Logic Synthesis – Part 2

Technology-Dependent Optimization

Prof. Chien-Nan Liu
Institute of Electronics
National Chiao-Tung Univ.

Tel: (03)5712121 ext:31211
E-mail: jimmyliu@nctu.edu.tw
http://www.ee.ncu.edu.tw/~jimmy

Courtesy: Prof. Jing-Yang Jou

Outline

- Synthesis overview
- RTL synthesis
- Two-level logic optimization
- Multi-level logic optimization

與製程無關

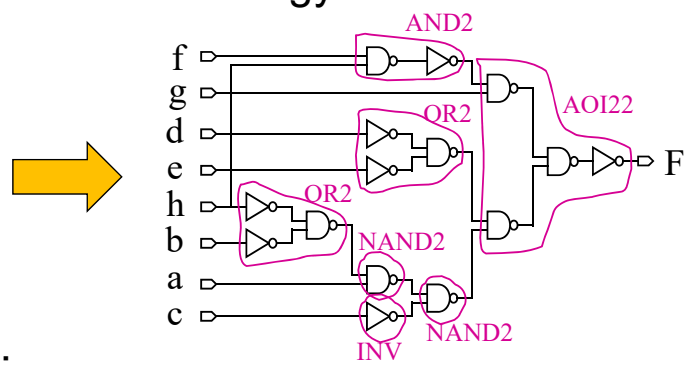
- Technology mapping
- Timing analysis
- Timing optimization
- Synthesis for low power

Technology Mapping

將Behavior Model 根據cell library中提供的gate轉為實際對應的電路，轉換後也要確保logic是對的

- Goal: implement an optimized Boolean network using a given library under specific technology

```
t1 = d + e;
t2 = b + h;
t3 = a t2 + c;
t4 = t1 t3 + f g h;
F = t4';
```



每個紅圈都是cell library 中的一個gate (cell)，我們需要找到最有效的轉換方式

- Library cells includes:
  - Combinational elements:
    - Single-output functions: AND, OR, AOI
    - Compound cells: adders, decoders
  - Sequential elements:
    - Registers, counters

通常一個cell library會有約300-400個cell，其中很多cell在邏輯上是相同的，只是具備不同的sizing，根據所需推力的不同來調用

如何挑選適當的standard cell來implement電路?

Major Approaches

- Rule-based systems: 制定Rule，根據不同的情況選擇對應的gate
  - Mimic designers' activity
  - Handle all types of cells

優點: 簡單快速，不管什麼情況都能應對
缺點: 不能保證是最佳解

- Heuristic algorithms: 使用Heuristic找到最佳近似解
  - Restricted to single-output combinational cells
  - DAGON approach 只能套用在single-output的combinational cell
- Most tools use a combination of both
  通常的tool都是兩種方法都採用

Rule-Based Library Binding

- Binding by stepwise transformations
- Data-base:
  - Set of patterns associated with best implementations
- Rules:
  - Select sub-network to be mapped
  - Handle high-fanout problems

使用rule-based應對multi-fanout的gate

- Advantages:
  - Applicable to all kinds of libraries

rule-base只是單純描述轉換方式，因此通常所有的library都通用

- Disadvantages:
  - Large rule data-base
    - Completeness issue
  - Data-base updates

如果rule寫得很完整，可能會造成檔案過大
但如果沒寫完整就可能導致某些特別的狀況無法被轉換

Heuristic Approach

- Find a partition of the given network such that each sub-network can be replaced by a library cell
- General approach:

選擇base function，類似元素週期表，使用各種元素就可以組合出我們想要描述的電路

  - Choose base function set for canonical representation
    - Ex: 2-input NAND and Inverter

選擇可以表示所有邏輯的邏輯閘-->NAND

  - Represent optimized network using base functions
    - Subject graph

將想要最佳化的目標電路使用base function表示 --> subject graph

  - Represent library cells using base functions
    - Pattern graph

將cell library中的standard cell使用base function表示 --> Pattern graph

  - Each pattern associated with a cost which is dependent on the optimization criteria

給予pattern graph中每個pattern一個cost，方便後續對應的時候計算cost

- Goal:

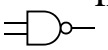
目標: 找到最低cost的對應方式(將subject graph用pattern組合出來)

  - Finding a minimal cost covering of a subject graph using pattern graphs

Example Pattern Graph (1/3)

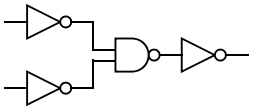
inv (1)

nand2 (1)



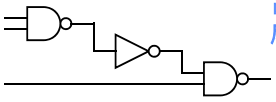
我們有的standard cell
其餘的東西都可以用這兩個cell組合出來
而組合出來的cost則需要對應實際layout大小才知道

nor2 (2)



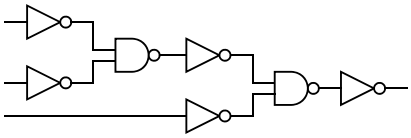
一樣先用OR-inv，再用DeMorgan's Rule換成NAND

nand3 (3)

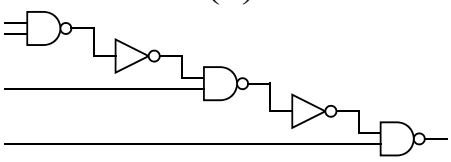


可以先用AND與inv組合出來，再把AND換成NAND

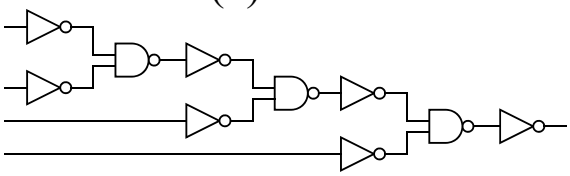
nor3 (3)



nand4 (4)

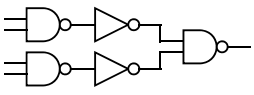


nor4 (4)

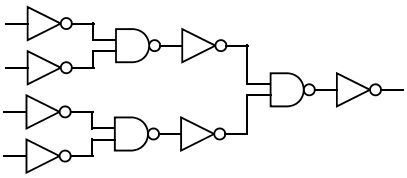


Example Pattern Graph (2/3)

nand4 (4)

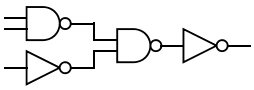


nor4 (4)

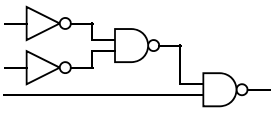


aoi21原本應該是長這樣:

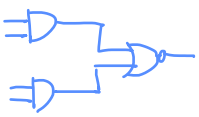
aoi21 (3)



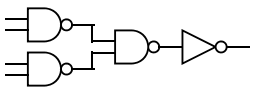
oai21 (3)



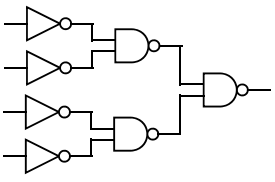
aoi22原本應該是長這樣:



aoi22 (4)

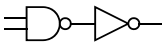


oai22 (4)

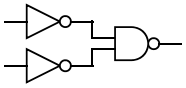


Example Pattern Graph (3/3)

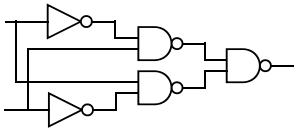
and2 (3)



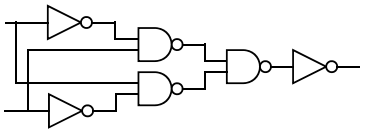
or2 (3)



xor (5)



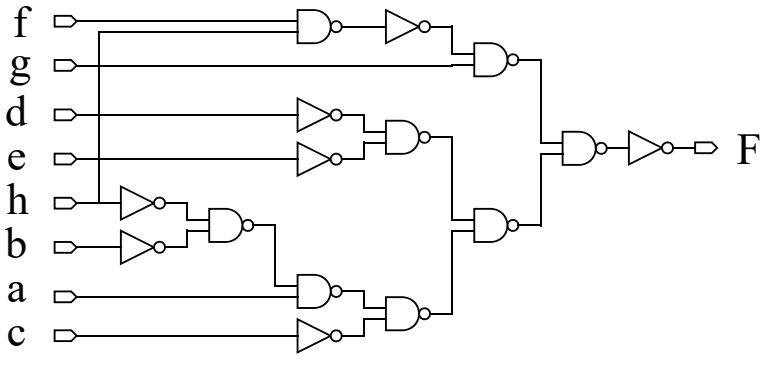
xnor (5)



Example Subject Graph

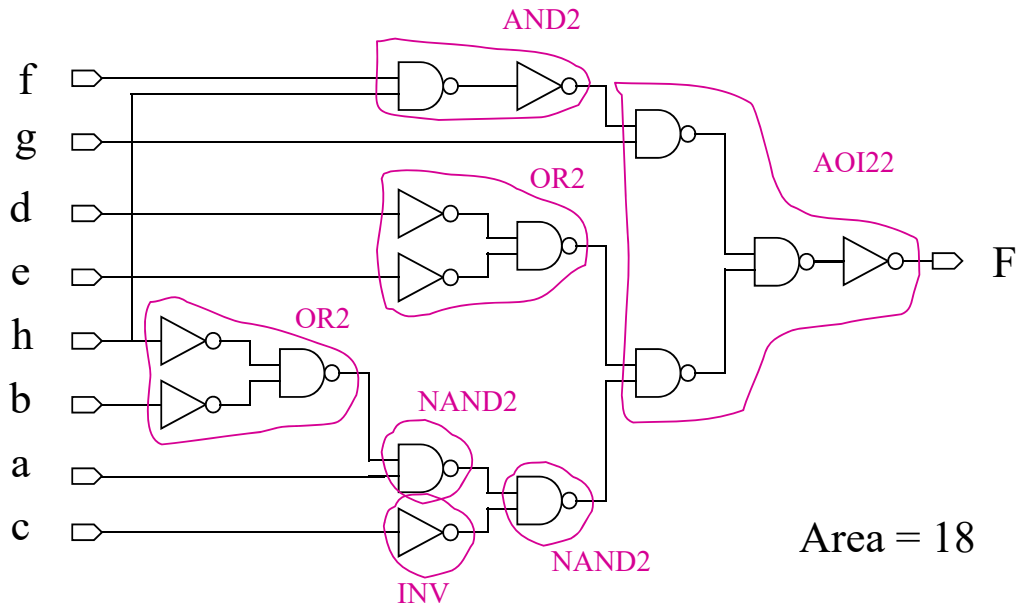
t1 = d + e;
t2 = b + h;
t3 = a t2 + c;
t4 = t1 t3 + f g h;
F = t4';

先將基本的電路畫出來，
然後全部換成NAND與inv
的組合



Sample Covers (1/2)

然後進行辨識，看那些東西合在一起
進行mapping可以使用最少的cost

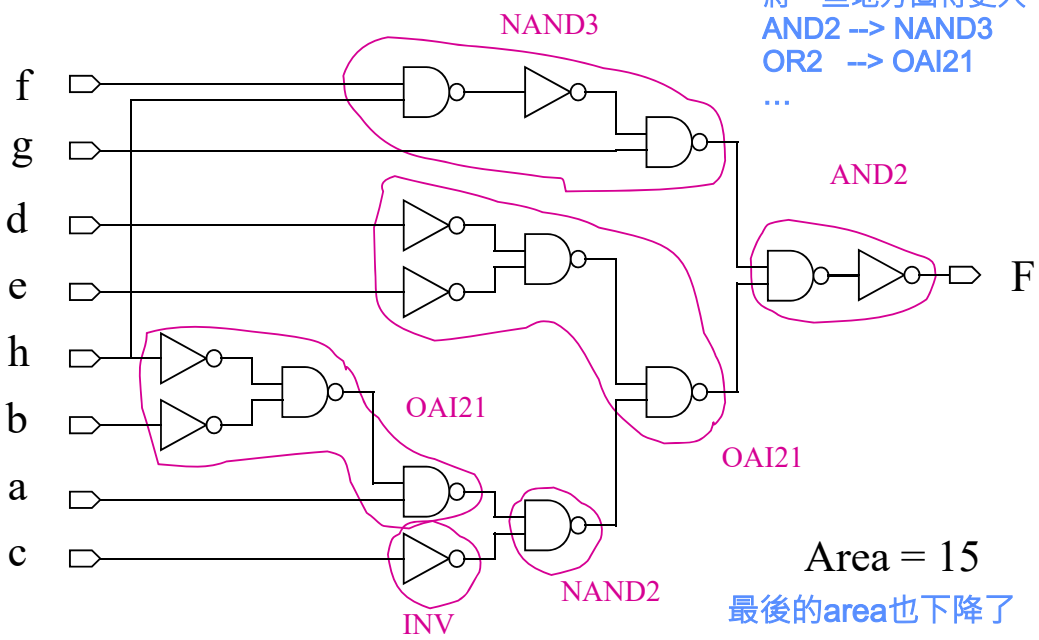


Area = 18

有一些東西無法group起來也沒關係，
因為NAND與inv本身都是standard cell

Sample Covers (2/2)

與上一張圖比較可以發現，這邊
將一些地方圈得更大，原本的
AND2 --> NAND3
OR2 --> OAI21
…

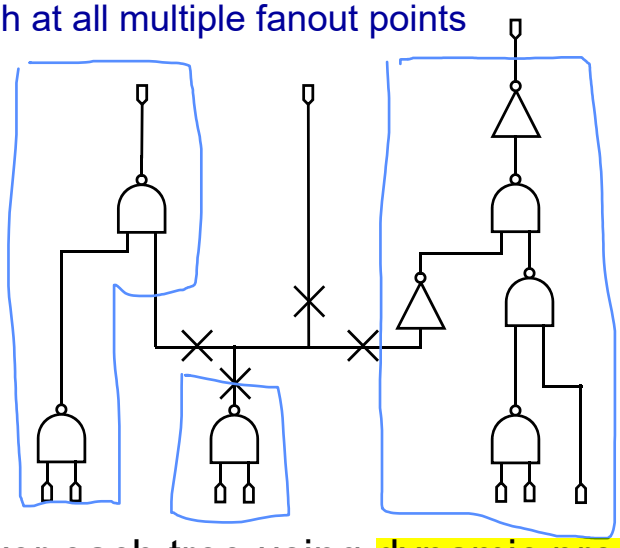


Area = 15

最後的area也下降了

DAGON Approach

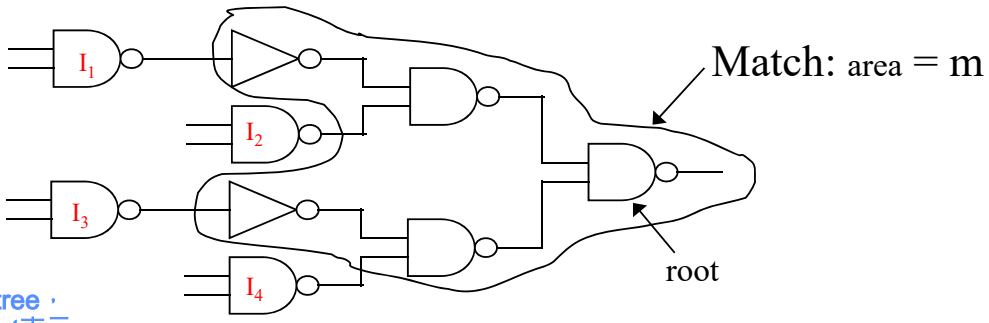
- Partition a subject graph into trees 如果電路中有multi-fanout的gate，DAGON就會將電路進行partition
  - Cut the graph at all multiple fanout points



- Optimally cover each tree using dynamic programming approach
- Piece the tree-covers into a cover for the subject graph

Dynamic Programming for Minimum Area

- Principle of optimality: optimal cover for the tree consists of a match at the root plus the optimal cover for the sub-tree starting at each input of the match



電路中會有很多切割好的subtree，我們可以將從root看進去的cost表示為recursive的形式

$$A(root) = m + A(I_1) + A(I_2) + A(I_3) + A(I_4)$$

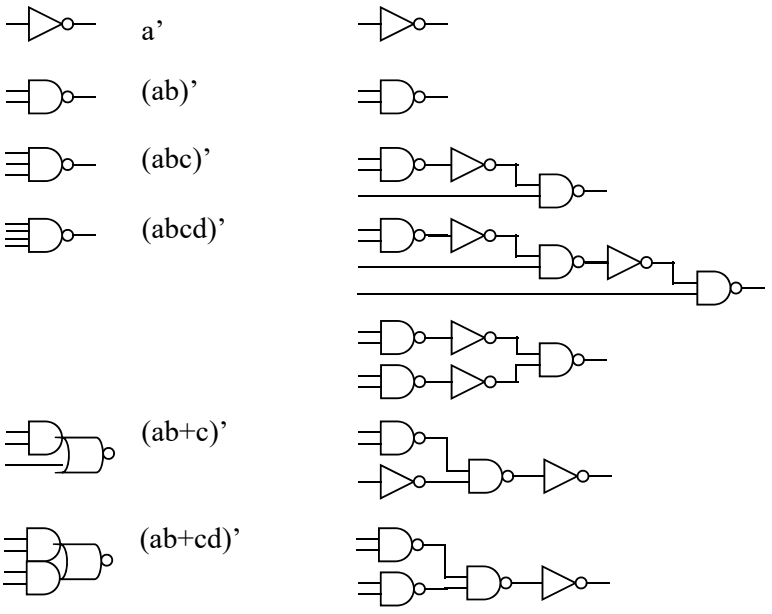
這樣就可以套用DP來解

$$\text{cost of a leaf} = 0$$

A Library Example

這個例子中所使用的library

INV	2	a’	
NAND2	3	(ab)’	
NAND3	4	(abc)’	
NAND4	5	(abcd)’	
AOI21	4	(ab+c)’	
AOI22	5	(ab+cd)’	

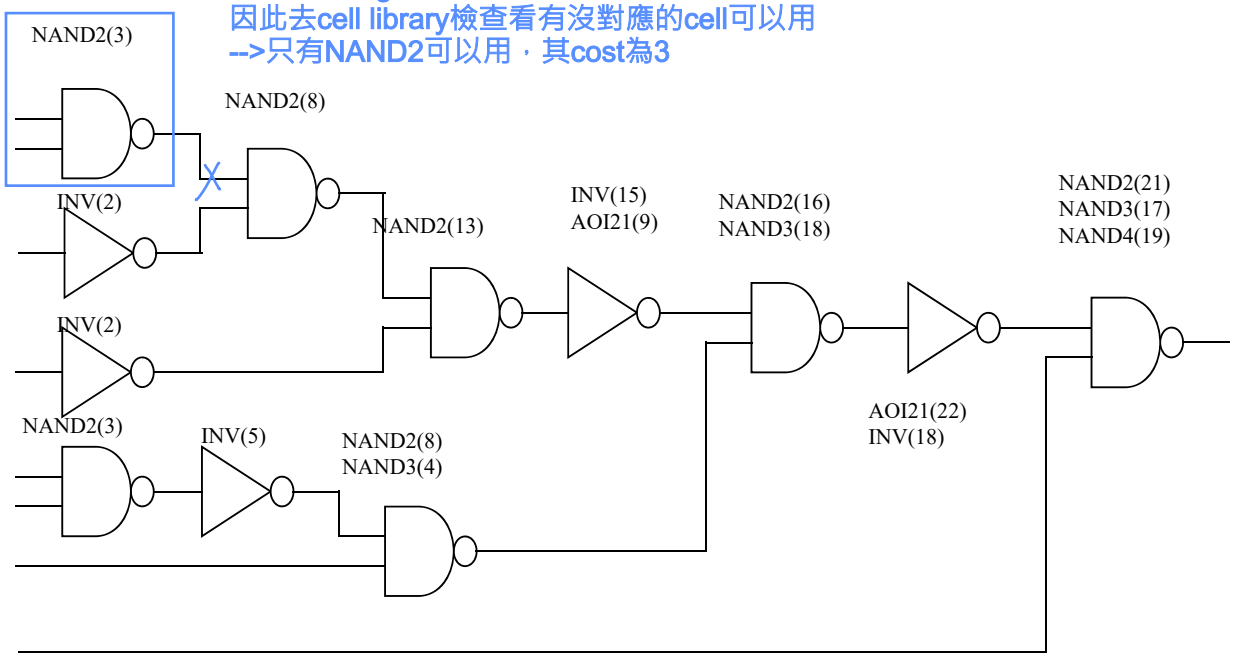


Library Element　　Canonical Form

Bottom-Up

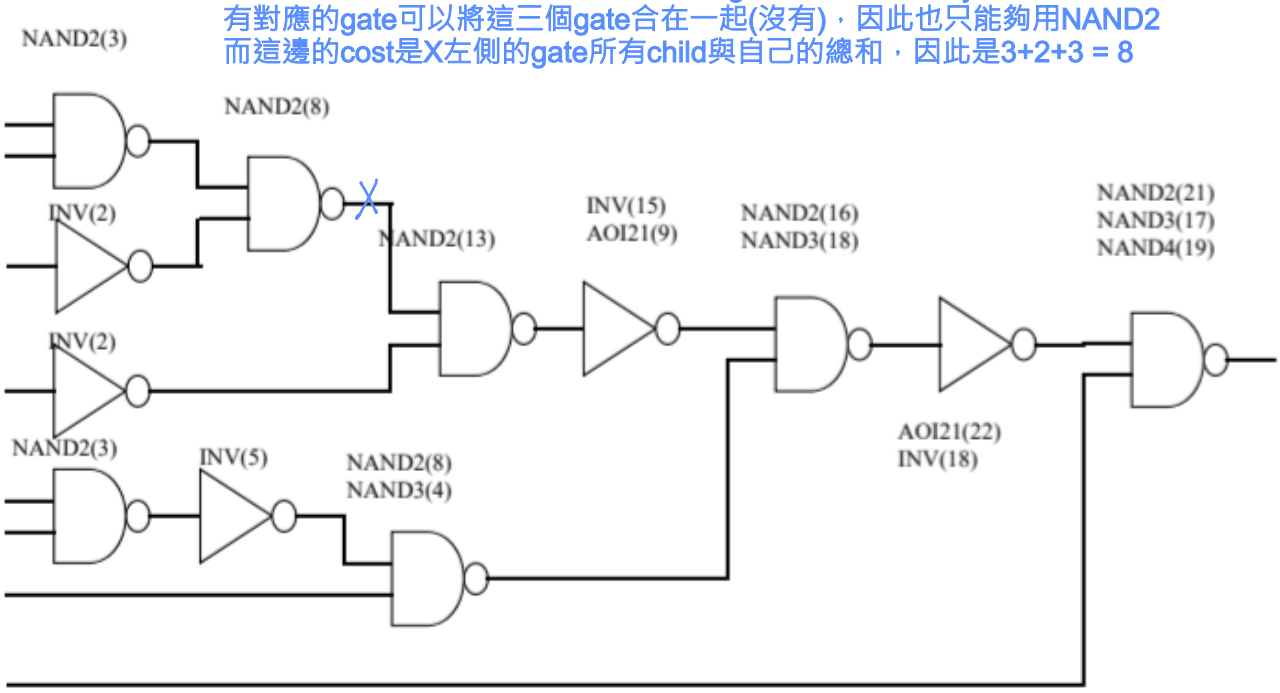
DAGON in Action

首先從這個gate開始向右走，由X往左看發現沒有東西，
因此去cell library檢查看有沒對應的cell可以用
-->只有NAND2可以用，其cost為3



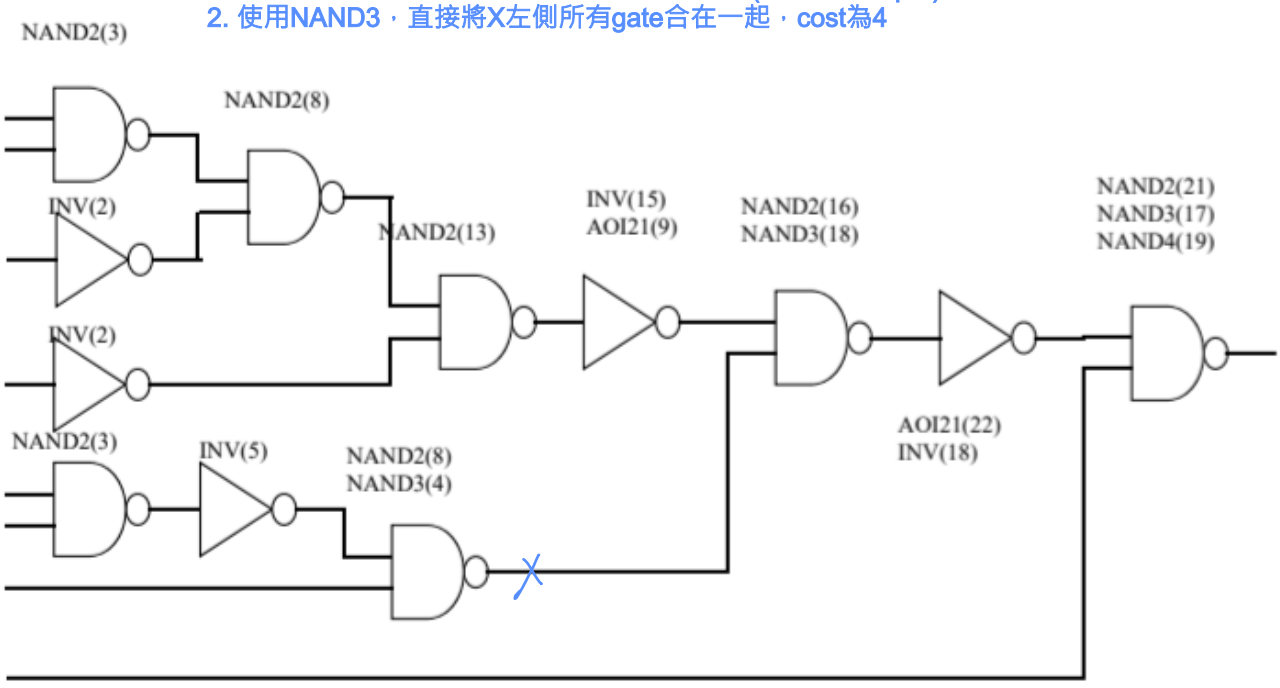
DAGON in Action

接下來向右走，從X處往左看，看到3個gate，去cell library尋找後發現沒有對應的gate可以將這三個gate合在一起(沒有)，因此也只能夠用NAND2
而這邊的cost是X左側的gate所有child與自己的總和，因此是3+2+3 = 8



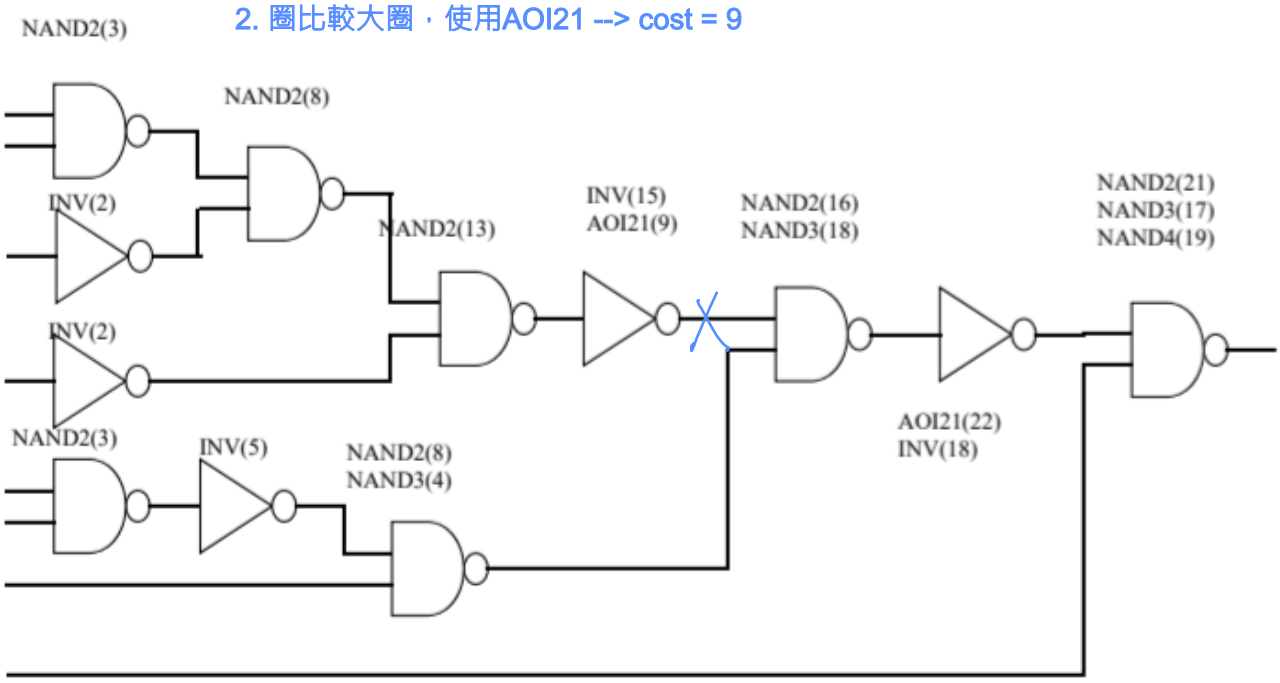
DAGON in Action

接下來走下面，由X往左看，發現有3個gate
現在有兩種選擇:
1. 使用NAND2，這樣他的cost需要將他的child(inv與一個input)與自己相加--> 5+0+3 = 8
2. 使用NAND3，直接將X左側所有gate合在一起，cost為4



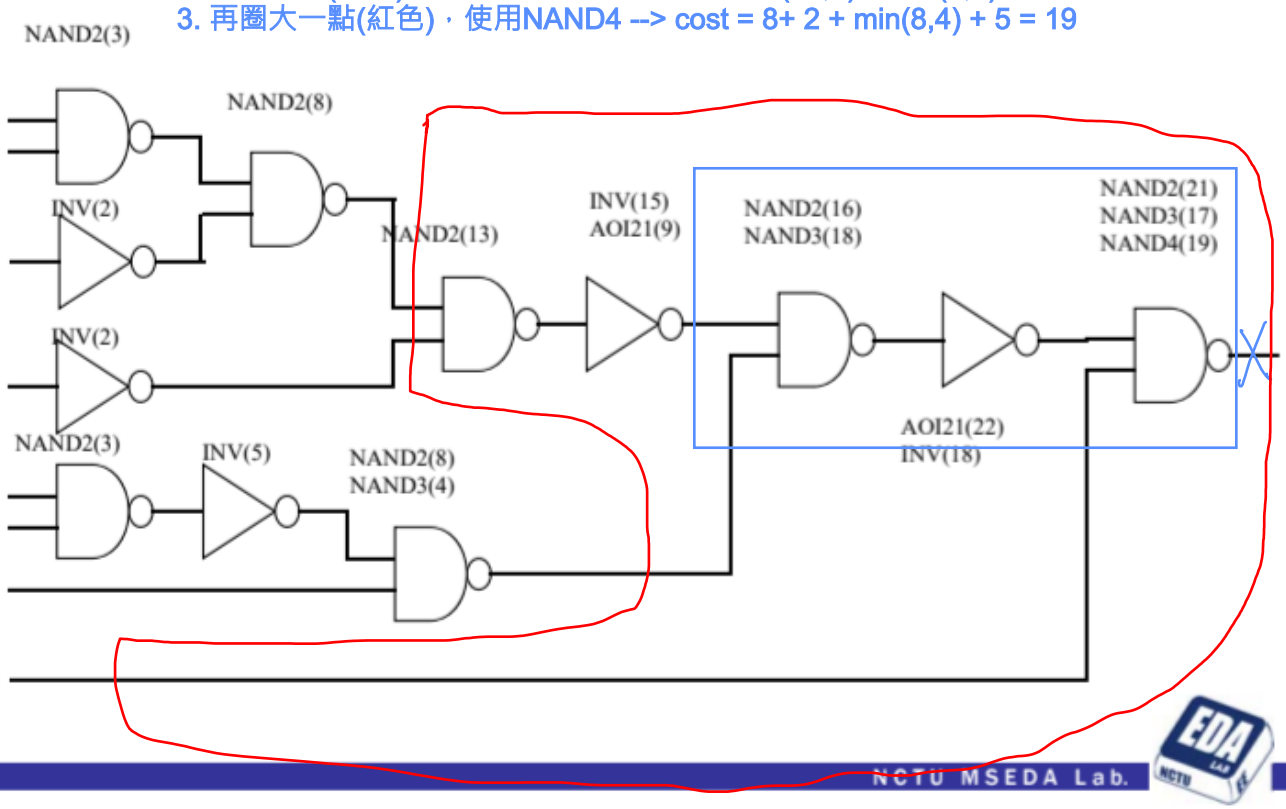
DAGON in Action

由X向左看，也可以發現有兩種組合
1. 自己一個-->使用inv --> cost = 13+2 = 15
2. 圈比較大圈，使用AOI21 --> cost = 9



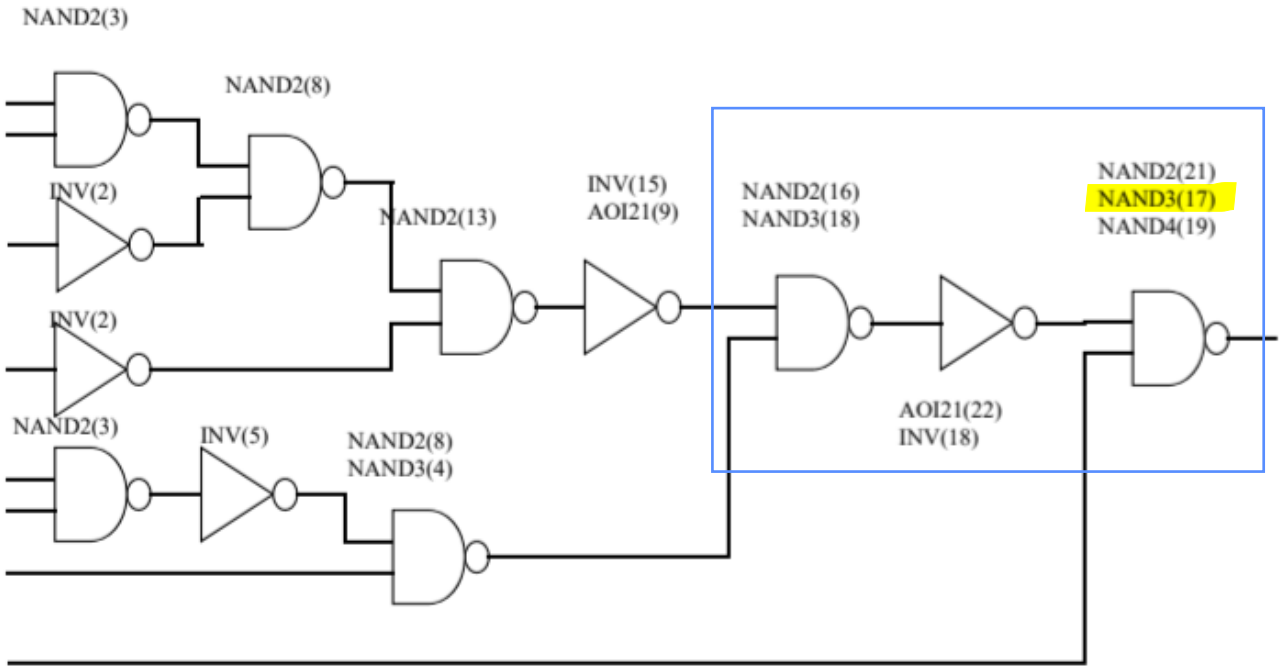
DAGON in Action

從output往回看，有三種組合
1. 自己使用NAND2 --> cost = min(22,18) + 3 = 21
2. 圈大一點(藍色)，使用NAND3 --> cost = min(15,9) + min(8,4) + 4 = 17
3. 再圈大一點(紅色)，使用NAND4 --> cost = 8+ 2 + min(8,4) + 5 = 19



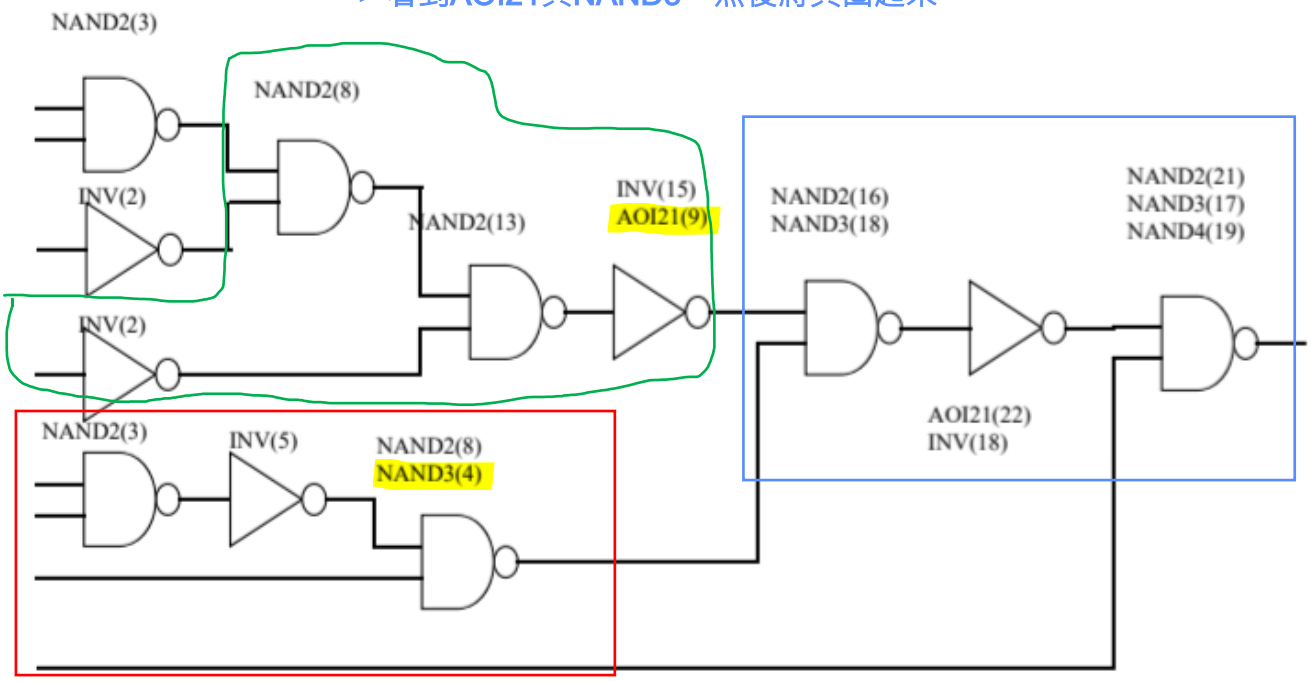
DAGON in Action

由output端往回trace，看到NAND3是最好的結果，
就先往回圈出NAND3



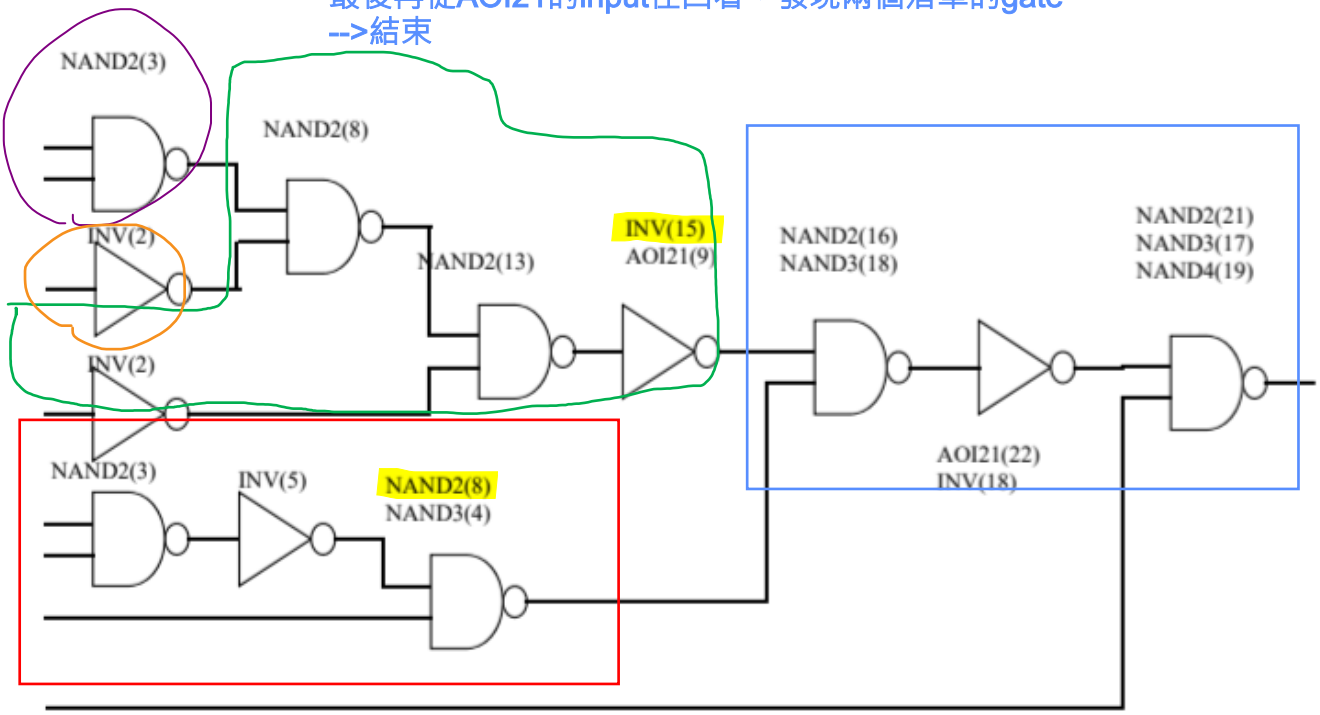
DAGON in Action

再從剛剛圈的NAND3的input往回trace，找最小的cost的組合
--> 看到AOI21與NAND3，然後將其圈起來



DAGON in Action

最後再從AOI21的input往回看，發現兩個落單的gate
-->結束



Features of DAGON

- Pros. of DAGON:
  - Strong algorithmic foundation
  - Linear time complexity
    - Efficient approximation to graph-covering problem
  - Given locally optimal matches in terms of both area and delay cost functions
  - Easily "portable" to new technologies 可以輕易地換成其他製程
- Cons. Of DAGON:
  - With only a local (to the tree) notion of timing 只能最佳化area，看不到loading與timing
    - Taking load values into account can improve the results
  - Can destroy structures of optimized networks 由於只能套用在single-fanout的電路，因此如果之前有做過resource sharing的優化(會導致multi-fanout)，會被DAGON切掉
    - Not desirable for well-structured circuits
  - Inability to handle non-tree library elements (XOR/XNOR)
  - Poor inverter allocation 如果library中有non-tree的standard cell，會導致DAGON找不到這些cell

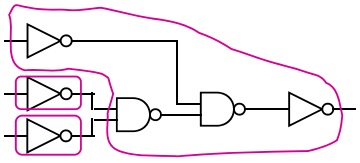
有時候加一些inverter可以使結果更好，但DAGON無法做到

可以透過先將那些XOR/XNOR先轉換好，再將剩餘的電路使用DAGON即可

Inverter Allocation

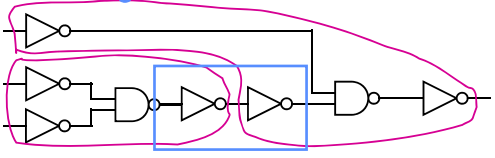
- Add a pair of inverters for each wire in the subject graph
- Add a pattern of a wire that matches two inverters with zero cost
- Effect: may further improve the solution

原本的電路:



2 INV
1 AOI21

在中間加入兩個inverter (相當於插入一個buffer)後，可以發現只需要兩個NOR2就完成原本需要三個gate才能完成的邏輯



2 NOR2

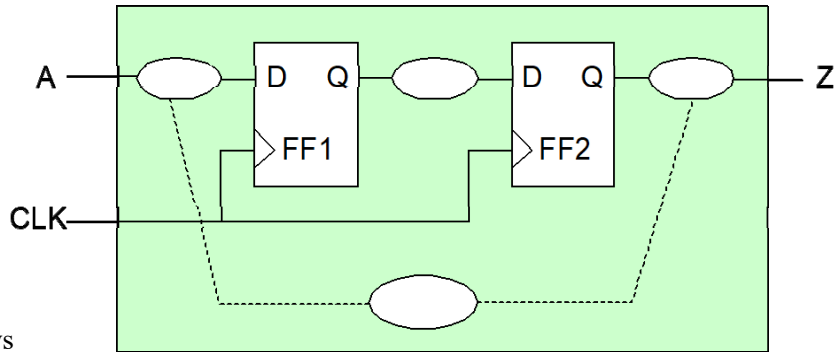
Outline

- Synthesis overview
- RTL synthesis
- Two-level logic optimization
- Multi-level logic optimization
- Technology mapping
- Timing analysis
- Timing optimization
- Synthesis for low power

Functionality vs. Performance

- Functionality: correctly implements specified function
  - Mostly checked by simulation (Logic, RTL, Behavior)
- Performance: correctly implements specified function at ***specified speed***
  - Can be checked by simulation, but …
- **Static timing analysis** (STA) is used to determine if a circuit meets timing constraints without simulation
  - Also a part of synthesis engine to evaluate the performance

不需要執行simulation，直接分析電路是否可以符合timing constraint



Reference :Synopsys

Performance Verification

- Simulation-based approaches
  - Pattern-dependent
  - Incomplete coverage
  - Slow
  - Accurate
  - Impractical for large designs

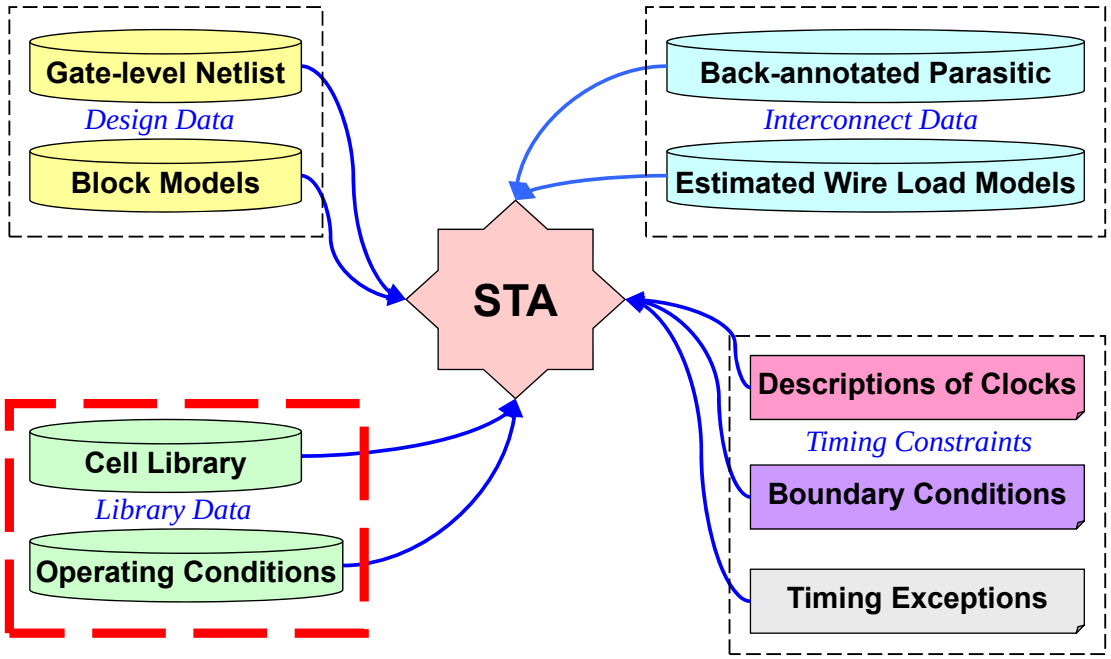
simulation-based的結果會根據input pattern的不同而有所改變，因此我們還需要額外去設計input pattern以提高coverage
對於大型design來說，使用simulation來驗證會花費大量的時間(因為需要各種不同的pattern去hit corner case)
但是simulation-base的結果會較為準確

- Static timing analysis techniques
  - Functionality assumed correct
  - Pattern-independent timing check
  - Complete coverage
  - Fast
  - Maybe inaccurate because of **false paths**

STA會去分析電路中所有的timimg path
因為不需要跑大量pattern，所以較快
只是STA就無法分析functionality是否正確

有些timing path在電路實際運行時其實根本不會發生，但是STA並不能知道這件事

Data Preparation for STA

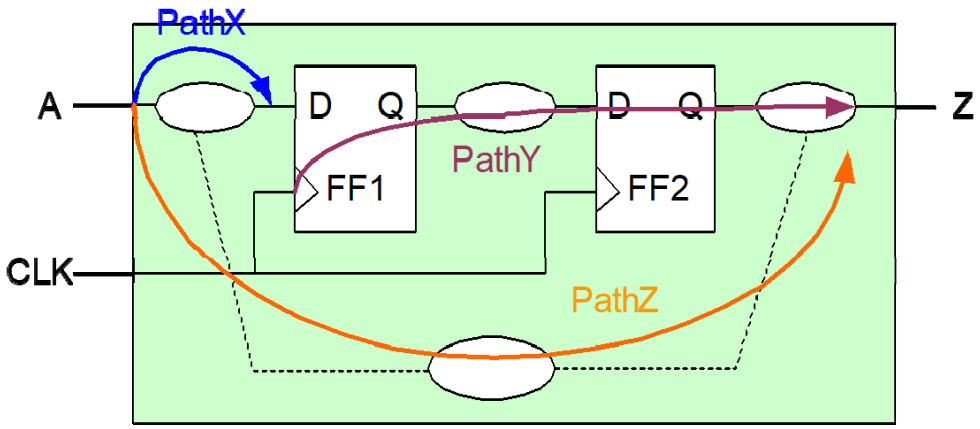


3 Steps in Static Timing Analysis

1. Design is broken down into sets of timing paths
2. Delay of each path is calculated
3. Path delay are checked to see if timing constraints have been meet

STA分析的最小單位就是path

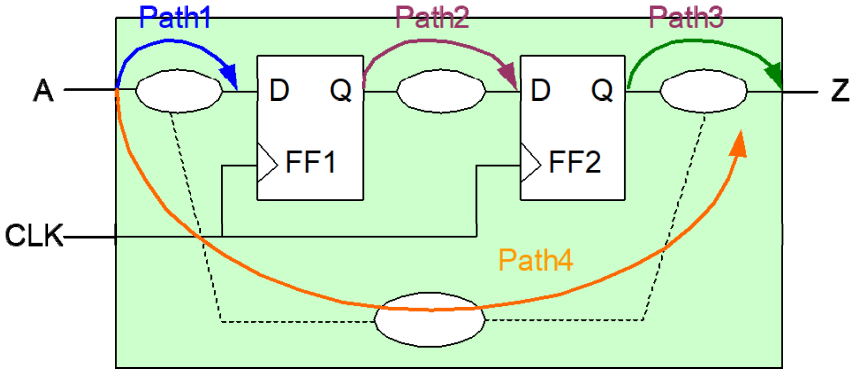
檢查每條path是否都滿足timing constraint
沒通過的path會寫成report



Reference :Synopsys

Timing Paths

- There are 4 types of timing path
  - Input port to data pin of FF (path 1) die input pin連結FF
  - Clock pin of FF to data pin of FF (path 2) tpcq+tpd
  - Clock pin of FF to output port (path 3) FF連結die output pin
  - Input port to output port (path 4) input pin經過combinational logic 後直接連到output pin

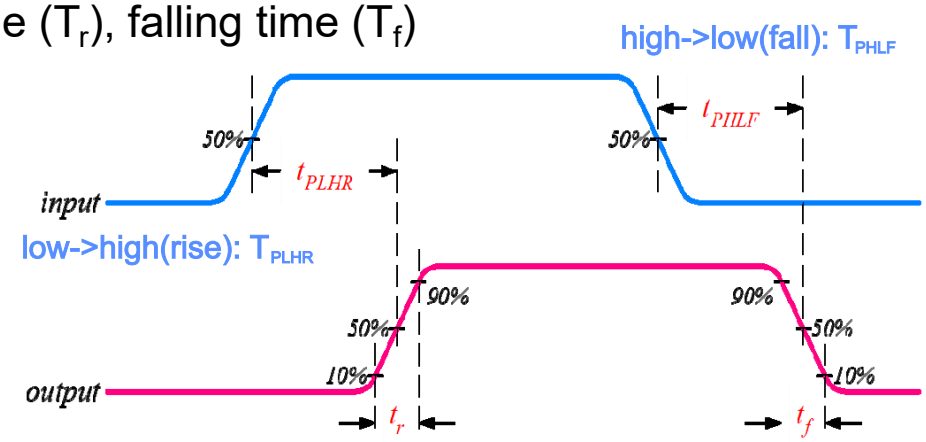


Stat point :
  input port
  Clock pin of FF
End point
  output port
  data pin of FF

Reference :Synopsys

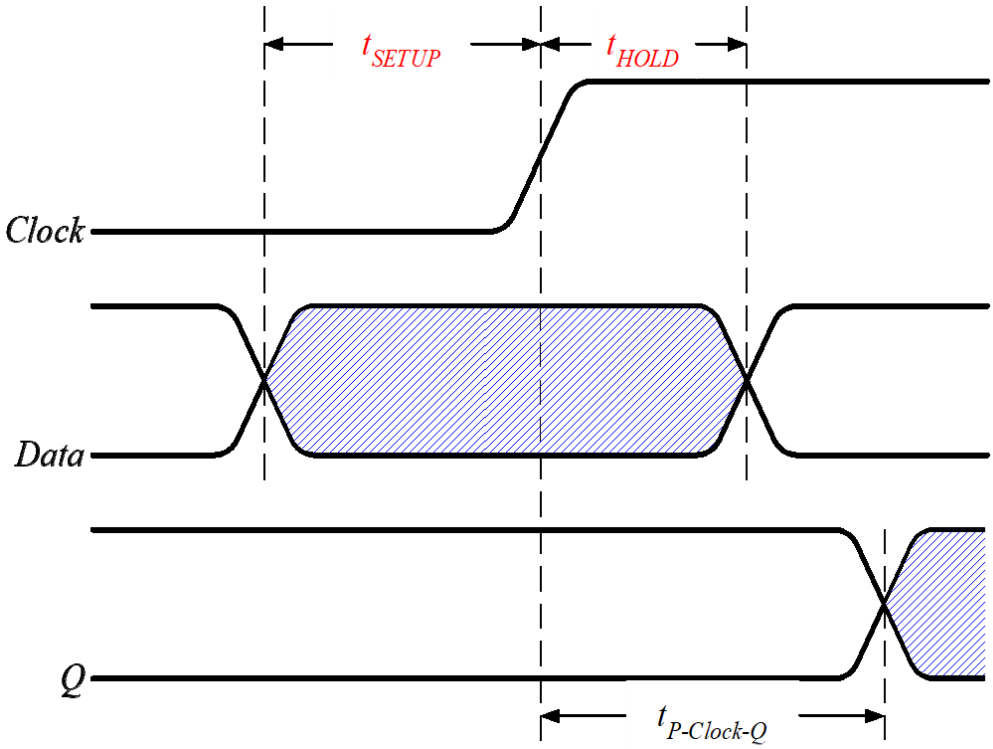
Definition of Timing Parameters

- Combinational circuits: propagation delay (T_{pLH}, T_{pHL}), rising time (T_r), falling time (T_f)



- Flip-Flop:
  - **Setup Time**: The length of time that data must stabilize before the clock transition → check maximum delay path
  - **Hold Time**: The length of time that data must remain stable at the input pin after the active clock transition → check min-delay path

Setup and Hold Time



Delay Model at Logic Level

1. unit delay model 直接假設每個gate的delay為1
   - Assign a delay of 1 to ***each gate***
2. unit fanout delay model 修正unit delay model，額外增加fanout對於delay的影響
   - Incorporate an ***additional*** delay for ***each fanout***
3. library delay model 現在最常用的，由cell library的製造者提供
   - Use delay data in the library to provide more accurate delay value
   - May use linear or non-linear (tabular) models
4. Post layout delay model 在model中加入寄生電容(parasitic capacitance)
   - Calculated based on known net parasitic data
   - Estimate cell delay based on SDF (standard delay format) files

Linear Delay Model

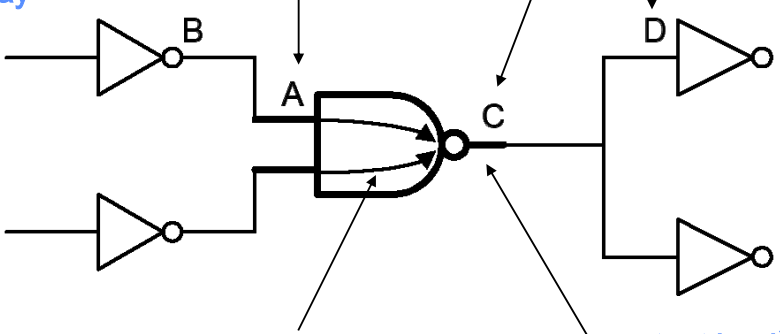
$$Delay = Dslope + Dintrinsic + Dtransition + Dwire$$

Ds : Slope delay : delay at input A caused by the transition delay at B

訊號不是理想的瞬間從0->1，因此斜率的大小會影響delay

導線的delay

Dw : Wire delay : time from state transition at C to state transition at D



D_I : Intrinsic delay : incurred from cell input to cell output

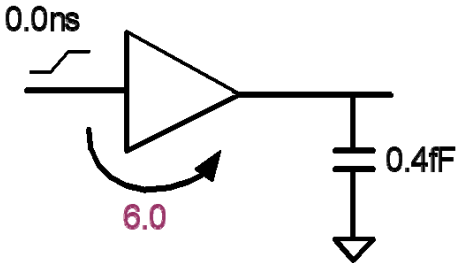
就是gate本身的delay

output load造成的delay

D_T : Transition delay : output pin loading, output pin drive

Tabular Delay Model

- Delay values are obtained by a look-up table
  - Two-dimensional table of delays (m by n)
    - with respect to input slope (m) and total output capacitance (n)
  - One dimensional table model for output slope (n)
    - with respect to total output capacitance (n)
  - Each value in the table is obtained by real measurement



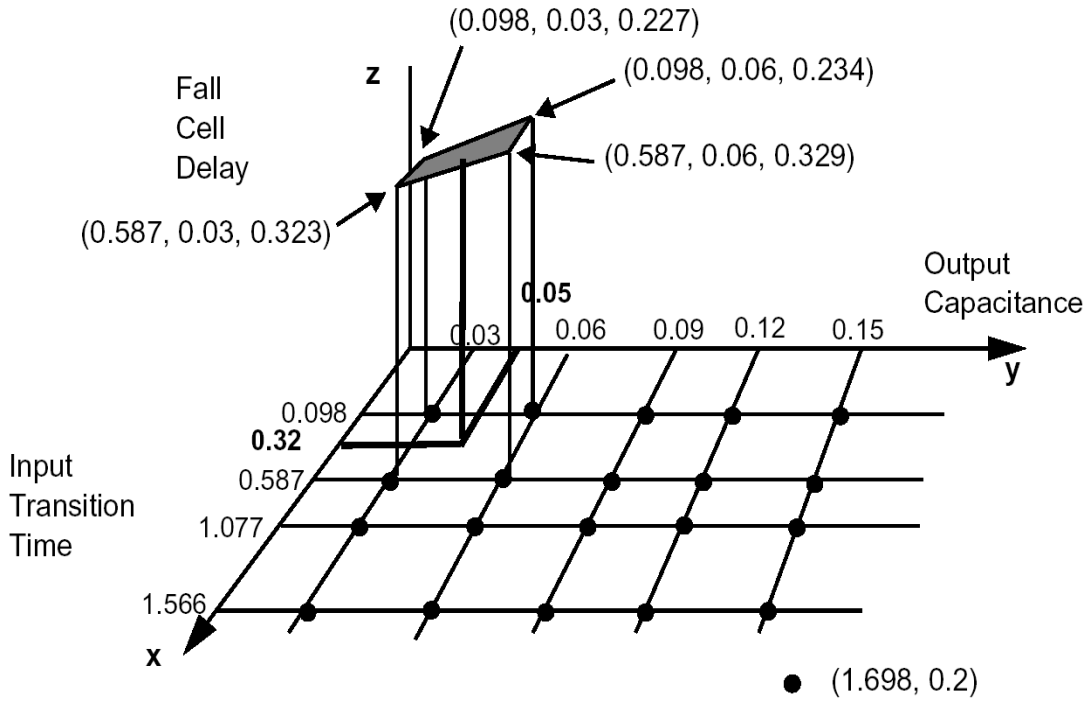
Total Output Load (fF)

Input Transition (ns)	0.2	0.3	0.4	0.5
0	3	4.5	6	7
0.1	5	8	10.7	13

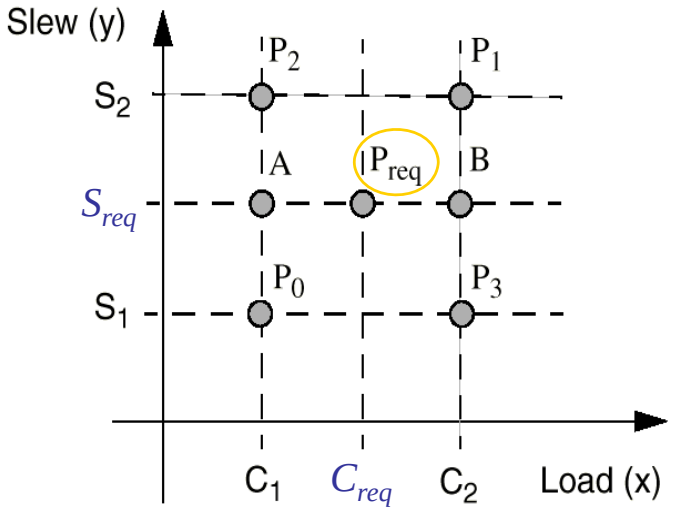
Cell Delay (ps)

- Can be more precise than linear delay model
  - table size↑ ➔ accuracy ↑ table建得越大，就越能涵蓋各種情況，但是占用的間也會很大
- Require more space to store the table

Illustration of Delay Calculation



2-D Table Interpolation



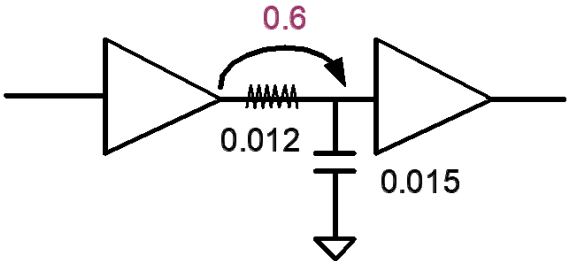
$$A = P_0 + \left(\frac{P_2 - P_0}{S_2 - S_1}\right)(S_{req} - S_1)$$

$$B = P_3 + \left(\frac{P_1 - P_3}{S_2 - S_1}\right)(S_{req} - S_1)$$

$$P_{req} = A + \left(\frac{B - A}{C_2 - C_1}\right)(C_{req} - C_1)$$

Net Delay Estimation

- Post-layout net parasitic data is extracted
  - Estimate net delay based on SDF
- Pre-layout net parasitic data cannot be accurately calculated 因為不知道實際走線的情況
  - Estimates net parasitic based on a wireload model
  - A wireload model is a set of tables
    - net fanout vs load
    - net fanout vs resistance
    - net fanout vs area

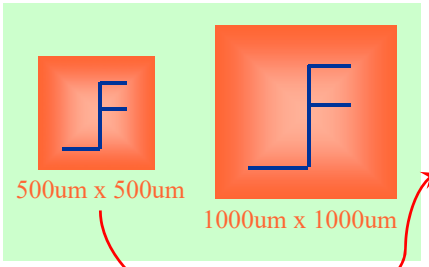


Net Fanout	Load (fF)	Resistance (mΩ)
1	0.015	0.012
2	0.030	0.016
3	0.045	0.020
4	0.060	0.024

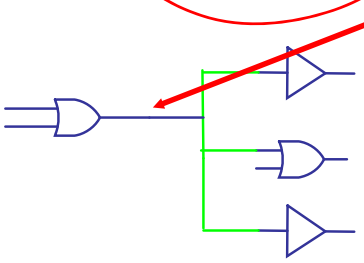
Reference :Synopsys

Wireload Model

- Determine wireload based on chip size
  → Very inaccurate!




```
wire_load ("500") (
resistance : 3.0                 /* R per unit length*/
capacitance: 1.3                 /* C per unit length */
area: 0.04                       /* area per unit length */
slop: 0.15                       /* extrapolation slope*/
fanout_length ( 1 , 2.1 )        /* fanout-length pairs */
fanout_length ( 2 , 2.5)
fanout_length ( 3 , 2.8)
fanout_length ( 4 , 3.3)
```



C_{wire} = (fanout=3, length =2.8) x capacitance coefficient (1.3) = 3.64 load units

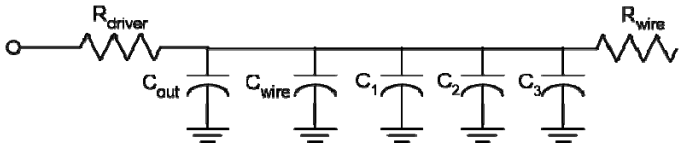
R_{wire} = (fanout=3, length =2.8) x resistance coefficient (3.0) = 8.4 resistance units

$Area_{Net}$ = (fanout=3, length =2.8) x area coefficient (0.04) = 0.112 net area units

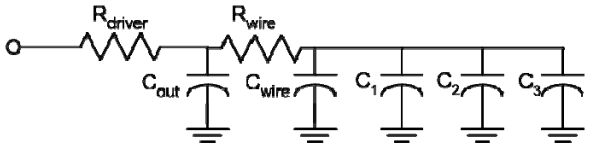
Interconnect Models

- Accurate net delay depends on the wire length
  - No such information at early design stage

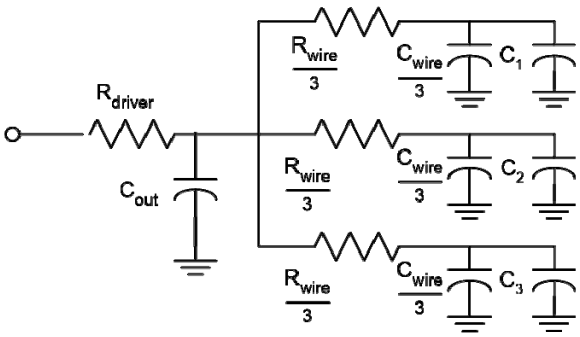
Best-Case RC Tree



Worst-Case RC Tree



Balanced Case RC Tree

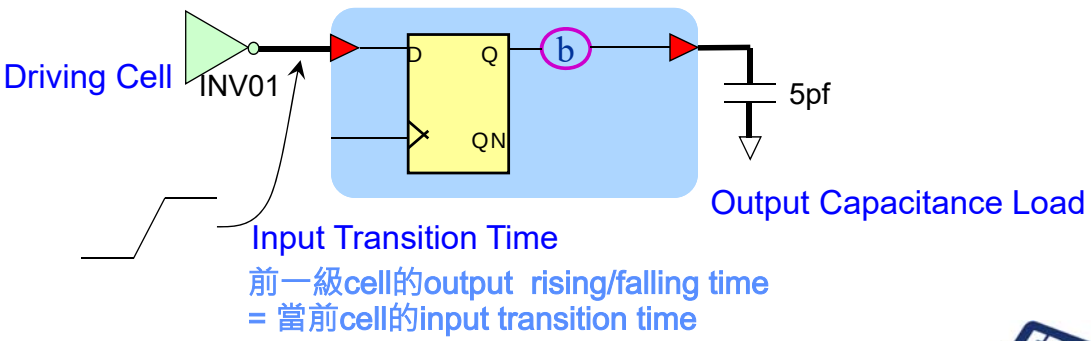


Interconnect Data

- Estimated delay information for nets based on a wire load model is used before P&R
- Back-annotated (Actual) delay information based on the P&R result is often described in the form of
  - SDF (timing information) – Standard Delay Format
    - SDF triplet: (min:typ:max) .sdf中會提供delay的range(min:typ:max)
  - RSPF – Reduced Standard Parasitic Format
  - DSPF – Detailed Standard Parasitic Format
  - SPEF – Standard Parasitic Exchange Format
    - SPEF also has syntax that allows the modeling of capacitance between different nets, so it is used in the crosstalk analysis

Boundary Conditions

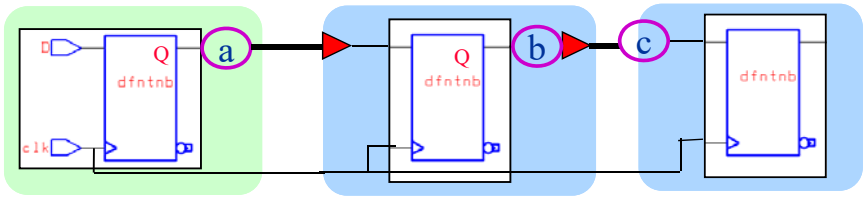
- Input driving cell
- Input transition time
- Output capacitance load
- Input delay (or Arrival Time)
- Output delay (or Required Time)



Input & Output Delay

- An input delay is the specification of an arrival time at an input port relative to a clock edge
- An output delay represents an external timing path from an output or inout port to a register

Input delay = $Delay_{clk-Q}$ + a



Input Block My Design Output Block

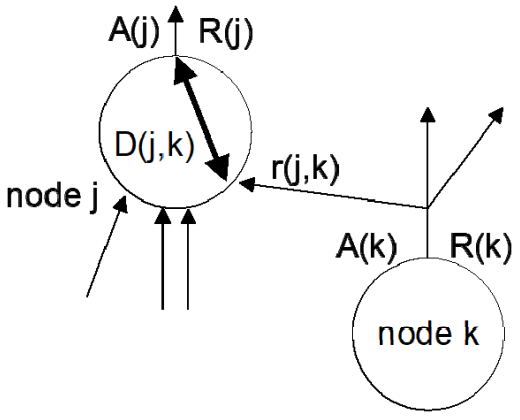
Output delay = c

Arrival Time and Required Time

- arrival time : calculated from input to output 完成的時間
- required time : calculated from output to input deadline
- slack = required time - arrival time slack就是完成的時間距離deadline還有多久

A(j): arrival time of signal j
R(k): required time or for signal k
S(k): slack of signal k
D(j,k): delay of node j from input k

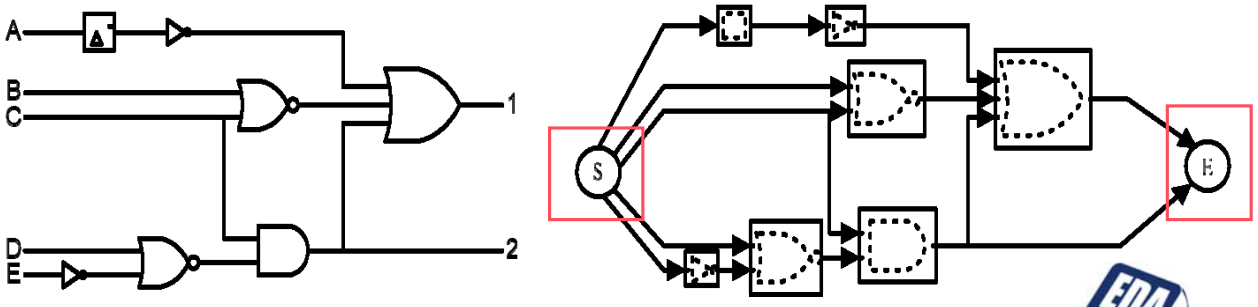
$A(j) = \max_{k \in FI(j)} [A(k) + D(j,k)]$
$r(j,k) = R(j) - D(j,k)$
$R(k) = \min_{j \in FO(k)} [r(j,k)]$
$S(k) = R(k) - A(k)$



Delay Graph

- Replace logic gates with delay blocks
  - Add start (S) and end (E) blocks
- The delay graph is G = (V, E)
  - Vertices : V ={ S, 1, 2, … , | V | -2, E } → logic blocks
  - Directed edges : E = (*u* ,*v*) *u, v* ∈V → signal flow
- Adjacency list : *Adj[u]* = (v_1, …, v_k)
  - (*u* ,v_i) ∈E *i* = *1* , …, *k*
- Number of input arcs to *u*: PredCount[u]

加入pseudo start/end point方便計算

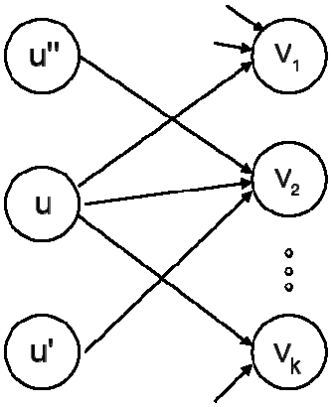


Longest and Shortest Path

- If we visit vertices in precedence order, the following code will need executing only once for each *u*

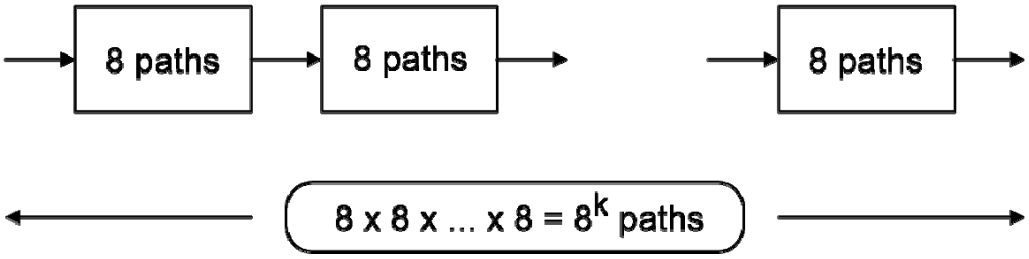
Update Successors[u]

```
1  for each vertex v ∈ Adj[u] do
2      if  A[v] < A[u] + Δ[u]   // longest
3          then A[v] ← A[u] + Δ [u]
4          LP[v] ← u  fi
5      if  a[v] >a[u] + δ[u]   // shortest
6          then a[v] ← a[u] + δ[u]
7          SP[v] ← u  fi
```



Path Enumeration

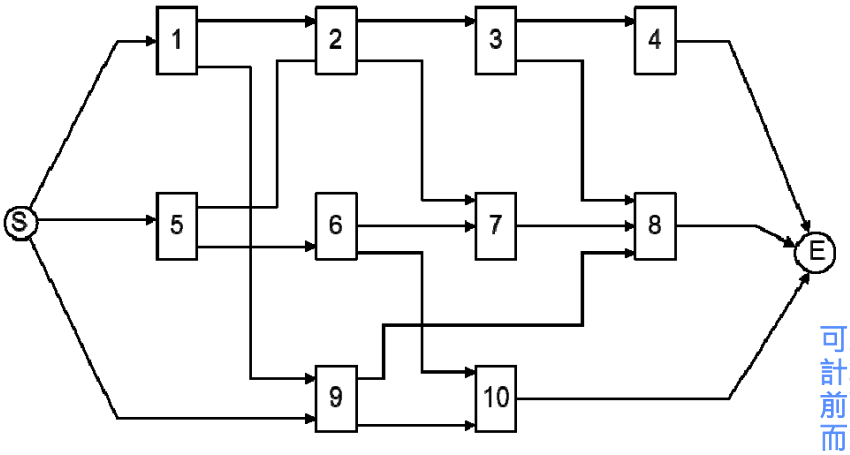
- Exhaustive enumeration can have exponential complexity
- Consider k copies of the example DG :



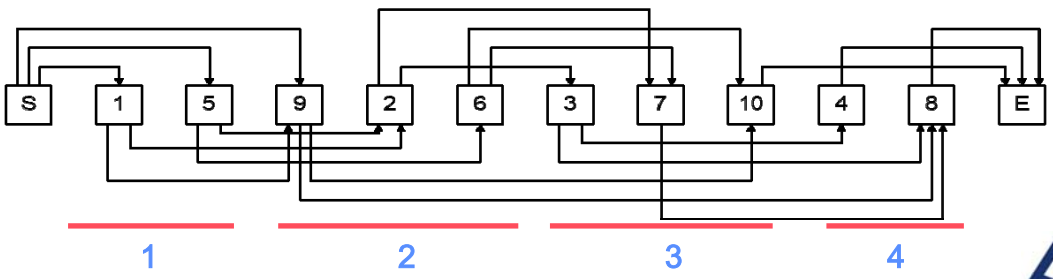
- Topological sort is proposed to solve this problem
  - Linear ordering of all the vertices in the DG such that if $(u, v) \in$ DG the u appear before v in the ordering

實際上不用全部的組合都試，我們只需去驗證真正具有先後關係的電路就好-->topological sort

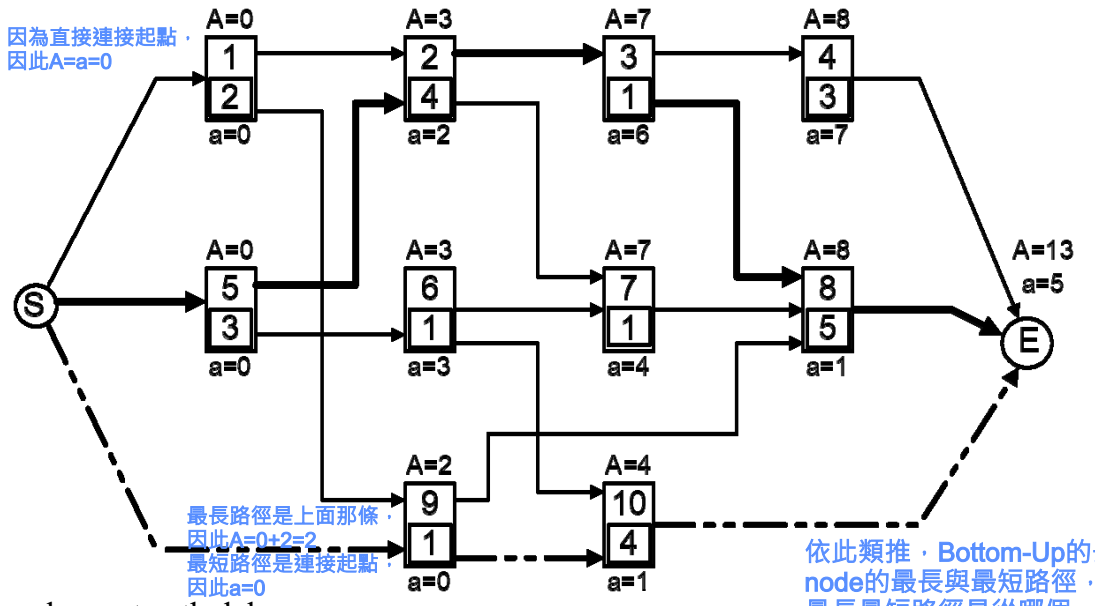
Delay Graph and Topological Sort



可以發現1, 5是最先可以被計算delay的node，因為他們前面沒有需要等待的node
而第二批可以被計算的就是9, 2, 6，依此類推



Delay Calculation



A=3 → longest path delay

2 → node number

4 → gate delay

a=2 → shortest path delay

P.S: The longest delay and shortest delay of each gate are assumed to be the same.

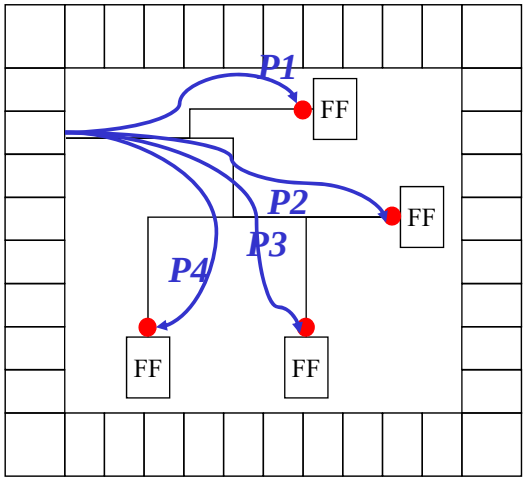
Timing Report

Point	Fanout	Incr	Path
.....			
U19/Z (INB)		0.38	50.76 r
n397 (net)	8	0.00	50.76 r
data_tri[5]/Z (BTD)		0.37	51.13 f
data[5] (net)	8	0.00	51.13 f
data[5] (inout)		0.00	51.13 f
data arrival time			51.13
clock clk (rise edge)		100.00	100.00
clock network delay (ideal)		0.00	100.00
clock uncertainty		-0.50	99.50
output external delay		-20.00	79.50
data required time			79.50
data required time			79.50
data arrival time			-51.13
slack (MET)			28.37

Meet timing !!

Clock Uncertainty Definition

- The maximum difference between the arrival of clock signals at sequential cells in one clock domain or between domains

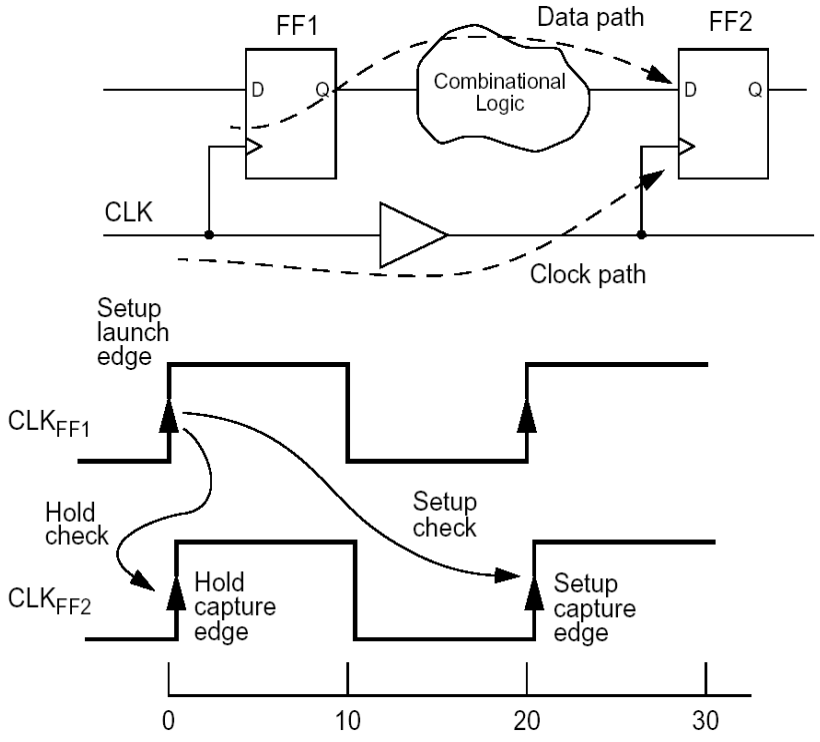


clk傳至各個FF的時間並不是一樣的，因此會取最快與最慢到達之間的時間差作為clk uncertainty，確保最慢接收到的FF一樣能正常運作

```
Arrival(P1) = 0.5ns 最快
Arrival(P2) = 1ns
Arrival(P3) = 1.2ns
Arrival(P4) = 1.3ns 最慢
```

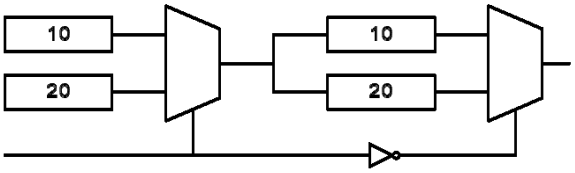
uncertainty
= 1.3 – 0.5
= 0.8ns

Setup and Hold Checking for FFs



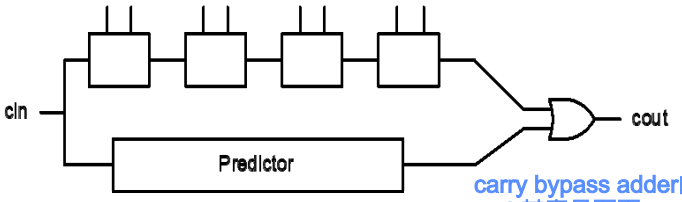
The False Path Problem

- Pattern-independent analysis calculates the delay of logic gates without regard to their function 由於STA並沒有實際輸入pattern進行測試，因此有些估計的path其實根本不會發生，導致估計過於保守
  - Can result in delays that are unnecessarily conservative
- Some paths are never sensitized in operation → **false path**
- Checking for false paths requires the function of the gates to be considered
  - Can approach the complexity of logic simulation or test generation
- Classical examples:



* Longest true path has delay 30

可以看到左邊兩個mux唯二可能出現的組合只有{0,1}與{1,0}，因此最長路徑其實只有10+20=30
但是STA會估計為20+20=40

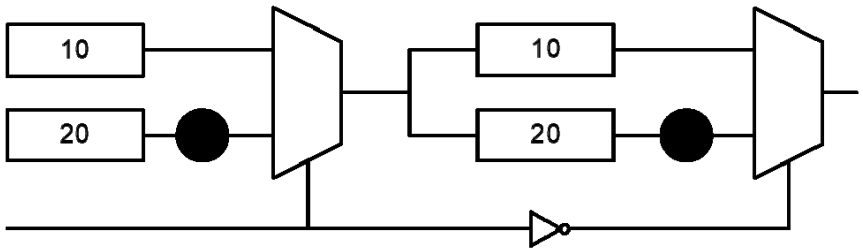


* Carry bypass adder skip the critical path

carry bypass adder的critical path其實是下面predictor的path，但是STA會估算成上面的

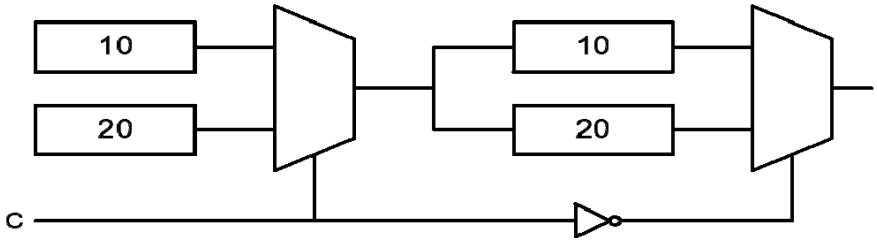
Ad-Hoc Methods

- **Path blocking**: user provides a list of nodes, or pair of nodes, that lie on paths of no interest 人工標記false path
  - Pair blocking works best with path enumeration



- **Case analysis**: User provides a list of cases (nodes and values) analyze the longest paths for every cases

Two cases:
C = 0
C = 1



打input進去，只需要少部分的邏輯分析即可去除一些false path

Path Sensitization

VALUE / GATE	Controlling value	NonControlling Value
AND	0	1
NAND	0	1
OR	1	0
NOR	1	0
NOT	0 , 1	---

0與1都是controlling value

controlling value: 只要有任何input出現x，output就會被決定，就會說x是這個gate的controlling value

- Path P = (I , G1 , G2 , … , Gn , O) is sensitizable if there is at least one vector which sensitizes the path

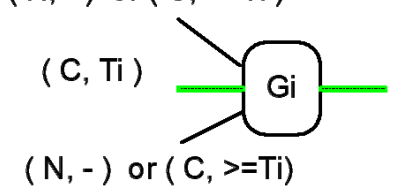
Path Sensitization

- Path P = (I , G1 , G2 , …, Gn , O) is sensitizable under input vector V, if each gate meets either of the following rules.

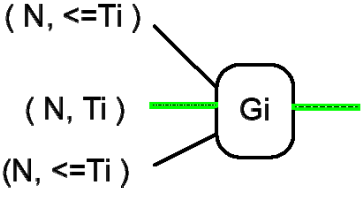
一條path中只要所有gate都滿足下面其中一個rule，就是sensitizable(意思就是只要量這條path的delay就好)

一個gate只要他的某個input出現controlling value，他的output就可以被決定了，因此這個gate的delay取決於出現controlling value的那條path

Rule 1) :
earliest controlling



Rule 2) :
latest non-controlling

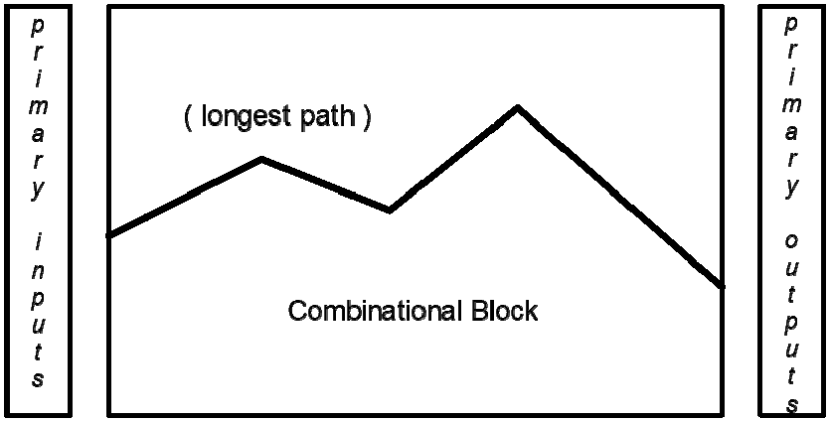


所有input都是non-controlling value的話，delay就是看最後到達的non-controlling delay

(stable value, stable time)
N : Non - controlling input , C : Controlling input
Ti = delay of partial path (I , G1 , G2 , … , Gi)

Accurate Performance

- Actual delay of the circuit: delay of the ***Longest Sensitizable Path*** 因為只有sensitizable path會影響delay，因此只要去量這些path並找出最長的就是電路中真正的delay
  - The longest sensitizable path can be much shorter than the longest path in a complex circuit 通常最長的sensitizable path都會比電路中的最長路徑短
  - Accuracy increases at the expense of computing efficiency 計算Longest Sensitizable Path也不會太費時



Outline

- Synthesis overview
- RTL synthesis
- Two-level logic optimization
- Multi-level logic optimization
- Technology mapping
- Timing analysis
- Timing optimization
  - Restructuring
  - Retiming & Resynthesis
- Synthesis for low power

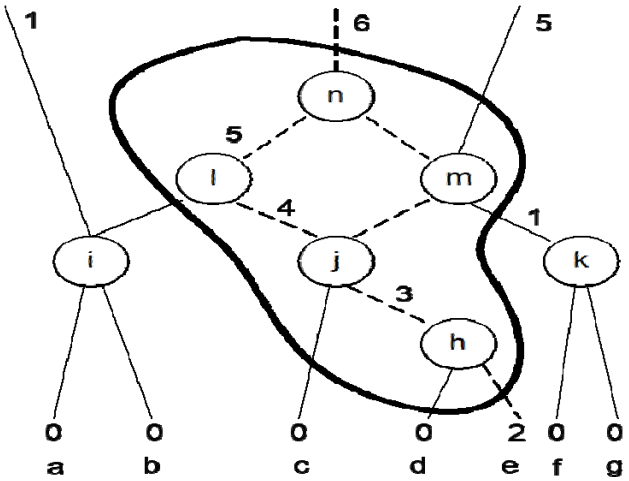
Restructuring Algorithm

While (circuit timing improves) do
    select regions to transform 選擇要重新合成的部分電路
    collapse the selected region
    resynthesize for better timing 重新合成
done

- Which regions to restructure ?
- How to resynthesize to minimize delay ?

Restructuring Regions

- All nodes with slack within ε of the most critical signal belong to the ε-*network* 所有與critical signal之間的slack 小於ε 的node，都屬於ε-network
- To improve circuit delay, necessary and sufficient to improve delay at nodes on cut-set of ε-*network*
  - Partially collapse this region and resynthesis for better timing



Find the Cutset

- The weight of each node is $W = W_x^t + \alpha * W_x^a$
  - W_x^t is potential for speedup 這兩個參數在下一頁會介紹
  - W_x^a is area penalty for duplication of logic
  - α is decided by various area/delay tradeoff
- ε: Specify the size of the ε-*network* ε 取太大會導致要重新合成的地方太多，取太小又可能變成沒有取到要修改的點
  - Large ε might waste area without much reduction in critical delay
  - Small ε might slow down the algorithm
- α: Control the tradeoff between area and speed
  - Large α avoids the duplication of logic
  - $\alpha = 0$ implies a speedup irrespective of the increase in area
- Apply the maxflow-mincut algorithm to generate the cutset of the ε-*network*

N(d)越大，代表這邊類似交通樞紐(經過此處的path較多)，只要把這邊解開就可以得到較大的改善

M(d)對應的是node，如果把一個node複製，其area就會變大，因此M(d)越大area penalty越大

The Weight of Each Node

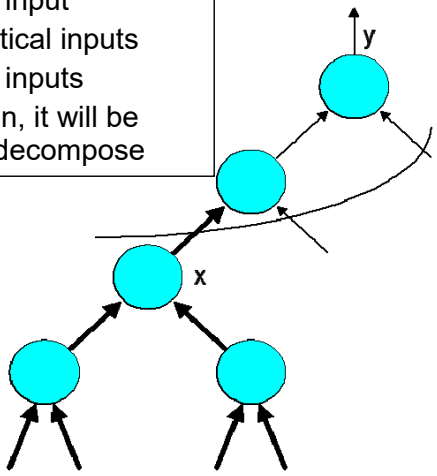
$$W_x^t(d) = \frac{|\{y \in N(d) \mid Sy \le \varepsilon\}|}{|N(d)|}$$

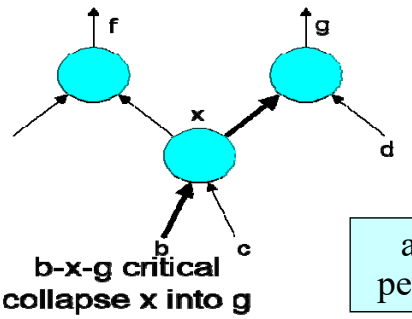
$$W_x^a(d) = \frac{|\{y \in M(d) \mid y.is.shared\}|}{M(d)}$$

這就只是一個很簡單的指標，讓我們能夠粗略的估計timing、area之間的trade off

- N(d) = number of inputs into resynthesis region
- M(d) = number of nodes in the resynthesis region

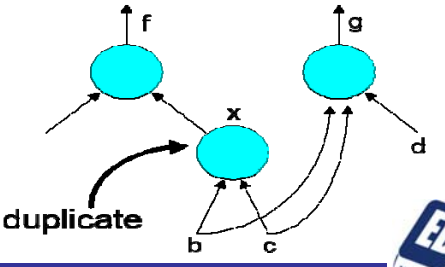
- Let d = 1 (collapsing depth)
  y ⇒ 1 critical input
  2 noncritical inputs
  x ⇒ 4 critical inputs
- If y is chosen, it will be easier to redecompose





b-x-g critical
collapse x into g

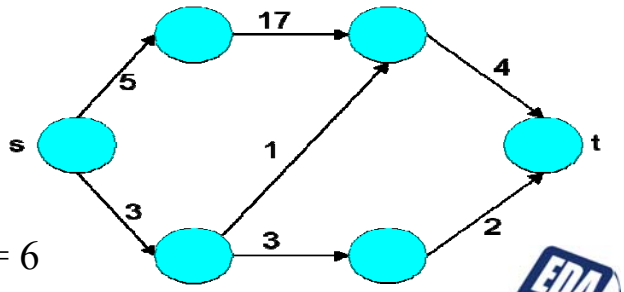
area penalty



Maximum Network Flow

- A network *N=(s, t, V, E, b)* is a diagram (V, E) together with a source s∈V and a sink t∈V with bound (capacity), b(u,v) ∈ Z^+ for all edge
- A flow f in N is a vector in $R^{|E|}$ such that
  1. $0 \le f(u,v) \le b(u,v)$ for all $(u,v) \in E$
  2. $\sum_{(u, v)\in E} f(u,v) = \sum_{(v, w)\in E} f(v, w)$ for all $v \in V-\{s,t\}$
- Maximum network flow problem: find the maximum allowed flow from s to t with the capacity constraints
  - Ford-Fulkerson algorithm
    Complexity = O(E|f*|)
  - Edmonds-Karps algorithm
    Complexity = O(VE^2)

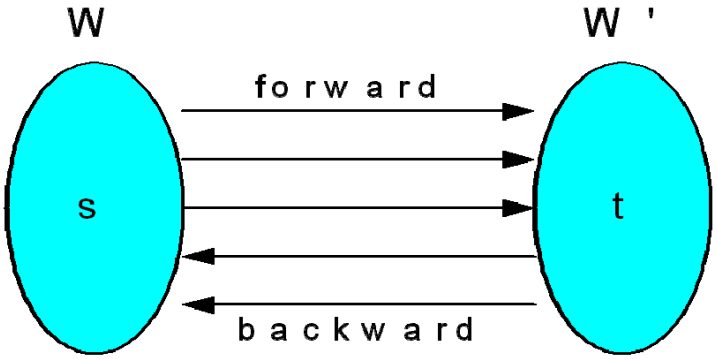
The value of the max flow |f| = 6



Max-Flow & Min-Cut 這兩個就是一樣的東西

- An s-t cut is a partition (W,W') of the nodes of V into sets W and W' such that s∈W and t∈W'.
- The capacity of an s-t cut

  $c(W,W') = \sum_{(i, j)\in E \text{ such that } i\in W,\ j\in W'} b(i, j)$
- Cut minimum nets to allow maximum flow !!



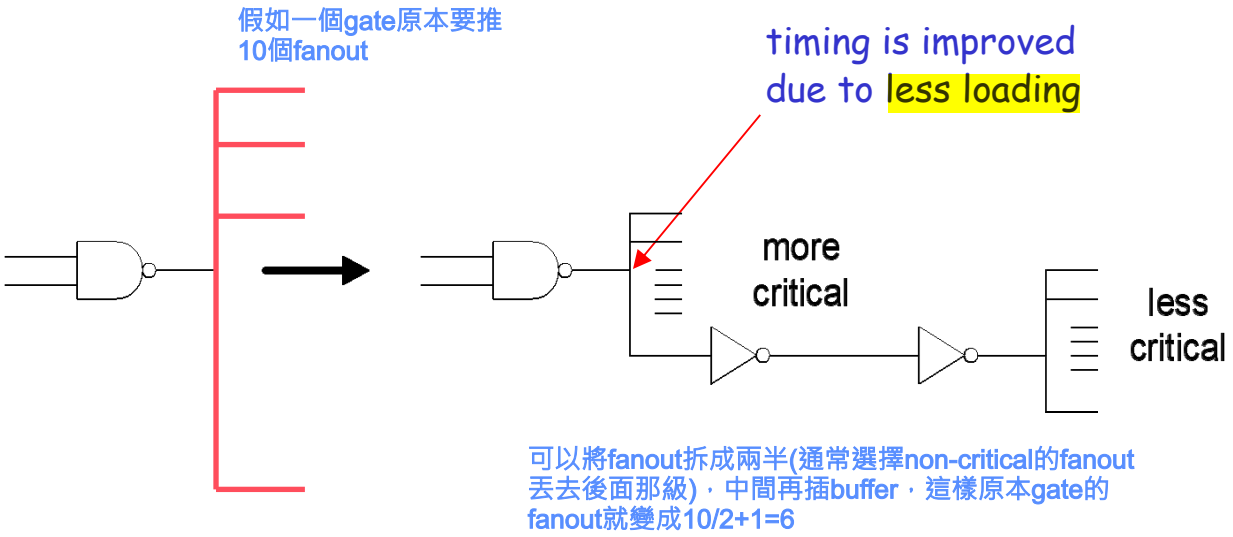
maxflow = mincut

Timing Optimization Techniques (1/8)

- Fanout optimization
  - Buffer insertion
  - Split
- Timing-driven restructuring
  - Critical path collapsing
  - Timing decomposition
- Misc
  - De Morgan
  - Repower
  - Down power
- Most of them will increase area to improve timing
  - Have to make a good trade-off between them

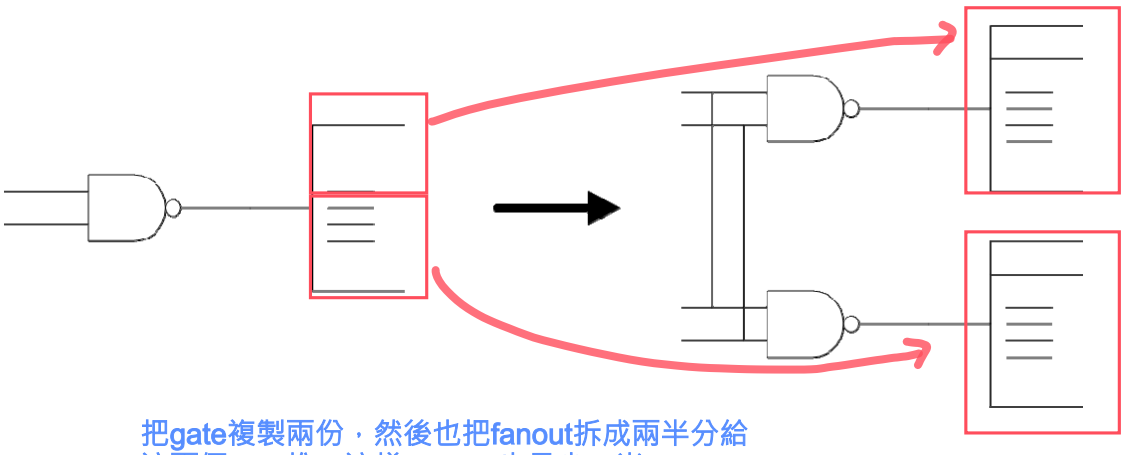
Timing Optimization Techniques (2/8)

- **Buffer insertion**: divide the fanouts of a gate into critical and non-critical parts and drive the non-critical fanouts with a buffer



Timing Optimization Techniques (3/8)

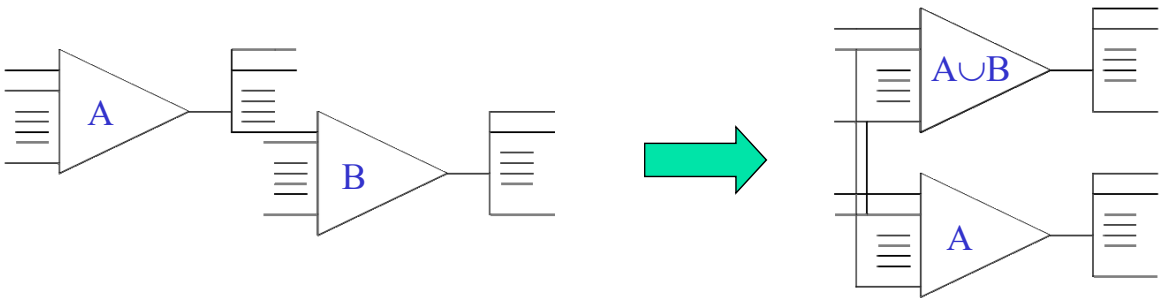
- **Split**: split the fanouts of a gate into several parts. Each part is driven with a copy of the original gate.



把gate複製兩份，然後也把fanout拆成兩半分給這兩個gate推，這樣loading也是少一半

Timing Optimization Techniques (4/8)

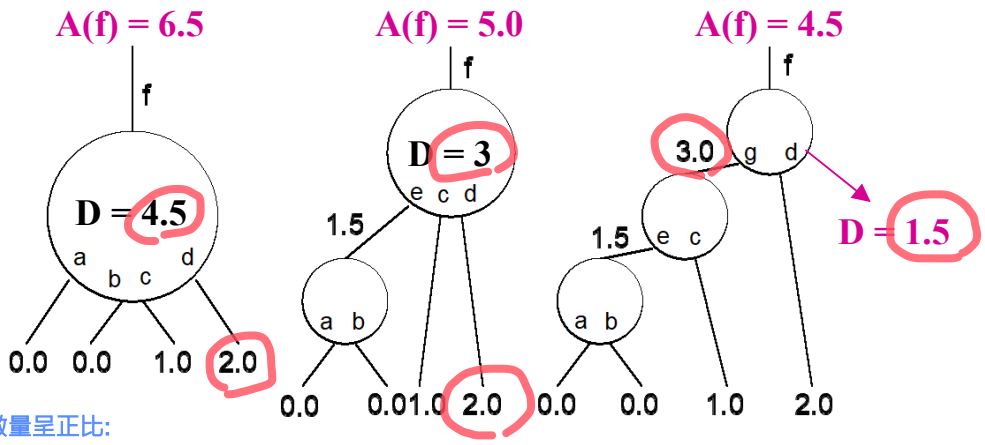
- **Critical path collapsing**: reduce the depth of logic networks



重新組合邏輯，將A、B兩個block做在一起就可以使其平行運作

Timing Optimization Techniques (5/8)

- **Timing decomposition**: restructuring the logic networks to minimize the arrival time



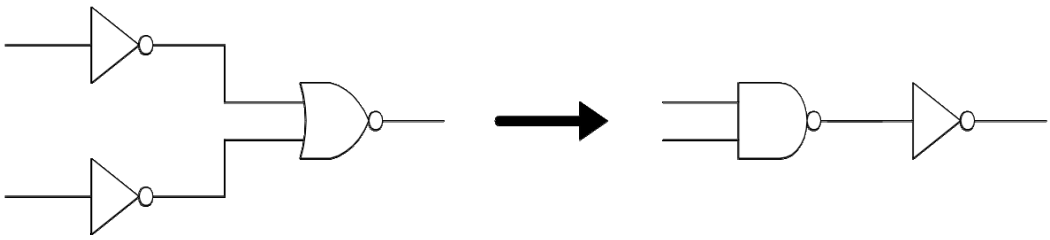
這邊假設gate delay與input數量呈正比:
4-input --> delay = 4.5
3-input --> delay = 3
2-input --> delay = 1.5

可以發現拆越多層delay越少，因為這樣其他較快到達的訊號就可以先被處理，不需要等最後一個抵達的訊號到達後才開始運算

Timing Optimization Techniques (6/8)

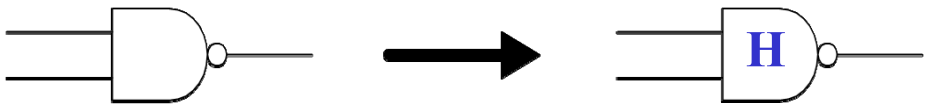
- **De Morgan**: replace a gate with its dual, and reverse the polarity of inputs and output
  - NAND gate is typically faster than NOR gate 詳情請見DIC

將NOR gate替換為NAND gate



Timing Optimization Techniques (7/8)

- **Repower**: replace a gate with one of the other gate in its logic class with higher driving capability

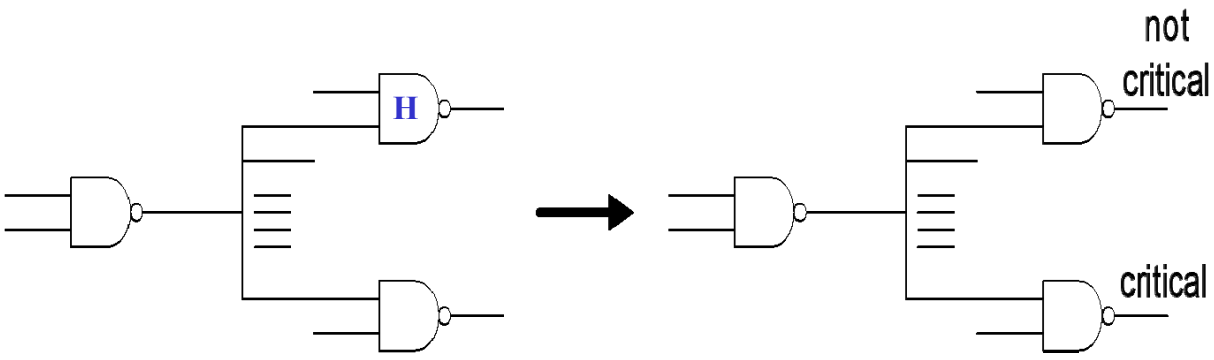


推不動就換一個sizing比較大(推力較大)的gate

Timing Optimization Techniques (8/8)

- **Down power**: reducing gate size of a non-critical fanout in the critical path

當timing都met後，可以適度地調降某些non-critical path的sizing

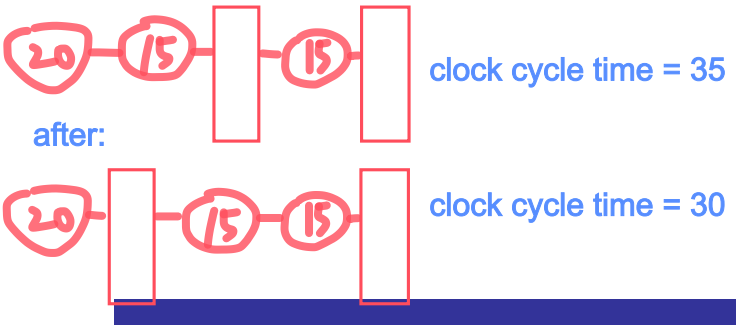


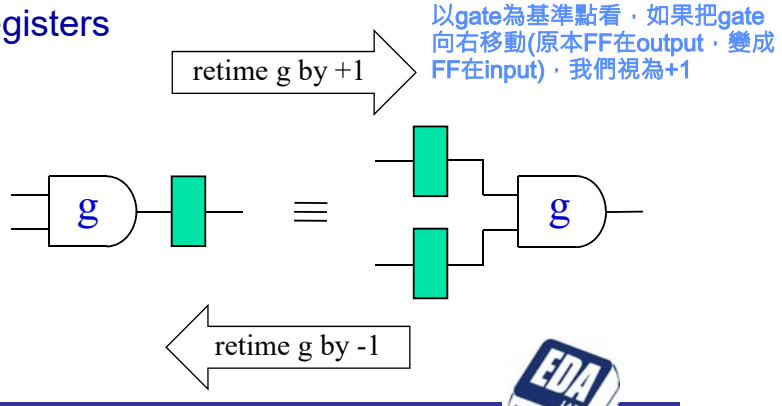
Outline

- Synthesis overview
- RTL synthesis
- Two-level logic optimization
- Multi-level logic optimization
- Technology mapping
- Timing analysis
- Timing optimization
  - Restructuring combinational circuit
  - Retiming & Resynthesis sequential circuit
- Synthesis for low power

Retiming

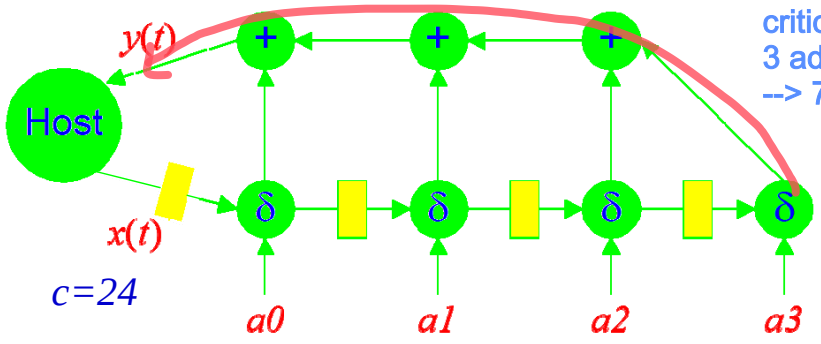
- Exploit the ability to move registers in a circuit
  - To minimize the cycle time
  - To minimize the the number of registers for a given cycle time
- Combinational logic is not modified
- Graph-Theoretic algorithms with polynomial time complexity
  - Vertex: combinational logic with propagation delay
  - Edge: number of clocked registers
  - $O(|V|^3 \lg |V|)$





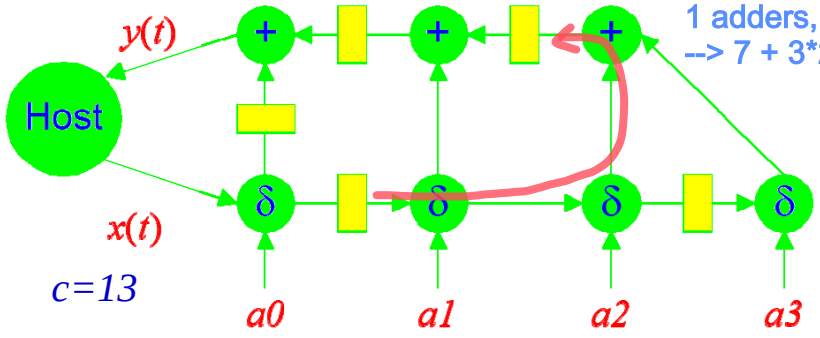
Example: Digital Correlator

- $y(t)=\delta(x(t),a0)+\delta(x(t\text{-}1),a1)+\delta(x(t\text{-}2),a2)+\delta(x(t\text{-}3),a3)$



critical path:
3 adders, 1 comparator
--> 7*3 + 3 = 24

$\delta(x,a)=0$ if $x \neq a$;
$\delta(x,a)=1$ if $x=a$



critical path:
1 adders, 2 comparator
--> 7 + 3*2 = 13

cost: 多了一個FF

- Delays:
  adder=7,
  comparator=3,
  host=0

Formulation

- Directed graph:
  - Nodes - combinational logic
  - Edges - connections (possible latched) between logic
- Weights
  - Nodes - combinational logic propagation delay
  - Edges - number of registers
- Path delay $d(P)$: sum of node delays along a path
- Path weight $w(P)$: sum of edge weights along a path
- D1: The propagation delay $d(v)$ is non-negative for each vertex v

檢查這三個rule　delay不可能是負的

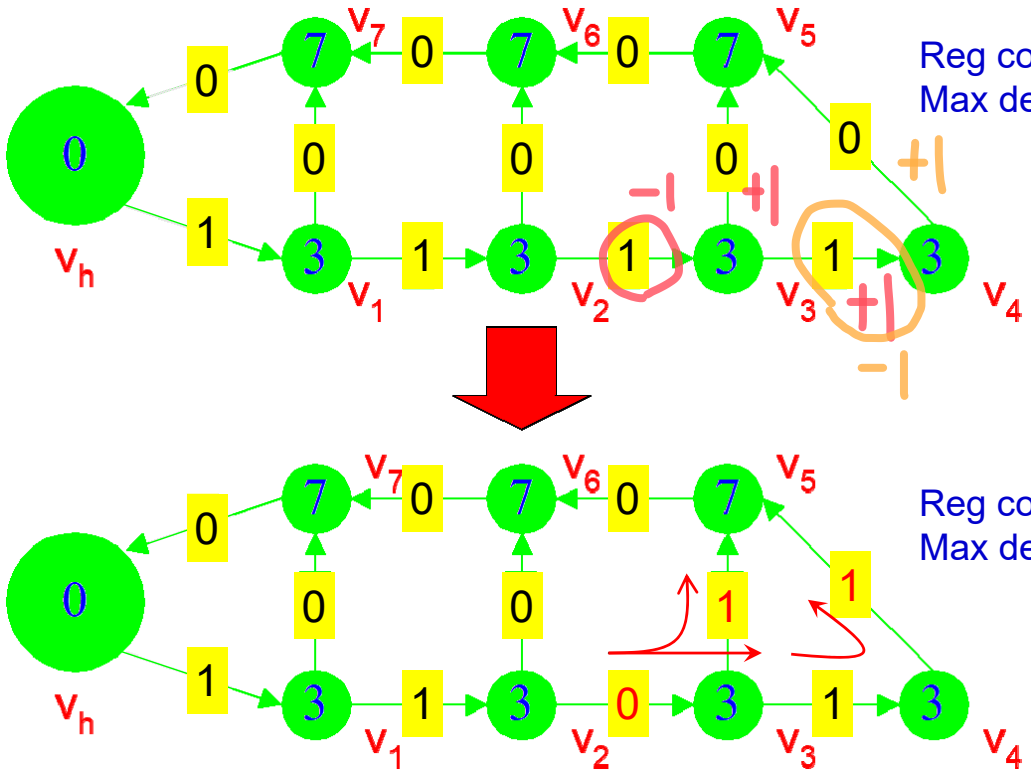
- W1: The register count $w(e)$ is non-negative for each edge e

register數量>=0

- W2: In any directed cycle, there is some edge with positive register count → no combinational loops

有combinational loop代表有latch，因此不能有loop

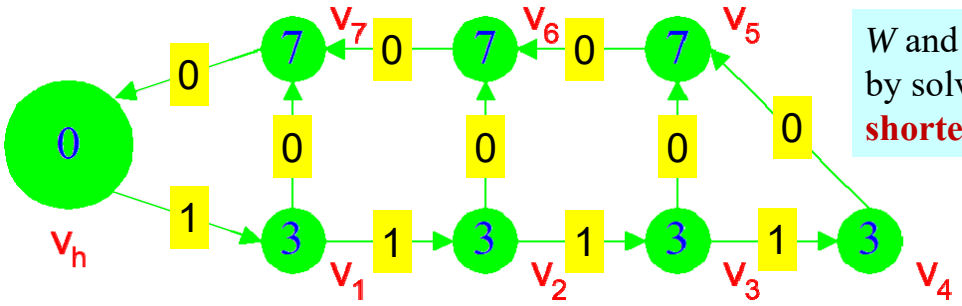
Example: Relocating Registers



Problem Definition: Optimal Retiming

- Given a graph G, find a legal retiming r of G such that the clock period $\Phi(G_r)$ of the retimed circuit G_r is as small as possible
- $W(u,v)$ is defined as the minimum number of registers on any path from vertex u to vertex v
- The critical path p is a path from u to v such that $w(p)=W(u,v)$
- $D(u,v)$ is defined as the maximum total propagation delay on any critical path from u to v

No. Registers (W) & Propagation Delay (D)



W and *D* can be obtained by solving **all-pair shortest-paths** problem

這邊就是把剛剛算W的路徑時經過的所有node的weight加起來
Ex:
vh與v5之間的最少register數量為3，要走的路徑是v1, v2, v3, v5，
因此
D = 3+3+3+7 = 16

這個表紀錄兩個node之間的最少register數量
-->W(u,v)

W	v_h	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_h	0	1	2	3	4	3	2	1
v_1	0	0	1	2	3	2	1	0
v_2	0	1	0	1	2	1	0	0
v_3	0	1	2	0	1	0	0	0
v_4	0	1	2	3	0	0	0	0
v_5	0	1	2	3	4	0	0	0
v_6	0	1	2	3	4	3	0	0
v_7	0	1	2	3	4	3	2	0

D	v_h	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_h	0	3	6	9	12	16	13	10
v_1	10	3	6	9	12	16	13	10
v_2	17	20	3	6	9	13	10	17
v_3	24	27	30	3	6	10	17	24
v_4	24	27	30	33	3	10	17	24
v_5	21	24	27	30	33	7	14	21
v_6	14	17	20	23	26	30	7	14
v_7	7	10	13	16	19	23	20	7

Optimal Retiming

- Let *G* be a synchronous circuit, and let *c* be the upper bound of clock period (any positive real number)
- Theorem: *r* is a legal retiming of *G* such that $\Phi(G_r) \le c$ if and only if

目標就是讓clock cycle time <= c

1. $r(v_h)=0$ 不會有register從起始點的左側移動到右側 --> $r(v_h) = 0$
2. $r(u)-r(v) \le w(e)$ for every edge $e(u,v)$
   - The move-out registers should not exceed the original registers
3. $r(u)-r(v) \le w(u,v)-1$ for every vertices *u* and *v* if $D(u,v) > c$
   - Breaking the long paths that exceeds the bound of clock period

$r(x)$ = no. of moved registers across vertex *x*
把register從x的左側移動到右側，r(x)++

w(e)代表的是register的數量，u,v是e的endpoints

因此r(v)是移入的register數量，r(u)是移出的register數量
--> w(e)+r(v)-r(u) >= 0

- Solve the integer linear programming problem 簡稱ILP
  - Bellman-Ford method in $O(|V|^3)$ 列出方程式後根據constraint找出最佳的整數解
- The set of *r*'s determine new positions of the registers

u-v path中，如果其total combinational delay已經超過我們的目標clock cycle time (c)，則u,v之間一定至少存在一個register(不然一定無法達成目標)
--> w(u,v) + r(v) - r(u) >= 1

Min-Delay Retiming

- Classical algorithm to find a legal retiming r, such that the cycle time c is minimized
  1. Compute *W* and *D*
  2. Sort the elements in the range of *D*
  3. Binary search the minimum achievable clock period by applying Bellman-Ford algorithm
  4. Derive the *r*(*v*) from the minimum achievable clock period found in Step 3
- Complexity = Bellman-Ford ($|V|^3$) x binary search ($lg|V|$) = $O(|V|^3 lg|V|)$
- Relaxation algorithm: Leiserson and Saxe [1]
  - Complexity = O(VE)

[1] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in Third Caltech conference on very large scale integration. Springer, 1983, pp. 87–116.

Min-Area Retiming

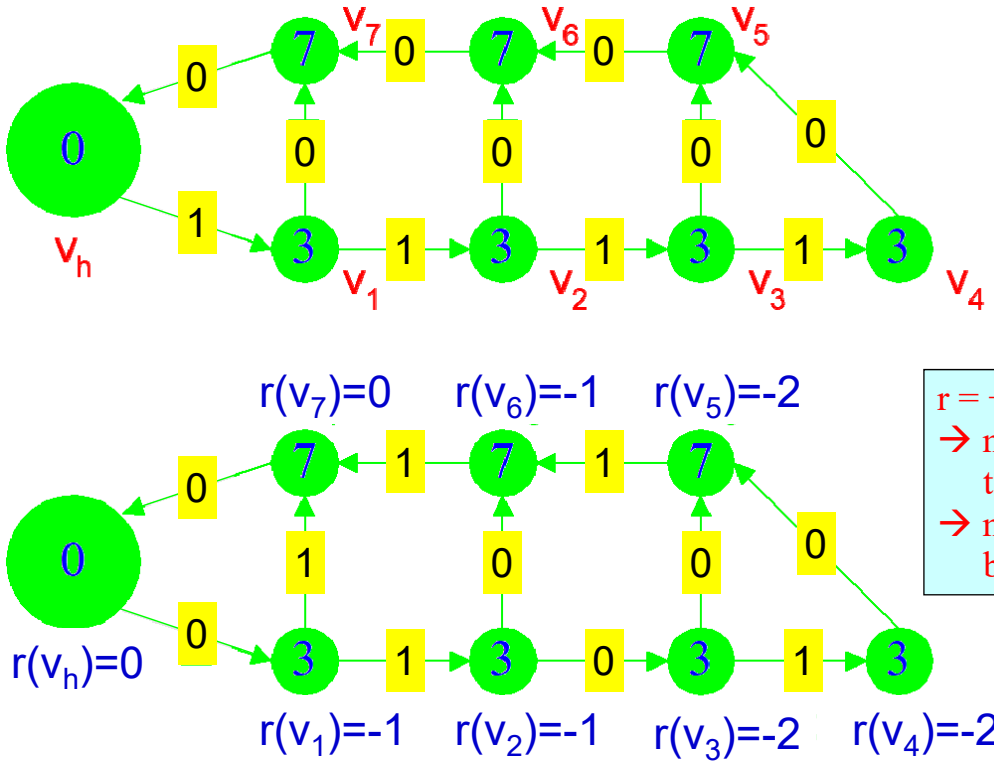
- Minimize the number of registers without delay constraints
  - Minimum-cost flow problem

$$\min : \sum_{\forall e_{uv}} r(u) - r(v) \wedge (\forall e_{uv}, r(u) - r(v) \leq w_{uv})$$

- Goldberg [3]: push-relabel method
  - Complexity = O(V^2 log(VC))
- A. Hurst et al. [2]: maximum network flow problem
  - Complexity = $O(R^2E)$
- Recent approach [4]: minimize area under a specific target clock period
  - Min-area approach [2] + min-delay retiming

[2] A. P. Hurst, A. Mishchenko, and R. K. Brayton, "Fast minimum-register retiming via binary maximum-flow," in Formal Methods in Computer Aided Design, IEEE, pp. 181–187, 2007.
[3] A. V. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," Journal of algorithms, vol. 22, no. 1, pp. 1–29, 1997.
[4] A. Hurst, A. Mishchenko, and R. Brayton, "Scalable min-register retiming under timing and initializability constraints," in Proc. 45th Design Automation Conference. ACM, pp. 534–539, 2008.

Retimed Correlator



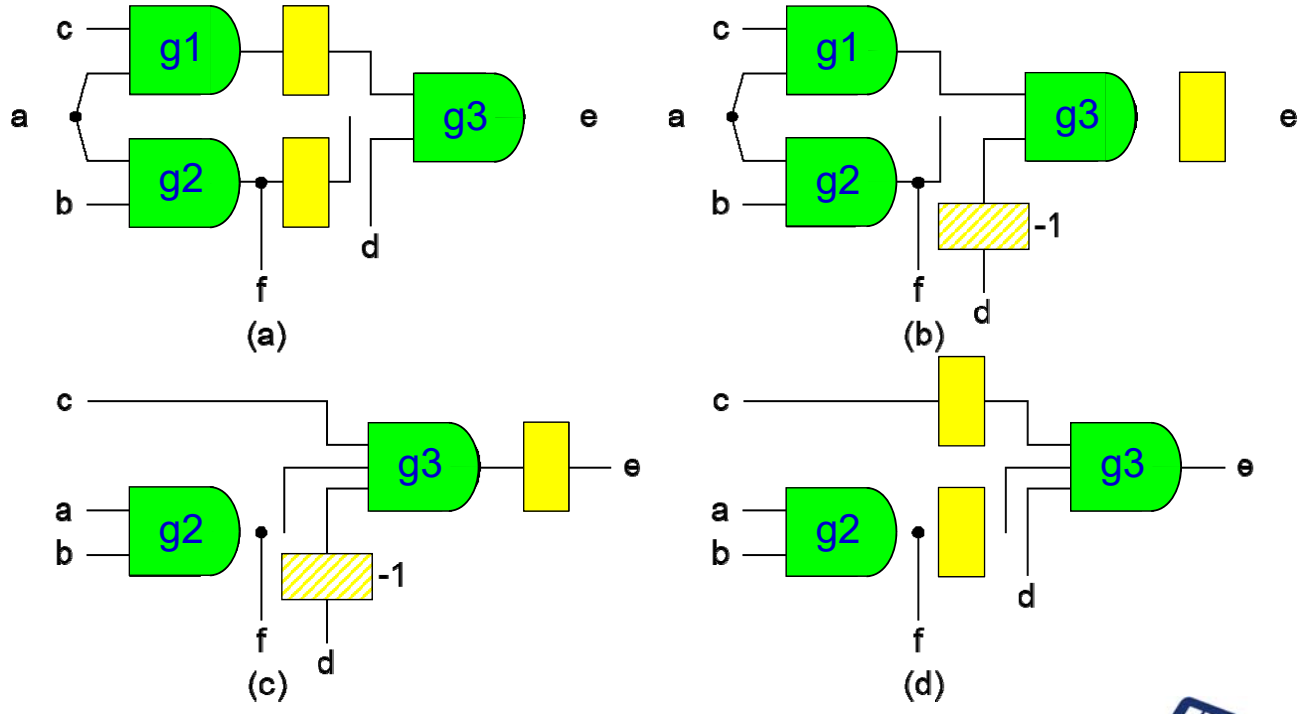
r = +1:
→ moving a gate to next cycle
→ moving a register backward

Retiming and Resynthesis

- Retiming offers the opportunity to resynthesize the combinational logic for further reduction [5]
- 3 Steps:
  1. Migrate all registers to the periphery of a sub-network
     - Peripheral retiming
  2. Optimize the sub-network with combinational technique
     - Resynthesis
  3. Replace registers back in the sub-network
     - Retiming

[5] S. Malik, E. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and Resynthesis: Optimization of Sequential Networks with Combinational Techniques," IEEE Transactions on Computer-Aided Design, January 1991.

Example: Retiming + Resynthesis



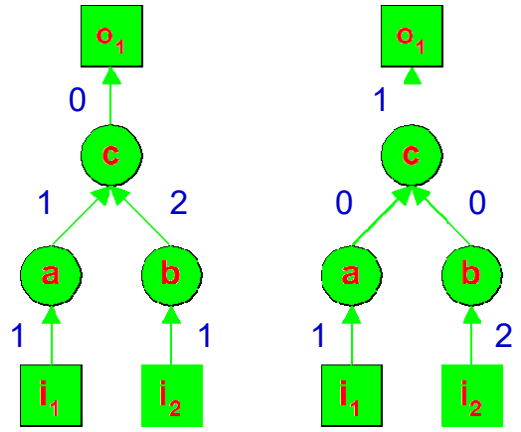
Peripheral Retiming

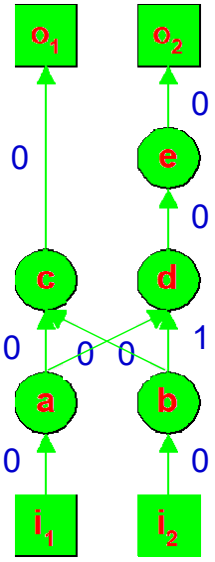
- A peripheral retiming is a retiming such that
  - $r(v)=0$ where v is an I/O pin
  - $w(u,v)+r(v)-r(u)=0$ where $e(u,v)$ in an internal edge
- Move all registers to the peripheral edges
- Leave a purely combinational logic block between two set of registers
- No two paths between any input i and any output j have different edge weights
- Exist α_i and β_j, $1 \le i \le m$, $1 \le j \le n$ such that $W_{i,j} = \alpha_i + \beta_j$

  $W_{i,j} = \sum_{path\ i_i \to o_j} w(e)$ if all paths between input i and output j have the same weight
- Complexity $O(e \cdot \min(m,n))$

Examples of Peripheral Retiming

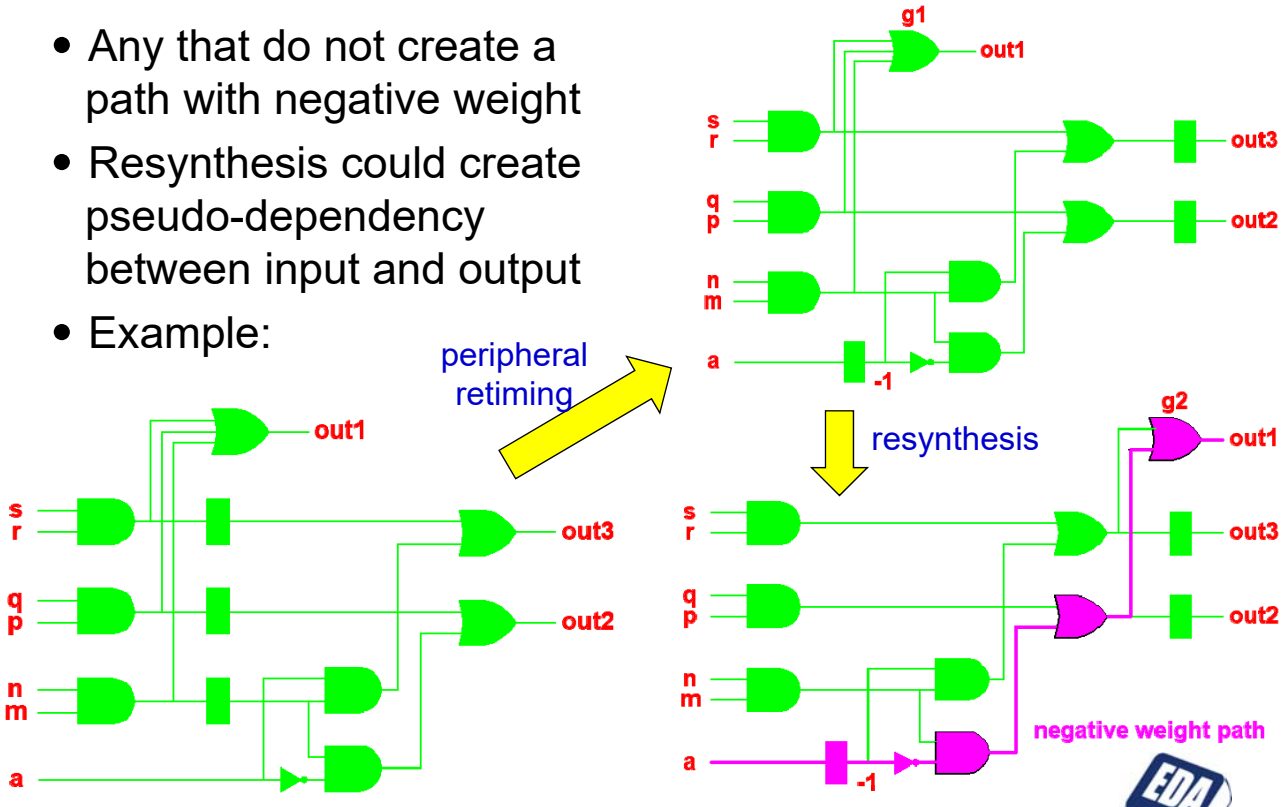
- Example 1:
  $W_{1,1}$=2, $W_{2,1}$=3,
  $\Rightarrow \alpha_1$=1, α_2=2, β_1=1
- Example 2:
  $W_{1,1}$=0,$W_{1,2}$=0,$W_{2,1}$=0, $W_{2,2}$=1
  $\Rightarrow$ no solution





Legal Resynthesis Operation

- Any that do not create a path with negative weight
- Resynthesis could create pseudo-dependency between input and output
- Example:



Challenges of Retiming Techniques

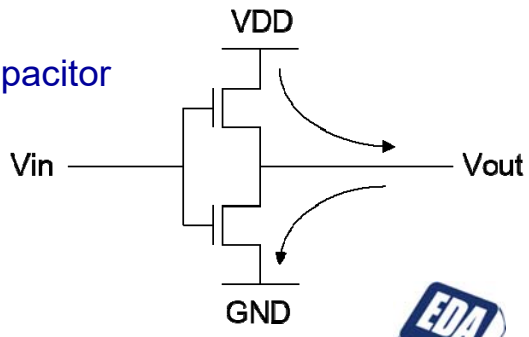
- Retiming complexity
  - Computing complexity is often too high
- Performance improvements due to retiming
  - Design performance of the physical netlist is not guaranteed due to the lack of placement info
  - To forecast whether the design performance could be improved becomes extremely important
- Side effects on verification
  - Sequential verification becomes difficult
  - Engineering Change Order (ECO)

Outline

- Synthesis overview
- RTL synthesis
- Two-level logic optimization
- Multi-level logic optimization
- Technology mapping
- Timing analysis
- Timing optimization
- Synthesis for low power

Power Dissipation

- Leakage power
  - Static dissipation due to leakage current
  - Typically a smaller value compared to other power dissipation
  - Getting larger and larger in deep-submicron process
- Short-circuit power
  - Due to the short-circuit current when both PMOS and NMOS are open during transition
  - Typically a smaller value compared to dynamic power
- Dynamic power
  - Charge and discharge of a load capacitor
  - Usually the major part of total power consumption



Power Dissipation Model

$$P = \frac{1}{2} * C * V_{dd}^{2} * D$$

- Typically, ***dynamic power*** is used to represent total power dissipation

  P: the power dissipation for a gate

  C: the load capacitance

  V_{dd}: the supply voltage

  D: the transition density

- To obtain the power dissipation of the circuit, we need
  - The **node capacitance** of each node (obtained from layout)
  - The **transition density** of each node (obtained by computation)

The Signal Probability

- Definition: The signal probability of a signal x(t), denoted by P_X^1 is defined as :

$$P_x^1 \equiv \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{+T/2} x(t)\, dt$$

where T is a variable about time.

- P_X^0 is defined as the probability of a logic signal X(t) being equal to 0.
- $P_X^0 = 1 - P_X^1$

Transition Density

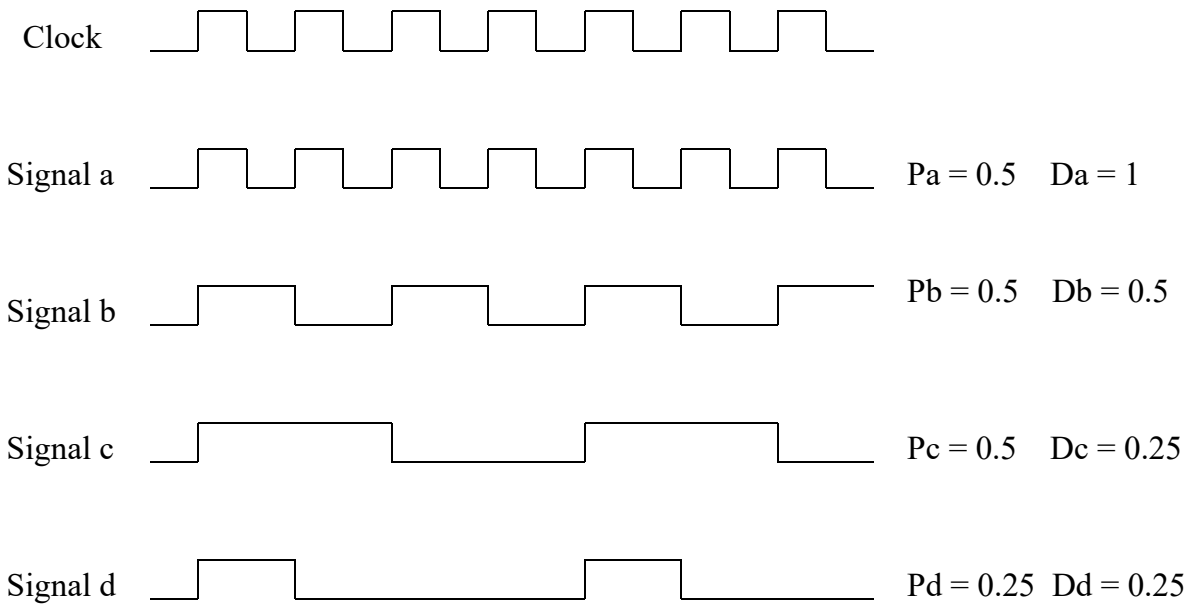
- Definition: The transition density Dx of a logic signal x(t), $t \in (-\infty, \infty)$, is defined as

$$Dx \equiv \lim_{T\to\infty} \frac{n_x(T)}{T \cdot f_c}$$

where f_c is the clock rate or frequency of operation.

- Dx is the expected number of transitions happened in a clock period.
- A circuit with clock rate 20MHz and 5 MHz transitions per second in a node, transition density of this node is 5M / 20M = 0.4

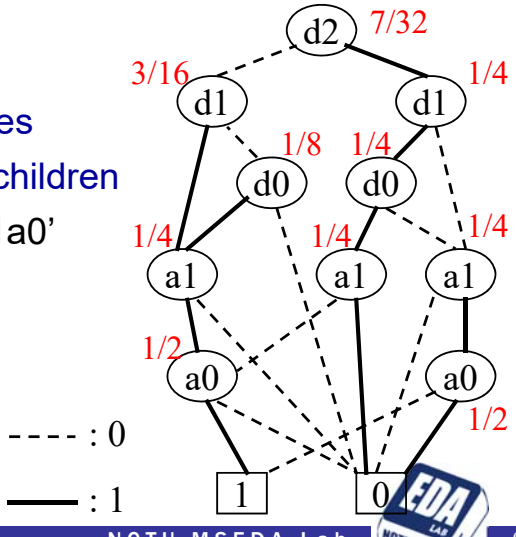
Signal Probability and Transition Density



The Calculation of Signal Probability

- BDD-based approach is one of the popular way
- Definition
  - p(F) : fraction of variable assignments for which F = 1
- Recursive Formulation
  - p(F) = [p(F[x=1]) + p(F[x=0])] / 2
- Computation
  - Compute bottom-up, starting at leaves
  - At each node, average the value of children
- Ex: F = d2'(d1+d0)a1a0 + d2(d1'+d0')a1a0' + d2d1d0a1'a0

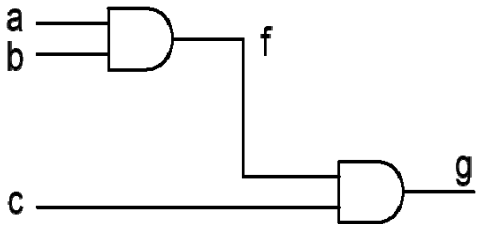
  p(F) = 7/32 = 0.21875



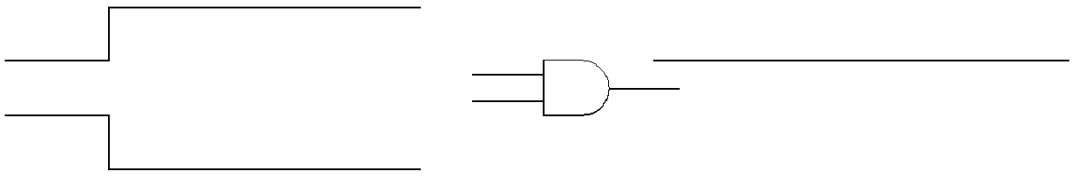
The Calculation of Transition Density

- Transition density of cube
  - f = ab
  - $D_f = D_a P_b + D_b P_a - 1/2\ D_a D_b$
  - $D_a P_b$ means that output will change when b=1 and a has changes
  - $1/2\ D_a D_b$ is the duplicate part when both a and b changes
- n-input AND :
  - a network of 2 -input AND gate in zero delay model
  - 3-input AND gate

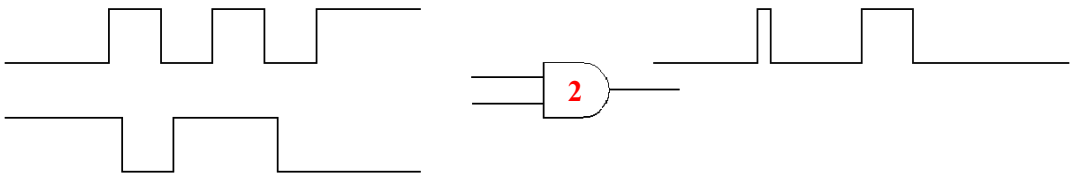
    $D_g = D_f P_c + D_c P_f - 1/2\ D_f D_c$
- Inaccuracy of this simple model :
  - Temporal relations
  - Spatial relations



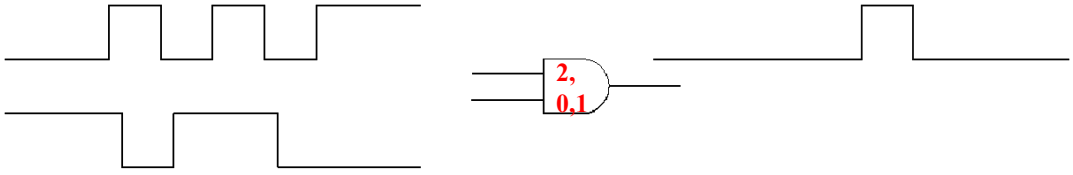
The Problem of Temporal Relations



(1) Without considering the Gate Delay and Inertial Delay

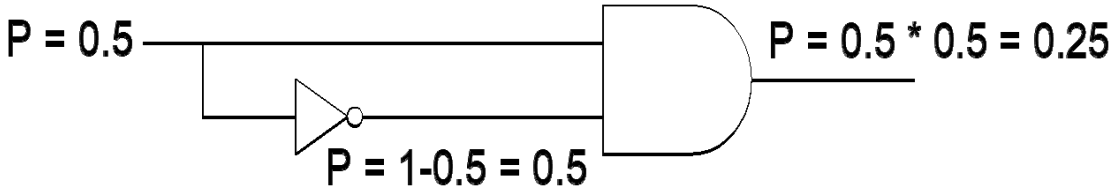


(2) Without considering Inertial Delay

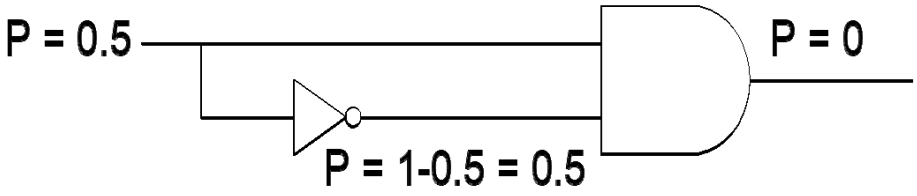


(3) Practical condition

The Problem of Spatial Correlation



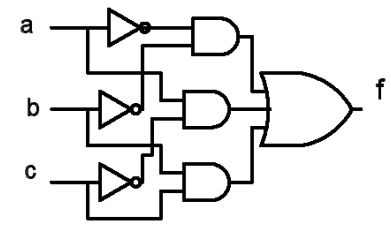
(a) Without considering Spatial Correlation

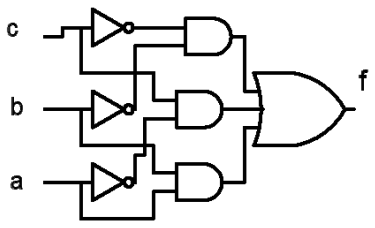


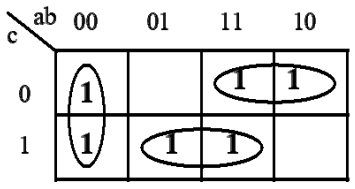
(b) Practical condition

Logic Minimization for Low Power (1/2)

- Consider an example:



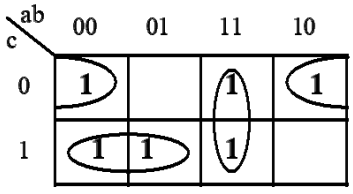




f = a'b' + ac' + bc

P = 108.7 μW

(a)



f = b'c' + a'c + ab

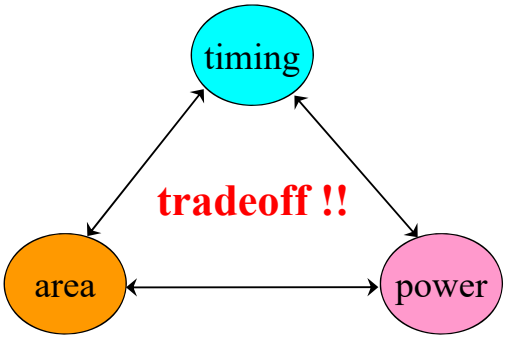
P = 115.5 μW

(b)

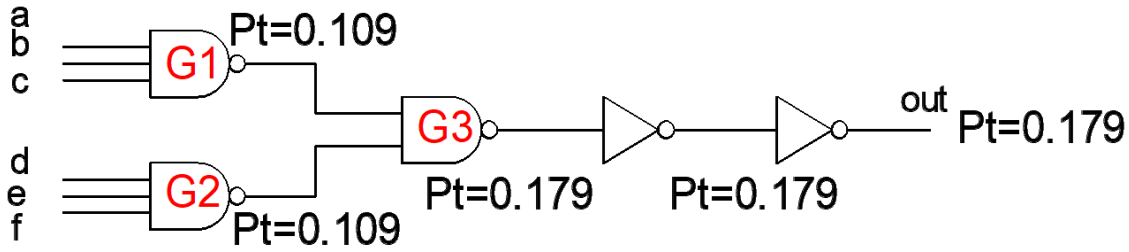
- Different choices of the covers may result in different power consumption

Logic Minimization for Low Power (2/2)

- Typically, the objective of logic minimization is to minimize
  - NPT : the number of product terms of the cover
  - NLI : the number of literals in the input parts of the cover
  - NLO : the number of literals in the output parts of the cover
- For low power synthesis, the power dissipation has to be added into the cost function for best covers



Technology Mapping for Low Power (1/3)

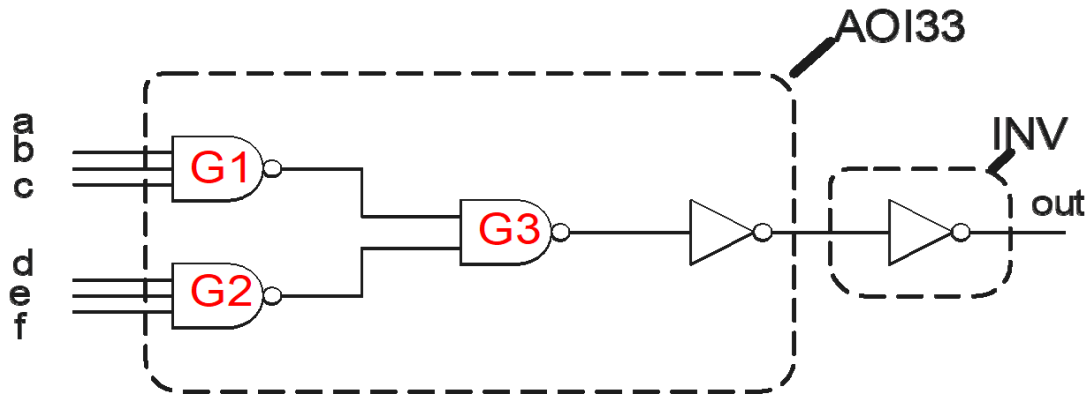


(a) Circuit to be mapped

Gate Type	Area	Intrinsic Cap.	Input Load
INV	928	0.1029	0.0514
NAND2	1392	0.1421	0.0747
NAND3	1856	0.1768	0.0868
AOI33	3248	0.3526	0.1063

(b) Characteristics of Library

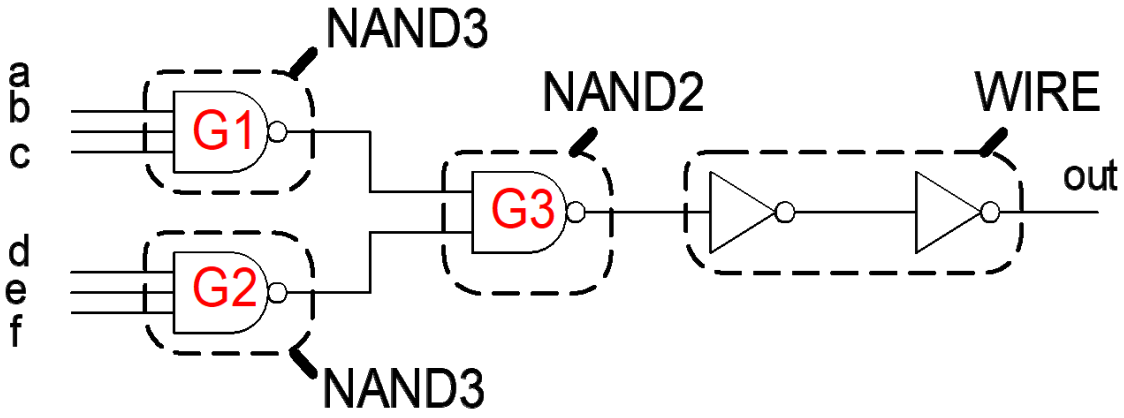
Technology Mapping for Low Power (2/3)



Area Cost: 4176
Power Cost: 0.0907

(a) Minimun-Area Mapping

Technology Mapping for Low Power (3/3)



Area Cost: 5104
Power Cost: 0.0803

(b) Minimun-Power Mapping