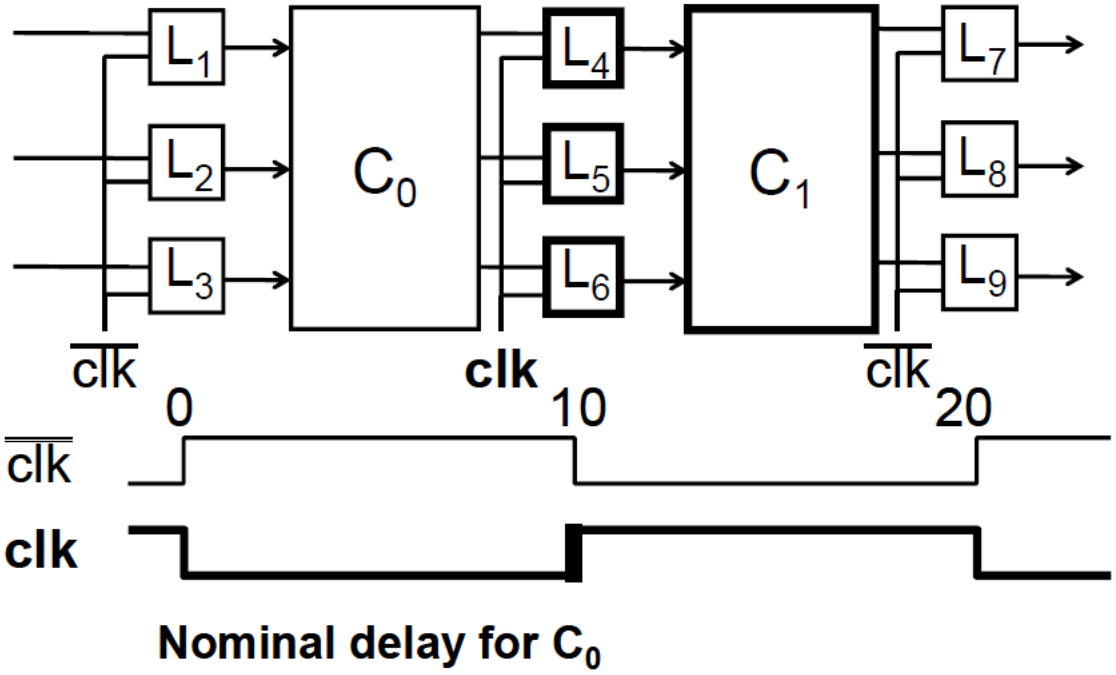
- In Latch-based linear pipeline designs, clock is set by the average of two (or possibly more) consecutive stages



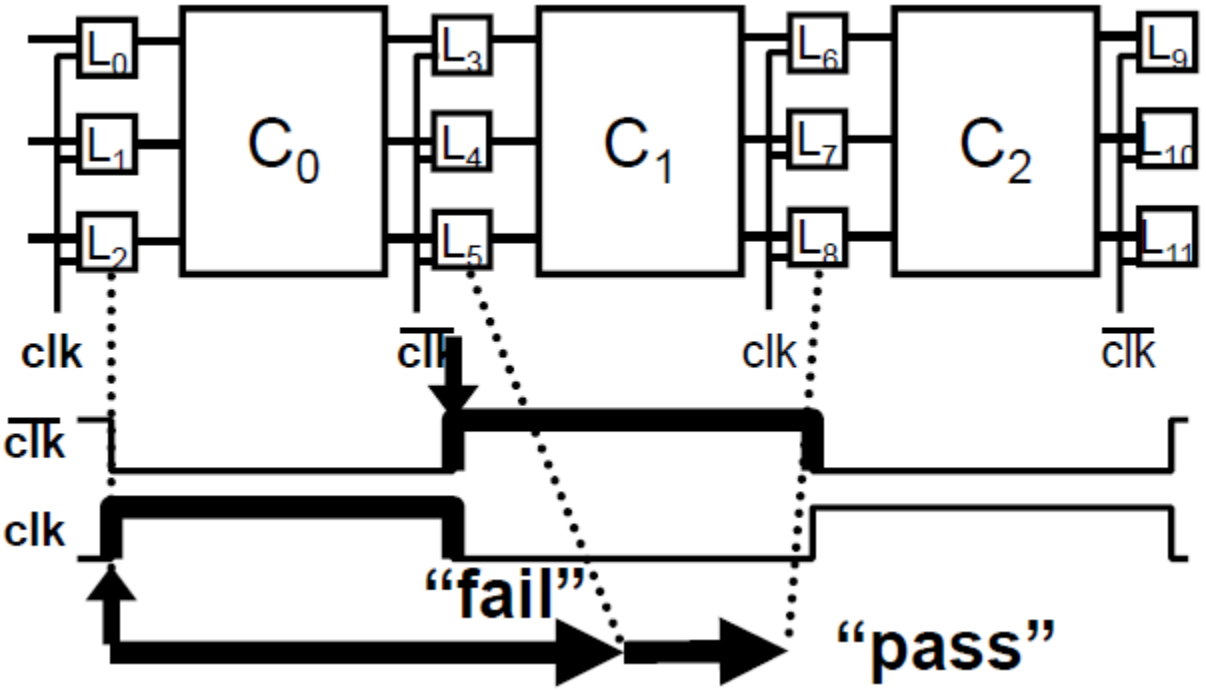
Time Borrowing (Cont.)



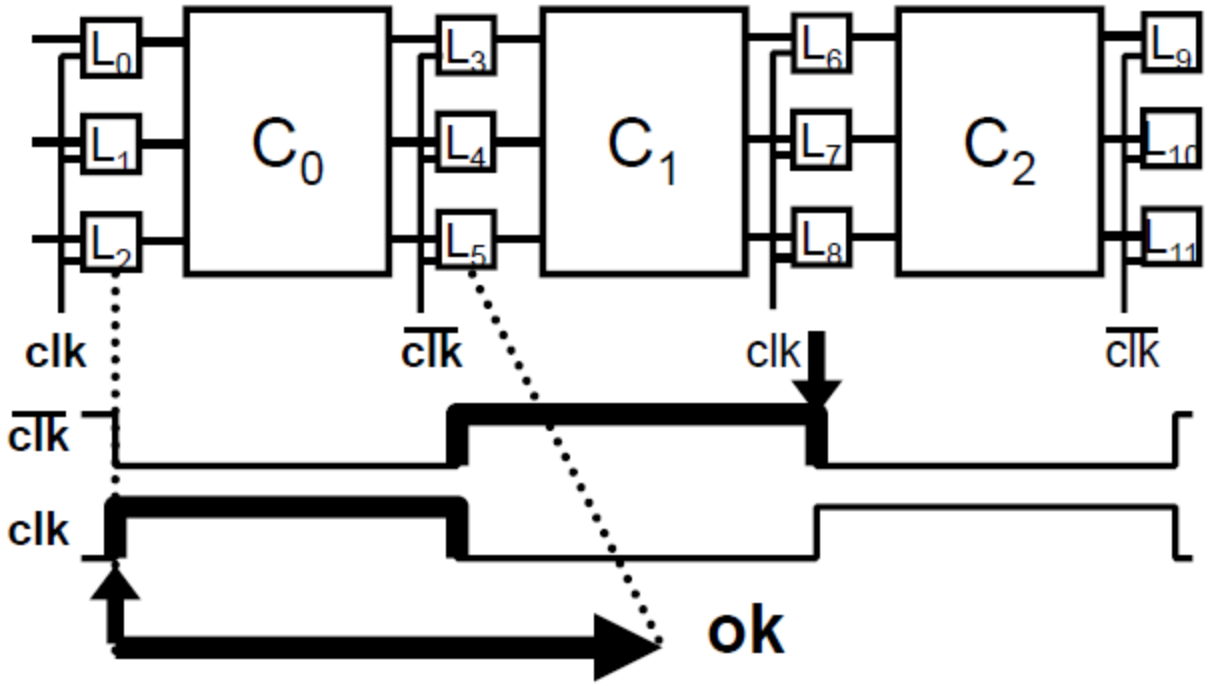
Time Borrowing (Cont.)



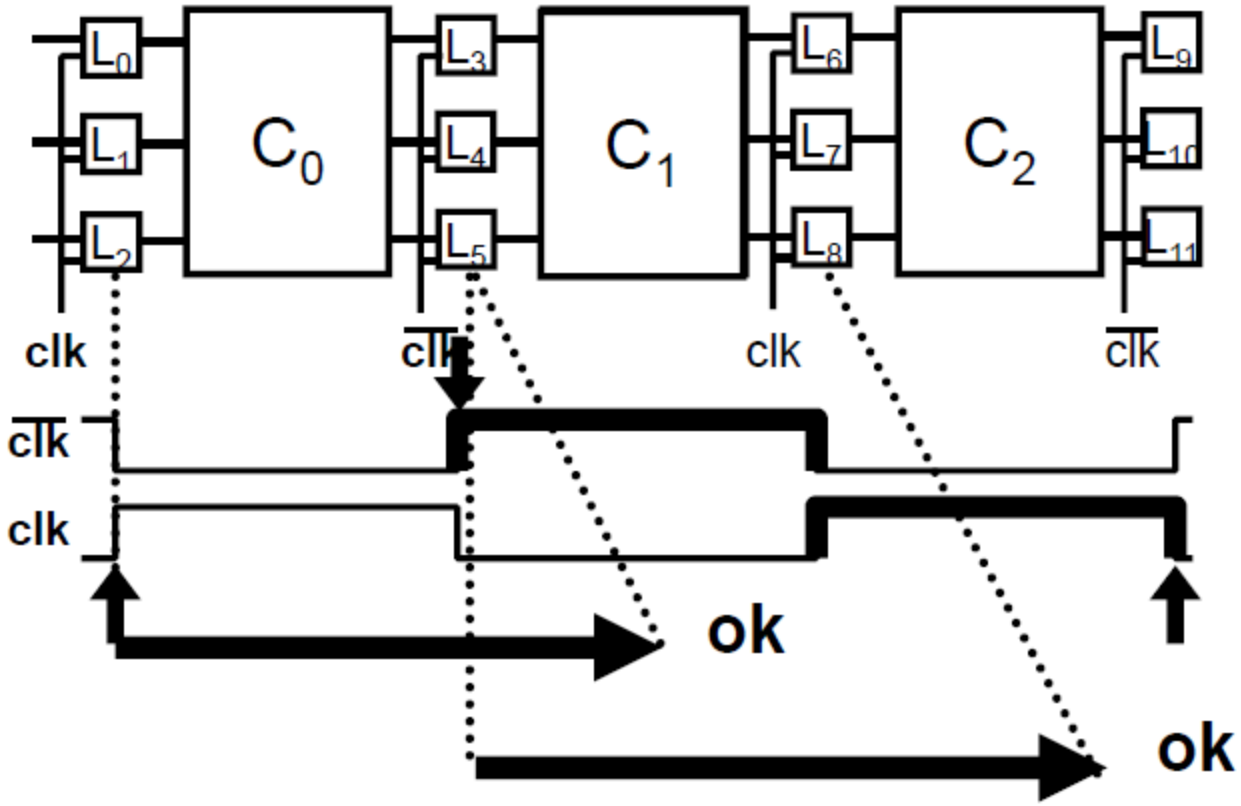
Time Borrowing (Cont.)



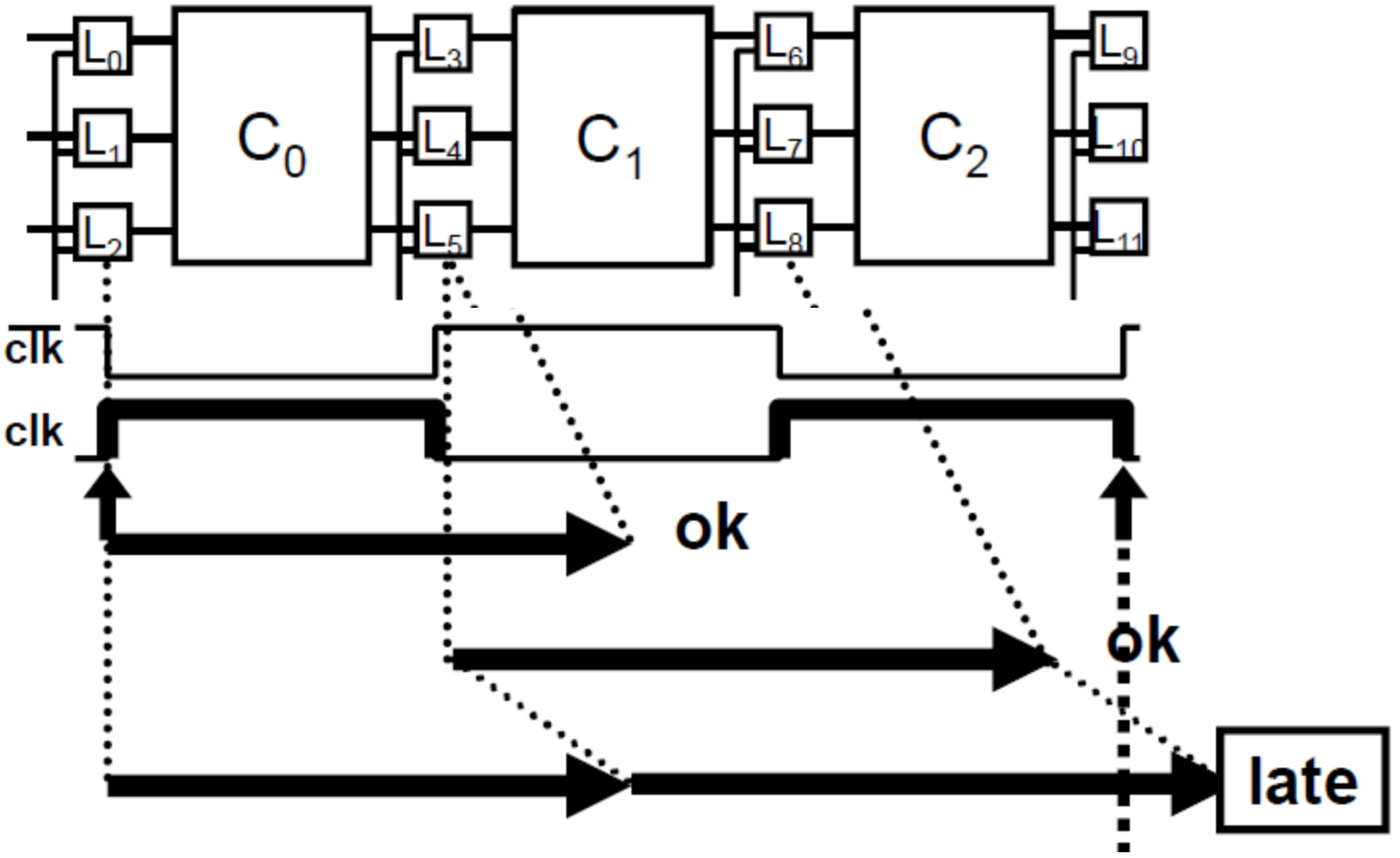
Time Borrowing (Cont.)



Time Borrowing (Cont.)



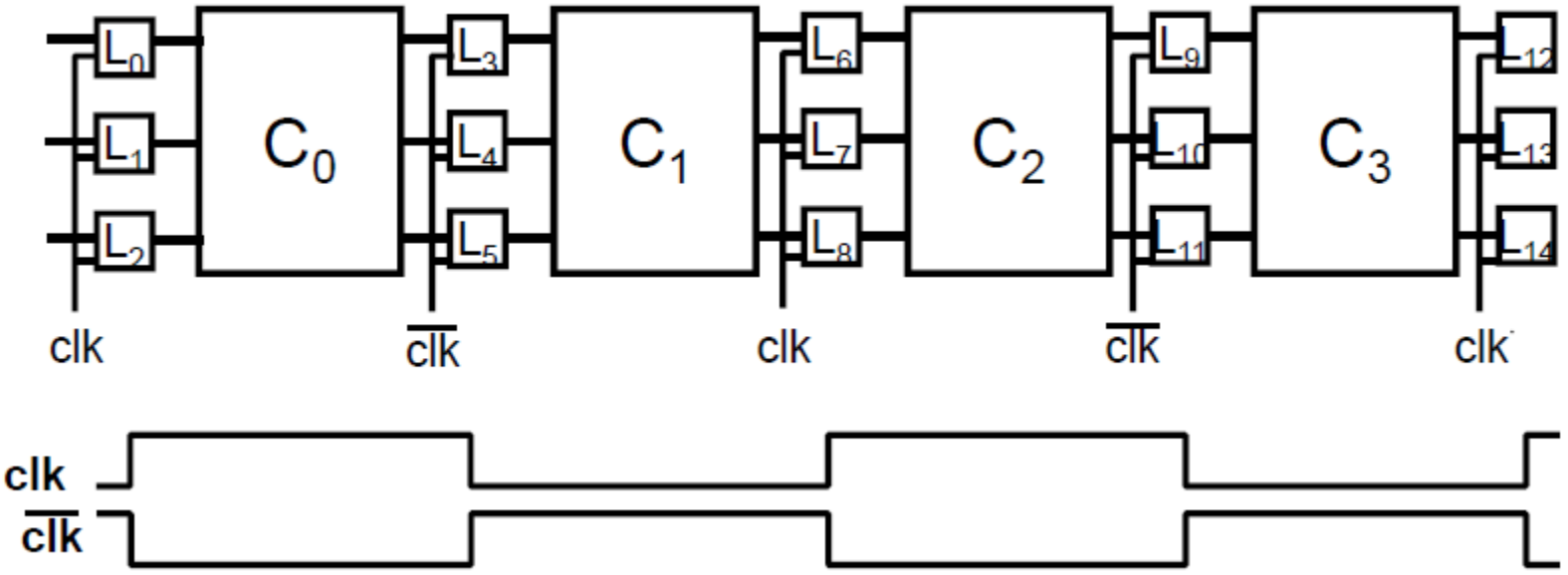
Time Borrowing (Cont.)



Time Borrowing (Cont.)

- m consecutive block
  - Nominal delay: mT/2
  - Maximum allowable: (m+1)T/2

Time Borrowing (Cont.)



Time Borrowing (Cont.)