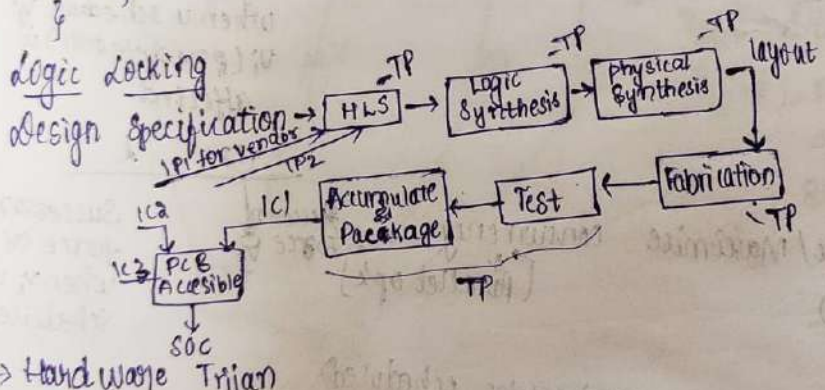

$TF(v_i, l) = SF(v_i, l) + PF(v_m, l) + PF(v_n, l) + SF(v_j, l) + SF(v_k, l)$

→ MR-LC (G, λ) obj. min. a.

{while (all operations are scheduled)
{ Compute the time frame (mobility) of
compute operations type probability
compute self-force, preced/successor force
Schedule the oper. $v_i$ with least schedule
force in its time frame
}
}




→ Hardware Trijan
- Malicious modification of the design.
- Reverse Engineering
- IP Piracy & over binding

Sol: Logic locking

Attack → identity keys
Threat model
what information can
be available for the
attacker
- Extract a gate level
netlist from the lay
Reverse engineering
- Attacker has info
the design (whit
- Buy functional
(by oracle) - Bla
can output of n

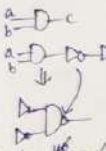
attack
- Functional att
- SAT atta
- ATPG atta
boun
- Re.s
- Structural a
- ML bound at

→ Random Logic Locking:

| a | b | a ⊕ b (XOR) | a ⊙ b (XNOR) |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

So attacker easily finds that if XOR key is 0, XNOR key is 1, so to make that Bubble Pushing

Bubble Pushing



→ output ~~productivity~~ (corruptability):

% of time un Key output will get wrong by the key insertion) need to get high corruptability so that attacker v key insert he get output wrong mostly.

→ here we do in random location

→ fault Analysis based Logic locking (FLL):-

Here we insert where the output gets wrong mostly/ the corruptability is high.

VLSI Testing

Stack-at-zero } physical
Stack-at-one } fault

→ ATP → Automatic Test Pattern Generation (tool)

→ Fault Impact

Strong Logic locking:

Make the keys interdependent

[dependent on other values of keys]

one key value is

SAT Attack

Satisfiability Problem:

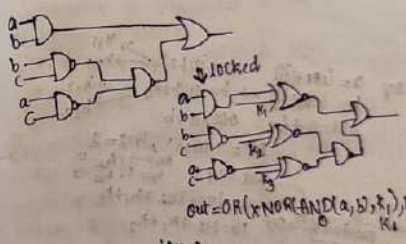
Given a formula $f$, is there a input for which $f$ becomes TRUE

Boolean formula: $f = a \wedge b$? Yes $a=1, b=1, f=1$

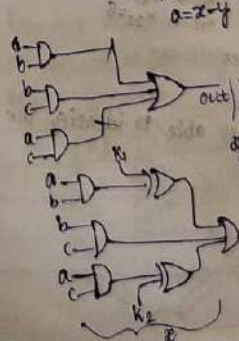
$f = (a \wedge b) \wedge (a' \wedge b')$ unsatisfiable

Int Domain $x+y>10$; $x,y$ int $x=5, y=200, f=$ TRUE

SMT solver — can't be solved by SAT solver but by SMT solver




if(){
a=x+y; a=(ite, c, x+y, x-y)
else
a=x-y
}
if else




output

| a b c | actual | locked | Possible Key |
|---|---|---|---|
| 0 1 1 | 1 | 1 | 00,01,10,11 |
| | 0 | 0 | Key=00 |
| | 1 | 1 | Key=11 |

because it identifies at least one key.

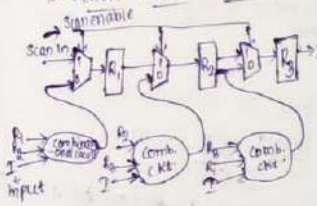
a should be 1 if c=1 → ATPG

b, c=1 if output is 1

Sensitise input for a particular design i.e. given output only input can be guessed.

→ Structural attack based on structure of the locking

* Scan chain Access:




R1R2R3

why Testing → some manipulation malfunction may happen [two wires connected / short circuit] before fabrication testing should be done.

Scanin=1 enable → then for registers, combinational circuits input will not be from

Scanin=0 enable → chain is broken / circuit is working as it is (original one) remain will

Input will be from combinational circuits for registers.

Scanenable=1 → helps to set a specific value to the registers.

→ N-reg in scan chain
- Run n-clk with SE=1
- one clk run the design with SE=0
- Run n clk with SE=1

→ C-level Locking: → - constant - condition - operation → Semantic information

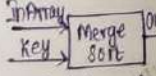
→ how can u lock a C-code?

Expression locking

a=b+c

a=k1 ? x+y : b+c

a=k1 ? b+c : b-c




- Arithmetic Exp
- conditional statement
- conditionally true (correct value, k1=0) ( " " k1=1)

if (c)
a=b+c;
else
a=b-c;

if (c ⊕ k0) → true k0=0
a=b+c or
else
a=b-c

if (c ⊕ k0) → true k0=1
a=b-c
else
a=b+c

constant locking

a=b+5 k3=6 ⇒ a=b+3⊕k3

5 = 0101
6 = 0110
k3 0011

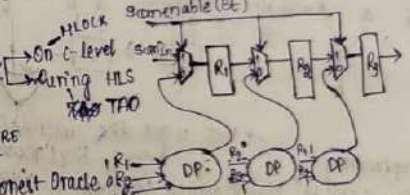
0⊕0 → k3
0011 → 3
0101 → 5

→ Logic Locking

Behavioural Level → On C-level (HLock) / during HLS TAO

RTL / Gate level → ASSURE HOST Dishonest Oracle

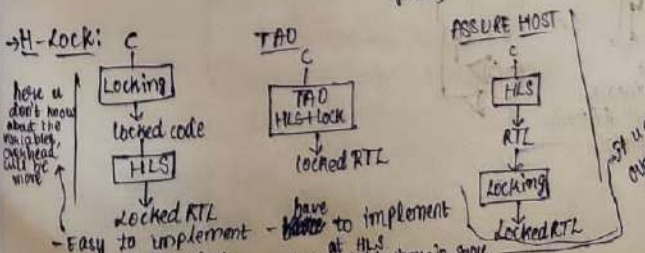
difficult to identify semantic info → so functional / structural locking

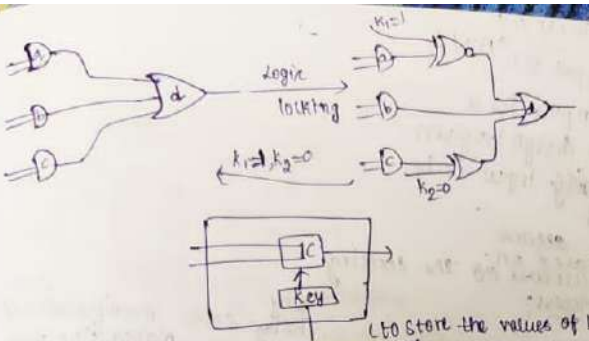



Normal mode SE=0 [reg take input from datapath]

[for n clk cycles keeping SE=1 & some value will be set and after that n clk cycles 1 clk cycle keep SE=0 then reg. take the value that I set from data path]

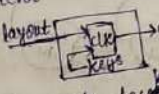
→ H-Lock:






Attack → to identify keys  Defence

- Threat model
  - what information can be available for the attacker
    - Different abstraction level
      - C-level
      - RTL
      - Gate level
    - Oracle guided attack – locked netlist + oracle.
    - Oracle less attack – locked netlist
- Extract a gate level netlist from the layout by reverse engineering (locked design)
- Attacker has information about the design (white box)
- Buy functional IP from market (by oracle) – Black box to check our output or not.




attack

- Functional attack
  - SAT attack
  - ATPG attack
  - Resynthesis bound attack.
- Structural attack
- ML bound attack.

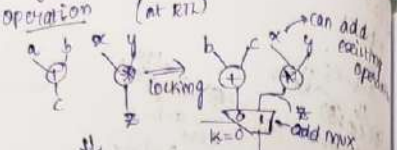
ASSURE

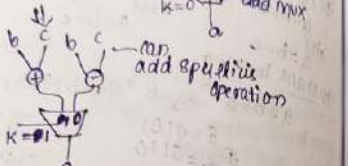
RTL → HDL Parser → AST (Abstract Syntax tree) → Locking → Locked RTL → RTL Extraction → AST

ASSURE
- constant
- operation
- condition

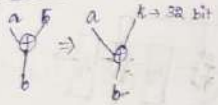
→ highly corruptable.

operation (at RTL)

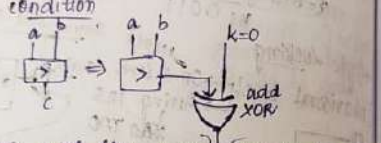



constant




condition

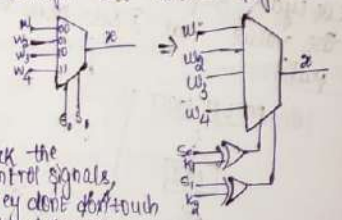



→ HOST
- control corruptability is high
- corruptability should be high in host low corruptability

+

high corruptability
- corrupt the control signals

→ then u get the wrong output if u give wrong key [mostly u get wrong if attack technique is strong, can identify the key easily]

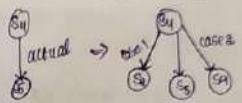


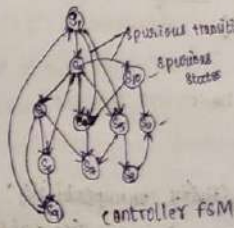
Lock the control signals, they don't touch the datapath

spurious transition in the controller FSM:

$S_1$: if ($[k_0,k_1]$==2'b00 & reg_r==10)
next state = $S_9$ // spurious transition (low corruptable locking)
else if ($[k_0,k_1]$==2'b01 & reg_r==5)
next_state = $S_9$ // spurious transition
else:
next_state=$S_5$; // original transition

corruptability will be less → always will not go to the wrong output




spurious transition

$S_2$: next_state $S_3$

$S_2$: if($k_0$=01 & reg5=12)
next=$S_{10}$ // new transition state

→ Combinational Logic Level:

→ Gate Level Logic locking
- combinational → U find many of defence → RLL, FLL, SLL kind of techniques
- sequential part
  - U have flipflops

combinational → Pre-SAT era → strong attack in this domain (one of the) → RLL, FLL, SLL kind of techniques

- SAT attack → most powerful [develop techniques that are SAT attack resistant after SAT attack]
- Post-SAT, Sarlock, Anti-SAT, Cycle SAT, SFLL, Probably Secure, Dishonest Oracle
- delay clock, clock locking (many techniques these are)

Sequential part
- Locking the FSM part
- lock the datapath regs (R → store the data; R → do the (reg. that implements finite state machine (FSM)) the FSM) → 2 types of reg.

Compatibility graph:

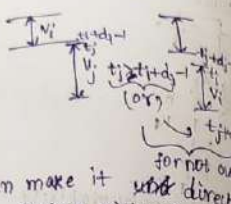
$G=(V,E)$, $V=[v_i \mid v_i$ is a node$]$ in sequence graph; $i=1,\ldots,n$

$E=(v_i,v_j)$ is scheduled as $v_i$ & $v_j$ are not overlapping

In $O(n^2)$ time complexity construct the comgraph for the schedule

$v_i$ schedules in $t_i \Rightarrow \langle t_i, t_i+d_i-1\rangle$

$v_j$ " " $t_j \Rightarrow \langle t_j, t_j+d_j-1\rangle$ interval



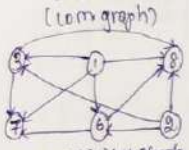
$v_i$ & $v_j$ are not overlapping

$t_j \geq t_i+d_i$

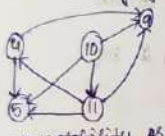
or $t_i \geq t_j+d_j$

this is undirected graph, u can make it directed by fixing either $(v_i \rightarrow v_j)$ or $(v_j \rightarrow v_i)$

(com graph)

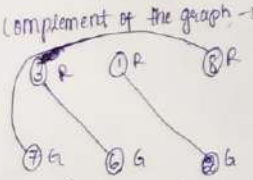


compatability graph for MUL



compatability graph for ALU

Complement of the graph - conflict



conflict graph

Finding Min. FU & Bind ≡ Graph coloring problem of conflict graph

Each colour ≡ FU

Nodes with same colour binds to the same FU

#colour = #FU

→ U need at least 8 colours to do this [conflict graph]

↳ chromatic number $\chi(G(V,E))$

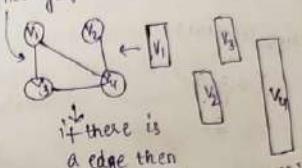
→ for generic behaviour (with loop for the fn call), the problem is NP-complete

- Behaviour with only basic block (non-hierarchical function) the FU alloc & bind problem in polynomial time solvable - left edge Algorithm

→ for solving fu allocation & binding - left edge algorithm.

Give a set of intervals, it will give u a set of intervals that are non-overlapping - Left edge Algorithm

conflict graph



interval graph

if there is a edge then they overlap

for this graph u can map the conflict graph to interval graph

→ all interval graphs are conflict graphs but all conflict graphs are not interval graphs.

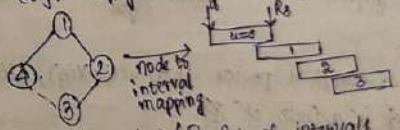
→ in some cases u cannot map conflict → interval graph.

→ Resource - Allocation & Binding

Operations → FU's

Variables → Reg's

conflict Graph ≡ Interval Graph



node to interval mapping

Left Edge Algorithm ($\mathcal{I}$) / $\mathcal{I}$ = set of intervals

{ sort $\mathcal{I}$ in ascending order based on $l_i$ (left edge)

c=0

while( some interval has not been chosen),

{ $S=\phi$, $r=0$,

while (∃ an element in $\mathcal{I}$ whose left edge coordinate is larger than r)

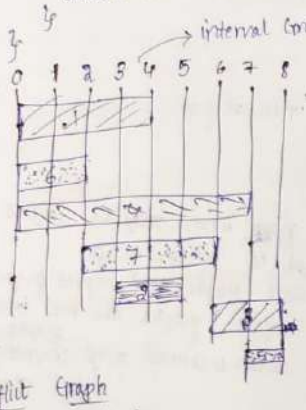
{ find first element in $\mathcal{I}$ with $l>r$;

$S=S\cup(r)$;

$r=r_s$

delete s from $\mathcal{I}$;

C++ O(nlogn)
label s with colour Gi




→ interval Graph

→ req. 4 colours
1, 3 ✓
6, 7, 5 ✓
4 ✓
2 ✓

Conflict Graph

$\alpha(G)$ = max. nodes having same colour (independent set)

$\chi(G')$ = # Colour needed to colour the graph G'

$\omega(G) \leq \chi(G)$

$\alpha(G) \leq k(G)$

Compatibility Graph

$\omega(G)$ → Cardinality of max clique

$k(G)$ → degree cover

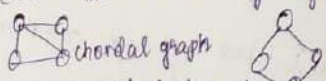
#min degree to cover G (Graph)

Perfect Graph → A graph is perfect then $\omega(G) = \chi(G)$, $\alpha(G) = k(G)$

Chordal Graph:- If there is a cycle of ...

A Graph G is chordal iff for any cycle having more than 3 edges, there is a chord.

(chord means a connecting edge between corners)



chordal graph

not a chordal graph.

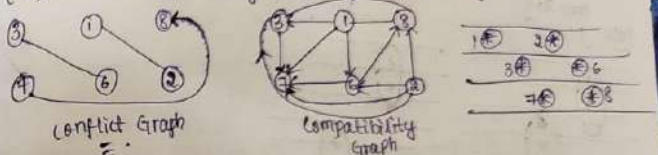
→ interval graph ⊆ Chordal graph ⊆ Perfect graph.

Comparability Graph: the edges satisfies the transitive orientation
$i \in (v_i, v_j) \in E, (v_j, v_k) \in E \Rightarrow (v_i, v_k)$.

Gilmore & Hoffman theorem:
An undirected graph is an interval graph iff it is chordal & its complement graph is comparability graph

Conflict graph is chordal

Compatibility graph is comparability Graph



Conflict Graph ≡ Chordal Graph ≡ Comparability Graph

[if there is no cycle also by default it is a chordal graph]

→ FU-Allocation & Binding - Hierarchial Model

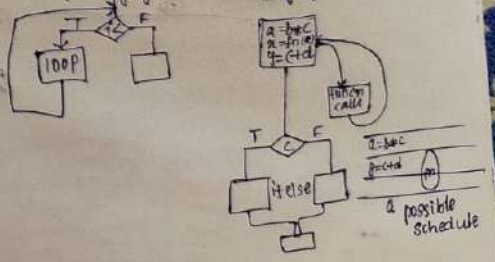
→ Conflict graph is interval graph then this problem is polynomially solvable.

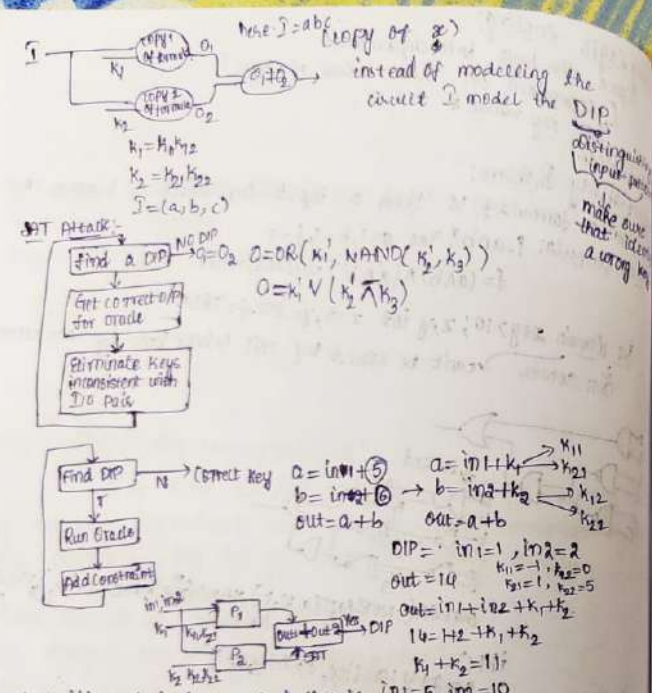
→ is chordal & is comparability graph (Heuristic orientation)

→ for non-hierarchial graph (No control flow / Single BB) → basic block

↳ Conflict graph is interval graph

Hierarchial graph
- Function call
- If - else
- Loop






Note: I = abc (copy of x)

instead of modelling the circuit I model the DIP (distinguishing input pair)

make sure that it does not eliminate a wrong key

$K_1 = k_{11} k_{12}$

$K_2 = k_{21} k_{22}$

$I = (a, b, c)$

SAT Attack:

Find a DIP → NO DIP → $O_1 = O_2$

$O = OR(k_1', NAND(k_2', k_3))$

$O = k_1' \vee (k_2' \wedge k_3)$

Find DIP → Correct Key

$a = in1 + k_{11}$, $a = in1 + k_{21}$

$b = in2 + k_{12}$, $b = in2 + k_{22}$

$out = a + b$

DIP = $in_1 = 1$, $in_2 = 2$

$k_{11} = 1$, $k_{12} = 0$

$k_{21} = 1$, $k_{22} = 5$

$out = 14$

$out = in1 + in2 + k_1 + k_2$

$14 = 12 + k_1 + k_2$

$k_1 + k_2 = 11$;

$in_1 = 5$, $in_2 = 10$

$k_{11} = 5$ $k_{21} = 3$

$k_{12} = 6$ $k_{22} = 8$

→ U have to build a logic locking such that it handles SAT attackable. SAT attack should remove few of wrong keys not all → no of iterations increases (if it is exponential) then it takes so much time

→ another one is SAT attack tool is not able to identify the DIP

→ $T = \sum_{i=1}^{\lambda} t_i$

$\lambda$ = #iteration = DIPn

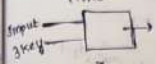
$t_i$ = time taken by SAT solver to get $DIP_i$

↳ increase atleast one of these

strong DIP → eliminate many wrong keys (in a time)

weak DIP → very few wrong keys in one iteration

if key space is high then eliminate 1 of 2 will take exponential time. will not effect much, it




| $x_1$ | $x_2$ | $x_3$ | |
|---|---|---|---|
| 0 | 0 | 0 | $K_0$ |
| 0 | 0 | 1 | $K_1$ |
| 1 | 1 | 1 | $K_7$ |

→ low output corruptability

Vulnerable to structural attack

Hamming Distance:

diff. of no of bits between 2 seq.

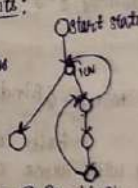
$\begin{matrix} 0 & 1 \\ 0 & 1 \end{matrix} = 2$ HD = 2

Sequential components:

→ Registers / FF

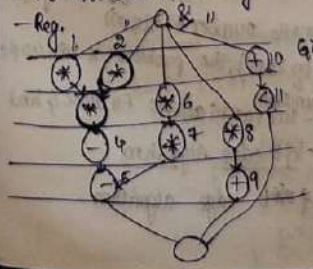
→ Datapath registers

→ FSM regs




Resource Allocation & Binding:

- FU Allocation & Binding
- Reg.



Given a schedule a graph which is a compatability graph

Compatability among the operations.

Ex:- 1 & 2 can't be scheduling using the same one MUL since they're running in parallel so they (1 & 2) are not compatible

## Function calls

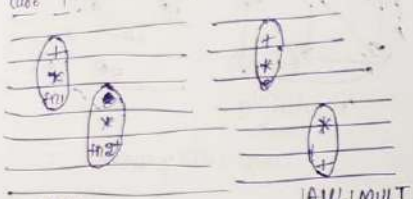
case(i) function may be called only once

case(ii) A function may be called more than once

case 1: Conflict graph is interval graph

```
z = fn1(a, b);
y = fn2(a, b, c);
out = z + y;
```

case 1:



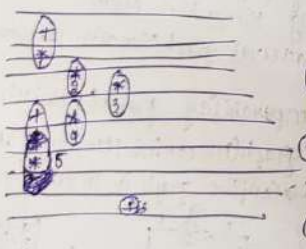

1 ALU
2 MULT → overlapping

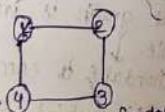


1 ALU, 1 MULT

case 2:

1. z = fn1(a,b)
2. r = p * q
3. e = d * e
4. y = fn1(r, 7)
5. out = z + y





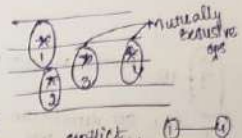
not a chordal graph

but the same function is called twice which is ⑦

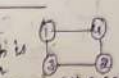
this is not a chordal graph (not having ① — ③ chord) so this is not a interval graph, so left edge algorithm is not applicable so heuristic algorithm should be applied.

if-else

```
2. a = b * c
3. d = a * c
4. if (1)
5.   y = p * c
   else
6.   y = p * y
```




conflict graph is not a interval graph



not a conflict graph

loop

conflict Graph is interval graph

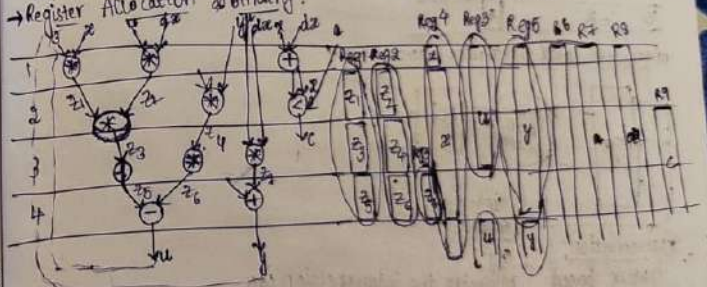
## Register Allocation & Binding:

variables ⇒ Register

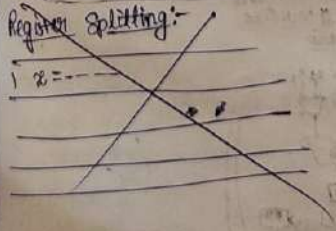
Obj. Allocate min regs for variable bind var to regs such that there is no conflict.

when lifetime of two variables overlap.

## Register Allocation & Binding:




## Register Splitting:



## Register Splitting:-

1. x = ...
4. y = z + ...
7. x = ...
9. ... = x
10.

x → R1

Register Spilling:

1. x = ...
2. M[1] = R1
10. = x

x → R1, R2 = M[1]

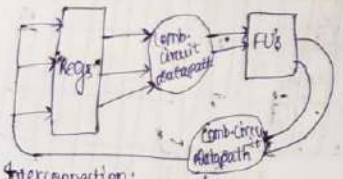
var — Reg

x1, x2 → R1 — Multiple variables mapped to one reg

x → R1, R2 — one variable can mapped to multiple regs

Register splitting

## Datapath and Controller Generation:



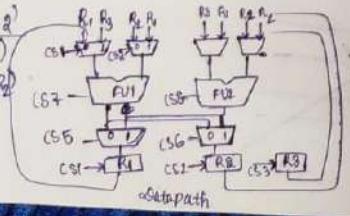

Interconnection:

i) MUX based
ii) BUS based

Minimize the interconnection cost
#MUX, #Switch, #MUXsize

MUX Based Datapath

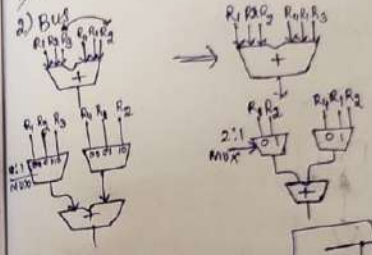
1. $R_1 = FU_1(R_1, R_2)$
   $R_2 = FU_2(R_3, R_2)$
2. $R_2 = FU_1(R_2, R_1)$
   $R_1 = FU_2(R_1, R_2)$




| | CS1 | CS2 | CS3 | CS4 | CS5 | CS6 | CS7 | CS8 | CS9 | CS10 | CS11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| T1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| T2 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |

## Datapath and Controller Generation:

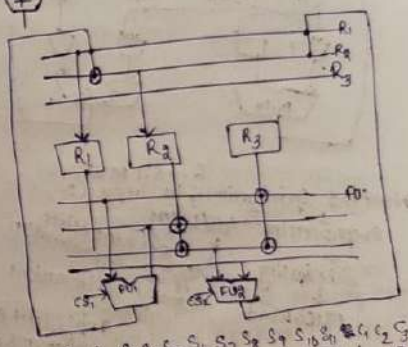
1) MUX based
2) Bus




Bus based Architecture

$R_1 = FU_1(R_1, R_2)$
$R_2 = FU_2(R_3, R_2)$

$R_2 = FU_1(R_3, R_1)$
$R_1 = FU_2(R_1, R_2)$

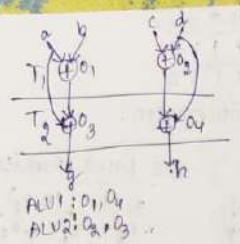



| | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $S_7$ | $S_8$ | $S_9$ | $S_{10}$ | $S_{11}$ | $S_{12}$ | $S_{13}$ | $S_{14}$ | $S_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Control Signals $T_1$ | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| Control Signals $T_2$ | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

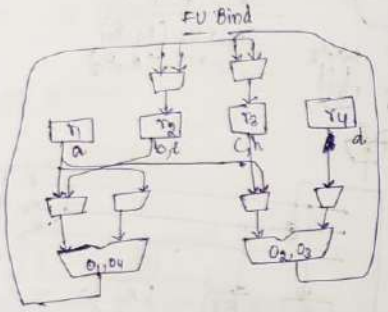
Optimization objective:

Find min. no. of buses to accomodate all data transfers or find max. no. of data transfer that can be accomodated in a fixed no. of buses.

→ e = a + b
T1 f = c + d
T2 g = a + e
T2 h = f + d
r1 = a
r2 = b, e, g
r3 = d




FU Bind




6 → 2:1 MUX

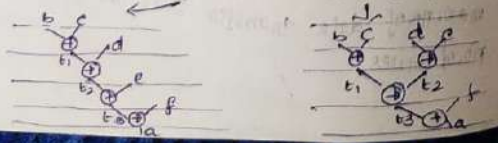
→ Front-end Optimization & its impact :-

Preprocessing → Clang/LLVM → IR → Intermediate Representation optimizations
↓ (Applies many)
Scheduling
↓
Alloc & Bind
↓
CP - DP Generation
↓
RTL

a = b + c + d + e + f
t1 t2 t3 → 3-address form
t3

t1 = b + c
t2 = t1 + d
t3 = t2 + e
a = t3 + f

t1 = b + c
t2 = d + e
t3 = t1 + t2
a = t3 + f

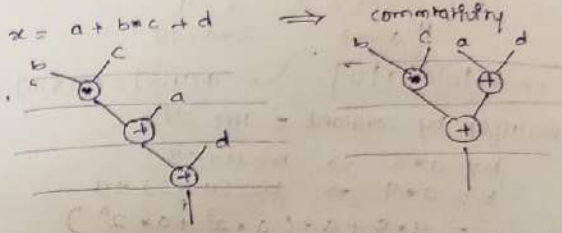






- parenthesis to force complex to schedule as per your intension.

x = a + b * c + d ⟹ commutativity




associativity & distributivity → rewrite expressions to reduce the tree height.

* Constant propagation

a = 0
b = a + 1 ⟹ a = 0, b = 1
c = 2 * b c = 2

1) Reduce latency, resource

* Copy propagation

a = x (both x and a are storing same value, use any one of them)
b = a + 1
c = 2 * a ⟹ b = x + 1, c = 2 * x (Reduce the reg)

* Common Sub-expression elimination
- identify the CSE and remove them
- Reduce latency and resources.

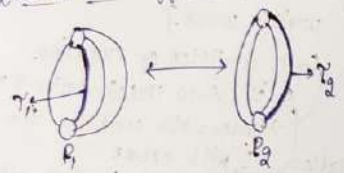
a = x + y
c = y + x ⟹ a = x + y, c = a, b = a + c
b = a + c

copy propagation ⟹ a = x + y, b = a << 1

two are not equal ⇒ they are equal.
if I check $C_{\gamma_1} = C_{\gamma_2}$ it returns SAT ⇒ there is one input for which they are equivalent but doesn't say for all inputs they are equivalent so i have taken the negation of the formula

→ Can I make the complexity less from $O(n^2 \times c)$ to $O(n \times c)$. (same no. of traces)

Data driven Approach:




Take a random input.
check the trace for $P_1$ & for the same input check the trace for $P_2$
Now we check for these traces $\gamma_1, \gamma_2$ so the complexity reduces to $O(n)$.

Coverage driven Testing (concolic Testing)

→ Identify the corresponding Traces between two programs
→ then do coverage driven testing.

→ Revised algorithm

4. Merge compatible traces in $T_1$
5. Merge compatible traces in $T_2$
6. Apply data driven approach to identify corresponding traces.
7. For each corresponding trace pair. $\langle \gamma_1, \gamma_2 \rangle (\gamma_1 \in T_1, \gamma_2 \in T_2)$
   check $\gamma_1 \equiv \gamma_2$

Binary Decision Diagram (BDD)

$f_1 = x'y'z + yz + xz$
$f_2 = yz$
→ $f_1 \equiv f_2$ ?

Construct truth table for both the functions and check the values but if there are more variables we cannot check using truth table.

→ internally u can give to SAT solver and it checks if they are equivalent or not.

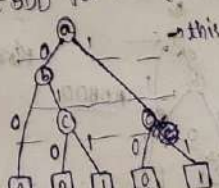
BDD / OBDD → ROBDD
- OBDD → ordered
- ROBDD → Reduced order (it is canonical Representation)

if u construct ROBDD for both the functions then u get both ROBDD's are same.

→ Any two boolean formula, their ROBDD is the same if they are equivalent.
→ I can use BDD to reduce the circuit size also. (ex. for $f_1$ we need 4 AND, & OR whether for $f_2$ we need 1 AND so in that way we can use BDD we can reduce the circuit size).

Rooted Tree → BDD is a rooted tree where each level represents one variable of the formula and the leaf nodes are 0/1.

→ To check (a+b)(a+c) → construct BDD for this (BDD is nothing but like a truth table.)
considering root node as a
each node has two child corresponding to its value 0/1.



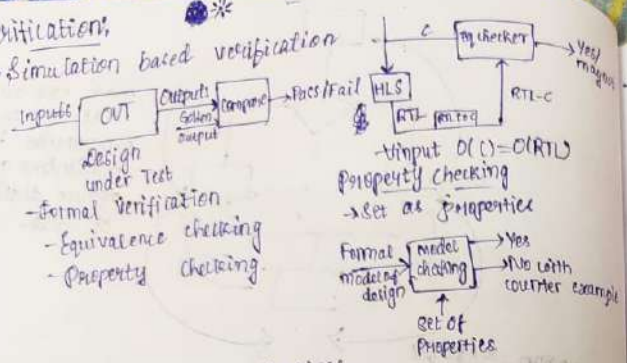

→ this layer is a
" b
" c.
so each layer corresponds to a variable & its order (it is order → OBDD)

OBDD (a,b,c)

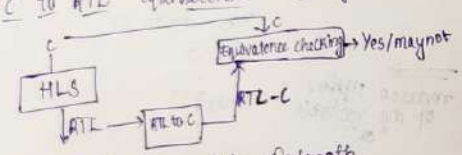
OBDD → where order of the variables are given.

## Verification:

→ Simulation based verification




→ C to RTL equivalence checking:-

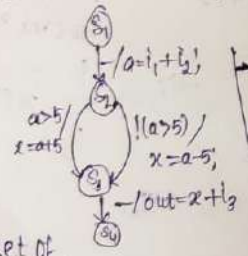



Finite State Machine with Datapath

fn(i1, i2, i3)
{ a = i1 + i2
if (a > 5)
x = a + 5;
else
x = a - 5;
out = x + i3;
return out;
}

every state should have condition & the corresponding operation otherwise we keep blank (for no condition)

implementation (operations)




even they have dependency if we keep in same state then they run in parallel.
So first level can be put in 1-transition, 2nd level can be in next transition

FSMD have set of states, set of operations, set of inputs, output & certain conditions will be there for this condition this transition will hold & for this transition this operation happens

(If there is a for loop or loop there will be backedge.)

→ operations that are in one state are parallel.

for ex. a = b+c;
x = a+y; ⇒ a = b+c
x = b+y

1. Identify all the traces → Trace: one execution path of the program.
(given a program u can identify the traces.)

[→ for acyclic we assume there is no data dependency we unroll & find the trace.
→ for data dependency this won't work.]

2. ∀ γ ∈ T identify the $C_\gamma$ & $S_\gamma$ ; $C_\gamma$ is condition of execution.
$S_\gamma$ is data transformation of γ.
γ → trace
under which condition the γ will execute
If γ executes what will be the final value of outputs.

Substitution Method/Symbolic Simulation

$[C_{\gamma_1}, S_{\gamma_1}] = [T, <a, x, out>]$

$[C_{\gamma_2}, S_{\gamma_2}] = [T, <i_1+i_2, x, out>]$

$[C_{\gamma_3}, S_{\gamma_3}] = [i_1+i_2 > 5, <i_1+i_2, i_1+i_2+5, out>]$

$[C_{\gamma_4}, S_{\gamma_4}] S_{u_4} = [\underbrace{i_1+i_2 \le 5}_{\text{condition of execution}}, <i_1+i_2, i_1+i_2+5, \underbrace{i_1+i_2+i_3+5}_{\text{data transformation of } \gamma}>]$

because im bothered without.

3. Perform step 1 & step 2 for $P_1$ & $P_2$ $(T_1, T_2)$.

4. For each $\gamma_1 \in T_1$, identify $\gamma_2 \in T_2$ s.t. $\gamma_1 \equiv \gamma_2$ → complexity → $O(n^2 \times c)$
(when both traces are same)

→ C to RTL Equivalence Checking

$\gamma_1 \equiv \gamma_2$ iff
$C_{\gamma_1} \equiv C_{\gamma_2}$ &
$S_{\gamma_1} \equiv S_{\gamma_2}$

i will give a formula for the eqn solver whether it is SAT or UNSAT. based on that we can check the equivalence.

Two expressions:

check $C_{\gamma_1} != C_{\gamma_2}$

check SAT → if returns SAT ⇒ no equivalent.
if UNSAT ⇒ there doesn't exist any input for which these

## Operator strength Reduction

9 | 0 | 1 | 0 | 0 | 1 | a  a = 9

a << 1  2 * a = 18

18 | 1 | 0 | 0 | 1 | 1 | 0 → a[4:0] = (a[3:0], 0)

① Multiply by constant ≡ left shift

b = a * 8 ⇒ b = a << 3

b = a * 9 ⇒ b = (a << 3) + a

= a * 8 + a = ( a * $2^3$ + a * $2^0$ )

② Division by constant

a = b/2 ⇒ a = b >> 1

```
c = 7
for (int P=0; P<N; P++)
    y[P] += c * P;
```

⟹ Induction variable & imp invariant

```
c=7, K=0;
for (int P=0; P<N; P++)
{
    y[P] += K;
    K = K + c;
}
```

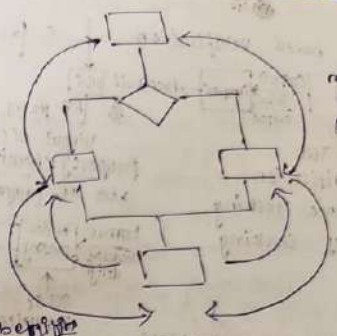
③ Code motion

- Move instruction within a program without changing the functionality

```
a = b + c;
d = d - e;
if (c)
    x = x + y;
else
    x = x - d;
```

⟹

```
if (c) {
    a = b + c;
    x = x + a;
}
else {
    d = d - e;
    x = x - d;
}
```



more code such that redundant computation is minimized and reduce duplication of code.

code motion before

(a)

```
a = ---
 :
if()
 - -
else if()
 - -
 - -
b = a + c
```

⟹ reduce lifetime of the variable "a"

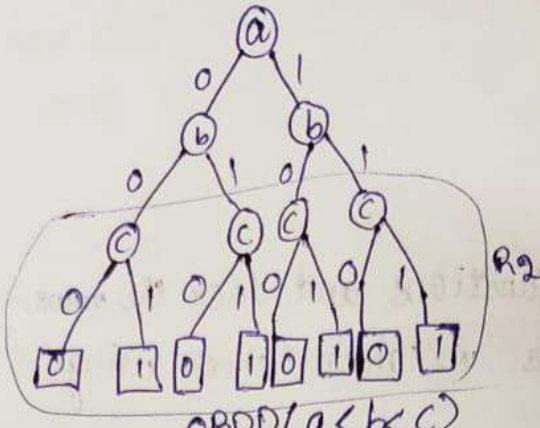
```
a = ---
b = a + c
if()
 - -
else if()
 - -
```

(b) loop invariant code motion

```
for (int P=0; P<n; P++)
{
 - - -
 t = a + b;
 x[P] = t + y;
}
```

⟹

```
t = a + b;
for (int P=0; P<n; P++)
{
 - - -
 x[P] = t + y;
}
```




OBDD(a<b<c)

root leaf

OBDD → ROBDD

R1) Leaf node has only one 0 node
1 node

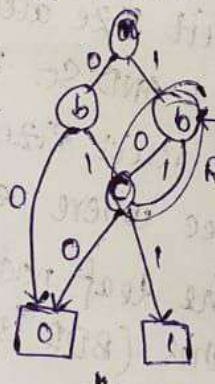
R2) if for a node V, if left(V) = right(V); remove V.

~~3) If the source~~

~~3) If the~~

R3) For two nodes $V_1, V_2$ if left($V_1$) = left($V_2$) & right($V_1$) = right
remove one $V_1$ & $V_2$ (i merge $V_1$ & $V_2$ into a single node).

Apply the above rules bottom up.

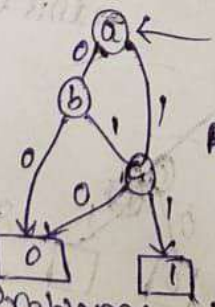



we want to get this

~~This~~ problem is ~~Polynomial~~ as polynomial time solvable → O(n log n)

till this node it is ROBDD.

but Satisfiability problem is NP complete




ROBDD

~~Klaas~~

Homework

(b, c, a) } take these variable orders & construct ROBDD
(c, a, b)

→ Satisfiability Problem:- It will give the inputs for which the ~~Po~~ output is 1.