Institute of Electronics, National Chiao-Tung Univ.
Mixed-Signal Electronic Design Automation LAB

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

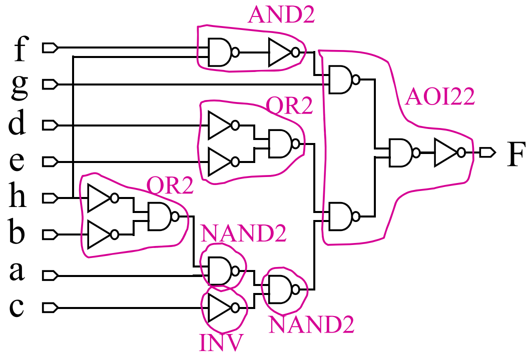
Outline

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

Technology Mapping

- 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’;



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

Major Approaches

- Rule-based systems:
  - Mimic designers’ activity
  - Handle all types of cells
- Heuristic algorithms:
  - Restricted to single-output combinational cells
  - DAGON approach
- Most tools use a combination of both

Rule-Based Library Binding

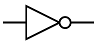
- Binding by stepwise transformations
- Data-base:
  - Set of patterns associated with best implementations
- Rules:
  - Select sub-network to be mapped
  - Handle high-fanout problems
- Advantages:
  - Applicable to all kinds of libraries
- Disadvantages:
  - Large rule data-base
    - Completeness issue
  - Data-base updates

Heuristic Approach

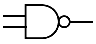
- Find a partition of the given network such that each sub-network can be replaced by a library cell
- General approach:
  - Choose base function set for canonical representation
    - Ex: 2-input NAND and Inverter
  - Represent optimized network using base functions
    - Subject graph
  - Represent library cells using base functions
    - Pattern graph
  - Each pattern associated with a cost which is dependent on the optimization criteria
- Goal:
  - Finding a minimal cost covering of a subject graph using pattern graphs

Example Pattern Graph (1/3)

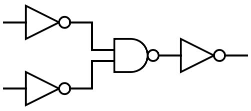
inv (1)



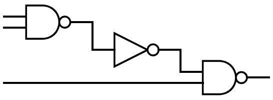
nand2 (1)



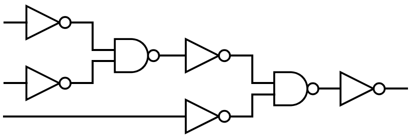
nor2 (2)



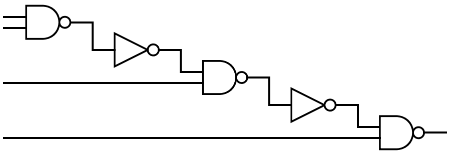
nand3 (3)



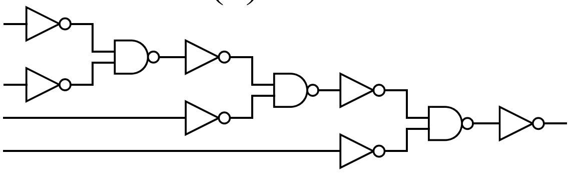
nor3 (3)



nand4 (4)

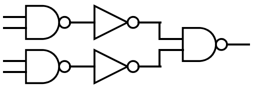


nor4 (4)

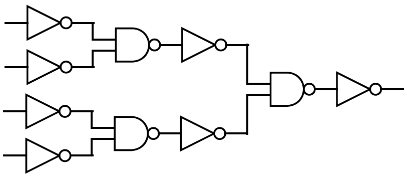


Example Pattern Graph (2/3)

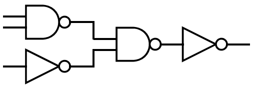
nand4 (4)



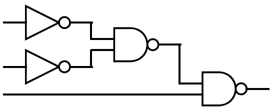
nor4 (4)



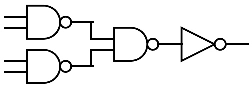
aoi21 (3)



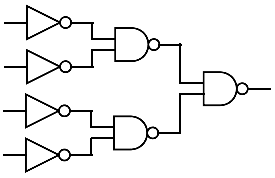
oai21 (3)



aoi22 (4)

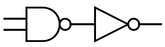


oai22 (4)

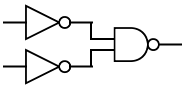


Example Pattern Graph (3/3)

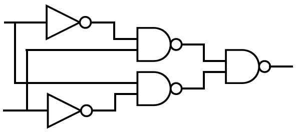
and2 (3)



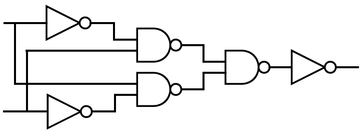
or2 (3)



xor (5)

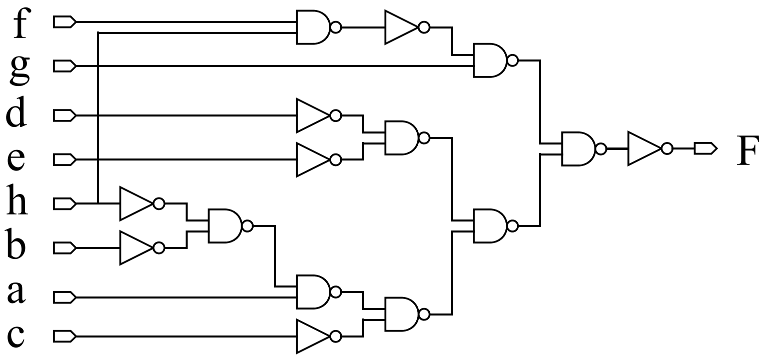


xnor (5)

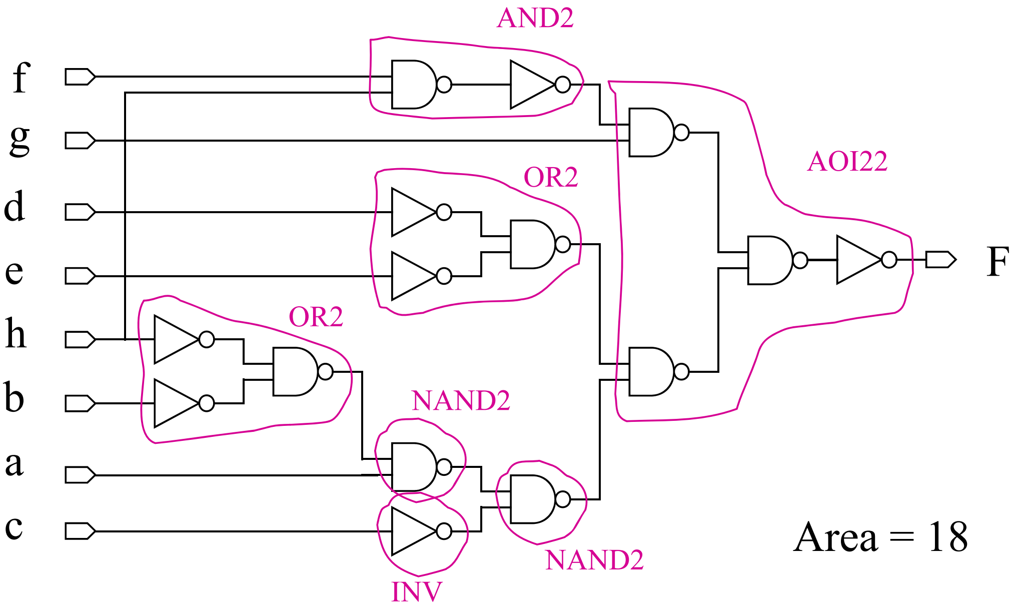


Example Subject Graph

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

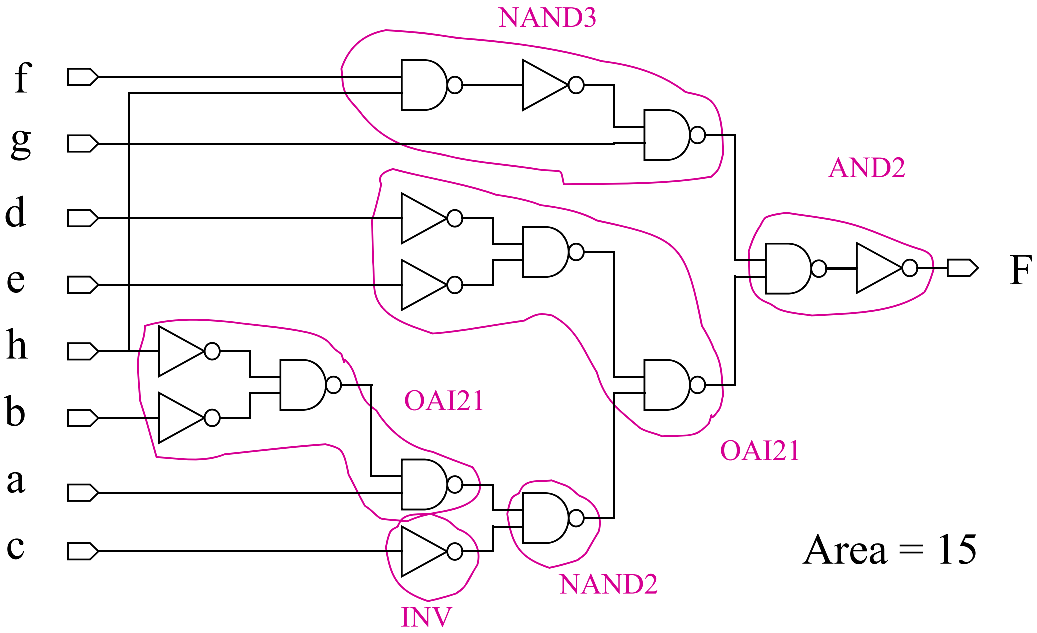


Sample Covers (1/2)



Area = 18

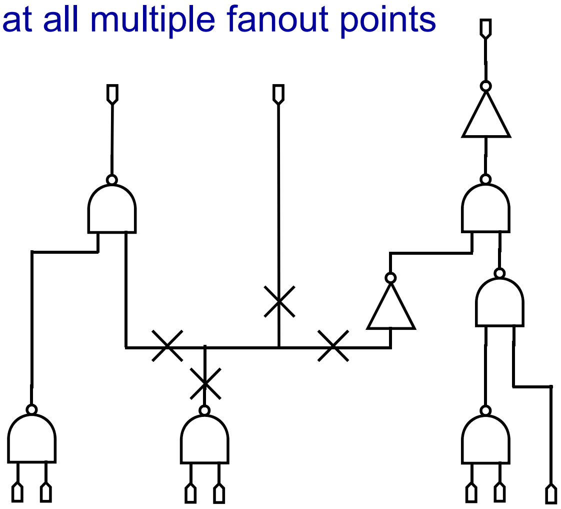
Sample Covers (2/2)



Area = 15

DAGON Approach

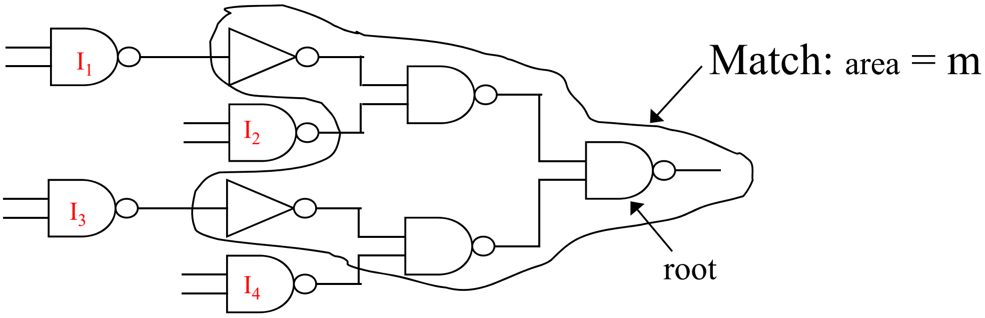
- Partition a subject graph into trees
  - Cut the graph at all multiple fanout points



- 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

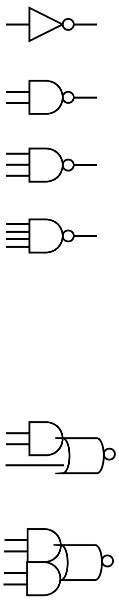


$$A(root) = m + A(I_1) + A(I_2) + A(I_3) + A(I_4)$$

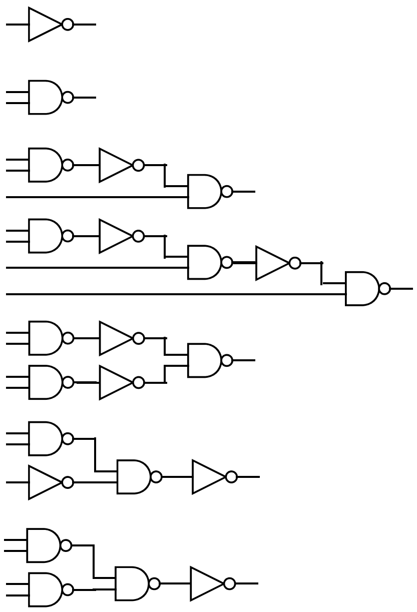
cost of a leaf = 0

A Library Example

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

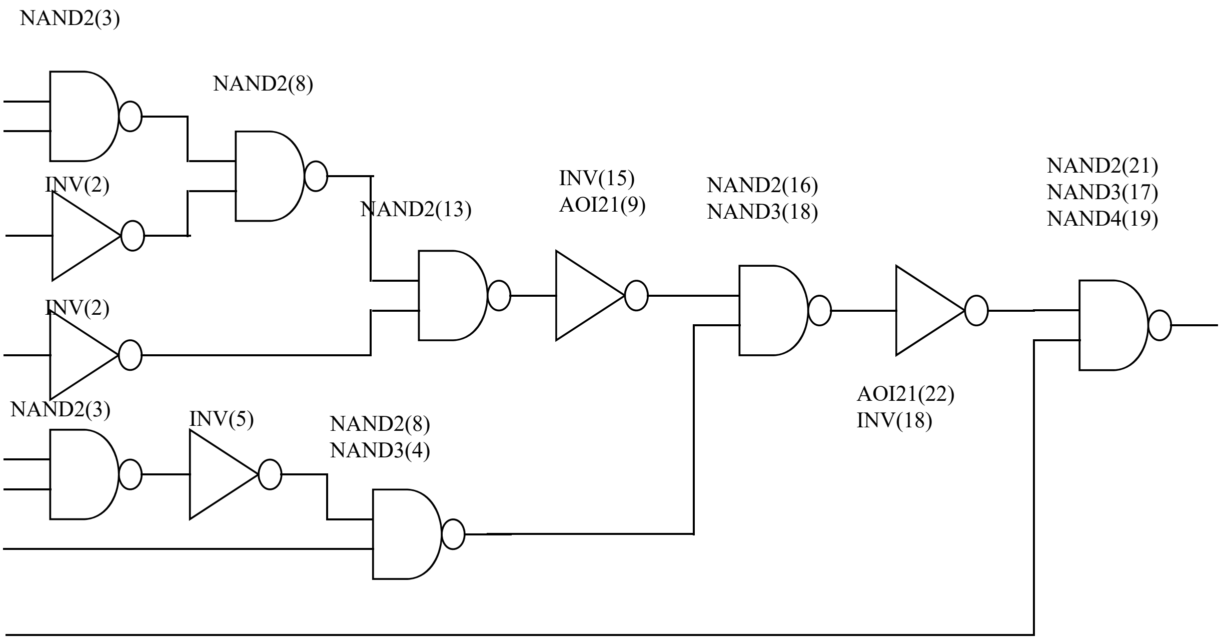


Library Element



Canonical Form

DAGON in Action

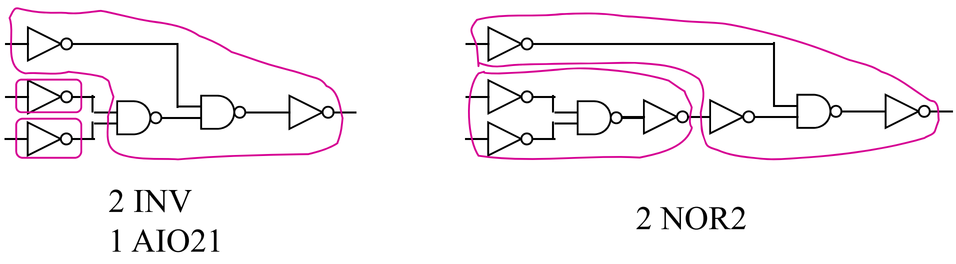


Features of DAGON

- 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
- Cons. Of DAGON:
  - With only a local (to the tree) notion of timing
    - Taking load values into account can improve the results
  - Can destroy structures of optimized networks
    - Not desirable for well-structured circuits
  - Inability to handle non-tree library elements (XOR/XNOR)
  - Poor inverter allocation

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

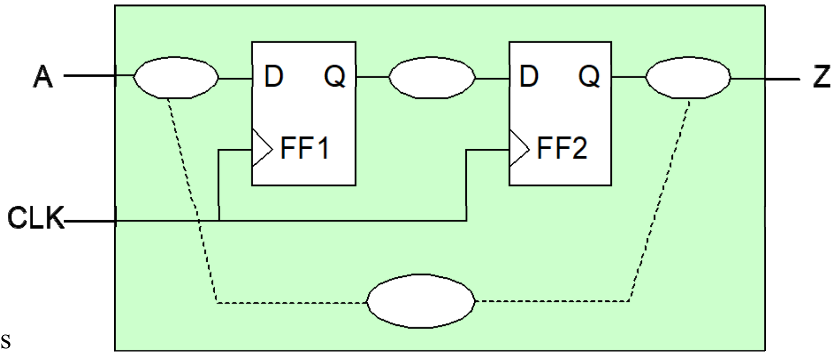


Outline

- 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

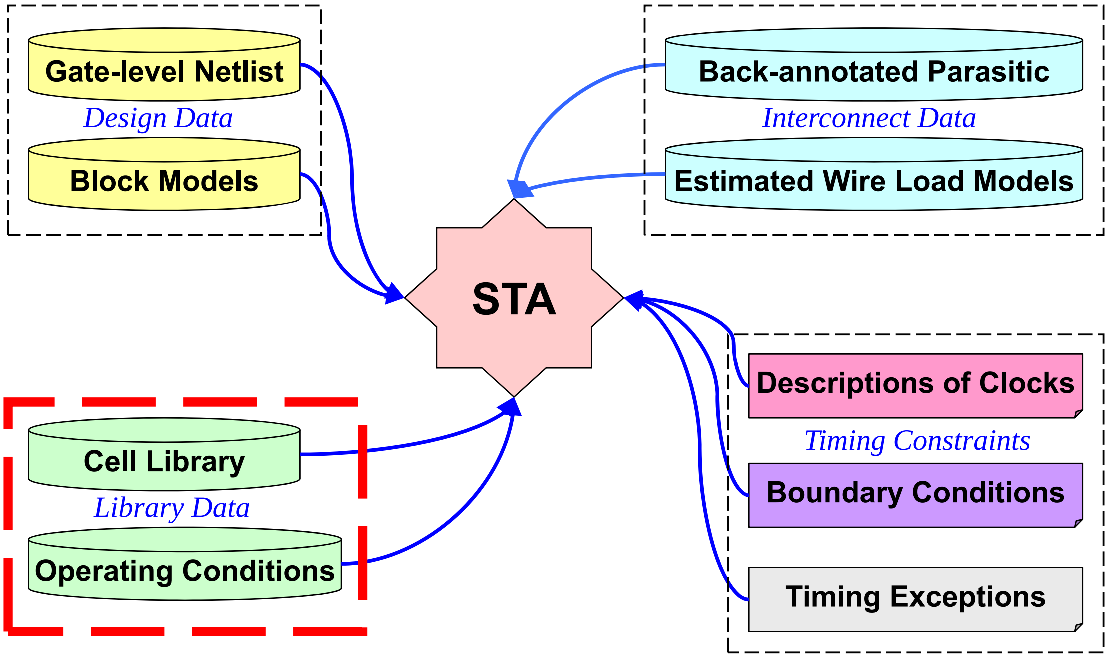


Reference :Synopsys

Performance Verification

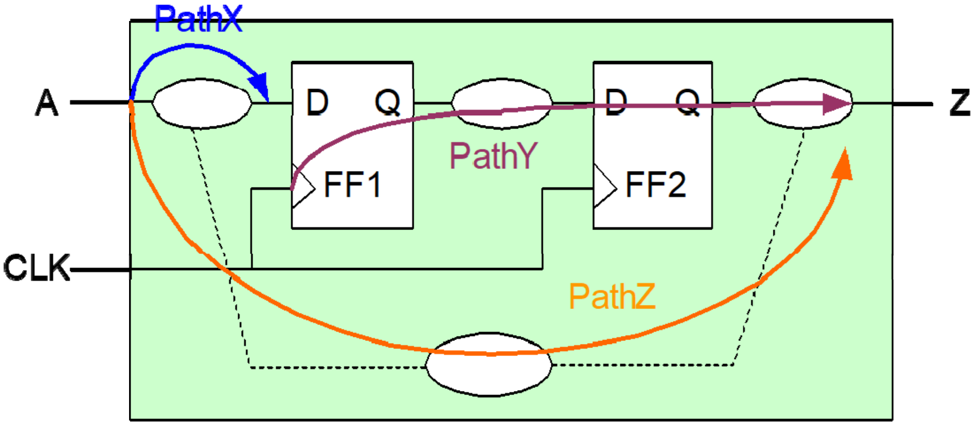
- Simulation-based approaches
  - Pattern-dependent
  - Incomplete coverage
  - Slow
  - Accurate
  - Impractical for large designs
- Static timing analysis techniques
  - Functionality assumed correct
  - Pattern-independent timing check
  - Complete coverage
  - Fast
  - Maybe inaccurate because of **false paths**

Data Preparation for STA



3 Steps in Static Timing Analysis

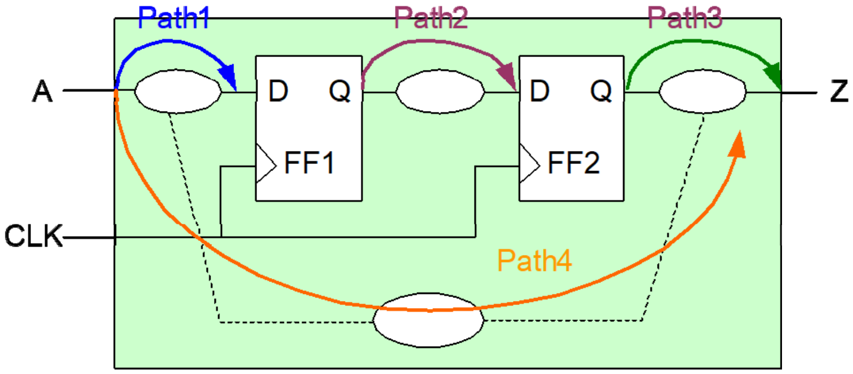
1. Design is broken down into sets of timing paths
2. Delay of each path is calculated
3. Path delay are checked to see if timing constraints have been meet



Reference :Synopsys

Timing Paths

- There are 4 types of timing path
  - Input port to data pin of FF (path 1)
  - Clock pin of FF to data pin of FF (path 2)
  - Clock pin of FF to output port (path 3)
  - Input port to output port (path 4)

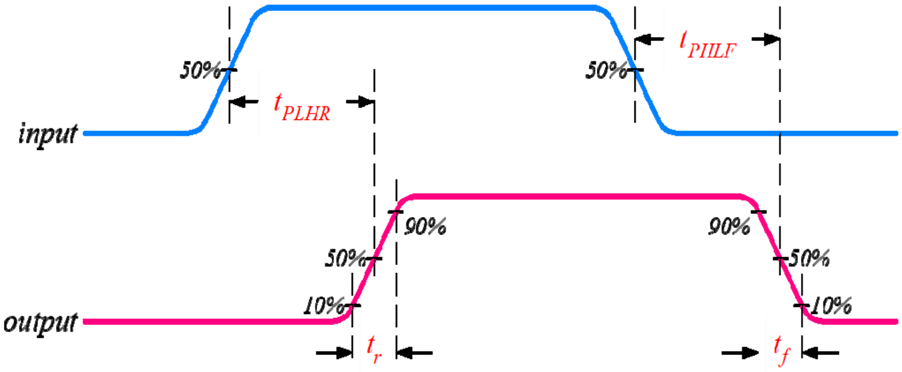


Stat point :
input port
Clock pin of FF
End point
output port
data pin of FF

Reference :Synopsys

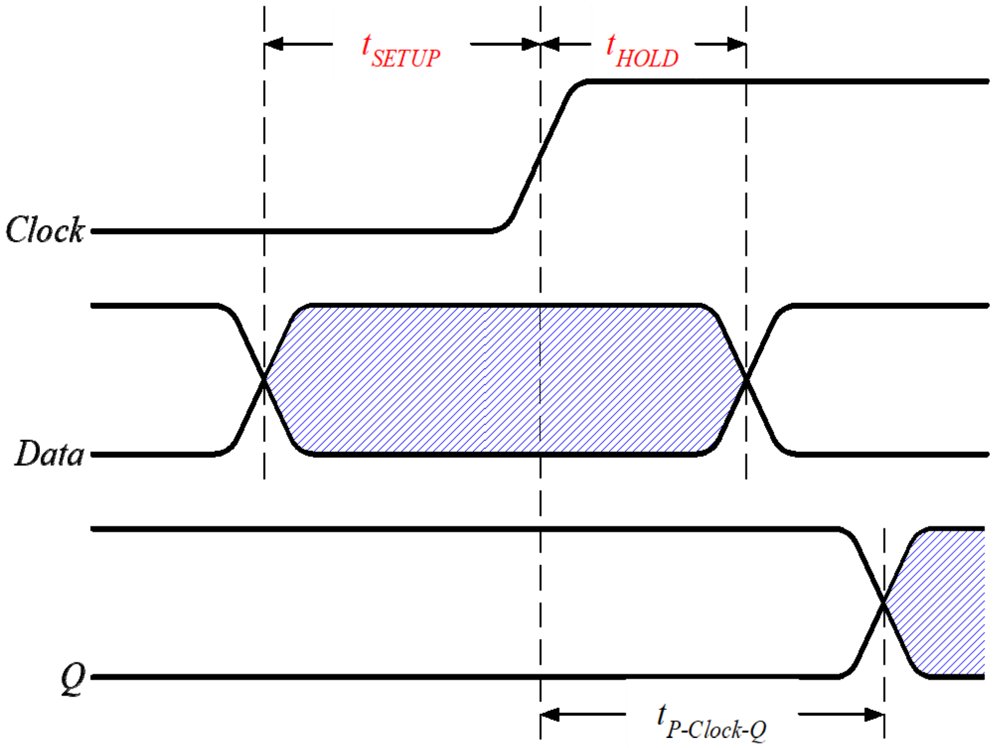
Definition of Timing Parameters

- 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

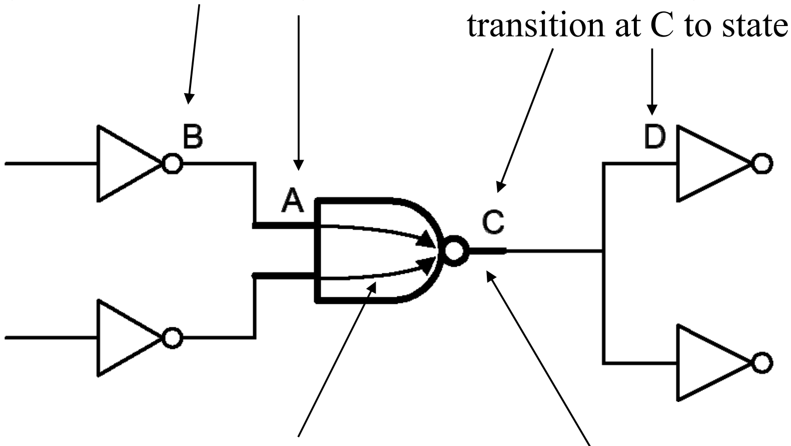
1. unit delay model
   - Assign a delay of 1 to ***each gate***
2. unit fanout delay model
   - Incorporate an ***additional*** delay for ***each fanout***
3. library delay model
   - Use delay data in the library to provide more accurate delay value
   - May use linear or non-linear (tabular) models
4. Post layout delay model
   - 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

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

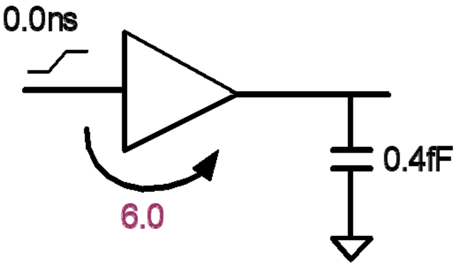


D_I : Intrinsic delay : incurred from cell input to cell output

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



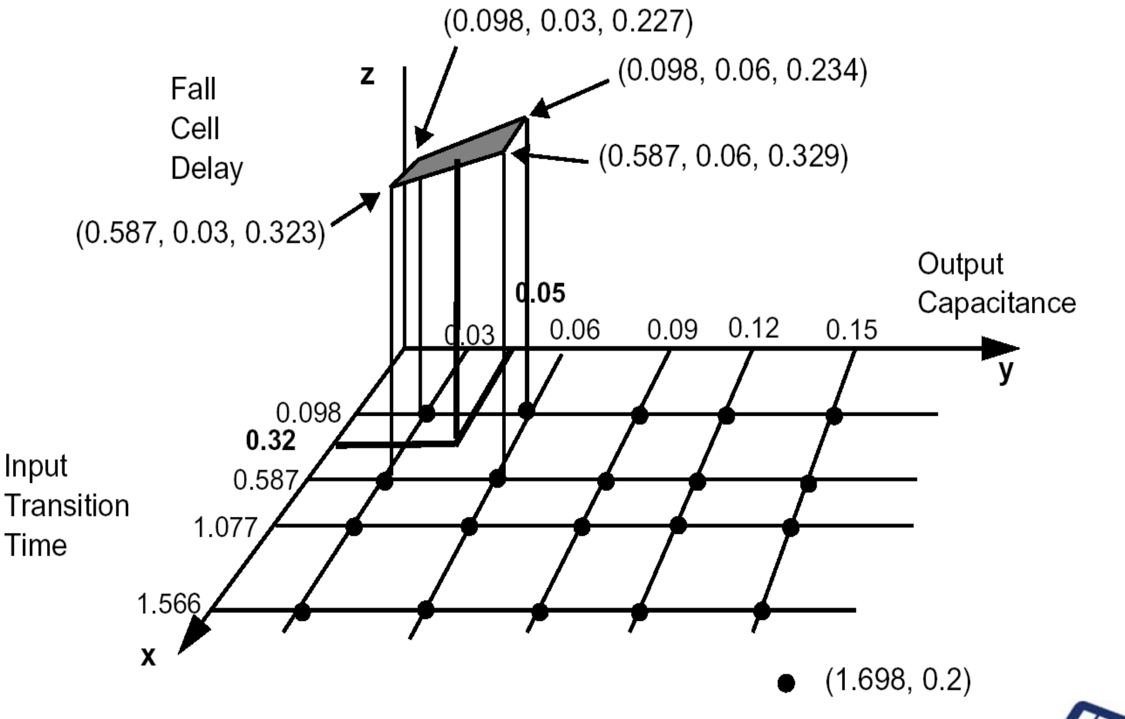
Total Output Load (fF)

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

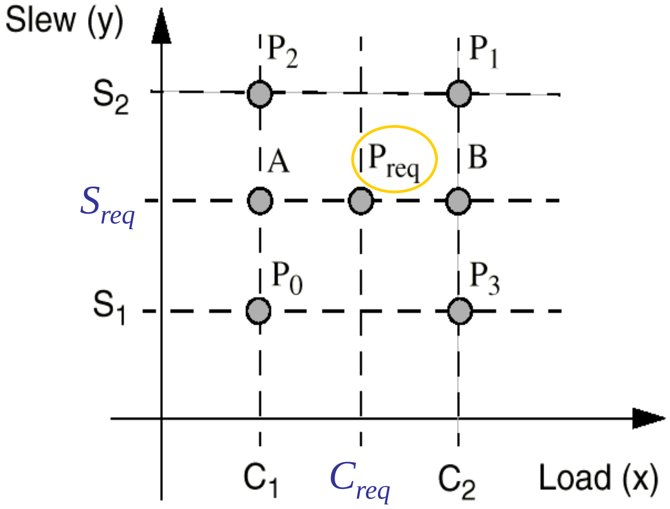
Cell Delay (ps)

- Can be more precise than linear delay model
  - table size↑ ➔ accuracy ↑
- Require more space to store the table

Illustration of Delay Calculation



2-D Table Interpolation



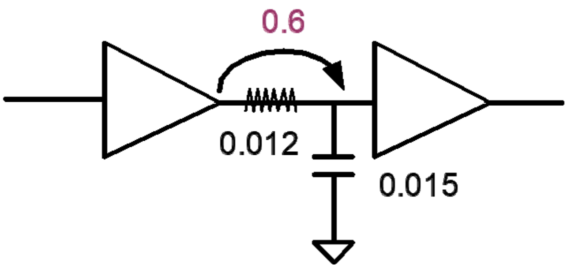
$$A = P_0 + \left(\frac{P_2 - P_0}{S_2 - S_1}\right)(S_{req} - S_1)$$

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

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

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
  - 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

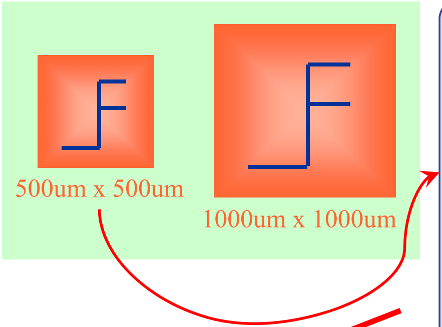


Net Fanout	Load (fF)	Resistance (mΩ)
1	0.015	0.012
2	0.030	0.016
3	0.045	0.020
4	0.060	0.024

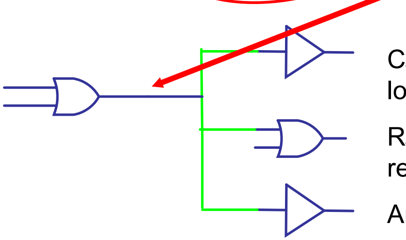
Reference :Synopsys

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

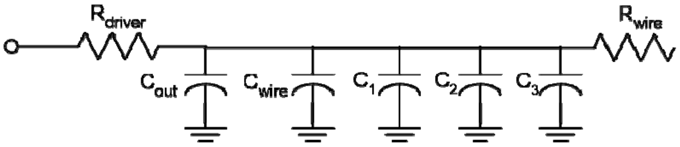
R_{wire} = (fanout=3, length =2.8) x resistance coefficient (3.0) = 8.4 resistance units

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

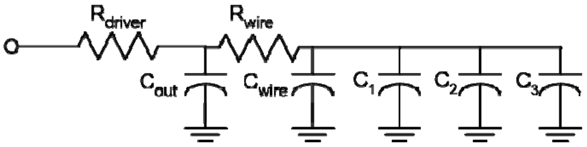
Interconnect Models

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

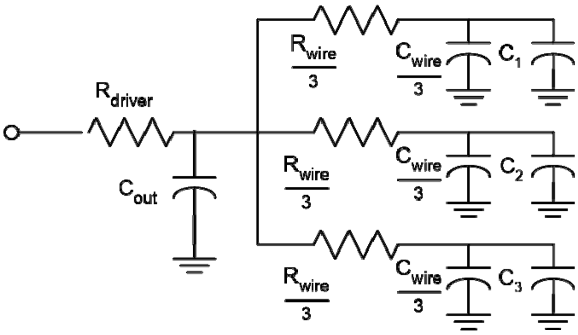
Best-Case RC Tree



Worst-Case RC Tree



Balanced Case RC Tree

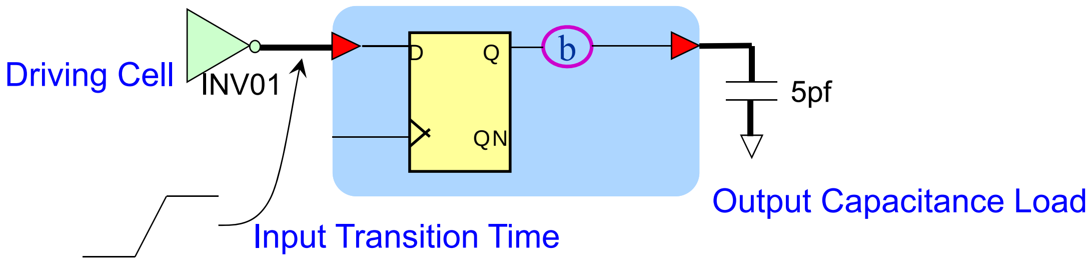


Interconnect Data

- 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)
  - 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

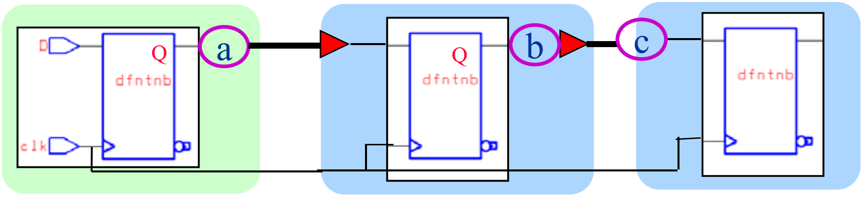
- Input driving cell
- Input transition time
- Output capacitance load
- Input delay (or Arrival Time)
- Output delay (or Required Time)



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
- required time : calculated from output to input
- slack = required time - arrival time

A(j): arrival time of signal j

R(k): required time or for signal k

S(k): slack of signal k

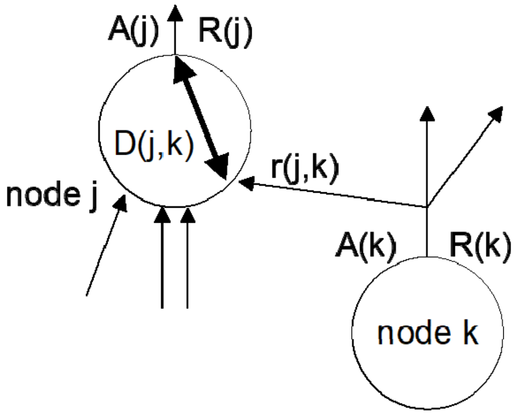
D(j,k): delay of node j from input k

$A(j) = \max_{k \in FI(j)} [A(k) + D(j,k)]$

$r(j,k) = R(j) - D(j,k)$

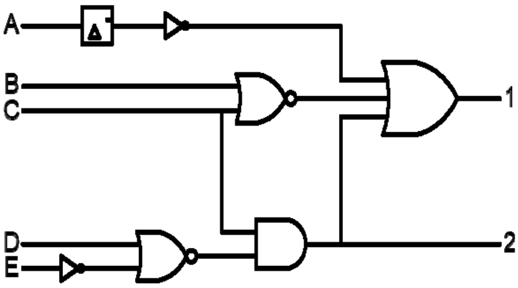
$R(k) = \min_{j \in FO(k)} [r(j,k)]$

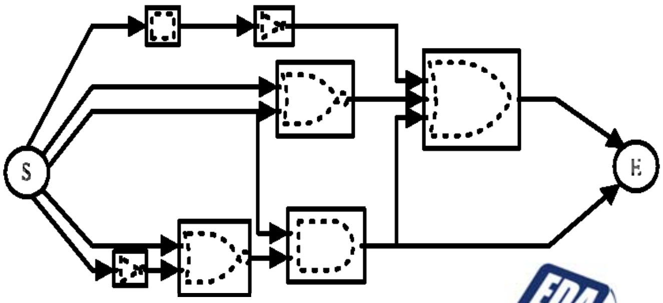
$S(k) = R(k) - A(k)$



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* $\in$V → signal flow
- Adjacency list : *Adj[u]* = (v_1, …, v_k)
  - (*u* ,v_i) $\in$E *i* = 1 , …, *k*
- Number of input arcs to *u*: PredCount[u]



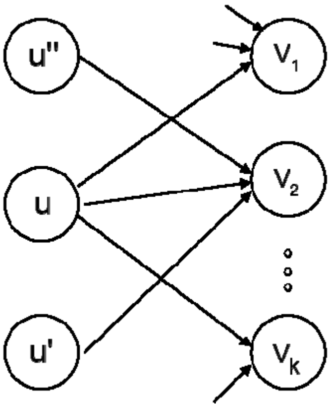


Longest and Shortest Path

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

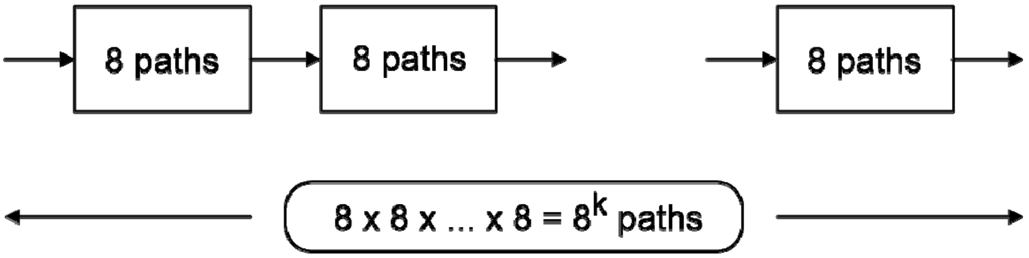
Update Successors[u]

```
for each vertex v $\in$ Adj[u] do
    if A[v] < A[u] + $\Delta$[u]   // longest
        then A[v] $\leftarrow$ A[u] + $\Delta$ [u]
        LP[v] $\leftarrow$ u fi
    if a[v] >a[u] + $\delta$[u]   // shortest
        then a[v] $\leftarrow$ a[u] + $\delta$[u]
        SP[v] $\leftarrow$ u fi
```



Path Enumeration

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



8 paths

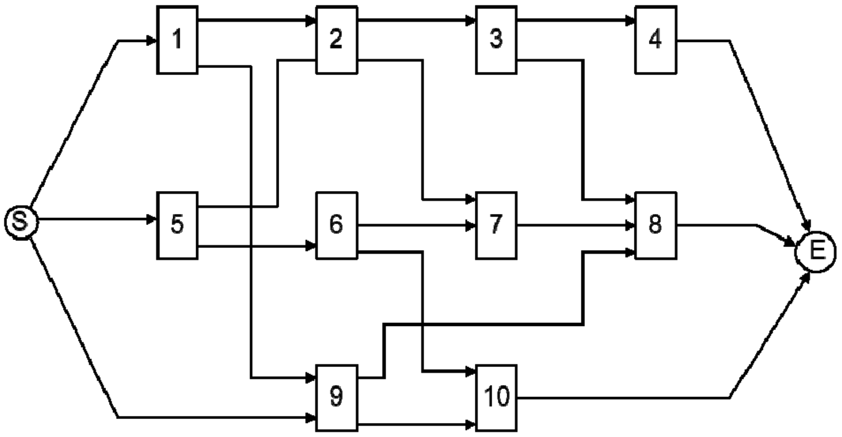
8 paths

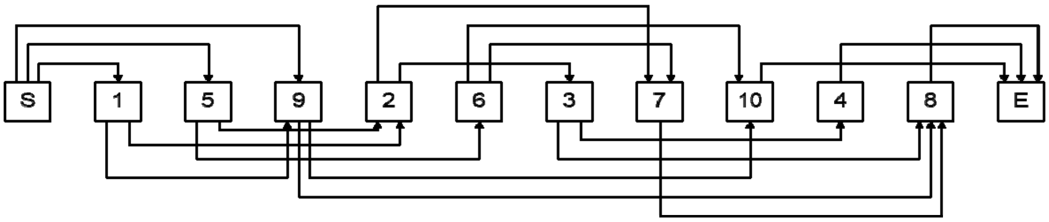
8 paths

8 x 8 x ... x 8 = 8^k paths

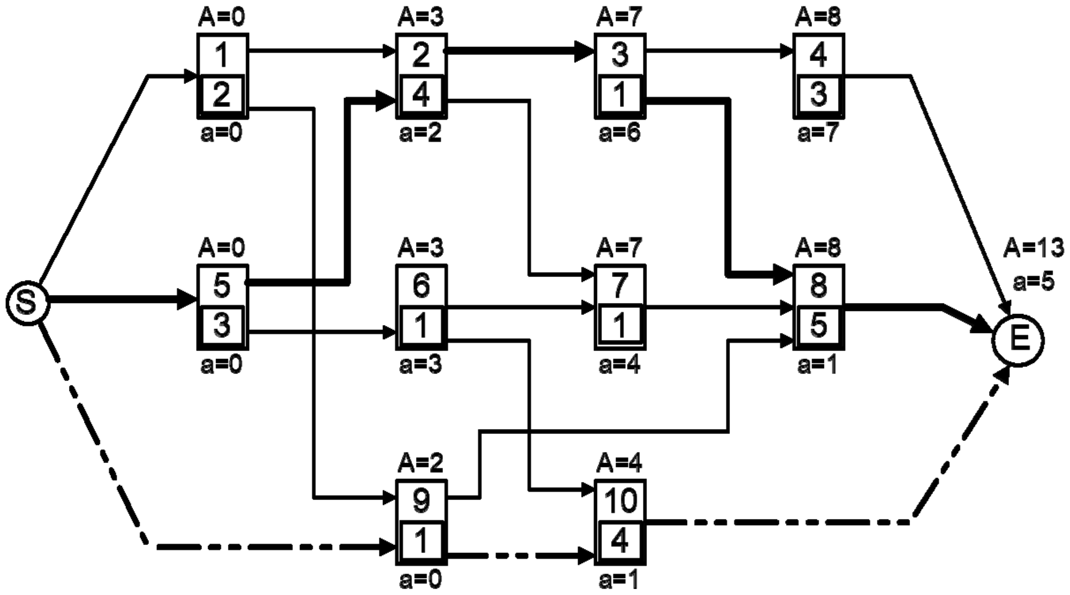
- Topological sort is proposed to solve this problem
  - Linear ordering of all the vertices in the DG such that if $(u, v) \in$ DG the u appear before v in the ordering

Delay Graph and Topological Sort





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

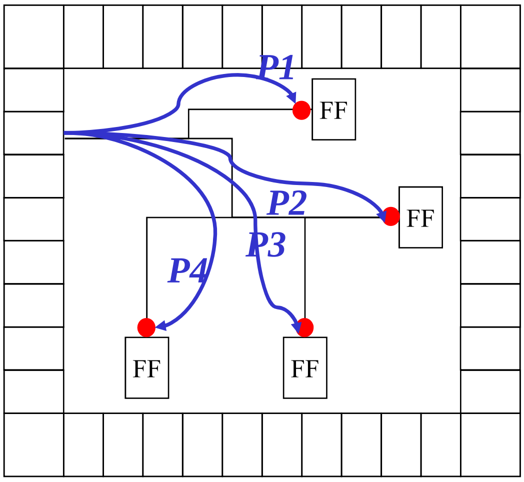
```
Point                     Fanout    Incr     Path
-------------------------------------------------
  ......
U19/Z (INB)                         0.38     50.76 r
n397 (net)                   8      0.00     50.76 r
data_tri[5]/Z (BTD)                 0.37     51.13 f
data[5] (net)                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

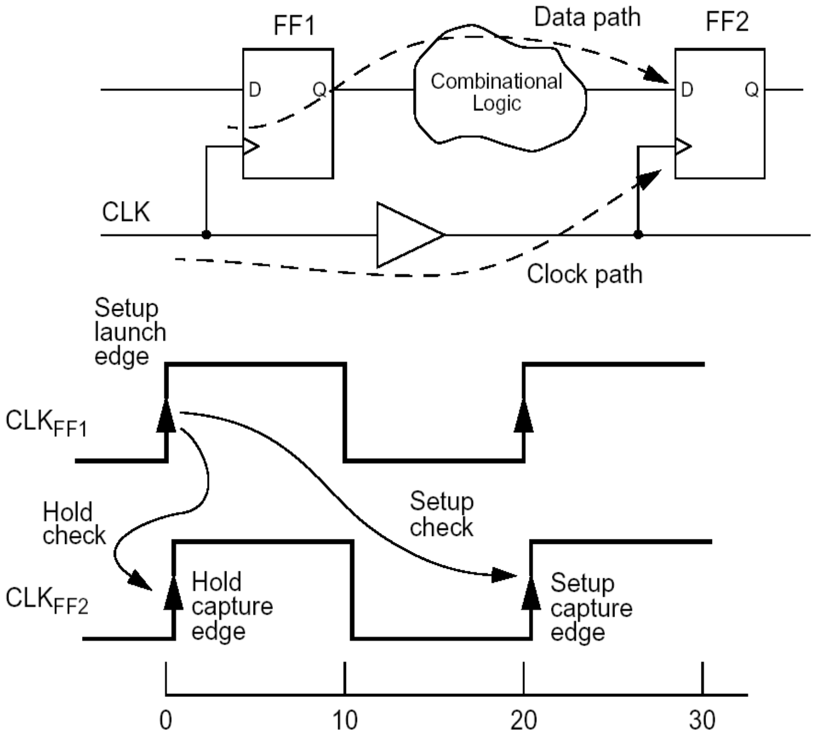
- The maximum difference between the arrival of clock signals at sequential cells in one clock domain or between domains



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

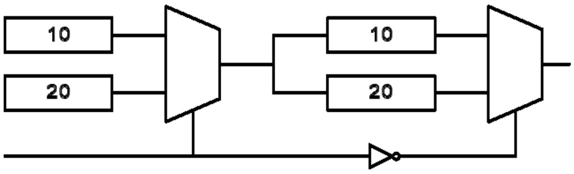
uncertainty
= 1.3 – 0.5
= 0.8ns

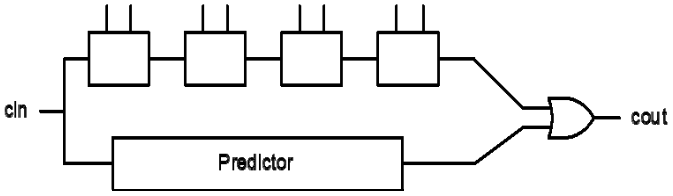
Setup and Hold Checking for FFs



The False Path Problem

- Pattern-independent analysis calculates the delay of logic gates without regard to their function
  - 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:



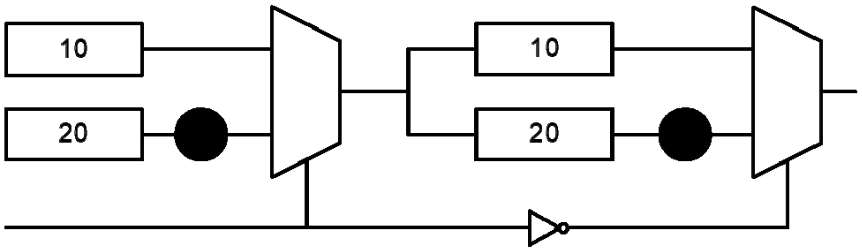


* Longest true path
has delay 30

* Carry bypass adder
skip the critical path

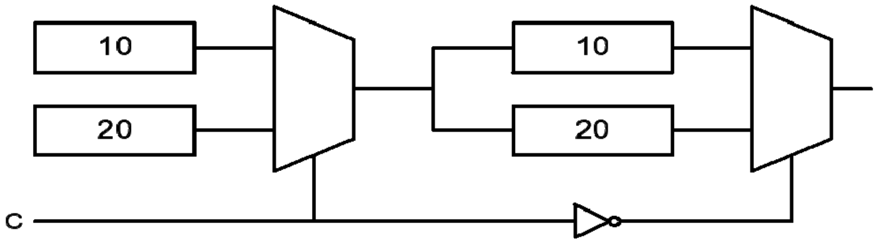
Ad-Hoc Methods

- **Path blocking**: user provides a list of nodes, or pair of nodes, that lie on paths of no interest
  - 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



Path Sensitization

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

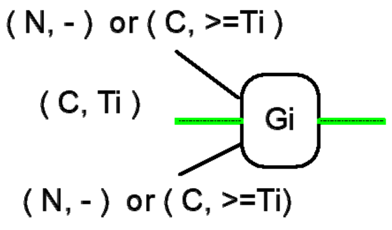
- Path P = (I , G1 , G2 , … , Gn , O) is sensitizable if there is at least one vector which sensitizes the path

Path Sensitization

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

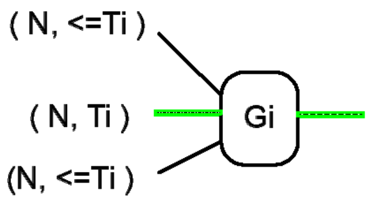
Rule 1) :

earliest controlling



Rule 2) :

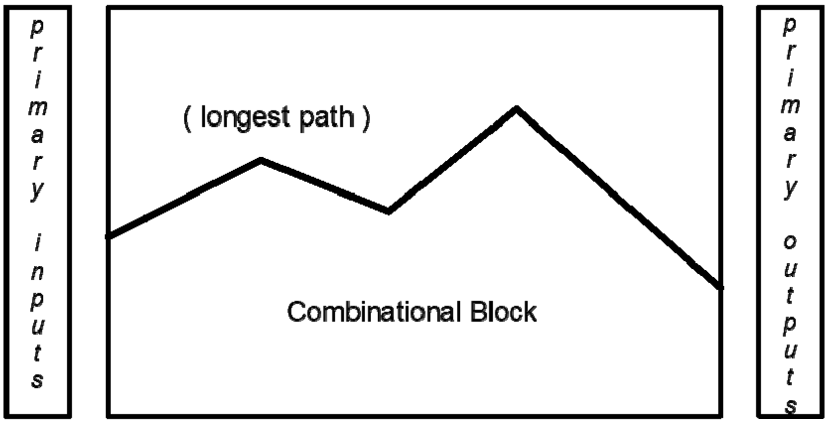
latest non-controlling



(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***
  - The longest sensitizable path can be much shorter than the longest path in a complex circuit
  - Accuracy increases at the expense of computing efficiency



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

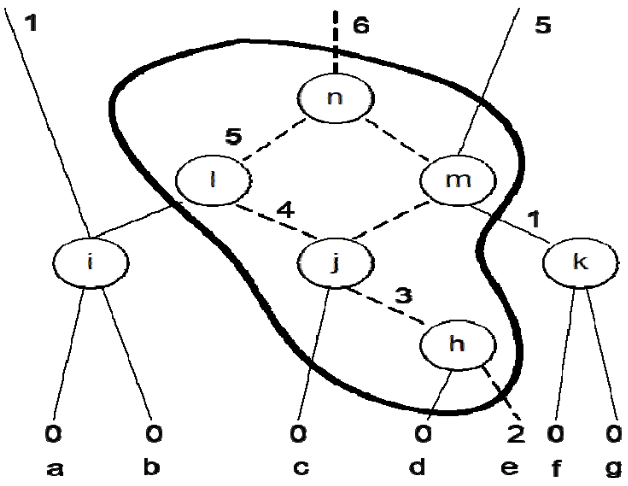
Restructuring Algorithm

```
While (circuit timing improves ) do
        select regions to transform
        collapse the selected region
        resynthesize for better timing
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*
- 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
  - 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*
  - 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*

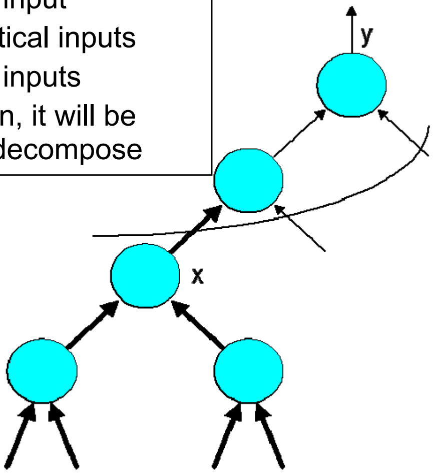
The Weight of Each Node

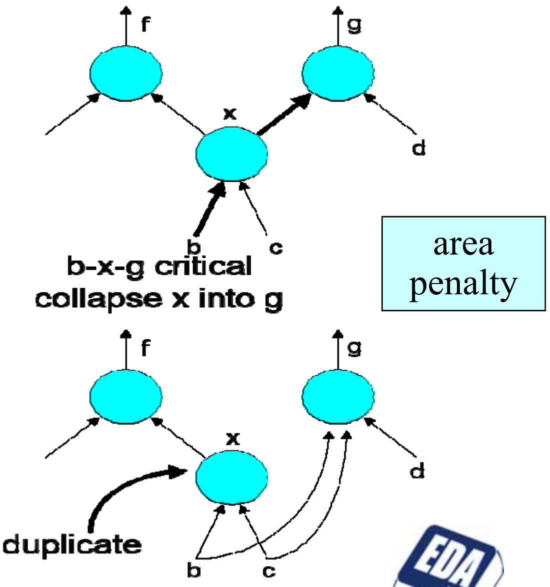
$$W_x^t(d) = \frac{|\{y \in N(d) \mid Sy \le \varepsilon\}|}{|N(d)|}$$

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

- 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

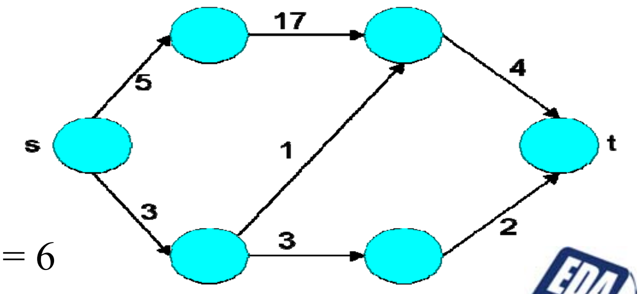
Maximum Network Flow

- A network *$N=(s, t, V, E, b)$* is a diagram (V, E) together with a source $s \in V$ and a sink $t \in V$ with bound (capacity), $b(u,v) \in Z^+$ 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

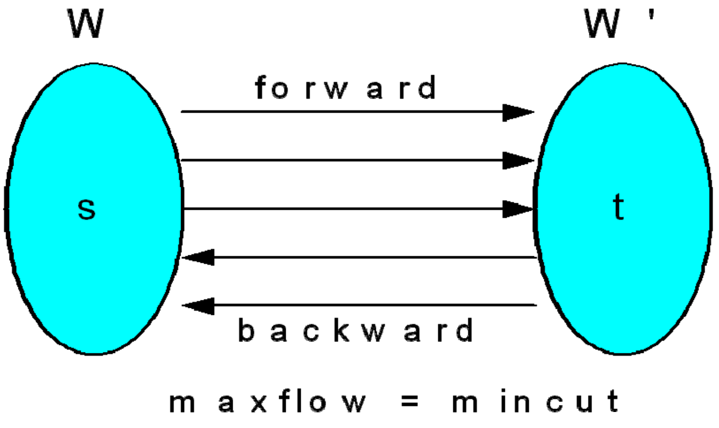


Max-Flow & Min-Cut

- An s-t cut is a partition (W,W’) of the nodes of V into sets W and W’ such that $s \in W$ and $t \in W’$.
- 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 !!

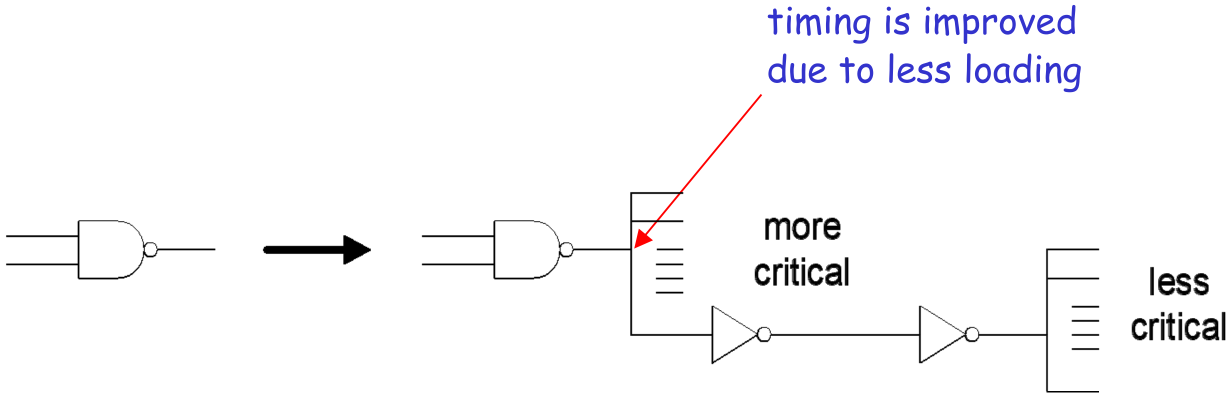


Timing Optimization Techniques (1/8)

- 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

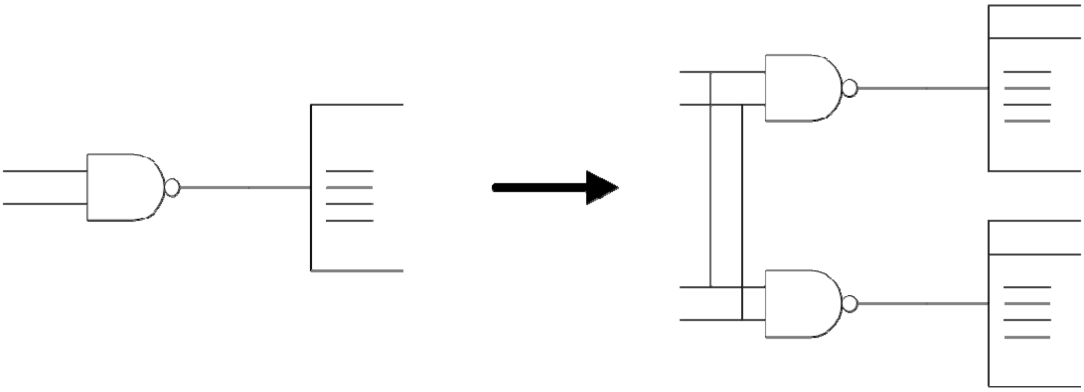
Timing Optimization Techniques (2/8)

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



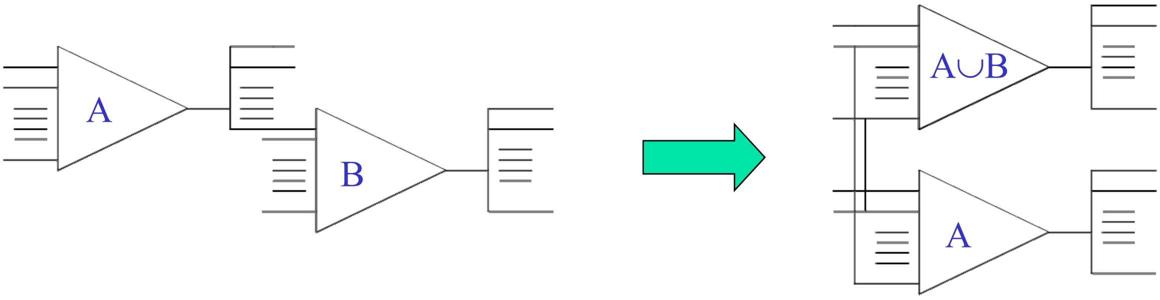
Timing Optimization Techniques (3/8)

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



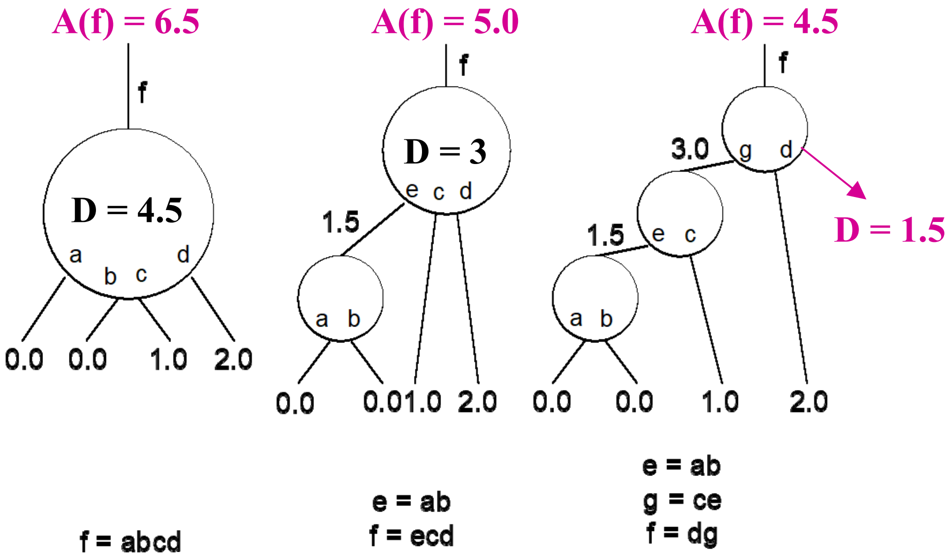
Timing Optimization Techniques (4/8)

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



Timing Optimization Techniques (5/8)

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



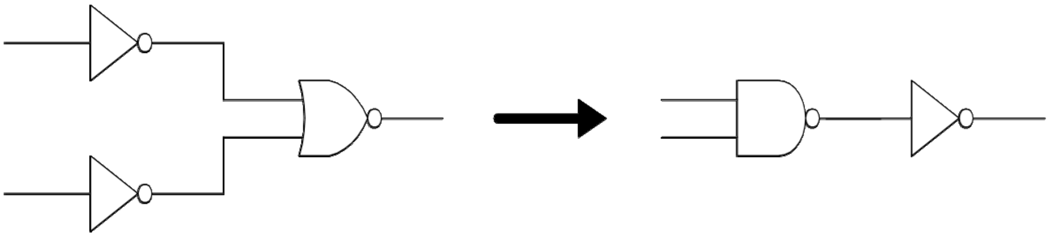
f = abcd

e = ab
f = ecd

e = ab
g = ce
f = dg

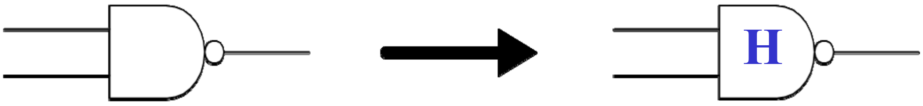
Timing Optimization Techniques (6/8)

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



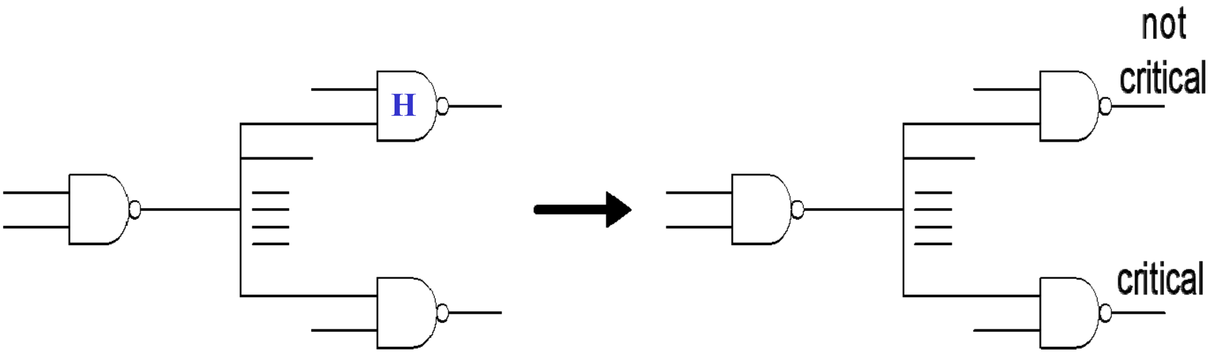
Timing Optimization Techniques (7/8)

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



Timing Optimization Techniques (8/8)

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

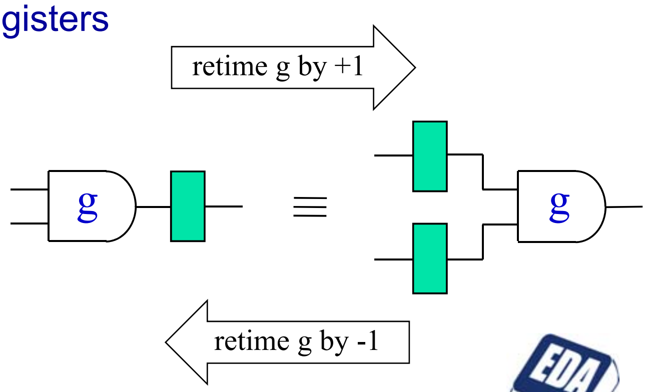


Outline

- 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

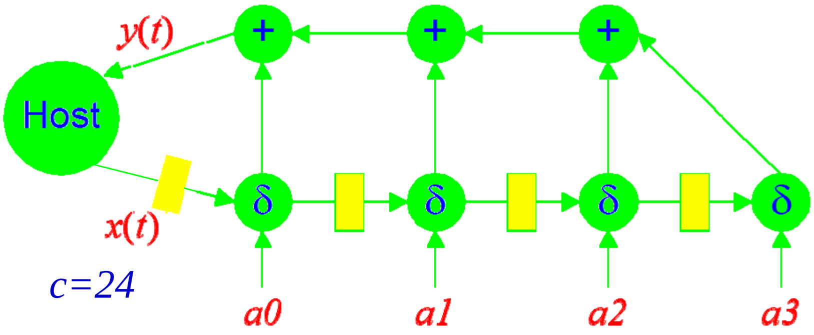
Retiming

- 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
  - $O(|V|^3 \, lg \, |V|)$

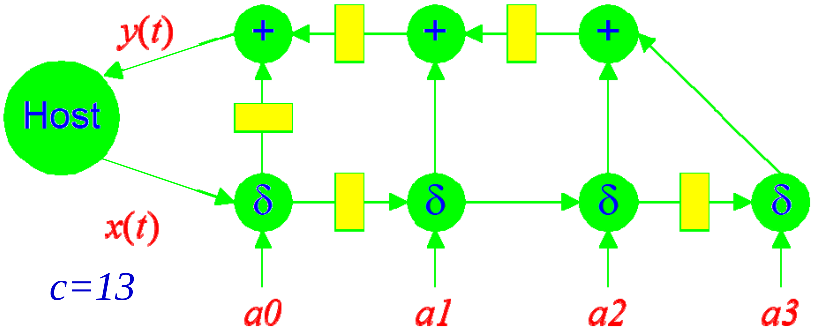


Example: Digital Correlator

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



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

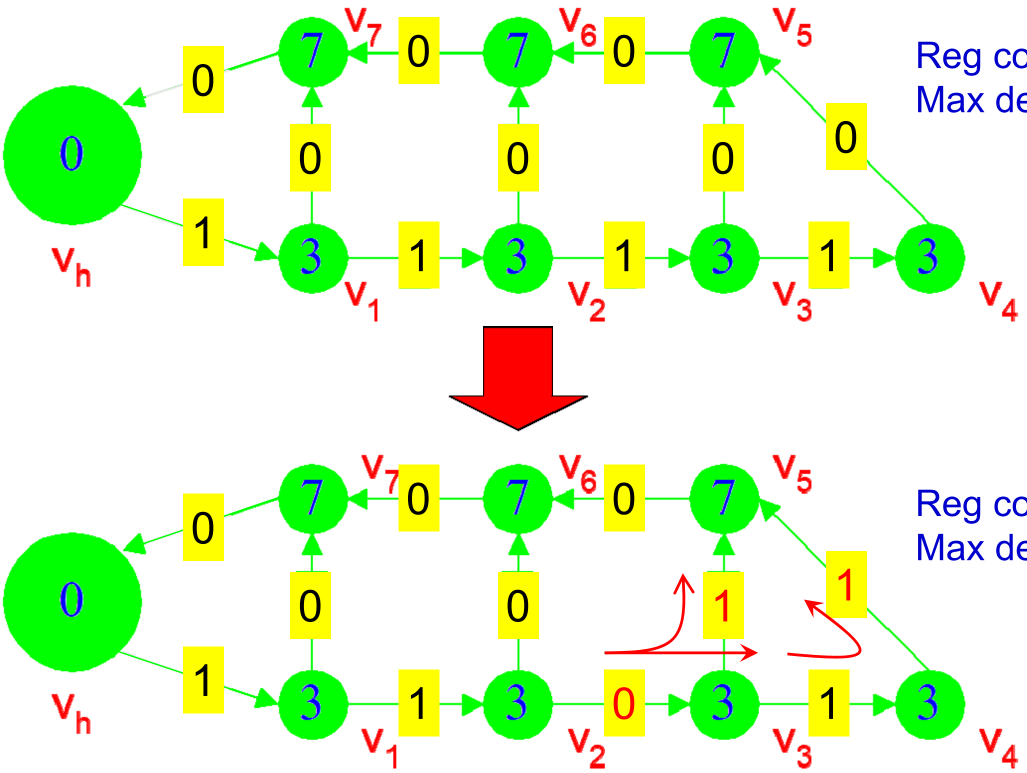


- Delays:
  adder=7,
  comparator=3,
  host=0

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
- W1: The register count $w(e)$ is non-negative for each edge e
- W2: In any directed cycle, there is some edge with positive register count → no combinational loops

Example: Relocating Registers



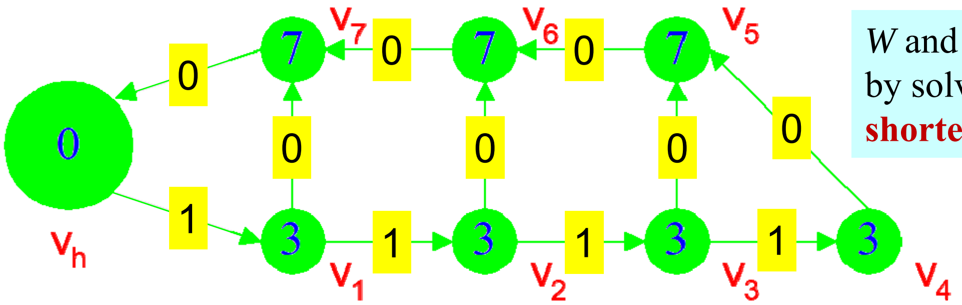
Reg count = 4
Max delay = 24

Reg count = 5
Max delay = 21

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

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 $\Phi(G_r) \leq c$ if and only if
  1. $r(v_h)=0$
  2. $r(u)-r(v) \leq w(e)$ for every edge $e(u,v)$
     - The move-out registers should not exceed the original registers
  3. $r(u)-r(v) \leq w(u,v)-1$ 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*

- Solve the integer linear programming problem
  - Bellman-Ford method in $O(|V|^3)$
- The set of *r*'s determine new positions of the registers

Min-Delay Retiming

- Classical algorithm to find a legal retiming r, such that the cycle time c is minimized

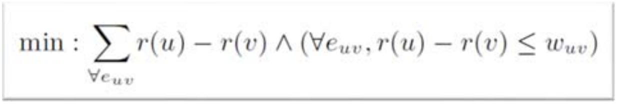
  1. Compute *W* and *D*
  2. Sort the elements in the range of *D*
  3. Binary search the minimum achievable clock period by applying Bellman-Ford algorithm
  4. Derive the *r*(*v*) from the minimum achievable clock period found in Step 3

- 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



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

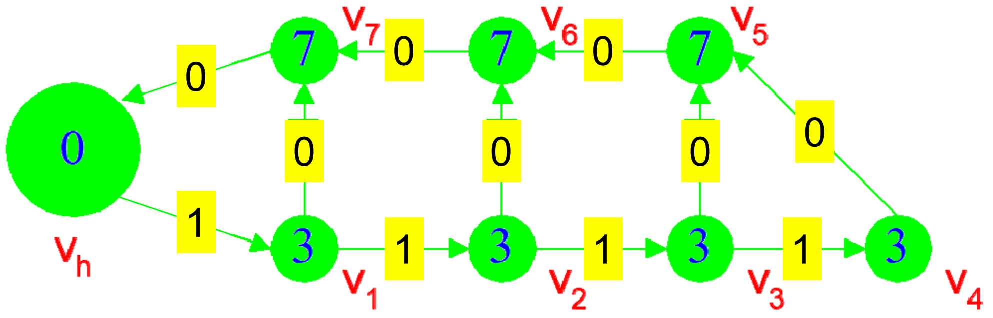
- Goldberg [3]: push-relabel method
  - Complexity = O(V^2 log(VC))
- A. Hurst et al. [2]: maximum network flow problem
  - Complexity = $O(R^2E)$
- Recent approach [4]: minimize area under a specific target clock period
  - Min-area approach [2] + min-delay retiming

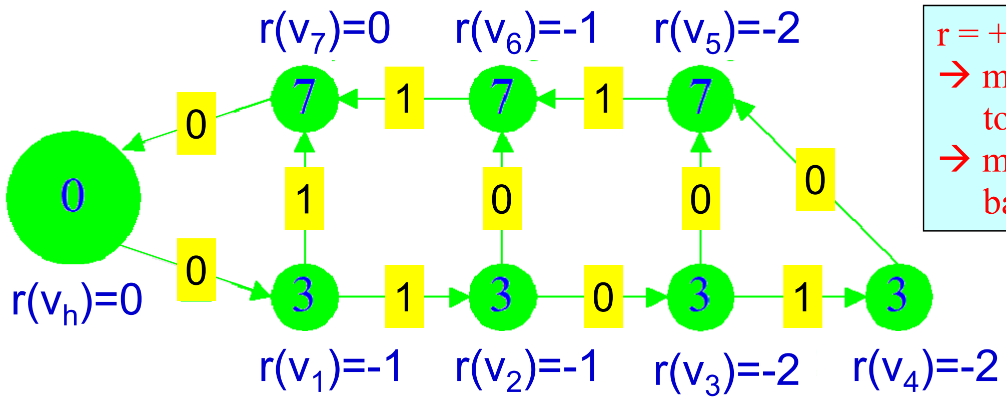
[2] A. P. Hurst, A. Mishchenko, and R. K. Brayton, "Fast minimum-register retiming via binary maximum-flow," in Formal Methods in Computer Aided Design, IEEE, pp. 181–187, 2007.

[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





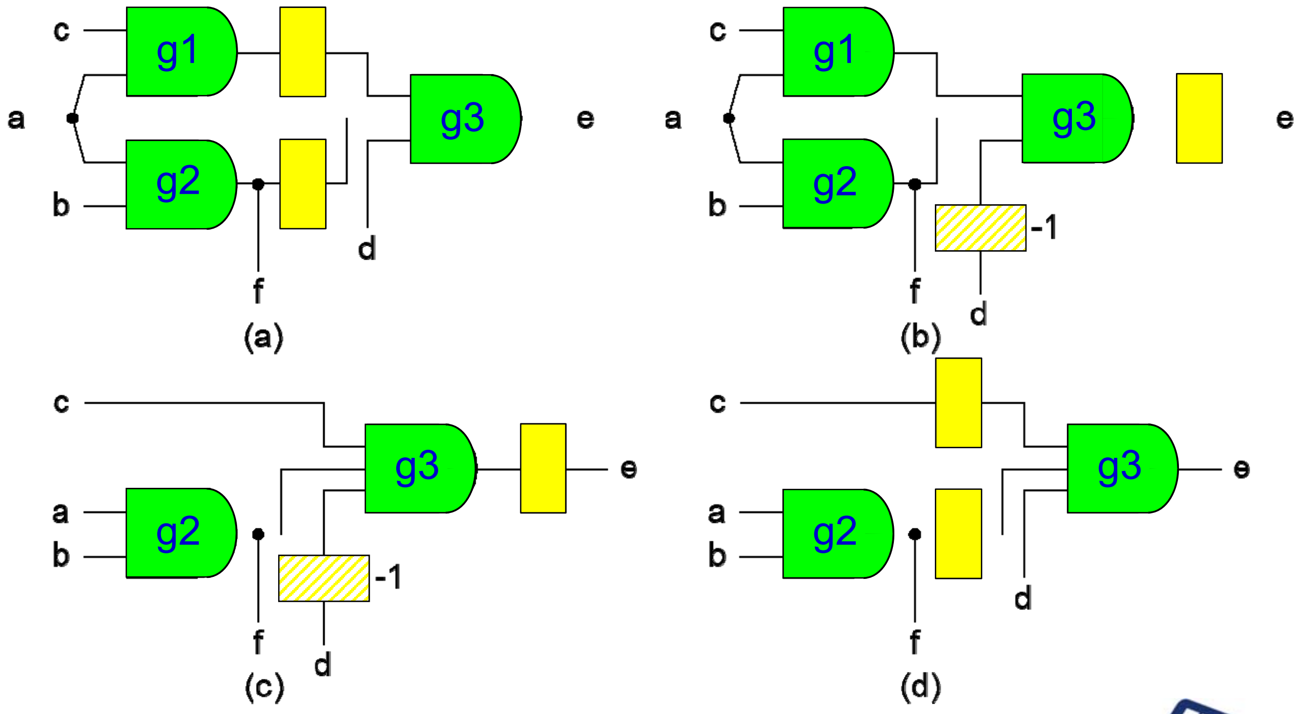
r = +1:
→ moving a gate to next cycle
→ moving a register backward

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 , $1 \leq i \leq m$, $1 \leq j \leq$ 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:

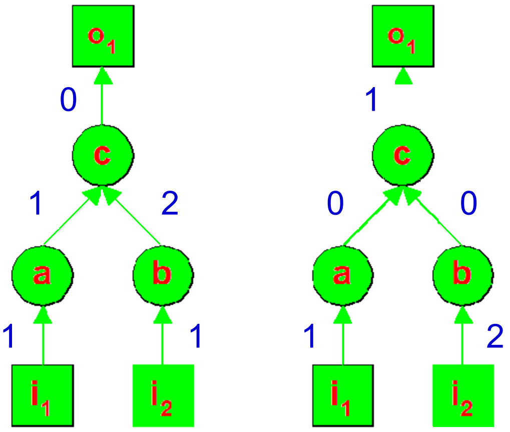
  $W_{1,1}=2,\ W_{2,1}=3,$

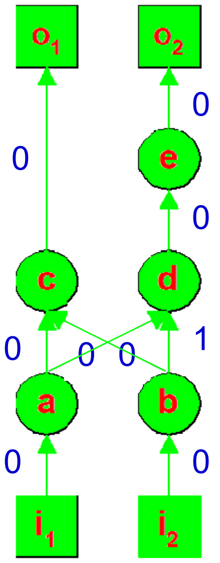
  $\Rightarrow \alpha_1=1,\ \alpha_2=2,\ \beta_1=1$

- Example 2:

  $W_{1,1}=0, W_{1,2}=0, W_{2,1}=0,\ W_{2,2}=1$

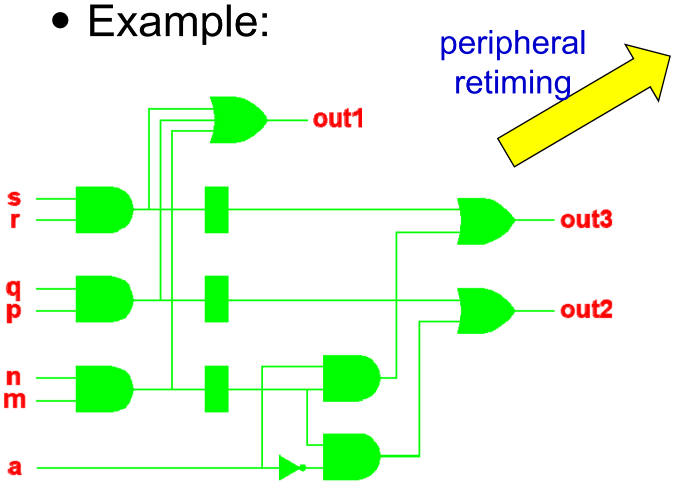
  $\Rightarrow$ no solution

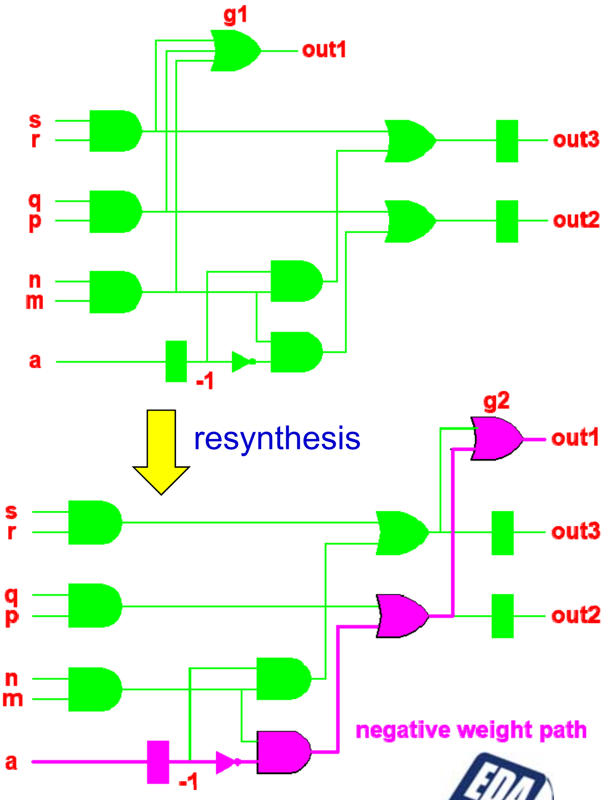




Legal Resynthesis Operation

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





Challenges of Retiming Techniques

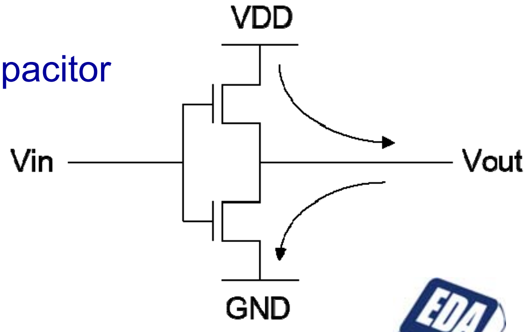
- Retiming complexity
  - Computing complexity is often too high
- Performance improvements due to retiming
  - Design performance of the physical netlist is not guaranteed due to the lack of placement info
  - To forecast whether the design performance could be improved becomes extremely important
- 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

$$P = ½*C*V_{dd}^2*D$$

- 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

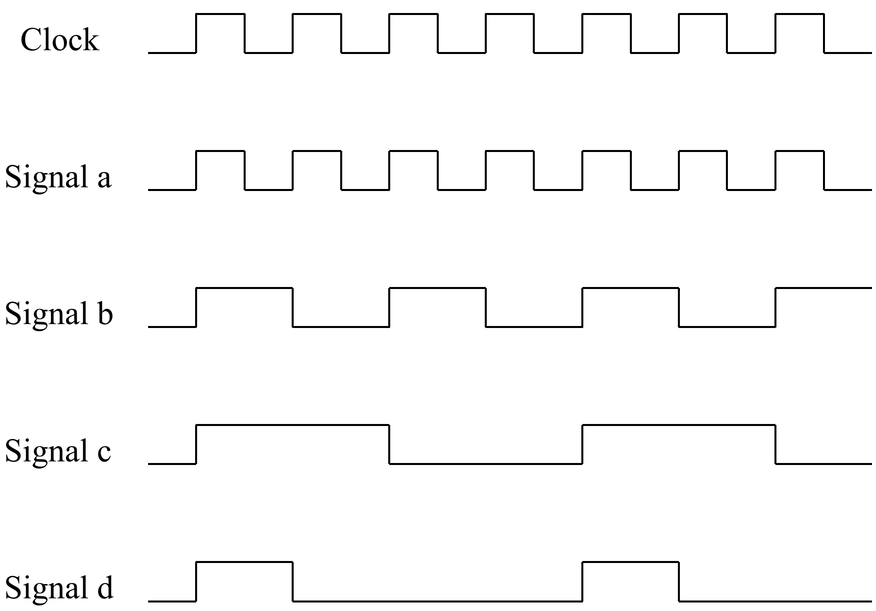
- Definition: The transition density Dx of a logic signal x(t), t∈(-∞ , ∞) , is defined as

$$D_X \equiv \lim_{T\to\infty} \frac{n_X(T)}{T\cdot f_c}$$

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



Pa = 0.5 Da = 1

Pb = 0.5 Db = 0.5

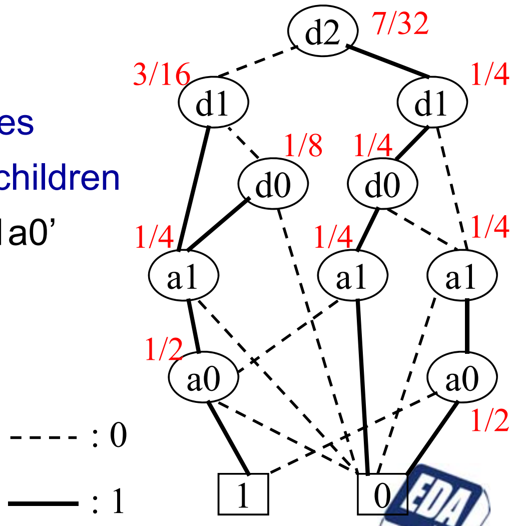
Pc = 0.5 Dc = 0.25

Pd = 0.25 Dd = 0.25

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

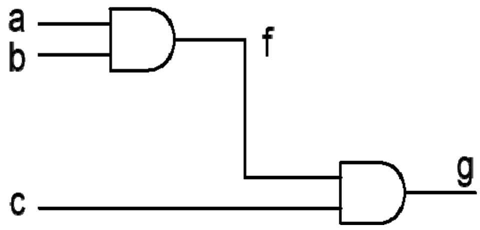
  p(F) = 7/32 = 0.21875



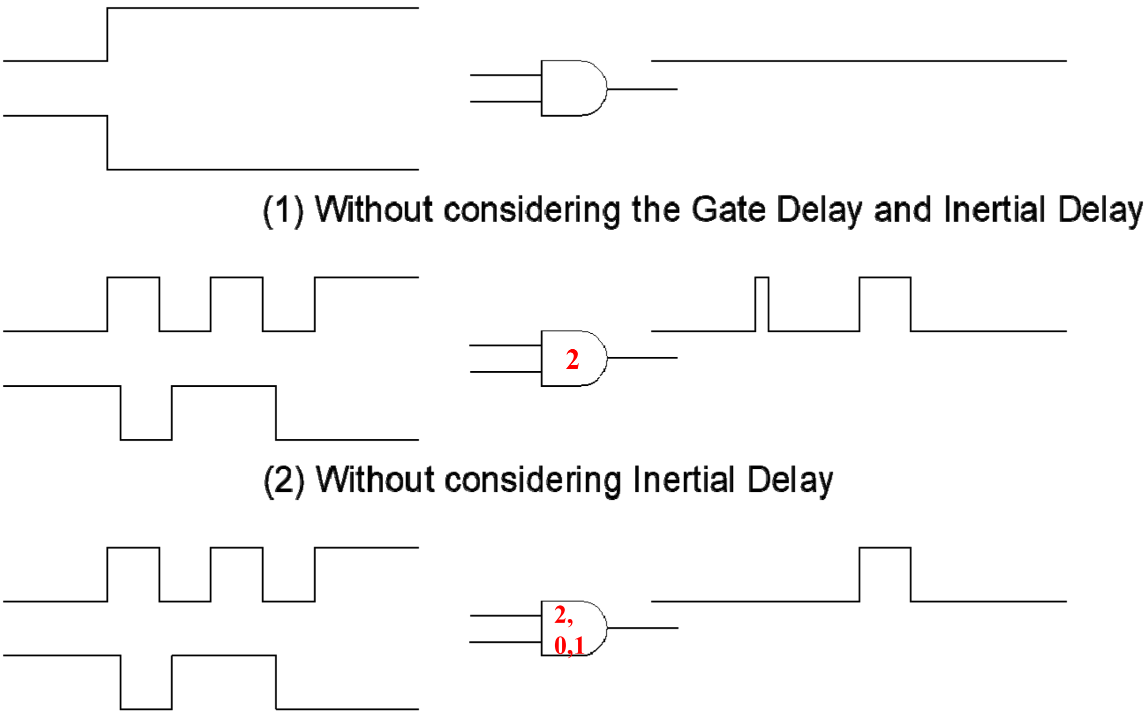
The Calculation of Transition Density

- 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

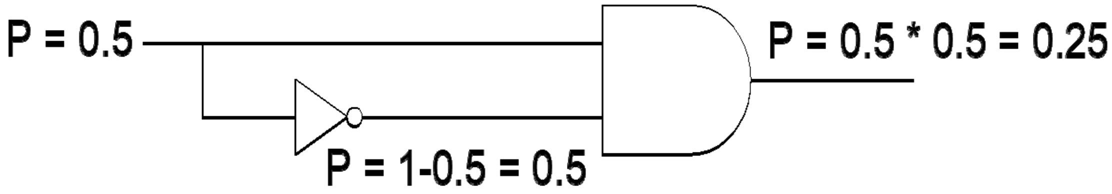


(1) Without considering the Gate Delay and Inertial Delay

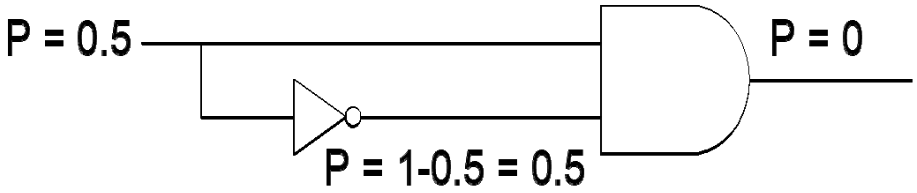
(2) Without considering Inertial Delay

(3) Practical condition

The Problem of Spatial Correlation



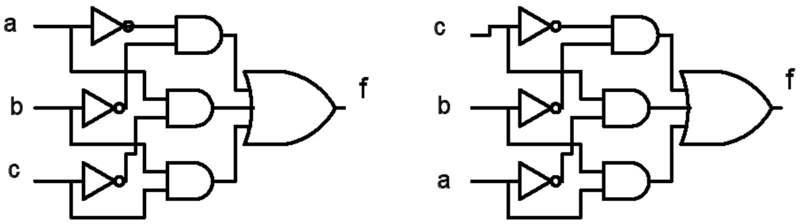
(a) Without considering Spatial Correlation



(b) Practical condition

Logic Minimization for Low Power (1/2)

- Consider an example:



c \ ab	00	01	11	10
0	1		1	1
1	1	1	1	

f = a'b' + ac' + bc

P = 108.7 μW

(a)

c \ ab	00	01	11	10
0	1		1	1
1	1	1	1	

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

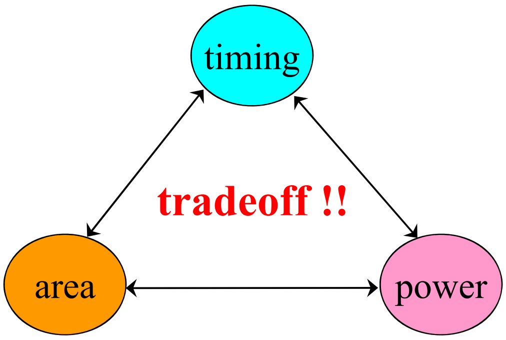
P = 115.5 μW

(b)

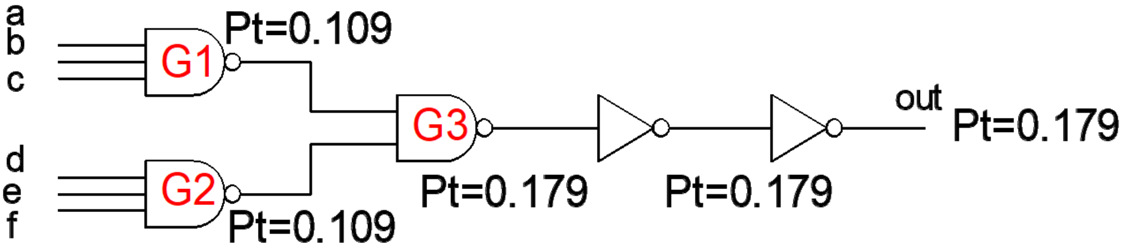
- Different choices of the covers may result in different power consumption

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)

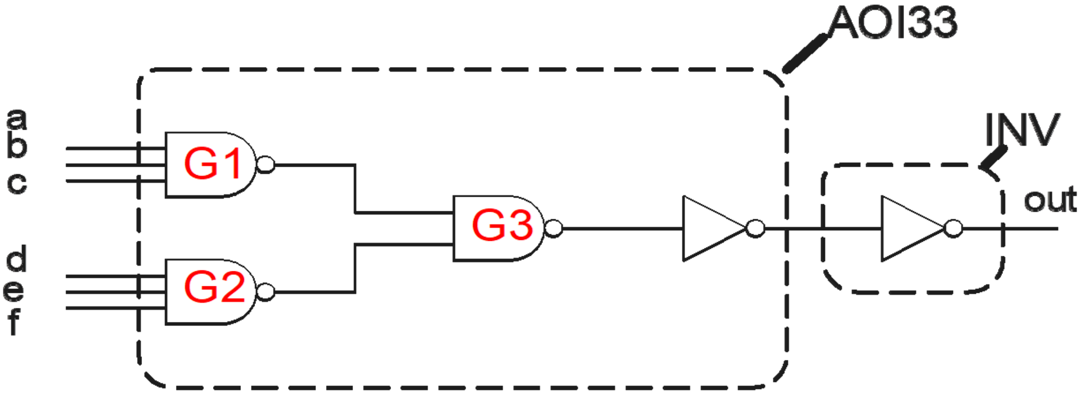


(a) Circuit to be mapped

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

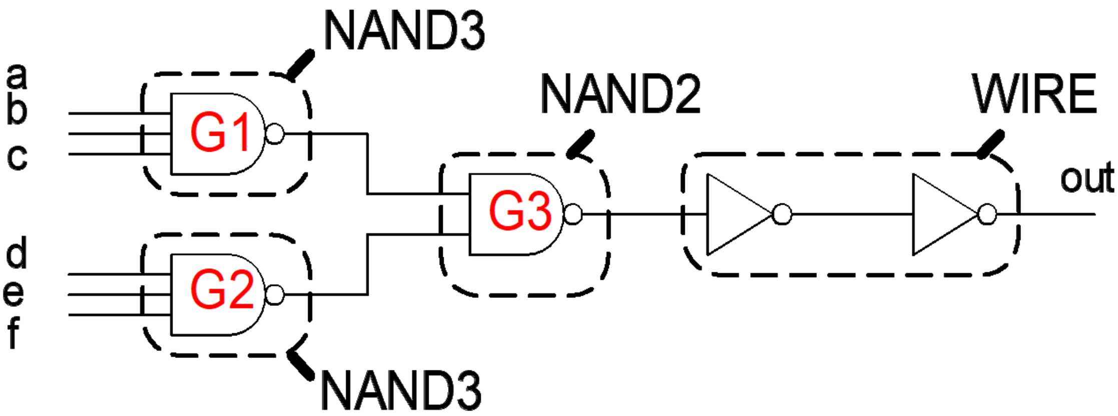
Technology Mapping for Low Power (2/3)



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