677 CAD & VLSI

✡ Books: ① Synthesis & optimization of Digital Circuits (R. to)

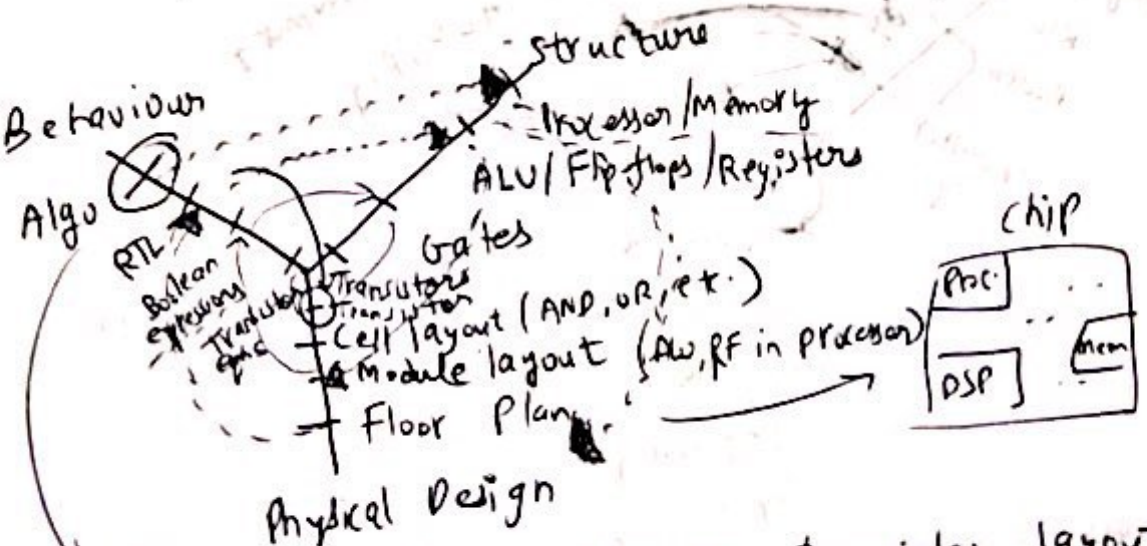
- Area + Performance + Testability + Power + Security + Intelligence (new)
- Testability - Design should be as open as possible
  Security - " " " closed "

✡ Computer Aided Design:

1. High-level synthesis → high-level algo to RTL
2. Logic Synthesis → RTL to gate-level
3. Physical Design → gate-level to PCB

Design

4. Design verification → correctness
5. Testing

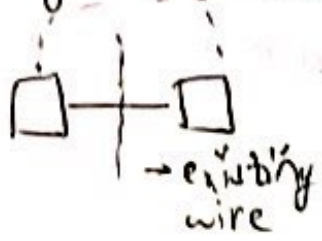


We have to go from Algo to transistor layout in a systematic order

- We can go from Algo to transistor eqn by following a spiral-like path

- We need a rough estimate at every level

- Delay incurred in wire is also crucial

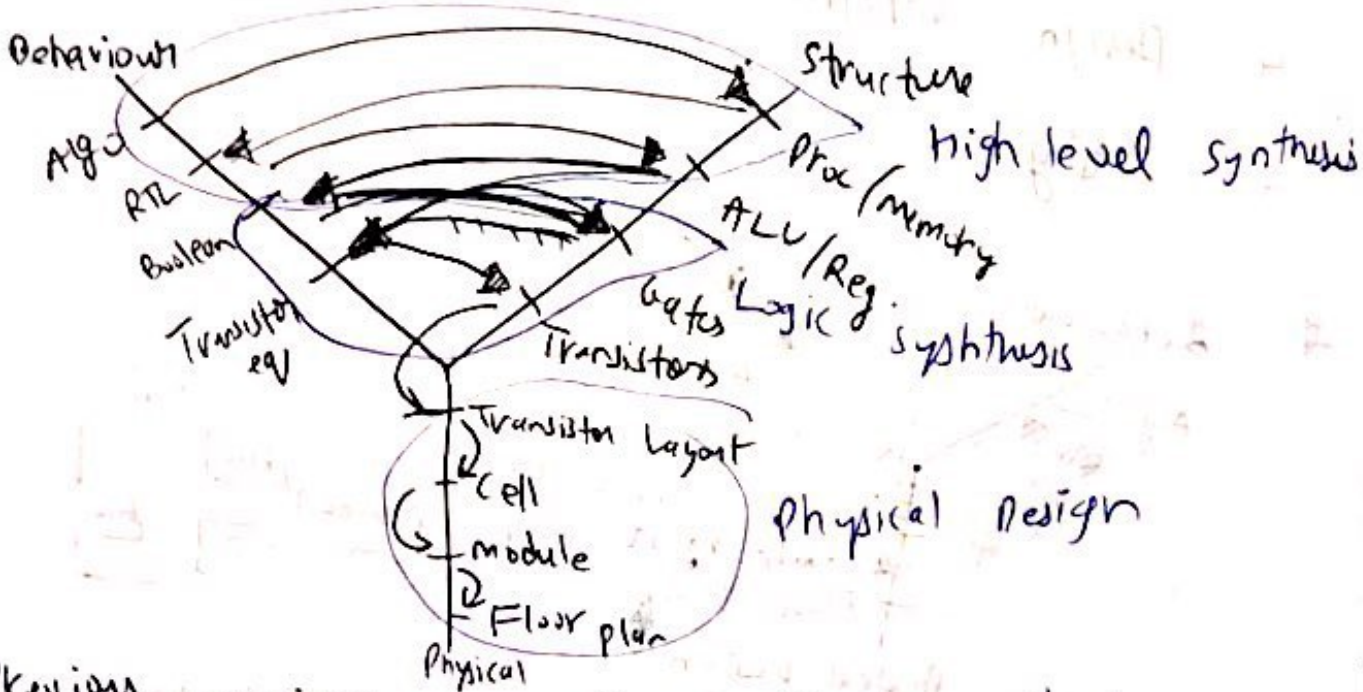


Routing has huge no. of possibilities

- Top-down approach is systematic.
  Gajski's Y-chart (discussed on the last page)

- Bottom-up is difficult because we don't focus on the specifications

☆ Hybrid approach :



Previous experience can help us skip some steps

Behaviour & Structural - Top-down

Physical Design - Bottom-up

- high-level synthesis output: Datapath & controllers
  → input to Logic synthesis & output: schematic

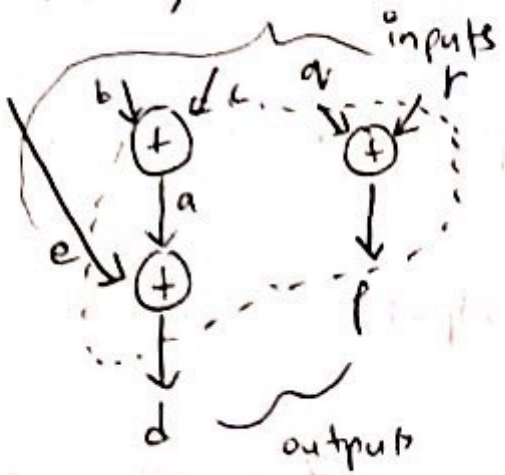
- Routing algorithms ~~are~~ give approximately best sol^n since iterating over all possibilities is ~~impossible~~ not practical.

- FPGA is better for low-volume products since fixed costs are very high

- FPGAs are much faster than simulation (slower than ASIC) & hence they are used for prototyping

14/8/19 Algos can also be represented graphically:

$a = b + c$, $d = a + e$, $p = q + r$



we can implement this algo by simply replacing each node with the corresponding ALU block

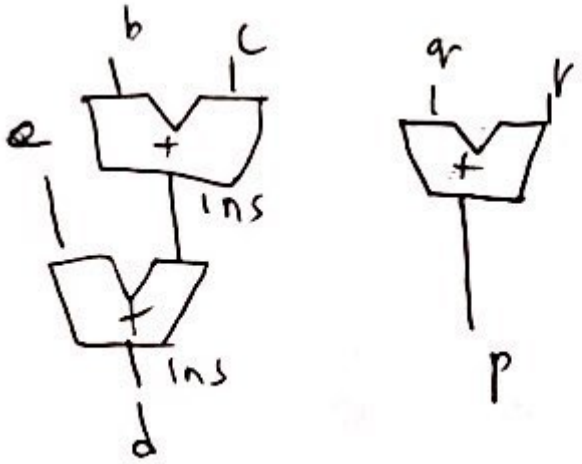
Resources required: 1) Functional units

2) Steering data

If asynchronous arrival of inputs: 3) Memory

- Optimized? Parameters are PPTASI

For now, let the parameters be Area, Performance



} optimized in terms of performance

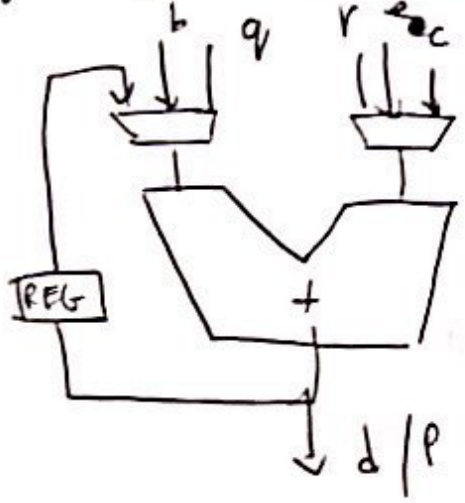
Not in terms of area

Delay = 2ns

∴ 3 units of area

2 ns delay

Using only one adder:



∴ Time taken = 3 ns : not optimal

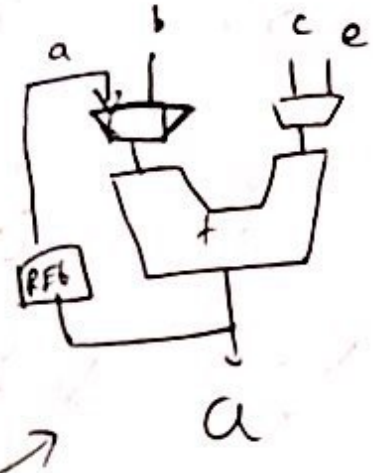
Area = 1 unit : optimal

we also need 5 control signals
(2 + 2 for MUX, 1 for register)

• Intermediate:

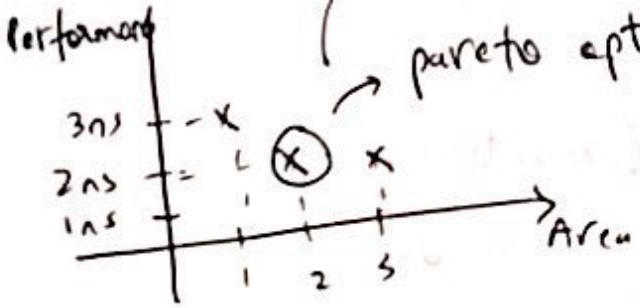
Time : 2 ns

Area : 2 units



(Assuming storing logic take up negligible space)

∴



pareto optimal

· Solving diff. equations: eg:

$$\frac{d^2y}{dx^2} + 3xy\frac{dy}{dx} + 3y = 0$$

$\frac{dy}{dx} = u$

$y(0) = y$

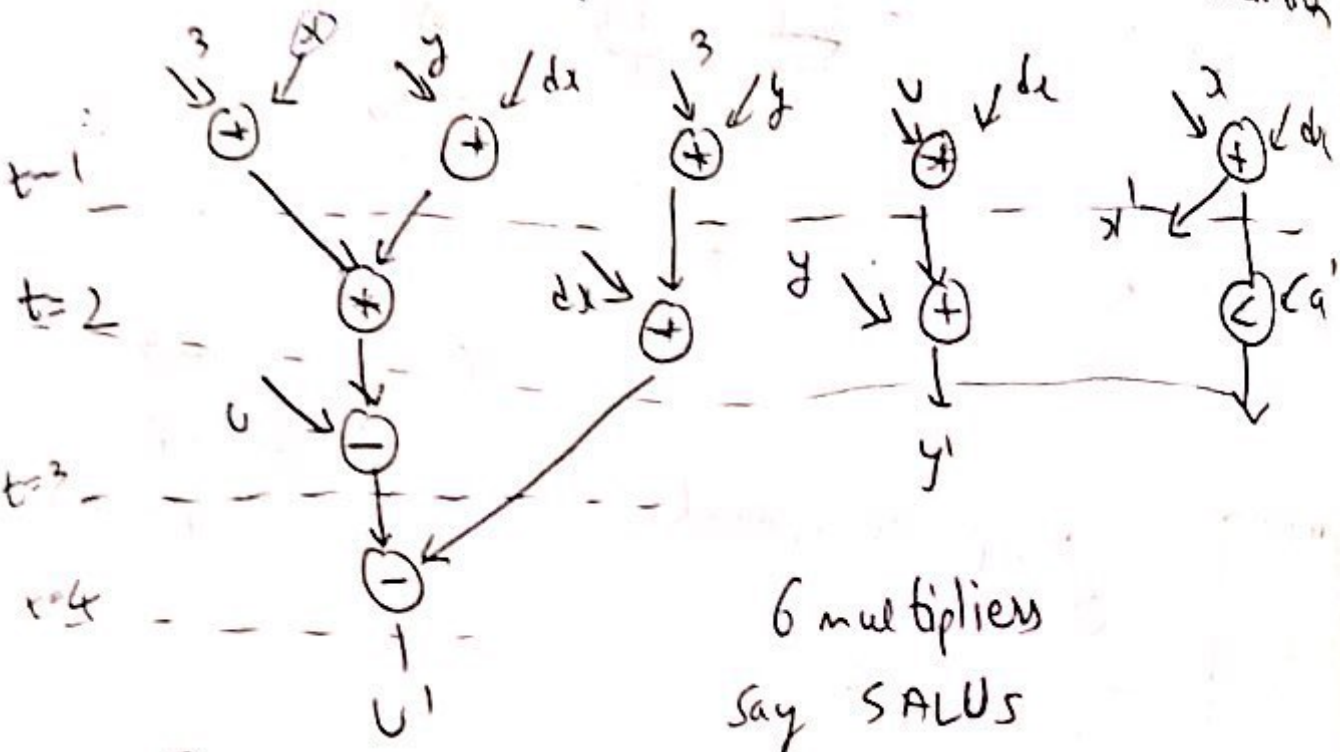
$x(0) = 0$

$x' = x + dx$

$y' = y + u\,dx$

$u' = u - 3xy\,dx - 3y\,dx$

$x < a$: Termination condition



6 multipliers

Say 5 ALUs

Huge design

Sequential graph

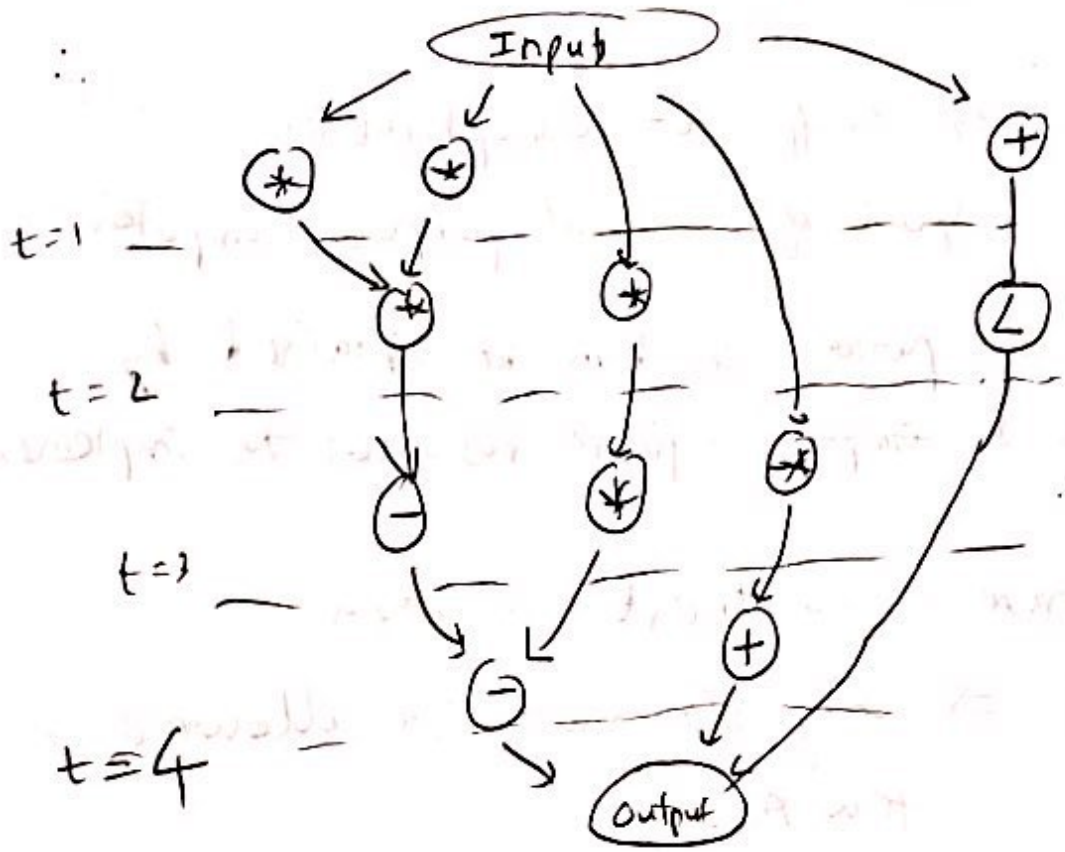
sequencing

We can reduce from 6 multipliers, 5 ALUs to 4 multipliers, 2 ALUs (since at one stage, we need only the components at that stage

e.g.

stage	1	2	3	4	
mul	4	2	–	–	→ max = 4 → 4 MUL
ALU	1	2	1	1	→ max = 2 → 2 ALU

However, we can further optimize by shifting the nodes downwards / upwards



But this was done intuitively. some systematic method has to be developed

- We need to also map the hardware to their job. placing of operation in space → binding
- placing of operations in time → scheduling

* optimal / Scheduling / Binding

Area | Performance

↓ ↙

Constraints: eg. 30 fps for video processing

optimization: Speed of general purpose computer

- Upper bound on performance can be obtained by longest path & ~~simple~~ infinite resources to implem

- We can have a constraint on Area

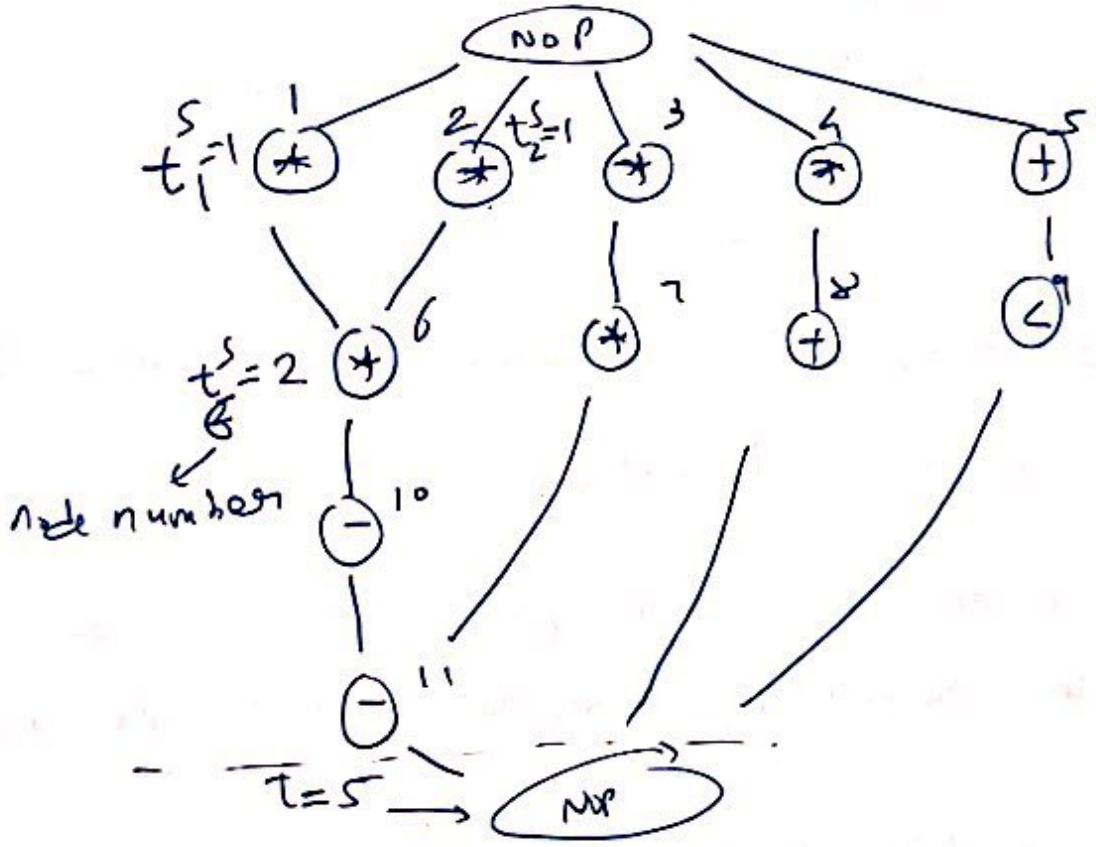
eg. ≤ 9 units ⇒ Only one multiplier allowed, few ALUs

∴ Place in time - scheduling - defines the state machi

Place in space - Binding - Defines the steering logic

19/8/19 Policy 1 : ASAP

As soon as a job finishes, assign next time ~~unit~~ slice to all its successor. Don't care about required resources. This is precisely what we did in the first case.

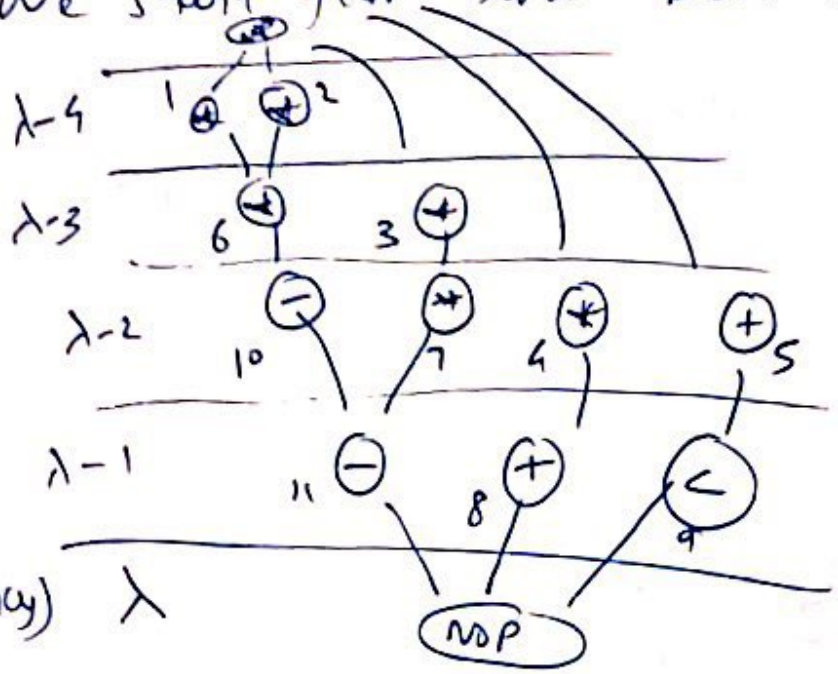


$$\text{Latency} = \lambda = t_{\text{last node}}^{\text{start}} - t_0^s = 5 - 1 = 4$$

ASAP will give an upper bound on performance

Policy 2: ALAP (As late AP)

We start from the bottom



2 Mult
3 ALUs
= 11 unit area

ASAP & ALAP will give same performance but different resource allocation

- Apart from the maximum path, we can move around the remaining nodes to optimize resources

- Now suppose performace constraint is not 4 but 5

∴ Node 1,2 can be moved up by maximum of 1 node

∴ Slack for 1,2 = 1

" " 3 = 2 (Can be done at t=1,2 also)

Slack = $t_i^L - t_i^S$ (L → ALAP time, S → ASAP time)

* We can use already-established optimization techniques to solve for optimal resources

· For ILP: eg. 400 wooden boards, 450 person hours

Chairs: 40 profit, 5 boards + 10 person hours

Table: 50 " , 10 " + 15 "

∴ A of x chairs & y-tables, profit = $40x + 50y$

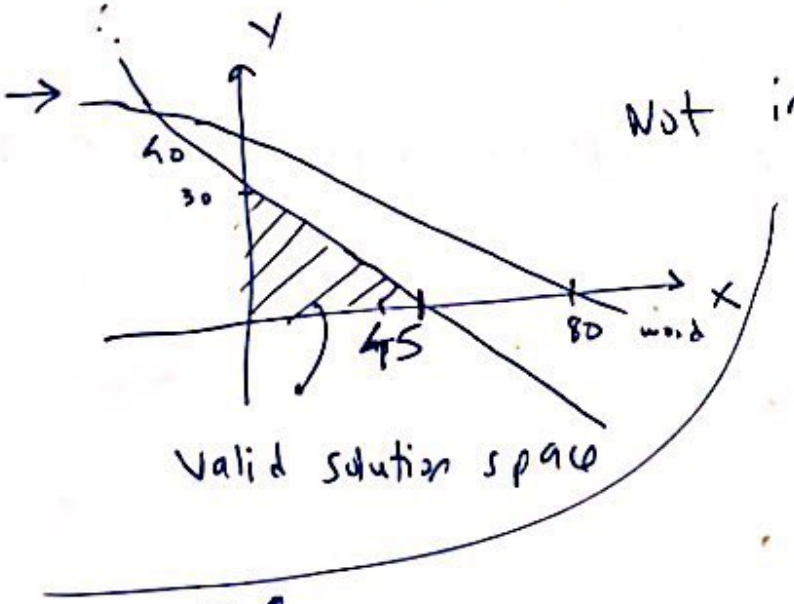
→ Goal: Maximize $\{40x + 50y\}$

Constraints: $5x + 10y \leq 400$ — Wood

$10x + 15y \leq 450$ — hours

$x \geq 0, y \geq 0$

∴



Not intersecting ⇒ not interesting

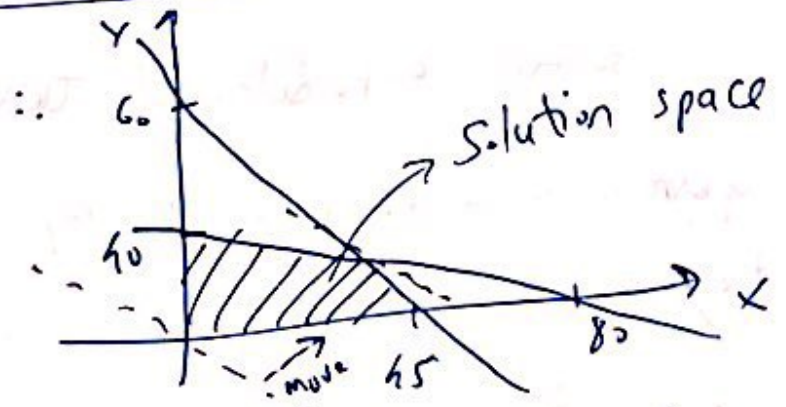
Valid solution space

$5x + 10y \leq 400$

$20x + 15y \leq 900$

$\text{Profit} = 40x + 50y$

$\therefore y = -\frac{4}{5}x + \frac{P}{50}$

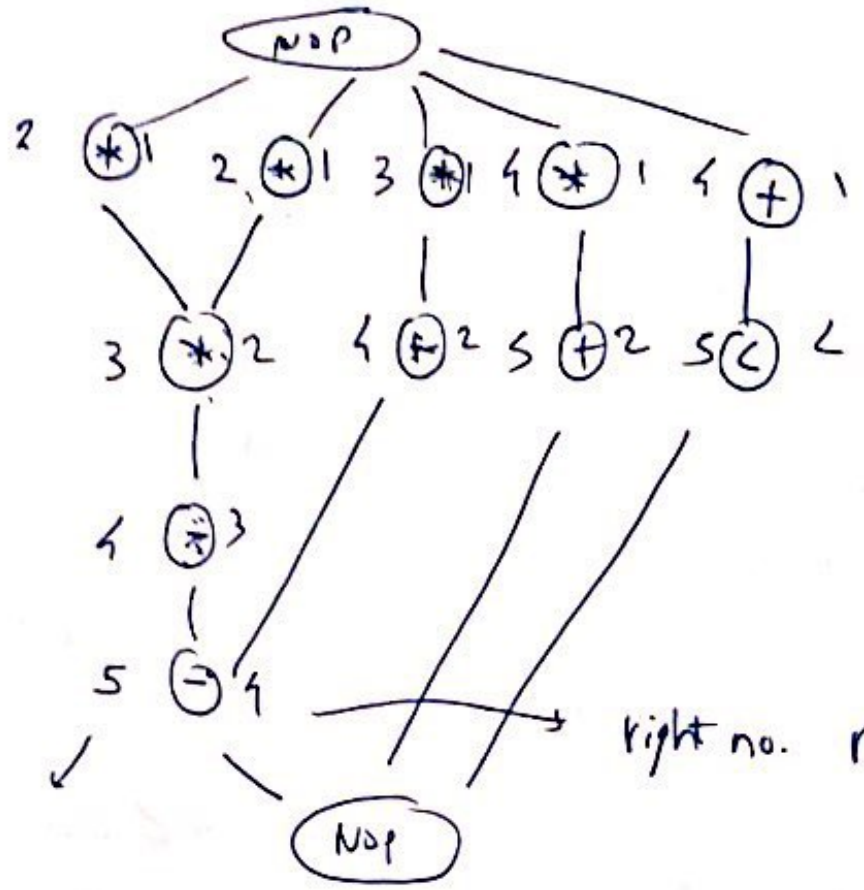


∴ Solution space

Start with $P = 0$ & slope $= -\frac{4}{5}$

Move the line away from origin

LP
- ILP : only integers
- MLP : mixed - some integers & some fractions
- ZLP : only 0,1 allowed (subset of ILP)



Suppose t=5
(Best performance is t=4)

right no. represents ASAP time

left no. represents ALAP time

Formulating the ILP:

Let x_{il} : operation i is ~~defined~~ scheduled at time l

$\therefore x_{il} = \begin{cases} 1 & \text{if operation } i \text{ is scheduled at step } l \\ 0 & \text{otherwise} \end{cases}$

Total : i : 1 to 5, l : 1 to 11 $\Rightarrow$ Total = 55 such variables

· Generating the Constraints:

1) $\sum_{l} x_{il} = 1$: Every operation is scheduled in ONLY one cycle

eg. $x_{11} + x_{12} + x_{13} + x_{14} + x_{15} = 1$

But ALAP & ASAP tells us that $l = 1$ or 2 ONLY

$\therefore x_{13} = x_{14} = x_{15} = 0$

$\therefore$ * $x_{il} = 0$ for $l >$ ALAP(i) and $l <$ ASAP(i) *

2) But we haven't considered dependencies i.e. ~~operation should be scheduled before~~ Dependent operation should be scheduled after completion of its predecessor

i.e. i^{th} operation is dependent on j^{th} operation and hence $l_i > l_j$

$\therefore$ * $\sum_{l} (x_{il}\, l) \geq \sum_{l} (x_{jl}\, l) + d_j$ *

d_j : time of execution of operation j

$\sum_{l} (x_{il}\, l)$: start time of operation i

$\sum_{l} (x_{jl}\, l)$: start time of operation j

eg: $1 : x_{311} + 2\, x_{312}$

For our given graph, $d_j = 1 \quad \forall j$

$$2x_{62} + 3x_{63} - 1.x_{11} - 2x_{12} - 1 \geq 0$$ — For node 6

This tells us that either $x_{11} = 1$ or $x_{12} = 1$ if $x_{63} = 1$

$$2x_{82} + 3x_{83} + 4x_{84} + 5x_{85}$$
$$-1.x_{41} - 2x_{42} - 3x_{43} - 4x_{44} - 1 \geq 0$$

For node 8 ↗

- Resource constraints:

At any point in time, suppose we cannot have more than 'a_m' multipliers & 'a_k' ALUs

Cycle 1: $x_{11} + x_{21} + x_{31} + x_{41} \leq a_m$, $x_{51} \leq a_k$

Cycle 2: $x_{12} + x_{22} + x_{32} + x_{42} + x_{62} + x_{72} \leq a_m$

$$x_{52} + x_{82} + x_{92} \leq a_k$$

⋮

OPTIMIZATION FUNCTION

Area: $\min(1.a_k + 4a_m)$

↘ If multiplier occupies 4 units of area
ALU " 1 unit "

. To maximize performance : we can minimize :

$\sum_i \sum_l (\gamma_{il}\, l)$ will automatically minimize l for longest path

Note: The above will give ASAP if area constraint is relaxed.

. We can also co-optimize area & performance :

Minimize(δ.Area + $(1-\delta)$ Performance)

Both should be normalized to the same scale

✱ Heuristic-based Scheduling :

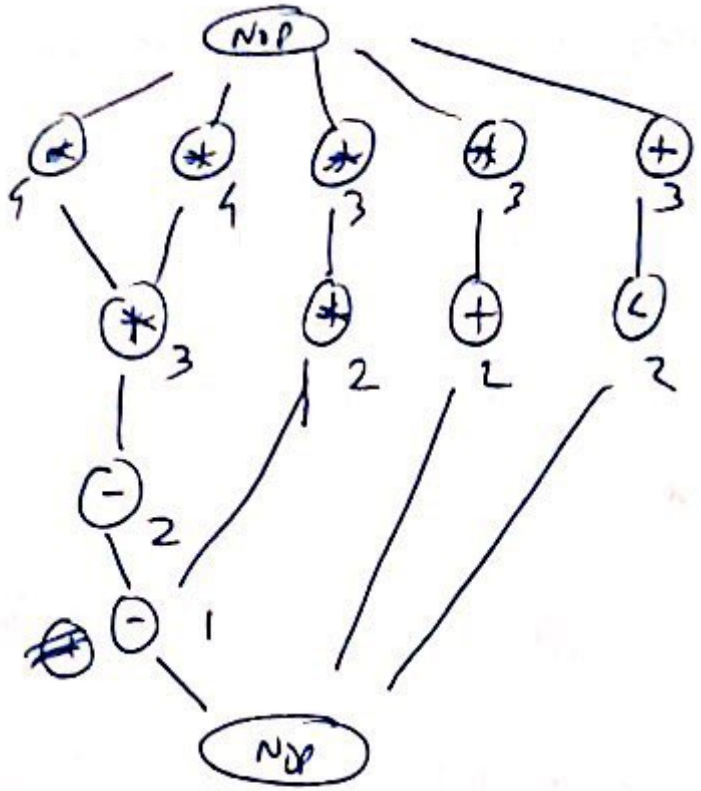
1. List-scheduling under resource constraint (2M + 2ALU)

$U1 = \{1, 2, 3, 4\}$ $\quad U2 = \{5\}$

U - unscheduled list - contains list of operations

. when set contains lesser operations than ~~it~~ constraints, no problem ✓

. when set contains more nodes to be scheduled : we can give more priority to nodes which lie in the longest path.



Calculate min dist. of each node from the bottom
higher the distance ⟹ Higher the priority

∴ $S_1^1 = \{1,2\}$ (1 → mul.) $S_1^2 = \{5\}$ (2 → ALU)

time

$U1 = \{3,4,6\}$ (7 not eligible) $U2 = \{9\}$ (8 not yet eligible)

$S_2^1 = \{3,6\}$ $S_2^2 = \{9\}$

$U1 = \{4,7\}$ $U2 = \{10\}$

$S_3^1 = \{4,7\}$ $S_3^2 = \{10\}$

$U1 = \{\}$ $U2 = \{8,11\}$

$\therefore S_4^1 = \{\}$ ✓ $S_4^2 = \{8,11\}$ ✓

✓

× — × — ×

This may not give the best but gives optimum sol^n

- We can also have a variant where operations take different no. of time steps

Say multiplication takes 2 time units:

$U1 = \{1,2,3,4\}$ $U2 = \{5\}$

$S_1^1 = \{1,2\}$ $S_1^2 = \{5\}$

- - - -

$U1 = \{3,4\}$ $U2 = \{9\}$

$P1 = \{1,2\}$ $P2 = \phi$

in progress

$|S|+|P| \leq a_M \Rightarrow S_2 = \phi$

↓ resources

- - - -

$U1 = \{3,4,6\}$ $U_2 = \phi$

$P1 = \phi$ $P2 = \phi$

$S_3^1 = \{3,6\}$ $S_2^2 = \phi$

- - - -

$U1 = \{4\}$ $U2 = \phi$

$P1 = \{3,6\}$ $P2 = \phi$

$|S|+|P| \leq 2 \Rightarrow S_4^1 = \phi$ $S_4^2 = \phi$

- - - -

$U1 = \{4,7\}$ $U2 = \{10\}$

$P1 = \phi$ $P2 = \phi$

$\therefore S_5^1 = \{4,7\}$ $\therefore S_5^2 = \{10\}$

$U1 = \phi$
$P1 = \{4, 7\}$
$S_6^1 = \phi$

$U2 = \phi$
$P2 = \phi$
$S_6^2 = \phi$

- - - - -

$U1 = \phi$
$P1 = \phi$
$S_7^1 = \phi$

$U2 = \{11, 8\}$
$P2 = \phi$
$S_7^2 = \{8, 11\}$

❖ Generating an ILP if different latencies:

1. Uniqueness will remain the same

2. Dependency: We had a d_j term which took care of the execution time

∴ If op. i is dependent on op. j:

$$\sum l x_{il} \geq \sum l x_{jl} + d_j \rightarrow = \begin{cases} 1 & \text{for addition} \\ 2 & \text{for multiplication} \end{cases}$$

eg. $2x_{62} + 3x_{63} + 4x_{64} + 5x_{65}$

$$- 1 \cdot x_{11} - 2x_{12} - 3x_{13} - 4x_{14} - 2 \geq 0$$

3. Resource constraints:

multiplier

① $x_{11} + x_{21} + x_{31} + x_{41} \leq 2$

②: $x_{01} + x_{21} + x_{31} + x_{61} + x_{12} + x_{22} + x_{32} + x_{42} + x_{62} + x_{72} \leq 2$

already running — new possibilities

✡ For different kind of multipliers:

2 ALUs: 1 latency, 1 unit area

2 multipliers: — 2 latency, 3 unit area

— 1 latency, 5 unit area

Use resource based scheduling ... k denote which multiplier

$x_{ijk} = 1$ if operation i is scheduled at time j on multiplier k

uniqueness: $(x_{111} + x_{112}) + (x_{121} + x_{122}) + \cdots (x_{151} + x_{152}) = 1$

⋮

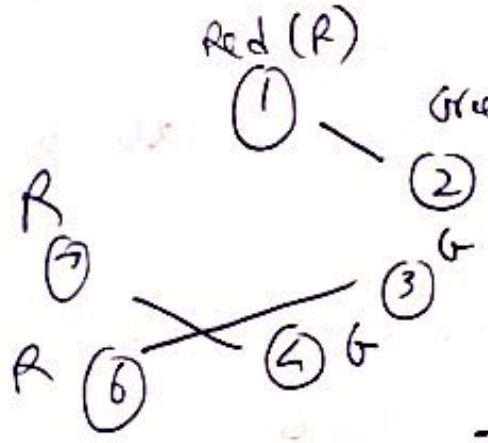
dependency: $\Sigma (x_{i21} + x_{i22})\, 2 \geq$

* Binding : Mapping logic units / multipliers to the tasks

- Sharing of operations is possible only when operations are not concurrent.

, We can map the problem to a graph-based problem

① Make a conflict graph FOR each operation, where edge (i,j) exists between node i's node j° if i^th's j^th operation are scheduled at the same time. For multiplication:



Now use graph colouring methods

Condition : No two nodes connected to each other can have the same colour

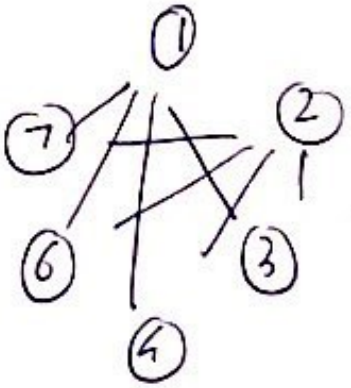
∴ Map colours → resources ~~to be mapped~~

e.g. red — M1, Green - Multiplier 2

② We can also solve the complement of the above method:

Compatibility Graph:

i,j are connected if operations i & j are NOT concurrent

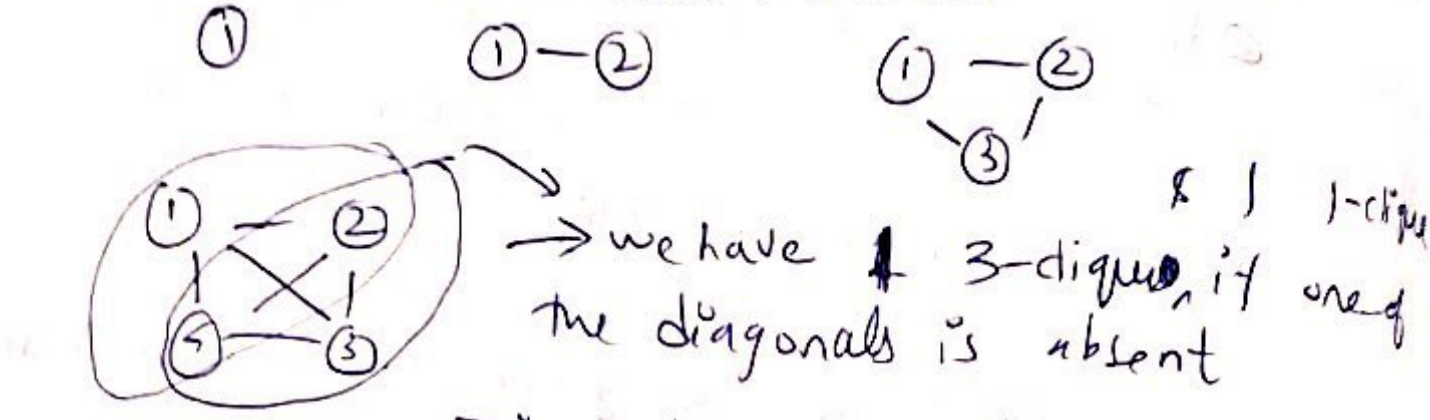


... continue

· Either of the two will be sparse & hence easier to process.

② continued: We can solve compatibility graph using the MIN CLIQUE method:

clique: COMPLETE subgraph within a graph
all nodes are connected to each other



Now that we have minimum no. of cliques, just assign each unit to one clique

∴ 

Resource 1

Resource 2

✡ BINDING is done post solving the ILP

· If there are multiple solutions to the clique problem, we can optimize for steering logic, reliability, etc.

e.g. V_{TP} changes if -ve bias is applied & hence if operated for a long time, delay increases (NBIT)

∴ workload among all the units must be distributed
i.e. Clique sizes should be uniform

✡ Co-optimization of area & performance:

$\int$. Area + (1-d) Performance (latency)

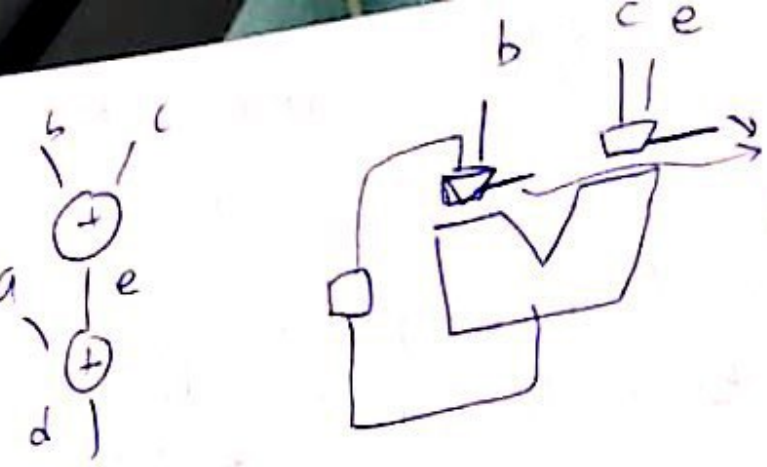
we need to normalize both so that they can be optimized together

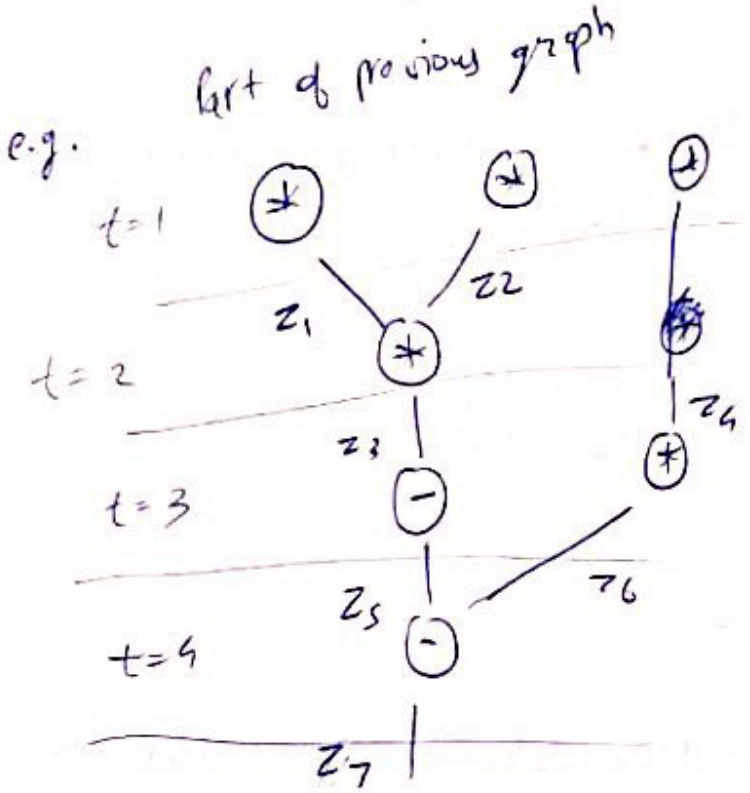
(Like feature scaling)

eg. we can divide by max area & max latency to get them on the scale 0-1 & now use

Co-optimization.

✡ once operations are assigned to resources, we develop the steering logic





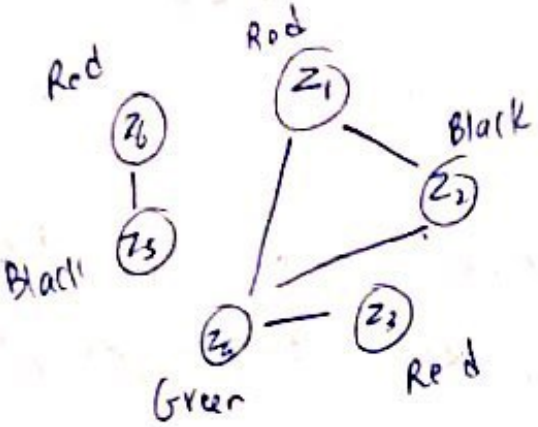
Any signal which is crossing the time boundary needs to be stored

- we can assign one register for each intermediate signal (∴ 6 registers in this case)

- we can minimize # by sharing the resources
  i.e. binding operations to physical registers

∴ variables having overlapping liveness range need different registers / resource

. We can use graph coloring again:
make conflict graph, i.e. edge(i,j) exists if node i & node j have overlapping liveness range.

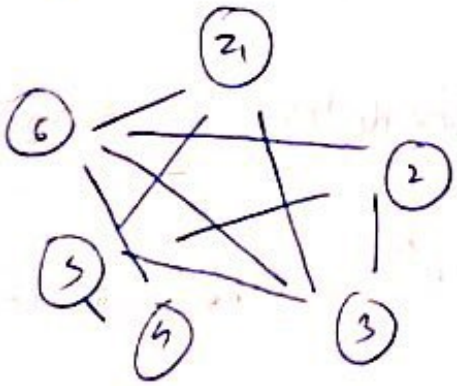


∴ 3 unique colours
⇓
3 registers required

∴ Red: z_1, z_3, z_6 will get the same register

. We can also use compatibility graph:
edge (i,j) exists if nodes i & j DON'T have overlapping liveness range

Solve for min-clique

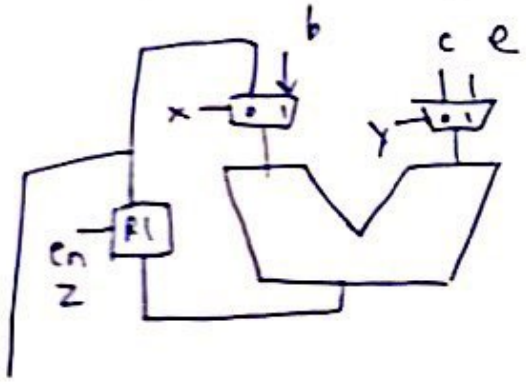


o we also have external variables which need to be stored in registers. 2 registers for u, v, y additional

* If steering logic area is small, we do sch... before binding.
But if it is significant, we need to do Concurrent scheduling & binding

* We need a state machine which specifies all control signals, outputs, next state logic, etc.

eg. $a = b + c$, $d = a + e$



	X	Y	Z
t=1	1	0	1
t=2	0	1	1

redundant always keeps 1

* Co-optimization of scheduling & binding:

$$x_{il} = \begin{cases} 1 & \text{if operation } i^{th} \text{ is scheduled in time step } l \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ik} = \begin{cases} 1 & \text{if } i^{th} \text{ operation is binded with } k^{th} \text{ resource} \\ 0 & \text{otherwise} \end{cases}$$

Constraints:

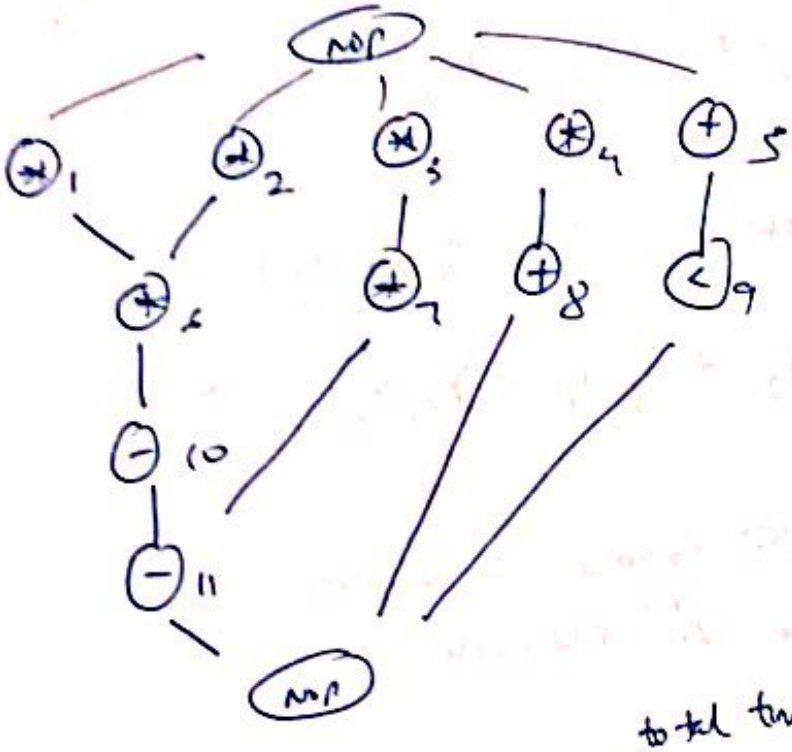
1) Uniqueness: $\sum_{t=1}^{T} x_{it} = 1 \quad \forall\, i$ (Each op scheduled exactly once)

(time)

$\sum_{r} y_{ir} = 1$ (spatial uniqueness)

i.e. if op 1 is assigned to 1st multiplier, it cannot be ~~ass~~ assigned to any other multiplier

For the graph we have already analysed:



total time = 5

$x_{11} + x_{12} + x_{13} + x_{14} + x_{15} = 1$

$y_{11} + y_{12} = 1$ } % resources 1, 2 are multipliers 3, 4 are adders

$y_{21} + y_{22} = 1$

$y_{53} + y_{54} = 1$

2) Dependency: $\sum l x_{il} - \sum l x_{jl} - d_j \geq 0$ if i^{th} op. is dependent on j^{th}

eg. 6 is dependent on 1 & 2:

$$2x_{62} + 3x_{63} - 1 \cdot x_{11} - 2x_{12} - 1 \geq 0$$

3) Resource constraint: $x_{11} + x_{21} + x_{31} + x_{41} \leq 2$ — multiplication

$$x_{12} + x_{22} + x_{32} + x_{42} + x_{62} + x_{72} \leq 2$$ —

4) Sharing of resources: (Non-linearity)

$$X_{11} Y_{11} + X_{21} Y_{21} + X_{31} Y_{31} + X_{41} Y_{41} \leq 1$$

first is scheduled — first resource is assigned

Similarly for the 2^{nd} resource:

$$X_{11} Y_{12} + X_{21} Y_{22} + X_{31} Y_{32} + X_{41} Y_{42} \leq 1$$

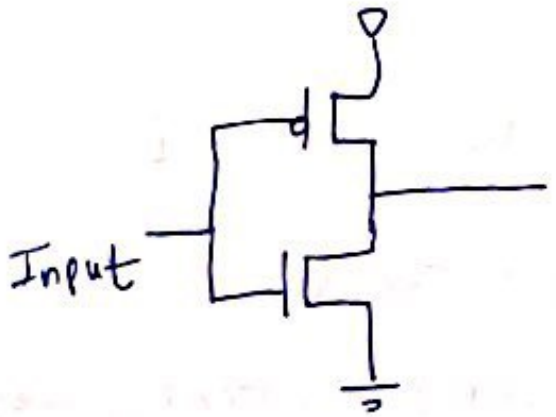
* Minimization objective:

$$\delta \sum l x_{il} + (1-\delta) \sum w_r a_r$$

performance — weighted areas of FU

- Control-dominated circuits will be optimized by considering steering logic

✡ Optimization for Power :



Unfortunately, inverter it has both static & dynamic leakage

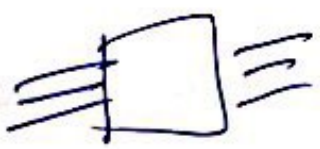
Ideally, 0 & ∞ resistance are expected

- Dynamic leakage ∝ No of Transitions / Toggles
- Static leakage ∝ # transistors

But # transistors is already taken care of in area

~ 10 -1. of total power

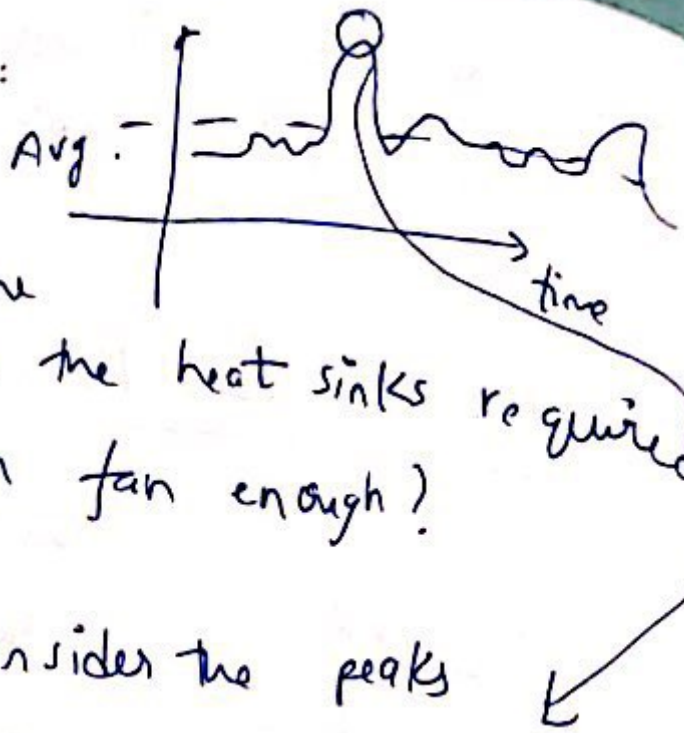
- Dynamic leakage is difficult to estimate since ~~it is~~ ~~de~~ toggling depends on input & Input is not fixed



Do simulations & plot toggles for some inputs

we only care about relative power dissipation

- Power is instantaneous:

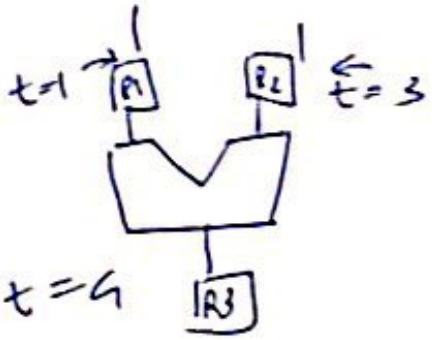


- Average power will ~~influence~~ determine the heat sinks required (e.g. cooling using liquids or fan enough?)
- But we need to also consider the peaks
- Since the wires supplying V_{DD} to the transistors (supply rails to inverter) can carry constant power, we will see a voltage drop.
- But current inverters run at around 0.8 V and we might wander close to 0.4 V (Threshold voltage)
- Also, delay $\propto \frac{1}{(V_{DD} - V_{TH})}$ $\Rightarrow$ increased delay

  V_{DD} decreased, V_{TH} fixed

  may cause synchronization problems
- 

∴ Peak power determines the power delivery network



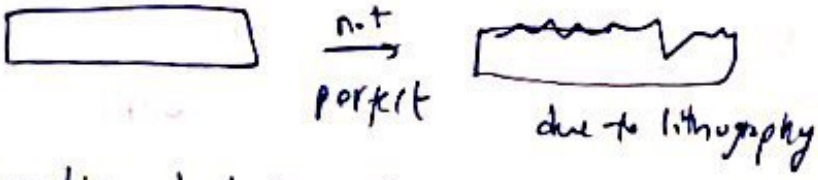
Even though we are not storing value of addition in R3, there is still power loss ~~from~~ at t=1 because since inputs have changed, combinational logic will still calculate the result.

So even though we want to store in R3 the result at t=4, power loss at t=1 also!

- Called spurious switching / useless power
- To reduce this, we need to ensure that switching occurs simultaneously.
- Register rebinding & scheduling will be required now.

☆ Testability : 1) Hardware defects

2) Algo defects eg. OR instead of XOR in the code



. (Systematic defects - Poor understanding of new ~~tech~~ technology)

. (Random defect - crack in the wafer
prevalent in mature devices)

Any defect will give rise to a logical ~~explanation~~ implication.

So, to test random defects :

(inverter does not give diff. output for x)

∴ Even though it is a ~~logical~~ error, doesn't matter manufacturing

☆ Stuck at Fault model :

→ either at 0 or 1. NEED NOT mean that they are directly connected to +5/gnd

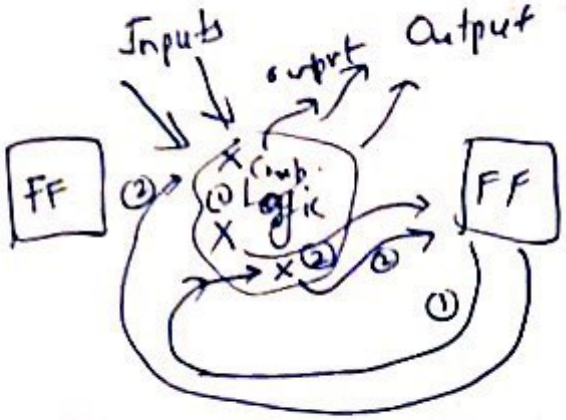
Only covers logical defects

• Total configurations of stuck at fault : 3 (2B if we assume 2 billion nets)
→ either SA0, SA1, no fault

But multiple SAF vectors are mainly the same as that of single SAF

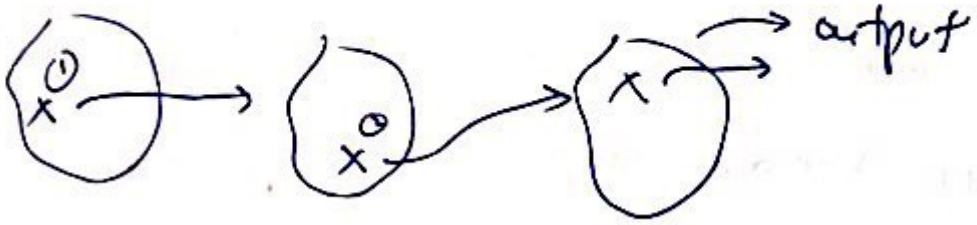
∴ single stuck at fault is good enough assumption

- Much bigger headache when it comes to sequential circuits



we may not be able to always propogate the fault to a primary output

So, we need to keep passing it through the logic till we are able to propogate it to the output

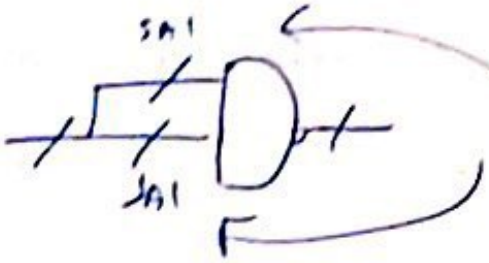


Unrolling increases the complexity

- Sequential depth: Max. no. of FF between any pair of input & output.

Since more the no. of FF (higher sequential depth), increases the Unrolls required, we should keep this in mind during design

e.g. binding of registers should be done in such a way that sequential depth is minimized



Total 8 possible SSAF.
only 2 are redundant. ($\because (\bar{x}.x) = \bar{x}$)
All others are testable

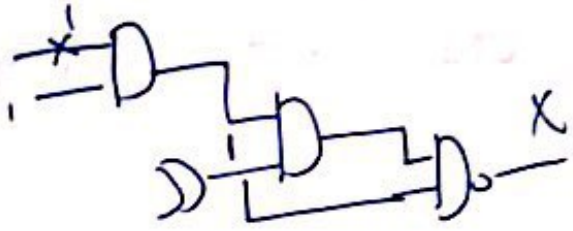
* Fault coverage = $\frac{\#\text{ detectable faults (Test vectors exist)}}{\#\text{ Total Faults}}$
FC

Fault Efficiency = $\frac{\#\text{ detectable}}{\#\text{ Total} - \#\text{ redundant}}$
FE

- In this case, FC = $\frac{6}{8} \times 100 = 75\%$.

FE = $\frac{6}{8-2} \times 100 = 100\%$.

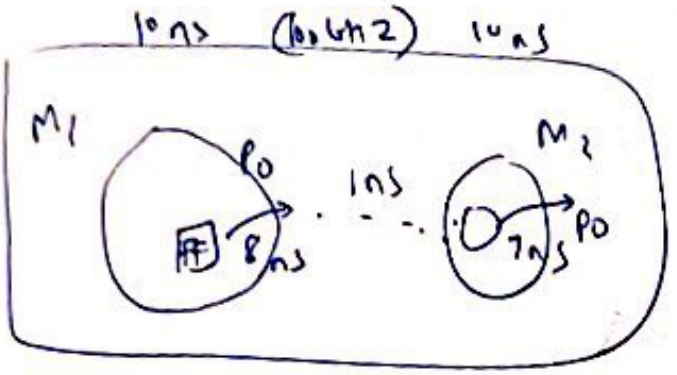
- Redundant faults are those which don't cause any logical malfunctioning.
- Some faults can be malfunctioning, yet a test vector won't exist in some cases.



- Calculating test vector for redundant ~~fault~~ faults may take a very long time. So, it we stop midway, we don't know if its redundant or not.

But our FE fails since # redundant faults ↓

* More problems in sequential circuits:

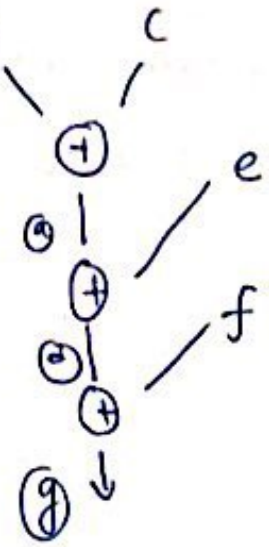


PI → FF
FF → PO
PI → PO
FF → FF

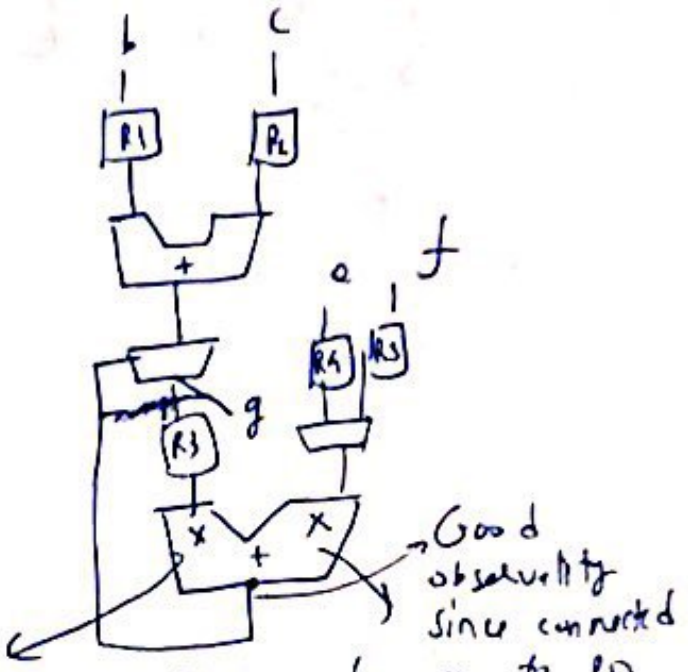
Although M1 & M2 can separately operate at 100 MHz, If we directly connect ~~them~~ them, total delay = 16 ns & hence we can't use 100 MHz clock

Standard practice: Use FF as ~~inter~~ intermediate storage We will need ~~more~~ 2 clock cycles to pass data from one module to the other but clock freq. will remain high

e.g.



⇒



. To test for fault, we cannot directly apply 4 to PO primary input ~~& par~~. It has to be passed through an adder first. ∴ Not easy to control

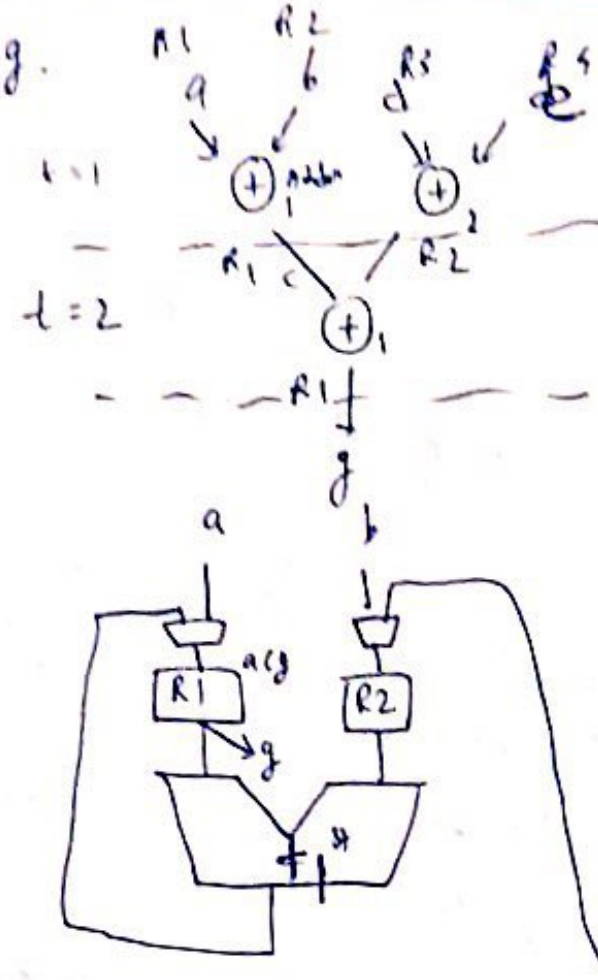
. But the fault on the right can be easily tested since inputs can be directly driven

1. Reduce sequential depth ⟵ scheduling & binding both affect this
2. Controllability enhancement } Register-binding optimization done during
3. observability "

. In this case, we can observe the output & hence the faults can be atleast observed

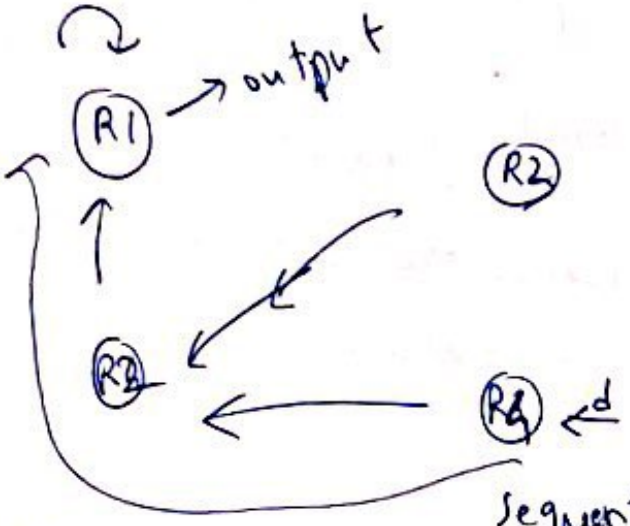
∴ Good observability

e.g.



for choice between registers at $t=1$ & $t=2$

We put R1 after multiplexer because all inputs must be registered (discussed on last page)

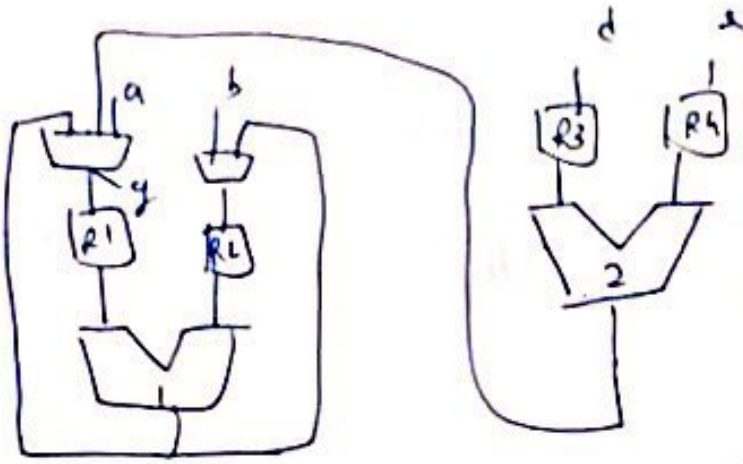


paths from register to register

Sequential depth = 3 since minimum registers a bit will pass through in longest path is 3 (R4 → R2 → R1 → output)

(max. no of registers between a pair of i/p / output)

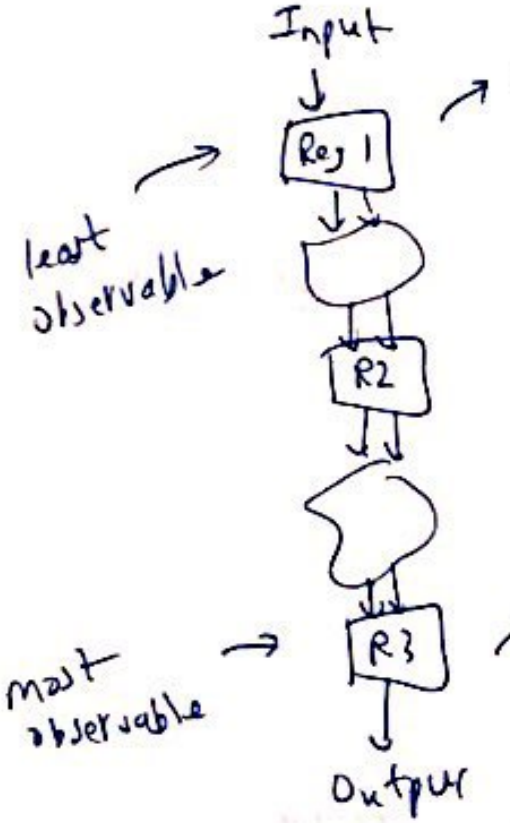
If we simply swap R1 & R2 at t=2:



All are controllable since each register directly connected to primary input.

Now, sequential depth = 2 not 3 but at the cost of a 4 to 1 mux instead of 2 to 1 MUX

Recap of observability & controllability



least observable

most controllable since we can directly drive in values from input

most observable

least controllable since we need to backcalculate the input which can give the desired output

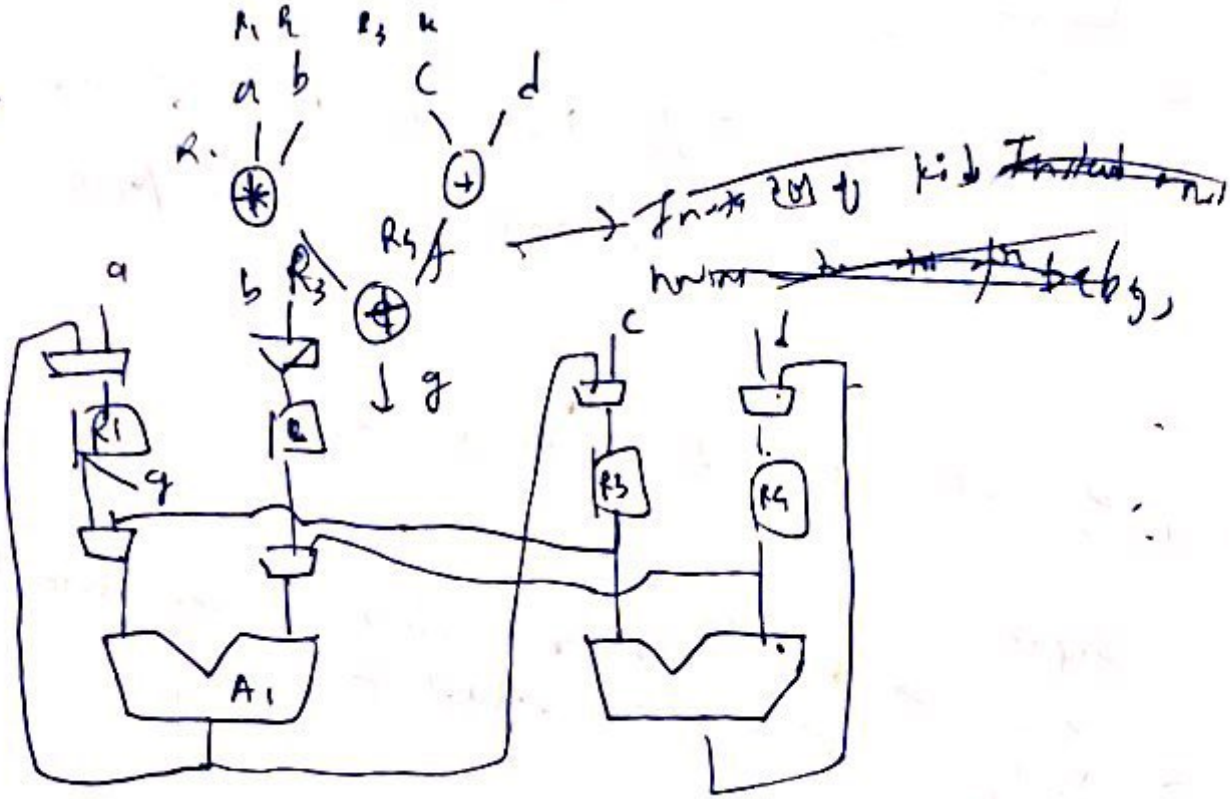
∴ No. of registers between primary input & current register $\propto \frac{1}{\text{Controllability}}$

registers between output & current registers $\propto \frac{1}{\text{observability}}$

· Heuristic for assigning registers: (Left Edge Alg.)

Start from t=1, scan from left to right & pick one which has emptied recently

e.g.



Sequential depth = 2

write again

Now: Modify LEA to increase testability (register binding

- More priority is given to reduction of sequential depth
- Testability is dependant on scheduling, register binding & resource binding

* Security : hardware attacks are now popular

RSA, AES is mathematically impossible to break!

But side-channel attacks such as power probing, etc. are used

Clearly, testability vs security is a trade-off

testability ↓ wants the circuit to be open for probing

security ↓ closed down as much as possible

∴ Assuming $N = 4$:

$$x_{11} + x_{21} + x_{31} + x_{41} + \underbrace{x_{73} + x_{43}}_{\text{can be executed at } t=3} \leq a_{max}$$

$$x_{62} + x_{72} + x_{32} + x_{42} \leq a_m$$

Designing the functional units :

① ON set : input value which evaluate to 1
OFF set : complement

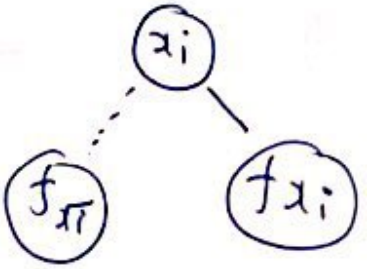
~~but~~ Instead of storing entire truth table, store whichever set is smaller

. It is a canonical representation

② Truth table - not scalable

③ BDD : based on Shannon's expansion :

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



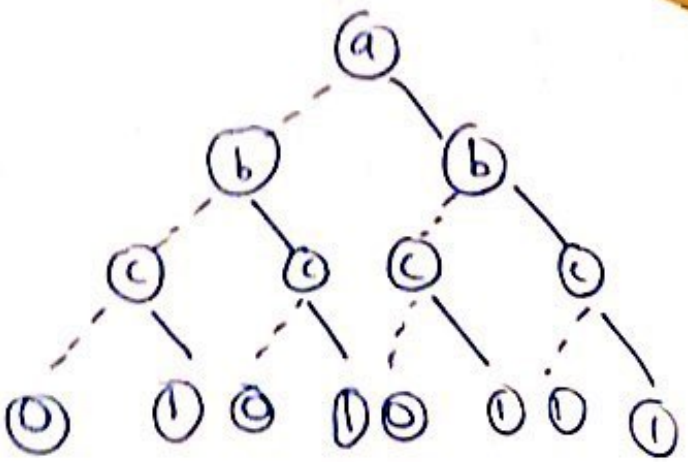
$$f_{x_i} = f(x_1, x_2, \ldots x_{i-1}, 1, x_{i+1}, \ldots x_n)$$

Similarly for $f_{\overline{x_i}}$

Continue with x_j as the next decomposition variable

e.g.

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



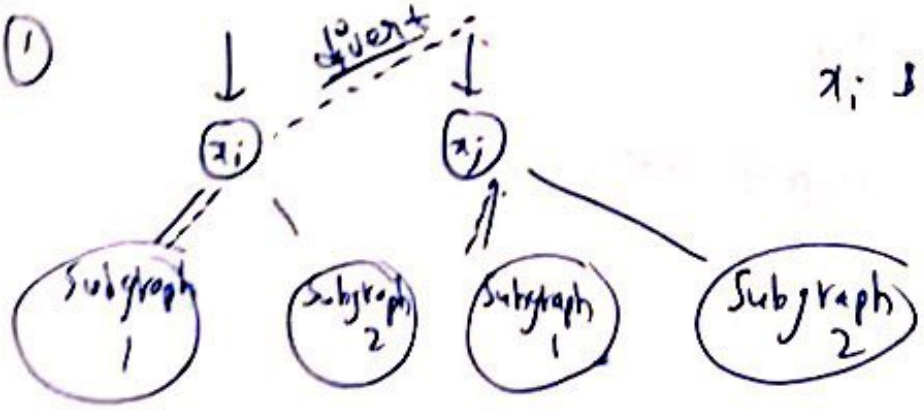
Total nodes = $2^n - 1 + 2^n$

($2^n - 1$: internal nodes; 2^n: leaves)

We can store only 2 leaves but still, total of $2^n + 1$ no

Removing redundancies:

(1)



x_i & x_j are isomorphic

(2)

→ redundant. No decision-making at this node

- Clearly, it is ordered since it depends on the order of variables chosen

. For many functions, the ROBDD grows linearly (for some like multiplier, it is still bad) but still better than truth table, which grows exponentially.

. we can compare 2 functions $f_1 \stackrel{?}{\equiv} f_2$ by comparing the trees & hence test a new implementation

. (a) But how can we start contracting/reducing it if it starts with $2^{n+1} - 1$ nodes?

→ Use decomposition property to build it incrementally

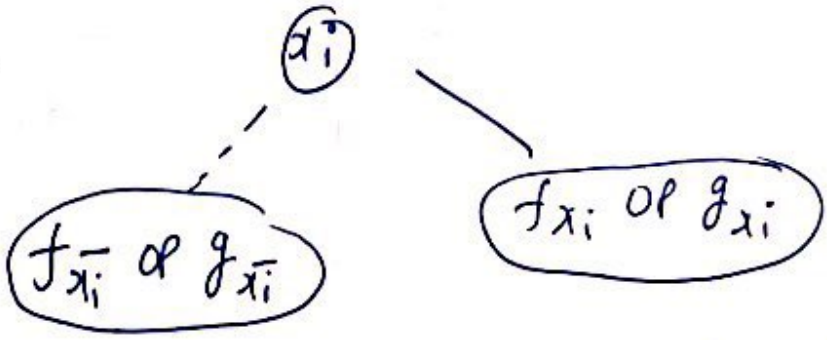
(1) Base cases ✓ (2) Combination properties ✓

Use some sort of DP algo.

Let $f = x_i f_{x_i} + \bar{x_i} f_{\bar{x_i}}$

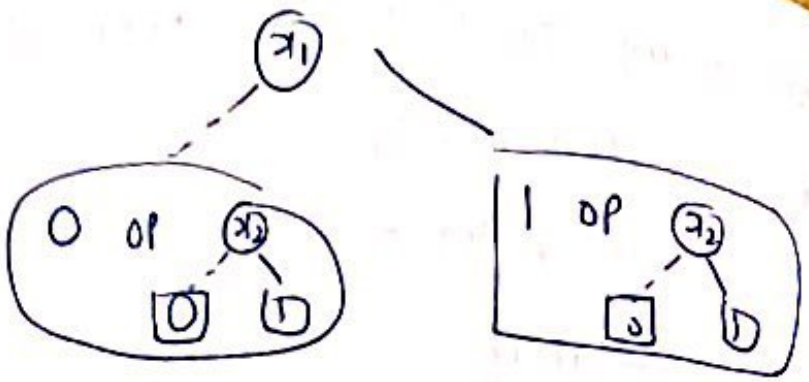
$g = x_i g_{x_i} + \bar{x_i} g_{\bar{x_i}}$

Then, f op g :

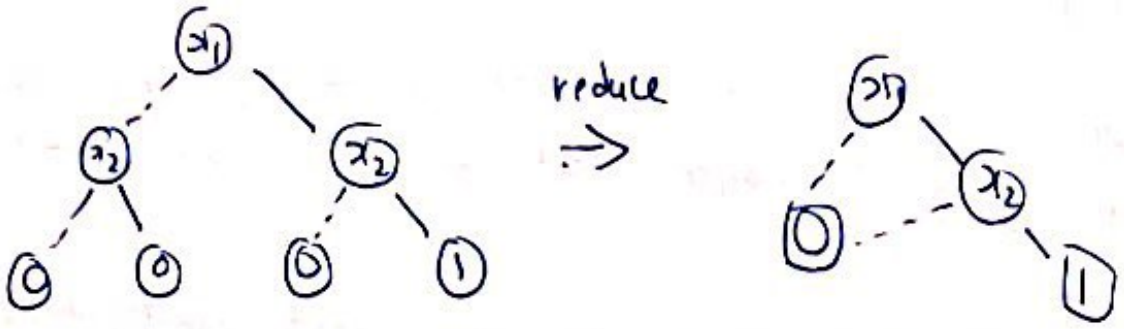


when g itself is independent of x_i, $g_{\bar{x_i}} \equiv g_{x_i}$

e.g. $x_1 . x_2$:

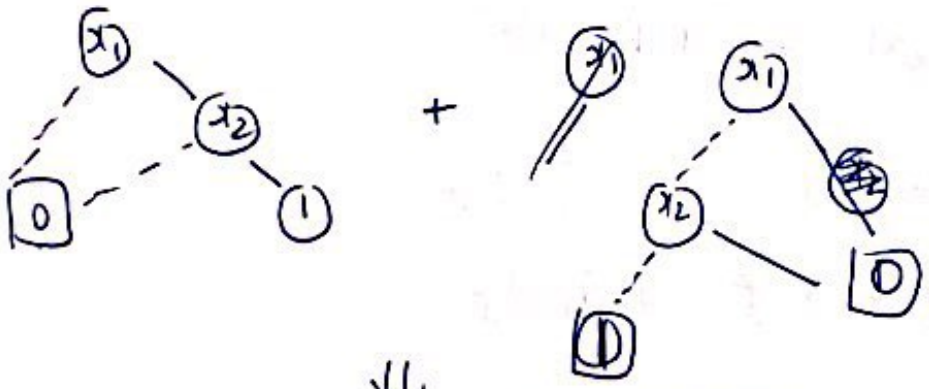


→

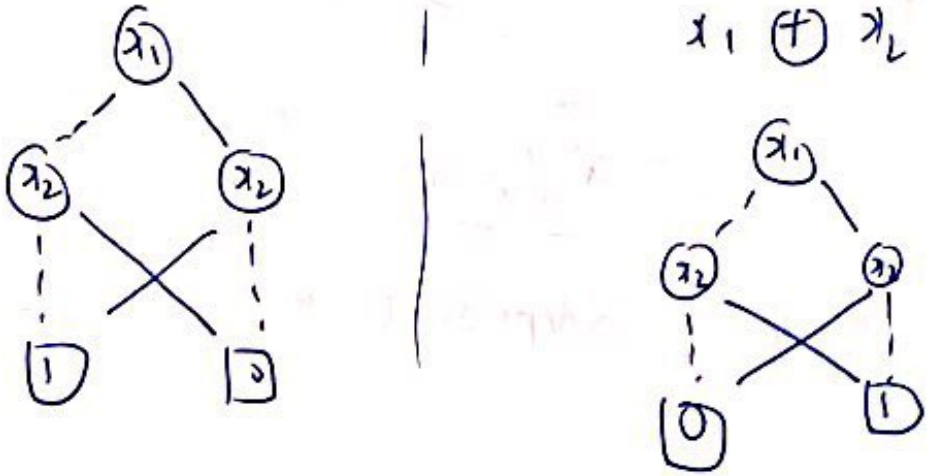


- Taking complement is as simple as swapping the nodes

- $x_1 x_2 + \bar{x_1} \bar{x_2} = x_1 \odot x_2$:



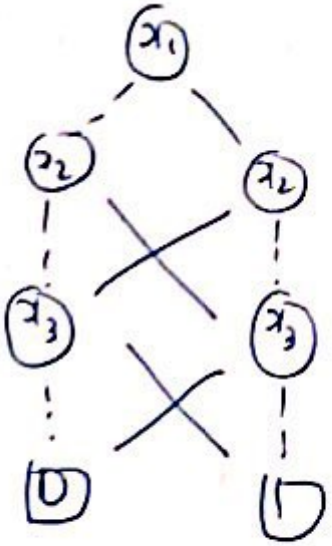
⇓



Total nodes = 5

$x_1 \oplus x_2 \oplus x_3$:

Total nodes = 7



∴ $x_1 \oplus x_2$: 5 nodes

$x_1 \oplus x_2 \oplus x_3$: 7 nodes } linear increase in # nodes

$x_1 \oplus x_2 \oplus x_3 \oplus x_4$: 9 nodes

Many common functions such as adder have XOR's XNOR operations !!!!

∴ Growth of ROBDD is linear

• (Doesn't scale for multiplier)

• Can be implemented in circuit by replacing all nodes with muxes.

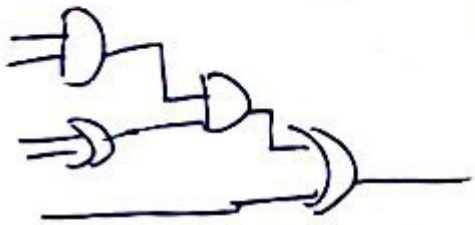
∴ Applications of ROBDD:

1) Equivalence checking : $f_1 \stackrel{?}{\equiv} f_2$

2) Implement directly

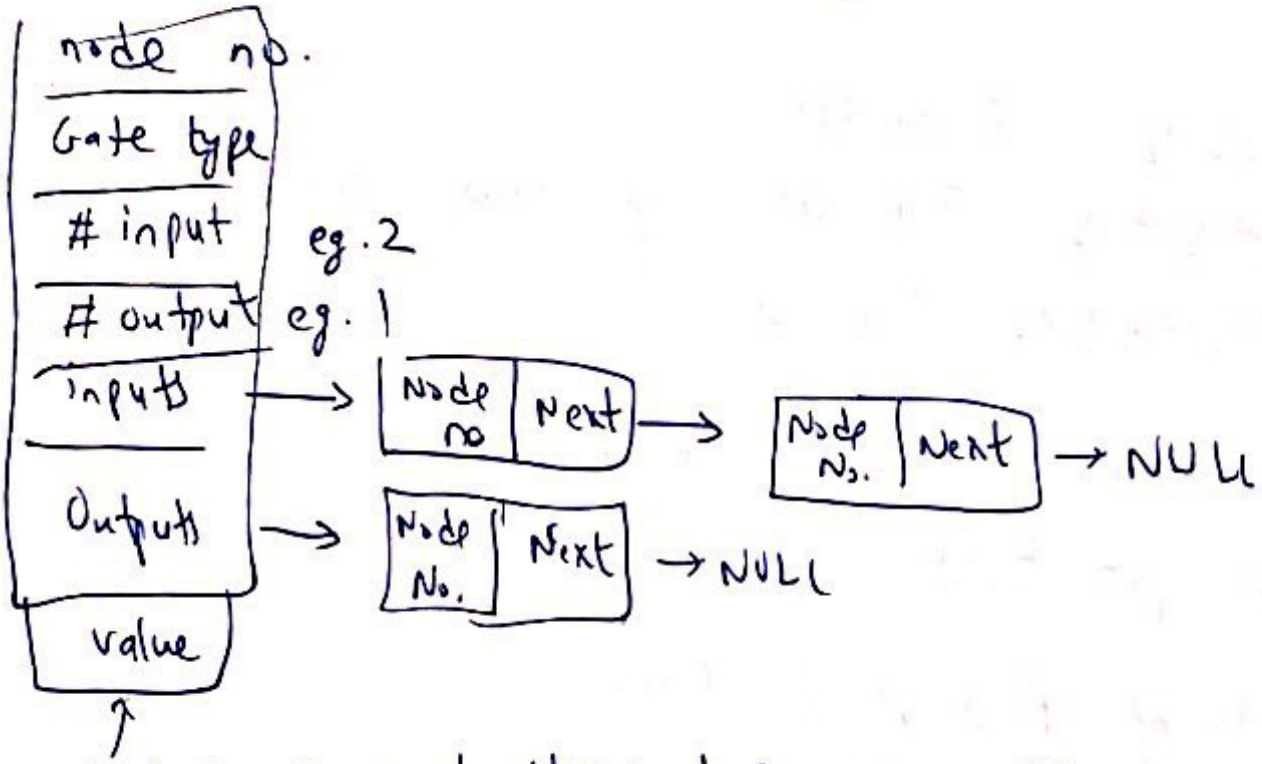
· We can store the netlist in the form of a graph

e.g.



Each gate, & input will serve as a node in a graph

Each node will have the following properties :



Calculate & and store during simulation

we need to proceed in a proper manner i.e. assign a level to each gate

Level (any primary input) = 0

Level (any gate) = max(level of inputs) + 1

Called compiled code simulation

. when testing multiple inputs, we can check if the current output changed.

If at a particular gate, both inputs to it did not change despite a change in the primary input, we need not evaluate it again & hence save time

↗ called as event-driven simulation

✡ however, we can pipeline the process by using multiple processors (one at each level)

∴ Simple compiled code simulation may be better in the practical world

. If d is delay of a MUX: total delay = n d

Since a max of n MUX can come in our way

Area $\propto$ n

.

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

eg. $a_1 b_1 + a_2 b_2$ — $a_1 a_2 b_1 b_2$

— $a_1 b_1 a_2 b_2$

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