Combinational Logic optimization

- power optimization can be achieved at system, architectural (behavioral), register-transfer, logic level, transistor, and physical design, circuit level

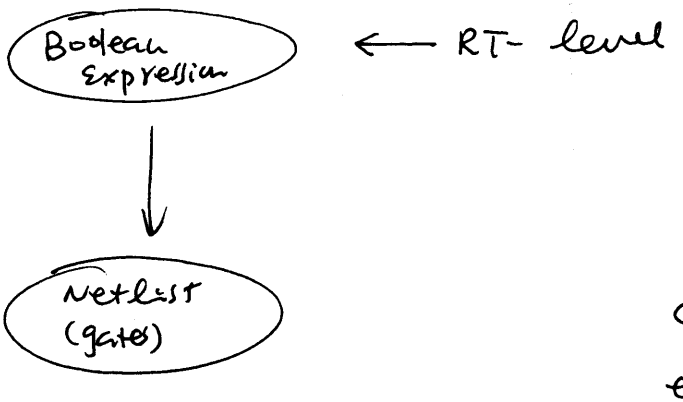
- Logic synthesis design level.

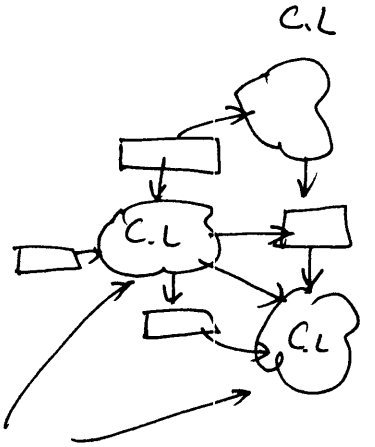
  Automatic synthesis of gate-level netlists, minimizing some objective function subject to various constraints

  E.g., goal: minimal area

  Subject to: delay requirement.

- Techniques used for logic synthesis





change all Boolean equations to gates.

* Extraction
* Re-substitution
* Collapsing

→ Try to optimize several functions

* Decomposition
* Factorization

→ Try to optimize a single function

Extraction: Look for expressions observed many times. Extract them and implemented only once. Then, they are shared.

E.g.,

$$F = (a+b) \cdot c \cdot d + e$$

$$G = (a+b) \cdot \bar{e}$$

literal count = 16

$$H = c \cdot d \cdot e$$
$$I = (a+b) \cdot c \cdot d \cdot e$$

we identify $a+b$, $c \cdot d$ as common terms.

$$X = a+b$$

$$Y = c \cdot d$$

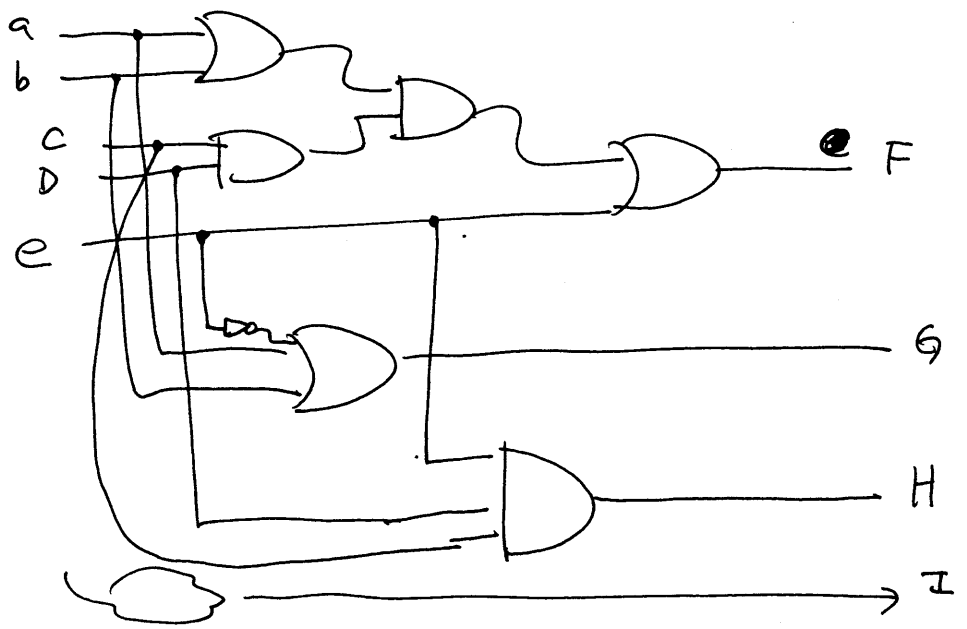
$$F = X \cdot Y + e$$

literal count = 14

$$G = X \cdot \bar{e}$$

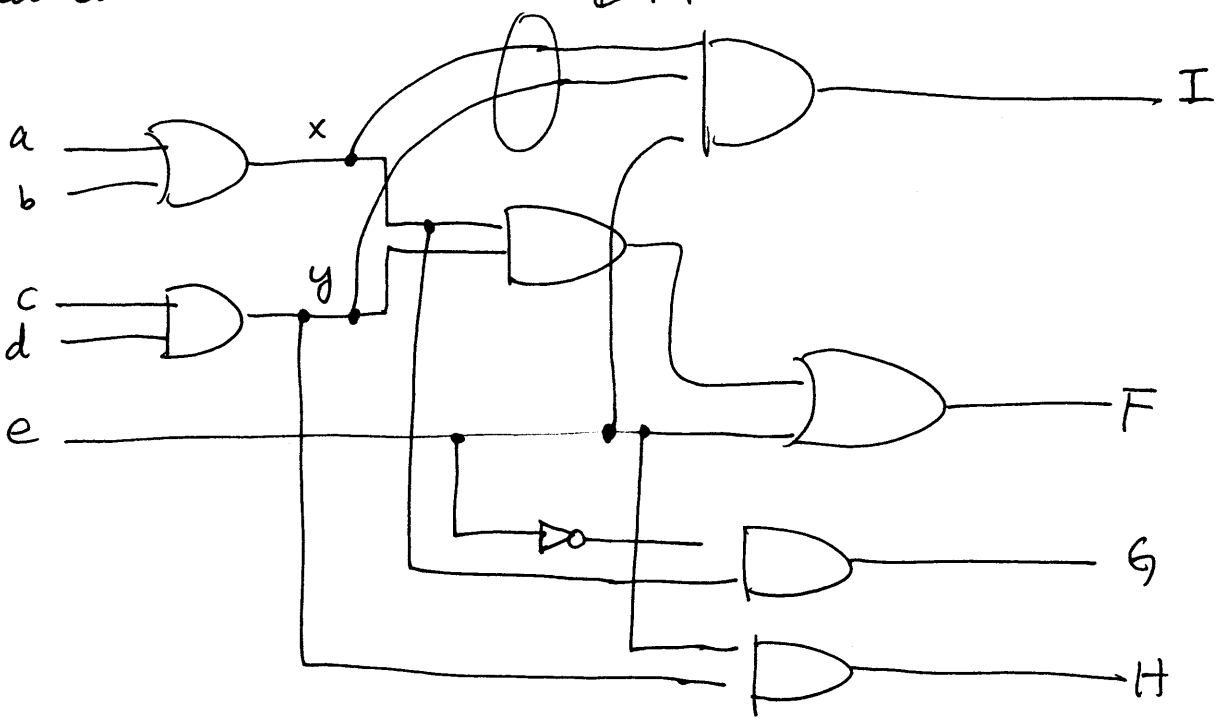
$$H = Y \cdot e$$
$$I = X \cdot Y \cdot e$$

orginal ckt



New ckt:

use 2 lines to replace 4 lines.



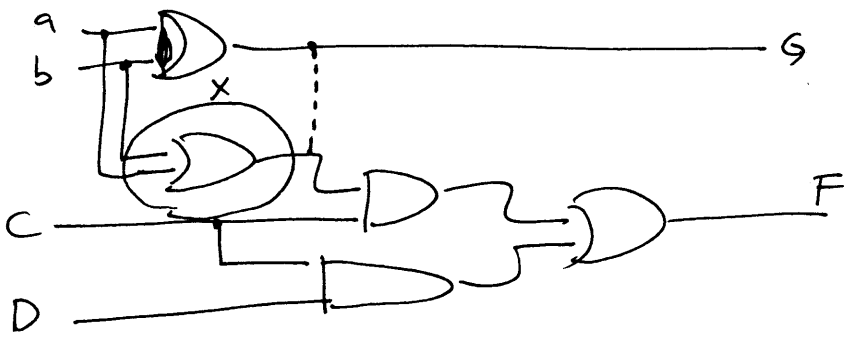
* use literal count to estimate the ckt area.

Re-substitution ~~Decomposition~~ : use the function of a node already in the network to implement the function of another node in the ckt.

$$G = a+b$$

$$F = (a+b)\cdot c + cd$$

$$\Rightarrow F = G\cdot c + cd$$



Factorization: Identify logic sharing within the same function.
Derive a factored form from a sum-of-product form.

E.g. $F = a\cdot c + a\cdot d + b\cdot c + b\cdot d + e$ Literal count = 9

$\Rightarrow F = c(a+b) + d(a+b) + e$

$= (a+b)(c+d) + e$ literal count = 5

Decomposition Reexpress a single function as a collection of new functions.

Two goals:
① smaller-sized expressions are more likely to be substituted by other functions
② Reduce the size of expressions to that of library cells

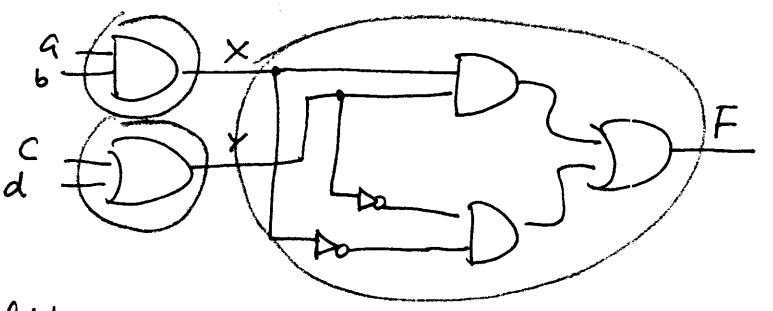
E.g. if too many "+", then still need to decompose!!! p.278 of De Micheli!

$F = a\cdot b\cdot c + a\cdot b\cdot d + \bar{a}\,\bar{c}\,\bar{d} + \bar{b}\,\bar{c}\,\bar{d}$

$X = a\cdot b$

$Y = c+d$

$F = X\cdot Y + \bar{X}\,\bar{Y}$



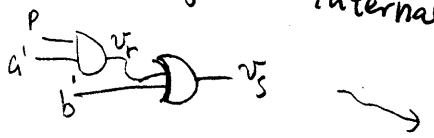
★ Split a complex function into several gates.
(increase speed)

Collapsing. the collapsing of an internal vertex is its removal from the network.
• inverse of substitution

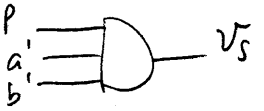
Example:

$v_r = p + a'$

$v_s = v_r + b'$

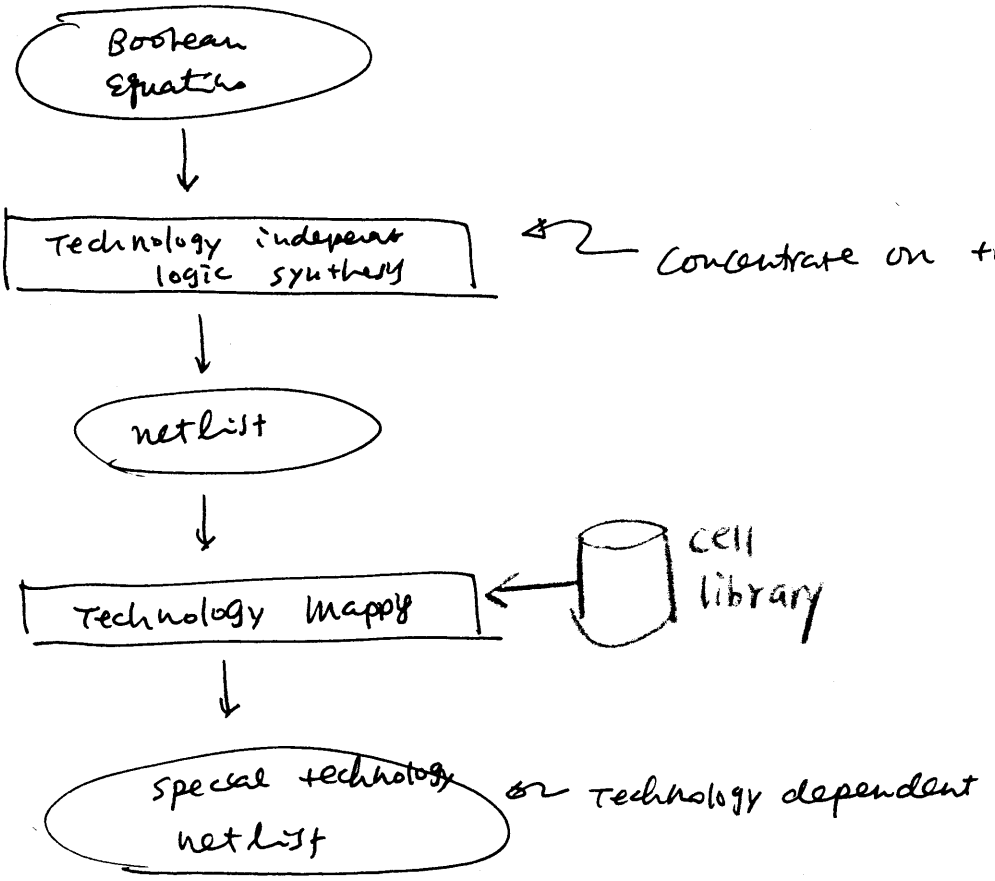


A collaping of $v_r \Rightarrow v_s = p + a' + b'$



Motivation: Try to remove a simple local function
e.g. v_r does not have fanout to other nodes or
v_r is not a PI.

- Technology independent power optimization



* Includes • algebraic logic restructuring techniques.
  • Boolean optimization techniques.

* power optimization by common sub-expression extraction.

Given a boolean network, find a set of nodes to introduce in the network such that the power cost of the network in the Sop form is minimized.

Example:

$F_1 = a(xy) + a(uw) + vz$

$F_2 = b(xy) + b(uw) + vz$ with the following switching probabilities:

$p(a) = 0.97$	$p(u) = 0.91$	$p(x) = 0.67$
$p(b) = 0.02$	$p(v) = 0.93$	$p(y) = 0.47$
$p(c) = 0.51$	$p(w) = 0.35$	$p(z) = 0.65$

How to rearrange the boolean equations s.t. the power consumption can be minimized.

Def: Two functions are orthogonal if they do not have common literals.

Eg, $f = a+b$, $g = c+d$

f and g are orthogonal. ($g \perp f$)

Def. Function g is an algebraic divisor of f if $\exists\, h, r$ s.t. $f = g \cdot h + r$ where $h \neq \phi$ and $g \perp h$.

Example: $f = abc + abd + de$

$g = a \cdot b + e$

$\frac{abc}{ab} = c$. $\frac{abd}{ab} = d$ $\frac{de}{ab} = \phi$

$\therefore h_1 = \{c, d\}$

$\frac{f}{g} = \underset{h}{\underline{d}} + \underset{r}{\underline{\underline{abc}}}$

$\frac{abc}{e} = \phi$ $\frac{abd}{e} = \phi$. $\frac{de}{e} = d$

$\therefore h_2 = \{d\}$

$\therefore h = d$. $h_1 \cap h_2$

Def. The primary divisors of f is $P(f) = \{ f/c \mid c \text{ is a cube} \}$.

↑ a conjunction of literals

$f = abc + abde$

$f/a = bc + bde$ is a primary divisor.

Def: The kernels of f are $K(f) = \{ K / K \in P(f),$ the only cube dividing K evenly is $1 \}$.

Example:

$f = a \cdot b \cdot c + a \cdot b \cdot d \cdot e$

$f/a = bc + bde$ is a primary divisor, but not a kernel.

$f/ab = (c + de)$ is a (kernel) and ab is called a (co-kernel).

Example:

$G = af + bf + ace + bce$

Kernels: $ce+f$, $ce+f$, $ae+be$, $a+b$, $a+b$

Co-kernels: a, b, ~~e~~, f, $c \cdot e$

$G/a \Rightarrow ce+f$; a

$G/b \Rightarrow ce+f$; b

$G/c \Rightarrow ae+be$; ~~e~~

$G/f \Rightarrow a+b$; f

$G/ce \Rightarrow a+b$; $c \cdot e$

∴ f can be factored to $(c+de)(ab)$ to reduce the literal count !!!

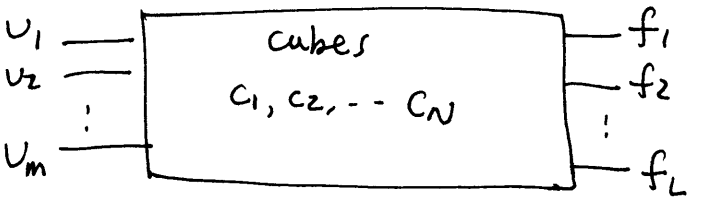
7 → 5

$H = ade + cde$

(Kernel) $H/de = a + c$

(Co-kernel) de

Kernel Extraction targeting Low power



$D = d_1 + d_2 + \cdots + d_p \Leftarrow$ a kernels of $f_1 - f_L$ used as a divisor.

$Q = \{q_1, q_2, \cdots q_R\} \Leftarrow$ Co-kernels

Example: # of literals = 16

$F_1 = axy + auw + vz$

$F_2 = bcxy + bcuw + vz$

$D = (xy) + (uw) \leftarrow$ kernel (d_1, d_2)

$q_1 = a$

$q_2 = bc$

$\Rightarrow$

$D = xy + uw$

$F_1 = aD + vz$

$F_2 = bcD + vz$

# of literals = 13

$R = 2, \; p = 2$

The area saved by extracting D is

$$(R-1)\sum_{i=1}^{P} lit(d_i) + (P-1)\sum_{i=1}^{R} lit(q_i) - R$$

- $(R-1)\sum_{i=1}^{P} lit(d_i)$: Literals saved by not repeating the kernel
- $(P-1)\sum_{i=1}^{R} lit(q_i)$: Literals saved by not repeating the co-kernels
- R: # of Literals introduced by extracting kernel D

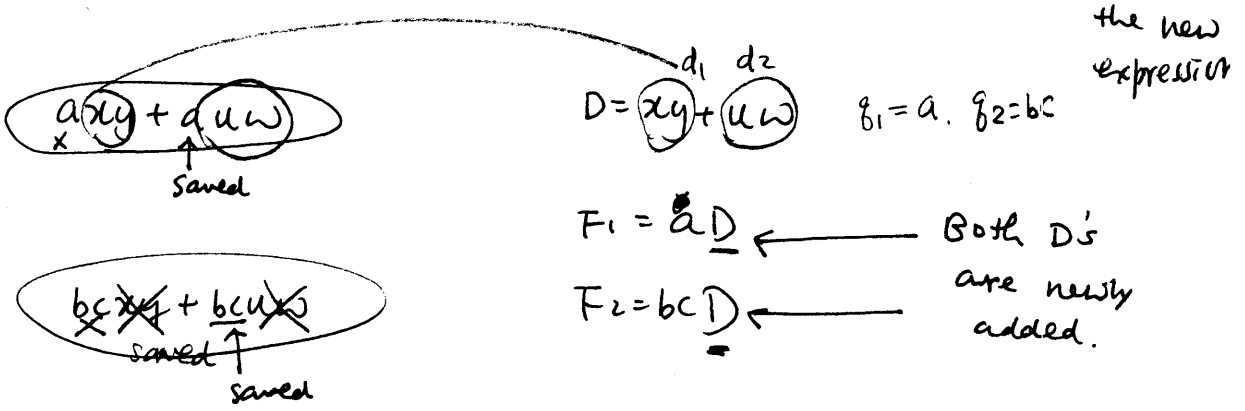
Example:

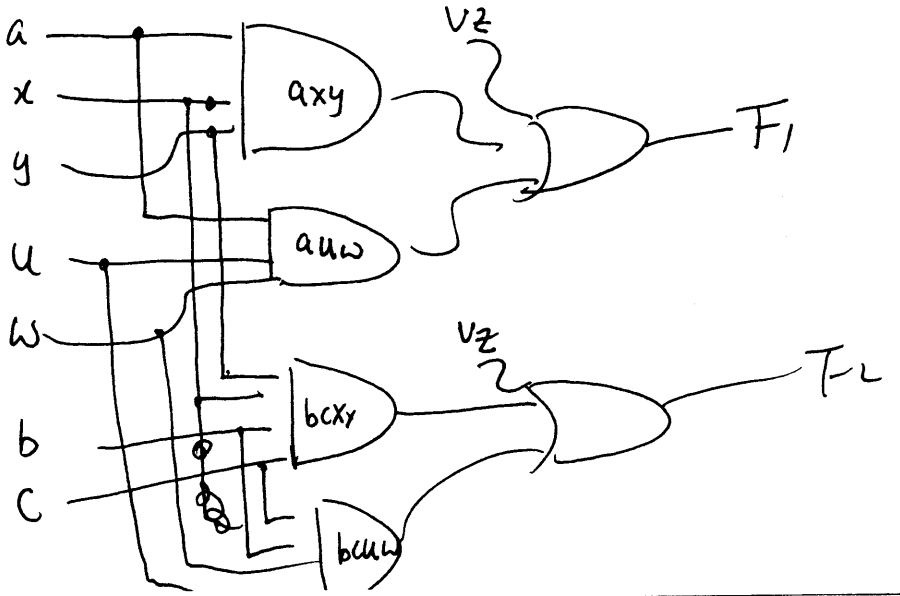
$$(2-1)\left(lit(d_1)+lit(d_2)\right) + (2-1)\left(lit(q_1)+lit(q_2)\right) - 2$$

$lit(d_1)=2$, $lit(d_2)=2$, $lit(q_1)=1$, $lit(q_2)=2$

$= 4+3-2 = 5$

There are 2 D's in the new expression





The power saved by extract of D is: Assume all capacitances are constant = unit. ∴, don't care it.

Equation (3) in page 274 of textbook.

$$R-1 \sum_{i=1}^{M} E(V_i)\, LS(V_i, D)$$

$E(V_i)$: switching activity at node V_i.

$LS(V_i, D)$: # of times V_i appear in D

★ $(2-1) * \underbrace{(t(x) + t(y) + t(u) + t(w))}_{\text{switching activity} \times \text{Capacitance} = \text{effective capacitance}} +$

now, ignore capacitance as 1.

x, y, u, w originally feeds 2 gates for each signal, now only one to kernel ($\underline{xy + uw}$)

∴ power saved by the reduction in the load on the inputs of the kernel.

★ $(2-1) * (t(a) + t(b) + t(c)) +$

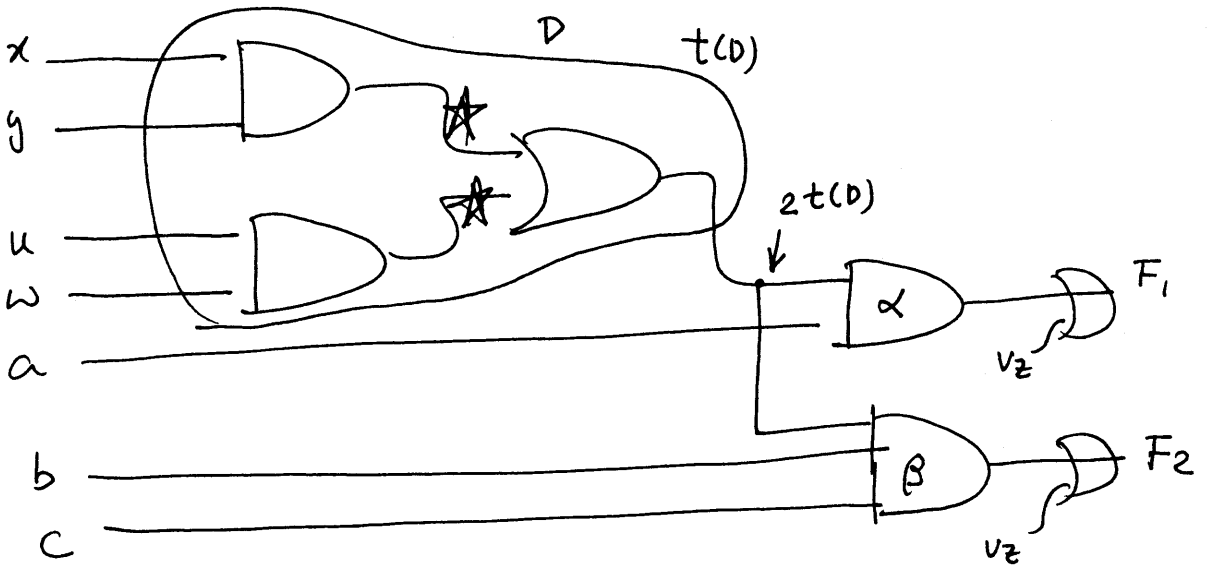
Each of signals a, b, c feeds 2 gates originally. Now, only one to the co-kernal (a, bc)

is power saved by the reduction in the load on the inputs to the co-kernels of the given kernel.

★ $t(axy) + t(auw) + t(bcxy) + t(bcuw) -$

power saved by removing cubes (gates) from the original function.

$2 * t(D) -$



power consumption added to output of the new node (D) inserted.

- $(t(xy) + t(uw)) -$

power consumption at the output of the cubes of the new node (D) inserted.

$(t(aD) + t(bcD))$

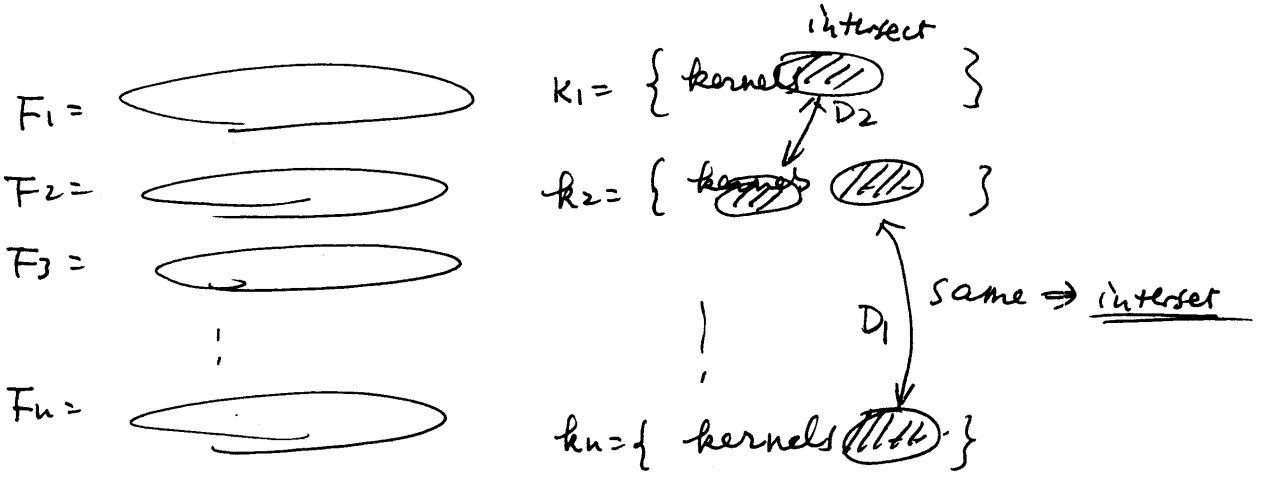
power consumption at the output of the new cubes α, β.

For the above example, with $P(a)$ -- $P(z)$, we can calculate all t values.

Finally, the power value = 0.647

∴ the kernel extraction for D can same power.

Summary



① Compute K : the set of all kernels for all functions

② Generate D : the set of all kernel intersections for kernels in K.

③ Extract the sub-expression $D_i \in D$ which has the maximum power value.

④ Repeat ③ where there exists a sub-expression D_i with a positive power value.

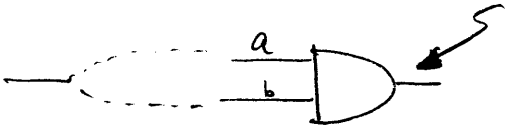
- power optimization by substitution

- Substitution does not always guarantee a reduction in the power cost of the ckt.

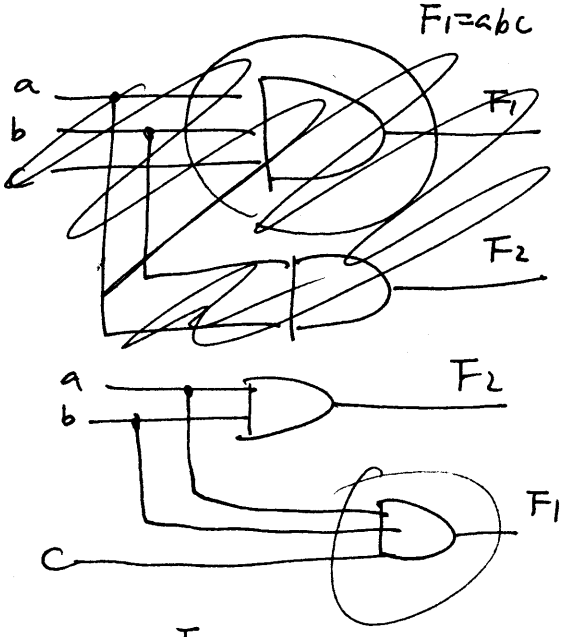
- Example: $F_1 = abc$ $\quad F_2 = ab$

$P(a) = P(b) = P(c) = 0.9, \quad P(F_2) = 0.5$

$P(F_2) \neq P(a) \times P(b)$ due to spatial dependence between a and b.



Implementation one:



F_2 to

$\downarrow$

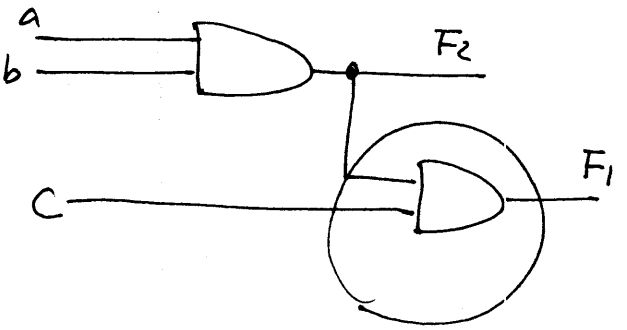
$(0.5)(0.1) \times 2 + 0.9 \times 0.1 \times 2$ (5)

loss in new connects $= 0.5 + 0.18$

$= 0.68$

Implementation 2:

$F_1 = F_2 \cdot c$



For F_1: only input changes.

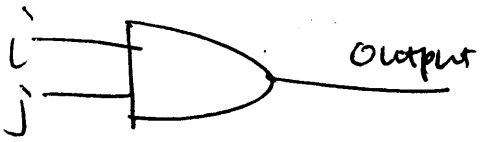
Input of F_1 has been changed from

$(0.9)(0.1) \times 2 + (0.9 \times 0.1) \times 2 +$

same in a, b. ↑ a ↑ b

$(0.9 \times 0.1) \times 2 = 0.18 \times 3 = 0.54$

E.g.



Only 00 & 11 can be applied to i & j.

$\therefore P(\text{output}=1) = P(i=1, j=1)$

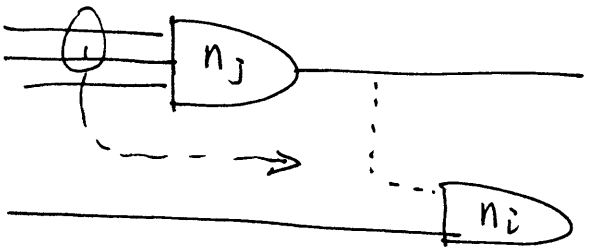
$= P(i=1)$

$= 0.9$

instead of

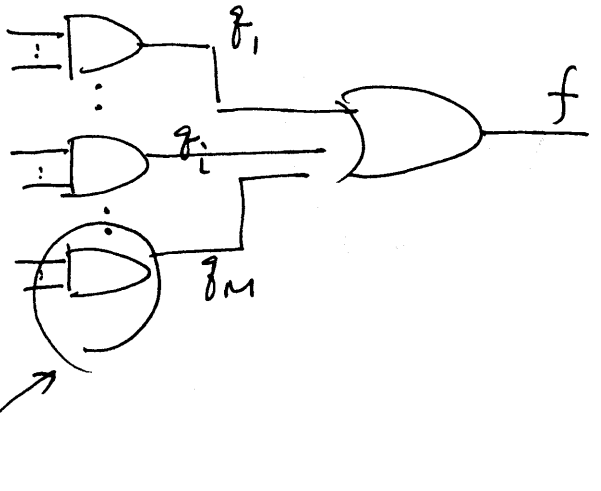
$0.9 \times 0.9 = 0.81$

- For literal count, we use one line ($F_2 \to F_1$) to replace 2 lines ($a \to F_1$, $b \to F_1$).
- For power cost, the gain in $a \to F_1$ (or $b \to F_1$) (save) is less than the loss in $F_1 \to F_2$ (or 5).
- Conclusion



* A node n_j can be substituted into a node n_i ?
* The power cost of n_i is computed before and after substitution n_j.
* Substitution is performed if the power cost of n_i is reduced after substitution.

- Boolean optimization techniques
  - Take advantage of don't cares.
  - Show by two-level circuits



power consumption for each product term:

$$P(q_i) = \frac{V_{dd}^2 f}{2}\left(C_{AND} \cdot E(q_i) + \sum_{l_j \in lit(q_i)} C_{IN} E(l_j) \right)$$

switching activity

load seen by the inputs of AND

load seen by the output of the AND gate

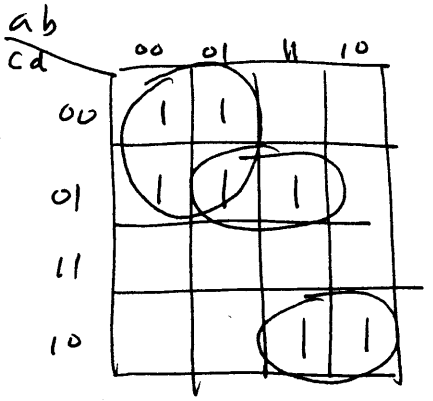
The power consumption of f:

$$P = \frac{V_{dd}^2 \cdot f}{2}\left(C_{OR} \cdot E(f) + \sum_{q_i \in Q} P(q_i) \right)$$

load seen by the output of the OR gate.

Prob. Given a boolean function f with input set $V = (v_1, v_2, \ldots v_N)$ and signal probability $p(i)$ for each input, find a two-level implementation of the function f s.t. the power as given by the above equation is minimum.

- For minimizing area, only prime implicants need to be included.



(Any non-prime implicant g can be improved by making g prime).

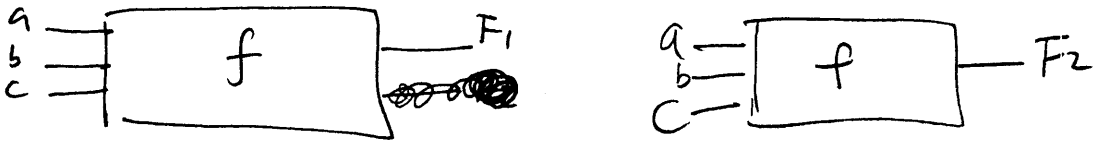
For minimum area: we use $\bar{a}\bar{c} + b\bar{c}d + ac\bar{d}$

For power. we may use: $\bar{a}\bar{c} + ab\bar{c}d + ac\bar{d}$

- Not the case for minimizing power consumption

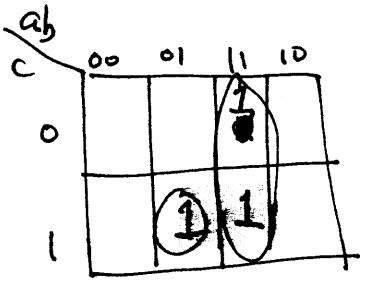
Example:

$p(a) = 0.9, \quad p(b) = p(c) = 0.5$



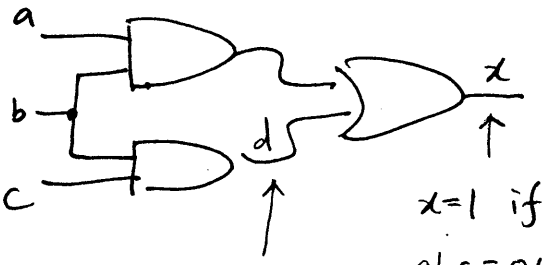
$F_1 = ab + bc$ ⟶ for area, we use this

$F_2 = ab + \bar{a}bc$ ⟶



F_1 implementation: $ab + bc$

F_2 implementation: $ab + \bar{a}bc$



$P_1 = 0.25$

$P_0 = 0.75$

$2P_1 \cdot P_0 = 0.375$

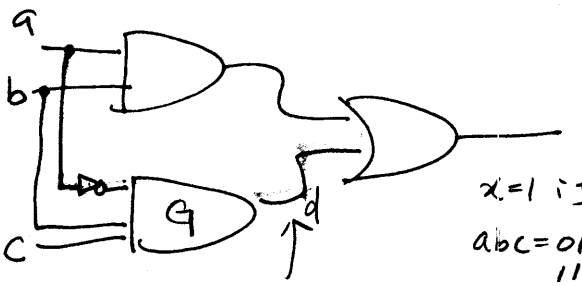
$x=1$ if

abc = 011,
110
111

$P_1 = 0.9 \cdot 0.5 \cdot 0.5 + 0.9 \cdot 0.5 \cdot 0.5 + 0.1 \cdot 0.5 \cdot 0.5$

$= (0.25)(1.9)$

$= 0.475$



$x=1$ if

abc = 011
110
111

$P_0 = \bar{a}bc$

$= 0.1 \cdot 0.5 \cdot 0.5$

$= 0.025$

$2P_1P_0 = 2 \cdot 0.025 \cdot 0.975$

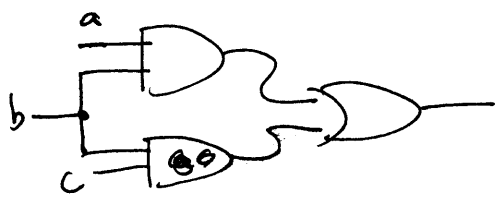
$= 0.04875$

This implementation

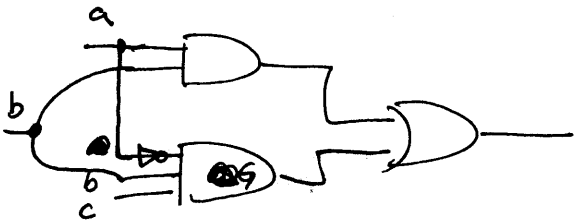
* Should choose

* If output load of G is very large, than F_2 implementation saves power.
(e.g., G fanouts to several other gates).

- d in this side saves 0.32625 of switching prob.
- But, a and $\bar{a}$ both added:
$0.1 \cdot 0.9 \cdot 2 \cdot 2 = 0.36$
- So, power increase about 0.03375



F1



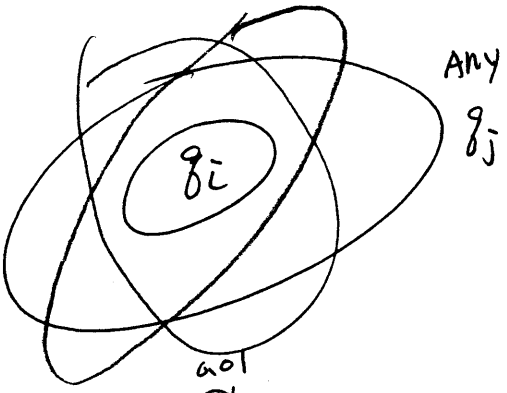
F2

The reduction in power at output of node ~~is more than~~ can't offsets the increase in the power due to including literal a

∴ ~~Reduc~~ increase power dissipation.

- Def. An implicant g_i is a <u>power prime implicant</u> (PPI) if

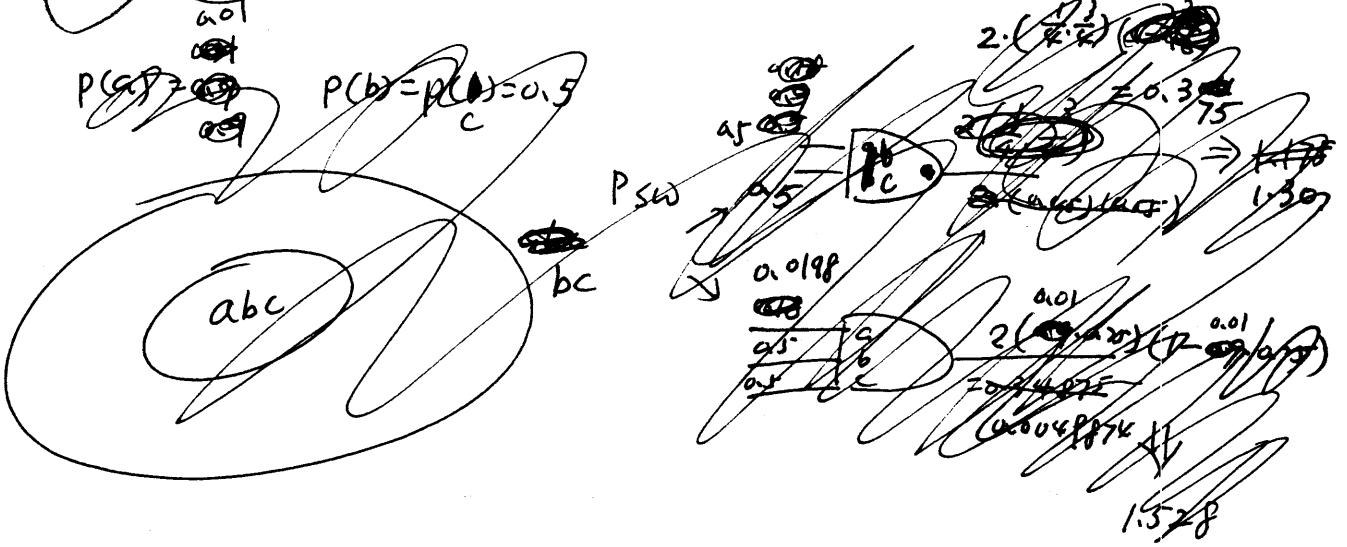
$$\forall g_j \quad g_i \subset g_j \iff pwr(g_i) < pwr(g_j).$$



★ This definition is very restrictive !!!

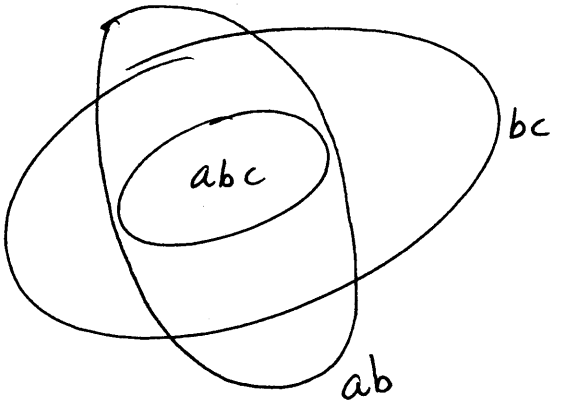
$2 \cdot \frac{3}{16} \cdot \frac{13}{16} =$ ~~0.42~~ 0.30

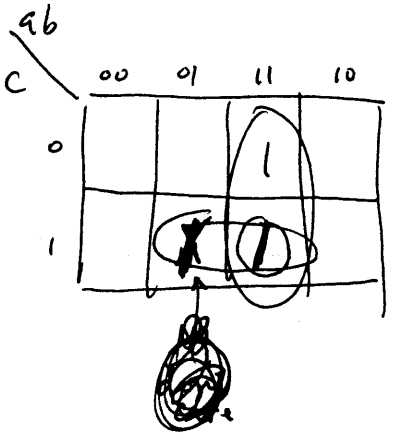
Example:



Example: use a very exasperating case

$p(a) = 0.01 \qquad p(b) = p(c) = 0.5$





$[2 \cdot 0.5 \cdot 0.5]\ [2(0.5)(0.5)] + 2\left[\frac{1}{2}\right]\left[\frac{3}{4}\right] = 0.5 + 0.375 + 0.5$

$= 1.375$

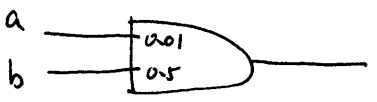
$2(0.01)(0.99) + 2(0.5)(0.5) + 2(0.5)(0.5) + 2\,[0.01 \times 0.25]\,[1 - 0.01 \times 0.25]$

$= 0.0198 + 1 + 2(0.0025)(0.9975)$

$= 1.0198 + 0.005 = 1.025$

If $P_{wr}(abc) < \min[P_{wr}(abc\text{'s predecessor PPIs})]$

$\Rightarrow$ abc is a PPI.



$2(0.01)(0.99) + 2(0.5)(0.5) + 2\,[0.005]\,[1 - 0.005]$

$= 0.0198 + 0.5 + 0.01$

$= 0.529$

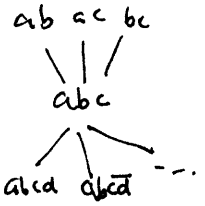
$\therefore$ abc is not a PDW-prime implicant (PPI).

Definition: predecessor cubes of an implicant g_i with n literals

Successor cubes of an implicant g_i ...

Example:

Cube (abc) for a function with inputs $\{a, b, c, d, e\}$ has predecessor cubes $(ab)(ac)(bc)$ and successor cubes $(abcd)(abc\bar{d})(abce)$ and $(abc\bar{e})$ for exple.



Basic idea: ① Generate all prime implicants (they are PPI's obviously)

② Develop power prime implicants from all PIs

③ Find ~~minimum~~ PPI's to cover the function with minimum power consumption.

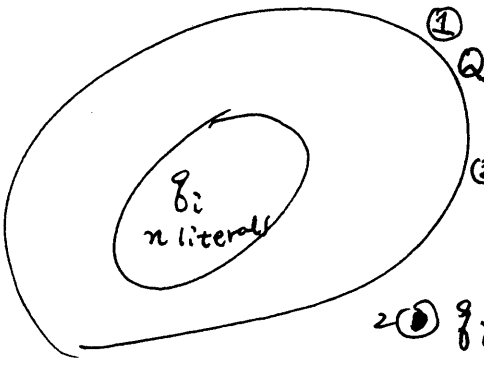
Lemma: If (ab) is a PPI and successors abc, abd, abe are not PPI's, then none of the cubes $abcd$, $abce$, $abde$ and $abcde$ is a PPI.

(Example)

(Assume we have a, b, c, d, e literals)

Theorem:

~~Example~~:



① Q: all PPI's with $(n-1)$ literals

② ~~If g_j $\forall g_j \in Q$ and g_j has the smallest power dissipation in Q~~

② g_i has at least one predecessor cube in Q

③ g_i is a PPI $\Leftrightarrow \forall g_j \in Q,\ g_i \subset g_j,\ Pwr(g_i) < Pwr(g_j)$

Key point: Just need to check the immediate predecessor's power.

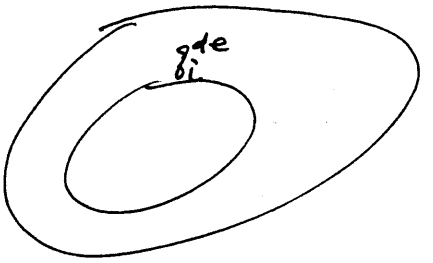
Note: we can use ← of the above theorem to check for g_i.

Def: Given g_i, an implicant of function f, $g_i^{l_1 l_2 \cdots l_k}$ represents the implicant generated by lowering literals $l_1 l_2 \cdots l_k$ in g_i.

Example:

$g_i = abc$

$g_i^{de} = abcde$

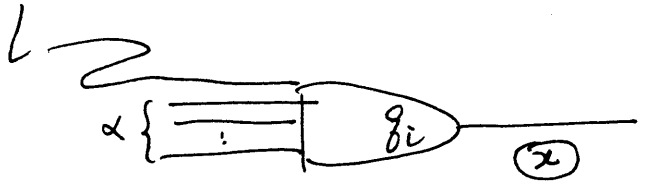


Lemma: Given implicant g_i with $p(g_i) = x$ and literal L:

$$pwr(g_i^L) < pwr(g_i) \iff p_L < \frac{x - x^2}{1 + x^2}$$

$p(L=1)$

How to get this important property?



$$C_{AND} \cdot E(g_i + L) + \left[\sum_{\alpha} C_{IN} E(l_j)\right] + C_{IN} E(L) \quad < \quad C_{AND} \cdot E(g_i) + \sum_{\alpha} C_{IN}(E($$

switching activity — $pwr(g_i^L)$ — $pwr(g_i)$

$$\Rightarrow \quad E(g_i + l) + E(l) < E(g_i)$$

$2p_i(1-p_i)$ · $2x(1-x)$

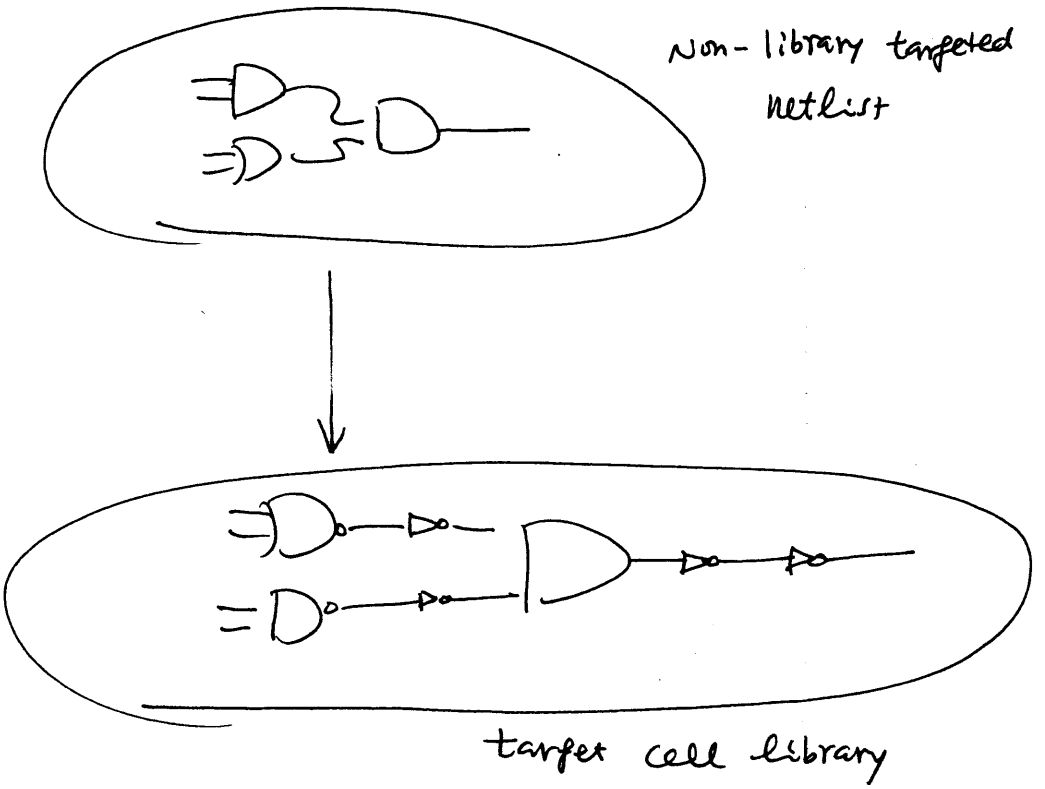
$$2\left[\frac{p_1 p_2 \cdots p_m}{L} \cdot p_L\right]\left[1 - \frac{p_1 p_2 \cdots p_m}{L} \cdot p_L\right] = 2p_L x (1 - p_L x)$$

```
function Generate_PPI(F)
begin
    P = generateAllPrimes(F);
    PP = initializePP(P); place prime implicants with n literals in PPn.
    for (i=1; i<N; i++) do
        Q = findSuccessorCubes(PPi); ← all implicants in Q has i+1 literals
        for each (q ∈ Q)  do
                 i+1 literals
            qj = findMinPowerPredecessor(q, PPi); ← return the predecessor cube
            i-literal                  i+1 literals  i-literals   of q which has
            l = literalLoweredInQ(q, qj)                          the smallest power cost.
                                  i+1 literals  i-literal
            if (P(l) < (P(qj) - P²(qj)) / (1 + P²(qj))) then
                PPi+1 = PPi+1 ∪ q;
            endif
        endfor
    endfor
    return PP
end
```

- Once the set of all PPIs of the function are generated, a minimum covering problem will be used to select a set of PPIs which cover the function and have minimal power cost.

* power optimization for PLA : no time to cover.

- Low power Technology Mapping



Two phases:

1: The logic network is decomposed into basic gates (Technology decomposition)

2. The basic-gate network is covered using the library gates by a tree covering techniques. (cell binding)

- The role of technology mapping:

  finish the synthesis of ckt by performing the final gate selection from a particular cell library.

- It is not the role of technology mapping to

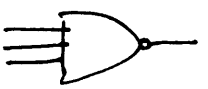
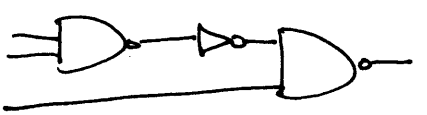
  ① Change the ckt structure radically

  e.g. should not find common subexpressions between two or more parts of the ckt

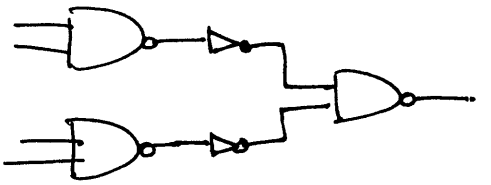
  ② Reduce the number of levels of logic along the critical path.

  e.g., TM can just choose fastest gates along the critical path, and use the most area-efficient combination of gates off the critical path.

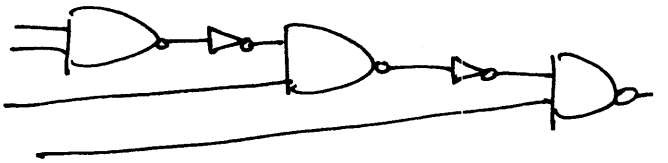
- Example cell library used for T.M.

Gate	Cost	symbol	primitive	DAG
Inverter	2			Represent everything by 2-input NAND gates.
NAND2	3			
NAND3	4			

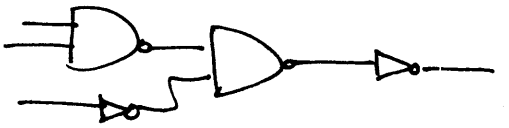
NAND 4 5



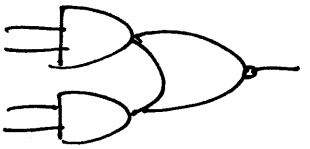
or

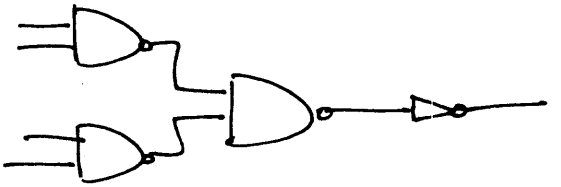


AOI21 4

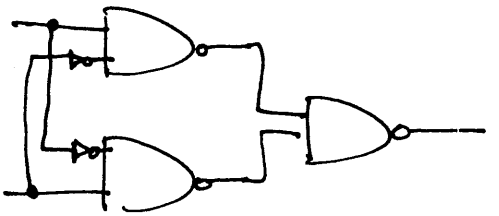


AOI22 5





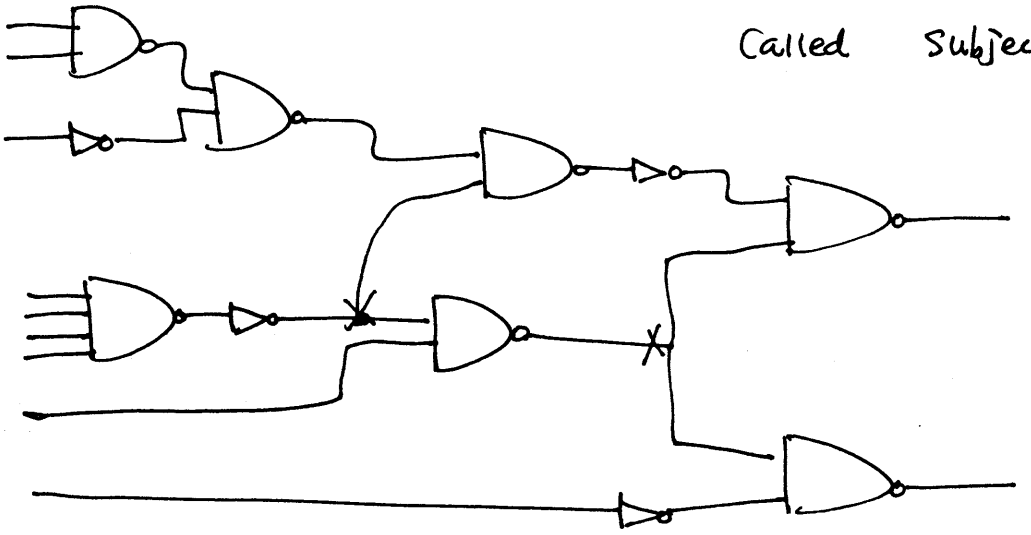
XOR 4



- T.M

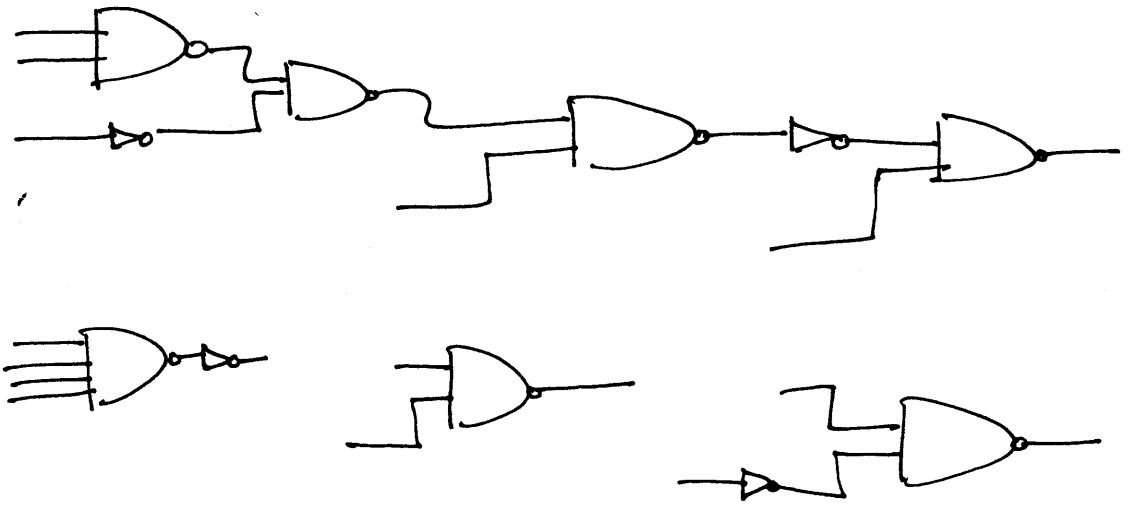
Step 1: Convert general ckt synthesized to NAND gates + inverters

Exaple:

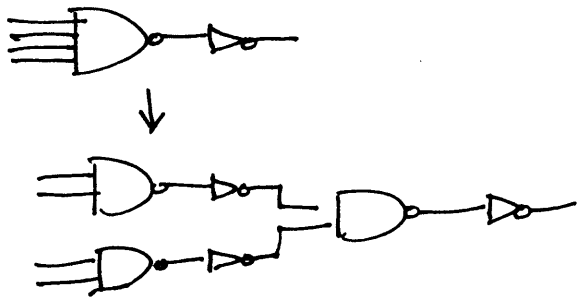


Called Subject DAG
(directed acyclic graph)

Step II: ~~Technology decomposition~~
Subject Graph partitioning (Remove all fanouts)



Step III. Technology Decomposition (Decompose everything to inverters or 2-input NAND gates)

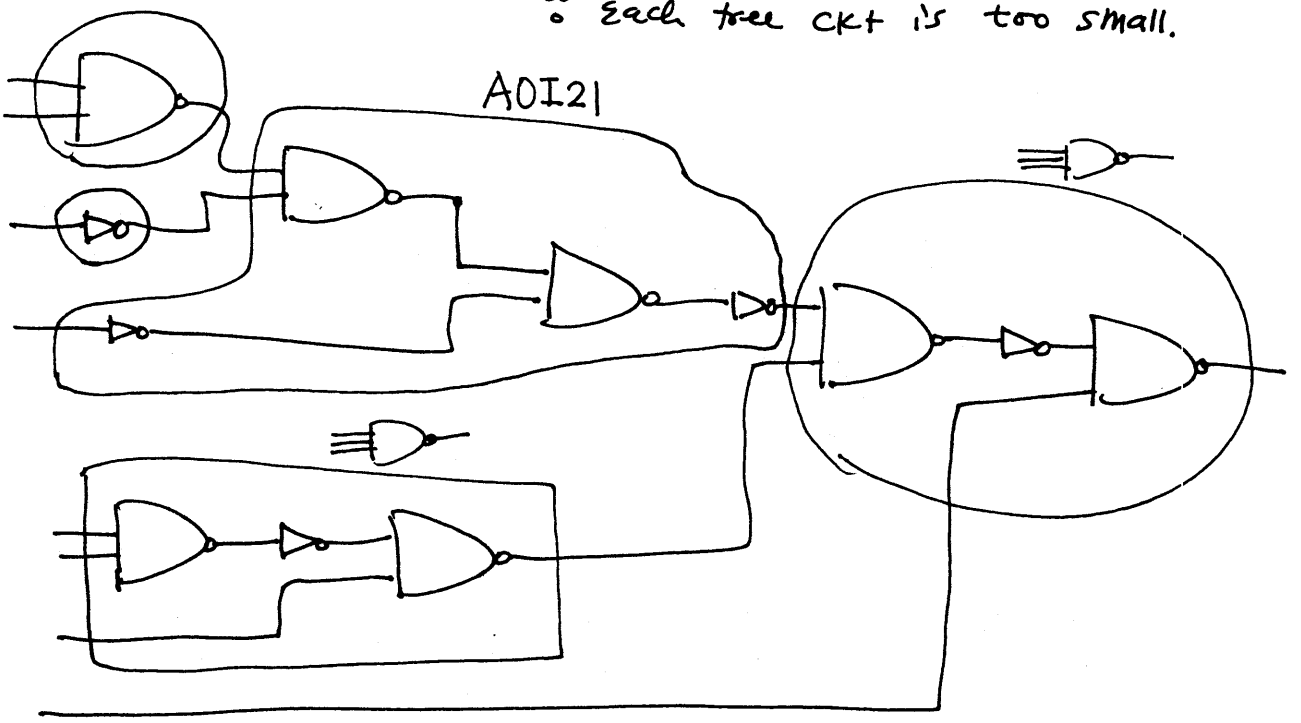


all others are not changed

Step 4. Tree matching using primitive DAG

(Note: show by another ckt example)

∴ Each tree ckt is too small.



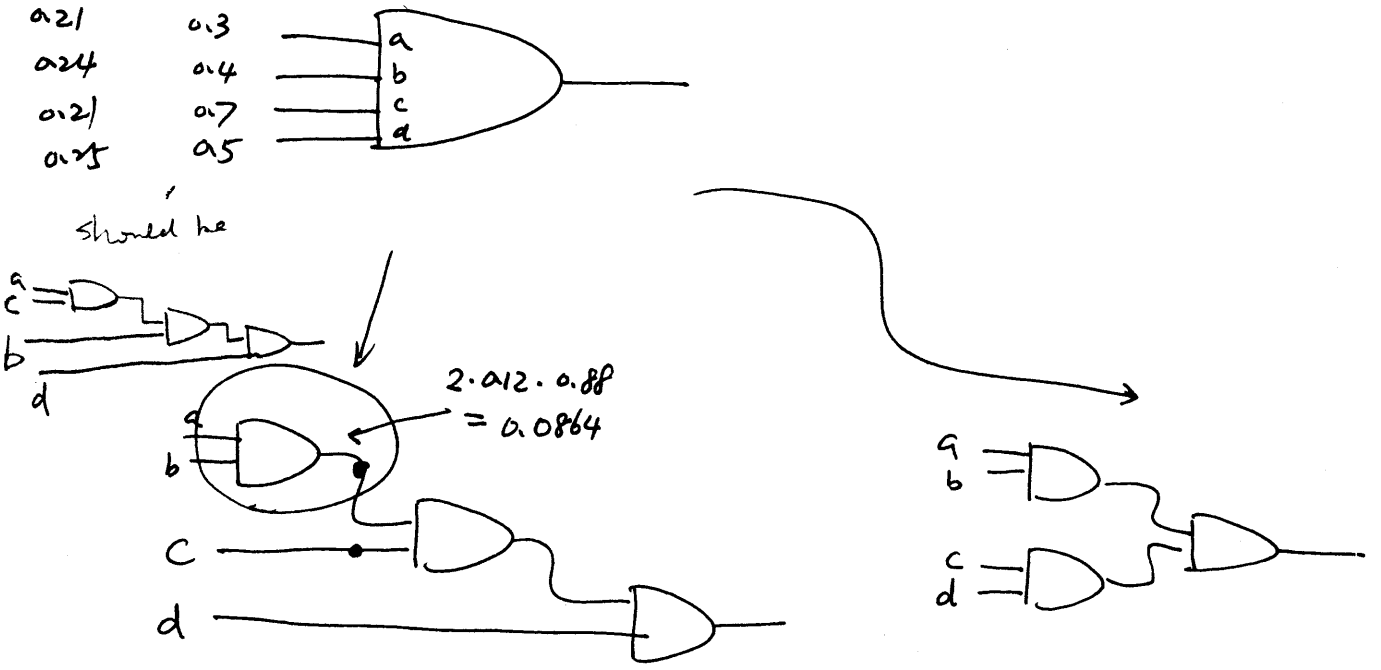
• Technology Decomposition for Low-power

* Convert a Boolean network to another set consisty of only 2-input NAND and inverter gates.

* Try to minimize the sum of switching activities at the internal nodes of the network.

* principle: try to inject high switchig activity inputs to the decomposition tree as late as possible

Exaple: show by AND.



Like Huffman's Code generation

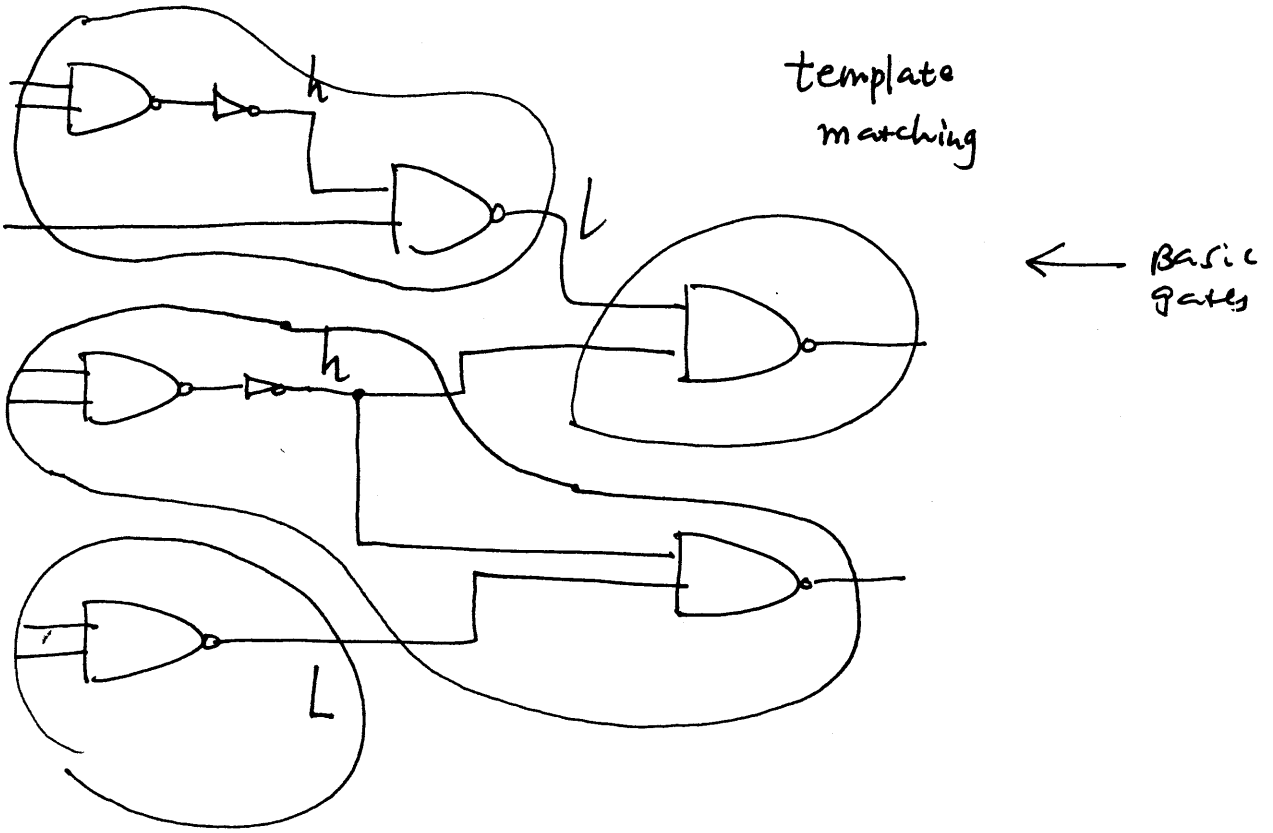
$E^A(SW) = 0.246$

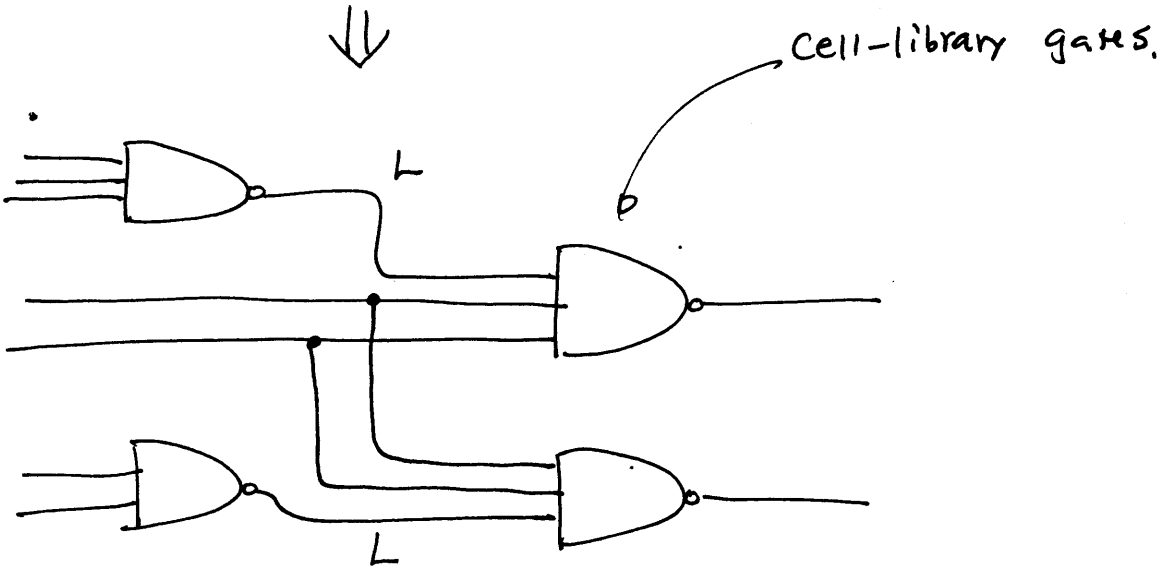
$E^B(SW) = 0.512$

Cell Binding for Low-power

- principle: Try to hide nodes with high switching activity inside the gates where they drive smaller load capacitances.

Example:



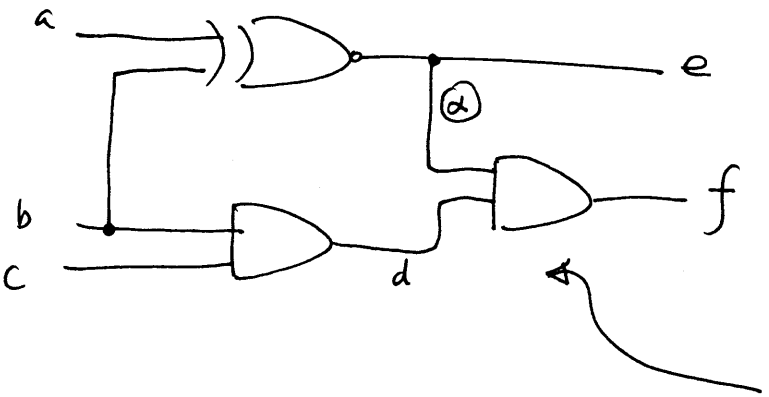


- ~~Power~~ post-Mapping power optimization

- performed after technology mapping has been finished.

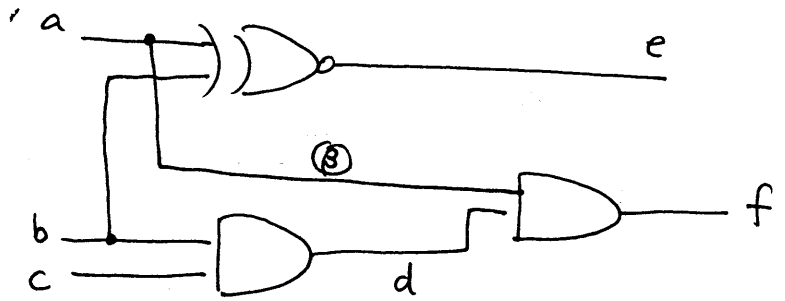
- Try to reduce power further by re-wiring.

Example:



All gates are provided by cell library.

functionally Equivalent



If the switching activity of β is < that of α & input gate capacitance is large

$\Rightarrow$ the ~~circuit~~ re-wiring can save significant of power.