* Order of variables has a huge impact on the size & shape of ROBDD

eg. $a_1b_1 + a_2b_2$ — $a_1a_2b_1b_2$

$a_1b_1a_2b_2$

NP-hard : only heuristics exist to chose the right order

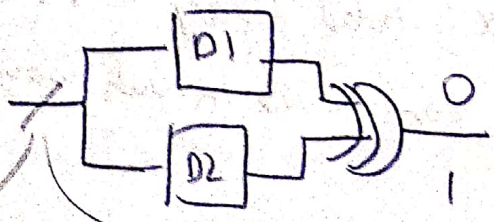
· Software verification is a much harder problem

∴ ALGO → HLS → LS → Gate level Netlist

very difficult

If BDD is very big, ~~then~~ then what?

If BDD is small, directly compare ROBDD



1 → If 1, then some problem

explicit way of testing all inputs is not scalable → Counter example

∴ Satisfiability problem (SAT problem)

∴ We want to find set of inputs such that

$f_1 \oplus f_2 = 1$ i.e. $f_1(x_1, x_2, \ldots x_n) \oplus f_2(x_1, x_2, \ldots x_n) = 1$

Suppose we use POS method

e.g.

$\therefore f_1 \oplus f_2 = (x_1 + x_2 + x_3)(\bar{x_1} + \bar{x_2} + x_3) \ldots = 1$

C_1 — clause; C_2

$\therefore C_1 = 1, C_2 = 1, \ldots C_m = 1$

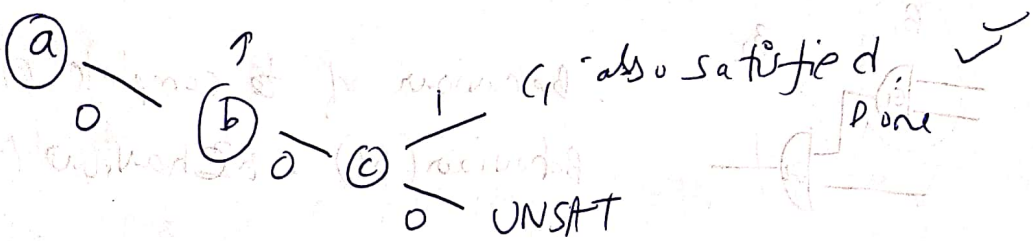
e.g. $(a+b+c)(\bar{a}+\bar{b}+c)(\bar{a}+\bar{c}) = 1$

C_1 C_2 C_3

Assign variables one-by-one & check if clauses are satisfied:

DPLL algorithm

C_2, C_3 are satisfied



∴ 001 is one input which will give diff. output

But for testing, we don't want SAT and hence we may need to explore entire tree.

∴ May not be scalable

Logic implication:

$X \rightarrow Y$

X	Y	Z
0	0	1
0	1	1
1	0	0
1	1	1

when X is 0, it means anything is acceptable $\Rightarrow z = 1$

when X is 1, Y should follow Y

$\therefore X \rightarrow Y \equiv \bar{X} + Y$

$\therefore$ for an AND gate: $a, b \Rightarrow c$

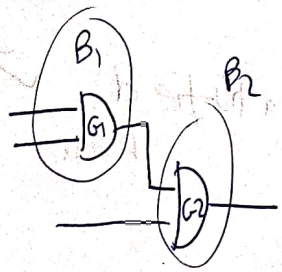
$(a=0) \rightarrow (c=0)$

$(b=0) \rightarrow (c=0)$

$((a=1)(b=1)) \rightarrow (c=1)$

$\left.\begin{array}{l}(\bar{a} \rightarrow \bar{c})(\bar{b} \rightarrow \bar{c})(ab \rightarrow c) \\ \therefore (a+\bar{c})(b+\bar{c})(\bar{a}\bar{b}+c)\end{array}\right\}$

$\bar{a}+\bar{b}+c$



$\therefore$ Behaviour of the complete circuit:

Behaviour(B1) . Behaviour(B2)

Clause for OR:

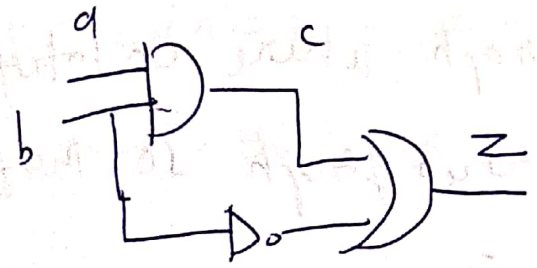
$a=1 \rightarrow c=1$

$b=1 \rightarrow 1=1$

$(a=0)(b=0) \rightarrow 1=0$

$\therefore (a \rightarrow c)(b \rightarrow c)(\bar{a}\bar{b} \rightarrow \bar{c})$

$\therefore (\bar{a}+c)(\bar{b}+c)(a+b+\bar{c})$



$\therefore (a+\bar{c})(\bar{b}+\bar{c})(\bar{a}+\bar{b}+c)$
$\times \quad (b+d)(\bar{b}+\bar{d})$
$\times \quad (\bar{c}+z)(\bar{d}+z)(c+d+\bar{z})$
$\times \quad Z \rightarrow z$ should be 1

we can use ~~SAT~~ solver to find a, b.

∴ we use this method to generate the POS clauses, which is then passed to a SAT solver like DPLL algo

- Variants of DPLL:

1) Unit clause rule: $(a+b+c)(\bar{a}+\bar{b}+c)(\bar{a}+\bar{c})$

when $a=b=0$, clause 1 reduces to a unit clause i.e. only c.

∴ Instead of going to $c=0$, we can directly consider $c=1$

when the unit clause occurs in the middle of the tree, we will get a significant improvement. (Not obvious when c is leaf)

Called, Boolean Constraint Propogation

* Consensus : $(x+y)(\bar{x}+z) = (x+y)(\bar{x}+z)(y+z)$

extra term

$(a+b+c)(\bar{a}+d+c)(\bar{a}+\bar{b}+\bar{c})$

Applying consensus on (

$(a+b+\bar{a}+\bar{b})$ ✓ 1

$(\bar{a}+d+\bar{b})$

we are not getting any unit clause and hence consensus is of no use

But: $(a+b)(\bar{a}+c)(\bar{b}+d)(\bar{c}+d)$

$(b+c)$

$(c+d)$

d → ~~prime~~ essential

But order in which we need to apply is difficult to predict

· Conflict Clause recording (recap):

$(a+b)(\bar{b}+c+d)(\bar{b}+e)(\bar{e}+\bar{d}+f)$

$c=0, f=0, a=0 \Rightarrow b=1 \Rightarrow d=1 \Rightarrow e=1$ ⇒ last clause UNSAT

∴ $a=0, c=0, f=0$ can be stored so that we don't traverse it again

. Local learning. Assign a vector and keep flipping those variables which occur in the unsat clauses. But some sat may become unsat after flipping.

Tabbi method

. What about **sequential circuits**?

Say we have 2 machines:

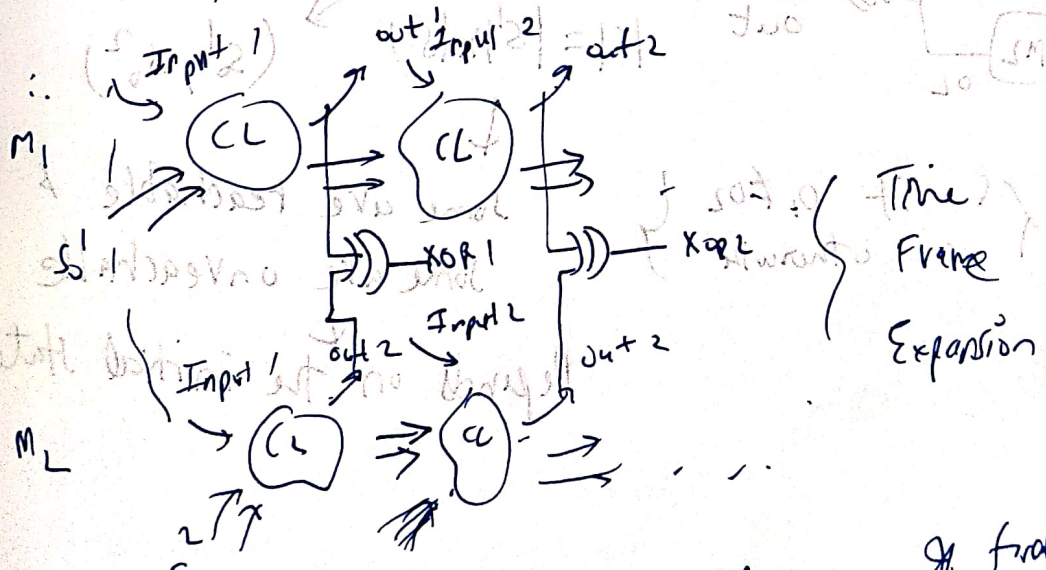
$M_1 = (S^1, S_0^1, X, O, \delta^1, \lambda^1)$

$M_2 = (S^2, S_0^2, X, O, \delta^2, \lambda^2)$

$M_1 \stackrel{?}{=} M_2$. For equivalent machine,

The two machines should give the same output for the same input sequence.

δ, λ, etc. does not matters



If final = 1, machines NOT equivalent

- There is a practical limit till which we can expand timeframe

- we can now use SAT solver to see if there exists an input for which $OR(XOR_1, XOR_2, \ldots XOR_t) = 1$

2) State minimization:

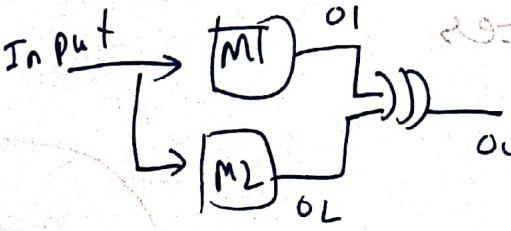
$M_1 \rightarrow M_2$
$\downarrow \quad \downarrow$
$M_1' \quad M_2'$

If $M_1' \equiv M_2'$, $M_1 \equiv M_2$

But $M_1' \not\equiv M_2' \not\Rightarrow M_1 \not\equiv M_2$

Since minimized machine is NOT unique/canonical

3) Reachability: For product machine:



$out = \begin{cases} 1 & \text{if } o_1 \neq o_2 \\ 0 & \text{otherwise} \end{cases}$

$(S, s_0, X, O, \ldots)$

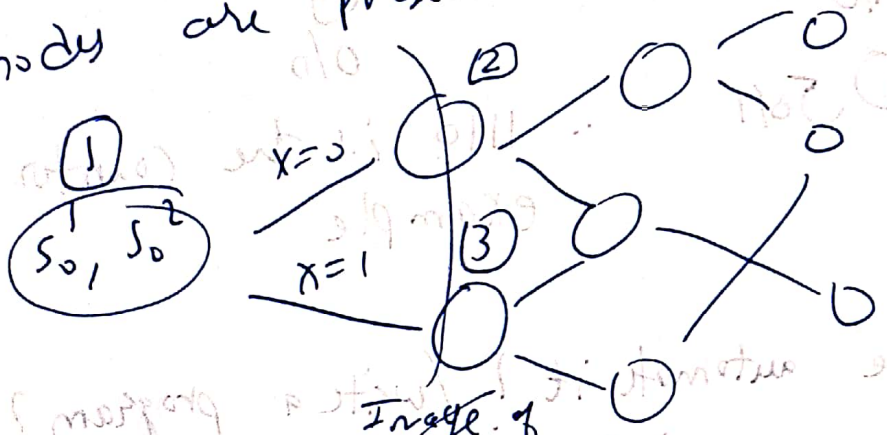
$|S| = |S^1| \times |S^2|$ → (s_0^1, s_0^2)

Some are reachable & some are unreachable

Depends on the initial state

∴ we can stop when we have covered all $n_1 \times n_2$ states ($\leq n_1 \times n_2$: whichever are reachable)

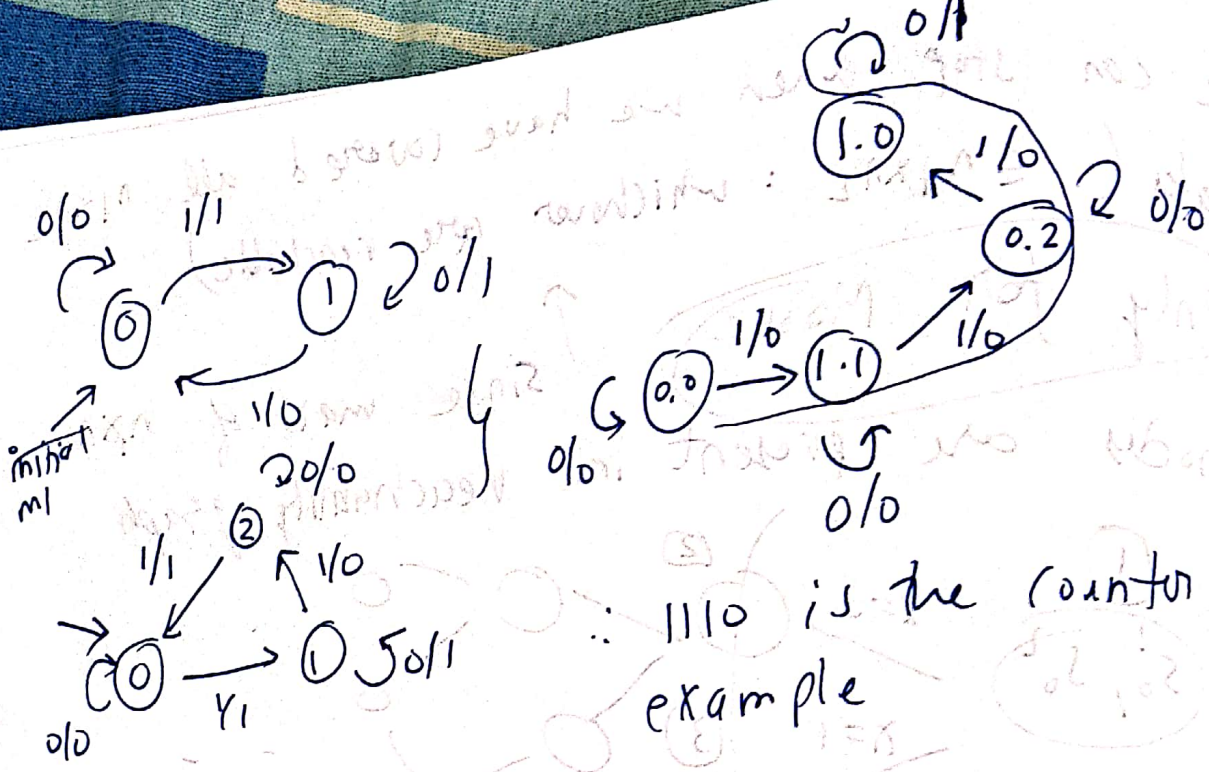
Only for Moore ↑ since max of $n \times n_2$ nodes are present in reachability graph



For Moore, we can associate each output with a node & hence we need to check only $n_1 \times n_2$ times

pre-
. Image of node 2 & 3 = 1

Images of node 1 = 2, 3



∴ 1110 is the counter example

- how can we automate it? / write a program?

→ 1) Representing the reachable states

2) Images of those reachable states

- we can store the reachable states as a boolean fun

S_2	S_1	S_0	Reachable
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0 → 1
1	1	1	1

$\bar{S_2}\bar{S_1}\bar{S_0} + \bar{S_2}S_1\bar{S_0} + S_2S_1S_0$

Store in ROBDD

Now suppose we explore a new ne

e.g. $S_2\bar{S_1}\bar{S_0}$.

we can OR the two ROBDDs and store it in a compact form

Termination condition:

1) ROBDD ~~expl~~ says all states explored
2) output = 1 at any stage
3) ..

For previous qn M_1, M_2:

For M_1: δ^1: Let S_0 denote the state & X is 1-bit input

∴

S_0 \ X	0	1
0	0	1
1	1	0

$$\delta^1 = \bar{S_0}X + S_0\bar{X}$$

$$0 = f^1 = S_0X + S_0\bar{X}$$

↳ coincidence in this case

For M_2: Let S_1S_0 denote the 2 bits used to represent state

$S_1'S_0'$ are the transition states.

For δ_0^1:

S_1S_0 \ X	0	1
00	0	1
01	1	0
11	X	X
10	0	0

$$\therefore \delta_0^2 = \bar{S_1}\bar{S_0}X + S_0\bar{X}$$

For δ_1^1:

S_1S_0 \ X	0	1
00	0	0
01	0	1
11	X	X
10	1	0

$$\therefore \delta_1^2 = S_0X + S_1\bar{X}$$

$t_0 \equiv (\bar{s_0} X + s_0 \bar{X})$ i.e. $t_0 \equiv f^1(s_0, X)$

$t_1 \equiv \bar{s_2} s_1 X + s_1 \bar{X}$ i.e. $t_1 \equiv f_1^2(s_1, s_2, X)$

$t_2 \equiv s_1 X + s_2 \bar{X}$ i.e. $t_2 \equiv f_2^2(s_1, s_2, X)$

$$R = \pi_i (t_i \equiv f(s, X))$$

$$\therefore R(t, s, X) = (t_0 \equiv \bar{s_0} X + s_0 \bar{X}) \odot (t_1 \equiv \bar{s_2} s_1 X + s_1 X) \odot (t_2 \equiv s_1 X + s_2 \bar{X})$$

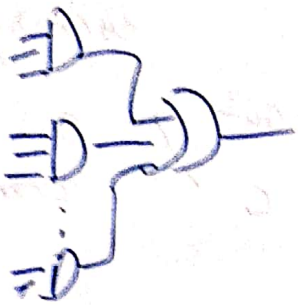
Can be Replaced with XNOR

$\therefore$

t_2	t_1	t_0	s_2	s_1	s_0	X	output of $R(t, s, X)$
0	0	0	0	0	0	0	1 → since func will evaluate to 1 because 0.0 is reachable from 0.0
"	"	"	"	"	"	1	0

ROBDD: implementation is area-efficient but
↳ delay
↳ delay of each mux

total inputs

Min terms:  Less huge area
Less delay

ROBDD: will give optimized SOP (for fixed order)

Then, implement (Path1 + Path2 ... + Path m)

Area is determined by no. of inputs to gate

In CMOS: 2 gate for NOT
4 gate for 2-NAND/...
6 gate for 3-NOR/...

Eg. $\bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}d + \bar{a}cd + \ldots + c\bar{d}$
3 literals 3 literals 2 literals

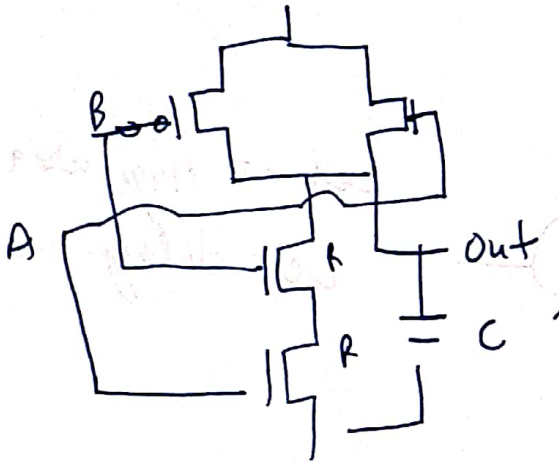
Goal should be to minimize the sum of literals in each term

Proxy for area

A	B	a
0	0	1
0	1	0
1	0	1
1	1	0

· CMOS delay is dependent on the no. of inputs

· delay is ~~defined~~ $\propto$ to time taken for capacitor to charge



R seen by capacitor to discharge $\propto$ no. of inputs and hence time will change

✡ 3-level logic: NOT + AND + OR

parallel stacks for each of these

NOT will have a fixed delay (say 1 unit)

AND will have delay = max (no of literals in ~~the~~ SOP expression)

OR gate delay = # SOP terms

e.g. $ab + bcde$ — 4 delay for AND + 2 delay for OR = 6

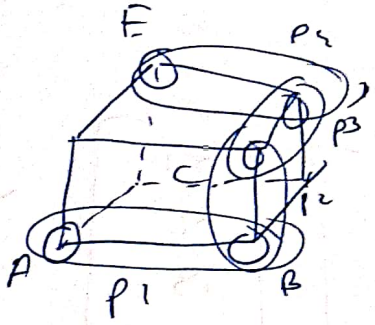
$abc + cde$ — 3 " " + 2 " = 5

For area, it won't matter since Area $\propto \Sigma$ literals

· Quine Mcluskey: Converts implicants → prime implicants

e.g.

A, B, C, D, E are implicants



P1 & P4 are essential
Either P2 & P3 need to be selected

A	P1
B	P1+P2
C	P2+P3
D	P3+P4
E	P4

Since all need to be covered,

A B C D E = 1

i.e. $(P1)(P1+P2)(P2+P3)(P3+P4)(P4) = 1$

SAT problem

But SAT will also give $P1 = P2 = P3 = P4 = 1$ which is not what we want

But we can convert it into SOP form

e.g. if above eg gets reduced to

$P_1P_2P_3 + P_1P_4P_3 + (P_1P_3)$ → best option.

Technically, we should take weighted sum since no of literals in the P_i terms may be different

i.e. choose term from SOP which minimizes $\sum_{i=1}^{n}$ no of literals (P_i) for $P_1P_2 \ldots P_m$

- matrix method:

$$\begin{bmatrix} & P1 & P2 & P3 & P5 \\ A & 1 & 0 & 0 & 0 \\ B & 1 & 1 & 0 & 0 \\ C & 0 & 1 & 1 & 0 \\ D & 0 & 0 & 1 & 1 \\ E & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_5 \end{bmatrix} \geq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{i.e.}$$

$\therefore Ax \geq b$ where A is above matrix

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_5 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

- Minimum covers: P_1, P_4, P_3

- Shannons Expansion:

$$f(x_1, x_2 \ldots x_n) = x_i f_{x_i} + \bar{x_i} f_{\bar{x_i}}$$

$$= x_i f_{x_i} \oplus \bar{x_i} f_{\bar{x_i}}$$

$$\text{Boolean diff} = \frac{\partial f}{\partial x_i} = f_{x_i} \oplus f_{\bar{x_i}}$$

If $\frac{\partial f}{\partial x_i} = 1$, it means that f depends on x_i since f ~~differ~~ changes when x_i is flipped

- Consensus operator: $f_{x_i} \cdot f_{x_i'}$
- Smoothing operator: $f_{x_i} + f_{x_i'}$
- Positional representation:

Say $f(a,b,c) = ab + bc + ca$

Represent each term by 2 bits:

00 → void
01 → 0 complement
10 → 1 complement
11 → don't care

e.g.

a	b	c	
01	01	01	abc
a	—	b	
10	10	01	$\bar{a}\bar{b}c$
= 01	01	11	ab

when we do $ab \cdot \bar{a}$

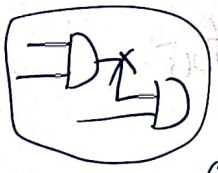
01	01	: ab	AND
10	11	: $\bar{a}$	
(00)		→ void indicates that the term will vanish	

∴ we can directly take AND of bits

. Logic synthesis:

1) Exact synthesis: — ROBDD : (performance XX)
— Quinn-McKluskey (performance ☐)

2) Testability :



① Algorithmic method
② Algorithmic "
③ SAT based
④ BDD based

$f(x_1, x_2, \ldots x_n) = x_i f_{x_i} \oplus \bar{x_i} f_{\bar{x_i}}$

faulty output f_α

Say $\alpha \to x_i$ stuck at 0. Then, $f \oplus f_\alpha = 1$

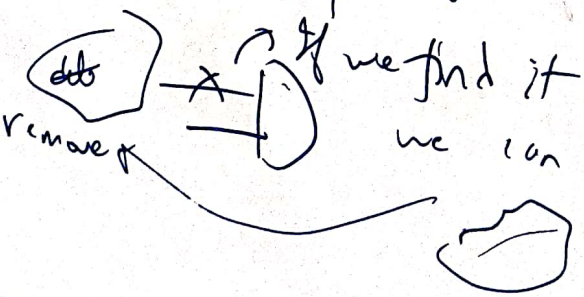
$\therefore x_i f_{x_i} \oplus \bar{x_i} f_{\bar{x_i}} \oplus f_{\bar{x_i}} =$

$= x_i f_{x_i} \oplus \bar{x_i} f_{\bar{x_i}} \oplus (x_i \oplus \bar{x_i}) f_{\bar{x_i}} = x_i (f_{x_i} \oplus f_{\bar{x_i}})$

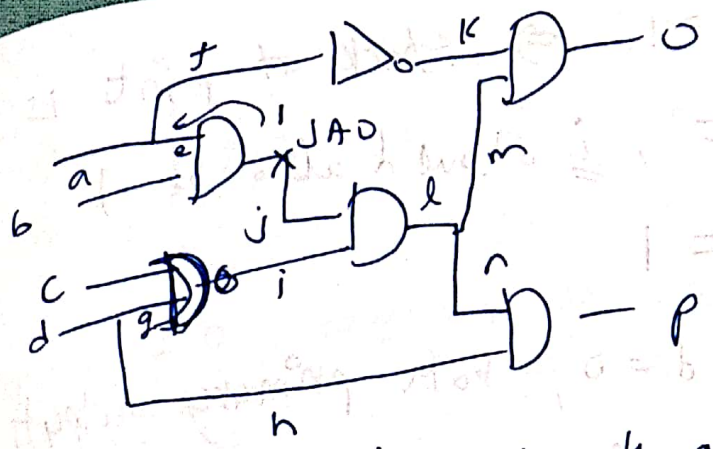
$\frac{\partial f}{\partial x_i}$

If no solution exists ⇒ redundant fault

We can remove the part of the circuit which leads to this redundant fault



If we find it is redundant, we can remove comb. logic



how to find primary inputs algorithmically

→ ① Sensitize a fault

② Assign objective : j = 1

③ Traverse a path to one of the inputs and assign a value : val objective ⊕ inversion parity

eg. if 3 NAND/NOR/NOT gates between point & input, inversion parity = 1 & hence apply input : $\overline{\text{val-obj}}$

if 0/2/4.. gates, parity = 0 ⇒ input = val-objective

∴ a = 1 since 1 ⊕ 0 = 1

↙ objective

⑤ Forward simulate (Assign primary input the calculated value)

⑤ check whether objective is satisfied.

Yes → Explore the output through fault propagated to primary output

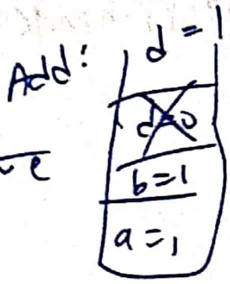
NO → go back to ③

In our case, set b = 1 ⇒ stuck at point is 1

Now, to propagate it, i should also be 1

New objective : i = 1

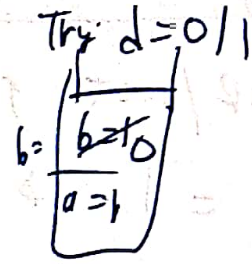
- But if we assign d = 0, both primary outputs will be 0



∴ ~~whenever we~~ flip latest value assigned to variable & evaluate it again

- If even after flipping, we can't ~~find~~ satisfy the objective, go down the stack & flip variables

∴ If d = 1 had failed :



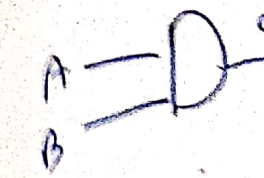
Called PODEM algo

- which path to choose ?

→ ~~Those~~ That which is most controllable

Primary inputs are the most controllable : say value = 1

~~Controllability (node) = max (Contr(i) ... ctrl(i) ... control(in)~~ min



$CO_A = 1$, $CO_A = 1$
$CO_B = 1$, $CI_B = 1$
} controllability ∪ 1 for all primary inputs

for $C = 0$, Any one can be 0.

$\therefore CO_C = 1 + \min(CO_A, CO_B)$

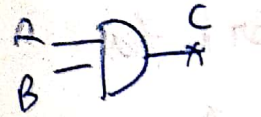
→ I will take any of the path

$CI_C = 1 + CI_A + CI_B$

+ since both should be 1 & hence it gets tougher

For which path ~~should also~~ leads to observable result:

~~But~~ ~~go~~



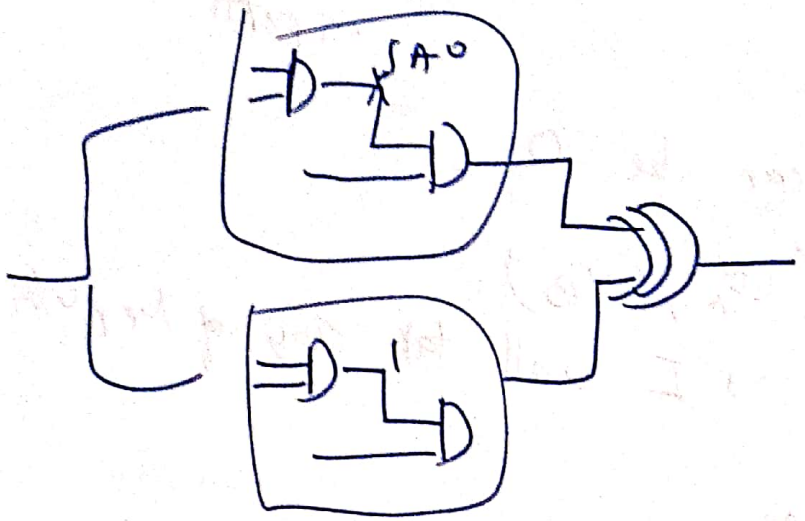
$O_C = 1$ (since primary output)

$O_A = 1 + O_C + CI_B$ → because B should ~~be~~ supply exactly that value which makes A observable

$O_B = 1 + O_C + CI_A$

we can also use observability of primary inputs to construct a ROBDD (lower the numbers ⇒ higher above in the tree ~~be~~ so that size of tree is reduced)

3) SAT based: Replicate the circuit



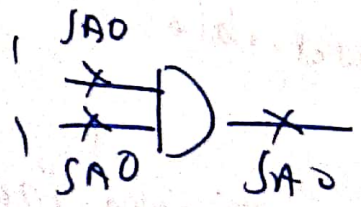
∴ while designing circuits, we must keep in mind:

1) Test generation is easy

2) # TV must be small

(Keep in mind that 1 TV can work for multiple faults) (So just reducing the no. of nets need not always work)

e.g.



1 is TV for all 3 faults

$f(x_1, x_2 \ldots x_n) = x_i f_{x_i} \oplus \bar{x}_i f_{\bar{x}_i}$

$= f_{x_i} \oplus \bar{x}_i \frac{\partial f}{\partial x_i}$

$= x_i f_{x_i} \oplus (1 \oplus x_i) f_{\bar{x}_i}$

$= x_i f_{x_i} \oplus f_{\bar{x}_i} \oplus x_i f_{\bar{x}_i}$

$= f_{\bar{x}_i} \oplus x_i (f_{x_i} \oplus f_{\bar{x}_i})$

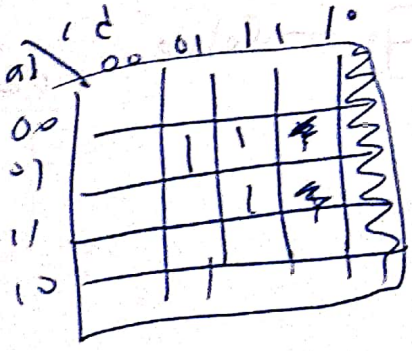
$= f_{\bar{x}_i} \oplus x_i \frac{\partial f}{\partial x_i}$ — positive Davio expansion

$= f_{x_i} \oplus \bar{x}_i \frac{\partial f}{\partial x_i}$ — -ve " "

↙ Similarly expand for rem. variables

. Once we have the expression in REED MULLER term, we need only (n+1) test vectors

e.g.



$\bar{a}bd \oplus abcd$

$= (1 \oplus a)\, bd \oplus abcd$

$= bd \oplus abd \oplus abcd$ ✓

Only XOR & AND gates required

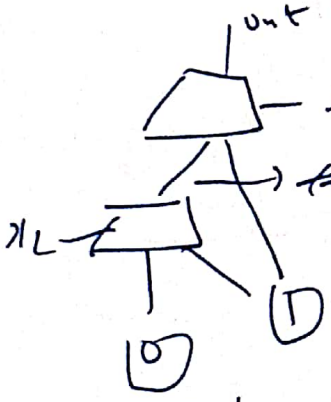
∴ If testability is most important, we will design it in such a way.

Circuit is 100% testable

Scanned by CamScanner

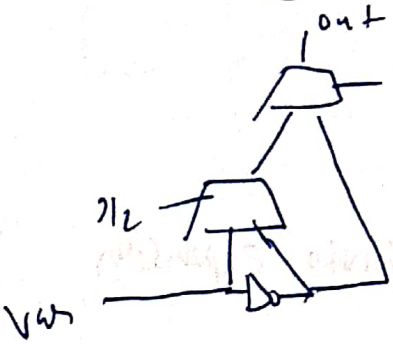
BDD is nearly 100% testable (since leaf nodes are static)

e.g.



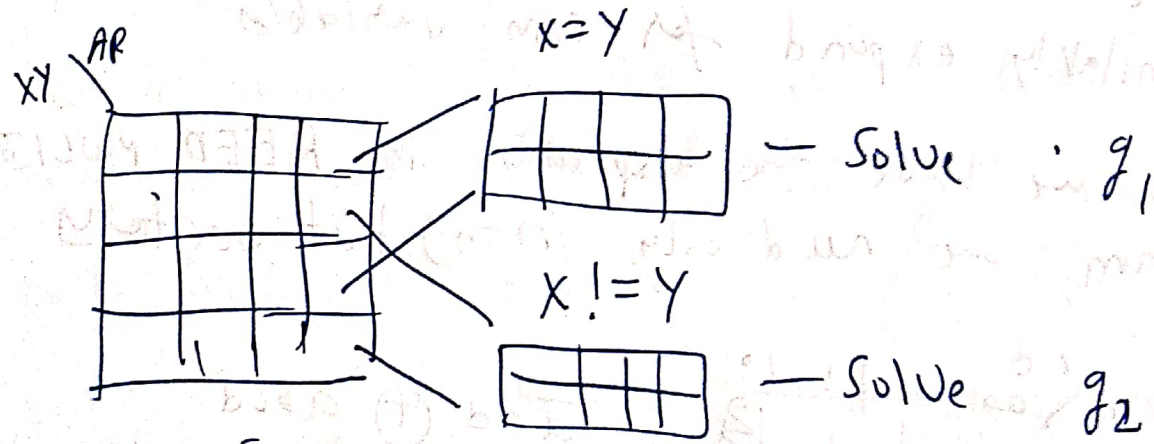
~~out~~ can't have 0 when $x_2 = 1$

Instead:

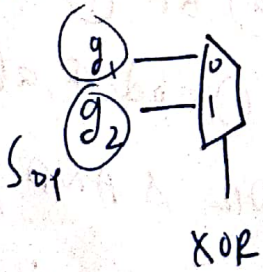


depending on var, we can cover all cases.

- Projecting on a plane: (EPSOP)



Then,



re

31/10/19

. Exact minimization is slow

. we can speed up the process by using some heuristics

→ 1) Give minimal circuit (need not the best)

2) Very fast → ≥ literals is minimum (be)

No enumeration of PIs :

(1) Expansion : If we have $ab\bar{c}$: we can expand it to ab if abc also exist

∴ we try to expand the PI  in all direction and whenever we hit a node which is not part of our SOP, we stop

(2) Reduction : Break down the term so that it can be accomodated in some other term

(3) Reshape : ~~Can~~ first reduce & then again expand

(5) Irredundancy check : Single cube containment.
check if a PI is wholly contained in another PI e.g. $ab\bar{c}$ in ab ~~is~~ and hence keeping both is redundant. keeping ab is enough

· ESPRESSO, PRESO, MIN1 are some of the tools

✱ Unate function: If a variable occurs in ONLY one of the forms (complement or uncomplement)

e.g. $\bar{x}\bar{y} + \bar{y}\bar{z}$, $a\bar{b} + bc + c\bar{d} + a\bar{d}$

· $f(x_1, x_2 .. x_n) = x_i f_{x_i} + \bar{x_i} f_{\bar{x_i}}$

$\bar{f} = \overline{(x_i f_{x_i} + \bar{x_i} f_{\bar{x_i}})} = \overline{x_i f_{x_i}} \cdot \overline{\bar{x_i} f_{\bar{x_i}}}$

$= (\bar{x_i} + \overline{f_{x_i}}) \cdot (x_i + \overline{f_{\bar{x_i}}})$

$= x_i \overline{f_{x_i}} + \bar{x_i}\,\overline{f_{\bar{x_i}}} + \overline{f_{x_i}}\,\overline{f_{\bar{x_i}}}$

$\therefore \bar{f} = x_i \overline{f_{x_i}} + \bar{x_i} \overline{f_{\bar{x_i}}}$ (consensus)

we can use recursion to find $\overline{f_{x_i}}$ and $\overline{f_{\bar{x_i}}}$

✱ Finding cofactors using positional representation:

$f = ab + bc + \bar{a}$

(10 11 11) AND
01 01 11
11 01 01
10 11 11

To find $f_{\bar{a}}$: we intersect it with $a = 0$

00 01 11 ✗ → because 00 = NULL
10 01 01 ✓
10 11 11 ✓

These have $\bar{a}$

$f_{\bar{a}}$ & f_a should be free of a

∴ we now take complement of $\bar{a}$ = 01 00 00

Now take OR with remaining terms:

10	01	01
01	00	00
11	01	01

10	11	11
01	00	00
11	11	11

↳ tautology

$\Rightarrow f_{\bar{a}} = 1$

we OR with complement so that remaining bits do not get disturbed. Only the ones we want to eliminate will become 11, others will remain the same. (This can be done wrt not just one literal but any general terms)

. we can use this to check for containment

eg. $f = ab + \bar{a}c$. $f_{bc} = 1$ will can be proved

∴ when bc is contained in $ab + \bar{a}c$

* To decide order of variables for expansion, choose the term which has least overlap with

4/11/19

- Multilevel : $p = ab + bc = (a+c)b$ common term
  $q = qe + ce = (q+c)e$ common subexpression

we can save area by reusing common terms

∴ Extracting a common sub-expression is the challenging task

① Algorithmic methods . ② rule-based

Boolean Algebra

- very computationally expensive

Linear Algebra

- Polynomial abstraction
- Faster but weaker

- Boolean substitution :

$h = a + bcd + e$, $q = a + cd$

$\therefore h = a + bq + e$ ✓ (since $a + ab = a$)

- Algebraic :

$t = ka + kb + e$, $t = kq + e$

Because $q = a + b$

$q + c + qc$
- gate

- Boolean properties 1) Complement
  2) Symmetric distribution laws
  3) Don't care sets

Algebraic Division

- $f_{dividend} = f_{divisor} \cdot f_{quotient} + f_{remainder}$
- Support of $f_{divisor}$ & $f_{quotient}$ is disjoint i.e. no common variables

e.g. $f_{dividend} = ac + ad + bc + bd + e$

$f_{divisor} = a + b$

Then, $f_{quotient} = c + d$ & $f_{remainder} = e$

$\{a, b\} \cap \{c, d\} = \phi$

- We can have same var. in quotient & divisor

* Algorithm: (for above expression)

$A = \{ac, ad, bc, bd, e\}$ (cubes), $B = \{a, b\}$

Take first term in divisor

(1) $i=1$: $C_1^B = a$, $D = \{ac, ad\} \Rightarrow$ quotient $= \{c, d\}$

(2) $i=2=n$: $C_2^B = b$, $D = \{bc, bd\} \Rightarrow$ quotient $= \{c, d\}$

Then, $Q = \{c, d\} \cap \{c, d\} = \{c, d\}$

$\therefore$ Remainder $= \{e\}$, $f_{quotient} = c + d$, $f_{rem.} = e$

✱ Given f_i & f_j, f_i / f_j is empty when:

① f_j contains a variable not in f_i — $f_i = a + b + d$, $f_j = bc$

② f_j contains a cube whose support is not contained in that of any cube of f_i

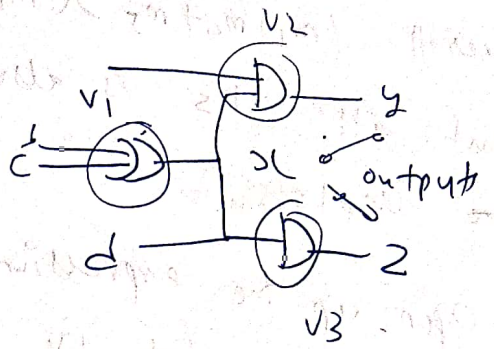
~~If any of the 2 conditions are met~~

③ f_j contains more terms than f_i — e.g. $f_i = a + b + bc$, $f_j = a + b + c + bc$

⑤ The count of any variable in f_j is higher than max length term in f_i — eg. $f_i = a + b + cd$, $f_j = abc$

✱ Library: we can replace common circuits (like $a\bar{s} + bs$ with mux) with known fixed/optimized implementations

Library	Cost
AND2	4
OR2	4
OA21	5



possibilities: $m_1 = \{V_1, OR2\}$

$m_2 = \{V_2, AND2\}$

$m_3 = \{V_3, AND2\}$

$m_4 = \{V_1, V_2, OA21\}$

$m_5 = \{V_1, V_3, OA21\}$

①	②	③	④
m_1	m_4	m_1	m_1
m_2	m_5	m_2	m_3
m_3		m_5	m_5
12	10	13	13 → cost

We can use vertex covering

V_1 : $m_1 + m_4 + m_5$

V_2 : $m_2 + m_4$

V_3 : $m_3 + m_5$

} But we can't directly use SAT since m_4 ~~will~~ cannot provide input to V_3

$\therefore$ m_2 requires m_1 ($\bar{m_2} + m_1$) & m_3 requires m_1 ($\bar{m_3} + m_1$)

∴ overall: $(m_1+m_4+m_5)(m_2+m_4)(m_3+m_5)(m_2+m_3)(m_3+m_5)$

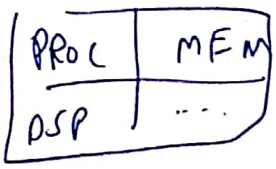
But all 1s is also a valid solution

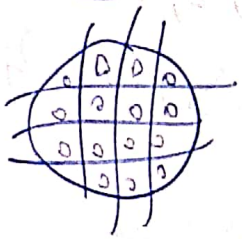
~~be~~ useless!

→ Open up the expression & get SOP terms

we can get cost by summing the costs of each implicant

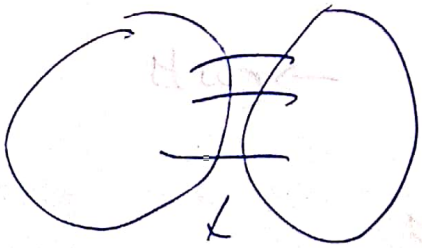
✡ Partioning & Floorplanning





Wafer is sliced so that a defect does not make entire wafer unusable

Interconnections between partitions should be minimized



Represent circuit as a directed acyclic graph

∴ We want to find K partitions of almost equal size so that cuts are minimum

Algo: Kernighan-Lin (KL)

. Iteratively swap two nodes & see if there is a decrease in # cuts

$D_a = -(\Sigma \text{internal edges} - \Sigma \text{external edges})$

↳ change in cut size by transferring node a from partition 1 to partion 2

$g_{ab} = D_a + D_b - 2C_{ab}$

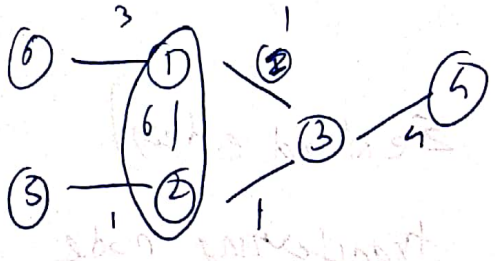
↳ since edge between (a,b) is counted twice & gain won't change after swapping them

↓
gain by swapping (a,b)

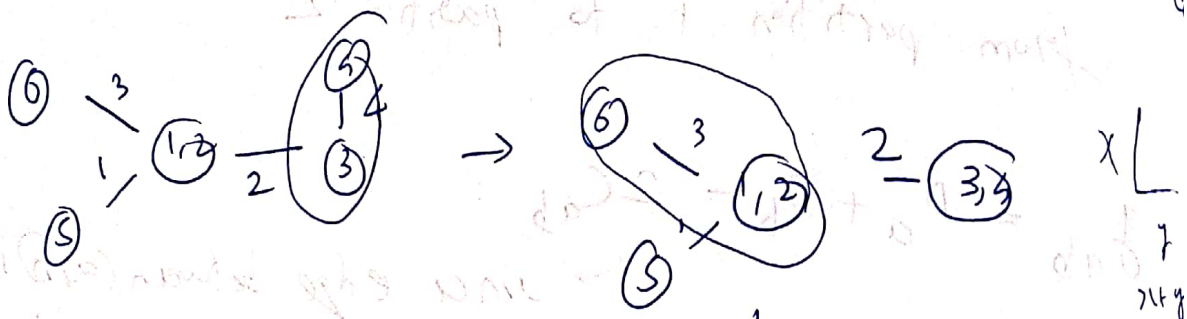
. After swapping (a,b), we need to update D's & g values for remaining nodes

. we do all swaps & check intermediate values of gain. wherever it was highest, we use those swaps only.

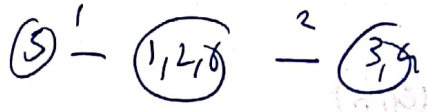
Clustering : (weights are the costs)



- Bottom-up process
- Nodes with a high cost are first collapsed into a super node



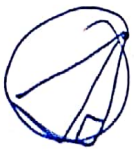
- we stop whenever ~~we hit the~~ we hit the required no. of clusters



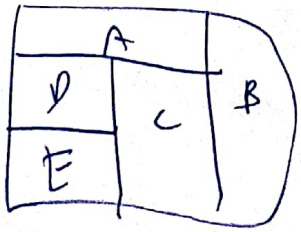
- This won't give equipartitions but we can impose a condition on the max. no of nodes in the supernode

Floorplanning : After partitioning, we will get all the modules which need to be implemented

- ~~We need~~ It also includes the decisions required for finalizing the shape
- once we have the shapes :



e.g.



⇒ 2D - bin packing problem

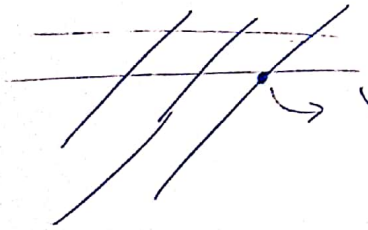
* Criteria used to measure quality of floorplan
1. Maximize routability
2. Minimize total length of wire
3. Minimize delay of max length wire

Or cooptimize : $\alpha A + (1-\alpha) L$

, use ILP on (x_{i1}, x_{i2}) → centre of rectangular module
(y_{i1}, y_{i2})

only constraint: Modules should be non-overlapping

* Routing : horizontal & vertical rails

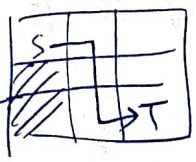


- Lower layers are local connection
- Top 2 are power & ground

- Global routing : Determine 'approximate' region
- Detailed routing : Place exact wires

- Picking the first net will determine all remaining decisions. ∴ 2 net in the critical path should be shortest.

* Grid:



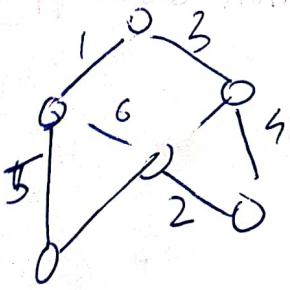
finding the path is the goal

used by some other module

we minimize the number of bends since each bend will need a connection via a via

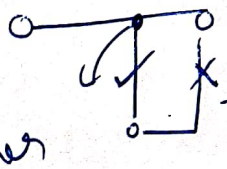
OR give appropriate weightage to # vias & # bends

- To connect nodes ~~A~~ together (short)



Use Kruskal's algorithm to find minimum spanning tree. Edges of this tree will be the final wires to be laid

- STei



→ will give longer path

Steiner point

better (shorter total length)