EE141-Spring 2012 Digital Integrated Circuits

Lecture 20
Sequential Circuits

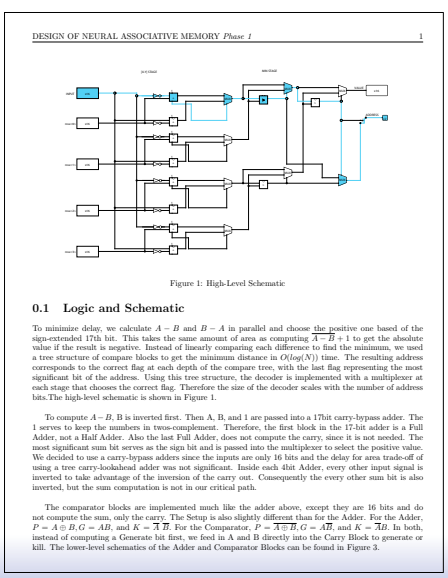
Administrativia

- Phase 2 due next We!
- Phase 1 graded.
- Do not forget homeworks.

Phase 1

- Many groups converged to similar solution
  - Parallel carry-select adders for |a-b|
  - Comparator tree
  - Simple decoder
  - Leading to worst-case delay of ~ 1 nsec
- Hard to do better – however
  - Not very regular
  - Hard to scale to larger arrays

Example of good report



DESIGN OF NEURAL ASSOCIATIVE MEMORY *Phase 1* 1

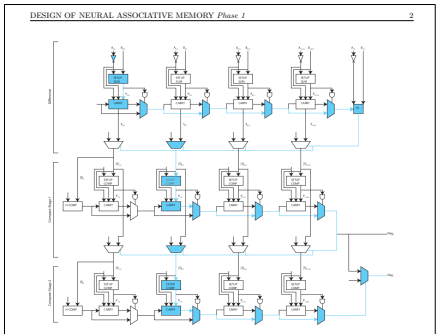
Figure 1: High-Level Schematic

0.1 Logic and Schematic

To minimize delay, we calculate $A - B$ and $B - A$ in parallel and choose the positive one based of the sign-extended 17th bit. This takes the same amount of area as computing $\overline{A - B} + 1$ to get the absolute value if the result is negative. Instead of linearly comparing each difference to find the minimum, we used a tree structure of compare blocks to get the minimum distance in $O(log(N))$ time. The resulting address corresponds to the correct flag at each depth of the compare tree, with the last flag representing the most significant bit of the address. Using this tree structure, the decoder is implemented with a multiplexer at each stage that chooses the correct flag. Therefore the size of the decoder scales with the number of address bits.The high-level schematic is shown in Figure 1.

To compute $A - B$, B is inverted first. Then A, B, and 1 are passed into a 17bit carry-bypass adder. The 1 serves to keep the numbers in twos-complement. Therefore, the first block in the 17-bit adder is a Full Adder, not a Half Adder. Also the last Full Adder, does not compute the carry, since it is not needed. The most significant sum bit serves as the sign bit and is passed into the multiplexer to select the positive value. We decided to use a carry-bypass adders since the inputs are only 16 bits and the delay for area trade-off of using a tree carry-lookahead adder was not significant. Inside each 4bit Adder, every other input signal is inverted to take advantage of the inversion of the carry out. Consequently the every other sum bit is also inverted, but the sum computation is not in our critical path.

The comparator blocks are implemented much like the adder above, except they are 16 bits and do not compute the sum, only the carry. The Setup is also slightly different than for the Adder. For the Adder, $P = A \oplus B, G = AB$, and $K = \overline{A}\,\overline{B}$. For the Comparator, $P = \overline{A \oplus B}, G = A\overline{B}$, and $K = \overline{A}B$. In both, instead of computing a Generate bit first, we feed in A and B directly into the Carry Block to generate or kill. The lower-level schematics of the Adder and Comparator Blocks can be found in Figure 3.



DESIGN OF NEURAL ASSOCIATIVE MEMORY *Phase 1* 2

Figure 2: Critical Path

0.2 Critical Path Flow Diagram

The critical path is highlighted in the schematic in Figure 1. A lower-level view of the critical path is show in Figure 2. Figure 3 shows the gate-level critical path.

0.3 Delay and Sizing

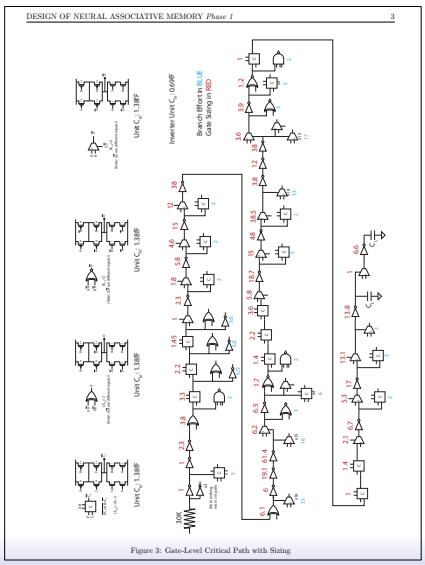
The sizing of each gate can be found in Figure 3, along with the branching effort of each branch. To size the gates in our critical path, we calculated the total path effort and found the optimal number of stages to minimize delay. Then we found how much we should multiply each minimum-sized gate to achieve the optimal effective fanout per stage. Since the output address is two bits, both need to drive the load capacitance. We split the critical path into two paths at C_L, shown in Figure 3, and found the path efforts for each.

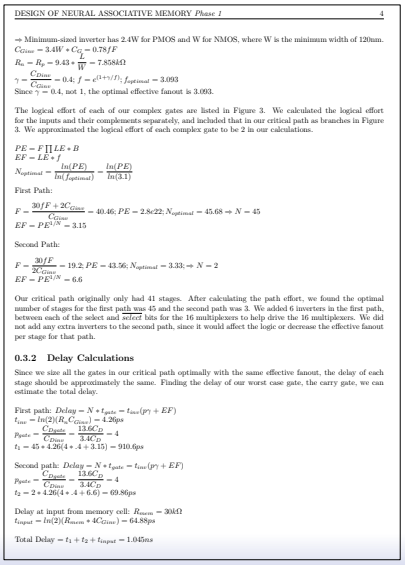
0.3.1 Path Effort Calculations

MOSFET Parameters: $W = 120um, L = 100um, C_G = 1.91fF/m, C_D = 0.77fF/m, R_{sqn} = 9.43k\Omega/square$, and$R_{sp} = 22.32k\Omega/square$.

Minimum-sized inverter with equal pull up and pull down strength: $\frac{R_{sp}}{R_{sqn}} = \frac{22.32}{9.43} = 2.4$

Example of good report





DESIGN OF NEURAL ASSOCIATIVE MEMORY *Phase 1* 4

⇒ Minimum-sized inverter has 2.4W for PMOS and W for NMOS, where W is the minimum width of 120nm.
$C_{Gline} = 3.4W * C_G = 0.78fF$
$R_n = R_p = 9.43 * \frac{L}{W} = 7.858k\Omega$
$\gamma = \frac{C_{Dline}}{C_{Gline}} = 0.4$; $f = e^{(1+\gamma/f)}$; $f_{optimal} = 3.093$
Since $\gamma = 0.4$, not 1, the optimal effective fanout is 3.093.

The logical effort of each of our complex gates are listed in Figure 3. We calculated the logical effort for the inputs and their complements separately, and included that in our critical path as branches in Figure 3. We approximated the logical effort of each complex gate to be 2 in our calculations.

$PE = F\prod LE * B$
$EF = LE * f$
$N_{optimal} = \frac{ln(PE)}{ln(f_{optimal})} = \frac{ln(PE)}{ln(3.1)}$

First Path:

$F = \frac{30fF + 2C_{Gline}}{C_{Gline}} = 40.46$; $PE = 2.8e22$; $N_{optimal} = 45.68 \Rightarrow N = 45$
$EF = PE^{1/N} = 3.15$

Second Path:

$F = \frac{30fF}{2C_{Gline}} = 19.2$; $PE = 43.56$; $N_{optimal} = 3.33$; $\Rightarrow N = 2$
$EF = PE^{1/N} = 6.6$

Our critical path originally only had 41 stages. After calculating the path effort, we found the optimal number of stages for the first path was 45 and the second path was 3. We added 6 inverters in the first path, between each of the select and $\overline{select}$ bits for the 16 multiplexers to help drive the 16 multiplexers. We did not add any extra inverters to the second path, since it would affect the logic or decrease the effective fanout per stage for that path.

0.3.2 Delay Calculations

Since we size all the gates in our critical path optimally with the same effective fanout, the delay of each stage should be approximately the same. Finding the delay of our worst case gate, the carry gate, we can estimate the total delay.

First path: $Delay = N * t_{gate} = t_{inv}(p\gamma + EF)$
$t_{inv} = ln(2)(R_n C_{Gline}) = 4.26ps$
$p_{gate} = \frac{C_{Dgate}}{C_{Dinv}} = \frac{13.6C_D}{3.4C_D} = 4$
$t_1 = 45 * 4.26(4 * .4 + 3.15) = 910.6ps$

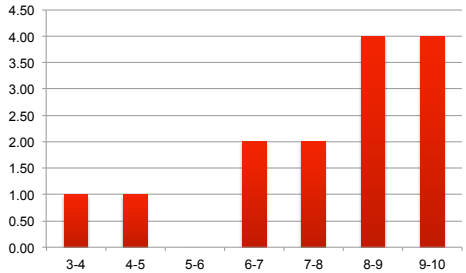
Second path: $Delay = N * t_{gate} = t_{inv}(p\gamma + EF)$
$p_{gate} = \frac{C_{Dgate}}{C_{Dinv}} = \frac{13.6C_D}{3.4C_D} = 4$
$t_2 = 2 * 4.26(4 * .4 + 6.6) = 69.86ps$

Delay at input from memory cell: $R_{mem} = 30k\Omega$
$t_{input} = ln(2)(R_{mem} * 4C_{Gline}) = 64.88ps$

Total Delay $= t_1 + t_2 + t_{input} = 1.045ns$

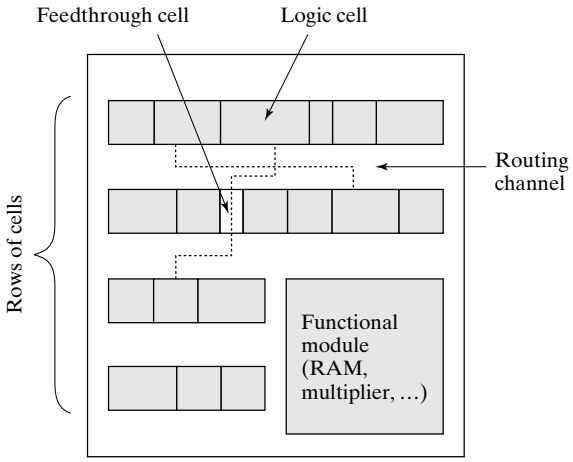
Project Phase 1 (1/3 of the grade)

- Average: 7.93
- Median: 8.38
- StdDev: 1.86

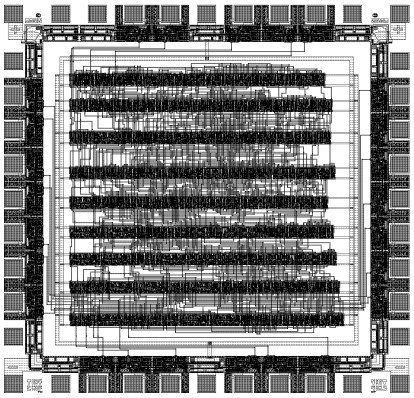


CMOS Layout

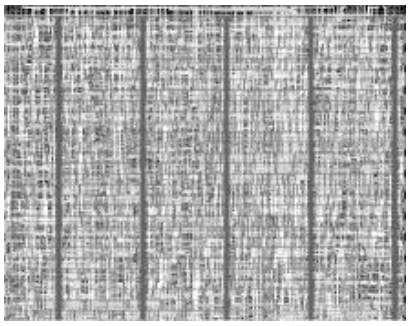
Standard Cell Methodology



Standard Cells – Then and Now

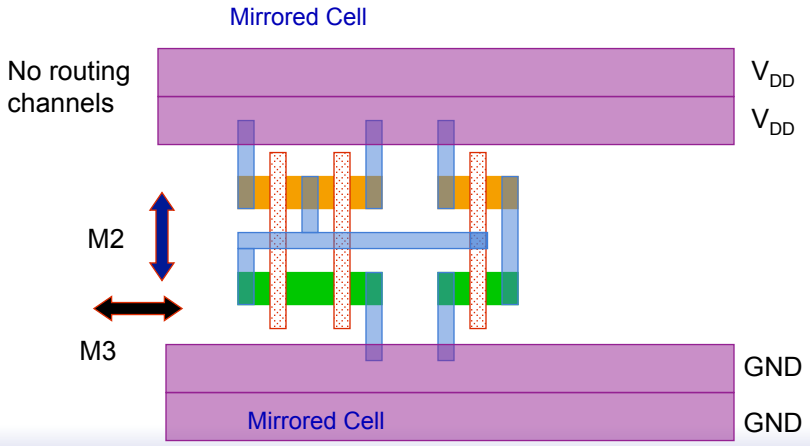


(a)



(b)

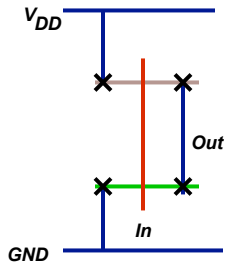
Standard Cell Layout Methodology



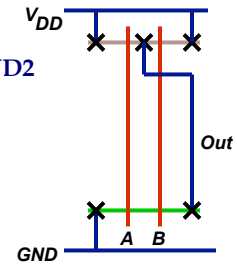
Stick Diagrams

Contains no dimensions
Represents relative positions of transistors

Inverter

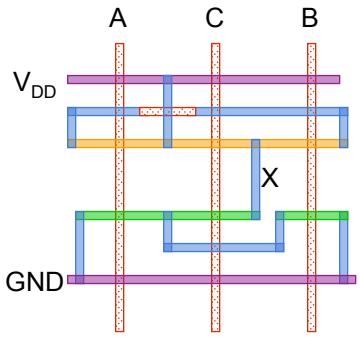


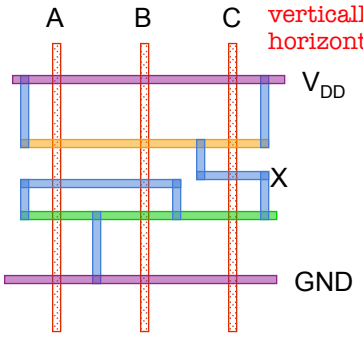
NAND2



Two Versions of $\overline{C \cdot (A + B)}$

different direction in different mental:
vertically metal one
horizontally metal two



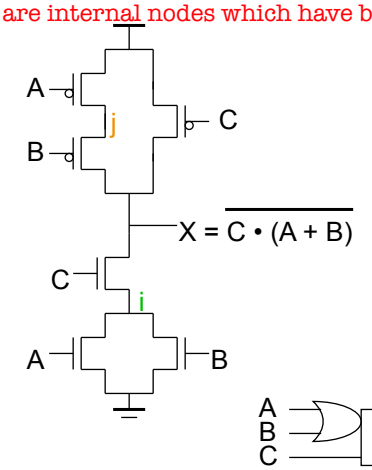


The right version is better than the left version. For the same function, different order of these variants may have different layouts. Some layouts maybe better than other and have higher density.

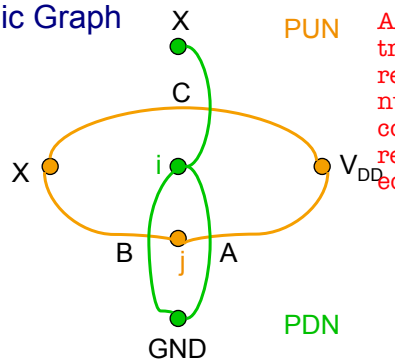
destination: find an ordering of our input transistors that allows us to have a continuous diffusion strip.

Logic Graphs

i and j are internal nodes which have been labeled



Logic Graph



An edge will represent a transistor. An node will represent a place where a number of circuits branches come together. An orange edge represents a PMOS. A green edge represent a NMOS.

Consistent Euler Path

can visit the same vertex many times but can't visit the same path more than once.

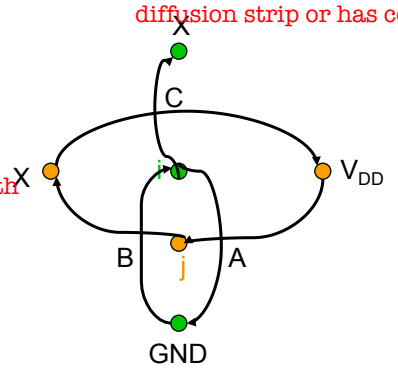
The order of edge of Euler Path means the layout has less diffusion strip or has continuous diffusion strip.

A B C

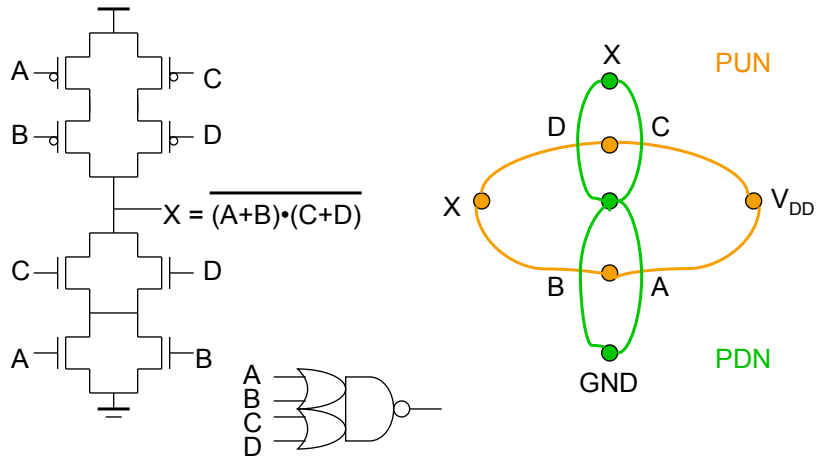
Has PDN and PUN

B C A not consistent Euler Path

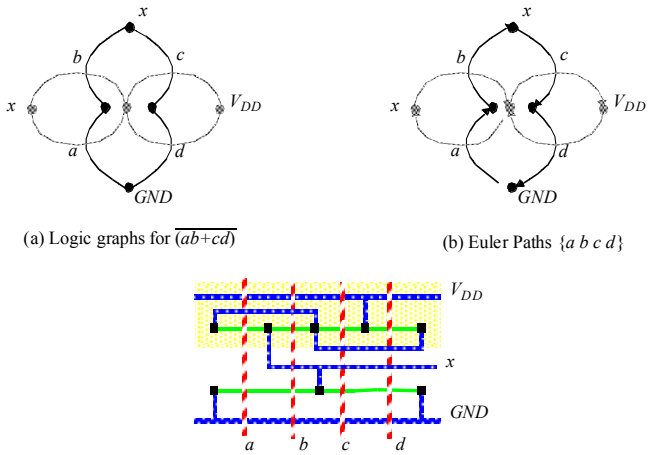
Has PUN, but no PDN



OAI22 Logic Graph



Example: x = ab+cd



(a) Logic graphs for $\overline{(ab+cd)}$

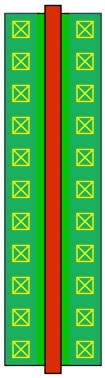
(b) Euler Paths {a b c d}

(c) stick diagram for ordering {a b c d}

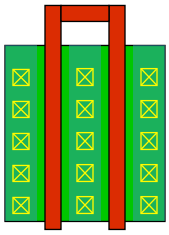
Multi-Fingered Transistors

One finger

resistance is too big.



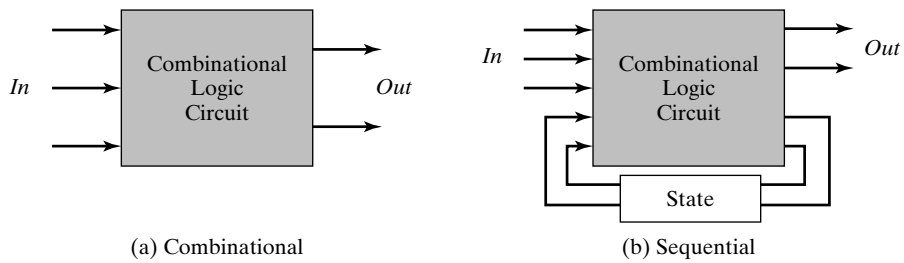
Two fingers (folded)



Less diffusion capacitance

Sequential Logic

Combinational vs. Sequential Logic



(a) Combinational (b) Sequential

Output = *f*(*In*) **Output = *f*(*In, Previous In*)**

Why Sequencing?

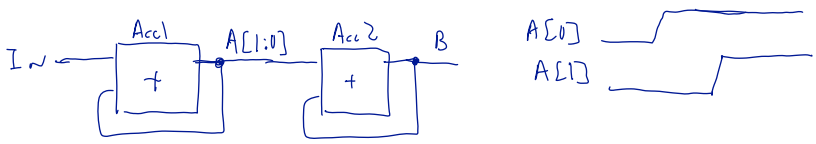
Two key (related) reasons that we need sequencing:

(1) Need to know when we are finished with a job

Why Sequential Logic?

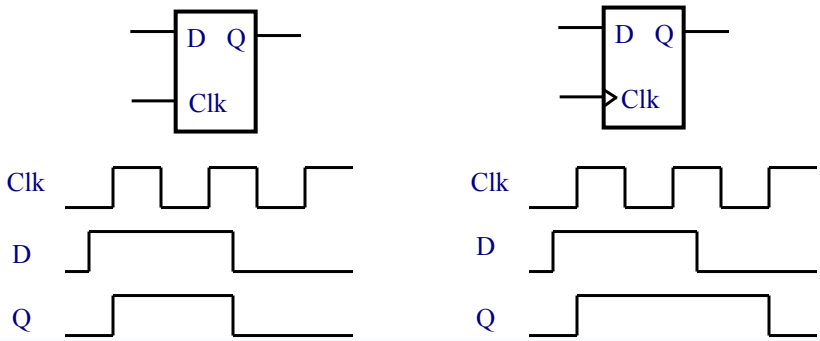
Two key (related) reasons that we need sequencing:

(2) Need to order events (aligning fast and slow)

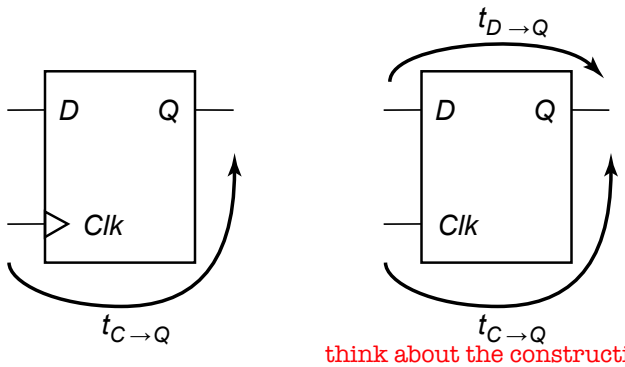


Latch versus Register (Flip-flop)

- Latch: level-sensitive
  clock is low - hold mode
  clock is high - transparent
- Register: edge-triggered
  stores data when
  clock rises



Characterizing Timing



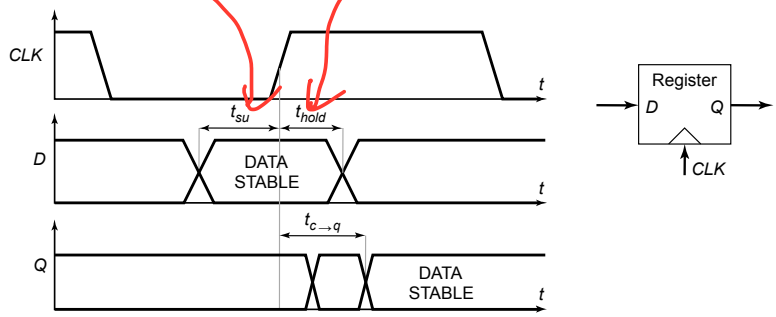
Register

think about the construction of register: two latches.

Latch

Timing Definitions – Set-up and Hold Times

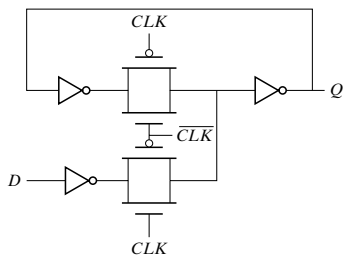
set-up time directly add up to delay, so it's important to make it small.



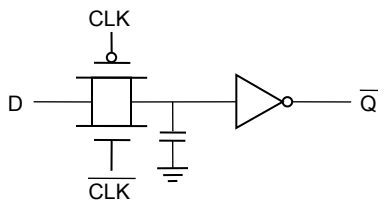
Storage Mechanisms

Register cares about performance,
while SRAM cares about the density.

Static

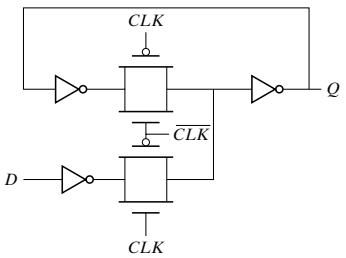


Dynamic



Writing into a Static Latch

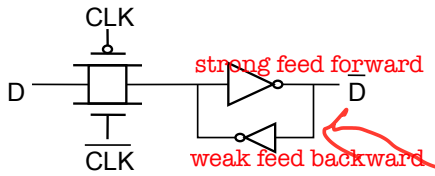
Use the clock as a decoupling signal,
that distinguishes between the transparent and opaque states



like a 2-to-1 MUX

Converting into a MUX

general approach to storage signal.

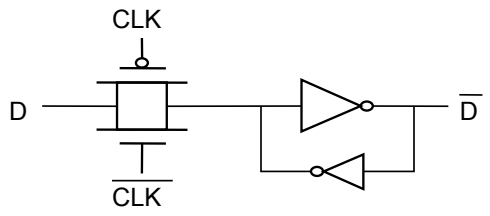


New signal will fight with old signal which will cause delay. And only strong signal will win.So to storage the new signal, we have to make an asymmetrical construction.

Forcing the state
(can implement as NMOS-only)

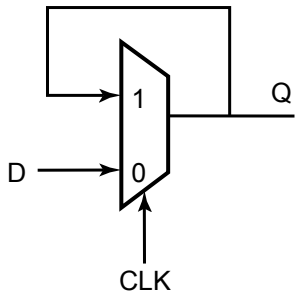
force approach to storage signal

Pseudo-Static Latch



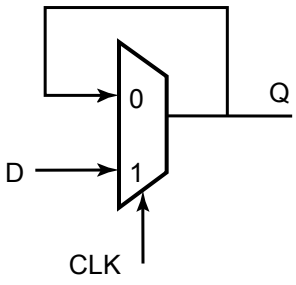
Mux-Based Latches

Negative latch
(transparent when CLK= 0)



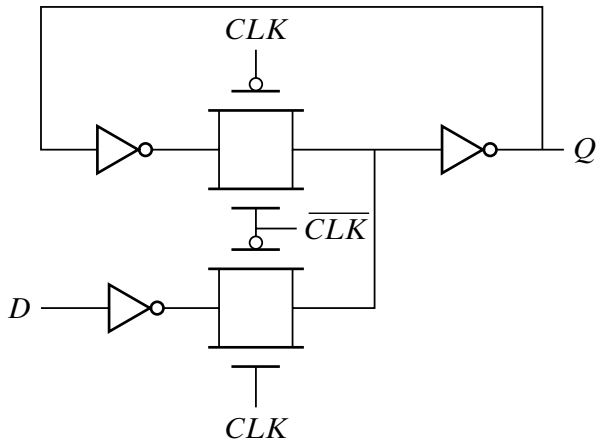
$$Q = Clk \cdot Q + \overline{Clk} \cdot In$$

Positive latch
(transparent when CLK= 1)

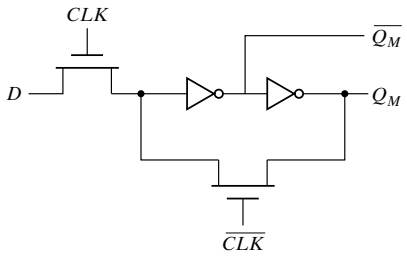


$$Q = \overline{Clk} \cdot Q + Clk \cdot In$$

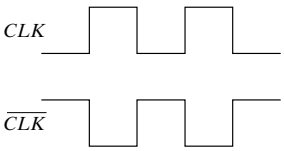
Mux-Based Latch



Mux-Based Latch



(a) Schematic diagram



(b) Non overlapping clocks

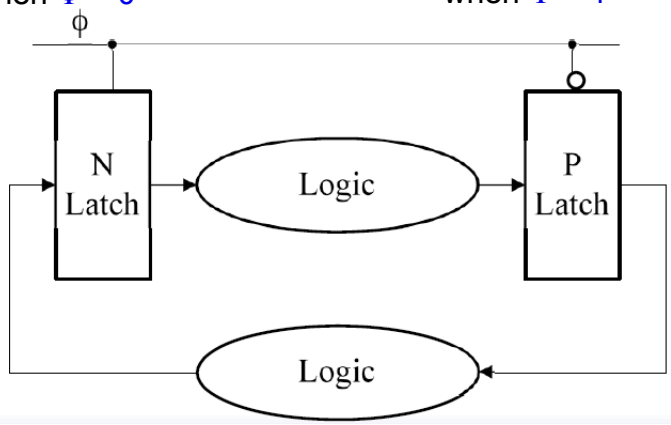
NMOS only

Non-overlapping clocks

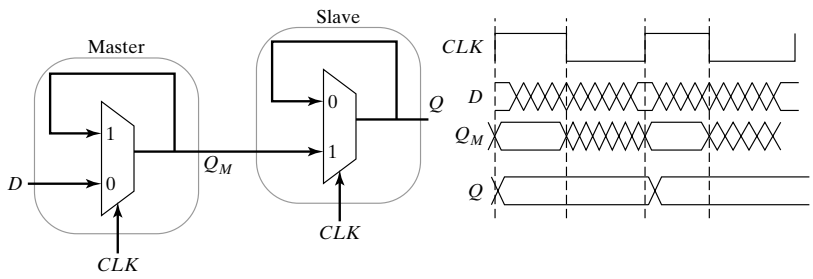
The race problem

Latch-Based Design

- N latch is transparent when $\Phi = 0$
- P latch is transparent when $\Phi = 1$



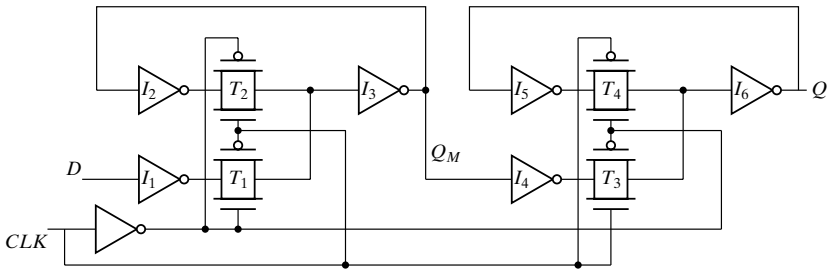
Master-Slave (Edge-Triggered) Register



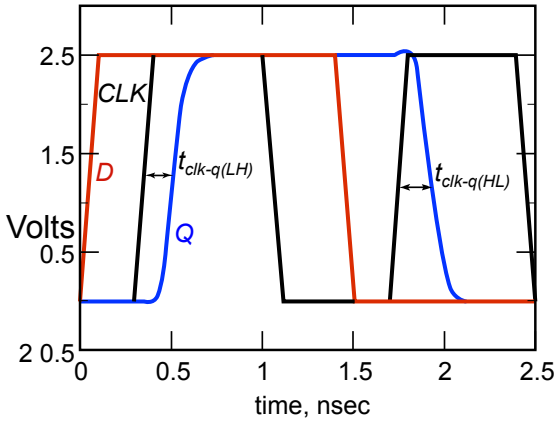
Two opposite latches trigger on edge
Also called master-slave latch pair

Master-Slave Register

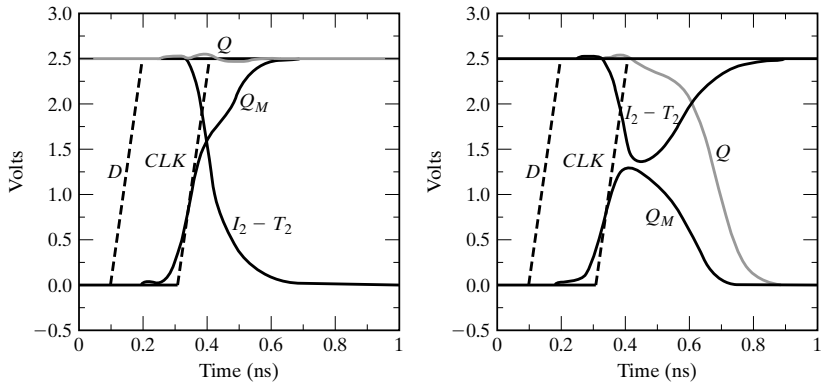
Multiplexer-based latch pair



Clk-Q Delay

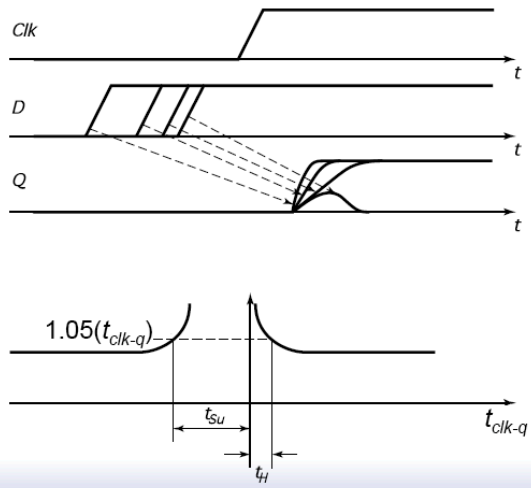


Setup Time



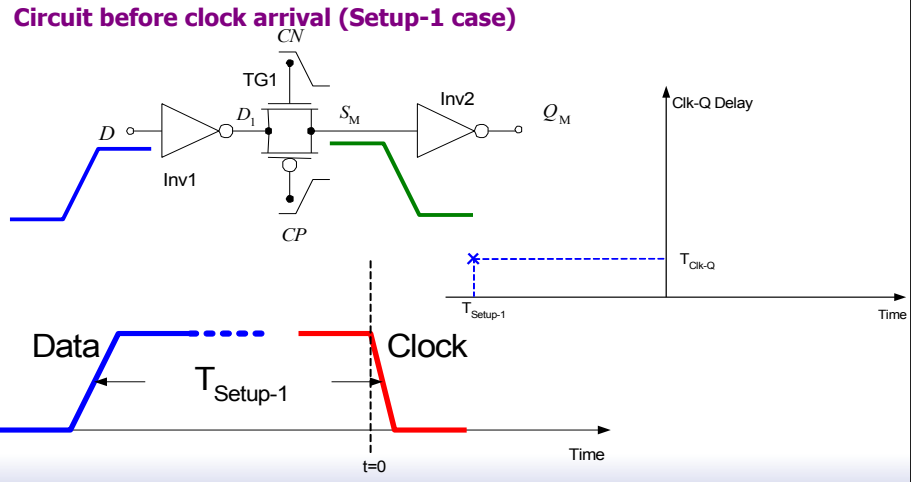
(a) T_{setup} = 0.21 ns (b) T_{setup} = 0.20 ns

More Precise Setup Time



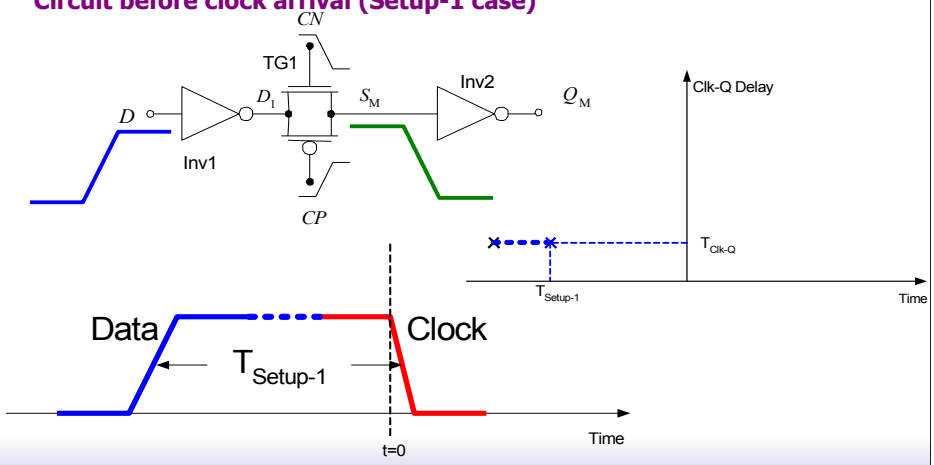
Setup-Hold Time Illustrations

Circuit before clock arrival (Setup-1 case)



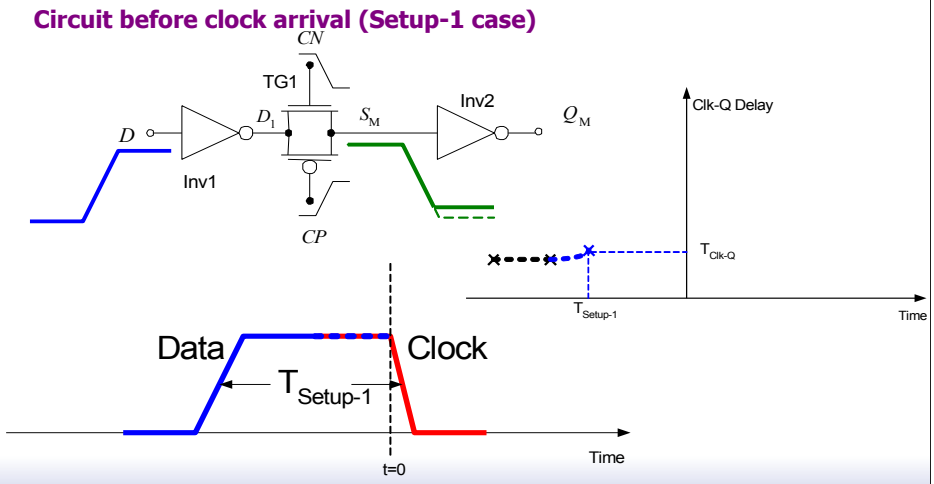
Setup-Hold Time Illustrations

Circuit before clock arrival (Setup-1 case)



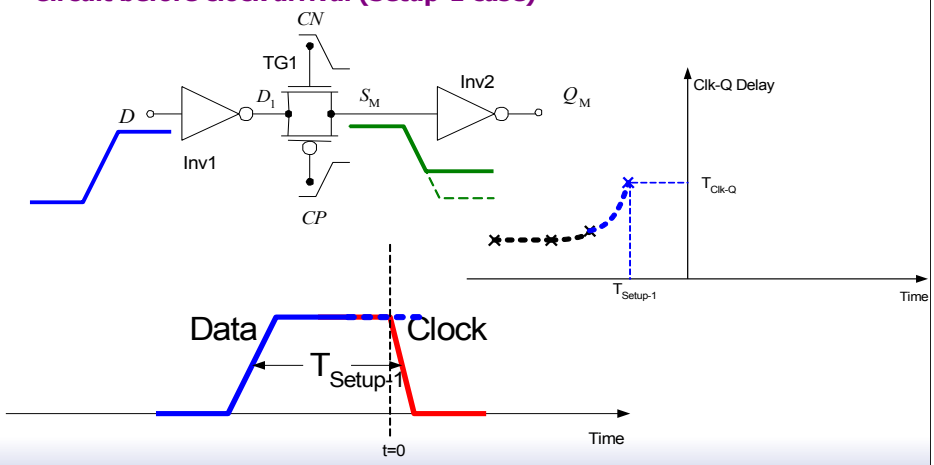
Setup-Hold Time Illustrations

Circuit before clock arrival (Setup-1 case)



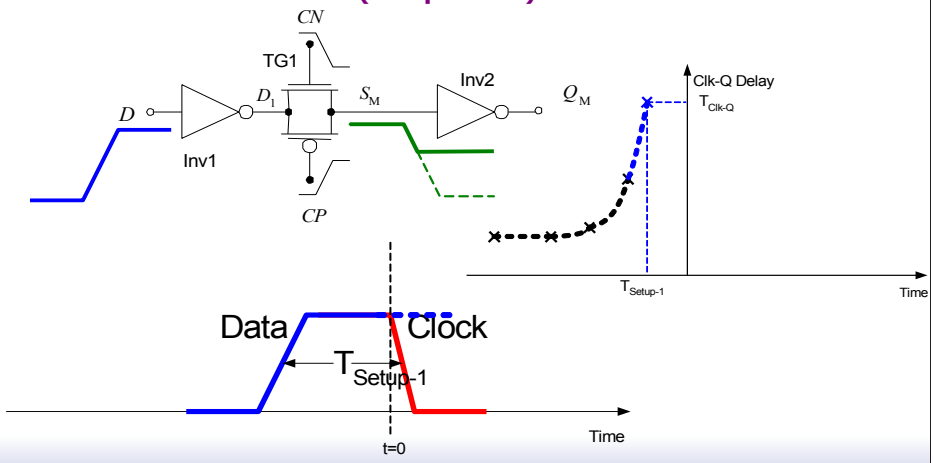
Setup-Hold Time Illustrations

Circuit before clock arrival (Setup-1 case)



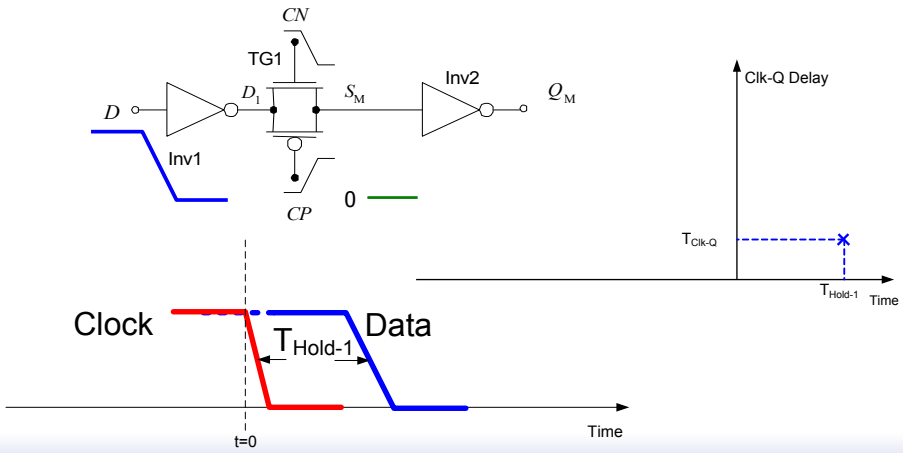
Setup-Hold Time Illustrations

Circuit before clock arrival (Setup-1 case)



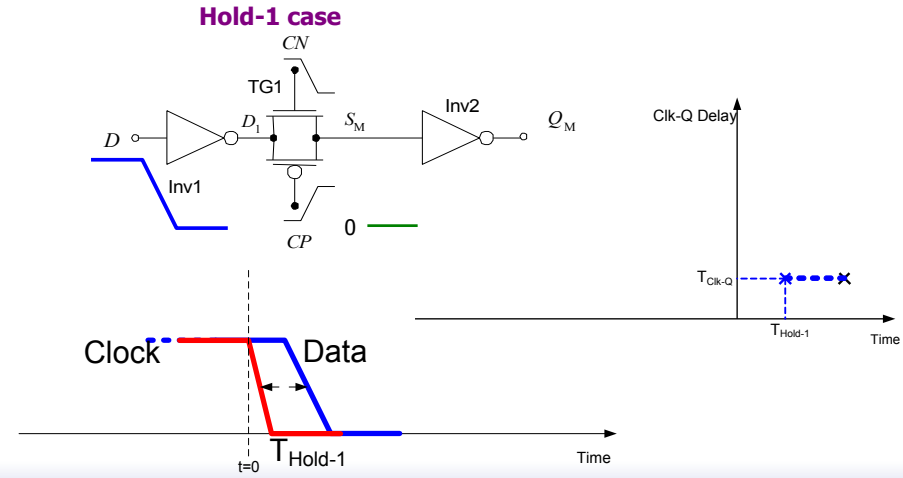
Setup-Hold Time Illustrations

Hold-1 case



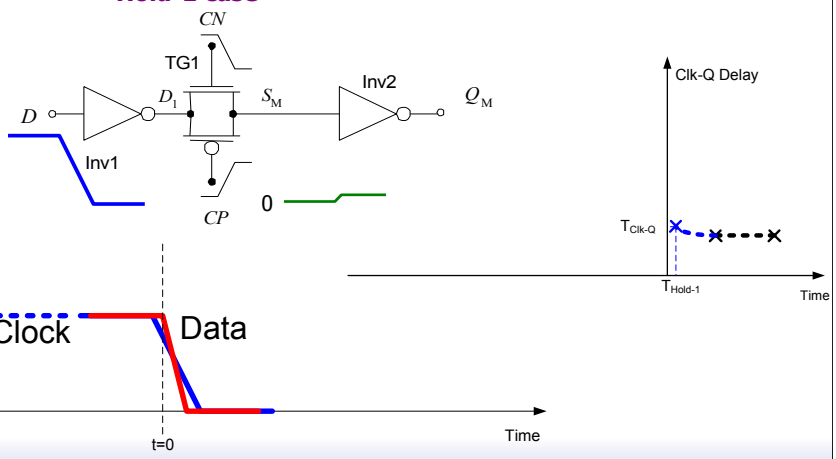
Setup-Hold Time Illustrations

Hold-1 case



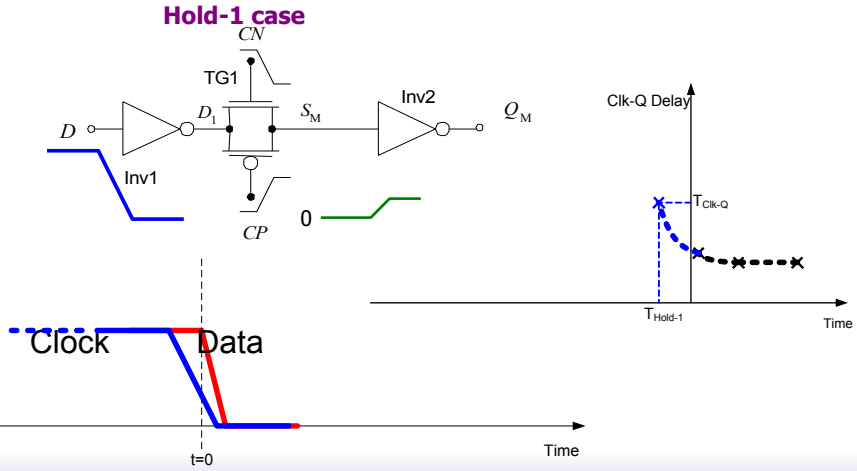
Setup-Hold Time Illustrations

Hold-1 case

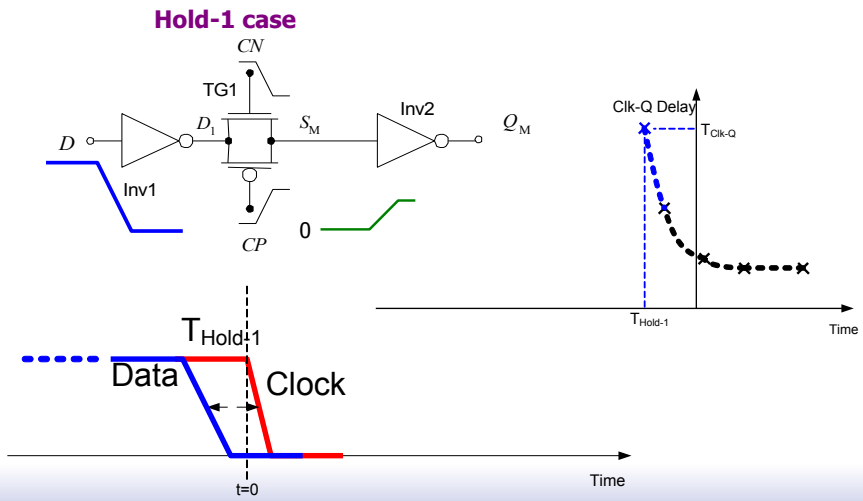


Setup-Hold Time Illustrations

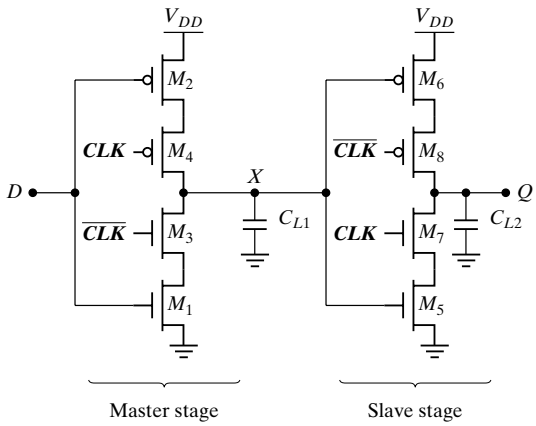
Hold-1 case



Setup-Hold Time Illustrations

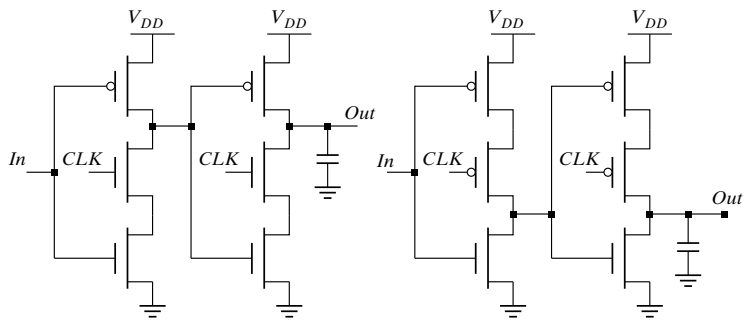


Other Latches/Registers: C²MOS



Keepers can be added to "staticize (?!)"

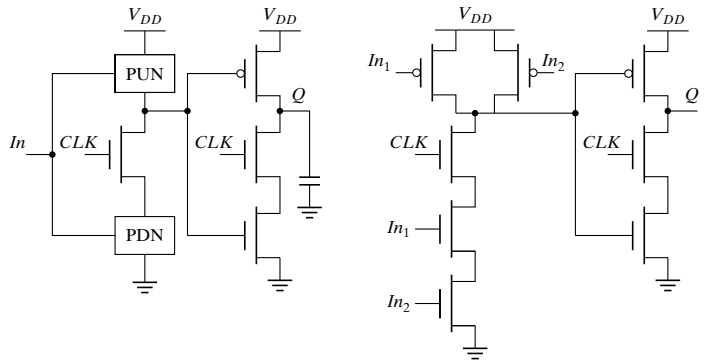
Other Latches/Registers: TSPC



Positive latch
(transparent when CLK= 1)

Negative latch
(transparent when CLK= 0)

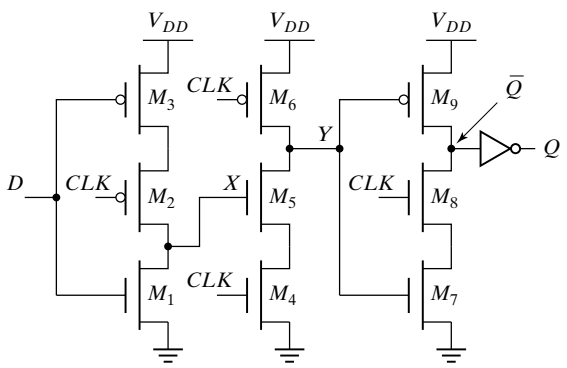
Including Logic in TSPC



Example: logic inside the latch

AND latch

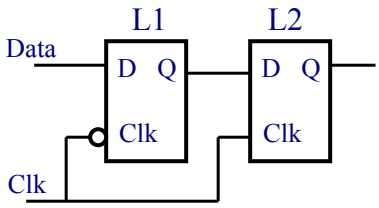
TSPC Register



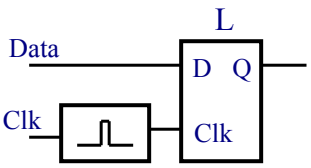
Pulse-Triggered Latches

Ways to design an edge-triggered sequential cell:

Master-Slave Latches



Pulse-Triggered Latch



Why not route the pulse?