Sequential Circuit Synthesis and optimization for low Power

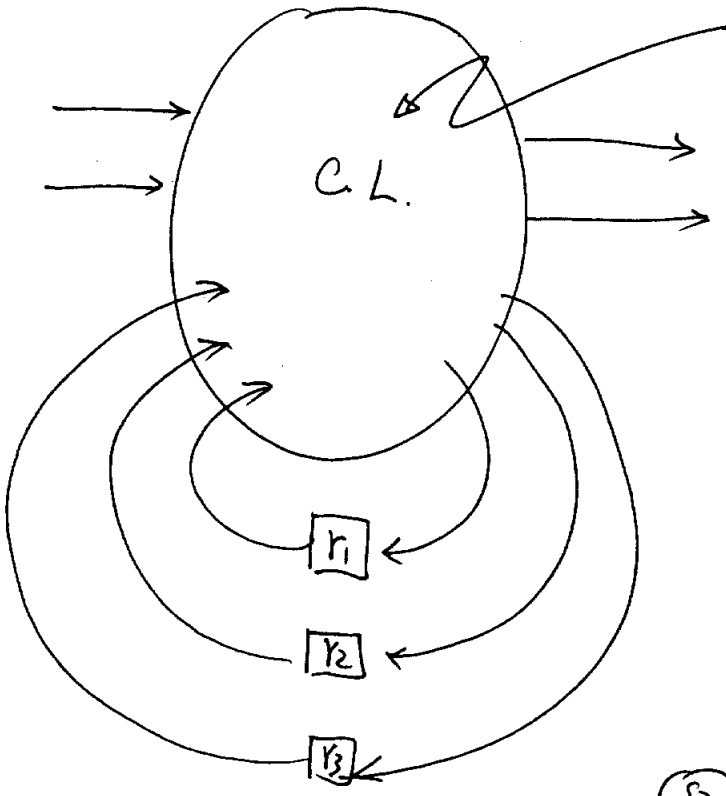
- Sequential ckt Synthesis Flow.

① State assignment To be discussed.

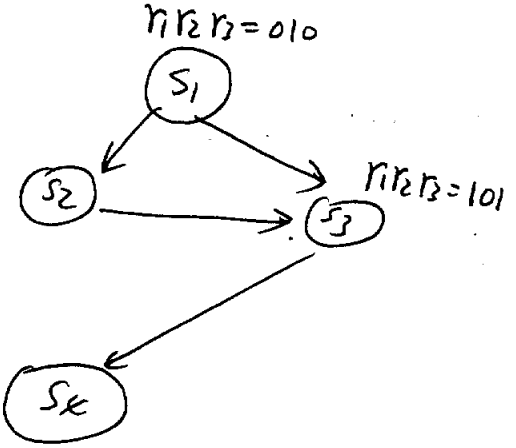
② Logic optimization (technology independent) Done!

③ Technology mapping Done!

④ post-mapping optimization (e.g., re-wiring). Done!

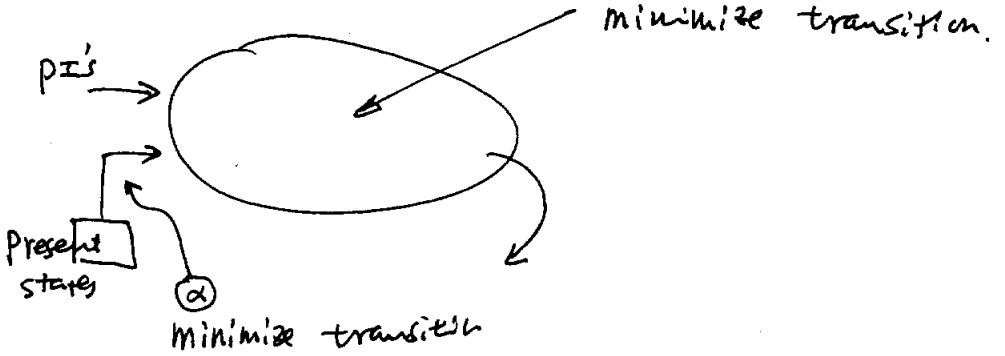


once states are assigned, CL can be optimized for power.



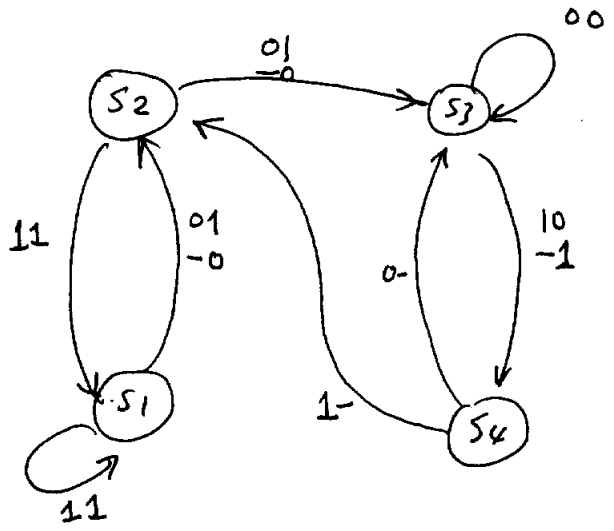
- We will discuss ① Low-power state assignment
  ② Re-timing
  ③ Clock shut-down

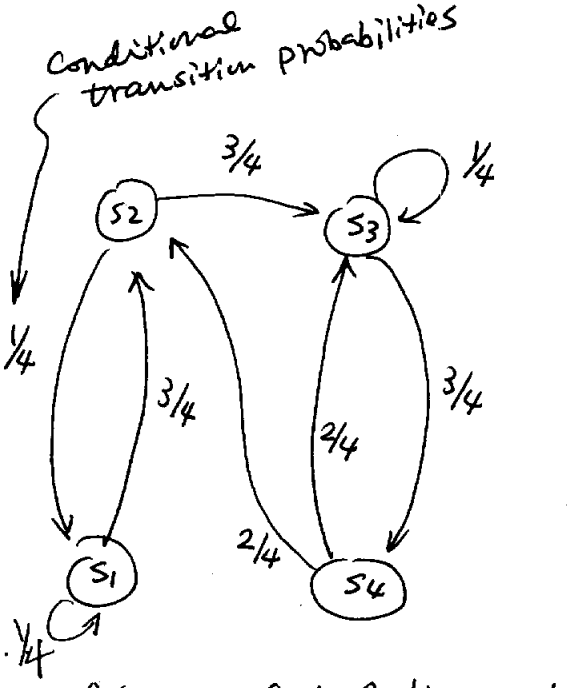
- Low-power State Assignment



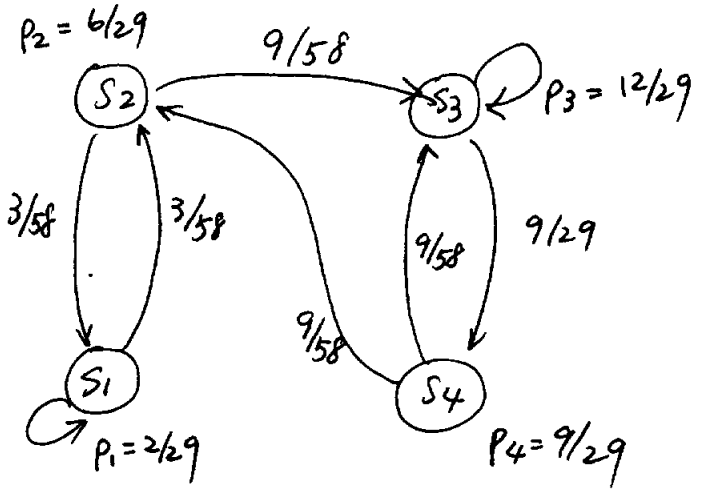
Try to find state assignments to reduce the # of logic transitions at the ckt present-state input (α) between two consecutive clock cycles.

- STG & probabilities





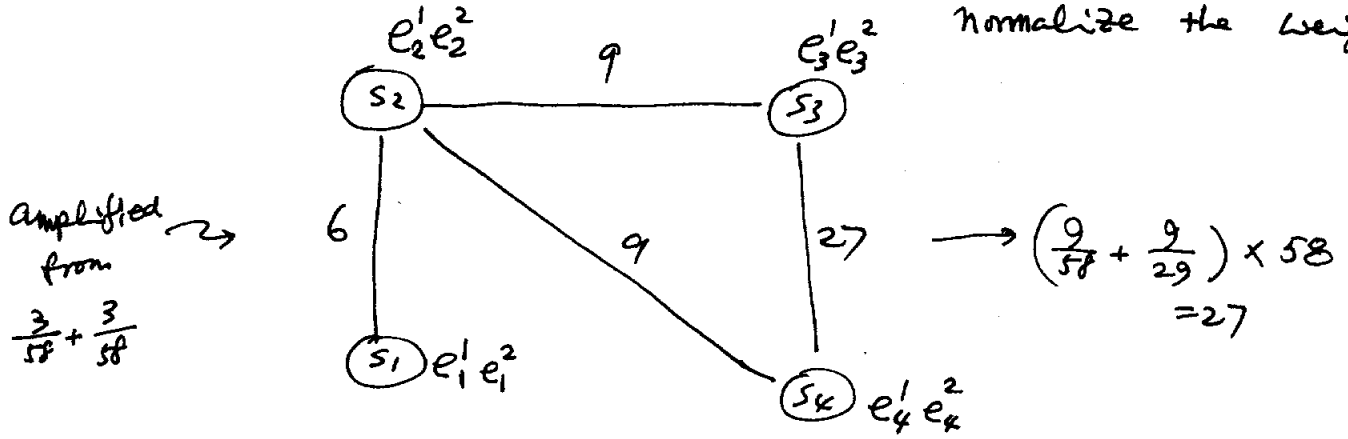
Assume equiprobable and independent input signals



$\frac{12}{29} \times \frac{3}{4} = \frac{9}{29}$

- problem formulation: integer linear programming (ILP)

Show by example:



* use 2 state variables

$$\begin{array}{c} \\ S_1 \\ S_2 \\ S_3 \\ S_4 \end{array} \begin{array}{c} \begin{array}{cc} e_1 & e_2 \end{array} \\ \begin{bmatrix} e_1^1 & e_1^2 \\ e_2^1 & e_2^2 \\ e_3^1 & e_3^2 \\ e_4^1 & e_4^2 \end{bmatrix} \end{array}$$

$$e_1^1 \oplus e_2^1 + e_1^2 \oplus e_2^2 \geq 1$$
$$e_1^1 \oplus e_3^1 + e_1^2 \oplus e_3^2 \geq 1$$
$$\vdots$$
$$e_3^1 \oplus e_4^1 + e_3^2 \oplus e_4^2 \geq 1$$

} Constraints

Minimum cost:

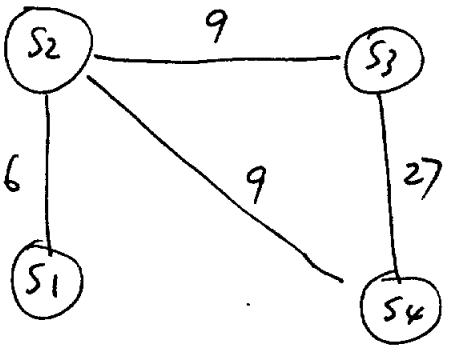
$$6\,(e_1^1 \oplus e_2^1 + e_1^2 \oplus e_2^2) + 9(e_2^1 \oplus e_3^1 + e_2^2 \oplus e_3^2)$$
$$+ 9\,(e_2^1 \oplus e_4^1 + e_2^2 \oplus e_4^2) + 27\,(e_3^1 \oplus e_4^1 + e_3^2 \oplus e_4^2)$$

∴ NP Complete

For large STGs, the exact ILP solution is not possible

- Semi-exact algorithm ⇒ Try to $\min\left(\sum_{h=1}^{\#\text{ of edges}} w_{i,j}^h\,(e_i^L \oplus e_j^L)\right)$

*Always try to glue states and assign same bit patterns for large-weight nodes.



	V_1	V_2
S_1	0	X
S_2	0	X
S_3	1	X
S_4	1	X

↑ Try to assign the first state variable

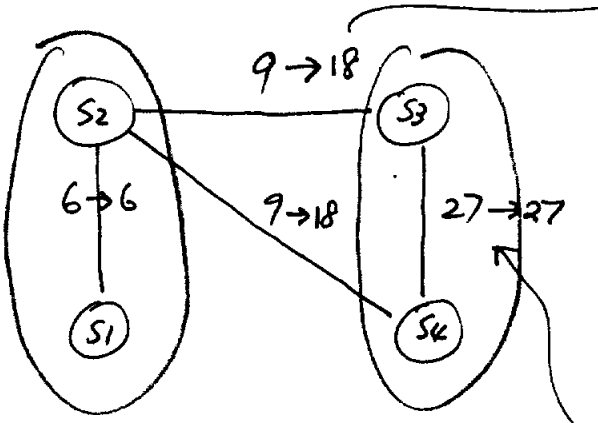
S_3, S_4 have the largest transition probability.

∴ should be as similar as possible

min cost: $0.27 + 0.6 + 1.9 + 1.9 = 18$ ✓ $(0\times27) + (0\times6) + (1\times9) + (1\times9) = 18$

other costs: (S2S3) (S1S4) we have $27.1 + 9.1 = 36$

Recompute the weights of the STG



→ the hamming distance between S_2 and S_3 is 1 (based on the particular state assignment)

$9(d_{ij}+1)$
$=$
1

two state with large hamming distance
↓
have been very different
↓
should be glue together by assigning larger weight

$27(d_{ij}+1) = 27$
$=$
0

	V_1	V_2
S_1	0	0
S_2	0	1
S_3	1	1
S_4	1	0

→ S_2 & S_3 high weight

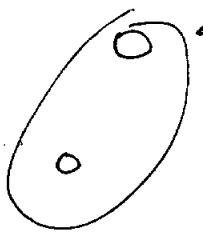
∴ minimize the difference

min. cost: $6\cdot1 + 18\cdot1 + 18\cdot1 + 27\cdot1 =$ _____
(S_1S_2) (S_2S_3) (S_2S_4) (S_3S_4)

other cost: _____. larger.

- Still, not a good solution for large-sized STG.

A Heuristic Algorithm



← just know these two, glue

how to find others for the same group??

Def. Indistinguishability class:

Two states s_i and s_j having the same partial code of order L (codes are the same with L bit) are said to belong to the same indistinguishability class

Example:

	v_1	v_2
s_1	0	X
s_2	0	X
s_3	1	X
s_4	1	X

S1 and S2 (S_3 and S_4) are in the same indistinguishability class.

Def. We call indistinguishability classes, $C_{k,L}$, $k=1,\dots n_{class}$, the groups of states with equal partial codes after the assignment of the preceding $L-1$ variables

Example:

	v_1	v_2
s_1	X	X
s_2	X	X
s_3	X	X
s_4	X	X

$C_{1,1}$: 1 = # of class; 1 = preceding 0 bit assigned

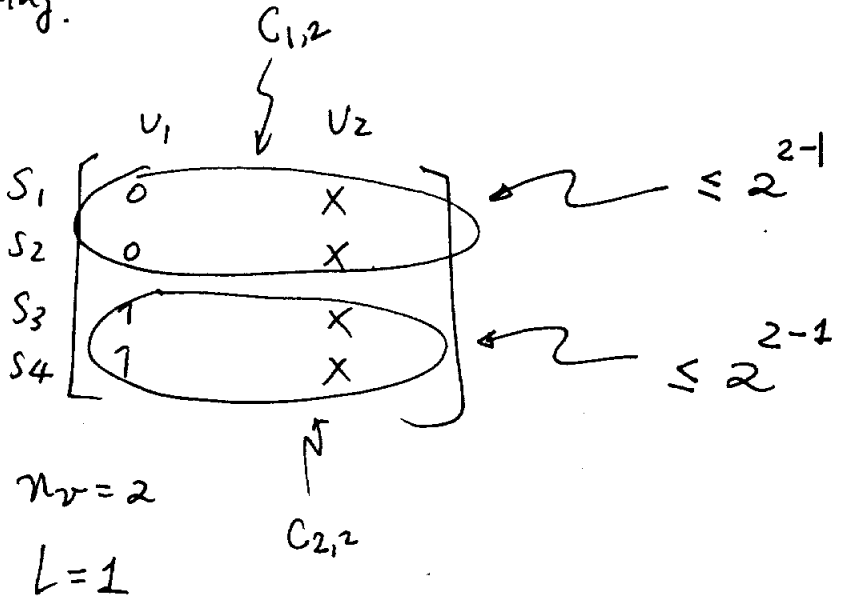
	v_1	v_2
s_1	0	X
s_2	0	X
s_3	1	X
s_4	1	X

$C_{1,2}$: 1 = # of class; 2 = preceding 1 bit has been assigned

$C_{2,2}$: 2 = # of class

property: The maximum # of indistinguishable partial state codes, after the assignment of the l-th bit, must be less than or equal to $2^{n_v - l}$ (n_v: # of state variables); otherwise, no unique code can be assigned to such states with the remaining bits of encoding.

Example:



The problem is

Min $\left(\sum_{h=1}^{\text{\# of edges}} w_{i,j}^{h} \left(e_i^l \oplus e_j^l \right) \right)$ such that ← try to assign l's state variable

$$\begin{cases} \sum_{s_i \in C_{K,l+1}} e_i^l \leq 2^{n_{var} - l} \\ \sum_{s_i \in C_{K,l+1}} e_i'^l \leq 2^{n_{var} - l} \end{cases}$$

$\forall C_{K,l}$

all $s_i \in C_{K,l+1}$ with $e_i^l = 1$

all $s_i \in C_{K,l+1}$ with $e_i^l = 0$

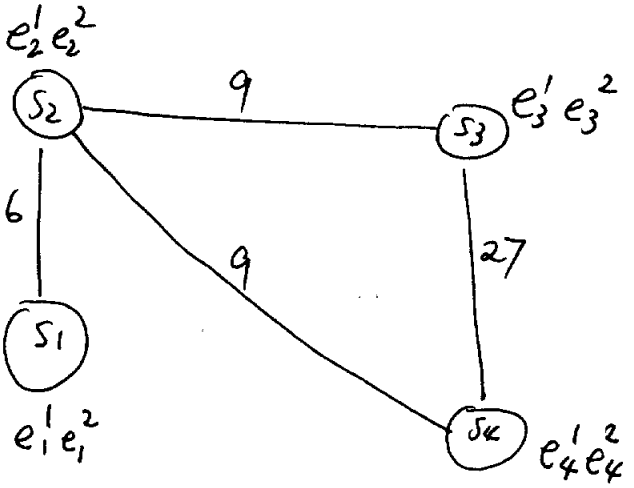
E.g. $n_{var} = 2$, $l = 1$.

```
Assign (S) {
    Sort edges by weight in decreasing order;
    for each edge (Si, Sj) {
    if (Si and Sj are not assigned)
    {
        if (no class violation)
        {
            x = select-bit (Si, Sj);
            ei = x; ej = x;
        }   else
        {
            x = select-bit (Si, Sj);
            ei = x; ej = x';
        }   else
    }   else
    if (Si or Sj not assigned)
    {
    β   Sh = unassigned (Si, Sj);
        Sg = assigned (Si, Sj);
        if (no class violation)
        {
            x = select-bit (Sh);
```

```
            e_h = x ;
        } else
    α
            e_h = ⊗ e_g^l ;
        }
    }
}
```

```
for  l = 1  to  nvar
{
    adjust class constraints;
    assign (s);
    adjust the edge weight;   ⇝ w_{i,j}^h (new)
}                                = w_{i,j}^h (old) (d_{i,j} + 1)
```

Example :



| = 1 ;

adjust class constraints. My assignment must have Each class must have $\leq 2^{2-1}$ states.

assign (5);

★ what's class violation?
If two states can be assigned the same code without violating the number of states in each indistinguishability → we call them "No class violation"

→ process (S3) —27— (S4)
none assigned 1st bit
no class violation;

x = select_bit(S_3, S_4), randomly choose 1;

$e_3^1 = 1$; $e_4^1 = 1$;

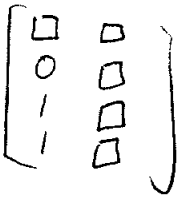
key point:
Alway try to assign the same code to two states, unless class violation might occu

→ process (S2) —9— (S3)

S_3 assigned,

class violation (assign 1 to e_2^1 cause violation)

$\therefore e_2^1 = 0$;



→ process (S2) —9— (S4)

~~S_4 assigned~~

~~Class violation (Assign 1 to S_2~~

Both S_2 and S_4 are assigned

Stop

→ process S_1 —6— S_2

S_2 assigned.

~~no~~ ~~state~~ NO class violation

∴ $e_1^1 = 0$;

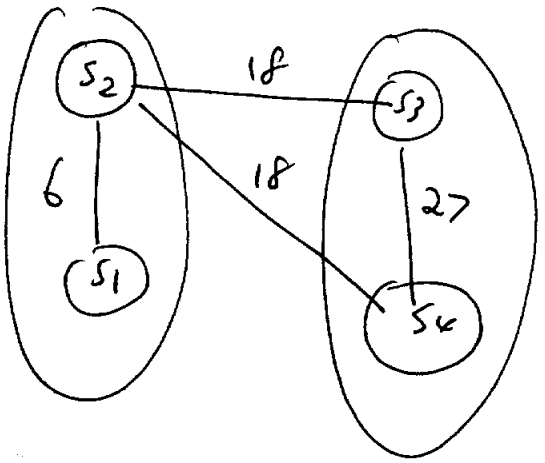
↑ S_0, S_1 & S_2 can be assigned the same code.

∴ 1st bit has been assigned

	e_1	e_2
S_1	0	×
S_2	0	×
S_3	1	×
S_4	1	×

We were stuck here!
Sometimes, your professor can be "stupid!!".

⇒ adjust the edge weights

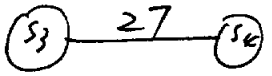


L=2

⇒ adjust the class constraints. My new assgnt for L=2 must have

Each class must have $\leq 2^{e-2}$ states

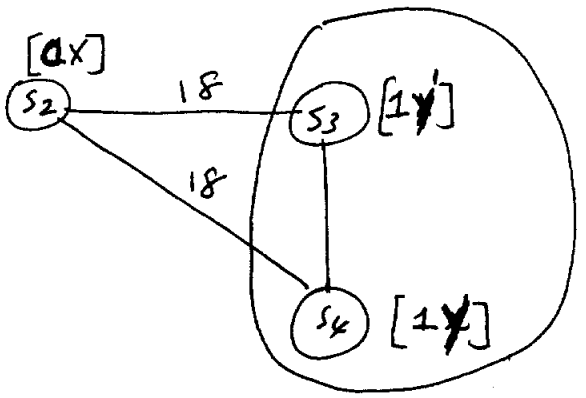
∴ assign (5)

⇒ process 

None assigned ~~first~~ 2nd bit, same class $\begin{bmatrix} 0 & \square \\ 0 & \square \\ 1 & \square \\ 1 & \square \end{bmatrix}$.

Class violation

∴ select-bit (S_3, S_4)



if Y assigned 0,
⇒ cost = ~~36~~ 18 + 18
= 36.

if Y assigned 1
⇒ cost = 18 + 18 = 36

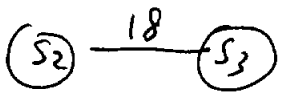
$\begin{bmatrix} 0 & \square \\ 0 & \square \\ 1 & 1 \\ 1 & 0 \end{bmatrix}$

X can be randomly chosen 0 or 1.

★ Try to find one with smaller cost

∴ $e_3^2 = 1$ $e_4^2 = 0$

$$cost = \sum_{\substack{\text{all other states with} \\ \text{diffent assign} \\ \text{from } S_3}}^{\text{weight}} (S_3, S_a) + \sum_{\text{weight}} (S_4, S_a)$$

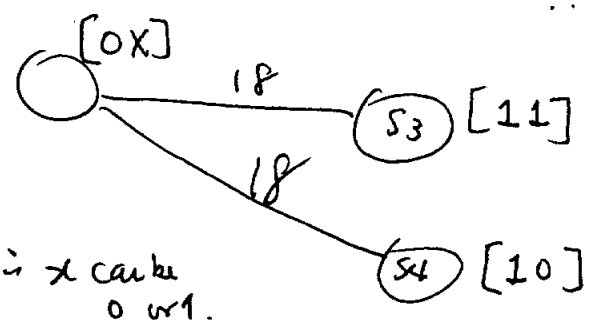
⇒ process 

S_3 : assigned

S_2 : unassigned

no class violation (S_2 and S_3 are not in the same class)

X = select-bit (S_2)

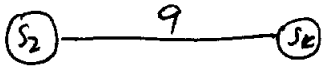


∴ x can be 0 or 1.

Let $e_2^2 = 1$

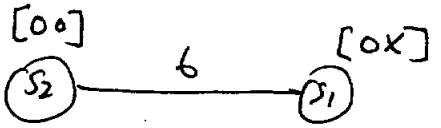
$$\begin{bmatrix} 0 & \square \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$\Rightarrow$ process



Both assigned

$\Rightarrow$ process



in the same class

Class violation $\longrightarrow$ S_2 assigned
S_1 unassigned

$e_1^2 = 0$

Done!

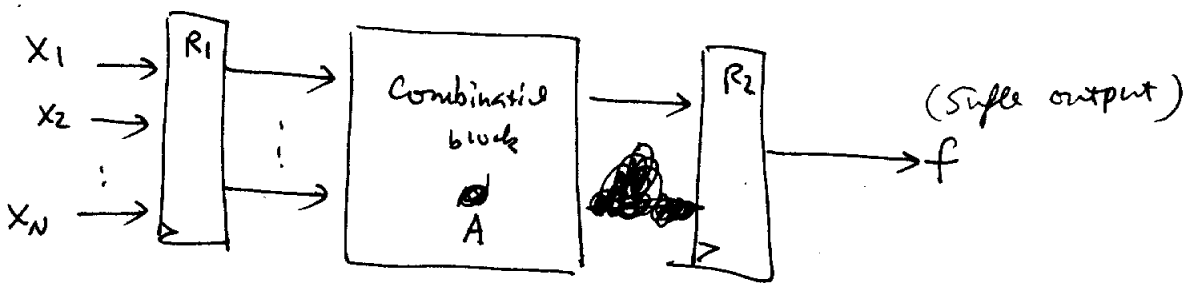
	e_1	e_2
S_1	0	0
S_2	0	1
S_3	1	1
S_4	1	0

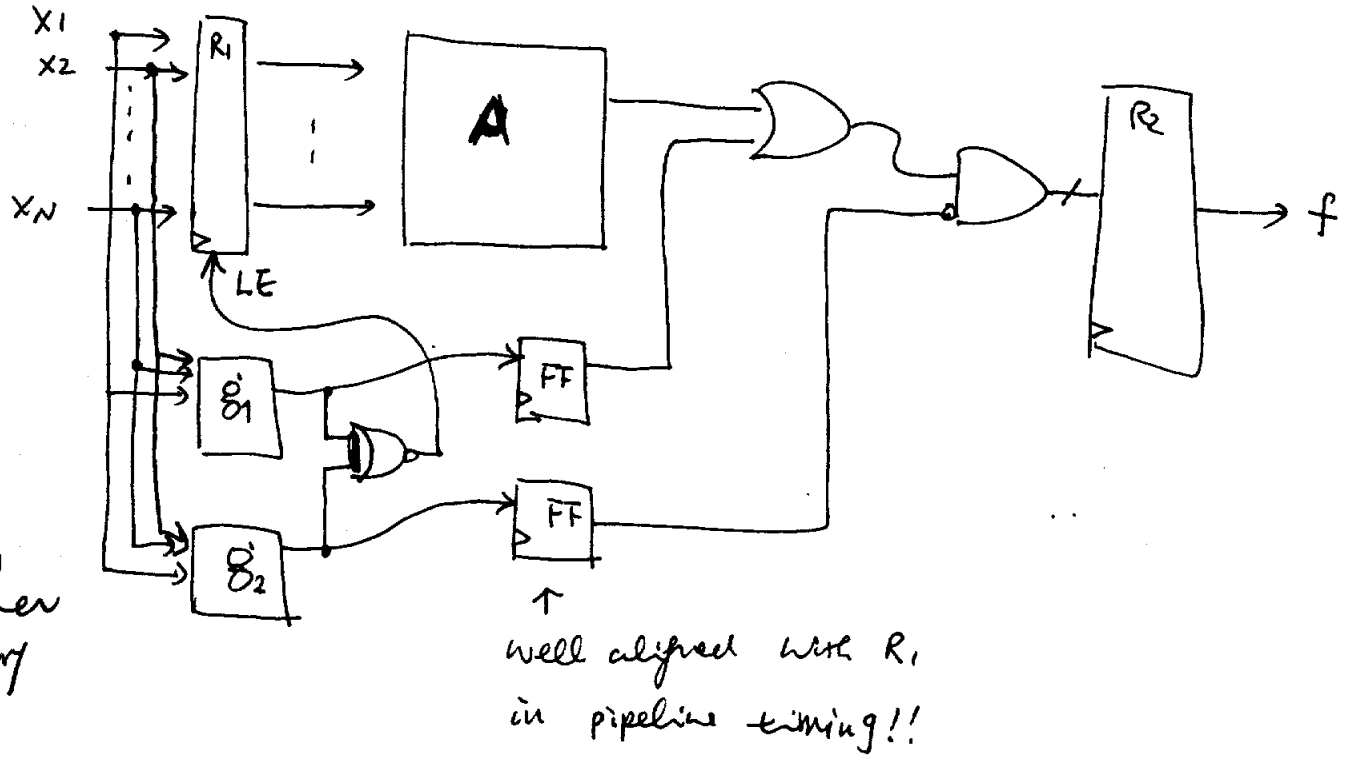
Logic Shut-down

- Try to shut down the logic which is not in use during some particular clock cycles, such that power dissipation can be reduced.

  * pre-computation
  * Gated clocks

- pre-computation : First architecture (pipeline structure)

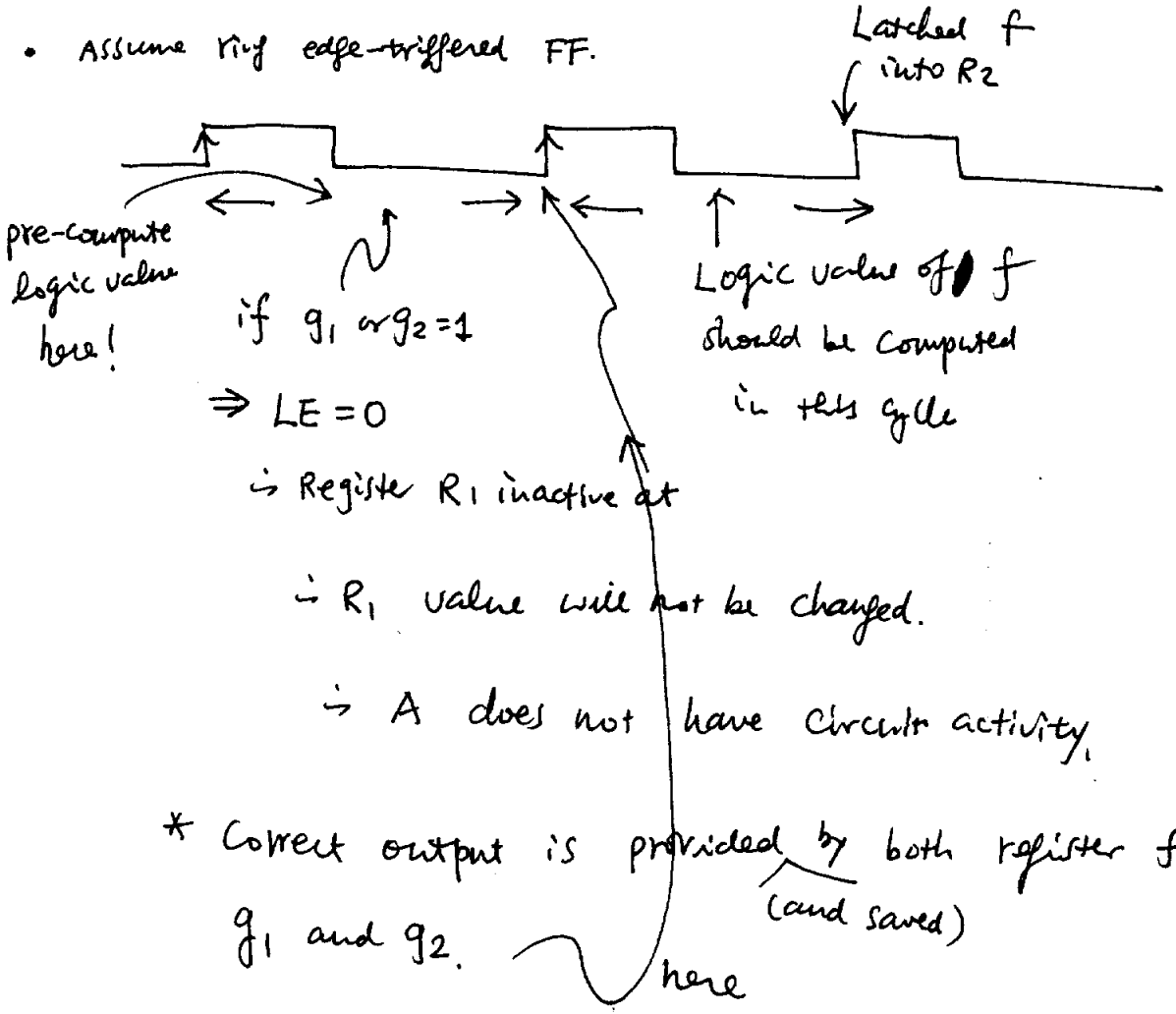




$g_1 = 1 \Rightarrow f = 1$

$g_2 = 1 \Rightarrow f = 0$

- g_1 and g_2 cannot be both 1.
- Assume ring edge-triggered FF.



∴ Registe R_1 inactive at

∴ R_1 value will not be changed.

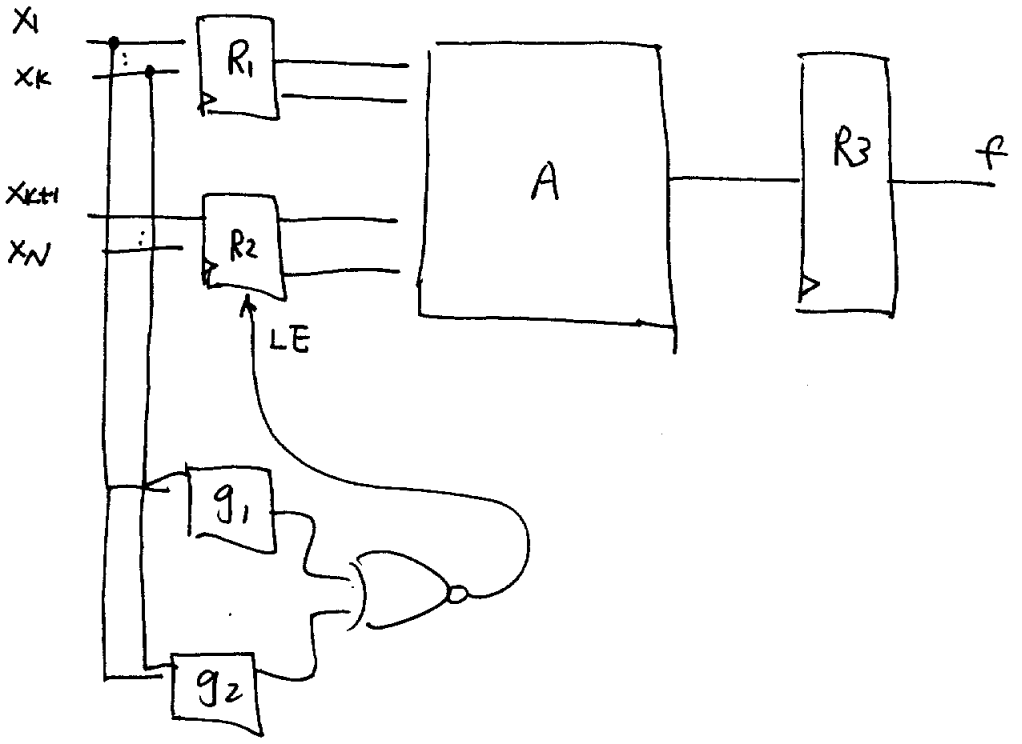
∴ A does not have circuit activity.

* Correct output is provided by both register following g_1 and g_2. (and saved) here

- problems:
  - * g_1 & g_2 can't be too complex.

    Otherwise, power saved at A will lose at g_1 and g_2.
  - * Choice of the predictor function is not easy.
  - * Add extra delay to the ckt in normal function. So can be applied to non-critical logic blocks only.

Pre-Computation: Second architecture



- Major difference:

① Inputs are partitioned into two subsets, and only one of them feeds g_1 and g_2.

(∴ Still use R_1 to generate function f) instead of use 2 extra FFs.

inputs of R_1 can uniquely determine f.

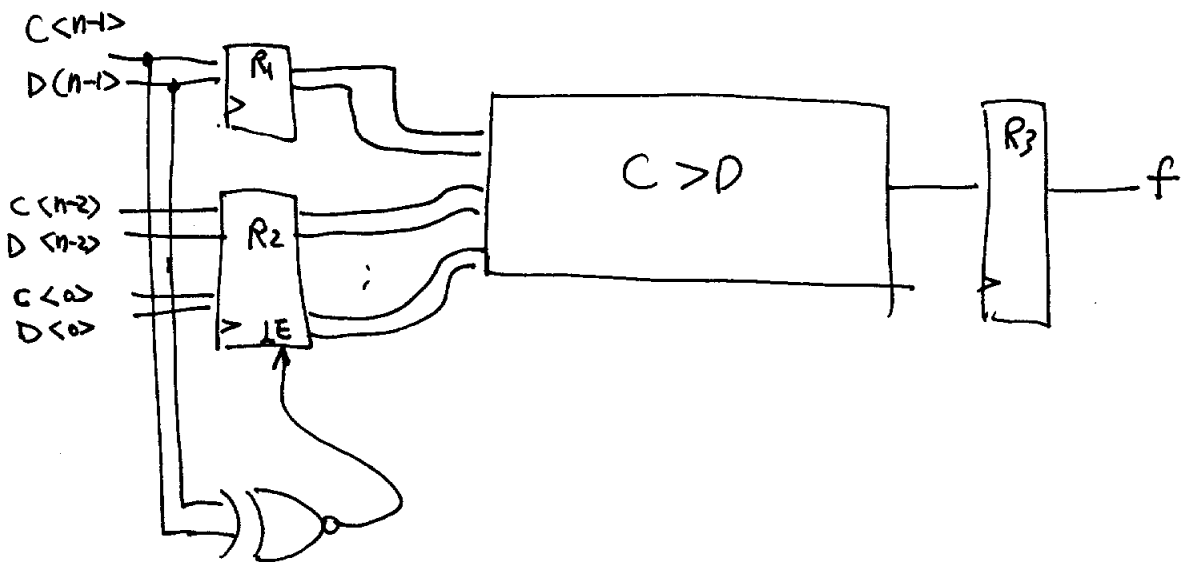
② Only one extra NOR gate is required.

Note:

⇒ • g_1 and g_2 only determine whether the selected subset of inputs to block A can be frozen

• the architecture does not calculate and store in advance the output values for the subsequent clock cycle.

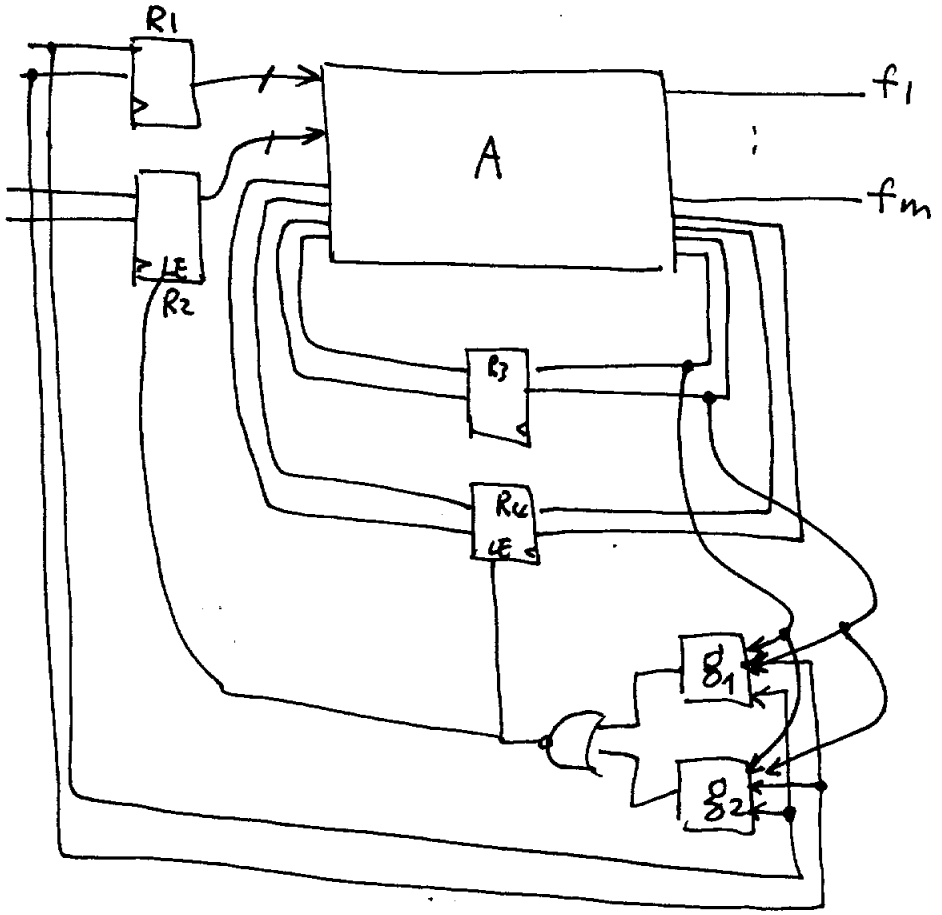
Example:



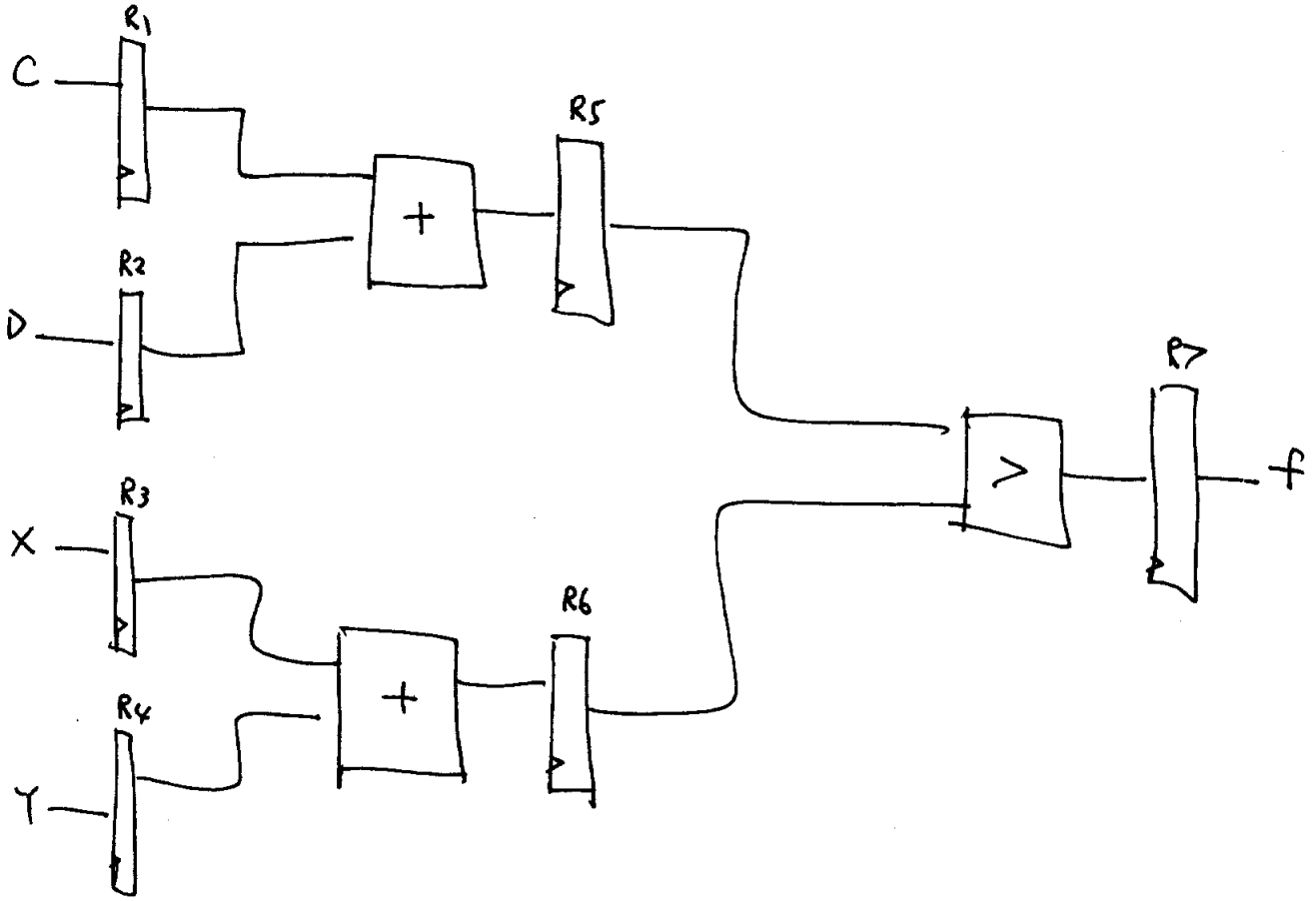
$$g_1 = C\langle n-1\rangle \, \overline{D\langle n-1\rangle}$$

$$g_2 = \overline{C\langle n-1\rangle} \, D\langle n-1\rangle$$

Generalized architecture for 2-type pre-computation



Example:



Compute $(C+D) > (X+Y)$

Two cycle pre-computation:

$$g_1 = C(n-1) \cdot D(n-1) \cdot \overline{X(n-1)} \cdot \overline{Y(n-1)}$$

$$g_2 = \overline{C(n-1)}\ \overline{D(n-1)} \cdot X(n-1) \cdot Y(n-1)$$

$$P(g_1 + g_2 = 1) = \frac{2}{16}$$

$\doteq$ 12.5% of time we can use the Lowpower mode.

Gated Clocks

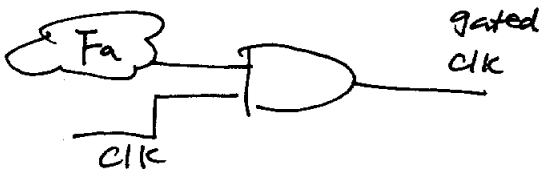
- Selectively stop the clock and thus force the C.L. to make no transition, whenever the computation to be performed is useless.

- Three-step procedure

  ① Determine a Boolean function, Fa, to activate the clock gating function

  ② Generate a single-output combinational logic which implements the activation function

  ③ Take the Logic AND of the activation ckt and the original CLK.



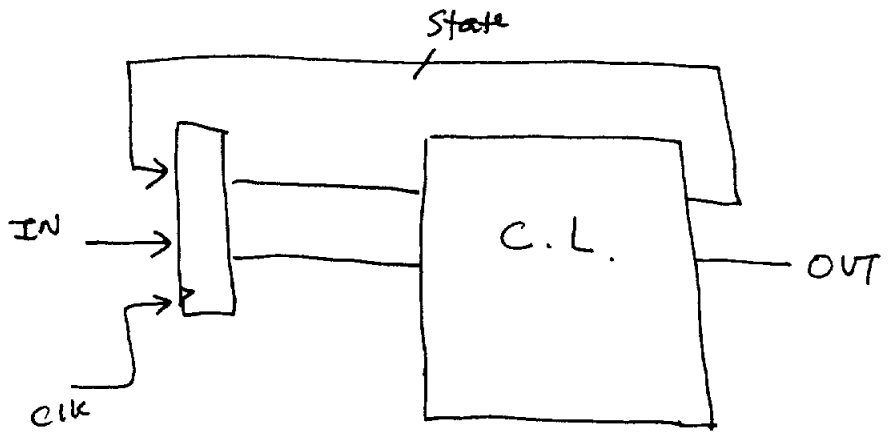
- Major problems.

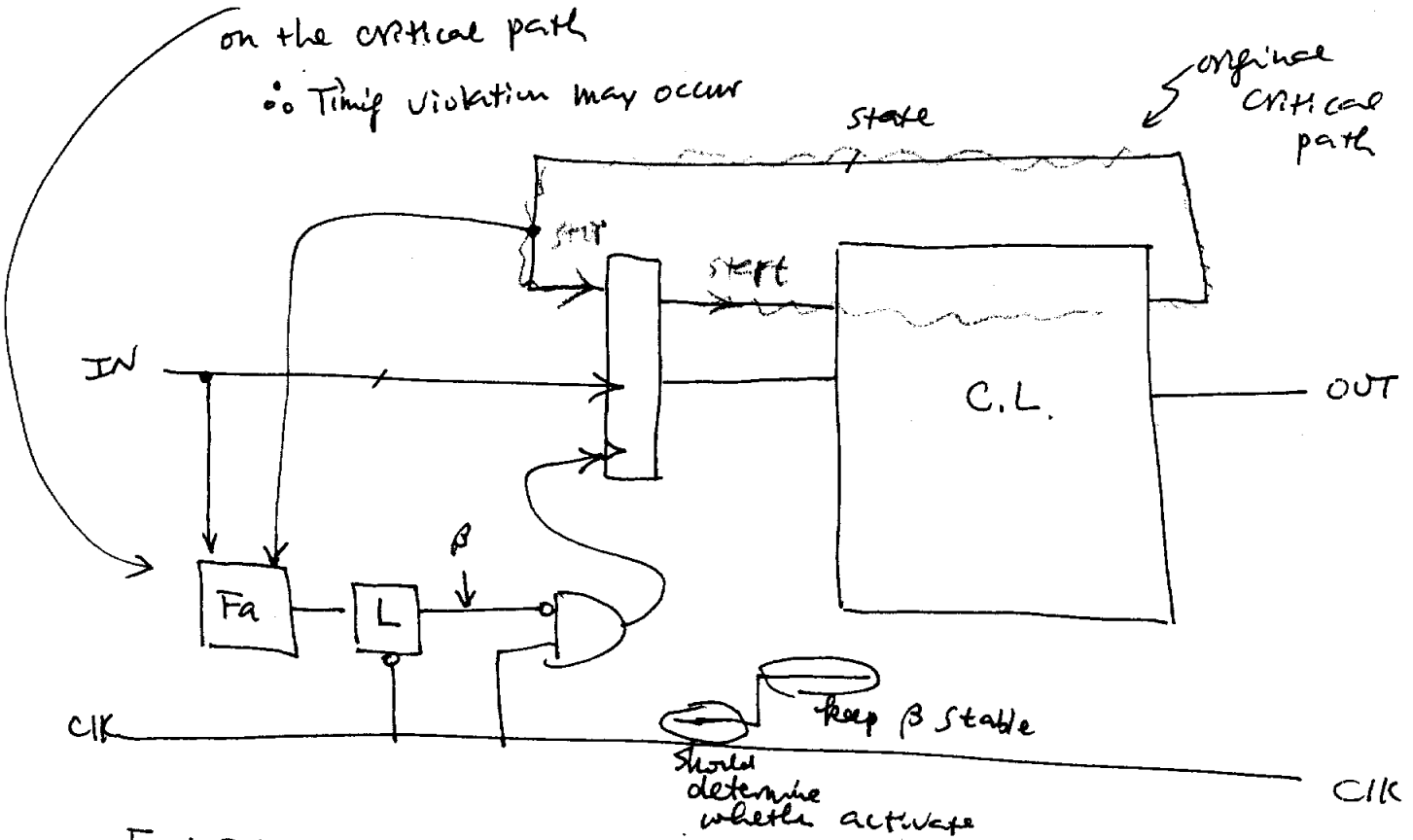
  ① Fa may introduce some timing violations.

  E.g., hazard & race

  ② properly detecting the idle conditions is not trivial

  ③ power consumed by Fa may offset the saving in CL P.C.





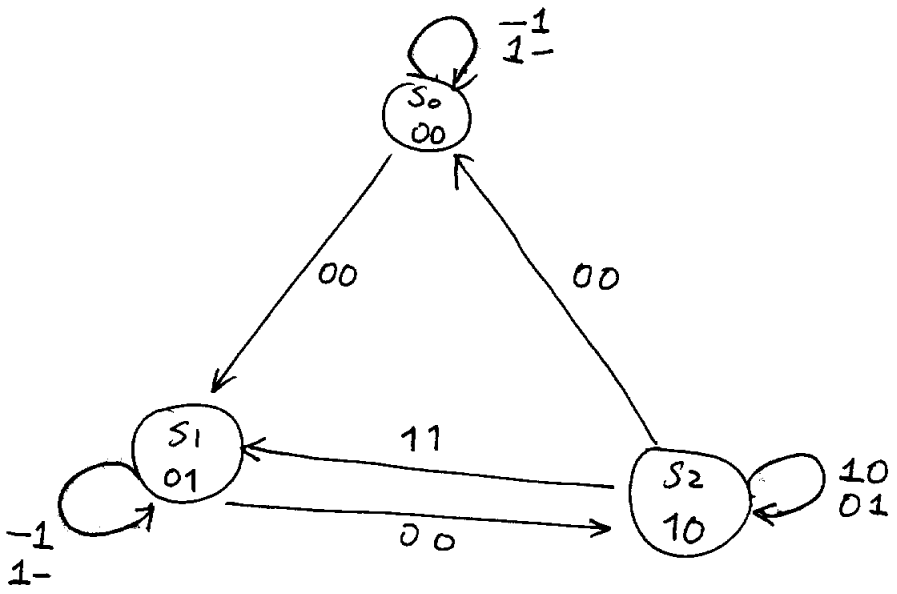
Fa: Selectively stop the clk of the ckt any time no state and output transition taker place.

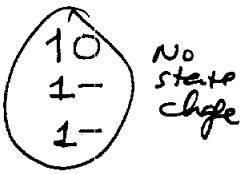
L: Trasparent when CLK = 0

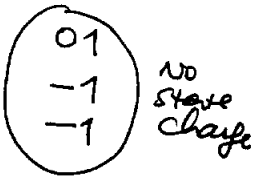
It filters glitches occur in the output of Fa.

How to identify the function of Fa ?

- Moore machine, find the self-loop states.

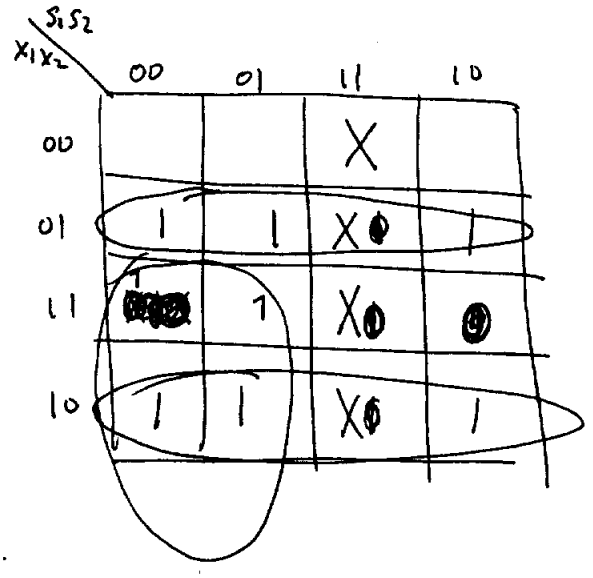






$$fa = x_1' x_2 + x_1 x_2' + x_1 s_1'$$

fa



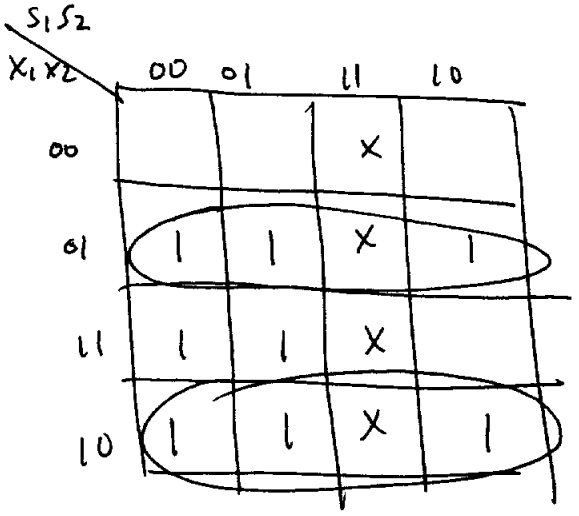
fa: 6 literals

Fsm: 23 literals

⇒ Area overhead = 25%

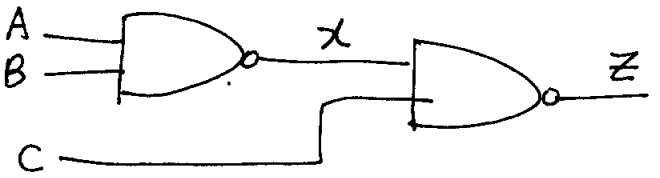
- If overhead is too large, try to reduce by selecting a subset of the f_a function

e.g.



$$f_a = x_1' x_2 + x_1 x_2'$$

① <a>



$P_1(X) = 1 - P_1(A)\,P_1(B) = 0.75 = \frac{3}{4}$

$P_1(Z) = 1 - \frac{1}{2}\cdot\frac{3}{4} = 1 - \frac{3}{8} = \frac{5}{8}$

$E_{sw}(A) = E_{sw}(B) = E_{sw}(C) = 2\cdot\frac{1}{2}\cdot\frac{1}{2} = \frac{1}{2}$

$E_{sw}(X) = 2\cdot\frac{3}{4}\cdot\frac{1}{4} = \frac{3}{8} = \frac{12}{32}$

$E_{sw}(Z) = 2\cdot\frac{5}{8}\cdot\frac{3}{8} = \frac{15}{32}$

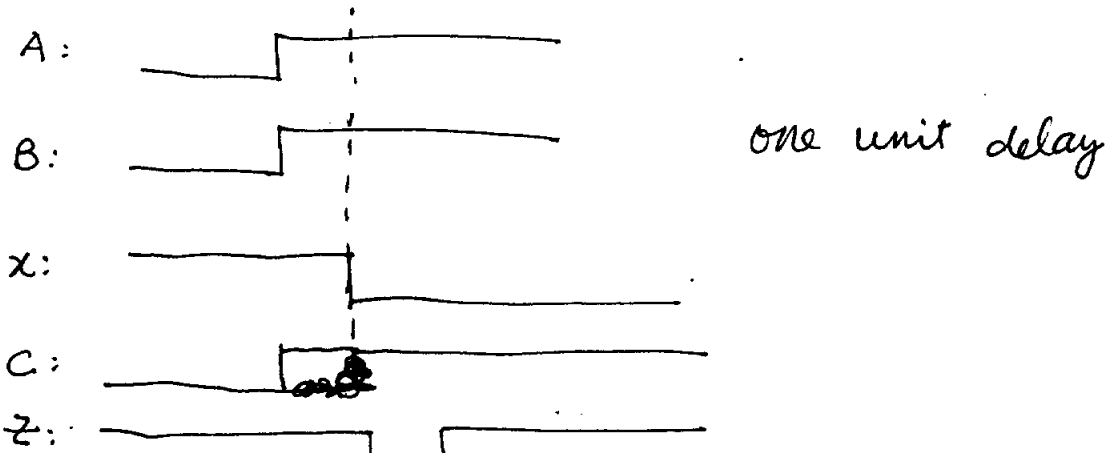
$$\text{power} = \frac{1}{2}\cdot(5)^2\cdot(300\times10^6)\left[\frac{1}{2}\times3\times13\times10^{-12} + \frac{27}{32}\times60\times10^{-12}\right]$$

$$= 12.5\times300\times10^6\left[19.5 + 50.625\right]\times10^{-15}$$

$$= 262969\times10^{-9}\ \text{W}$$

$$= 262.969\times10^{-6}\ \text{W} \quad \text{or } 262.969\ \mu\text{W}.$$

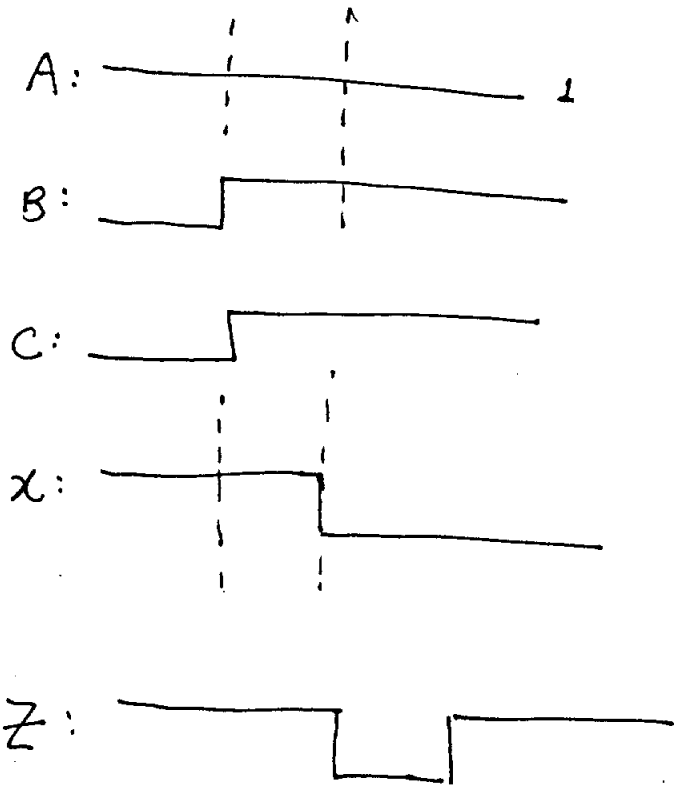
<b>



one unit delay

<C> Increase the power consumption

(2)



Totally, 5 transitions

∴ Short ckt powe consumption

$$= \frac{5}{2} \times \frac{\beta}{12} (V_{dd} - 2V_T)^3 \tau f \qquad f = 1$$

$$\therefore \frac{5}{24} \cdot \beta \cdot (V_{dd} - 2V_T)^3 \tau$$

Note: $\frac{\beta}{12} (V_{dd} - 2V_T)^3 \tau f$ is P_{short} for 2 transitions!!!

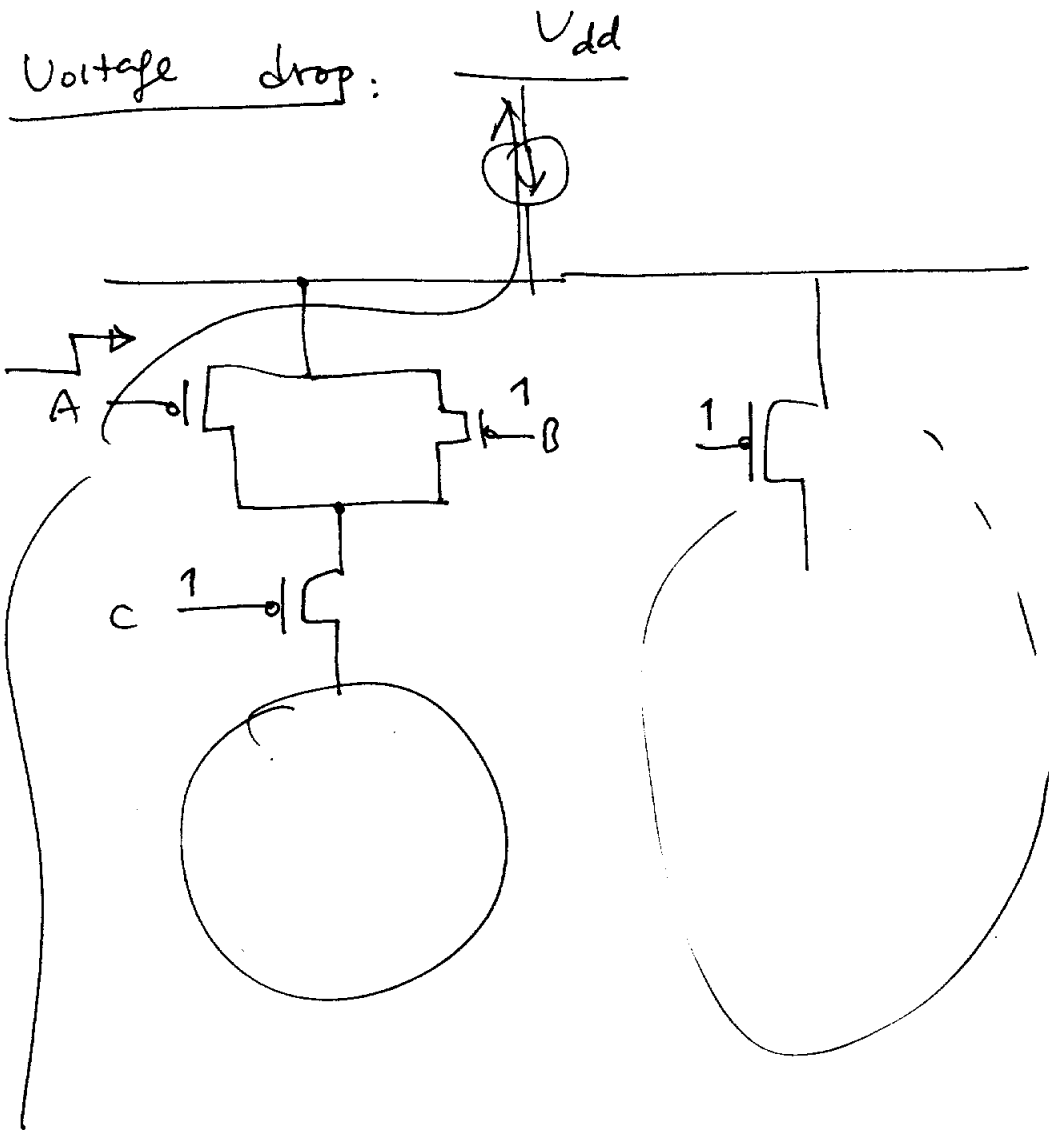
<3> Although the low-activity nodes might not be fully ~~simulation~~ simulated, this should be tolerable. For exaple, a node with switchig prob. = $\frac{3}{1000}$ might be estimated $\frac{1}{1000}$. However, this should not give too much trouble since the low-activity nodes contributed small power dissipation anyhow.

<4>

$$P_{ave} = \frac{1}{T}\int_0^T V_{dd} \cdot i_{dd}(t)\, dt$$

$$= \frac{V_{dd}}{T}\int_0^T i_{dd}(t)\, dt$$

Voltage drop:

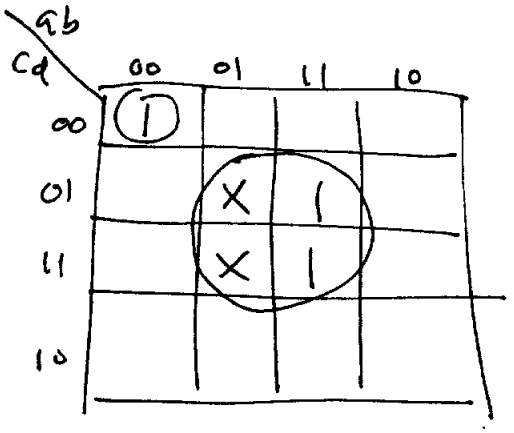


instantaneous reverse

current introduced by ⌐¬ transition

at A.

(5) This problem tests whether you have good understanding about relations between don't care and power dissipation.



* prime implicants : $\bar{a}\bar{b}\bar{c}\bar{d}$ and bd

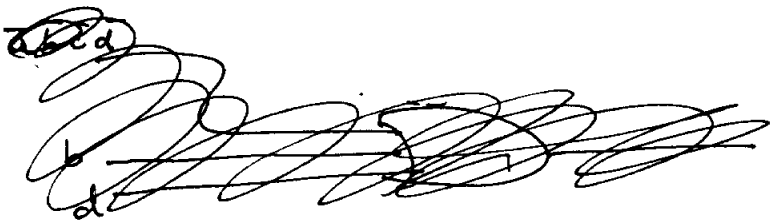
  we don't have to do anything for $\bar{a}\bar{b}\bar{c}\bar{d}$

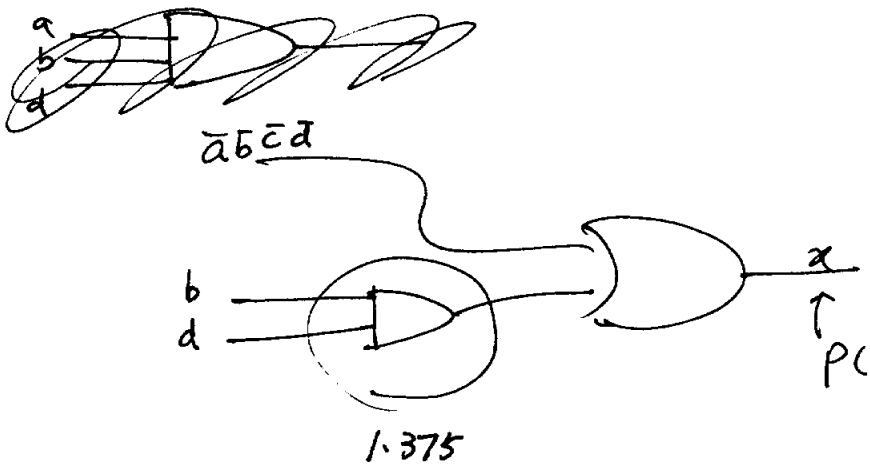
* must check abd, $b\bar{c}d$, bcd (Note: $\bar{a}bd$ must not be checked).

power (bd)

$$2(0.5)(0.5) + 2(0.5)(0.5) + 2\,\frac{1}{4}\cdot\frac{3}{4}$$

$$= 1 + \frac{3}{8} = \frac{11}{8} = 1.375$$





$P(x=1) = P(\bar{a}b\bar{c}\bar{d}) + P(bd)$

$= (0.9)\cdot(0.5)\cdot(0.1)\cdot(0.5) + (0.5)\times(0.5)$

$= 0.0225 + 0.25$

$= 0.2725$

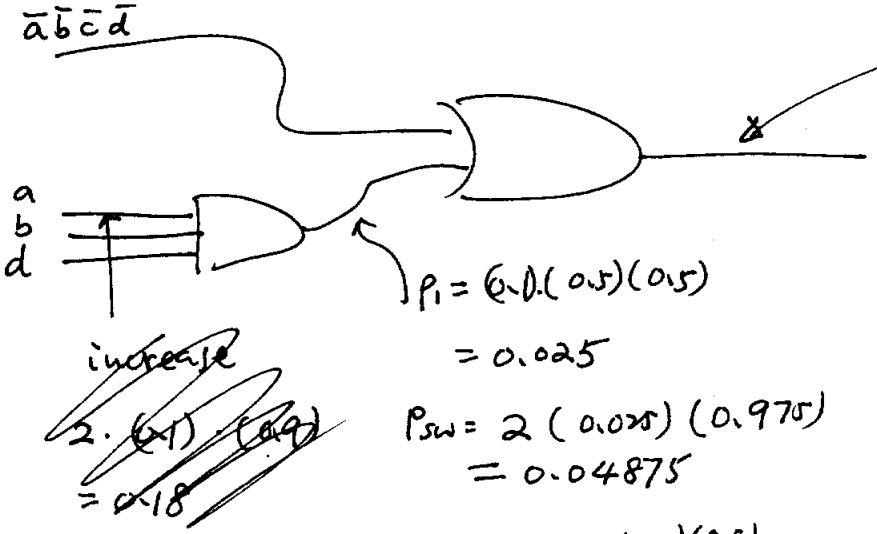
$2(0.2725)(1-0.2725)$

$= 0.3965$

$\sum$ $1.375 + 0.3965$

$= 1.7715$

How about abd ?



$P(x=1)$

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

$= 0.0225 + (0.1)(0.5)(0.5)$

$= 0.0225 + 0.025$

$= 0.0475$

$2(0.0475)\cdot(1-0.0475)$

$= 0.090.$

$P_1 = (0.1)(0.5)(0.5)$

$= 0.025$

$P_{SW} = 2(0.025)(0.975)$

$= 0.04875$

increase

$2\cdot(0.1)\cdot(0.9)$

$= 0.18$

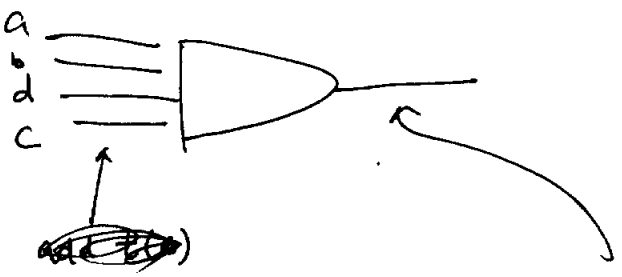
$2(0.1)(0.9) + 2(0.5)(0.5) + 2(0.5)(0.5)$

$= 0.18 + 1 = 1.18$

$\sum = 1.18 + 0.04875 + 0.090$

$= 1.31875$

$\therefore$ abd is a PPI



$1.18 + 2(0.9)(0.1)$

$= 1.36$

$P_1 = 0.05 \times 0.45$

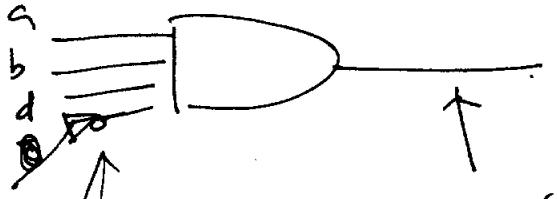
$= 0.0225$

$2 \cdot P_1 P_0$

$= 0.045$

$1.36 + 0.04 = 1.40$

Not a PPI



$1.18 + 2(0.9)(0.1) + 2(0.9)(0.1)$

$= 1.36 + 0.18$

$= 1.54$

$P_1 = (0.1)(0.5)(0.5)(0.1)$

$= 0.0025$

$2 \times (0.0025)(0.9975)$

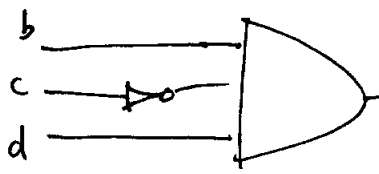
$= 0.005$

$1.54 + 0.005$

$= 1.545$

Not a PPI

b c'd



$P_1 = 0.5 \times 0.5 \times 0.1$
$= 0.025$

$P_{SW} = 2 \cdot 0.025 \cdot 0.975 = 0.049$

$2(0.5)(0.5) + 2(0.5)(0.5) + 2(0.9)(0.1) + 2(0.9)(0.1)$

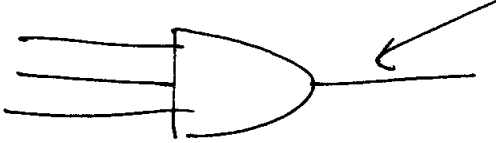
$= 1 + 0.36 = 1.36$

$+ 0.049$

1.409

NOT PPI

bcd



$P_1 = 0.5 \cdot 0.5 \cdot 0.9$
$= 0.225$

$P_{SW} = 2(0.225)(0.775)$
$= 0.35$

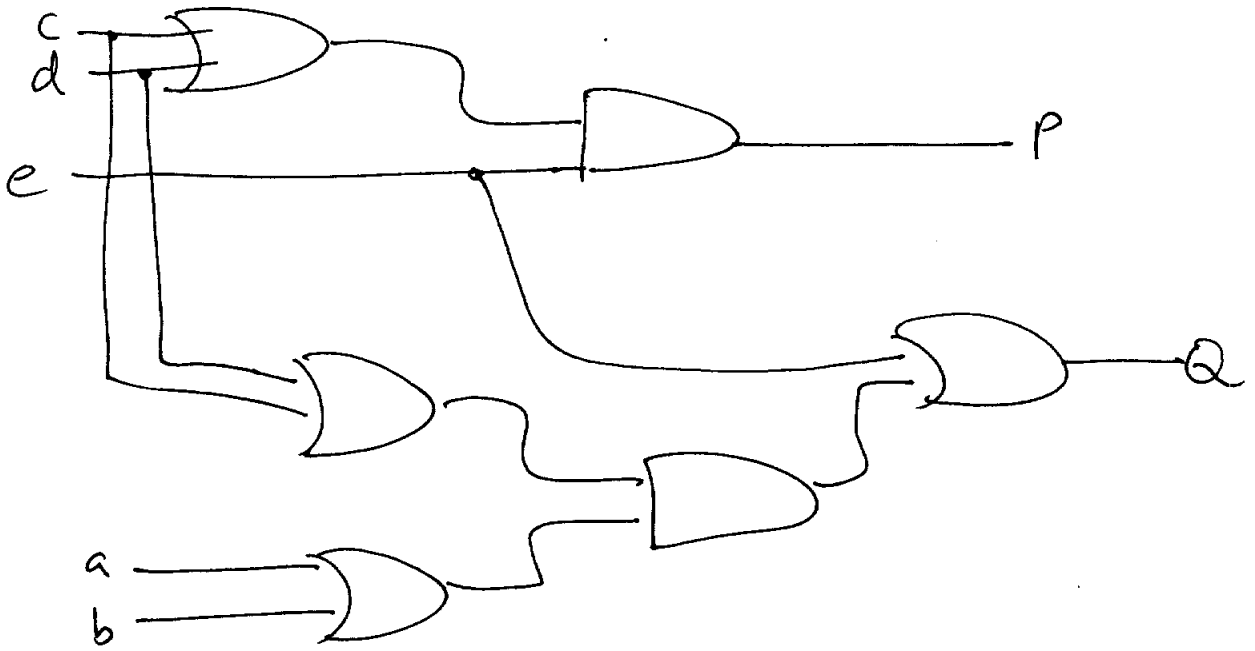
$2(0.5)(0.5) + 2(0.5)(0.5) + 2(0.1)(0.9)$

$= 1.18 + 0.35$

$= 1.53$ NOT < PPI

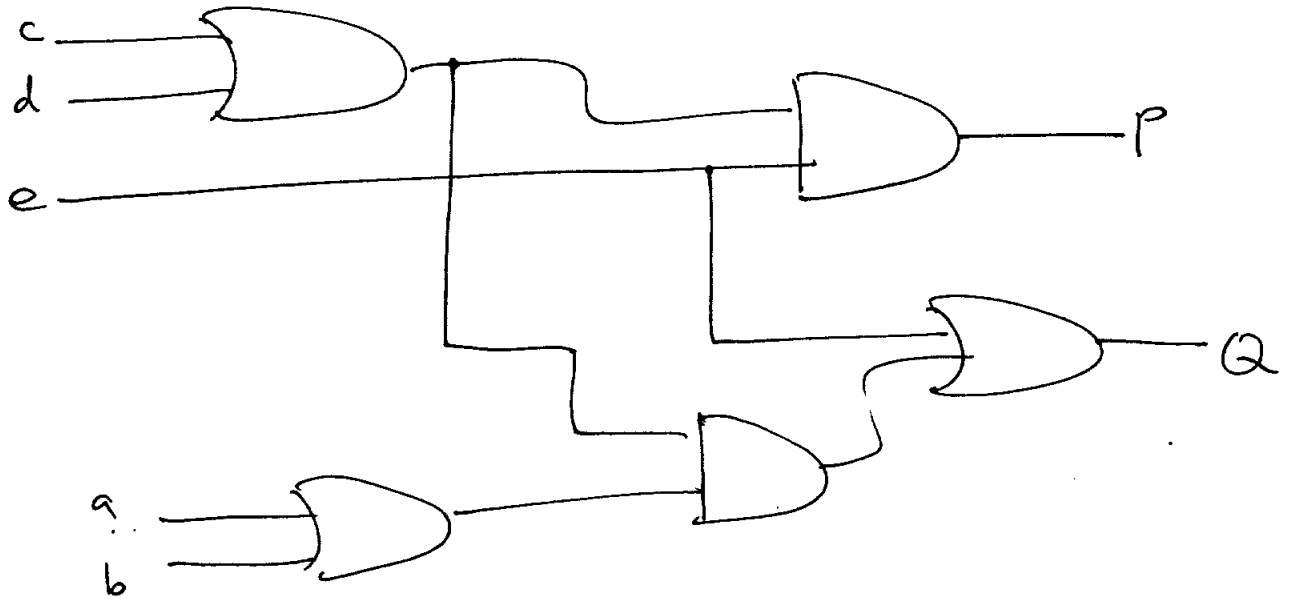
(6)

original ckt:



identify ctd as a kernel

new ckt



$$t(c) + t(d) + t(c \overset{c+d}{\otimes} d) - t(c+d)$$

$$= t(c) + t(d) \geq 0$$

$\therefore$ new design satisfies power.