Unit - 3

Estimation of Interconnect Parasitics

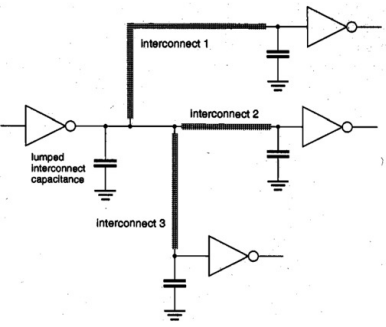
→ The classical approach for determining switch speed in logic gate is based on assumption that loads are capacitive & lumped

→ Conventional delay has 3 main components

i) internal parasitic capacitances of transistors

ii) interconnect capacitances

iii) input capacitances of fan-out gates



→ From the diagram, consider an inverter driving 3 other inverters linked by interconnection lines of different length & geometry

Assuming interconnection can be approximated by lumped capacitance, then total load (seen by primary inverter) = Σ capacitive components

→ But in reality, a lot of factor affect this

i) The line usually has non-negligible resistance in addition to capacitance

ii) L/w ratio of wire makes the interconnect a pure transmission line

iii) The interconnect is in proximity to many other lines causing capacitive/inductive coupling

→ If time of flight across intersection line $\left(\frac{l}{v}\right)$

i) T_{rise} or $T_{fall} < 2.5\left(\frac{l}{v}\right)$ ⇒ transition-line modelling

ii) $2.5\left(\frac{l}{v}\right) < T_{rise}$ or $T_{fall} < 5\left(\frac{l}{v}\right)$ ⇒ transition-line modelling (or) lumped modelling

iii) T_{rise} or $T_{fall} > 5\left(\frac{l}{v}\right)$ ⇒ lumped modelling → Close to ideal characteristics

l : interconnect line length

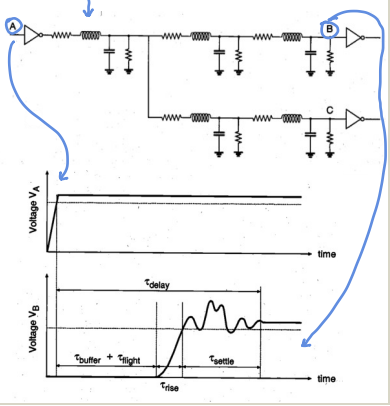
v : propagation speed

ex: 2cm VLSI chip ⇒ time of flight = 133 ps << on-chip rise & fall times ⇒ RC model

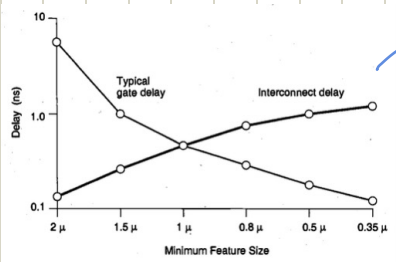
10cm multichip model ⇒ time of flight = 1 ns ≃ rise & fall times ⇒ RLCG model

RC : Resistance & Capacitance

RLCG : Resistance - Inductance - Capacitance - Conductance



→ But transition line effects aren't much concern as gate delays dominate line delays again, as technology advances to finer sub-micron design rules, intrinsic gate delays reduce and as complexity increases, chip size & worst case, line length of chip increase



As feature size ↓ Gate Delay ↓

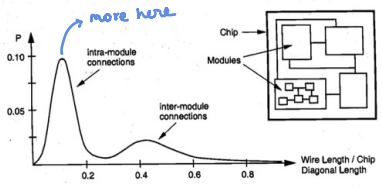
Interconnect Delay ↑

Smaller transistors switch faster, reducing intrinsic gate delay

Wires become thinner and closer together increasing R & C

Ideally we want low interconnect & gate delays

BUT both can't be reduced simultaneously

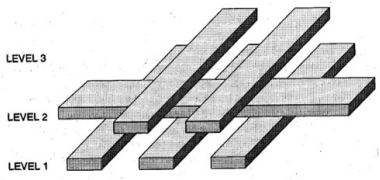


→ In a chip with many functional modules there are lot of intra-connections that are really short & easily simulated with conventional modules But there are fewer inter-connections that are long and most problematic

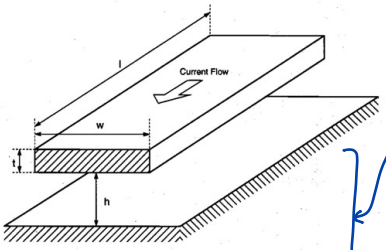
Interconnect Capacitance Estimation

→ These are among most difficult parameters to estimate accurately

→ Each wire is of different shape, thickness, vertical distance from substrate & typically surrounded by no. of other lines either on same or different level

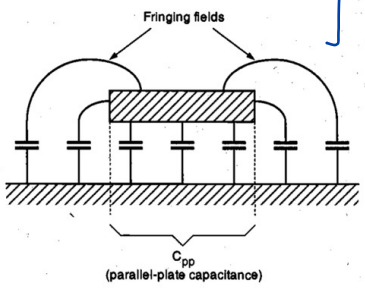


→ Consider a single interconnect of length 'l', width 'w' and thickness 't' and seperated from ground plane by dielectric oxide layer of height 'h' We can calculate capacitance (C_{pp}) normally but since wire thickness is comparable to ground-plane distance, fringing electric fields increase the total parasitic capacitance



→ Fringing-field factor $FF = \frac{C_{total}}{C_{pp}}$

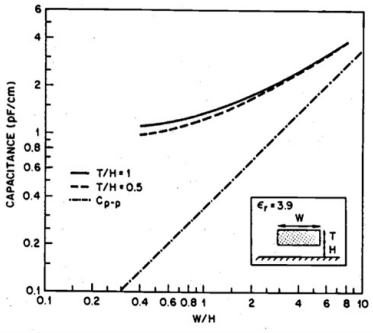
FF ↑ as $\left(\frac{w}{h}\right)$ ↓



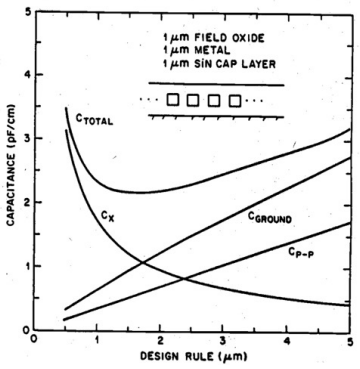
and as mentioned before, width can be reduced but not thickness

→ Yuan & Trick's formula to calculate capacitance

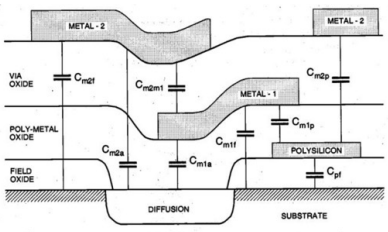
$$C = \begin{cases} \varepsilon \left[\frac{\left(w - \frac{t}{2}\right)}{h} + \frac{2\pi}{\ln\left(1 + \frac{2h}{t} + \sqrt{\frac{2h}{t}\left(\frac{2h}{t} + 2\right)}\right)} \right], & w \geq \frac{t}{2} \\ \varepsilon \left[\frac{w}{h} + \frac{\pi\left(1 - \frac{0.054t}{2h}\right)}{\ln\left(1 + \frac{2h}{t} + \sqrt{\frac{2h}{t}\left(\frac{2h}{t} + 2\right)}\right)} + 1.47 \right], & w < \frac{t}{2} \end{cases}$$



→ Actual wire capacitance decreases as width is reduced wrt thickness



$C_{Total} = C_{GROUND} + C_X$



Poly over field oxide	C_{pf}	Area	0.066	fF/μm²
		Perimeter	0.046	fF/μm
Metal - 1 over field oxide	C_{m1f}	Area	0.030	fF/μm²
		Perimeter	0.044	fF/μm
Metal - 2 over field oxide	C_{m2f}	Area	0.016	fF/μm²
		Perimeter	0.042	fF/μm
Metal - 1 over Poly	C_{m1p}	Area	0.053	fF/μm²
		Perimeter	0.051	fF/μm
Metal - 2 over Poly	C_{m2p}	Area	0.021	fF/μm²
		Perimeter	0.045	fF/μm
Metal - 2 over Metal - 1	C_{m2m1}	Area	0.035	fF/μm²
		Perimeter	0.051	fF/μm

→ Let consider a realistic scenario where interconnection line isn't isolated from surrounding structures & coupled with other lines running in parallel This coupling b/w interconnect lines is responsible for signal crosstalk (Crosstalk is noise generated due to capacitive coupling from switching neighboring lines)

→ Interconnect capacitances are difficult to calculate because of 3D Geometry so we take a guess for initial simulations & use CAD Tool to extract actual capacitances & back annotate them into our simulation to verify timing is still met post Layout

Interconnect Resistance Estimation

→ Parasitic resistance also plays a significant role on signal propagational delay

→ Resistance depends on i) Material used
ii) Dimensions of line
iii) No. and locations of contacts on that line

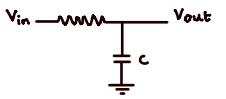
→ $R_{wire} = \frac{\rho l}{wt} = R_{sheet}\left(\frac{l}{w}\right)$

→ b/w 20 - 40 Ω/sq (Polysilicon Layer)
2 - 4 Ω/sq (Silicided Polysilicon)
0.1 Ω/sq (Aluminium)

$R_{sheet} = \frac{\rho}{t}$

Calculation of Interconnect Delay

→ RC Model



→ The simplest model that represents resistance and capacitive parasitics of interconnect

→ Assuming that
i) capacitance is initially discharged
ii) input signal is rising step pulse at t = 0

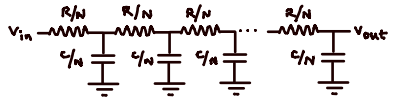
Then,

$$V_{out}(t) = V_{DD}\left(1 - e^{-t/RC}\right)$$

and V_{out} reaches 50% at $t = \tau_{PLH}$,

$$V_{50\%} = V_{DD}\left(1 - e^{-\tau_{PLH}/RC}\right) \Rightarrow \frac{V_{DD}}{2} = V_{DD}\left(1 - e^{-\tau_{PLH}/RC}\right) \Rightarrow \frac{1}{2} = e^{-\tau_{PLH}/RC} \Rightarrow 2 = e^{\tau_{PLH}/RC}$$

$$\boxed{\tau_{PLH} = 0.69RC}$$



→ To accurately get transient behavior of interconnect line using RC ladder network providing a series of distributed RC line which requires full scale SPICE simulations (or) methods like Elmore delay formula

P_i : Unique path from input node to node i $(i = 1, 2, \ldots N)$

$P_{ij} = P_i \cap P_j$: path b/w input & node i common to path b/w input & node j

→ Elmore Delay

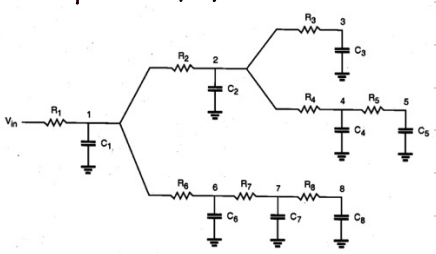
→ Consider RC network

i) No resistor loops in circuit

ii) All capacitors in RC tree are connected between node & ground

iii) One input node in circuit

Take this for example,



$$\tau_{Di} = \sum_{j=1}^{N} C_j \sum_{\forall k \in P_{ij}} R_k$$

Elmore Delay = First order time constant (approx.)

ex :

Node 1 ⇒ $R_1C_1 + R_1C_2 + R_1C_3 + R_1C_4 + R_1C_5 + R_1C_6 + R_1C_7 + R_1C_8$

Node 2 ⇒ $R_1C_1 + (R_1+R_2)C_2 + (R_1+R_2)C_3 + (R_1+R_2)C_4 + (R_1+R_2)C_5 + R_1C_6 + R_1C_7 + R_1C_8$

Node 3 ⇒ $R_1C_1 + (R_1+R_2)C_2 + (R_1+R_2+R_3)C_3 + (R_1+R_2)C_4 + (R_1+R_2)C_5 + R_1C_6 + R_1C_7 + R_1C_8$

Node 4 ⇒ $R_1C_1 + (R_1+R_2)C_2 + (R_1+R_2)C_3 + (R_1+R_2+R_4)C_4 + (R_1+R_2+R_4)C_5 + R_1C_6 + R_1C_7 + R_1C_8$

Node 5 ⇒ $R_1C_1 + (R_1+R_2)C_2 + (R_1+R_2)C_3 + (R_1+R_2+R_4)C_4 + (R_1+R_2+R_4+R_5)C_5 + R_1C_6 + R_1C_7 + R_1C_8$

Node 6 ⇒ $R_1C_1 + R_1C_2 + R_1C_3 + R_1C_4 + R_1C_5 + (R_1+R_6)C_6 + (R_1+R_6)C_7 + (R_1+R_6)C_8$

Node 7 ⇒ $R_1C_1 + R_1C_2 + R_1C_3 + R_1C_4 + R_1C_5 + (R_1+R_6)C_6 + (R_1+R_6+R_7)C_7 + (R_1+R_6+R_7)C_8$

Node 8 ⇒ $R_1C_1 + R_1C_2 + R_1C_3 + R_1C_4 + R_1C_5 + (R_1+R_6)C_6 + (R_1+R_6+R_7)C_7 + (R_1+R_6+R_7+R_8)C_8$

→ For uniform RC network

$$\tau_{DN} = \sum_{j=1}^{N} \left(\frac{C}{N}\right) \sum_{k=1}^{j} \left(\frac{R}{N}\right)$$

$$= \frac{C}{N} \cdot \frac{R}{N} \left(\frac{N(N+1)}{2}\right) = \frac{RC(N+1)}{2N}$$

If N is very large,

$$\tau_{DN} = \lim_{N \to \infty} \left(RC\left(\frac{N}{2N}\right) + \frac{RC}{2N}\right)$$

$$= \frac{RC}{2} + 0 = \frac{RC}{2} \quad //$$

Poly over field oxide	C_{pf}	Area	0.066	$fF/\mu m^2$
		Perimeter	0.046	$fF/\mu m$
Metal - 1 over field oxide	C_{m1f}	Area	0.030	$fF/\mu m^2$
		Perimeter	0.044	$fF/\mu m$
Metal - 2 over field oxide	C_{m2f}	Area	0.016	$fF/\mu m^2$
		Perimeter	0.042	$fF/\mu m$
Metal - 1 over Poly	C_{m1p}	Area	0.053	$fF/\mu m^2$
		Perimeter	0.051	$fF/\mu m$
Metal - 2 over Poly	C_{m2p}	Area	0.021	$fF/\mu m^2$
		Perimeter	0.045	$fF/\mu m$
Metal - 2 over Metal - 1	C_{m2m1}	Area	0.035	$fF/\mu m^2$
		Perimeter	0.051	$fF/\mu m$

Q. Consider uniform polysilicon line with length 1000μm & width 4 μm. Find total lumped resistance, if sheet resistance is 30 Ω/square

A. $R_{lumped} = R_{sheet} \times$ No. of Squares

$= 30 \times \frac{1000\mu}{4\mu} = 7500\,\Omega$

Q. Using the values from prev. question, calculate total capacitance

A. $C_{lumped\_total} = C_{parallel\text{-}plate} + C_{fringe}$

$C_{parallel\text{-}plate} =$ (Unit area Capacitance) x Area

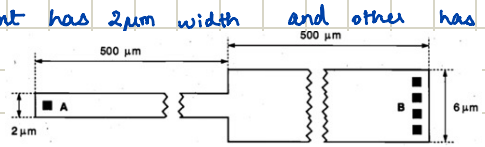
$= (0.066 fF/\mu m^2) \times (1000\mu \times 4\mu) = 264\,fF$

$C_{fringe} =$ (Unit Perimeter Capacitance) x Perimeter

$= (0.046\ fF/\mu m) \times (1000\mu + 4\mu + 1000\mu + 4\mu) = 92\,fF$

$C_{lumped\_total} = 264\,fF + 92fF = 356fF$

Q. Consider a polysilicon line consisting 2 segments, each 500μm. One segment has 2μm width and other has 6μm width. Find C_{lumped} & R_{lumped} (Assume $R_{sheet} = 30\,\Omega$/square)



A. $R_{lumped\_A} = 30 \times \left(\frac{500\mu}{2\mu}\right) = 7500\,\Omega$

$R_{lumped\_B} = 30 \times \left(\frac{500\mu}{6\mu}\right) = 2500\,\Omega$

$R_{lumped\_total} = 7500 + 2500 = 10000\,\Omega$

$C_{lumped\_A} = C_{parallel\ plate\ A} + C_{fringe\ A}$

$= (0.066\ fF/\mu m^2 \times 500\mu \times 2\mu) + (0.046\ fF/\mu m \times (500\mu + 2\mu + 500\mu + 2\mu)$

$= 66 + 46 = 112\,fF$

$C_{lumped\_B} = C_{parallel\ plate\ B} + C_{fringe\ B}$

$= (0.066\ fF/\mu m^2 \times 500\mu \times 6\mu) + (0.046\ fF/\mu m \times (500\mu + 6\mu + 500\mu + 6\mu)$

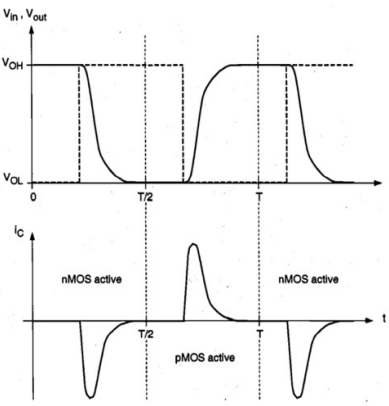
$= 198 + 46 = 244\,fF$

So in total,

$C_{lumped\_total} = C_{lumped\ A} + C_{lumped\ B}$

$= 112\,fF + 244\,fF$

$= 356\,fF$



Switching Power Dissipation

→ Static power dissipation of CMOS inverter is negligible

→ During switching events, output load capacitance is alternatively charged up & down while CMOS inverter dissipates power

→ Now take CMOS inverter with ideal step waveform with negligible rise & fall time

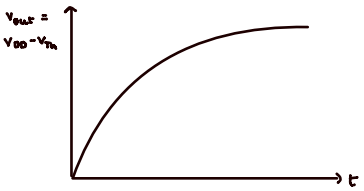
Assume Periodic input & output waveforms

$$P_{avg} = \frac{1}{T}\int_0^T v(t)\,i(t)\,dt$$

$$= \frac{1}{T}\left[\int_0^{T/2} V_{out}\left(-C_{load}\frac{dV_{out}}{dt}\right)dt + \int_{T/2}^{T}\left(V_{DD}-V_{out}\right)\left(C_{load}\frac{dV_{out}}{dt}\right)dt\right]$$

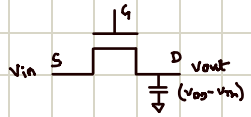
$$= \frac{1}{T}\left[-C_{load}\frac{V_{out}^2}{2}\Big|_0^{T/2} + \left(V_{DD}V_{out}C_{load} - \frac{1}{2}C_{load}V_{out}^2\right)_{T/2}^{T}\right]$$

$$P_{avg} = \frac{C_{load}V_{out}^2}{T}$$



Pass transistor

→

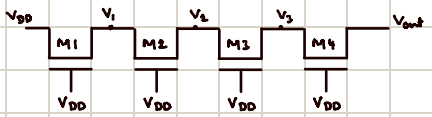


$V_g \geq V_{t,n}$ for nMOS to be ON

$V_g = V_{DD}$

Cascading a chain of Pass Transistors

1)



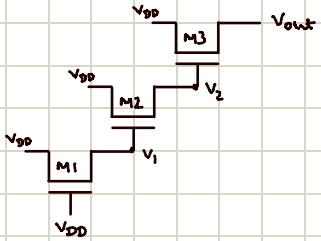
$V_1 = V_{DD} - V_{T,n1}$

$V_2 = V_{DD} - V_{T,n2}$

$V_3 = V_{DD} - V_{T,n3}$

$V_{out} = V_{DD} - V_{T,n4}$

2)



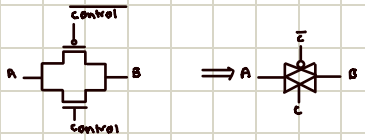
$V_1 = V_{DD} - V_{T,n1}$

$V_2 = V_{DD} - V_{T,n1} - V_{T,n2}$

$V_{out} = V_{DD} - V_{T,n1} - V_{T,n2} - V_{T,n3}$

Transmission Gate

→ A simple switch consisting NMOS & PMOS in parallel to get full swing of logic 0 and logic 1



C	A	B
0	0	H(1)
0	1	H(1)
1	0	0
1	1	1

H: High Impedance Mode

They are CMOS pass gates which pass both strong logic '1' and logic '0'

i) Implementation of AND using PT & TG

We know

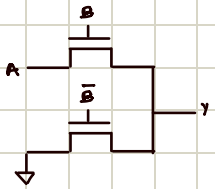
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

⇒ Use B as Control Signal

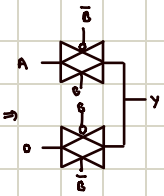
B = 1 ⇒ Y = A

B = 0 ⇒ Y = 0

Pass Gate ⇒



Transmission Gate ⇒



ii) Implementation of XOR using PT & TG

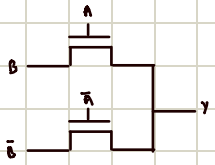
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

⇒ Use A as Control Signal

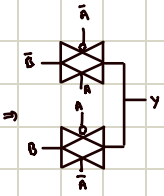
If A = 0 ⇒ Y = B

If A = 1 ⇒ Y = B̄

Pass Gate ⇒



Transmission Gate ⇒

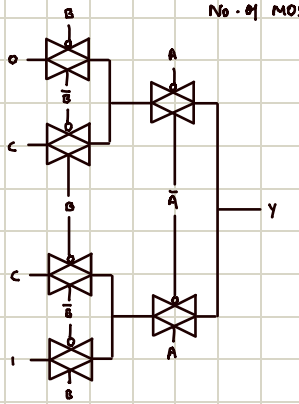


iii) Implementation of AB + BC + CA using TG

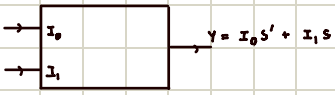
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

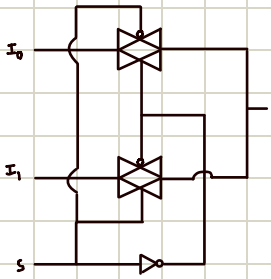
A	B	Y
0	0	0
0	1	C
1	0	C
1	1	1

Transmission Gate ⇒



iv) Implementation of 2:1 MUX using TG

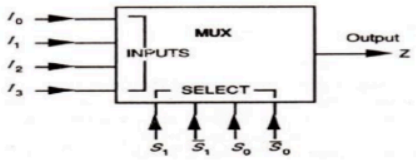




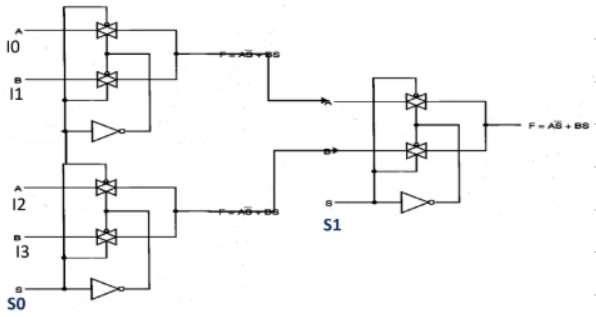
No. of MOS = 2 + 2 + 2 = 6 (TG, TG, Inv)

iv) Implementation of 4:1 MUX using TG

Example - 4:1 MUX using TG/ CMOS Pass Gate

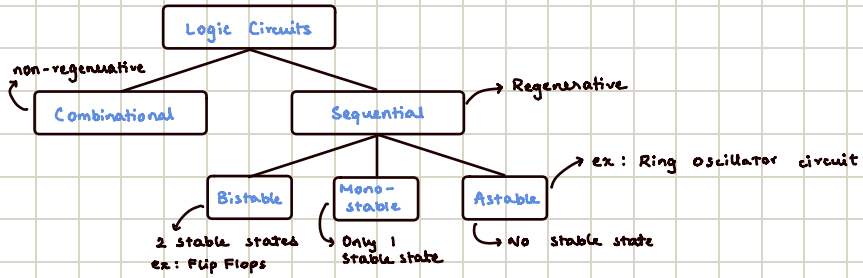


S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3



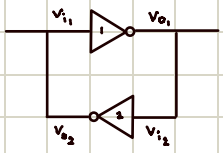
$$Y = I0.S0'S1' + I1.S0S1' + I2.S0'S1 + I3.S0S1$$

Classification of logic circuits based on temporal behaviour



Behaviour of Bistable Elements

→

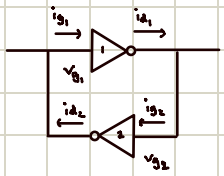


Here, $V_{o_1} = V_{i_2}$

$V_{o_2} = V_{i_1}$

→ In the graph, the VTC characteristics of both inverters intersect at 3 points. Voltage gain of inverters is larger than unity at unstable point, and a small voltage change at input of any of the inverters will be amplified, causing operation point to move to one of the stable points

→



At $V_{o_1} = V_{o_2}$ (unstable cond^n),

$i_{g_1} = i_{d_2} = g_m V_{g_2}$ (gate current; drain current)

$i_{g_2} = i_{d_1} = g_m V_{g_1}$

$g_m = g_{m_N} + g_{m_P}$ → Small signal transconductance

> At unstable state, all 4 Transistors are in saturation to get maximum gain

$$V_{g_1} = \frac{q_1}{C_g} \Rightarrow q_1 = C_g V_{g_1} \Rightarrow i_{g_1} = \frac{dq_1}{dt} = C_g \frac{dV_{g_1}}{dt} \Rightarrow g_m V_{g_2} = C_g \frac{dV_{g_1}}{dt}$$

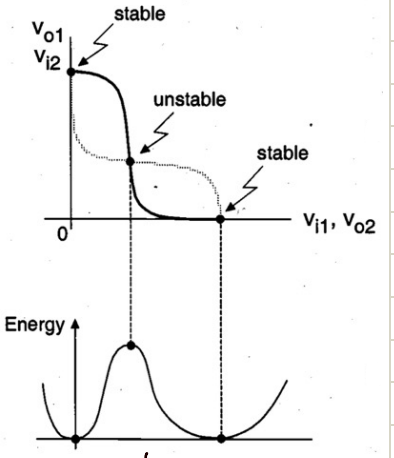
$$\text{and } V_{g_2} = \frac{q_2}{C_g} \Rightarrow q_2 = C_g V_{g_2} \Rightarrow i_{g_2} = \frac{dq_2}{dt} = C_g \frac{dV_{g_2}}{dt} \Rightarrow g_m V_{g_1} = C_g \frac{dV_{g_2}}{dt}$$

$$\text{So, } g_m V_{g_2} = C_g \frac{dV_{g_1}}{dt} \Rightarrow \frac{g_m}{C_g} q_2 = \frac{dq_1}{dt} \Rightarrow \frac{g_m}{C_g} q_2 = \frac{C_g}{g_m}\left(\frac{d^2 q_2}{dt^2}\right) \Rightarrow \left(\frac{d^2 q_2}{dt^2}\right) = \left(\frac{g_m}{C_g}\right)^2 q_2 = \frac{q_2}{\tau_0^2}$$

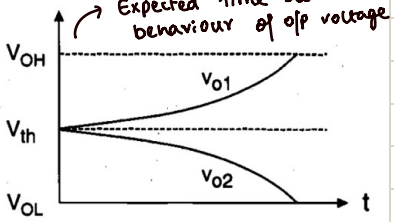
$$g_m V_{g_1} = C_g \frac{dV_{g_2}}{dt} \Rightarrow \frac{g_m}{C_g} q_1 = \frac{dq_2}{dt} \Rightarrow \frac{g_m}{C_g} q_1 = \frac{C_g}{g_m}\left(\frac{d^2 q_1}{dt^2}\right) \Rightarrow \left(\frac{d^2 q_1}{dt^2}\right) = \left(\frac{g_m}{C_g}\right)^2 q_1 = \frac{q_1}{\tau_0^2}$$

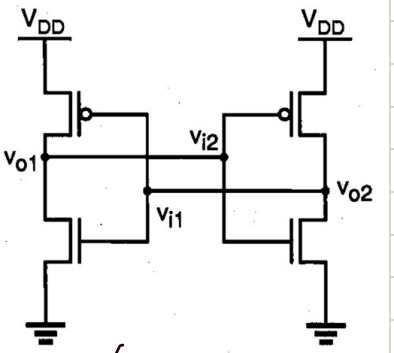
(where $\tau_0 = C_g / g_m$)

Derivation Continued ↓



→ Bistable behaviour





→ CMOS 2-inverter bistable element

Now assume $K^2 = \frac{1}{\tau_0^2} \Rightarrow \frac{d^2q_1}{dt^2} = K^2 q_1 \Rightarrow \frac{d^2q_1}{dt^2} - K^2 q_1 = 0$

Roots are $q_1(t) = Ae^{Kt} + Be^{-Kt}$

$= Ae^{t/\tau_0} + Be^{-t/\tau_0} \longrightarrow (1)$

$q_1(0) = A + B \longrightarrow (2)$

$q_1'(t) = \frac{A}{\tau_0} e^{t/\tau_0} - \frac{B}{\tau_0} e^{-t/\tau_0}$

$q_1'(0) = \frac{A-B}{\tau_0} \longrightarrow (3)$

Solving (2) & (3), $A = \frac{q_1(0) + \tau_0 q_1'(0)}{2}$ & $B = \frac{q_1(0) - \tau_0 q_1'(0)}{2}$

Put this in (1), $\frac{q_1(0) + \tau_0 q_1'(0)}{2} e^{-t/\tau_0} + \frac{q_1(0) - \tau_0 q_1'(0)}{2} e^{t/\tau_0}$

We know $q_1(0) = C_g V_{g_1}(0)$ (Initial Condition)

Then, $V_{o_2}(t) = \frac{1}{2}\left(V_{o_2}(0) - \tau_0 V_{o_2}'(0)\right)e^{-t/\tau_0} + \frac{1}{2}\left(V_{o_2}(0) + \tau_0 V_{o_2}'(0)\right)e^{t/\tau_0}$

$V_{o_1}(t) = \frac{1}{2}\left(V_{o_1}(0) - \tau_0 V_{o_1}'(0)\right)e^{-t/\tau_0} + \frac{1}{2}\left(V_{o_1}(0) + \tau_0 V_{o_1}'(0)\right)e^{t/\tau_0}$

For large values of t,

$V_{o_2}(t) = \frac{1}{2}\left(V_{o_2}(0) + \tau_0 V_{o_2}'(0)\right)e^{t/\tau_0}$

$V_{o_1}(t) = \frac{1}{2}\left(V_{o_1}(0) + \tau_0 V_{o_1}'(0)\right)e^{t/\tau_0}$

→ Magnitude of output voltage increase exponentially with time

Also polarity of output voltages should be opposite to satisfy charge conservation principle

Hence, V_{o_1} : $V_{Th} \longrightarrow V_{OH}$ or V_{OL}

V_{o_2} : $V_{Th} \longrightarrow V_{OL}$ or V_{OH}

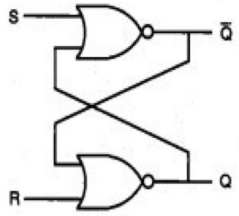
→ In a time interval T, signal travels the loop 'n' times,

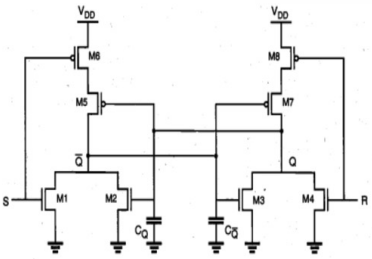
This is equivalent to same signal propagating along cascaded inverter chain consisting of '2n' inverters

So, $A^n = e^{T/\tau_0}$

where, A : loop gain (Combined voltage gain of cascaded inverters)

and expression describes time domain behaviour of diverging process until stable point is reached





Asynchronous CMOS SR Latch based on NOR2 gates

→ Uses 2 cross-coupled CMOS NOR Gates

Inputs : S and $\bar{S}$

Outputs : Q and $\bar{Q}$

No clock is used as it is asynchronous

S	R	Q	$\bar{Q}$	Description
0	0	Q_{prev}	$\bar{Q}_{prev}$	Hold State
1	0	1	0	Set
0	1	0	1	Reset
1	1	0	0	Invalid

→ It uses 4 nMOS & 4 pMOS

S	R	Q	$\bar{Q}$	M_1	M_2	M_3	M_4	Description
V_{OL}	V_{OL}	V_{OH}	V_{OL}	OFF	ON	OFF	OFF	Hold (Q=1)
V_{OL}	V_{OL}	V_{OL}	V_{OH}	OFF	OFF	ON	OFF	Hold (Q=0)
V_{OH}	V_{OL}	V_{OH}	V_{OL}	ON	ON	OFF	OFF	Set
V_{OL}	V_{OH}	V_{OL}	V_{OH}	OFF	OFF	ON	ON	Reset
				↓ Control Q		↓ Control $\bar{Q}$		

→ For transient analysis, we need to take events causing state change (initially reset being set or vice versa)

When SR Latch switches states, both outputs change simultaneously in opposite directions, making it difficult to solve 2 coupled differential equations which is complex

Instead we assume both transitions happen sequentially, which causes overestimation of switching time which requires calculation of total parasitic capacitance

$C_Q = C_{gb_2} + C_{gb_5} + C_{db_3} + C_{db_4} + C_{db_7} + C_{sb_7} + C_{db_8}$

$C_{\bar{Q}} = C_{gb_3} + C_{gb_7} + C_{db_1} + C_{db_2} + C_{db_5} + C_{sb_5} + C_{db_6}$

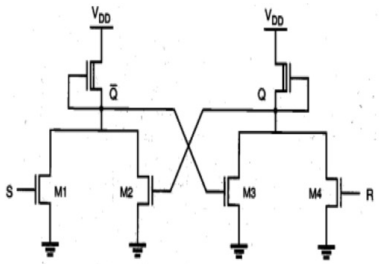
→ Assuming set signal is applied to the latch, $\bar{Q} \Rightarrow V_{OH} \rightarrow V_{OH}$ (M1 turns ON)

$Q \Rightarrow V_{OL} \rightarrow V_{OH}$ (M3 turns OFF)

Rise time of node Q is $\tau_{rise,Q}$ [SR-Latch] = $\tau_{rise,Q}$ [NOR2] + $\tau_{fall,\bar{Q}}$ [NOR2]

→ First order estimation gives an idea of delay w/o solving differential equation as it is simpler than simultaneous switching

→ In reality, M2 & M4 might assist in switching, making it even faster than estimated



Asynchronous Depletion-load nMOS SR Latch based on NOR2

→ From logic pov, it is identical to previous CMOS SR Latch using NOR2

Advantage :

→ Easy to fabricate

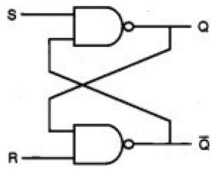
→ Used in early microprocessors due to ease of integration

→ We use $2(N+1) = 2(2+1) = 6$ MOS devices ✓ better
in CMOS logic $2(N) = 2(4) = 8$ MOS devices

Disadvantage :

→ CMOS circuit offer better alternative in terms of power dissipation & noise margin as they disipate visually no static power for preserving a state

→ Output voltages exhibit full swing from 0 to V_{DD} in CMOS logic



Asynchronous CMOS SR Latch circuit based on NAND2 gates

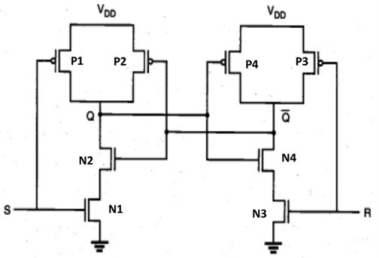
→

S	R	Q	Q̄	Description
0	0	0	0	Invalid
1	0	0	1	Reset
0	1	1		Set
1	1	Q_{prev}	$\bar{Q}_{prev}$	Hold State

→ Differences between NOR & NAND based SR Latches

Feature	NOR	NAND
Input type	Active-High	Active-Low
Circuit Complexity	Uses NOR2	Uses NAND2

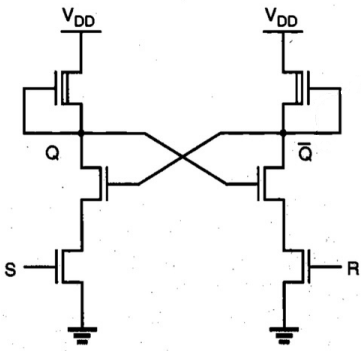
→ Other than logic, it has same characteristics, pros & cons as CMOS NOR2 logic



Asynchronous Depletion-load nMOS SR Latch based on NAND2

→ Constructed using 2 cross-coupled NAND2 made from nMOS transistors

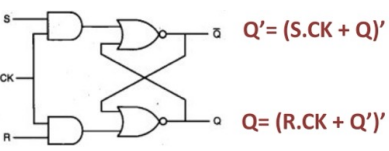
→ Operates the same way as CMOS SR Latch using NAND Gates



→ Differences between Depletion load nMOS & CMOS SR Latches

	Feature	Depletion Load	CMOS
1)	Technology	All nMOS	nMOS + pMOS
2)	Power Consump^n	High (static power)	Low (only dynamic)
3)	Noise Margin	Lower	Higher
4)	Voltage Swing	Not Full ($<V_{DD}$)	Full (0 to V_{DD})
5)	Area	Small	More than Dep. Load
6)	Era of use	Early ICs	Modern ICs

IC : Integrated Circuits



↳ AOI Implementation

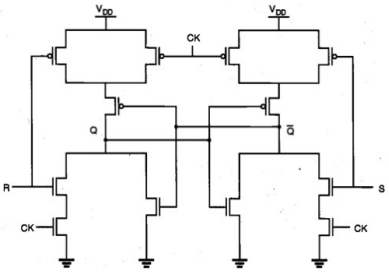
Synchronous CMOS SR Latch based on NOR2

→ We use level sensitive synchronous latch (Responds when CK is active)

Inputs : R, S, CK

Outputs : Q, $\overline{Q}$

→ An AND Gate is used to combine inputs (S, R) with Clock before entering NOR Latch



CK	S	R	Q_{next}	Operation
0	X	X	Q	Hold
1	0	0	Q	Hold
1	1	0	1	Set
1	0	1	0	Reset
1	1	1	–	Invalid

X : Don't care

Advantages

→ Controlled updates using clock gating

→ Reduces chances of unintended state changes

Disadvantages

→ Since it is level-sensitive, it is susceptible to glitches or changes when clock is high

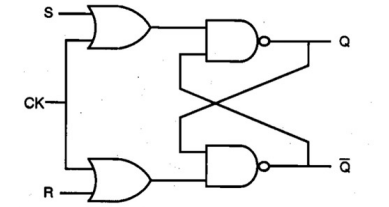
→ Still has invalid state

Synchronous CMOS SR Latch based on NAND2

→ We have 2 versions - active high & active low

→ Active Low

→ Latch responds when CK = 0

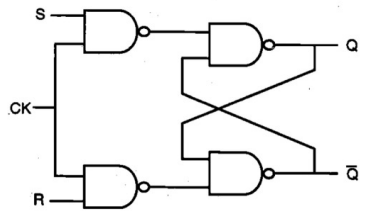


↳ Active Low

CK	S	R	Q_{next}	Operation
1	X	X	Q	Hold
0	1	1	Q	Hold
0	0	1	1	Set
0	1	0	0	Reset
0	0	0	–	Invalid

→ Active High

→ Latch responds when CK = 1



↳ Active High

CK	S	R	Q_{next}	Operation
0	X	X	Q	Hold
1	0	0	Q	Hold
1	1	0	1	Set
1	0	1	0	Reset
1	1	1	–	Invalid

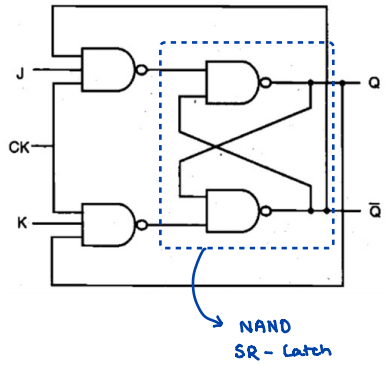
→ Comparision b/w NOR & NAND based Synchronous SR Latches

Feature	NOR	NAND
Input Logic Type	Active High	Active Low/High
Clock Sensitivity	CK=1	CK=0
Logic Implementation	AND + NOR	NAND
Transistor efficiency	Higher w AOI	Higher w OAI

Clocked JK Latch

→ SR Latch has a forbidden state which causes invalid output
JK Latch eliminates this issue by introducing feedback from outputs to inputs

→ Inputs : J , K , CK
Outputs : Q & $\overline{Q}$

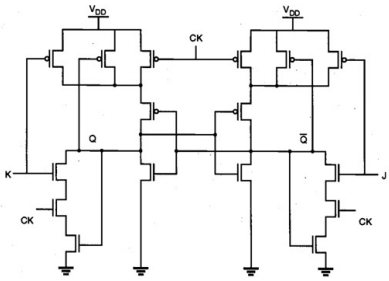


↳ NAND SR - Latch

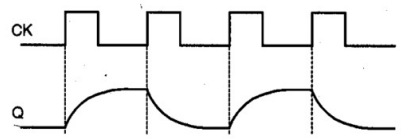
J	K	Q_n	$\overline{Q_n}$	S	R	Q_{n+1}	$\overline{Q_{n+1}}$	Operation
0	0	0	1	1	1	0	1	*hold*
		1	0	1	1	1	0	
0	1	0	1	1	1	0	1	*reset*
		1	0	1	0	0	1	
1	0	0	1	0	1	1	0	*set*
		1	0	1	1	1	0	
1	1	0	1	0	1	1	0	*toggle*
		1	0	1	0	0	1	

(Toggle Flips the input $Q \to \overline{Q}$, $\overline{Q} \to Q$, No invalid state)

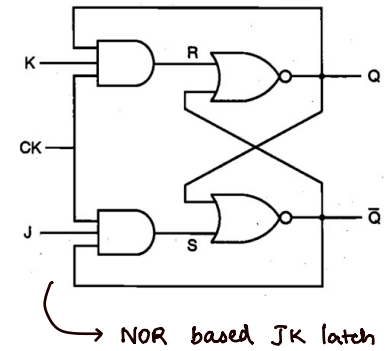
→ The all - NAND Implementation uses several NAND Gates leading to more transistors
→ We can use AOI form or NOR based for compact logic, Thus less transistor count



↳ CMOS AOI realization of JK Flip-flop



⇒ When J & K both are left high and clock keeps alternating between 0 and 1, we can see how Q keeps toggling on each rising edge of CK

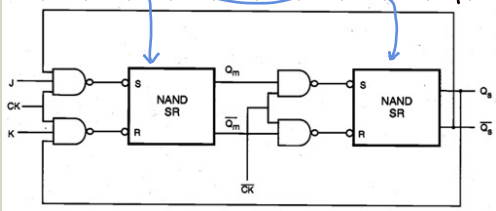


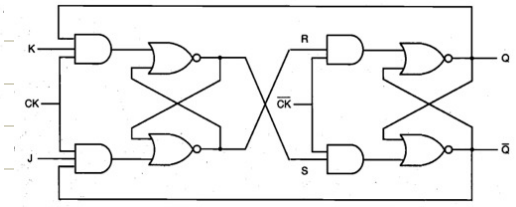
↳ NOR based JK latch

Master-Slave JK Flip-flop

→ Made of 2 clocked JK latches in series (Level Sensitive)
i) Master stage : Captures input when CK = 1
ii) Slave stage : Updates output when CK = 0

NOR based realization



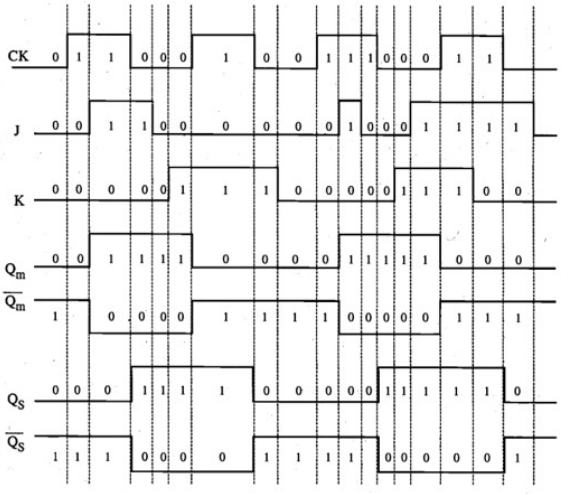


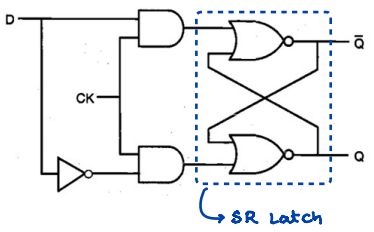
→ When CK = 1, master latch is enabled, reads input from J & K & output of master changes (Q_m) but it isn't final output yet.
When CK = 0, slave is enabled, master is disabled & slave copies state from master and final output Q is updated
→ The output changes only once per clock cycle, prevents multiple toggling within 1 CK pulse, avoids glitching, making it suitable for counters, registers, FSMs etc.,

J	K	Q_{next}	Operation
0	0	Q	Hold
1	0	1	Set
0	1	0	Reset
1	1	$\overline{Q}$	Toggle

NAND based : 36 Transistors
AOI based : 28 Transistors

Sample Input & Output of master-slave JK latch





CMOS D-Latch

→ It is a simplified SR latch

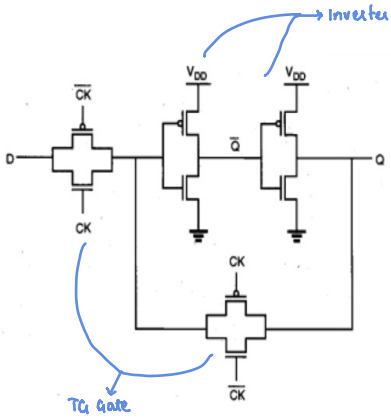
Inputs : D , CK

Outputs : Q , $\bar{Q}$

CK	D	Q_{next}	Operation
0	x	Q	Hold
1	0	0	Latch 0
1	1	1	Latch 1

→ When CK = 1, Latch is transparent (Q = D)

CK = 0, Latch holds previous value



→ It is level sensitive

→ This ensures S & R are never the same

→ We can make using 2 TG and 2 inverters

TG is controlled by CK & $\overline{CK}$

Input TG : ON when CK = 1 (Data sampled)

Feedback TG : ON when CK = 0 (Data Held)

→ Timing conditions :

D must be stable for short time before & after CK goes low

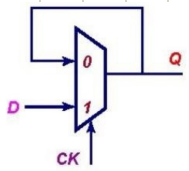
i) Setup time : Time before CK goes low when D must be valid

ii) Hold time : Time after CK goes low during which D must be valid

If they are violated, it can cause metastability

→ D-Latch as MUX based system

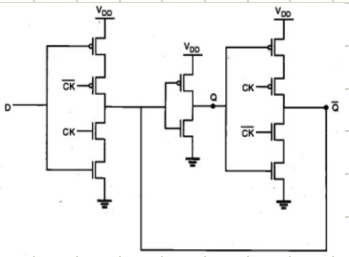
$Q = \overline{CK}.Q + CK.D$

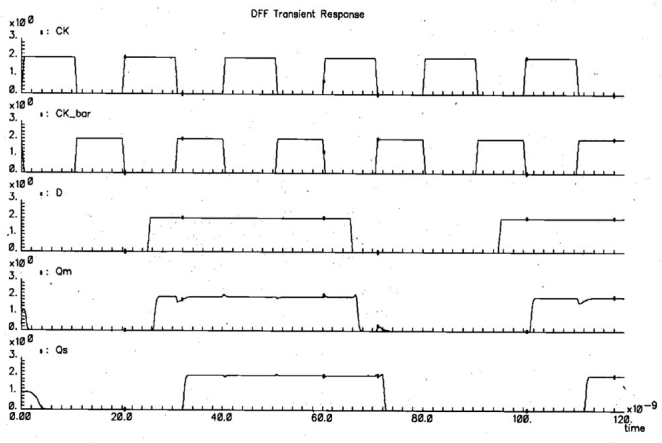


→ D-Latch using tri-state inverters

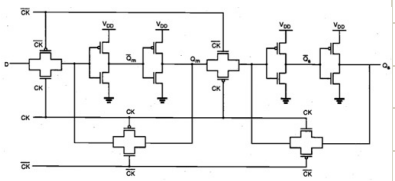
(The 1st tri-state inverter acts as input switch when clock is high & by then second tri-state is at high impedance state, output Q is following output)

(When CK is low, input buffer is inactive & 2nd tri-state inverter completes 2 inverter loop preserving state until next clock pulse)

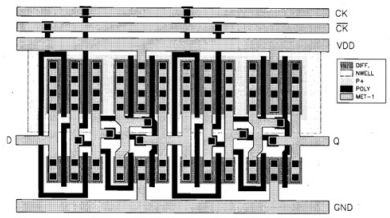




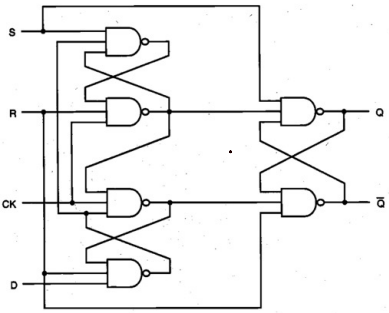
→ DFF Transient Response



CMOS negative edge-triggered Master slave D-Flip flop



Layout of CMOS DFF



NAND3 based positive edge triggered D-flip flop circuit (Doesn't need $\overline{CK}$)

Edge Triggered D Flip-flop (Master Slave DFF)

→ Built using cascading 2 D latches

Master : Transparent when CK = 1

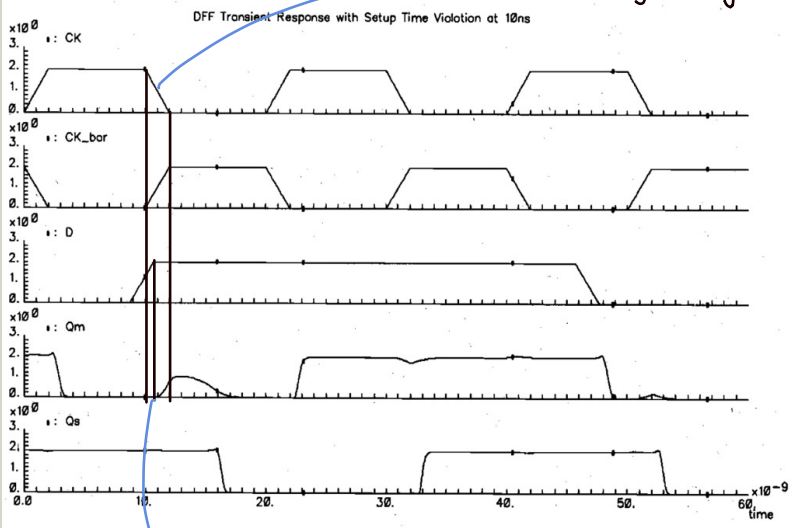
Slave : Transparent when CK = 0

Output changes only on falling edge of CK

→

Clock Edge	Action	
Rising Edge	Master tracks D	→ Not latched yet
Falling Edge	Slave updates Q	→ Latched

→ Output only changes on falling edge of CK



→ Output Q changed when D is high

→ This transient analysis shows setup time violation resulting in glitches / invalid state

Feature	D - Latch	D Flip-flop
Type	level sensitive	Edge triggered
Transparency	when CK=1	Never
Behaviour	Q follows D when enabled	Q updates only on edge
Glitch Sensitivity	High	Low
Common Use	Temporary Storage	Registers, Counter

Synchronous Dynamic Circuit Techniques

→ In modern digital circuits, speed, power & area are everything

→ Static CMOS is reliable, easy to design, but consumes more power (even when idle)

→ Dynamic Circuits use fewer transistors, consume power only when switching, stores charge temporarily on capacitor nodes

→ It is perfect for high speed pipelined (or) clocked digital systems

Dynamic pass transistor circuits

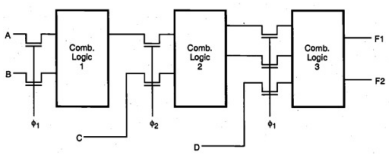
→ It is a synchronous type logic which uses :

i) nMOS pass transistors to transfer signals b/w logic states

ii) Parasitic capacitances to temporarily store logic levels

iii) 2 Phase clocking phase to control signal flow

→ The multistage circuit consists of cascaded combinational logic states, interconnected through nMOS pass transistors, inputs are driven by clock signals and input capacitances store signal b/w clock phases

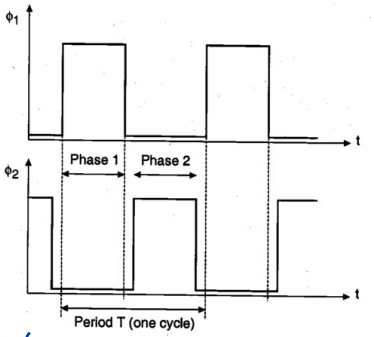


↳ Multistage pass transistors logic driven by 2 non-overlapping clocks

→ The clocking scheme (2 Phase clocking) uses 2 non-overlapping clock signals (ϕ_1 & ϕ_2) which ensures only 1 clock signal is HIGH at any moment preventing signal conflict

i) When ϕ_1 is active,
   - → Pass transistor controlled by ϕ_1 turn ON
   - → Inputs of stage 1 & 3 are transferred
   - → Stage 2 holds previous value on input capacitance

ii) When ϕ_2 is active,
   - → Pass transistor controlled by ϕ_2 turn ON
   - → Inputs of stage 2 are transferred
   - → Stage 1 & 3 holds data via charge storage



↳ Non-overlapping clock signals used for 2 phase synchronous operation

→ 2 Phase clocking ensures safe & synchronized signal flow, prevents overlapping operations & facilitates synchronous operation using periodic clock signals

→ The internal logic blocks can be built using -

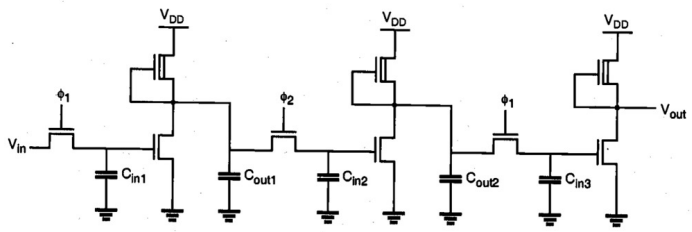
i) Depletion - load nMOS

ii) Enhancement - load nMOS

iii) CMOS Logic

Depletion - load dynamic shift register circuit

→



→ This shifts input data from 1 stage to next every clock cycle

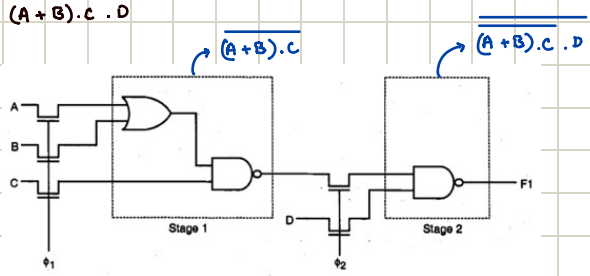
i) Stores data temporarily using charge on capacitors

ii) Moves it forward using clocked pass transistors

→ It has 3 identical stages, consisting of an input pass transistor, depletion load nMOS inverter & input capacitor at each stage which are driven by ϕ_1 & ϕ_2

→ Very common in register arrays, serial I/O, DRAM cells

ex: $\overline{\overline{(A+B).C}.D}$

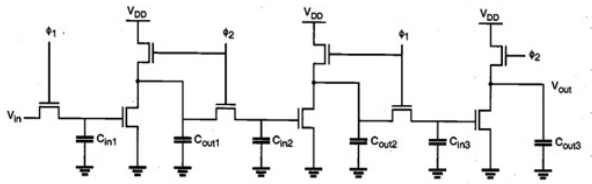


↳ This is a synchronous complex logic which is controlled by ϕ_1 & ϕ_2

Data moves from stage to stage using charge storage

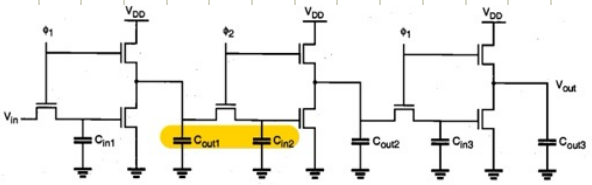
Enhancement - Load Dynamic Shift Register

→



→ Ratioed Logic

↳ Logic level depends on transistor strength ratio



→ Ratioless Logic

↳ Independant on strength ratio

Achieves full voltage swing

→ In ratioed logic,
pass transistor & load transistor are driven by opposite clocks
When ϕ_1 is high, P.T is ON, L.T is OFF
ϕ_2 is high, P.T is OFF, L.T is ON
V_{OL} is determined by driver-to-load strength ratio (if not strong enough, won't be able to pull output to 0V) Hence 'ratioed logic'

→ In ratioless logic,
P.T & L.T both are driven by same clock, so both will turn ON same time
Even logic-low outputs reach 0V regardless of size
& independant on driver-to-load ratio, Hence 'ratioless'

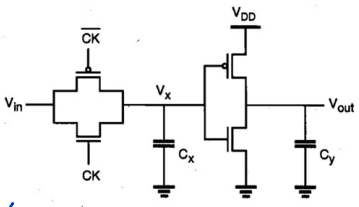
Feature	Ratioed	Ratioless
Clock phases	Pass : ϕ_1 Load : ϕ_2	Pass & Load : ϕ_1
V_{OL} control	Depends on Transistor ratio	Guaranteed 0V
Robustness	Less	More
Area efficiency	Good	Good
Sensitivity	Sizing dependant	Capacitor-ratio dependant

CMOS Transmission Gate Logic

→ Instead of using only pass gate, transmission gates are used which are controlled by ϕ and $\overline{\phi}$

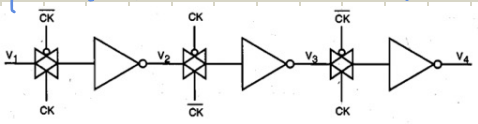
→ Using TG provides better signal quality (Full voltage swing), lower resistance (nMOS + pMOS work together)

→ During CK = 1, V_{in} is passed through T.G & charges input capacitor of inverter
CK = 0, T.G is OFF & voltage across capacitor is held



↳ Basic building block of CMOS Transmission Gate dynamic shift register

↳ Single phase CMOS shift Register

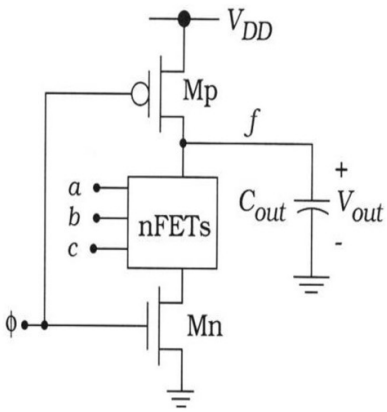


→ This is a chain of CMOS stages where odd no. stages driven by CK
even no. stages driven by $\overline{CK}$
Forming a clocked pipeline & each alternate stage accepts / holds data during different phases

Important Considerations :

1) Non-ideal clock behaviour
- CK & $\overline{CK}$ might not be perfectly non-overlapping due to rise/fall time & inversion delays
- This can cause loss of charge

2) Preffered Clocking
- Designers prefer ϕ_1 & ϕ_2 rather than CK & $\overline{CK}$ for more reliability & avoiding timing glitches



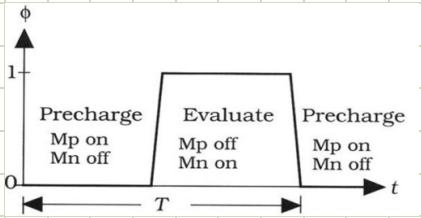
Dynamic CMOS Logic (Precharge - Evaluate Style)

→ It is a technique which uses fewer transistors than static CMOS & operates in 2 clock-controlled phases (Precharge & Evaluate) to implement any logic function

→ Usually used for high-speed & dense logic blocks like arithmetic units

→ During precharge phase (ϕ = LOW),
- pMOS precharge transistor turns ON
- output node is charged to HIGH (V_{DD})
- pull-down (nMOS) network is OFF

During evaluate phase (ϕ = HIGH),
- pMOS precharge transistor turns OFF
- pull-down (nMOS) network is ON
- based on input, if path is ON → Output discharges to 0
  if path is OFF → Output stays at 1

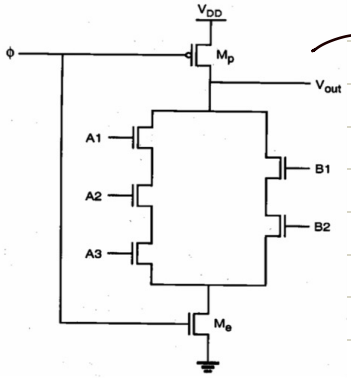


Precharge : pMOS charging to V_{DD}

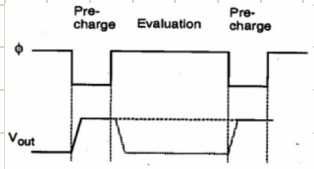
Evaluation : nMOS implementing logic

Dynamic node: Output node holding charge temporarily

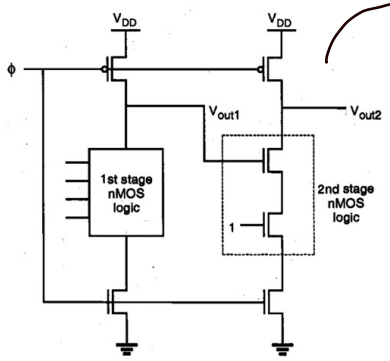
ex : $\overline{A_1A_2A_3 + B_1B_2}$



→ pMOS controlled by $\overline{\phi}$



Advantages	Disadvantages
→ High Speed	→ Charge Leakage
→ Low Area	→ Noise-sensitive
→ Ideal for complex logic	→ Non-inverting only



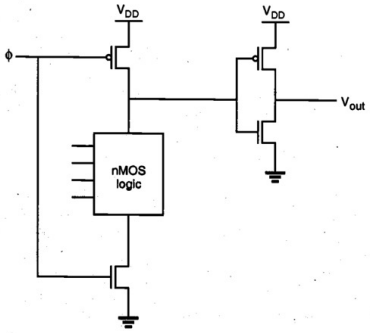
→ Cascading problem in dynamic CMOS

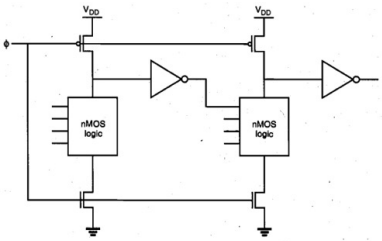
Only uses 1 pMOS & nMOS for logic

→ Imp. Design Notes

1) Input restrictions
   - → Input must transition from 0 to 1 during evaluate phase
     If it was from 1 to 0, it may pull down output incorrectly

2) Clocking Discipline
   - → Every cycle must be recharged to avoid wrong logic from leftover charge
   - → These circuits are non-inverting, so we can often chain them using inverters





↳ Cascoded domino CMOS logic to see the usage of cmos inverter

Domino Logic

→ It is a dynamic CMOS logic where multiple dynamic logic gates are cascaded

→ Each dynamic stage is followed by an inverter

→ Output of one gate triggers the next (like falling dynamos ////////)

→ The circuit consists of

i) precharge transistor - charges dynamic node when ϕ = Low

ii) evaluation network - pulls down output when ϕ = HIGH & logic condition is met

iii) CMOS inverter - Ensures full voltage swing & restores logic level

iv) Output of inverter - Goes to next stages of dynamic input

→ During pre-charge phase ($\phi = 0$),

pMOS precharge transistor turns ON, dynamic nodes charged to V_{DD} & inverters output LOW

→ During evaluate phase ($\phi = 1$),

pMOS precharge transistor turns OFF, nMOS evaluation network is turned ON,

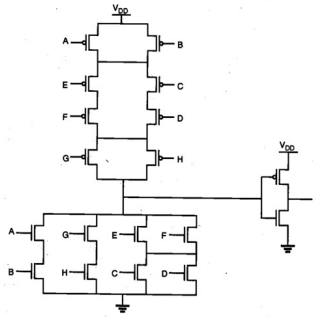
if dynamic node discharges, inverter output goes HIGH

else if node stays HIGH, inverter output remains LOW

→ Finally, output of each inverter feeds next dynamic stage's logic

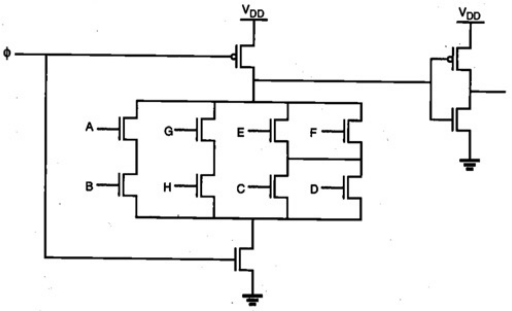
→ The reason why we use inverter is because dynamic gates can't be cascoded directly (inputs must take 0→1 Transitions in evaluate phase & 1→0 can corrupt node) So, inverter ensures full swing giving clean transitions

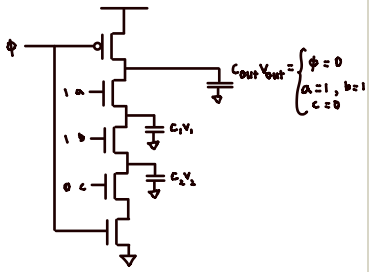
Advantages	Disadvantages	
Fast Logic (Only nMOS)	Noise - inverting only	
Compact	Clock skew issues	→ Clock must be well timed to avoid evaluation errors
Efficient use of AND-OR structures	Noise sensitive	
—	Charge Sharing	→ Intermediate nodes may lose charge unless sized correctly



$Z = AG + (C+D)(E+F) + GH$

conventional CMOS → converted to dynamic CMOS logic





Initially $Q = C_{out} V_{DD}$

After charge sharing,

$V_1 = V_2 = V_{out} = V_f$

$Q = C_{out}V_{out} + C_1V_1 + C_2V_2$

$= (C_{out} + C_1 + C_2) V_f$

By charge conservation,

$C_{out} V_{DD} = (C_{out} + C_1 + C_2) V_f$

$$V_f = \frac{C_{out} V_{DD}}{C_{out} + C_1 + C_2}$$

Charge Sharing

→ If output is charged at 1 but gets connected to other internal nodes which have uninitialized / floating voltages causing charge to spread out & voltage drops

→ It is caused by long nMOS stacks & internal nodes with parasitic capacitances

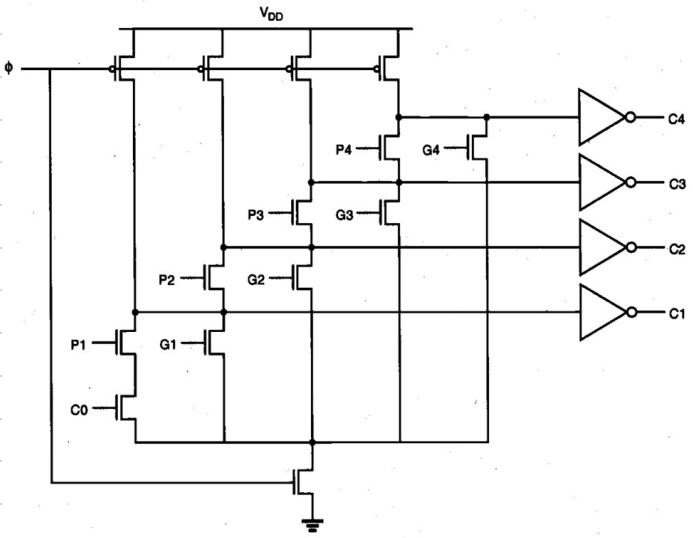
→ We can fix it by minimizing internal capacitances, use a slow weak pull-up pMOS and avoid deep nMOS stacks

Capacitive Coupling

→ When a dynamic node is physically placed next to switching signal, electric field from it can inject charge on node causing false switching

→ It is caused by high-speed wires close to sensitive dynamic nodes, dense layout, clock lines or switching signals nearby

→ It can be avoided by using shielding wires (GND or V_{DD} in between), create more space & add keeper to hold correct voltage



$= G_4 + P_4G_3 + P_4P_3G_2 + P_4P_3P_2G_1 + P_4P_3P_2P_1C_0$

$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1C_0$

$= G_2 + P_2G_1 + P_2P_1C_0$

$= G_1 + P_1C_0$

where $G_i = A_i \cdot B_i$

$P_i = A_i \oplus B_i$

C_1 : Output capacitance of precharge transistor

C_2 : Gate capacitance of pull-down network

C_3 : Capacitance to neighbouring wires (Crosstalk)

C_4 : Input capacitance of inverter (Load capacitance)

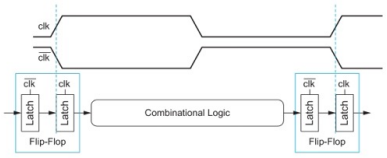
Unit - 4

Sequencing Static Circuits

→ Sequencing means using clocked storage elements to control the timing and order in which static logic blocks operate

→ The main reason for sequencing is that signals race through uncontrolled causing data to be lost or misaligned leading to circuit becoming non-deterministic & buggy

→ If sequencing is done right, we can move data (tokens) through logic blocks in controlled manner



↳ Flip flop as back to back latch

Sequencing Methods

→ There are 3 widely used methods of sequencing static logic

i) Flip flops

ii) 2 - Phase Transparent logics

iii) Pulsed Latches

→ **Flip - Flop based sequencing**

→ Flip-flop is placed at the boundary of each clock cycle and when rising edge occurs, input 'D' is sampled & sends to output 'Q'

→ Once sampled, data flows through combinational logic to next flip-flop and the next flip-flop samples the data on next rising edge

→ If data token arrives too early, it waits at flip-flop until next clock edge

Flip-flop can be viewed as 2 back-to-back latches,

one is transparent when CLK is high

other is transparent when CLK is low

→ **2 - Phase Transparent Latch**

→ Splits the clock into 2 non-overlapping phases (ϕ_1 & ϕ_2)

In one phase, Latch L1 is transparent & allows data to pass

In other phase, Latch L2 is transparent while L1 is opaque

→ This behaviour is analogous to canal lock gates where one gate is always closed

→ We can split the systems clock signal into 2 parts

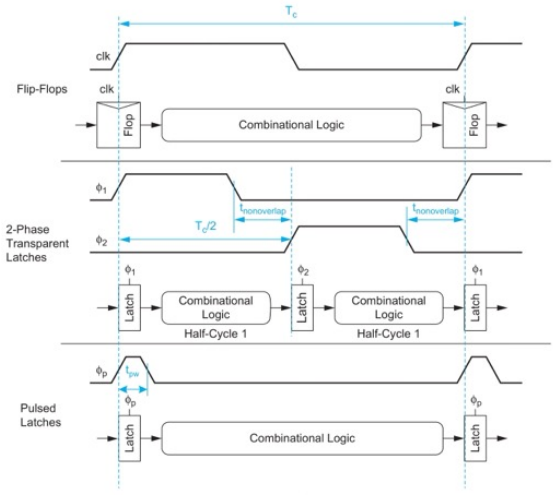
- Logic divided into 2 half-cycles
- 2 Latches never overlap transparency & controlled by small time gap '$t_{non-overlap}$'

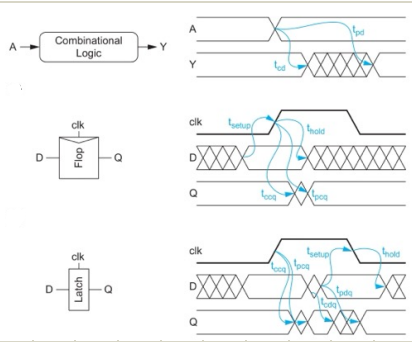
→ **Pulsed Latch**

→ Approach simplifies 2 - latch system by using 1 latch per cycle but instead of keeping the latch open during a full phase, it's made transparent for short pulse (t_{pw}) which is done by giving it a brief clock pulse (narrow high pulse)

→ Data moves forward only during that pulse & if pulse is shorter than time taken for data to go through logic, data can't move more than 1 stage but can let multiple stages if pulse width is long enough

→ Pulsed latches reduce area & power consumption because only 1 latch needed per stage





⇒ After input 'A' changes 'Y' might glitch or start changing after 't_{cd}'
Final value of 'Y' must stabilize by 't_{pd}' or else glitches occur

⇒ Input 'D' must be stable from 't_{setup}' before to 't_{hold}' after clock edge
Output 'Q' changes after 't_{ccq}' & settles by 't_{pcq}'

⇒ While CLK is high, output 'Q' tracks input 'D' & when CLK is low, that value is latched. During transparency, 'Q' changes after 't_{ccq}' and settles by 't_{pcq}'. After transparency, final value is latched after 't_{pdq}' / 't_{cdq}' depending on edge behavior

Term	Name
t_{pd}	Logic Propagation Delay
t_{cd}	Logic Contamination Delay
t_{pcq}	Latch/Flop Clock-to-Q Propagation Delay
t_{ccq}	Latch/Flop Clock-to-Q Contamination Delay
t_{pdq}	Latch D-to-Q Propagation Delay
t_{cdq}	Latch D-to-Q Contamination Delay
t_{setup}	Latch/Flop Setup Time
t_{hold}	Latch/Flop Hold Time
T_c	Clock Period

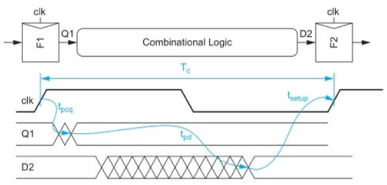
Max - Delay Constraints

→ In sequential circuit, we aim to complete all required logic operations within one clock cycle but sequencing elements introduce timing overhead which eats available time for computation

→ If the combinational delay is too large, receiving element misses it's setup time & samples wrong value which is called setup time failure (or) max - delay failure

→ To avoid this issue,
we must ensure the time from one output, through the logic, and into input of next element must be less than 1 clock cycle

→ We should analyse the available time & sequencing overhead for all 3 cases
i) Flip - flops
ii) 2 phase latches
iii) Pulsed latches



↳ Flip-flop max delay constraint

i) Flip-Flops

→ F_1 triggers on rising edge of clock & outputs data Q_1

This data flows through combinational logic reaching input D_2 of F_2, triggering next rising edge

→ Required condition, $T_c \geq t_{pcq} + t_{pd} + t_{setup}$

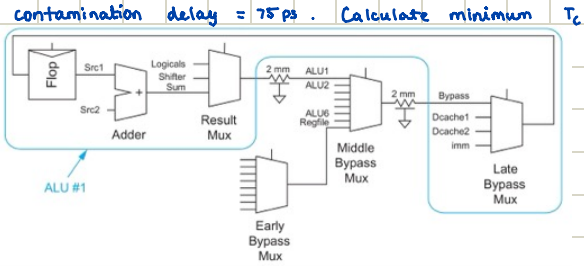
↳ Minimum Clock Period

$t_{pd} \leq T_c - (t_{pcq} + t_{setup})$

↳ Maximum allowed delay

$t_{pcq} + t_{setup}$: sequencing overhead

Q. In the circuit, the propagation delays & contamination delays are given. Suppose registers are built from flip-flops with setup time = 62ps, hold time = 10ps, propagation delay of 90ps & contamination delay = 75ps. Calculate minimum T_c at which it operates correctly



Element	Propagation Delay	Contamination Delay
Adder	590 ps	100 ps
Result Mux	60 ps	35 ps
Early Bypass Mux	110 ps	95 ps
Middle Bypass Mux	80 ps	55 ps
Late Bypass Mux	70 ps	45 ps
2-mm Wire	100 ps	65 ps

A. $t_{pd} = t_{adder} + t_{result\_mux} + t_{2mm\_wire} + t_{middle\_mux} + t_{2mm\_wire} + t_{late\_bypass} = 1000ps$

$T_c \geq t_{pcq} + t_{pd} + t_{setup}$

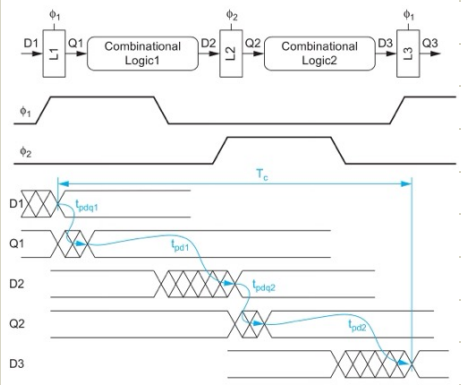
$\geq 90 + 1000 + 62$

$T_c \geq 1152ps$

ii) 2-Phase Transparent Latches

→ Logic split into 2 stages each bounded by transparent latches controlled by non-overlapping clocks ϕ_1 & ϕ_2

→ Data passes through L1 when ϕ_1 is HIGH, then passes through first logic block & reaches L2 during ϕ_2 and continues through second logic block to L3



$T_c \geq t_{pdq_1} + t_{pd_1} + t_{pdq_2} + t_{pd_2}$ → Min. Clock Period

$t_{pd} = (t_{pd_1} + t_{pd_2}) \leq T_c - (2t_{pdq})$

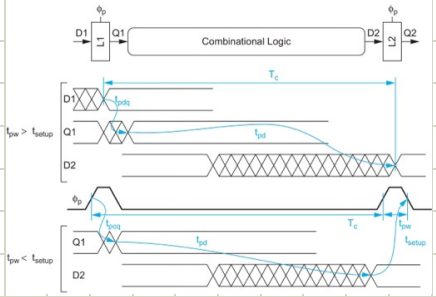
↳ Sequential Overhead

iii) Pulsed Latches

→ They use just one latch per cycle, controlled by short pulse

→ Latch is transparent for short pulse width (t_{pw})

Data must arrive during this pulse to be correctly latched



$$T_c \geq \max(t_{pdq} + t_{pd},\ t_{pcq} + t_{pd} + t_{setup} - t_{pw})$$

But if pulse width is wide enough to include setup time, then maximum logic delay is sequencing overhead

$$t_{pd} \leq T_c - \max(t_{pdq},\ t_{pcq} + t_{setup} - t_{pw})$$

Q. Recompute the ALU self-bypass path cycle time if flip-flop is replaced with pulsed latch. Pulsed latch has pulse width of 150 ps, setup time 40 ps, hold time 5 ps, clk to Q propagation delay of 82 ps, contamination delay 52 ps, D-to-Q propagation delay 92 ps

A. From diagram (Refer prev q.) $t_{pd} = 1000$ ps

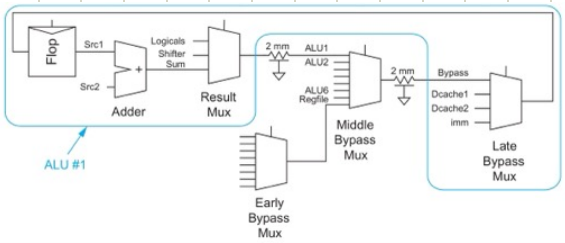
$t_{pdq} = 92$ps

$t_{pcq} = 82$ps, $t_{setup} = 40$ps, $t_{pw} = 150$ps

$T_c \geq \max(t_{pdq} + t_{pd},\ t_{pcq} + t_{pd} + t_{setup} - t_{pw})$

$\geq \max(1092\text{ ps}, 972\text{ ps})$

$T_c \geq 1092$ps



Min-Delay Constraints

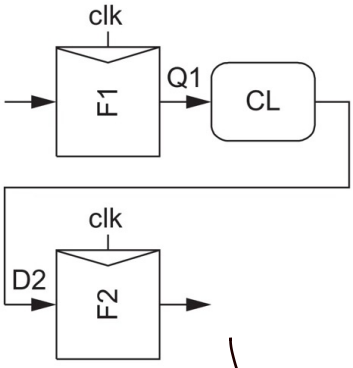
→ In max-delay we worried that data will arrive too late, now we worry about data arriving too early

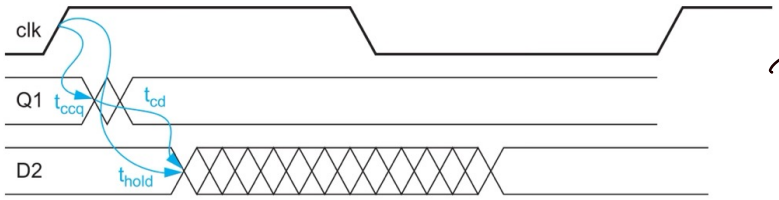
→ If data moves too fast through the logic & arrives before receiving element is ready, it can corrupt system state which is called hold time failure / race condition / min-delay violation

→ Unlike max-delay, we can't just adjust CLK to fix it. The entire circuit has to re-designed which is costly

→ To solve for minimum logic contamination delay,

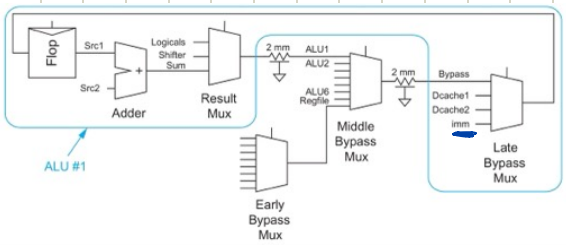
$$t_{cd} \geq t_{hold} - t_{ccq}$$





→ Flip-flop latch min-delay constraint

Q. In the ALU self-bypass circuit (same as prev q.), the earliest input to the late bypass multiplexer is the imm value coming from another flip-flop. Will the path experience any hold-time failures?



Element	Propagation Delay	Contamination Delay
Adder	590 ps	100 ps
Result Mux	60 ps	35 ps
Early Bypass Mux	110 ps	95 ps
Middle Bypass Mux	80 ps	55 ps
Late Bypass Mux	70 ps	45 ps
2-mm Wire	100 ps	65 ps

A. $t_{cd} = 45ps$

$t_{hold} = -10ps$ (-ve because value changes before clock edge, capturing right value)

$t_{ccq} = 75ps$

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

$45ps \geq -85ps \Rightarrow$ No hold-time failures

i) Min-delays in flip-flops

→ F1 triggers on rising edge → sends data to logic → data reaches F_2 but if t_{cd} is low, F_2 might latch the next value too early, violating t_{hold}

→ The only way to avoid is by ensuring $t_{cd} \geq t_{hold} - t_{ccq}$ is satisfied (Data doesn't arrive at F_2 earlier than t_{hold} after clock edge)

ii) Min-delays in 2-Phase Latches

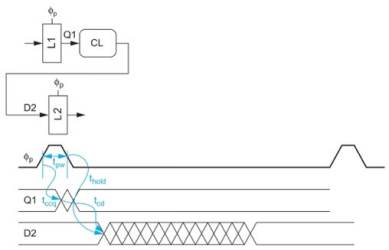
→ If we have latches L_1 & L_2 driven by ϕ_1 & ϕ_2 which are non-overlapping then delay must be long enough so that data doesn't smear through L_2 while it's still transparent

→ The minimum logic contamination delay through each phase of logic is

$$t_{cd_1}, t_{cd_2} \geq t_{hold} - t_{ccq} - t_{nonoverlap}$$

Where, $t_{non-overlap}$ = time when both clocks are low (Both latches be opaque)

↳ Usually we want it longer but many designs have CLK & $\overline{CLK}$ so $t_{non-overlap} = 0$ which



↳ Pulsed latch min-delay constraint

⚠ Paradox of flip-flops v/s Latches

→ Flip flops are made from 2 latches but need less delay to avoid hold time violations because of their design to avoid racing between internal latches, but if the latches are seperated, a delay must be added requiring more care

Q. If ALU self-bypass path uses pulsed latches in place of flip-flop, will there be hold-time problems?

A. Yes, $t_{cd} \geq t_{hold} - t_{ccq} + t_{pw}$

$t_{cd} = 45$ ps $t_{ccq} = 52$ ps

$t_{hold} = 5$ ps $t_{cd} = 150$ ps

Then $45 \not\geq 5 - 52 + 150 = 103$ ps

So path fails.

We can fix this by adding delay buffers after pulsed latch such that $t_{cd} \geq 103$ ps (or) replace pulsed latch with flip-flop

In summary!

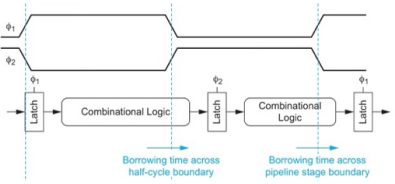
System	Hold time constraint	Implication
Flip-Flops	$t_{cd} \geq t_{hold} - t_{ccq}$	Safe for large t_{ccq}
2-Phase Latches	$t_{cd} \geq t_{hold} - t_{ccq} - t_{non-overlap}$	Needs more delay if no overlap
Pulsed Latches	$t_{cd} \geq t_{hold} - t_{ccq} + t_{pw}$	Easily violated for large t_{pw}

Time Borrowing

→ It is a technique used in latch-based systems where slow stage of logic can borrow time from next stage if it needs more time to complete

→ Flip-flops have hard timing edges, transparent latches provide soft boundaries

So if logic is too slow in 1 half cycle, it can keep running as long as next latch is closed



↳ Time Borrowing

→ In flip-flop, Data departs from F.F 1 at clock edge & must arrive and setup at FF2 before next clock edge. If late, it fails ⇒ No flexibility

Whereas in latch, Data departs L1 while its transparent & needs to arrive at L2 before it becomes transparent. This flexibility is called time-borrowing

→ From the diagram we can see first half, combinational logic finishes a bit late but since L2 is closed, the signal isn't lost & gets captured in next phase

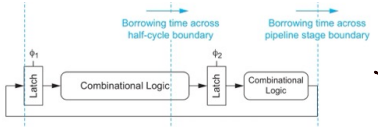
Time borrowing in feedback loops

→ Take a feedback loop, and if signal borrows time from next stage, then next stage starts later & so on.., Eventually causing too many delays hurting Throughput

→ Hence, only useful in internal balancing, but can't extend total cycle

ex: Self-bypass in an ALU

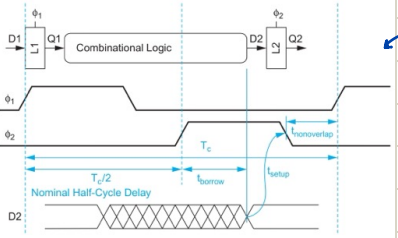
↳ Time is borrowed within cycle & but entire loop must still finish within 1 clock cycle



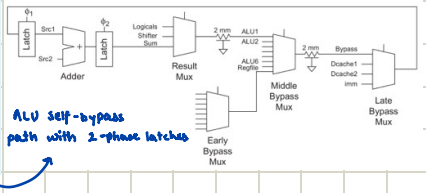
→ Max time that a stage can borrow

$$t_{borrow} \leq \left(\frac{T_c}{2}\right) - (t_{setup} + t_{non\text{-}overlap})$$

For more t_{borrow}, the lesser should be $t_{setup} + t_{non\text{-}overlap}$



Q. Suppose the ALU self-bypass path is modified to use two-phase transparent latches. A mid-cycle ϕ_2 latch is placed after the adder, as shown in Figure 10.14. The latches have a setup time of 40 ps, a hold time of 5 ps, a *clk*-to-*Q* propagation delay of 82 ps and contamination delay of 52 ps, and a *D*-to-*Q* propagation delay of 82 ps. Compute the minimum cycle time for the path. How much time is borrowed through the mid-cycle latch at this cycle time? If the cycle time is increased to 2000 ps, how much time is borrowed?



ALU self-bypass path with 2-phase latches

A. a) $t_{pdq_1} = 82$ ps

$t_{setup} = 40$ ps

$t_{adder} = 590$ ps $= t_{pd_1}$

$t_{all\ others} = t_{result\ mux} + t_{2mm} + t_{middle} + t_{2mm} + t_{late}$

$= 60 + 100 + 80 + 100 + 70 = 410$ ps

$= t_{pdq_2}$

$T_c = t_{pdq_1} + t_{pd_1} + t_{pdq_2} + t_{pd_2} = 590 + 82 + 410 + 82 = 1164$ ps

Element	Propagation Delay	Contamination Delay
Adder	590 ps	100 ps
Result Mux	60 ps	35 ps
Early Bypass Mux	110 ps	95 ps
Middle Bypass Mux	80 ps	55 ps
Late Bypass Mux	70 ps	45 ps
2-mm Wire	100 ps	65 ps

Nominal half cycle $= \frac{T_c}{2} = 582$ ps

Time taken by 1st half cycle, $t_{pdq_1} + t_{pd_1} = 590 + 82 = 672$ ps ⇒ Time borrowed $= 672 - 582 = 90$ ps

b) If $T_c = 2000$ ps, Nominal half cycle $= \frac{T_c}{2} = 1000$ ps ⇒ Time borrowing doesn't occur

Time borrowing in Pulsed Latches

→ Pulsed latches are like transparent latches with very narrow open windows
So they can borrow time - but only a small amount, if pulse width is wider than setup time

→ $t_{borrow} \leq t_{pw} - t_{setup}$

Q. If ALU self-bypass path uses pulsed latches, how much time will it borrow?

A. None, path is feedback loop which can't benefit from borrowing as it causes delay on next cycles hurting throughput.
There is no net gain in cycle time

Benefits of Time Borrowing

→ It has 2 benefits for system designer

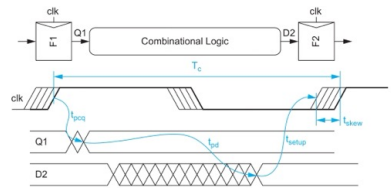
i) Intentional time borrowing
- → Used deliberately during design to balance pipeline or half cycle logic.
- → Avoids structural changes

ii) Opportunistic time borrowing
- → Happens automatically during manufacturing / runtime to absorb real-world timing variation.
- → Makes system more robust

Clock Skew

→ It is the difference in arrival times of clock signal at different parts of chip

→ Ideally, every flip-flop gets clock signal at exact same time
But in reality, wires have resistance, capacitance, delays
different clock paths take different routes
buffers inserted for strength & introducing delay
All these reasons cause clock edges to arrive earlier / later at different registers or latches



→ Impact of Clock skew on max delay (setup)

→ Assume Clock gets to F1 a little late
and Clock gets to F2 a little early

→ In this worst case, available time to complete logic computation is reduced

$T_c \geq t_{pd} + \underbrace{t_{pcq} + t_{setup} + t_{skew}}_{\text{sequencing overhead}}$

→ This makes max delay constraint tighter

→ Impact of Clock skew on min delay (Hold)

→ Assume Clock gets to F1 a little early
and Clock gets to F2 a little late

→ This makes it more likely that data arrives too soon causing hold-time failure

$T_c \geq t_{hold} - t_{ccq} + t_{skew}$

→ Clock skew effectively increases required hold time & designers must ensure contamination delay t_{cd} is long enough

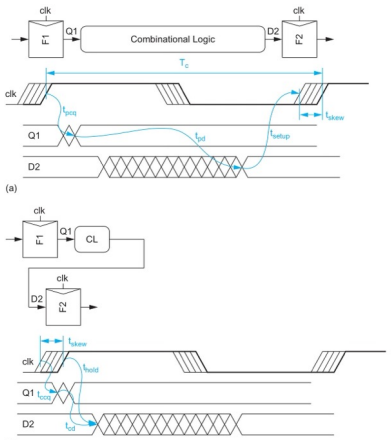


FIGURE 10.15 Clock skew and flip-flops

→ Clock skew in latch based system

→ Skew doesn't reduce max timing in latch-based systems because data passed through late when they're transpalent & timing is more relaxed

→ Available time = T_c − (2 latch delays)

↳ No need to substract t_{skew}

But this doesn't mean clock skew is inaffective. It affects hold time

$t_{cd} \geq t_{hold} - t_{ccq} + t_{non\text{-}overlap} + t_{skew}$

It also reduces window for time borrowing

$t_{borrow} \leq \left(\frac{T_c}{2}\right) - (t_{setup} + t_{non\text{-}overlap} + t_{skew})$

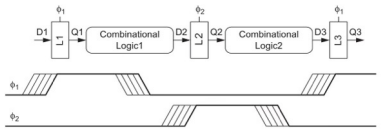


FIGURE 10.16 Clock skew and transparent latches

Q. If ALU self-bypass can experience 50 ps of skew from one cycle to next b/w flip-flops in various ALUs, what is minimum cycle time of system? How much clock skew can system have before hold-time-failure?

A. $t_{pd} \leq T_c - (t_{pq} + t_{setup} + t_{skew})$

$T_c \geq t_{pd} + t_{pq} + t_{setup} + t_{skew} = 90 + 1000 + 62 + 50 = 1202$ ps

Cycle time should increase from 50ps to 1202ps

Max skew $= t_{cd} - t_{hold} + t_{ccq}$
$= 45 - (-10) + 75$
$= 130$ ps

→ Clock skew in pulsed latch based system

→ They can tolerate amount of skew proportional to pulse width
If pulse is wide enough, skew won't increase sequencing overhead because data can arrive when latch is transparent

→ Skew also increases hold time & reduces amount of time for borrowing

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

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

$t_{borrow} \leq t_{pw} - (t_{setup} + t_{skew})$

Semiconductor Memory Arrays

→ Modern digital systems require large volumes of digital storage These are implemented using semiconductor memory which is built from transistors on a chip

→ In most systems 90% of chip area is devoted to memory blocks because of this, design of memory cells is a critical part of VLSI design

→ On-chip memory integration

→ As process technology has scaled, its possible to integrate entire memory arrays on-chip, rather than using external memory

→ Modern chips contain tens of millions of memory bits

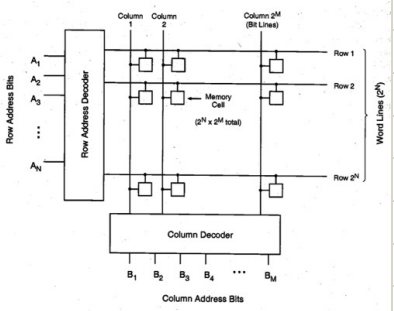
→ Memory-Dominated Design

→ Memory dominates area, power & performance constraints, so, logic designers must often work around timing & placement of memory

→ Memory can be classified into

SEMICONDUCTOR MEMORIES

Read Only Memory	Random Access Memory
i) Mask Programmed	i) SRAM (S-Static)
ii) Programmable ROM	ii) DRAM (D-Dynamic)
iii) Erasable PROM	
iv) Electrically EPROM	



Memory Array Structure

→ Most bits are organised as 2D arrays of bits

⇒ Rows: Controlled by word lines

⇒ Columns: Accessed through bit lines

ex:

N bits for row address → selects 2^N rows

M bits for column address → selects 2^M columns

→ Memory stores : $2^N \times 2^M = 2^{N+M}$ bits

total address bits = N + M

DRAM (Dynamic RAM)

→ DRAM is comparitively better than SRAM

i) High intensity integration is key in modern memory design

ii) 6T SRAM cells are too large for very dense memory array

iii) DRAM stores data as charge on capacitor instead of full latches

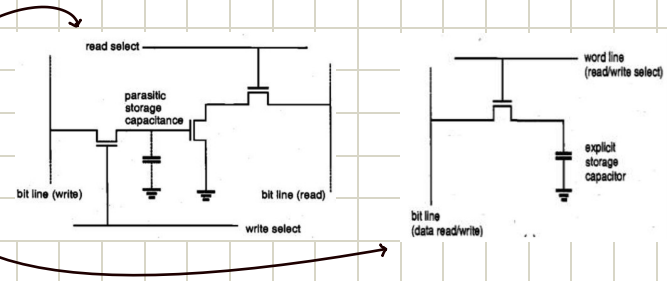
iv) DRAM are much smaller, supporting more bits per chip area

→ Even though DRAM stores charge in capacitor, DRAM can't retain data indefinitely because leakage current modifies / removes stored charge

DRAM evolution

1st Generation → 3T DRAM cells

later Generation → 1T DRAM cells



3T DRAM cell

→ It includes pre-charge circuit, 3T DRAM cell, Read / Write circuit

→ Consists 2 bit lines : D_{in} Writing

D_{out} Reading

2 word lines : W_s Write Select

R_s Read Select

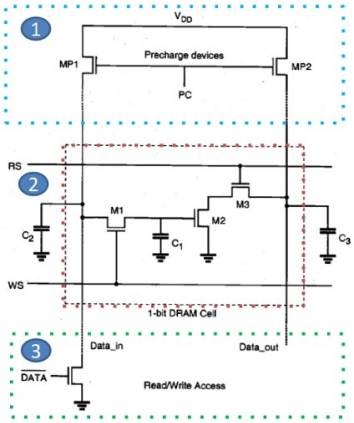
4 Transistors : M_1 Write access

M_2 Storage Gate

M_3 Read access

M_D For writing '0'

2 Clock Phases : ϕ_1 Precharge Phase & ϕ_2 Read / Write Phase



Write '1'	Read '1'	Write '0'	Read '0'
Data = 1 ⇒ Data' = 0 ⇒ M_D = OFF	RS. ϕ_2 = 1 ⇒ M_3 = ON	Data = 0 ⇒ Data' = 1 ⇒ M_D = ON	RS. ϕ_2 = 1 ⇒ M_3 = ON
D_{in} = HIGH	M_3 = ON ⇒ C_3 discharges	D_{in} = LOW	M_2 = OFF ⇒ No discharge
WS. ϕ_2 = 1 ⇒ M_1 = ON	Drop on D_{out} = logic '1'	WS. ϕ_2 = 1 ⇒ M_1 = ON	C_3 = HIGH ⇒ D_{out} = logic '0'
C1 gets charged via C_2		C_1 discharges ⇒ M_2 = OFF	
M_2 = ON			

→ Refresh Operation

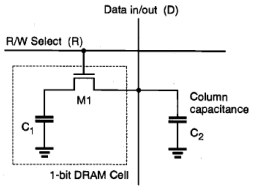
→ Charge on C_1 can't be stored forever, and leaks due to M_1 drain leakage

→ So data must be periodically read, inverted & written back is same cell location

→ Refresh frequency - every 2-4 ms

→ It is performed row-wise for higher efficiency

NOTE :
data is inverted because o/p level reflects inverse of stored data



1T DRAM Cell

→ Consists of 1 Transistor & 1 Capacitor
Stores data as charge in capacitor

→ Smallest & widely used DRAM cell (Ideal for high density memory)

→ Components

C_1 : Storage Capacitor C_2 : Column Capacitance

M_1 : nMOS access transistor

WL : Word Line - enables access to cell

BL : Bit Line - used for read & write

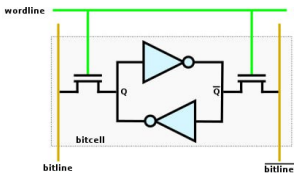
→

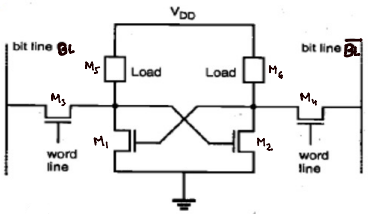
Write '0'	Write '1'	Read
BL Charged to 1	BL discharged to 0	BL is precharged to $\frac{V_{DD}}{2}$ an intermediate voltage
WL = HIGH M_1 = ON	WL = HIGH M_1 = ON	WL = HIGH M_1 = ON
Charge stored in C_1	C_1 discharges via M_1	If $C_1 = 1 \Rightarrow$ BL rises
C_1 holds '1'	C_1 holds '0'	If $C_1 = 0 \Rightarrow$ BL drops
		Sense amplifier detects small changes

⚠ Read is destructive
Capacitor loses charge
Data must be re written

→ Charge Sharing occurs b/w C_1 & C_2
If $C_1 = 0 \Rightarrow$ Fall in C_2 is detected by sense amplifier as '0' is stored in DRAM cell
If $C_1 = 1 \Rightarrow$ Rise in C_2 is detected by sense amplifier as '1' is stored in DRAM cell

→ Similar to 3T DRAM, Refresh Operation is done





6T SRAM

→ A Static memory cell which retains data as long as power is applied

→ Unlike DRAM, refreshing isn't required

→ 6 Transistors : 4 Transistors → Bistable Latch (Memory)
2 Transistors → Switches (Access gates for read & write)

→ It has faster read & write operations, not designed for density

→ Operations

i) Write

→ WL = HIGH ⟹ M_3 & M_4 : ON

→ For writing '1' : BL = 1, $\overline{BL}$ = 0
For writing '0' : BL = 0, $\overline{BL}$ = 1

→ M3 & M4 connect internal storage nodes to BL & $\overline{BL}$
Strong bit lines override the internal inverter state to force new value

ii) Hold

→ When WL is OFF, M3 & M4 are cut off
Cell is isolated from bit lines & cross-coupled inverters maintain their state until next read or write

→ No refresh required like DRAM

iii) Read

→ Bitlines BL & $\overline{BL}$ are precharged to V_{DD}

→ WL is HIGH connecting cell to bit lines

→ Depending on stored value,
Read '1' : Internal node connected to BL is HIGH
$\overline{BL}$ is pulled LOW via pulldown path
Read '0' : Internal node connected to BL is pulled LOW
$\overline{BL}$ remains HIGH

→ Sense amplifier detects small difference b/w BL & $\overline{BL}$ and amplifies to logic level

Design Constraints

→ Cell ratio - Ratio of Pull down transistor to access transistor - affects read stability

→ Access ratio - Ratio of access transistor to Pull down transistor - affects write stability

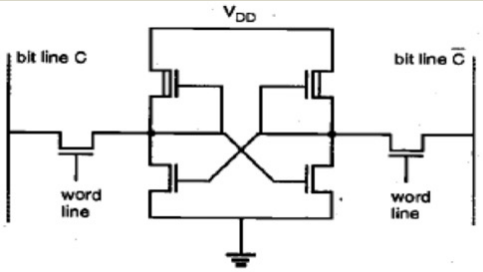
Trade - Offs

→ Increased Stability - harder to write

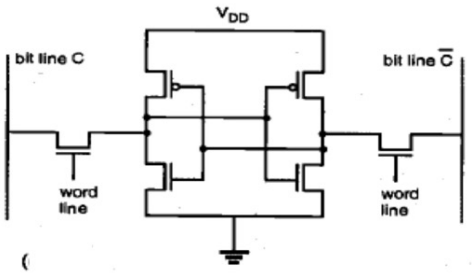
→ Faster access time - More power & area

→ High density - Smaller transistors, less NM

Depletion Load SRAM



Full CMOS SRAM



Design Strategy

→ To determine (W/L) :

i) Read shouldn't destroy saved data

ii) Cell shall allow modification during write

Case read '0' :

$$\frac{K_{n_2}}{K_{n_1}} = \frac{(W/L)_2}{(W/L)_1} < \frac{2(V_{DD} - 1.5V_{T,n})V_{T,n}}{(V_{DD} - 2V_{T,n})^2}$$

Case write '0' :

$$\frac{(W/L)_2}{(W/L)_1} < \frac{\mu_n}{\mu_p} \cdot \frac{2(V_{DD} - 1.5V_{T,n})V_{T,n}}{(V_{DD} + V_{T,p})^2}$$

Layout