Fast multiplier using CSA

7	6	5	4	3	2	1	0 → weight
a_4b_3	a_4b_2	a_4b_1	a_4b_0	a_3b_0	a_2b_0	a_1b_0	a_0b_0
a_3b_4	a_3b_3	a_3b_2	a_3b_1	a_2b_1	a_1b_1	a_0b_1	
	a_2b_4	a_2b_3	a_2b_2	a_1b_2	a_0b_2		
		a_1b_4	a_1b_3	a_0b_3			
			a_0b_4				

circled bits denote the possibility of reducing 3 to 2 using a CSA (For columns 1, 4 & 7, third i/p = 0)

We can do all this at once & continue until we have a final no.

→ Till now, the circuits used have no memory.

For ~~mem~~ memory,

$$\{x_k\}_{k=0}^{\infty} \longrightarrow \boxed{} \longrightarrow \{y_k\}_{k=0}^{\infty}$$

(k can be seen as time)

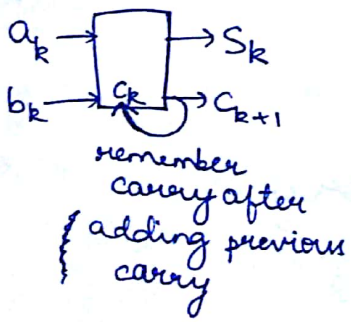
For memoryless, $y_k = f(x_k)$

Eg → remembering whether no. of 1's seen so far is odd or even

→ The loops implemented before were done in a combinational way.

Now, we'll implement using a sequential way.

Eg → addition



[Sequential circuits]

→ Here, we need to remember a 1-bit information

→ Usually, we can determine ~~a~~ the min. no. of k-bit memory specification required for a sequential circuit (finite memory).

eg of ∞ memory ⇒ o/p $y_k = 1$ if $\overline{x_0 \ldots x_{k-1}}$ is a palindrome

(can't be implemented)

→ We can implement only causal systems.

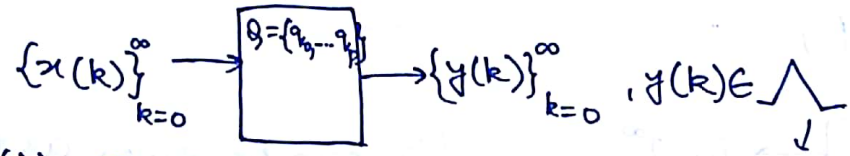
Eg of non-causal ⇒ $y_k = x_{k+1}$

⇓

can't be implemented in real life.

Mealy

Finite State Machine $(Q, \Sigma, \Lambda, \delta, \lambda, q(0))$



$x(k) \in \Sigma$ → set of symbols (finite)

set of symbols (finite set)

Index 'k' denotes time.

Q = finite set of states (inside the system)

$q(k)$ ⇒ state of the system at time 'k'.

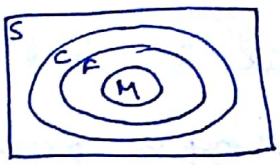
For mealy machine model,

$$y(k) = \lambda(x(k), q(k))$$

$$\& \quad q(k+1) = \delta(x(k), q(k))$$

$$q(k+1) = \underbrace{nq}_{\text{new variable (next q)}}(k)$$

States are a model for memory.



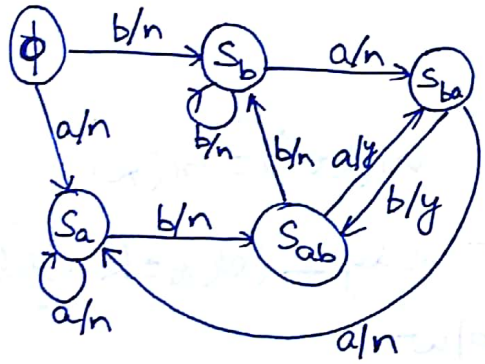
S = sequential
C = causal
F = Finite state
M = representable by this model)

Though M is a small part, it is enough

eg: $\Sigma = \{a, b\}$, $\Lambda = \{y, n\}$

specification ⇒ 'y' if last 3 entries are either aba or bab

Need to remember 2 bits of info.
Max. possible states ⇒ ϕ, a, b, 4 others
ϕ → nothing seen so far



⇒ 5 states required

⇒ State Transition Graph (STG)

$\delta(a, \phi) = S_a$; $\delta(b, S_a) = S_{ab}, \ldots$
$\lambda(a, \phi) = n$; $\lambda(a, S_{ab}) = y, \ldots$

Simulation

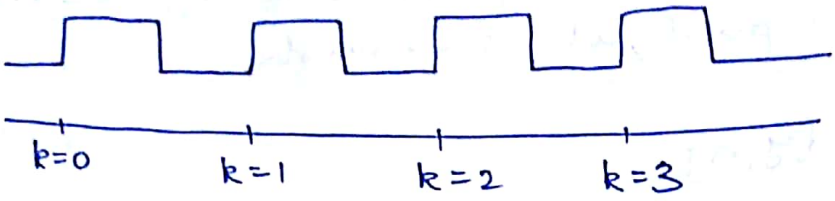
$y(k) \Rightarrow$	n	n	n	y	n	n
$q(k) \Rightarrow$	ϕ	S_a	S_a	S_{ab}	S_{ba}	S_a
$x(k) \Rightarrow$	a	a	b	a	a	b
k	0	1	2	3	4	5

Input ($x(k)$) and $q(k)$ are specified.

Regular expression

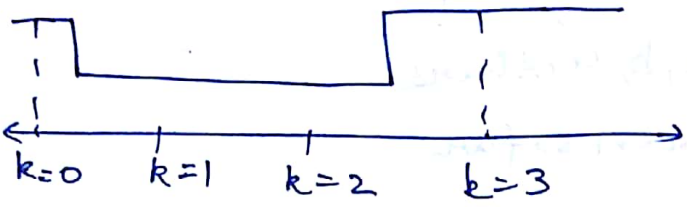
$a^* b^*$ ⇒ any no. of a's followed by any no. of b's.
$a^k (ab)^* b^l$ ⇒ 'k' no. of a's followed by any no. of (ab)'s followed by 'l' no. of b's.

For implementation, we need a clock to define time instants. We do this using a periodic waveform



Rising edge times defined as k's

Now, given,

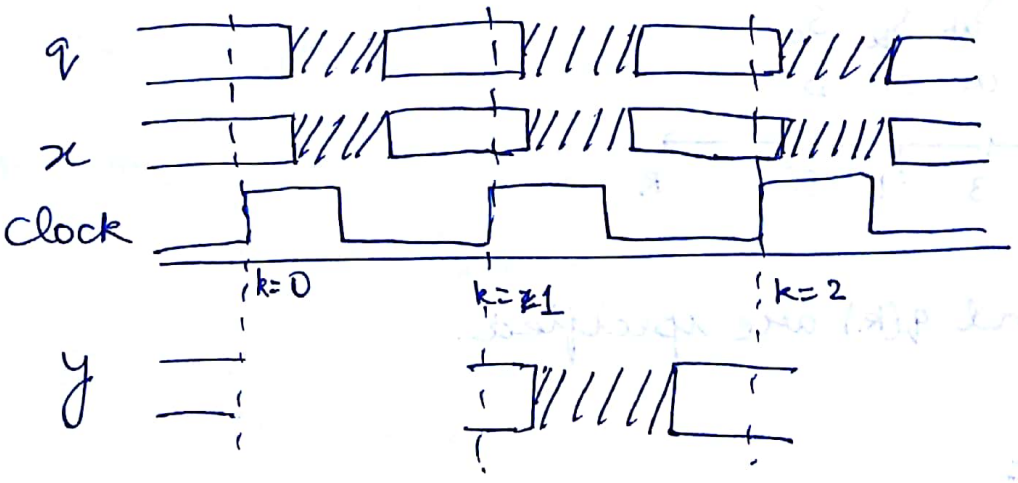


$x(0)=1$ $x(1)=0$ $x(2)=0$ $x(3)=1$

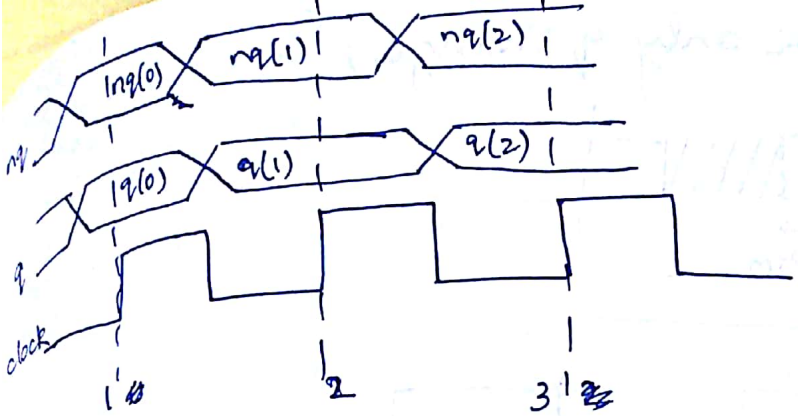
If we want to implement $w(k+1)=x(k)$

We need to match 'w' at edges (clock cycles) only. Doesn't matter in b/w.

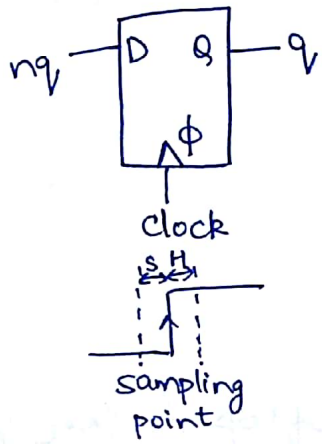
To define $x(k)$ & $q(k)$ unambiguously, they should be well defined at the rising edges of clock.



∴ a clock interval must be large enough to allow all signals to settle down. Sampling occurs at rising edge of clock.



circuit to do this : <u>(D-Flipflop)</u>



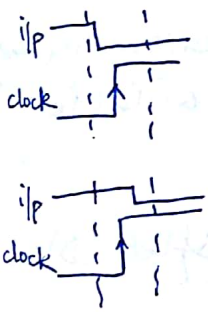
Delay D = time taken for 'q' to respond after sampling 'nq'.

The signal should be stable on an interval around sampling pt.

S = setup time

H = Hold time

Input should be stable in S+H time.

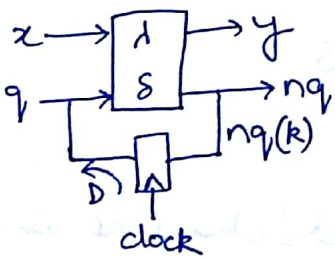


⇒ The change may not be read

⇒ Value not held for enough time

Not sure whether o/p = 0 or 1

FSM looks like:



→ multiple flipflops for multiple bits

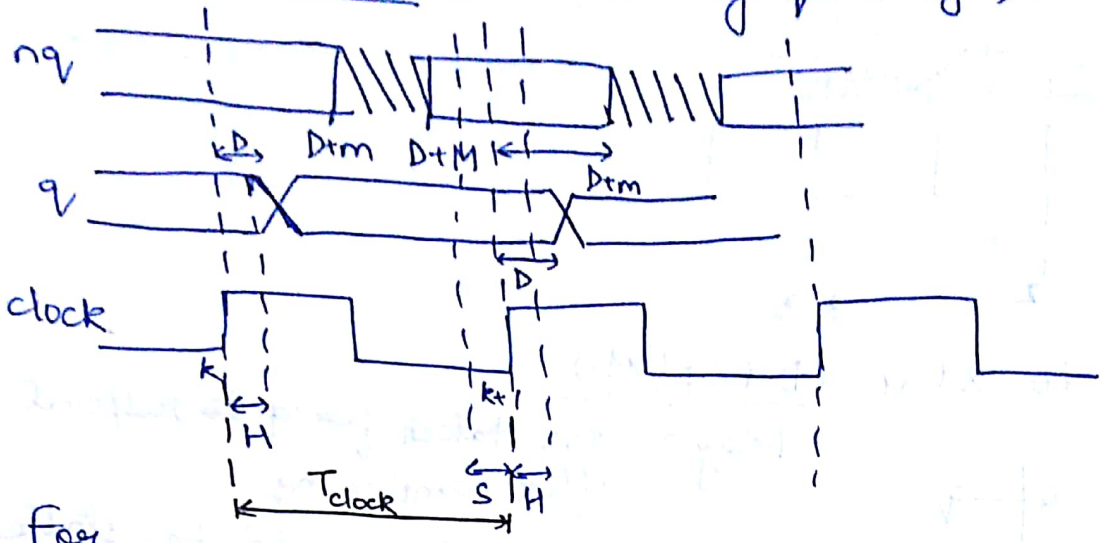
We need to know the min. sufficient period of clock s.t. i/p sampled are correctly stable.

We define m & M for δ & S circuit:

m = smallest delay from o/p to i/p

M = largest ——————

Clock period (assume only q changes)



For correct operation,

$$D+m \geq H \quad \text{——— (1)}$$

$$D+M \leq T_{clock} - S \quad \text{——— (2)}$$

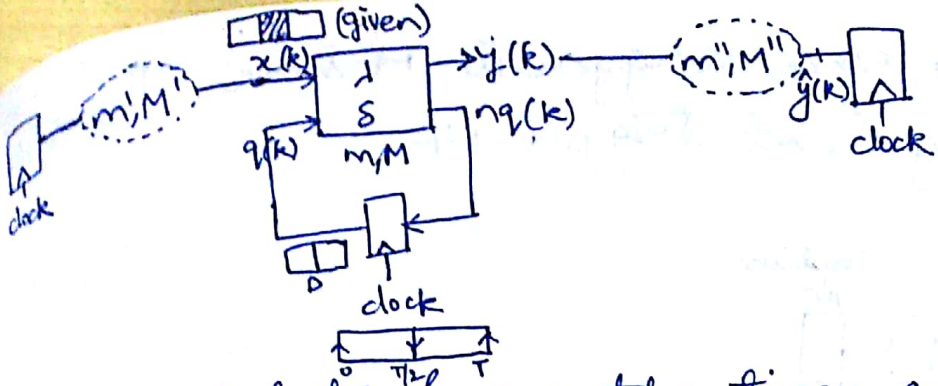
Note ⇒ The sampling for ~~flipf~~ flipflop is for 'nq'.
∴ we do not care if $D > H$ or not because q is not sampled for the flipflop.
After nq is sampled, q changes after D delay. (Doesn't matter what H is)
eqn (1) is more imp. because it is not controllable from outside the circuit.

☆ From hereon, we include the delay of flipflop (D) in m & M.

∴ above eqn become $m \geq H$
& $M \leq T_{clock} - S$

☆ $x[k]$ ⇒ synchronous
means, we know the interval where $x[k]$ changes in a clock period.
⇒ essential
⇒ time characteristics related to T_{clock}
⇒ if not, then circuit may change

We can say that $x[k]$ comes from another flipflop with same clock.

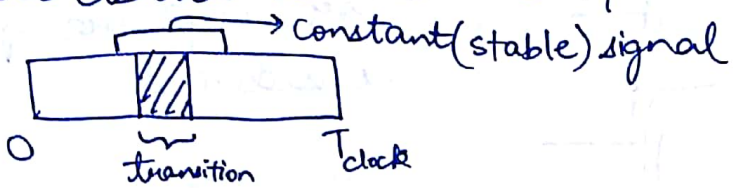


Here, we need to know the time characteristics of $y(k)$ also as it may be further used in another flipflop & we want correct sampling (say $\hat{y}(k)$).

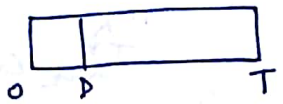
signature of a signal:

→ characteristics in a clock period

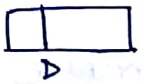
Eg ⇒



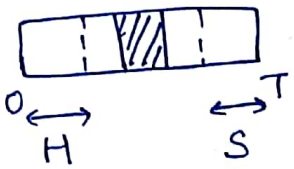
signature of a flipflop ⇒



D ⇒ may consist of D_{min} & D_{max}

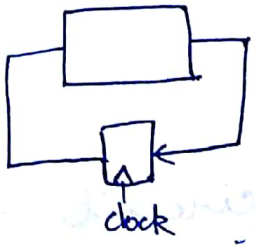
We find signature of $nq(k)$ using  & $x(k)$

nq ⇒



Transition period ~~b/w~~ within S&H limits of rising edges.

∴ For time considerations, FSM can be modelled as:

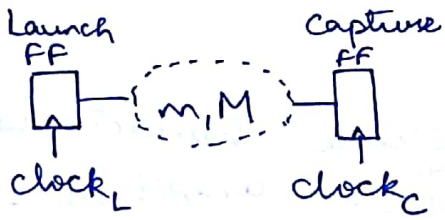


signal starts at a flipflop (called launch flipflop) & ends at a flipflop (called capture flipflop). Both may not be the same.

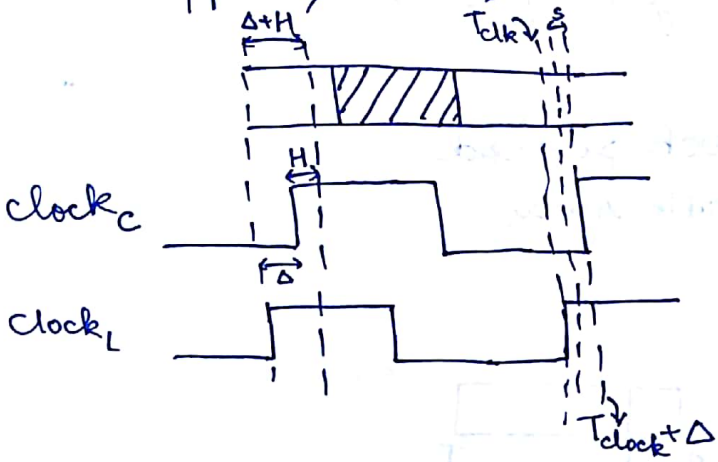
The clock of both the FF may not be the same.

* We are using edge-triggered FF here. There is another using level trigger FF.

Let,



Suppose, there's a skew of Δ b/w clocks.



Hold time & setup time for capture flipflop considered.

$\therefore m \geq H+\Delta$ ——— ①

& $M \leq (T_{clock}-S)+\Delta$ — ②

Now, ① is more difficult to meet than in previous case but ② is more relaxed.

<u>If $\Delta<0$</u>

① is somewhat relaxed

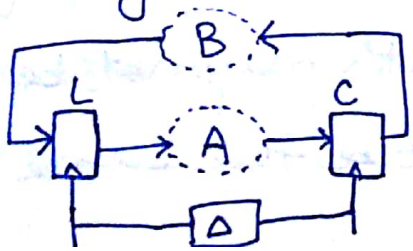
② is difficult to meet

① & ② can be written as:

$m-\Delta \geq H$

$M-\Delta \leq (T_{clk}-S)$

$\Delta \Rightarrow$ altering m & M of ~~clock~~ circuit



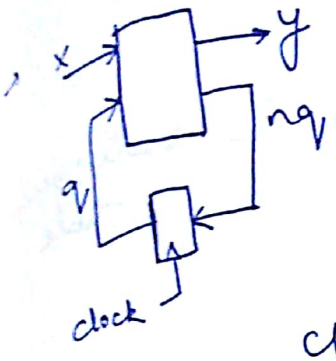
If $\Delta = 1ns$ (let)

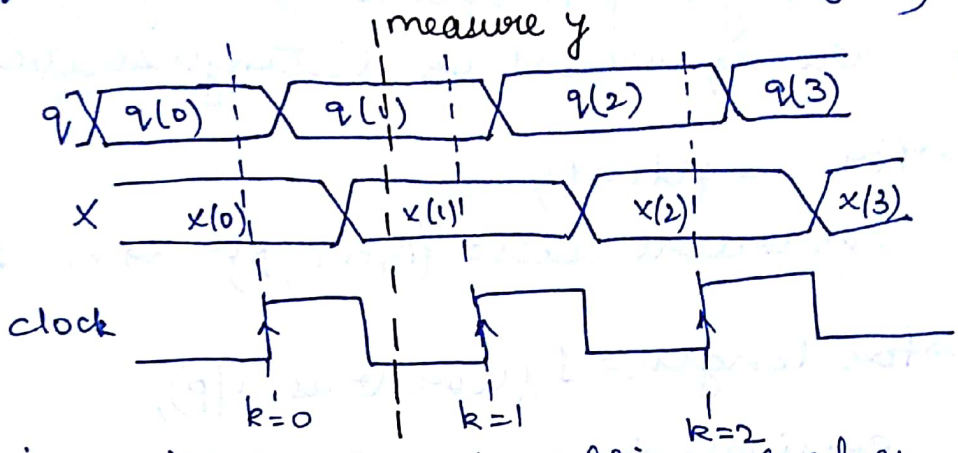
New M_1' of A can be (M_1+1)

New M_2' of B can be (M_2-1)

} using skew

If Δ = not specified with dir^n ⇒ both ~~will~~ should be (M−1)





changing i/p is done at falling ~~at~~ edges of clock ~~to~~ prevent setup & hold time error.

∴ Testbench pattern

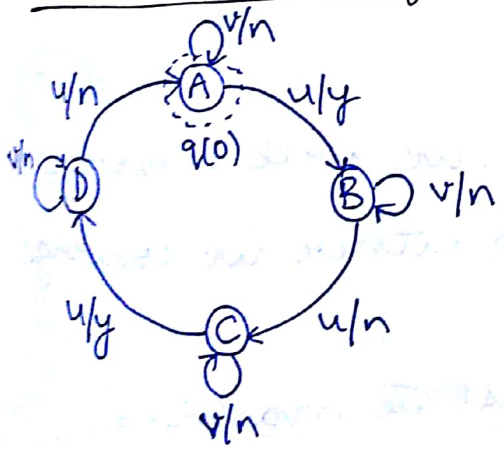
clock=1 ⇒ x=stable (rising edge)

clock=0 ⇒ X=changing (falling edge)

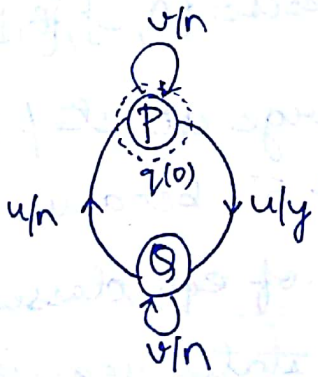
For y, we measure after some delay at falling edge of clock because there, q has not changed & is still q(k) but x changes from x(k−1) to x(k)

∴ here $\underbrace{\text{after some delay}}_{\text{because we want x(k−1) to have changes to x(k).}}$, y=y(k)

Minimisation of FSM



≡



Hence, minimized.

Equivalence of ~~both~~ the two state machines on the basis of properties (o/p sequence ~~to~~ an i/p sequence).

We ~~try~~ try to find i/p (length) where two states are equivalent or distinguishable.

→ For length = 0,
equivalent class = {A, B, C, D} ⇒ no pair is distinguishable

→ For length = 1 (u or v as i/p),
equivalent classes = {A, C}, {B, D}

→ For length = 2 ($x_0 x_1$)
Consider A, C & i/p $x_0 x_1$
Till x_0, A & C → equi.
Now, $A \xrightarrow{x_0} nA$, $C \xrightarrow{x_0} nC$ } If nA & nC are 1 eqv., then A & C are 2 eqv.

equivalent classes = {A, C}, {B, D}

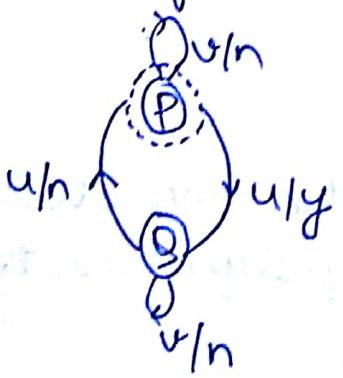
→ For length = 3 ($x_0 x_1 x_0$),
let $A \xrightarrow{x_0} nA$, $C \xrightarrow{x_0} nC$ } If nA & nC are 2 eqv., A & C are 3 eqv.

eqv. classes = {A, C}, {B, D}

We converge at step 2 only. Max. we will converge till step 5 because ~~of~~ at each step, either we converge ~~of~~ or no. of eqv. classes ↑

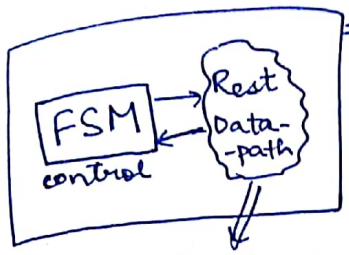
∴ max. 'n' states required for 'n' state machine.

∴ A, C = eqv. = P
B, D = eqv. = Q



Part 2

→ FSM is used for control.

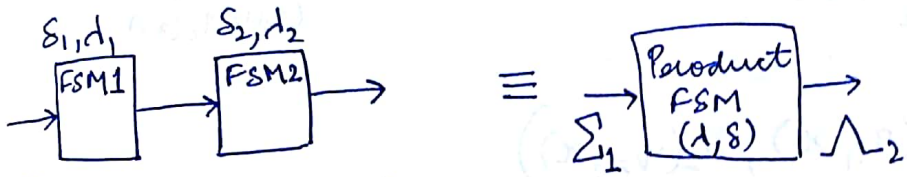


→ Also a state machine (with finite no. of states, finite memory)

Rest of the computation takes place here.

Building FSM

(i) Cascade form



Σ_1 Q_1 Λ_1 Q_2 Λ_2

Σ_2

$q_0^{(1)}$ $q_0^{(2)}$

⇓

initial states

$Q_1 \times Q_2$

$(q_0^{(1)}, q_0^{(2)})$

⇓

initial state

(Set of a states is a subset of $Q_1 \times Q_2$)

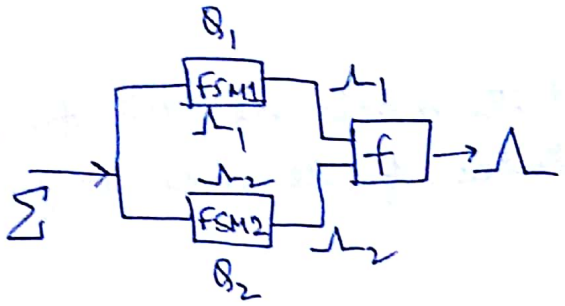
$$\delta((q_1, q_2), x) = (\delta_1(q_1, x), \delta_2(q_2, \lambda_1(q_1, x)))$$

$$\lambda((q_1, q_2), x) = \lambda_2(q_2, \lambda_1(q_1, x))$$

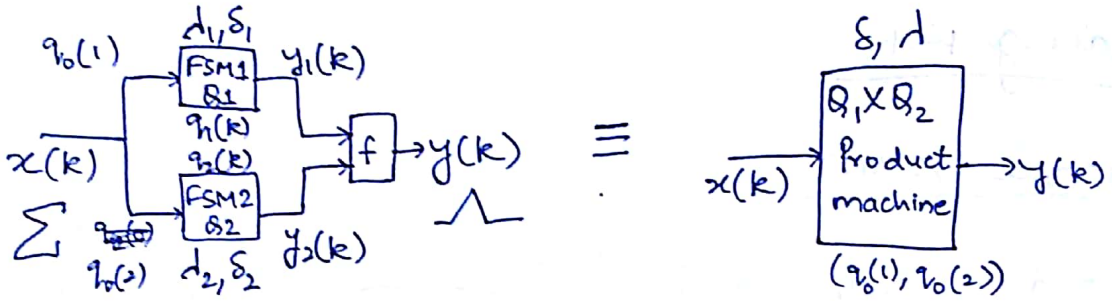
→ Product FSM helps in verifying if the break up into FSM 1 & 2 is correct

→

(ii) Parallel form



If $f = XOR$ & Λ is never 1, then FSM 1 & 2 are equivalent.



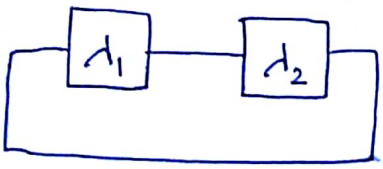
$$\delta((q_1, q_2), x) = (\delta_1(q_1, x), \delta_2(q_2, x))$$

$$\lambda((q_1, q_2), x) = f(\lambda_1(q_1, x), \lambda_2(q_2, x))$$

f = combinational func^n

Starting from $(q_0(1), q_0(2))$, we don't ~~need to~~ necessarily reach every state of $Q_1 \times Q_2$.

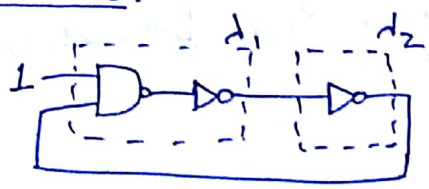
★



→ generally combinational loop

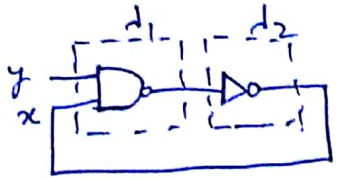
2 cases:

(i)



Oscillates b/w 1 & 0

(ii)

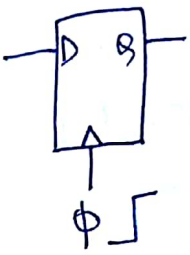


⇒ x can be forced to 0 (fix $y = 0 \Rightarrow x = 0$) then, $y = 1$, $x = 0$ (can store a bit)

$(y = 1, x = 0/1) \Rightarrow$ 2 possible

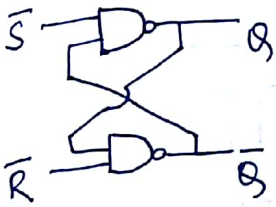
We generally don't want any combinational loops in our circuit due to strange behaviour

D-Flip Flop



We need to remember 1 bit of information. So, we need a combinational loop.

Claim: Without loop, ~~cir~~ combinational circuit won't have any memory.

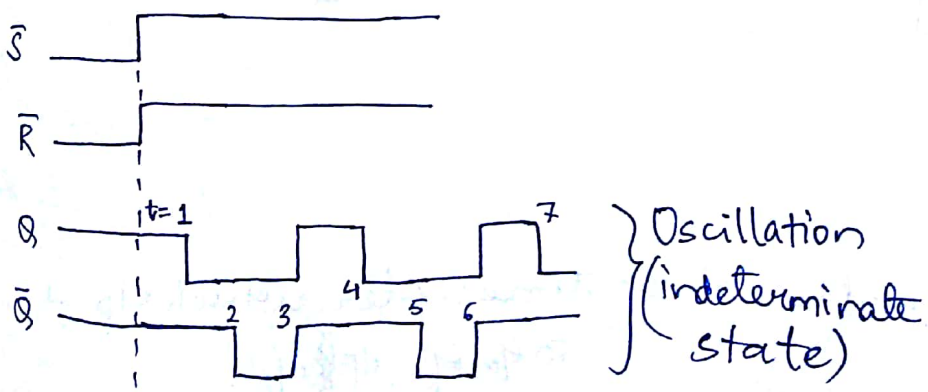


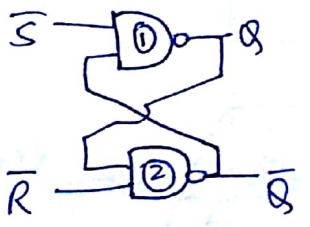
→ • has 1 bit of memory
• SR bistable
• $\bar{S}, \bar{R}$ means ~~set~~ inputs are "active low" (have effect when low)

	$\bar{S}$	$\bar{R}$	Q_n	$\bar{Q}_n$
(indeterminate)	0	0	1	1
(SET)	0	1	1	0
(HOLD)	1	1	Q_{n-1}	$\bar{Q}_{n-1}$
(RESET)	1	0	0	1

→ view $Q_n, \bar{Q}_n$ as symbols (Excitation Table)

Indeterminate case is problematic because if the next i/p are ($\bar{S}=1, \bar{R}=1$) (hold state), then $Q_n, \bar{Q}_n$ become indeterminate - either 0,1 or 1,0 or 0,0

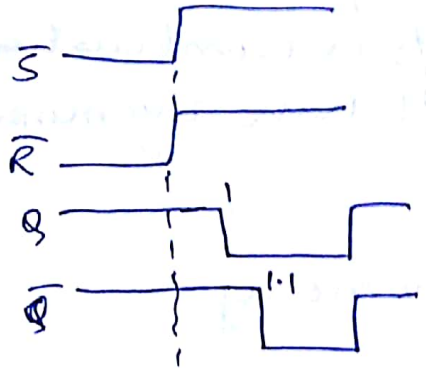




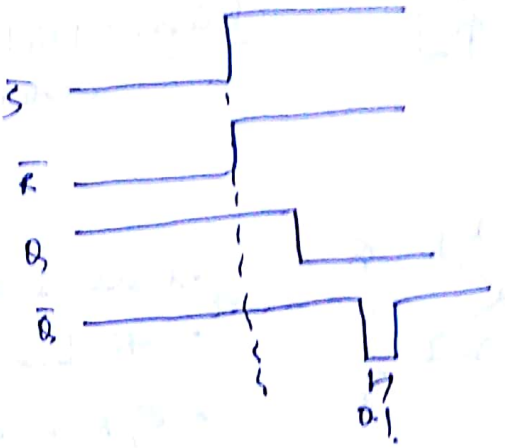
1, 2 = delays

Depends on delays of gates. This delay model is not accurate, the fall won't be like
It would be like: ,
∴ it'll settle at some point

If delay is 1 & 1.1



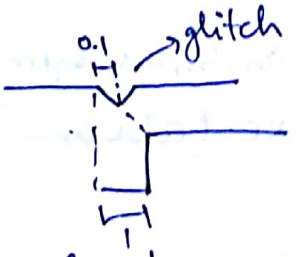
If delay is 0.1 & 1



A more realistic model of delay is:-

delay=1

$=l$



To see whole transition, $l \geq$ delay
thus in prev. case, $\bar{Q}=1$ & $Q=0$

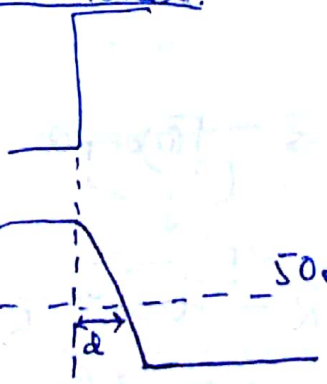
Delay model

delay=1

i/p =

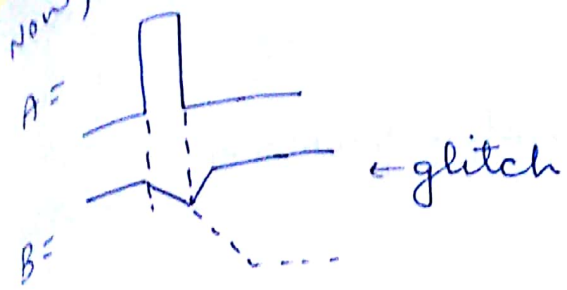
Ideal model o/p =

Another model:-

A = 

d = time after which o/p is 50% of i/p or o/p.

Now,



Wider the pulse, deeper the glitch.

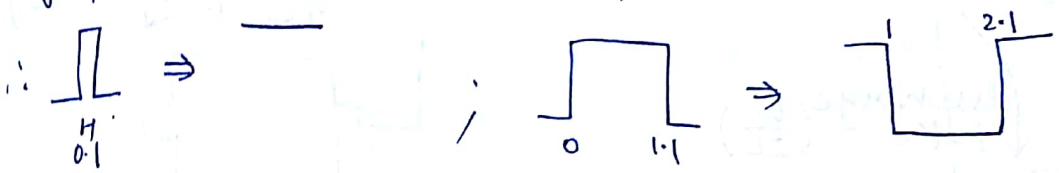
Model ⇒ this glitch is considered a disturbance when the lower point touches the midline of o/p

∴ if pulse width < d, o/p does not respond to i/p.

⇓

Inertial delay model

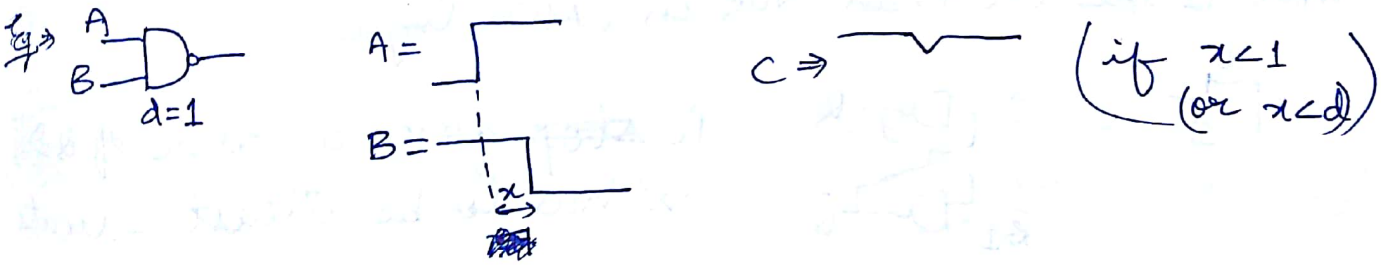
If pulse width > d, o/p responds

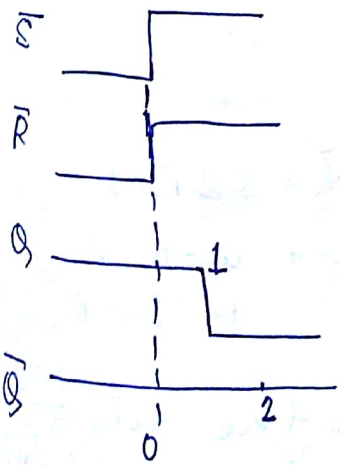


Transport delay

o/p is just the delay shifted version of i/p.

Eg ⇒ Optic fibres





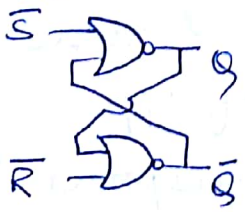
$Q = \overline{\bar{S}.\bar{Q}}$

→ Not responding as change in Q, $\bar{R}$ not held for t>2 seconds.

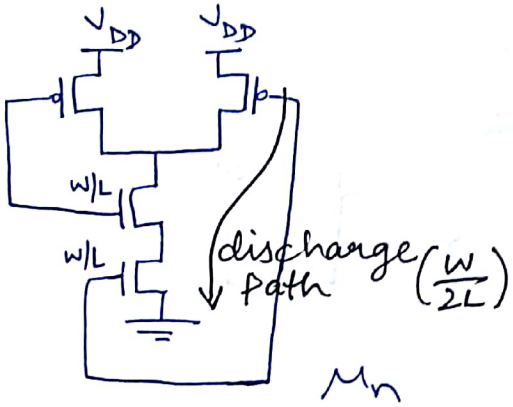
If delays of both gates are same & $x \geq d$, then o/p oscillates.

★ Going from (0,0) to (0,1) & (1,0) is not a problem.

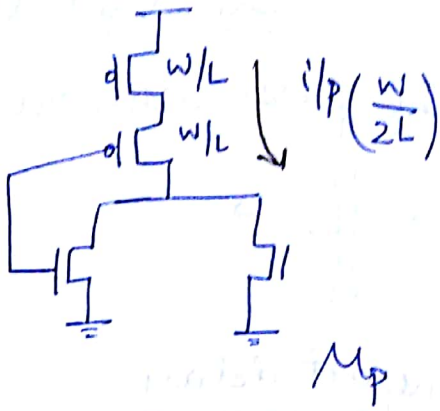
Similar for



NAND gate



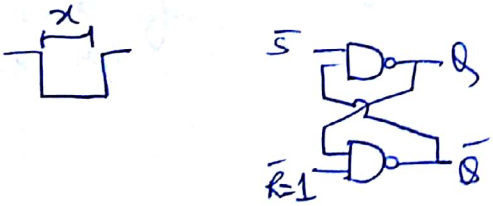
NOR gate



$\therefore$ $M_n > M_p$

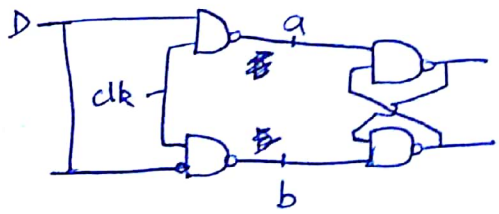
For same load & conditions, o/p current is more for NAND.

NAND is faster than NOR in CMOS logic.



To ~~stop~~ force a value of Q & $\bar{Q}$, 'x' has to be atleast 2 units.

Circuit ⇒ SR bistable with a clock



when clk = ~~high~~ low ⇒ $\bar{S}$ & $\bar{R} = 1$

clk = ~~low~~ high ⇒ $a = \bar{D} = \bar{S}$

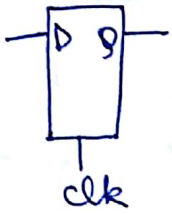
$b = D = \bar{R}$

$\therefore$ Q follows D till clk remains high or the value remembered is the value of D when clk changed from high to low.

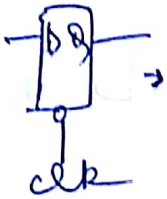
This is not an edge triggered FF
This is <u>Positive level triggered latch.</u>

Actually, a FF should sample D only during rising time. ∴ this circuit is not an edge triggered FF.

<u>Representation</u> ⇒

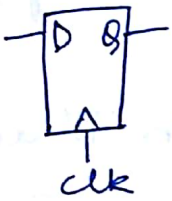


⇒ Latch (+ve edge trigged)

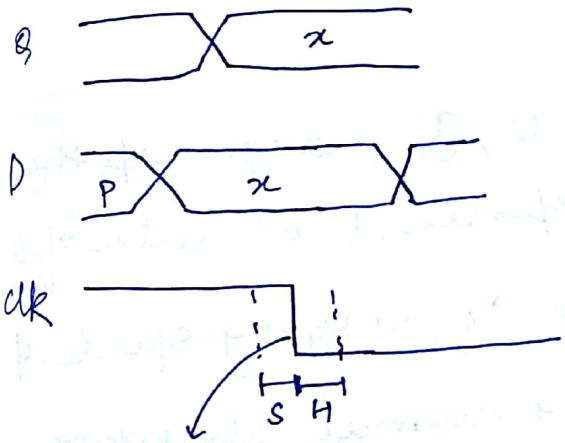


⇒ Latch (-ve edge triggered)

For Edge triggered FF, we put arrow on clk
(to denote sampling at 'rising' edges ~~sampled~~)



⇒ Rising Edge triggered FF

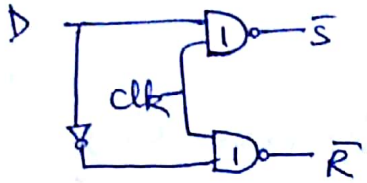


Sampling is done for all time when clk = high

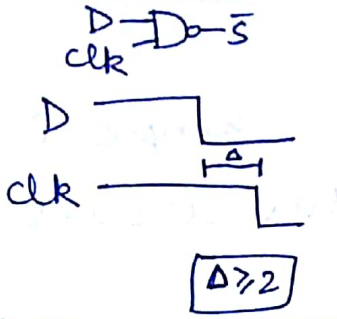
value here (just before) = x
No change when clk is low
⇓
can be considered as a sampling edge.

→

Setup & hold

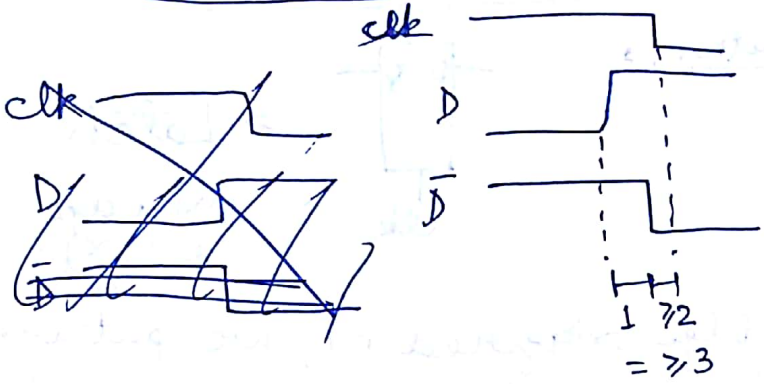


Upper gate :-



Pulse at $\bar{S}$ should have atleast 2 units width

Lower gate :-

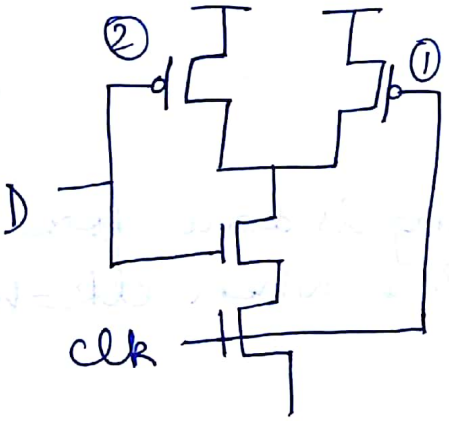


$\bar{D}$ & clock should be atleast 2 units apart.

∴ setup time = 3 units (max. of 2 & 3)

Hold time :-

Once the clock becomes '0', change in 'D' doesn't affect $\bar{S}$ & $\bar{R}$. Clock low means both $\bar{S}$ & $\bar{R}$ will become 1 & value ~~to~~ is restored.



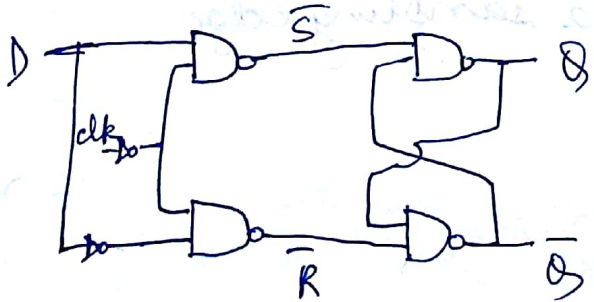
If clk = 0, ① charges up the o/p independent of value of D.

If D = 0 ⇒ charging speeds up

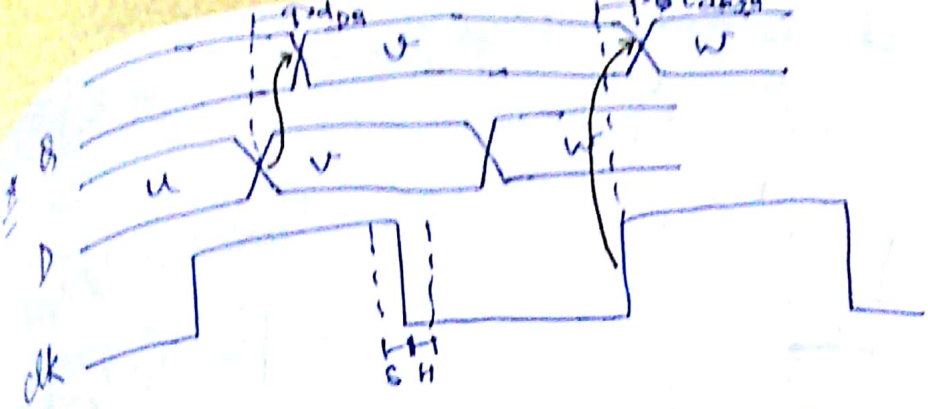
If D = 1 ⇒ normal charging

∴ there is a max & min delay for a gate

∴ Hold time = 0.

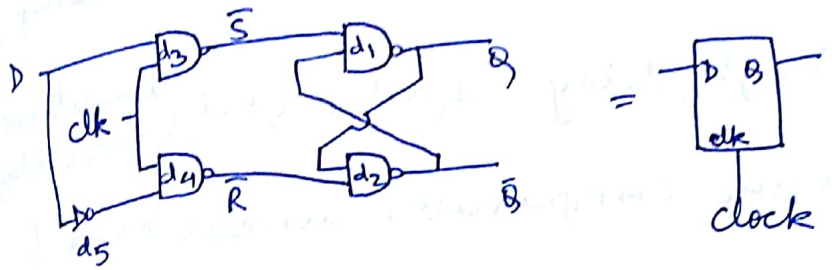
★ 

⇒ -ve level triggered latch



d_{DQ} = delay b/w D & Q (clock is high, so Q follows D)

$t_{clk \to Q}$ = delay b/w clock & Q (D changes in the time when clock is low, so change is going to be reflected in Q only ~~when~~ after clock becomes high again)

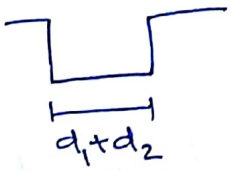


<u>Hold time = 0</u> because as soon as clk goes low, $\bar{S}$ & $\bar{R}$ are not affected by changes in D (theoretically)

<u>Setup time</u> :-

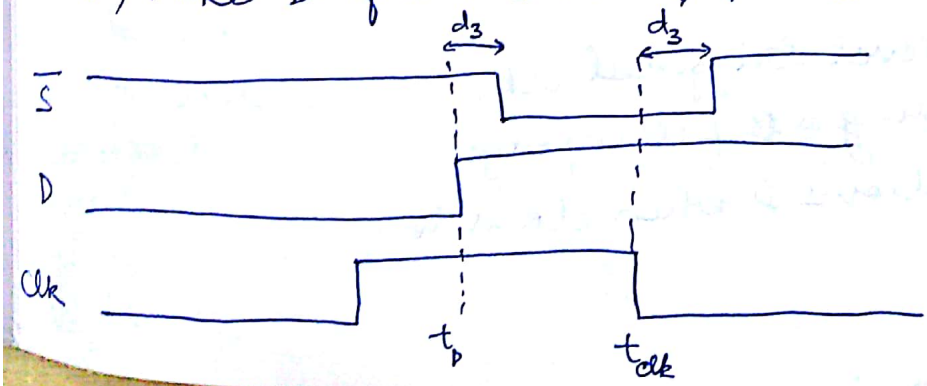
If $\bar{R}$ is kept high,

& $\bar{S}$ =



Then delay for $\bar{S}$ to transfer to $\bar{Q} = d_1 + d_2$

Now, take D from 0 to 1 & consider gate d_3.

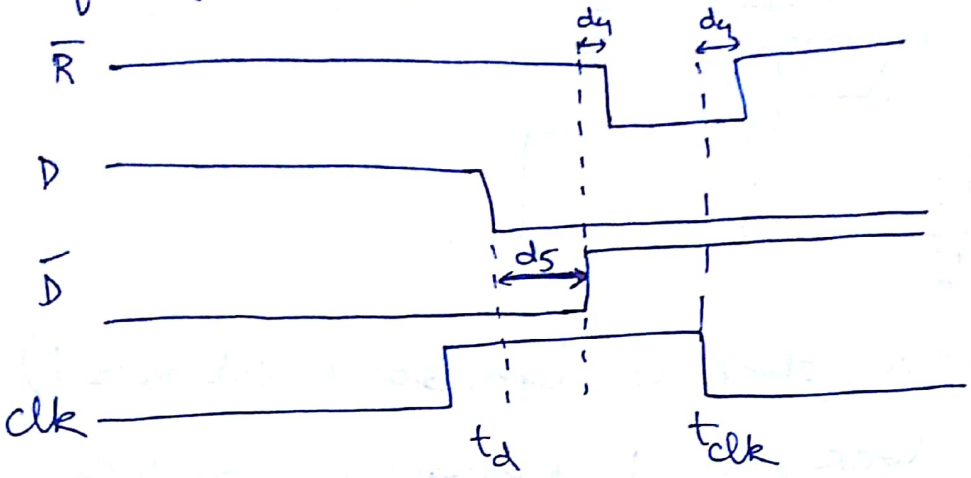


$t_{clk} - t_D \geq d_1 + d_2$

$\Rightarrow \boxed{t_D \leq t_{clk} - (d_1 + d_2)}$

$\therefore \boxed{S = d_1 + d_2}$

If D↓ when clk is ~~low~~ high (considering d_4)



$t_{clk} - t_d \geq d_5 + d_2 + d_1$

$\Rightarrow$ $\boxed{S = d_1 + d_2 + d_5}$

$\underline{t_{D \to Q}}$ (observable is Q)

If D↓ when clk = high, delay $= d_3 + d_1$ (shortest path, min delay)

If D↑ when clk = high, delay $= d_5 + d_4 + d_2 + d_1$ (longest path, max. delay)

[assuming d's are comparable, almost same]

$\underline{t_{clk \to Q}}$

If D = high & clk changes from 0 to 1, delay $= d_1 + d_3$

If D = low & clk changes from 0 to 1, delay $= d_1 + d_2 + d_4$

$\therefore$ $S = d_1 + d_2 + d_5$

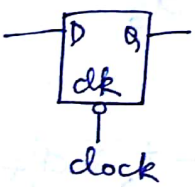
$H = 0$

$d_{D \to Q} = (d_1 + d_3,\ d_1 + d_2 + d_4 + d_5)$

$t_{clk \to Q} = (d_1 + d_3,\ d_1 + d_2 + d_4)$

} each delay consists of a range of number NOT fixed.

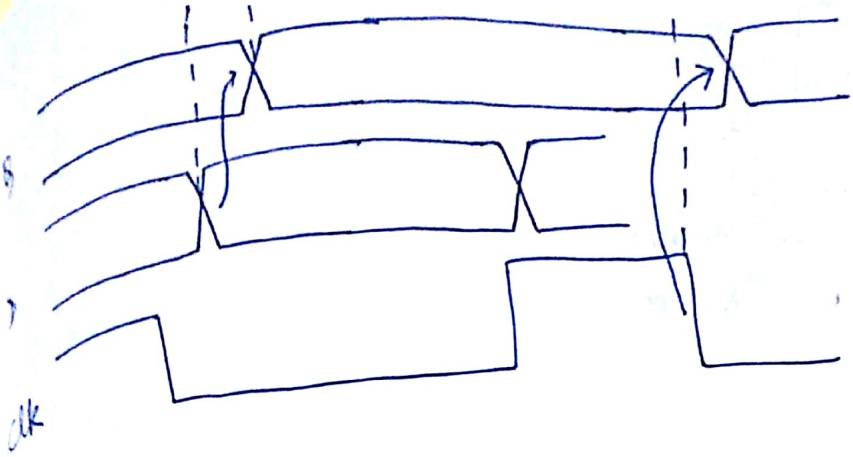
★

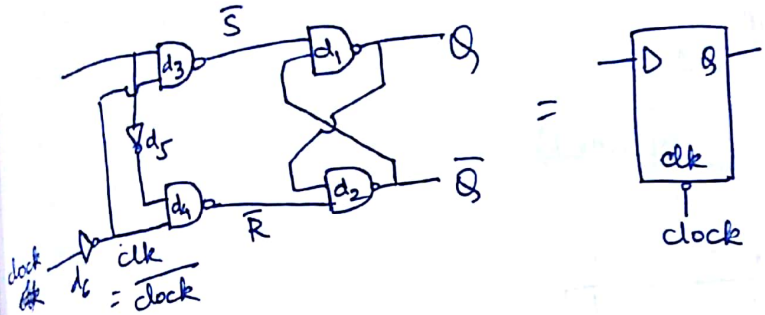


$\Rightarrow$ -ve level triggered clk.

sampling $\Rightarrow$ falling edge

Q follows D when clk is low.

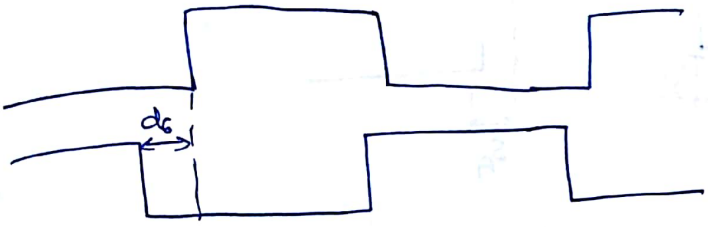




$d_{D\to Q}$ ~~tc~~ ~~clk→Q~~ doesn't change

Hold time (wrt $\overline{clk}$) was '0'

$\therefore$ $\boxed{H_{clk} = d_6}$

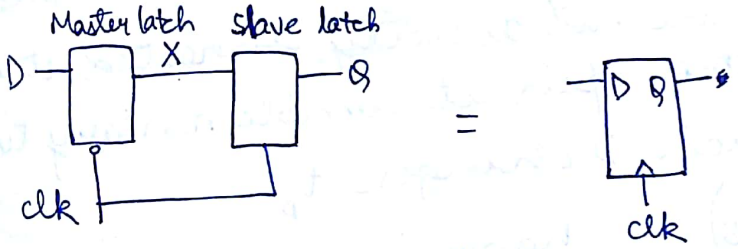


$H = 0$, $H' = d_6$ → new hold time

$S' = S - d_6$ → new setup time

$d_{clk\to Q} = (\text{older delays}) + d_6$

* Cascading a +ve & -ve level triggered results in a FF.

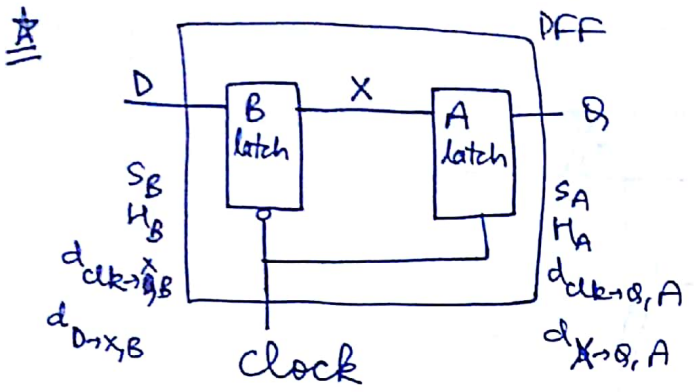


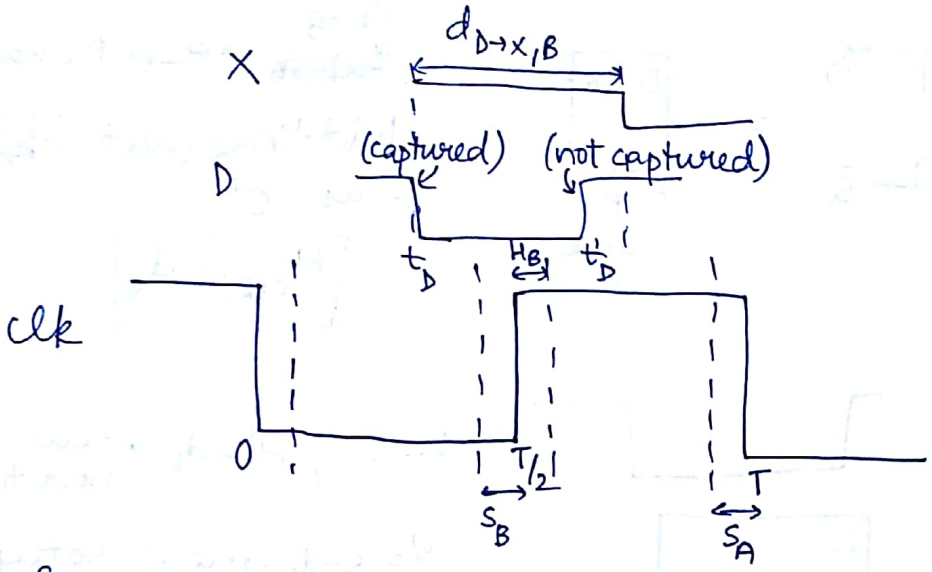
X follows D when clk is 0

clk changes from 0 to 1

Q gets the value X.

No path from D to Q as only 1 of the two latches work.





When clk = low, X follows D (B latch). At the rising edge of clk, the value of D sampled is remembered in X.

When clk = high, Q follows X & gets the sampled value of D. D & Q are not directly connected at any point as A & B latch open at complementary times.

Let, time at which D changes = t_D

$$t_D \leq \left(\frac{T}{2} - S_B\right) \Rightarrow \text{change in D affects X.}$$

If the change in D ~~does~~ should not affect X, then,

$$t_D \geq \left(\frac{T}{2} + H_B\right)$$

X can change earliest after $d_{clk\to X,B}$ time from falling edge of clk. But, this should not happen during the hold time of A.

$$\therefore d_{clk\to X,B} \geq H_A$$

At the latest, X can change $d_{clk \to X, B}$ time after t_D. This must not happen within S_A.

$$\therefore \frac{T}{2} - S_B + d_{D \to X, B} \leq T - S_A$$

$\therefore$ 4 eq^{n}s :-

$t_D \leq \frac{T}{2} - S_B$ $\Rightarrow$ setup time for DFF $= S_B$

(for capture)

$t_D \geq \frac{T}{2} + H_B$ $\Rightarrow$ Hold time for DFF $= H_B$

(for not capture)

$$\left.\begin{aligned} & d_{clk \to X, B} \geq H_A \\ & \frac{T}{2} - S_B + d_{D \to X, B} \leq T - S_A \end{aligned}\right\}$$ correctness constraints $\Rightarrow$ necessary for DFF to work correctly

Now, Q can change earliest at $d_{clk \to Q, A}$ time after rising edge of clock.

$$t_{Q,min} = \frac{T}{2} + d_{clk \to Q, A}$$

The latest Q can change is $d_{X \to Q, A}$ delay after the latest change in X.

$$t_{Q,max} = \frac{T}{2} - S_B + d_{X \to Q, A} + d_{D \to X, B}$$

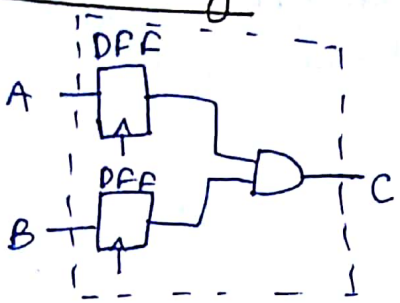
The delay $d_{clk \to Q, A}$ is $\left(t_Q - \frac{T}{2}\right)$.

$\therefore$ we get a range.

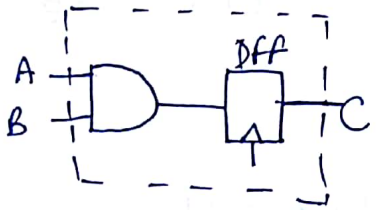
This delay depends on 'slave latch' (latch A).

Setup & hold times of DFF depend on 'Master latch' (latch B).

Retiming

① 

$\Rightarrow C(k+1) = A(k).B(k)$

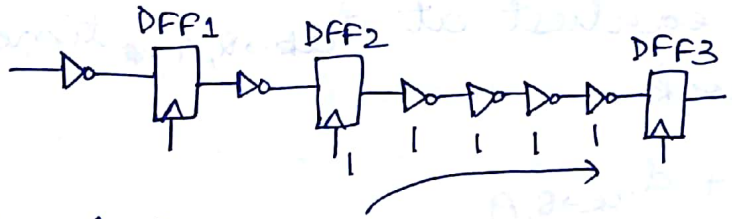
② 

$\Rightarrow C(k+1) = A(k).B(k)$

Functionally, both are same.
But S② is more than S① due to AND gate.
Also, $d_{clk \to C①} > d_{clk \to C②}$

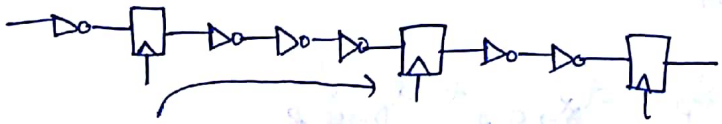
∴ Same ~~fun~~ funcⁿ, different times.

Eg →



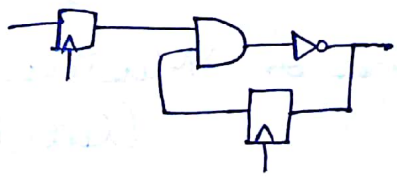
Min. clock period = 5 (let d=1, s=0)

Improving this, shift DFF2,

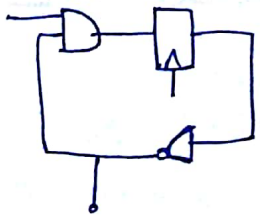


Now, min. clock period = 4
Functionality is same.
[Equalizing ~~g~~ gives optimal solⁿ]

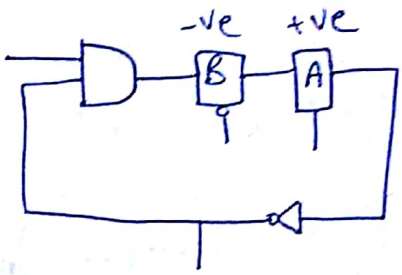
Eg →

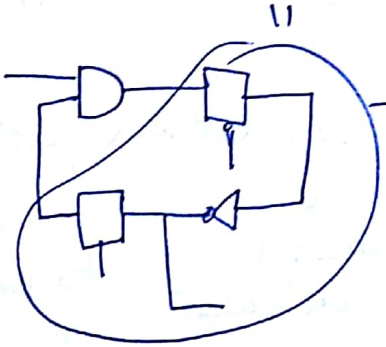


retimed as



Even more retiming can be done.

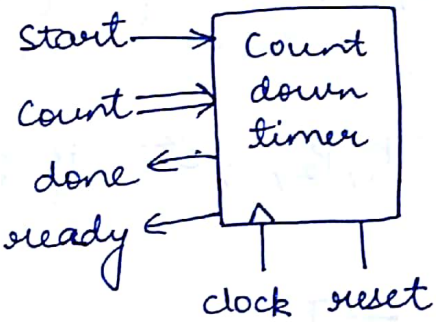




→ Sim. to DFF with a logic b/w 2 latches

RTL (Register Transfer ~~Logic~~ Level)

→ countdown timer :-



- counts down from counter to 0.
- ready ⇒ true means ready to operate
- count ⇒ a no. to be counted down
- start = high means start counting down
- done = high means 0 reached (finished counting down)

- Count decreases by 1 in 1 clock ~~cycle~~ pulse

Implementation using FSM

Let count = 32 bits

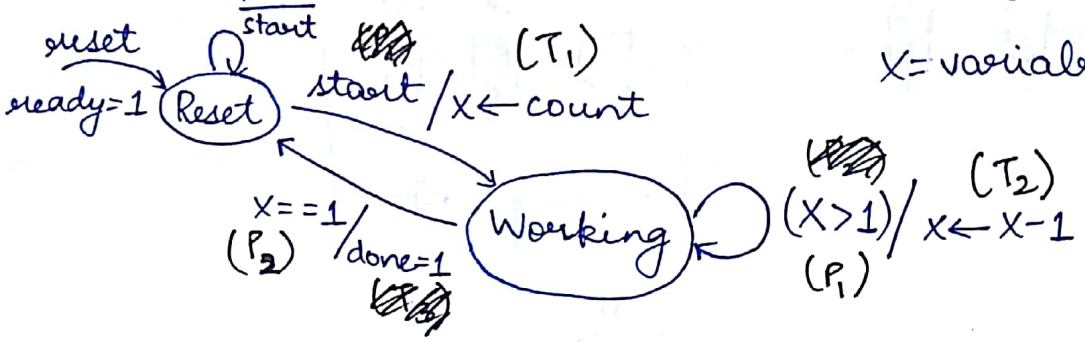
No. of states = $2^{32} - 1$

No matter where we are, if start = 1 ⇒ state is same as count

if start = 0 ⇒ move to lower state

On reaching 0 ⇒ done = 1

Instead, introduce a variable. (Only 2 states required)



X = variable, 32 bits

Each arrow has condition / action

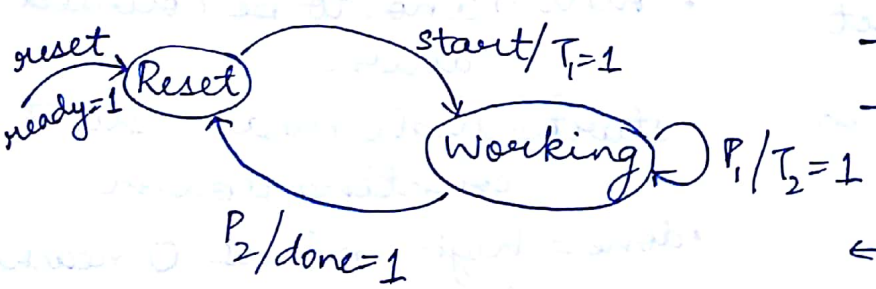
condition → evaluates to 'true' or 'false' → 'Predicate'

action → to be done if condition = true (Performing register transfer or assigning output)

This is RTL machine.

Transfers represented by $T_1, T_2, \ldots$, etc. If T_i = true, that transfer (T_i) happens.

Predicates are represented by $P_1, P_2, \ldots$, etc. If P_i = true, that predicate (P_i) = true.



T_1 — X ← count

T_2 — X ← X-1

P_1 — ~~X>1~~ (X > 1)

P_2 — (X == 1)

} comparat (can be implement

↓

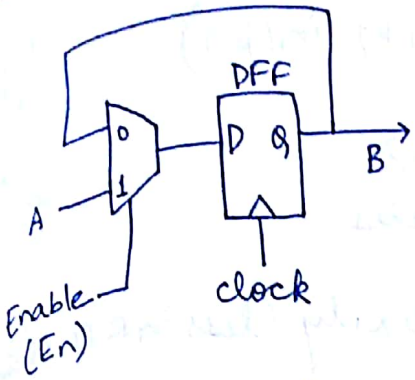
Pure FSM.

Supplies T_1 & T_2 which causes some transfers. Needs P_1 & P_2 depending on X.

* 'ready' has to be defined everywhere, but is alway '0' except at some point.

Convention ⇒ if not specified, ready = 0

Register Implementation

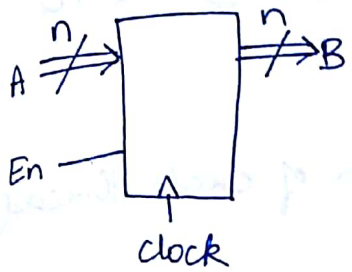


Single bit (1-bit) register

If En=0 ⇒ some old o/p stored in DFF
En=1 ⇒ 'A' stored

$B(k+1) = (En(k)\ ?\ A(k) : B(k))$

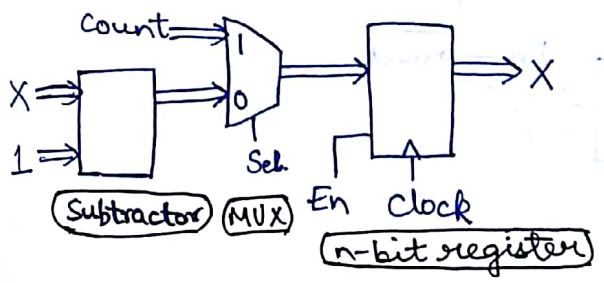
∴ if we want $B \leftarrow A$, at a clock edge, make En=1



(n-bit register)

'A' loaded to 'B' at rising edge of clock if En=1.

Now, implementation of the design is like:



T_1	T_2	Sel.	En
0	0	d	0
0	1	0	1
1	0	1	1
1	1	d	d

Program:

reset →
ready = 1
if start
x ← count, goto working
else, goto reset

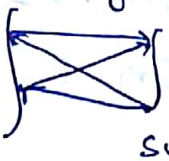
working →
if X > 1
X ← X-1, goto working
else, if X == 1
goto reset, done = 1

Transfers $\Rightarrow x(k+1) = f(x(k)\ldots)$

FSM outputs $\Rightarrow y(k) = f(\underbrace{x(k), q(k)}_{\text{state \& variables}}, in(k))$

★ 2 types of complexity $\Rightarrow$ state complexity (less in RTL than in FSM)
$\Rightarrow$ gate complexity (same in both RTL & FSM)

Each RTL machine is modelled as a thread.
Then, a thread can call a subthread (same as calling a func^n in a program).

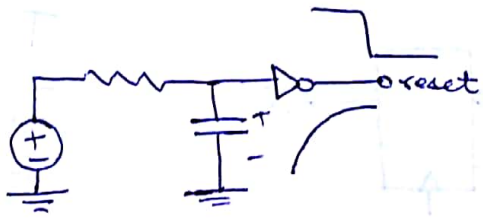


thread (Master) subthread (slave)

$\rightarrow$ decreases length of each thread
$\rightarrow$ easy to verify

Recursion is avoided (i.e. slave calling itself). As hardware is same, slave ① o/p needs to be memorized before calling slave ②.

★ Power ON reset $\Rightarrow$

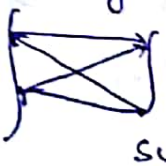


Transfers ⇒ $x(k+1) = f(x(k)\ldots)$

FSM outputs ⇒ $y(k) = f(\underbrace{x(k), q(k)}_{\text{State variables}}, in(k))$

★ 2 types of complexity ⇒ state complexity (less in RTL than in FSM)
⇒ gate complexity (same in both RTL & FSM)

Each RTL machine is modelled as a thread.
Then, a thread can call a subthread (same as calling a func^n in a program).

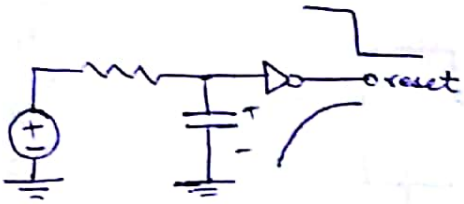


thread (Master) subthread (slave)

→ decreases length of each thread
→ easy to verify

Recursion is avoided (i.e. slave calling itself). As hardware is same, slave ① o/p needs to be memorized before calling slave ②.

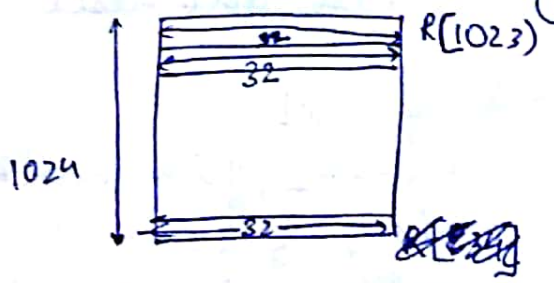
★ Power ON reset ⇒



★ RTL + RAM ≈ Turing ~~Machine~~ Machine
(not ∞ length tape)

<u>RAM (Random Access memory)</u>

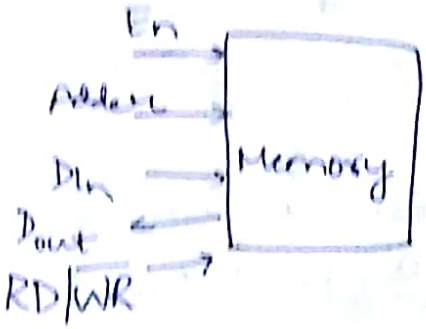
Eg ⇒ 1024 x 32 bits memory



Each is a 32 bit register.

We have to write/read from a particular register

We need an ~~border~~ adder i/p of ~~20~~ 10 bits specifying the register on which the work has to be done



specifies what ~~has~~ to be done

1 = Read, 0 = write

Here we take memory to be like register (work done in 1 clock cycle)

synchronous (single cycle operation)

For register,

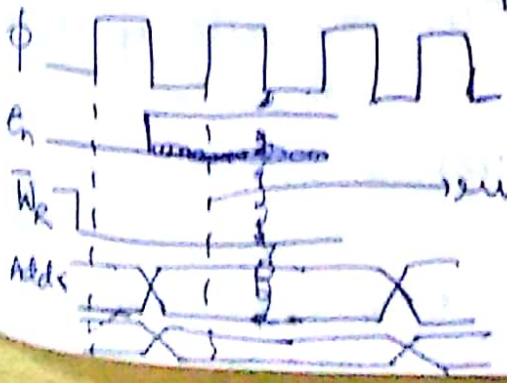
$Q(k+1) \Leftarrow [En\,?\,D(k) : Q(k)]$

Now, if $(\overline{W}_R(k) == 1)$ & $(En(k) == 1)$

$R[Addr(k)](k+1) \Leftarrow Din(k)$

If $(R_D(k) == 1)$ & $(En(k) == 1)$,

$Dout(k+1) = R[Addr(k)]$

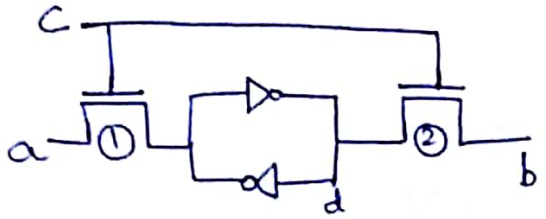


rising edge on which write operation occurs.

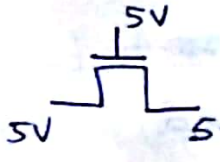
Take pics from KP

SRAM = static RAM (fast, expensive)
DRAM = Dynamic RAM (slow, cheap)

Another way to store a bit :-



A transistor is good ~~at~~ at passing 0 but not at passing 1.

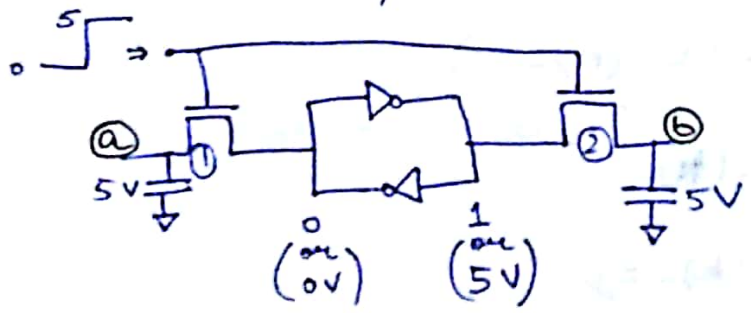


$5 - V_t \Rightarrow$ ↓ as this ↑, V_{gs} ↓, Resistance ↑ ∴ charging delay is high.

To write d=0,
b=0; a,c=1

② pushes d down nicely but ① forces e up slowly.
Flipping occurs with less delay.

Sim., to write d=1 ⇒ a=0; b, c=1
To retain 'a' value, c=0 (both transistors OFF)
To read a value,



Amplify the voltage diff. V_{ab}.
$V_a < V_b \Rightarrow$ then the value is detected

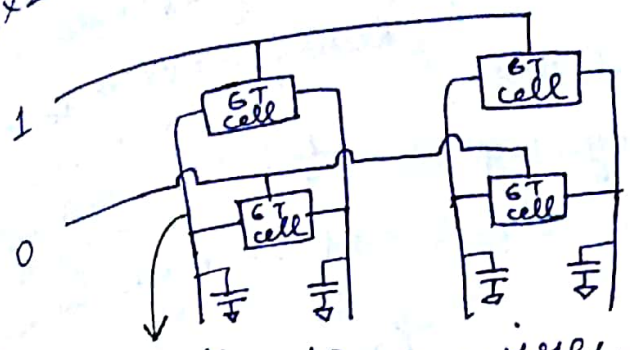
* ① is ON, pull up the '0' ∴ cap voltage ↓
② does not affect cap. voltage

∴ $V_a < V_b$ when stored value is as shown in fig.

This is called 6 transistor SRAM cell
(6T cell)

SRAM cell

2×2 memory



we can tie these wires as the control signals are expected to be mutually exclusive (efficient, as previously mutual exclusiveness was not necessary)

∴ 1024 × 32 bit memory.

64 vertical wires

1024 horizontal wires

32 amplifiers

Writing ⇒ force the 64 wires

Make the hori. wire corresponding to the address 1.

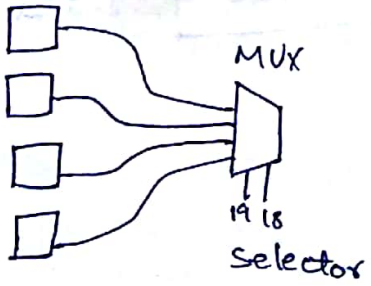
Sim. with reading.

⇒ size ↑, speed ↓

delay ∝ (no. of columns + no. of rows)

∴ for large memories, small memories are used repeatedly

256k × 64



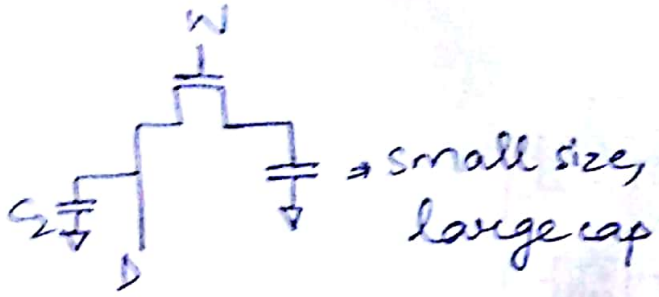
20 bits

19 – } choose where
18 – } to write

17 – }
⋮ } used to access with a bank
0 – }

SRAM does not forget the stored value even after reading it

DRAM cell



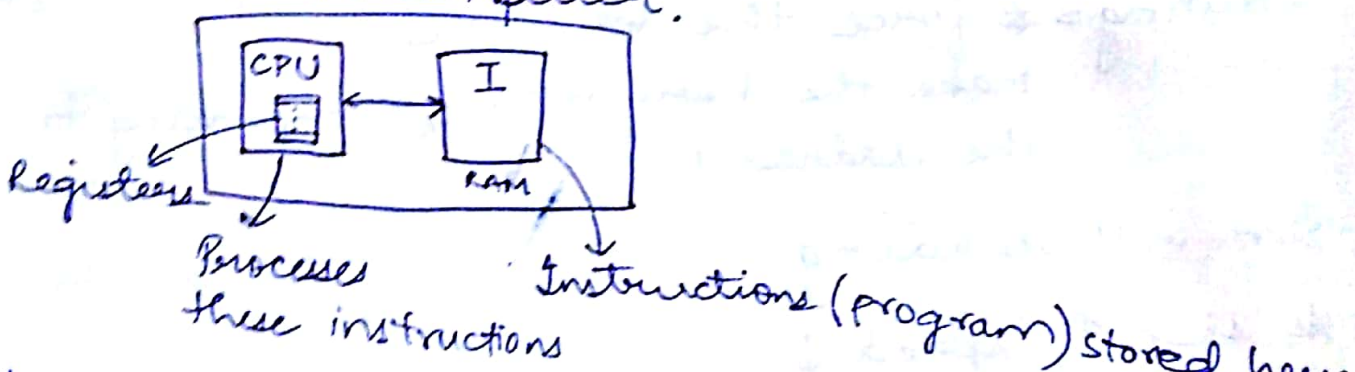
Writing 1 → make D = 1μ.
Reading → charge C_2 to a known voltage
if 1 ⇒ less ~~discharge~~ discharge of C_2
if 0 ⇒ more discharge of C_2
(Detect using amplifier)

Disadvantages:-

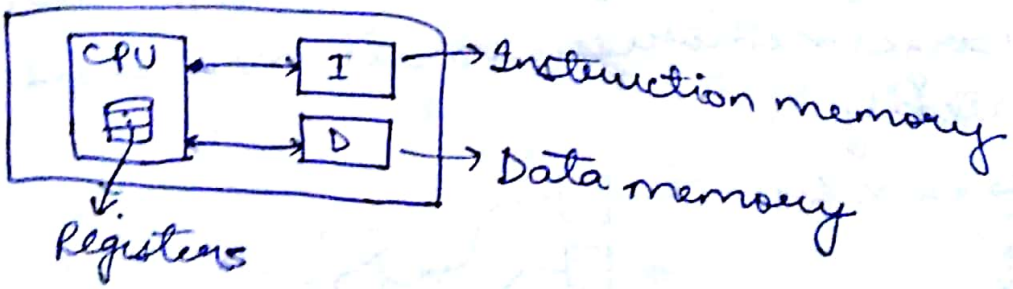
After reading, stored value is lost
In meantime, leakage in transistor causes stored value to be lost.
→ Delay is more (≈ 100 times slower than SRAM)

☆ Memory can be used to implement a finite tape Turing machine.

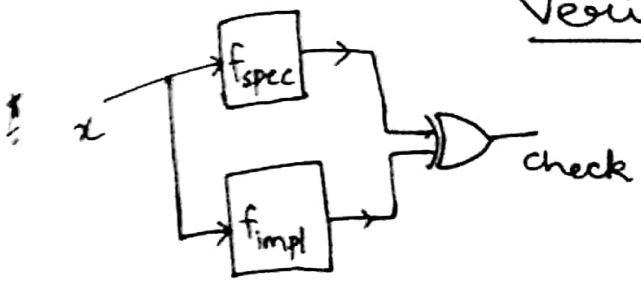
☆ Von Neumann Computer.



Harvard

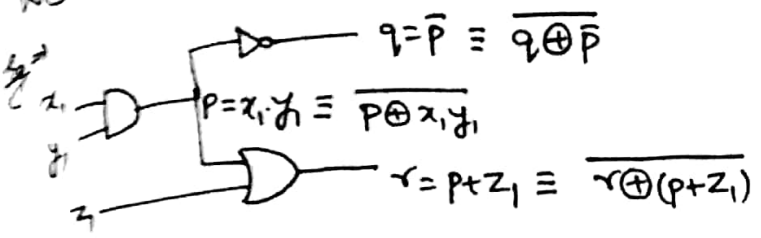


Verification



We need to describe each system thru a formula.

Eg:



$q = \bar{p}$ if $\overline{q \oplus \bar{p}}$ is '1'.
Similarly others.

Now, we 'AND' these three formula.

$\overline{q \oplus \bar{p}} \cdot \overline{p \oplus x_1 y_1} \cdot \overline{r \oplus (p + z_1)}$ ⇒ this formula describes the system.

All systems can be described using 1 formula.

$f_{spec}(\overset{\text{inputs}}{x, y}, \overset{\text{outputs}}{w})$

$f_{impl}(x, y', w')$

$A = f_{spec} \cdot f_{impl}$

$B = y == y'$

$A \Rightarrow B \equiv \cancel{f_{spec} \cdot f_{impl} \cdot (y \neq y')}$

The 'spec' must be correct. To confirm that the 'spec' is correct, we need some properties of the 'spec'.

Eg: If the 'spec' is an adder, we'll ~~check~~ verify its properties like commutativity, etc.

Both 'spec' & properties can be described by a formula.
Then, (f_{spec} ⇒ Property) should always be true.

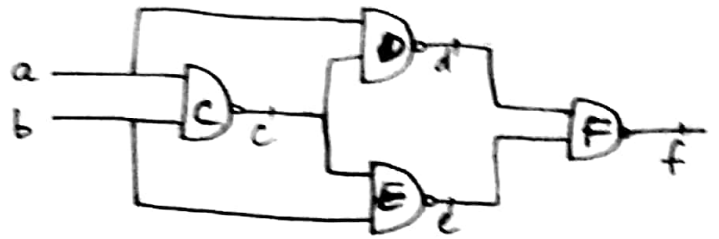
Testing

We need exhaustive list of tests to ensure that if anything goes wrong, there is a test to catch it.

There may be a defect is wiring or some component of some gate might not work properly or the output might be 'stuck-at' some value.

Single Stuck-at Model

→ one gate is defective (o/p stuck at 0 or 1)



⇓

8 possible defects

⇓

Simplification of delay

⇓

Thru this, we can capture any defect (possibility of missing something is less)

For c stuck at (s.a.) '0'

condition ① ⇒ c should give 1 $(\overline{a \cdot b} = 1)$

② ⇒ c must be observable at o/p

Now, $f = ac + bc = (a+b) \cdot c$

Keep $(a+b) = 1$

$\overline{ab} = 1$ & $(a+b) = 1$ ⇒ $(a,b) = (0,1)$ or $(1,0)$

Suppose o/p is $f(a,b,c) = out$

We want it to be affected by 'c'.

$\therefore f(a,b,0) \neq f(a,b,1)$

Or, $f_c \oplus f_{\bar{c}} = 1$

$$\left.\begin{aligned} f_c &= f(a,b,1) \\ f_{\bar{c}} &= f(a,b,0) \end{aligned}\right\} \text{Shannon's decomposition}$$

$\therefore c \,\&\, \frac{\delta f}{\delta c}$ should be 1.

Or, $c \cdot \frac{\delta f}{\delta c} = 1 \Rightarrow a \,\&\, b$ satisfying this will be possible test cases.

For test case (1, 0)

$c=1,\ d=0,\ e=1,\ f=1 \Rightarrow$ For a good circuit.

c s.a '0' can be detected

d s.a '1' can also be detected.

↳ fault is propogated to f (if d was stuck at '1', f would be '0').

Can't detect e s.a. 0 because even if fault is excited, it is not propogated.

f s.a. '0' can also be detected.

<u>for c s.a. 1</u>

$c=0 \Rightarrow \overline{ab} = 0$

$f_c \oplus f_{\bar{c}} = 1$

$\Rightarrow a+b=1$

$\Rightarrow a=1,\ b=1 \Rightarrow$ apply this test

Now, $c=0,\ d=1,\ e=1,\ f=0 \Rightarrow$ for good circuit

c s.a. '1' can be detected

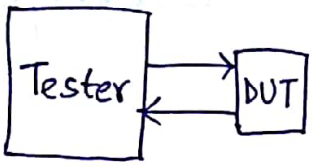
d s.a. '0' can also be detected

e s.a. '0' ——— " ———

f s.a. '1' ——— " ———

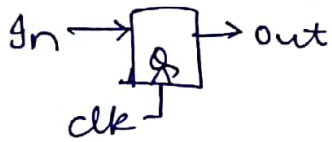
For e s.a.'1' ⇒ test case is $(a, b) = (0, 1)$

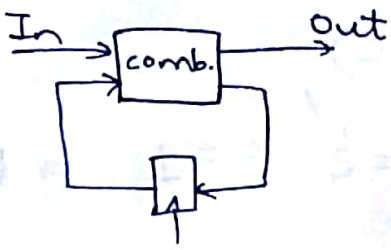
→



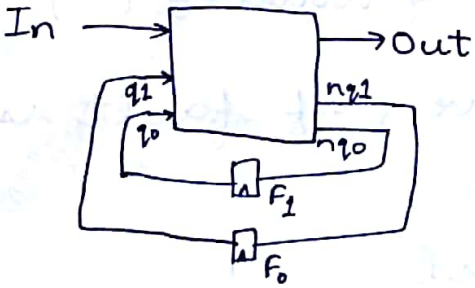
applies test on DUT.

Testing sequential circuits





For a defect in combinational, we have to fix both inputs or bring the machine to a particular state — problem



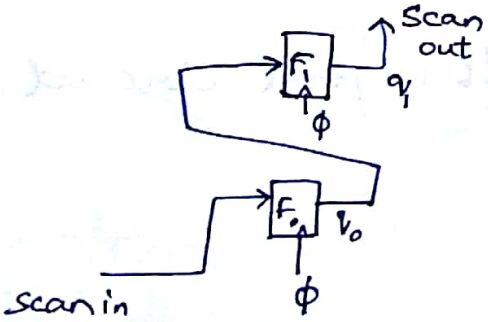
We want to control q_1 & q_0 & observe nq_1 & nq_0

Introduce scan chain,

can force a value at o/p of DFF

nq0 & nq1 are loaded into DFFs & scanned out.

Shift register used.



Then scan in i/p are shifted into q1 & q0.

Eg → Let q_1 & q_0 be 1 & 0 initially.

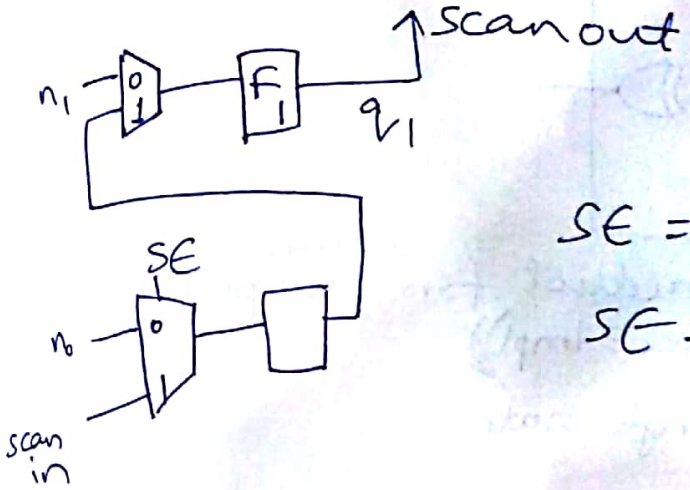
0 in → $q_0 = 0$, $q_1 = 0$

scan outs = 1

1 in → $q_0 = 1$, $q_1 = 0$

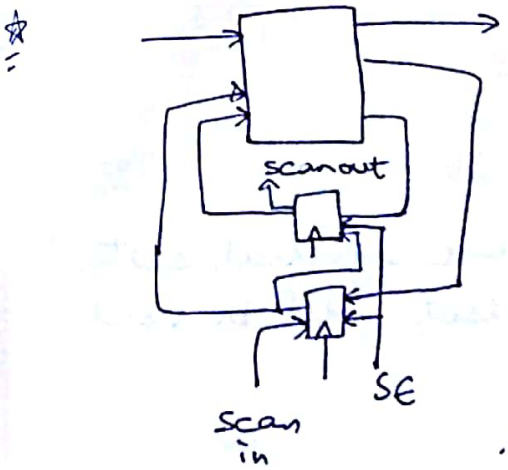
~~so~~ scan out = 0

This combined with normal functioning of DFF.



SE = 1 (testing)

SE = 0 (normal DFF)



SE = 1 (nq not getting loaded)

↓

scan in / scan out

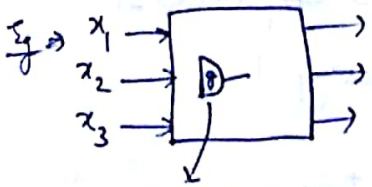
(will take many clk cycles in SE = 1 in all of them)

↓

SE = 0 & → $\phi\uparrow$ (rising edge)

∴ we can force q's & read nq's.

s.a. in flipflops can be done using scan out.



$g(x_1, x_2, x_3) = 1$

$out_1(g, x_1, x_2, x_3)$

choose x_1, x_2, x_3 s.t. out_1 depends on g.

$$\frac{\partial out_1}{\partial g} = out_1(1, x_1, x_2, x_3) \oplus out_2(0, x_1, x_2, x_3)$$

Finally, $g \cdot \left(\frac{\partial out_1}{\partial g} + \frac{\partial out_2}{\partial g} + \frac{\partial out_3}{\partial g} \right) = 1$

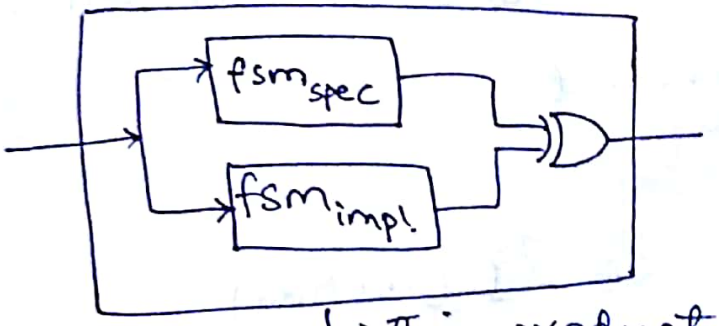
Sequential verification

confirmation that implementation satisfies a spec.

Spec = fsm

implementation = circuit

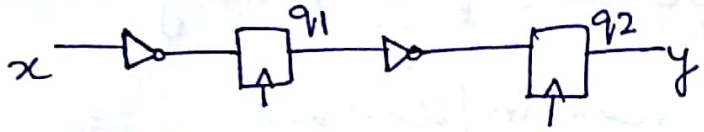
↳ convert into fsm

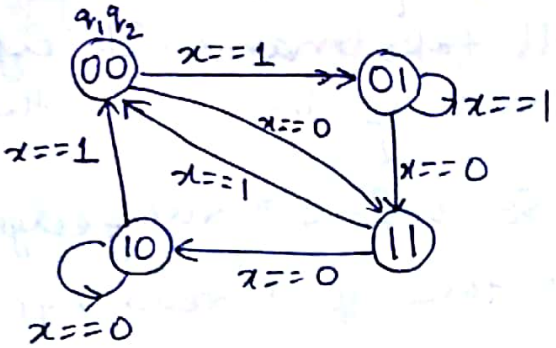


↳ This product fsm always outputs '0' (if spec ≡ impl.)

∴ reducible to single state.

* Convert circuit to FSM



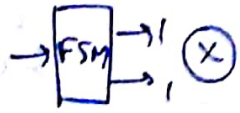


For initial state, reset will be used.

Checking the specification using properties.

Eg → To prove o/p is not simultaneously 1.

Determining set of reachable states



start from initial, move where next state can be, expand the reachability set until no new state can be reached.

See that in every reachable state o/p is not 11

$x\{k\} \rightarrow \boxed{FSM} \rightarrow y\{k\}$

→ prop. after a '1', ∃ atleast 3 '0' before next '1'.

⇓

$x\{k\} \rightarrow \boxed{FSM} \rightarrow \boxed{FSM'} \rightarrow 1/0$

def. on whether the seq. is found or not

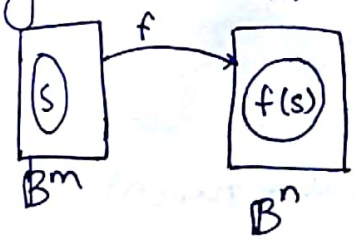
To check the prop.

⇓

develop a product FSM ~~prove~~
Prove that it can be reduced to only single state.

Verification

Using Boolean tools



$x \rightarrow f(x) = y$
$f(s)$ = image of set s
s = preimage of set $f(s)$

$f(x) = y \Rightarrow$ unidirectional (not very useful).

$\chi_f(x, y) = (f(x) \equiv y)$

Or $\chi_f(x, y) = \begin{cases} 1 & \text{when } y = f(x) \\ 0 & \end{cases}$

Eg → $f(x_1, x_2) = x_1 x_2 = y$

$\chi_f(x_1, x_2, y) = \overline{(y \oplus (x_1 x_2))}$

} χ called behavioural representation

In defining $\chi \rightarrow$ no i/p/o/p
Only 3 quantities.

Let $S \to \{(x_1, x_2) \mid x_1 = 1\}$

$f(S) \to \{y \mid y = f(x_1, x_2) \,\&\, x_1 = 1\}$

This can be written as:

$\chi_1(x_1, x_2, y) \cdot x_1 = 1$

$\therefore$ We need y for which $\exists$ x_1, x_2 s.t. $\chi_f(x_1, x_2, y) \cdot x_1$ is true.

existential quantifier

$\Sigma_y \to$ If $\exists\, x_1$ s.t. $f(x_1, x_2) = 1 \equiv f(0, x_2) + f(1, x_2) = 1$

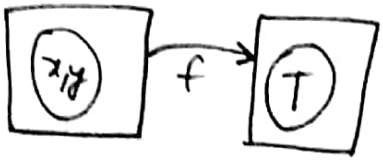
$\to$ If $\forall\, x_1$ s.t. $f(x_1, x_2) = 1 \equiv f(0, x_2) \cdot f(1, x_2) = 1$

$f(S) = \{y \mid \exists\, x$ s.t. $\chi_f(x, y) = 1 \,\&\, x \in S\}$

or

$\exists\, x$ s.t. $\underbrace{I_S(x)}_{\text{denotes that } x \in S \text{ (Indicator func}^n\text{)}} \cdot \chi_f(x, y)$ is true.

for



$f^{-1}(T) = \{x \mid \exists\, y$ s.t. $\chi_f = 1 \,\&\, y \in T\}$

OR $\exists\, y$ s.t. $I_T(y) \cdot \chi_f(x, y)$ is true.

<u>Using this to find set of reachable states</u>

<u>FSM</u> x, y, q $\quad q_0 =$ initial state

$y = \lambda(x, q)$

$q = \delta(x, q)$

& also if $x \in S$

$Q_0 = \{q_0\}$ reachable in 0 steps

calculate $Q_1 =$ reachable in 1 step & so on.

This is like calculating image given the func^n

$(nq = \delta(x,q))$ & $x \in S$, $q \in Q_0$

$\therefore Q_1$ is all possible nq s.t. $x \in S$

$q \in Q_0$ & $nq = \delta(x,q)$

$\therefore$ for Q_1, $\chi_\delta(x,q,nq) \cdot I_S(x) \cdot I_{Q_0}(q)$ is true

$\exists\, x, q$ s.t.

$\therefore$ reachable states upto 1 step $= Q_0 \cup Q_1$

This can be repeated.

Boolean func^n representation by formula

not always canonical & efficient.

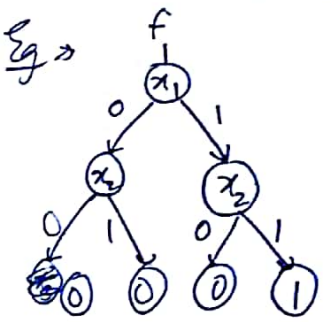
depends on diff. optimisation methods

<u>Another ~~method~~ way of representation</u>

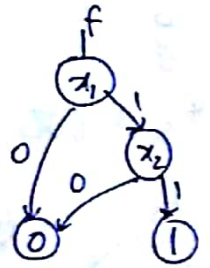
Using Shannon's decomp.

$f = x f_x + \bar{x} f_{\bar{x}}$

Eg →



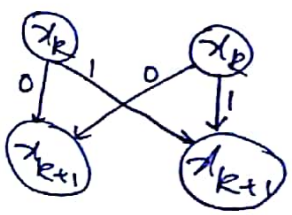
=



→ can't be reduced further

→ called Binary Decision Diagram (BDD)

→ Other possible red^n



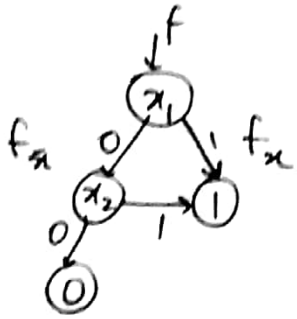
⇒ can merge x_k

* BDD for $x_1 + x_2$

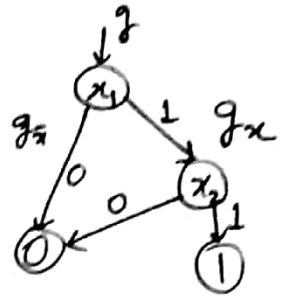
* $\underbrace{x_1 x_3}_{g} + \underbrace{x_1 + x_2}_{f}$

$f+g = x(f_x + g_x) + \bar{x}(f_{\bar{x}} + g_{\bar{x}})$

we can do it recursively.



+



$f+g = x_1(f_{x_1} + g_{x_1}) + \bar{x}_1(f_{\bar{x}_1} + g_{\bar{x}_1})$

f + g

$f_{x_1} + g_{x_1}$ $\qquad$ $f_{\bar{x}_1} + g_{\bar{x}_1}$

1 +

+ 0