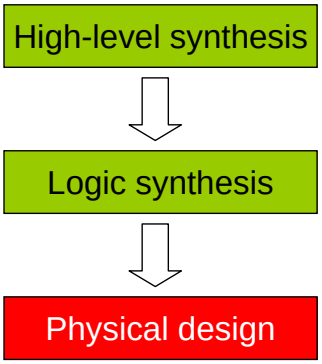
Introduction to Electronic Design Automation

Jie-Hong Roland Jiang
江介宏

Department of Electrical Engineering
National Taiwan University

Spring 2011

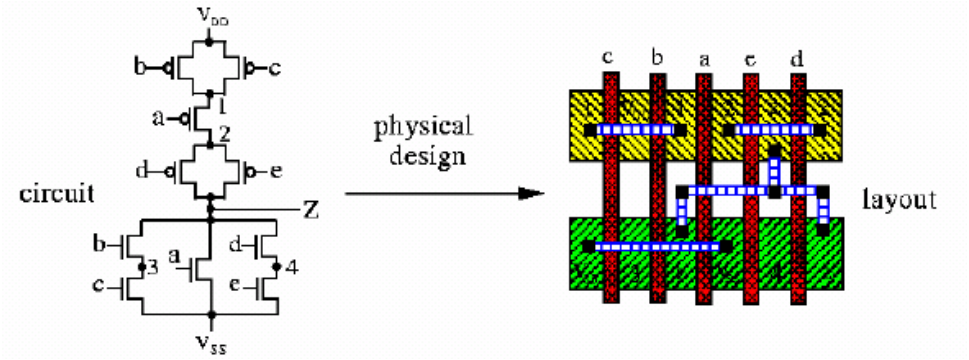
Physical Design



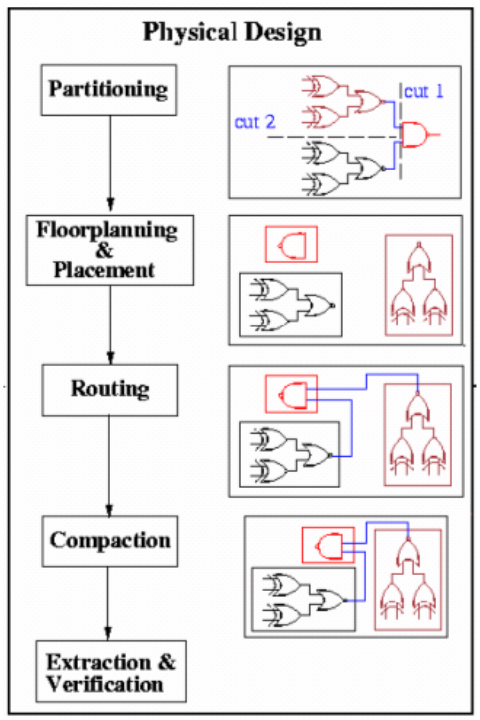
Slides are by Courtesy of Prof. Y.-W. Chang

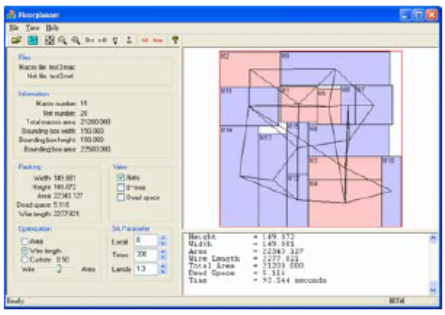
Physical Design

- Physical design converts a circuit description into a geometric description.
- The description is used to manufacture a chip.
- Physical design cycle:
  1. Logic partitioning
  2. Floorplanning and placement
  3. Routing
  4. Compaction
- Others: circuit extraction, timing verification and design rule checking



Physical Design Flow





B*-tree based floorplanning system

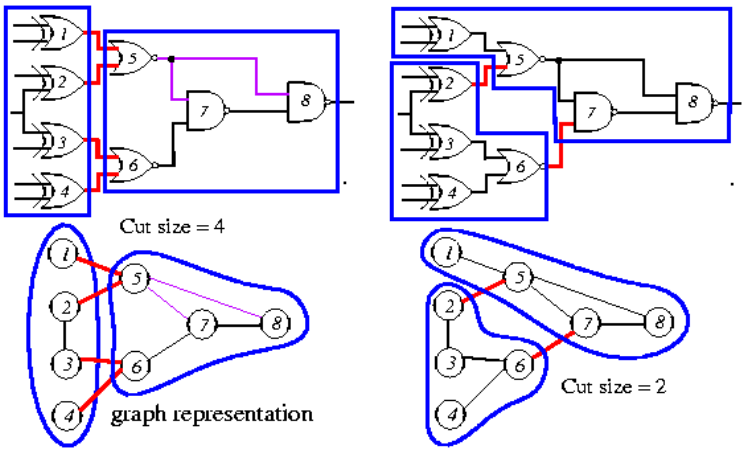
A routing system

Outline

- Partitioning
- Floorplanning
- Placement
- Routing
- Compaction

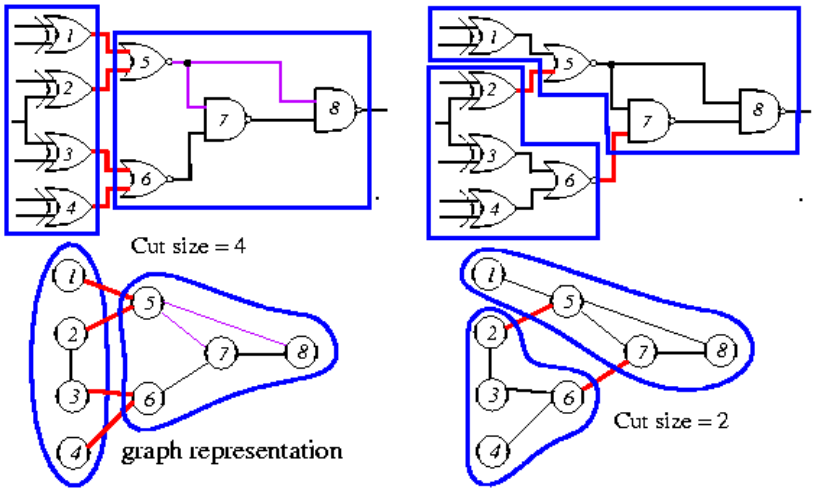
Circuit Partitioning

- Course contents:
  - Kernighang-Lin partitioning algorithm
  - Simulated annealing based partitioning algorithm



Circuit Partitioning

- **Objective:** Partition a circuit into parts such that every component is within a prescribed range and the # of connections among the components is minimized.
  - More constraints are possible for some applications.
- Cutset? Cut size? Size of a component?



Problem Definition: Partitioning

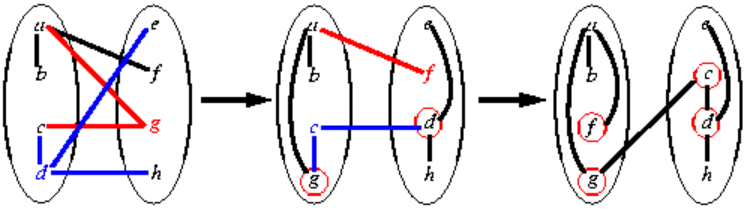
- ***k*-way partitioning:** Given a graph $G(V, E)$, where each vertex $v \in V$ has a **size** $s(v)$ and each edge $e \in E$ has a **weight** $w(e)$, the problem is to divide the set V into k disjoint subsets $V_1, V_2, \ldots, V_k$, such that an objective function is optimized, subject to certain constraints.
- **Bounded size constraint:** The size of the i-th subset is bounded by B_i (i.e., $\sum_{v \in V_i} s(v) \leq B_i$).
  - Is the partition balanced?
- **Min-cut cost between two subsets:** Minimize $\sum_{\forall e=(u,v) \wedge p(u) \neq p(v)} w(e)$, where $p(u)$ is the partition # of node u.
- The 2-way, balanced partitioning problem is NP-complete, even in its simple form with identical vertex sizes and unit edge weights.

Kernighan-Lin Algorithm

- Kernighan and Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, Feb. 1970.
- An **iterative**, **2-way**, **balanced** partitioning (bi-sectioning) heuristic.
- Till the cut size keeps decreasing
  - Vertex pairs which give the largest decrease or the smallest increase in cut size are exchanged.
  - These vertices are then **locked** (and thus are prohibited from participating in any further exchanges).
  - This process continues until all the vertices are locked.
  - Find the set with the largest partial sum for swapping.
  - Unlock all vertices.

K-L Algorithm: A Simple Example

- Each edge has a unit weight.



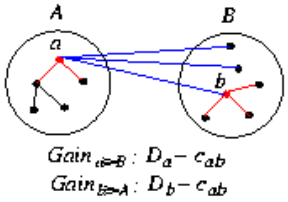
Step #	Vertex pair	Cost reduction	Cut cost
0	-	0	5
1	{d, g}	3	2
2	{c, f}	1	1
3	{b, h}	-2	3
4	{a, e}	-2	5

- Questions: How to compute cost reduction? What pairs to be swapped?
  - Consider the change of internal & external connections.

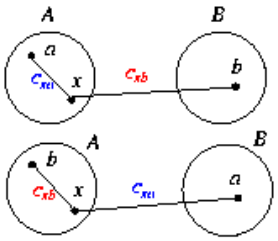
Properties

- Two sets A and B such that $|A| = n = |B|$ and $A \cap B = \emptyset$.
- **External cost** of $a \in A$: $E_a = \sum_{v \in B} c_{av}$.
- **Internal cost** of $a \in A$: $I_a = \sum_{v \in A} c_{av}$.
- D-value of a vertex a: $D_a = E_a - I_a$ (cost reduction for moving a).
- Cost reduction (gain) for swapping a and b: $g_{ab} = D_a + D_b - 2c_{ab}$.
- If $a \in A$ and $b \in B$ are interchanged, then the new D-values, D', are given by

$$D'_x = D_x + 2c_{xa} - 2c_{xb}, \forall x \in A - \{a\}$$
$$D'_y = D_y + 2c_{yb} - 2c_{ya}, \forall y \in B - \{b\}.$$



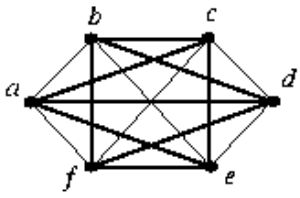
Internal cost vs. External cost



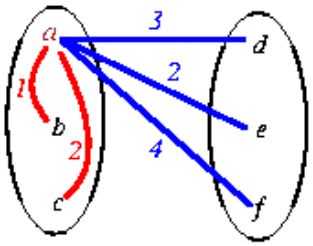
before swap	*after swap*	ΔC
$-c_{xa}$	$+c_{xa}$	$+2c_{xa}$
$+c_{xb}$	$-c_{xb}$	$-2c_{xb}$

updating D–values

A Weighted Example



	a	*b*	*c*	*d*	*e*	*f*
a	0	1	2	3	2	4
b	1	0	1	4	2	1
c	2	1	0	3	2	1
d	3	4	3	0	4	3
e	2	2	2	4	0	2
f	4	1	1	3	2	0



costs associated with a

Initial cut cost = (3+2+4)+(4+2+1)+(3+2+1) = 22

- Iteration 1

$$I_a = 1 + 2 = 3; \quad E_a = 3 + 2 + 4 = 9; \quad D_a = E_a - I_a = 9 - 3 = 6$$
$$I_b = 1 + 1 = 2; \quad E_b = 4 + 2 + 1 = 7; \quad D_b = E_b - I_b = 7 - 2 = 5$$
$$I_c = 2 + 1 = 3; \quad E_c = 3 + 2 + 1 = 6; \quad D_c = E_c - I_c = 6 - 3 = 3$$
$$I_d = 4 + 3 = 7; \quad E_d = 3 + 4 + 3 = 10; \quad D_d = E_d - I_d = 10 - 7 = 3$$
$$I_e = 4 + 2 = 6; \quad E_e = 2 + 2 + 2 = 6; \quad D_e = E_e - I_e = 6 - 6 = 0$$
$$I_f = 3 + 2 = 5; \quad E_f = 4 + 1 + 1 = 6; \quad D_f = E_f - I_f = 6 - 5 = 1$$

A Weighted Example (cont'd)

- Iteration 1:

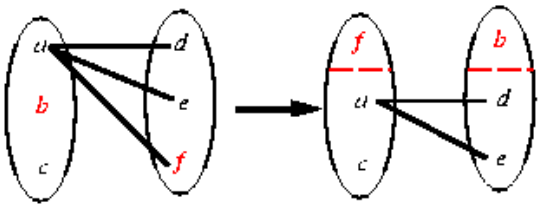
$$I_a = 1+2 = 3; \quad E_a = 3+2+4 = 9; \quad D_a = E_a - I_a = 9-3 = 6$$
$$I_b = 1+1 = 2; \quad E_b = 4+2+1 = 7; \quad D_b = E_b - I_b = 7-2 = 5$$
$$I_c = 2+1 = 3; \quad E_c = 3+2+1 = 6; \quad D_c = E_c - I_c = 6-3 = 3$$
$$I_d = 4+3 = 7; \quad E_d = 3+4+3 = 10; \quad D_d = E_d - I_d = 10-7 = 3$$
$$I_e = 4+2 = 6; \quad E_e = 2+2+2 = 6; \quad D_e = E_e - I_e = 6-6 = 0$$
$$I_f = 3+2 = 5; \quad E_f = 4+1+1 = 6; \quad D_f = E_f - I_f = 6-5 = 1$$

- $g_{xy} = D_x + D_y - 2c_{xy}$.

$$g_{ad} = D_a + D_d - 2c_{ad} = 6+3-2\times 3 = 3$$
$$g_{ae} = 6+0-2\times 2 = 2$$
$$g_{af} = 6+1-2\times 4 = -1$$
$$g_{bd} = 5+3-2\times 4 = 0$$
$$g_{be} = 5+0-2\times 2 = 1$$
$$g_{bf} = 5+1-2\times 1 = 4 \ (maximum) \quad (\hat{g_1} = 4)$$
$$g_{cd} = 3+3-2\times 3 = 0$$
$$g_{ce} = 3+0-2\times 2 = -1$$
$$g_{cf} = 3+1-2\times 1 = 2$$

- Swap *b* and *f*.

A Weighted Example (cont'd)



- $D'_x = D_x + 2\,c_{xp} - 2\,c_{xq}$, $\forall\, x \in A - \{p\}$ (swap p and q, $p \in A$, $q \in B$)

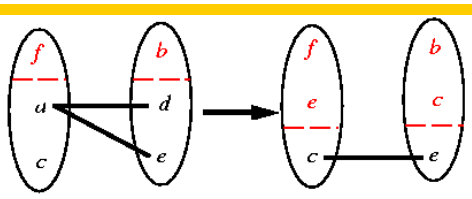
$$D'_a = D_a + 2c_{ab} - 2c_{af} = 6+2\times 1-2\times 4 = 0$$
$$D'_c = D_c + 2c_{cb} - 2c_{cf} = 3+2\times 1-2\times 1 = 3$$
$$D'_d = D_d + 2c_{df} - 2c_{db} = 3+2\times 3-2\times 4 = 1$$
$$D'_e = D_e + 2c_{ef} - 2c_{eb} = 0+2\times 2-2\times 2 = 0$$

- $g_{xy} = D'_x + D'_y - 2c_{xy}$.

$$g_{ad} = D'_a + D'_d - 2c_{ad} = 0+1-2\times 3 = -5$$
$$g_{ae} = D'_a + D'_e - 2c_{ae} = 0+0-2\times 2 = -4$$
$$g_{cd} = D'_c + D'_d - 2c_{cd} = 3+1-2\times 3 = -2$$
$$g_{ce} = D'_c + D'_e - 2c_{ce} = 3+0-2\times 2 = -1 \ (maximum) \quad (\hat{g_2} = -1)$$

- Swap *c* and *e*.

A Weighted Example (cont'd)



- $D''_x = D'_x + 2\,c_{xp} - 2\,c_{xq}$, $\forall\, x \in A - \{p\}$

$$D''_a = D'_a + 2c_{ac} - 2c_{ae} = 0 + 2 \times 2 - 2 \times 2 = 0$$
$$D''_d = D'_d + 2c_{de} - 2c_{dc} = 1 + 2 \times 4 - 2 \times 3 = 3$$

- $g_{xy} = D''_x + D''_y - 2c_{xy}$.

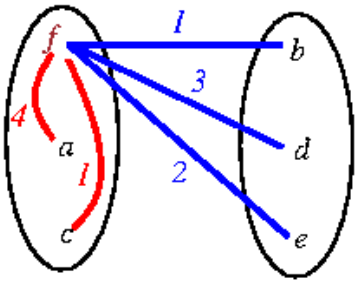
$$g_{ad} = D''_a + D''_d - 2c_{ad} = 0 + 3 - 2 \times 3 = -3 (\hat{g}_3 = -3)$$

  - Note that this step is redundant

- Summary: $\hat{g}_1 = g_{bf} = 4,\ \hat{g}_2 = g_{ce} = -1,\ \hat{g}_3 = g_{ad} = -3.\ (\sum_{i=1}^{n} \hat{g}_i = 0).$
- Largest partial sum $\max \sum_{i=1}^{k} \hat{g}_i = 4$ (k = 1) ⇒ Swap b and f.

A Weighted Example (cont'd)

	a	b	c	d	e	f
a	0	1	2	3	2	4
b	1	0	1	4	2	1
c	2	1	0	3	2	1
d	3	4	3	0	4	3
e	2	2	2	4	0	2
f	4	1	1	3	2	0



Initial cut cost = (1+3+2)+(1+3+2)+(1+3+2) = 18 (22−4)

- Iteration 2: Repeat what we did at Iteration 1 (Initial cost = 22-4 =18).
- Summary: $\hat{g}_1 = g_{ce} = -1,\ \hat{g}_2 = g_{ab} = -3,\ \hat{g}_3 = g_{fd} = 4.$
- Largest partial sum = $\max \sum_{i=1}^{k} \hat{g}_i = 0$ (k = 3) ⇒ Stop!

Kernighan-Lin Algorithm

Algorithm: Kernighan-Lin(G)
Input: $G = (V, E), |V| = 2n$.
Output: Balanced bi-partition A and B with "small" cut cost.

```
1  begin
2  Bipartition G into A and B such that |V_A| = |V_B|, V_A ∩ V_B = ∅,
       and V_A ∪ V_B = V.
3  repeat
4      Compute D_v, ∀v ∈ V.
5      for i = 1 to n do
6          Find a pair of unlocked vertices v_ai ∈ V_A and v_bi ∈ V_B whose
           exchange makes the largest decrease or smallest increase in
           cut cost;
7          Mark v_ai and v_bi as locked, store the gain ĝ_i, and compute
           the new D_v, for all unlocked v ∈ V;
8      Find k, such that G_k = Σ_{i=1}^{k} ĝ_i is maximized;
9      if G_k > 0 then
10         Move v_a1,...,v_ak from V_A to V_B and v_b1,...,v_bk from V_B to V_A;
11     Unlock v, ∀v ∈ V.
12 until G_k ≤ 0;
13 end
```


Time Complexity

- Line 4: Initial computation of D: $O(n^2)$
- Line 5: The **for**-loop: $O(n)$
- The body of the loop: $O(n^2)$.
  - Lines 6--7: Step i takes $(n - i + 1)^2$ time.
- Lines 4--11: Each pass of the repeat loop: $O(n^3)$.
- Suppose the repeat loop terminates after r passes.
- The total running time: $O(rn^3)$.
  - Polynomial-time algorithm?

Extensions of K-L Algorithm

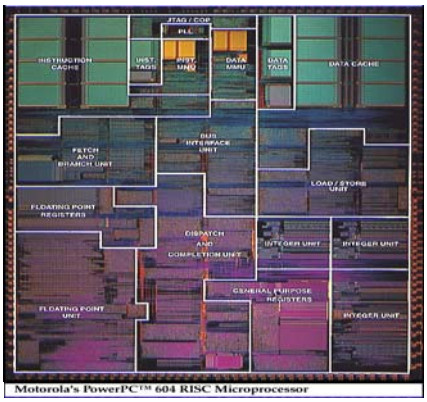
- **Unequal sized subsets** (assume $n_1 < n_2$)
  1. Partition: $|A| = n_1$ and $|B| = n_2$.
  2. Add $n_2 - n_1$ dummy vertices to set A. Dummy vertices have no connections to the original graph.
  3. Apply the Kernighan-Lin algorithm.
  4. Remove all dummy vertices.
- **Unequal sized "vertices"**
  1. Assume that the smallest "vertex'' has unit size.
  2. Replace each vertex of size s with s vertices which are fully connected with edges of infinite weight.
  3. Apply the Kernighan-Lin algorithm.
- ***k*-way partition**
  1. Partition the graph into k equal-sized sets.
  2. Apply the Kernighan-Lin algorithm for each pair of subsets.
  3. Time complexity? Can be reduced by recursive bi-partition.

Outline

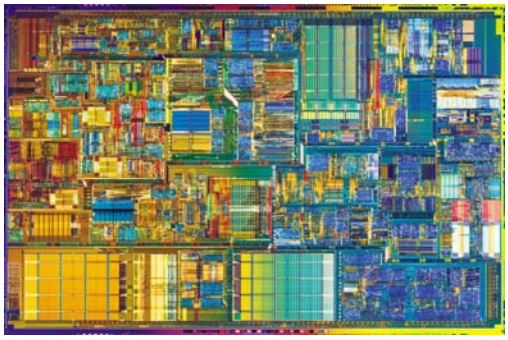
- Partitioning
- Floorplanning
- Placement
- Routing
- Compaction

Floorplanning

- Course contents
  - Floorplan basics
  - Normalized Polish expression for slicing flooprlans
  - B*-trees for non-slicing floorplans
- Readings
  - Chapter 10



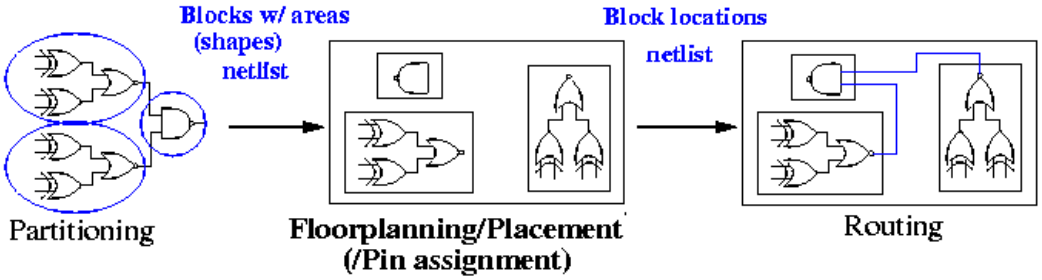
PowerPC 604



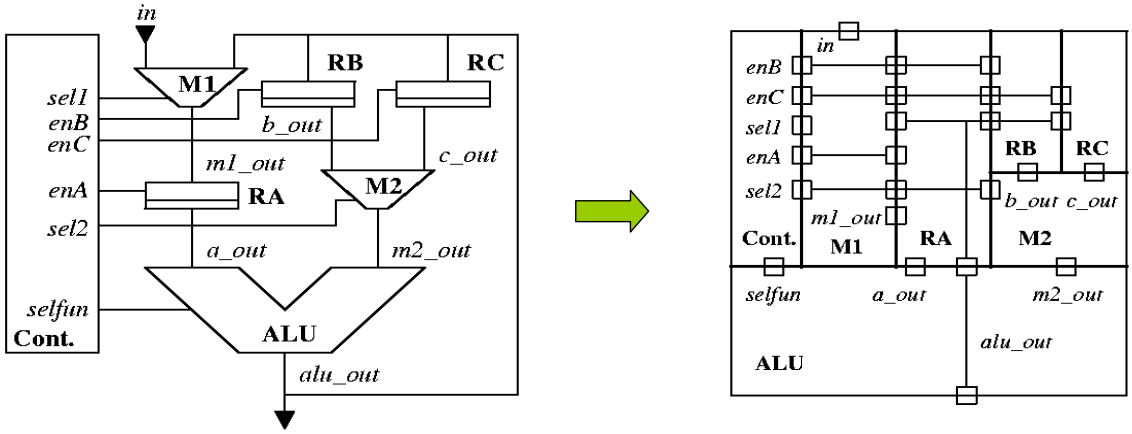
Pentium 4

Floorplanning

- Partitioning leads to
  - Blocks with well-defined **areas and shapes** (rigid/hard blocks).
  - Blocks with approximate areas and no particular shapes (flexible/soft blocks).
  - A **netlist** specifying connections between the blocks.
- Objectives
  - Find **locations** for all blocks.
  - Consider shapes of soft block and pin locations of all the blocks.



Early Layout Decision Example

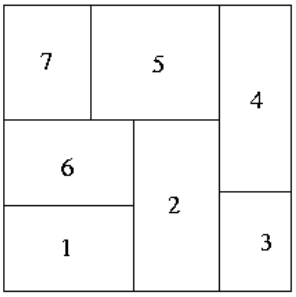


Early Layout Decision Methodology

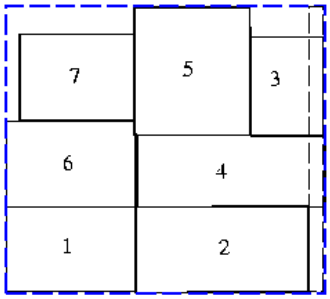
- An integrated circuit is essentially a two-dimensional medium; taking this aspect into account in early stages of the design helps in creating designs of good quality.
- Floorplanning gives early feedback: thinking of layout at early stages may suggest valuable architectural modifications; floorplanning also aids in estimating delay due to wiring.
- Floorplanning fits very well in a *top-down* design strategy, the *step-wise refinement* strategy also propagated in software design.
- Floorplanning assumes, however, *flexibility* in layout design, the existence of cells that can adapt their shapes and terminal locations to the environment.

Floorplanning Problem

- Inputs to the floorplanning problem:
  - A set of blocks, hard or soft.
  - Pin locations of hard blocks.
  - A netlist.
- Objectives: minimize area, reduce wirelength for (critical) nets, maximize routability (minimize congestion), determine shapes of soft blocks, etc.

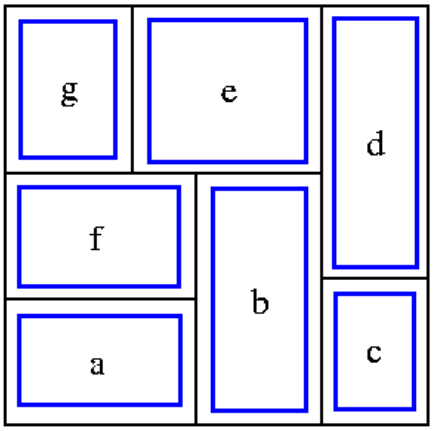


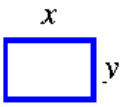
An optimal floorplan, in terms of area

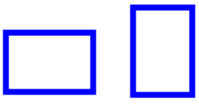


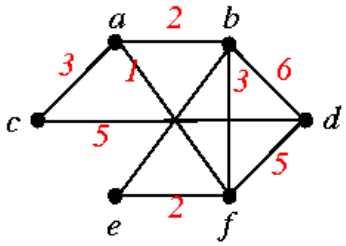
A non–optimal floorplan

Floorplan Design



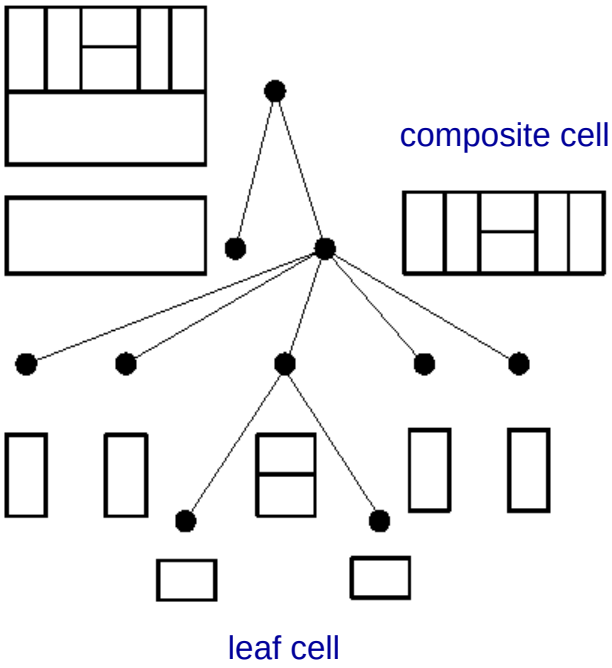
- *Modules:* 

- *Area: A=xy*
- *Aspect ratio: r <= y/x <= s*
- *Rotation:* 
- *Module connectivity*



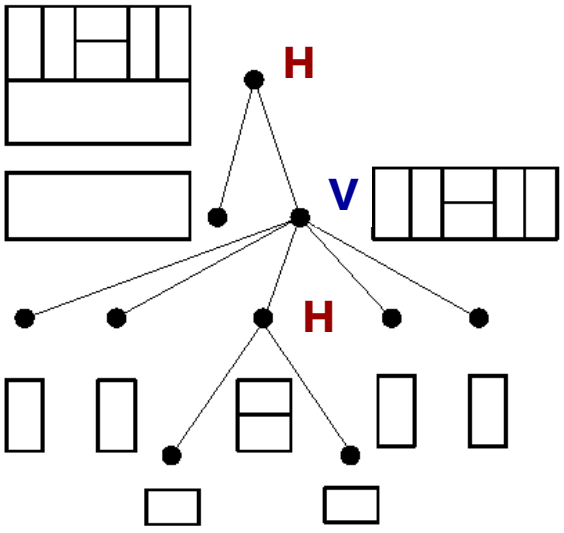
Floorplanning Concepts

- **Leaf cell (block / module):** a cell at the lowest level of the hierarchy; it does not contain any other cell.
- **Composite cell (block / module):** a cell that is composed of either leaf cells or composite cells. The entire IC is the highest- level composite cell.



Slicing Floorplan + Slicing Tree

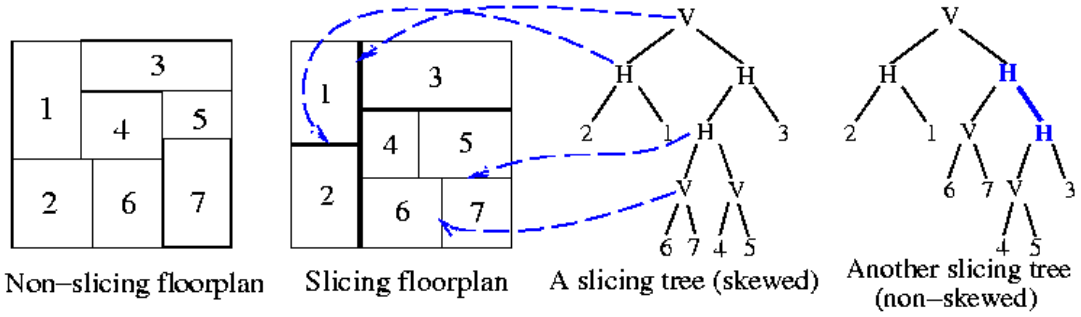
- A composite cell's subcells are obtained by a horizontal or vertical *bisection* of the composite cell.
- Slicing floorplans can be represented by a **slicing tree**.
- In a slicing tree, all cells (except for the top-level cell) have a *parent*, and all composite cells have *children*.
- A slicing floorplan is also called a floorplan of order 2.



H: horizontal cut
V: vertical cut
different from the definitions in the textbook!!

Skewed Slicing Tree

- **Rectangular dissection:** Subdivision of a given rectangle by a finite # of horizontal and vertical line segments into a finite # of non-overlapping rectangles.
- **Slicing structure:** a rectangular dissection that can be obtained by repetitively subdividing rectangles horizontally or vertically.
- **Slicing tree:** A binary tree, where each internal node represents a vertical cut line or horizontal cut line, and each leaf a basic rectangle.
- **Skewed slicing tree:** One in which no node and its **right** child are the same.



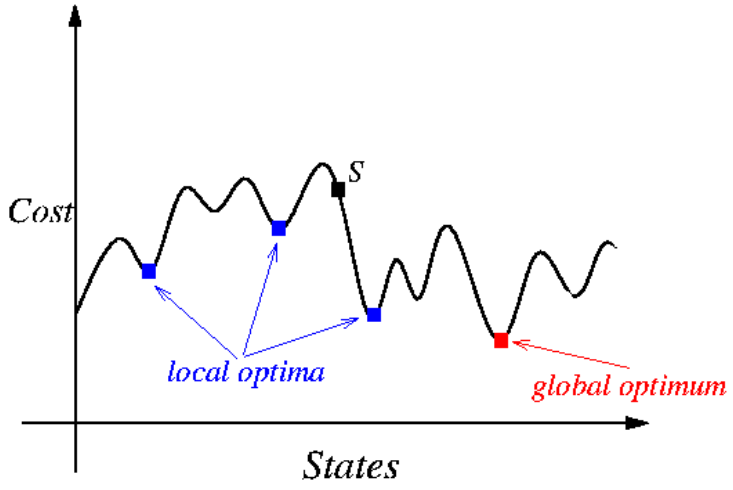
Non–slicing floorplan | Slicing floorplan | A slicing tree (skewed) | Another slicing tree (non–skewed)

Slicing Floorplan Design by Simulated Annealing

- Related work
  - Wong & Liu, "A new algorithm for floorplan design," DAC-86.
    - Considers slicing floorplans.
  - Wong & Liu, "Floorplan design for rectangular and L-shaped modules," ICCAD'87.
    - Also considers L-shaped modules.
  - Wong, Leong, Liu, *Simulated Annealing for VLSI Design*, pp. 31--71, Kluwer Academic Publishers, 1988.

Simulated Annealing

- Kirkpatrick, Gelatt, and Vecchi, "Optimization by simulated annealing," *Science*, May 1983.
- Greene and Supowit, "Simulated annealing without rejected moves," ICCD-84.



Simulated Annealing Basics

- Non-zero probability for "up-hill" moves.
- Probability depends on
  1. magnitude of the "up-hill" movement
  2. total search time

$$Prob(S \rightarrow S') = \begin{cases} 1 & \text{if } \Delta C \leq 0 \quad /* \text{ "down - hill" moves } */ \\ e^{-\frac{\Delta C}{T}} & \text{if } \Delta C > 0 \quad /* \text{ "up - hill" moves } */ \end{cases}$$

- $\Delta C = cost(S') - Cost(S)$
- T: Control parameter (temperature)
- Annealing schedule: $T = T_0, T_1, T_2, \ldots$, where $T_i = r^i T_0$ with $r < 1$.

Generic Simulated Annealing Algorithm

```
1 begin
2 Get an initial solution S;
3 Get an initial temperature T > 0;
4 while not yet "frozen" do
5    for 1 ≤ i ≤ P do
6        Pick a random neighbor S' of S;
7        Δ ← cost(S') - cost(S);
         /* downhill move */
8        if  Δ ≤ 0 then S ← S'
         /* uphill move */
9        if  Δ > 0 then S ← S' with probability e^(-Δ/T) ;
10 T ← rT;  /* reduce temperature */
11 return S
12 end
```


Basic Ingredients for Simulated Annealing

- Analogy:

Physical system	Optimization problem
state	configuration
energy	cost function
ground state	optimal solution
quenching	iterative improvement
careful annealing	simulated annealing

- Basic Ingredients for Simulated Annealing:
  - **Solution space**
  - **Neighborhood structure**
  - **Cost function**
  - **Annealing schedule**

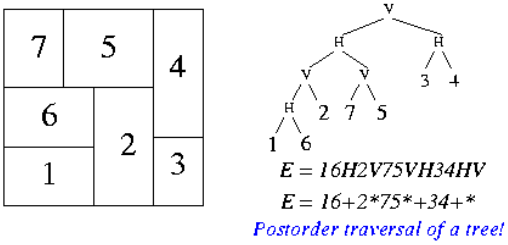
Solution Representation of Slicing Floorplan

- An expression $E = e_1\ e_2 \ldots\ e_{2n-1}$, where $e_i \in \{1, 2, \ldots, n, H, V\}$, $1 \le i \le 2n-1$, is a **Polish expression** of length $2n-1$ iff
  1. every operand j, $1 \le j \le n$, appears exactly once in E;
  2. (**the balloting property**) for every subexpression $E_i = e_1 \ldots e_i$, $1 \le i \le 2n-1$, # operands > # operators.

1 6 H 3 5 V 2 H V 7 4 H V

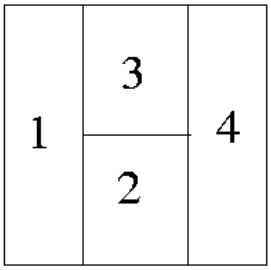
# of operands = 4 ..... = 7
# of operators = 2 ..... = 5

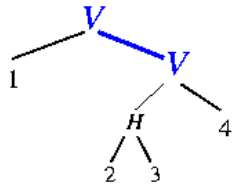
- Polish expression ↔ Postorder traversal.
- *ijH*: rectangle *i* on bottom of *j*; *ijV*: rectangle *i* on the left of *j*.



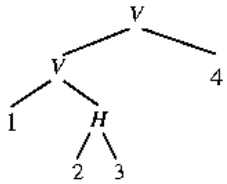
E = 16H2V75VH34HV
E = 16+2*75*+34+*
Postorder traversal of a tree!

Redundant Representations



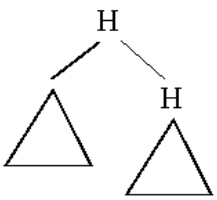


E = 123H4VV
non–skewed!

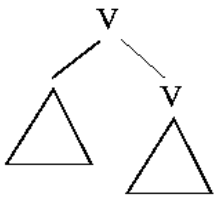


E = 123HV4V
skewed!

Non–skewed cases



..... HH .....

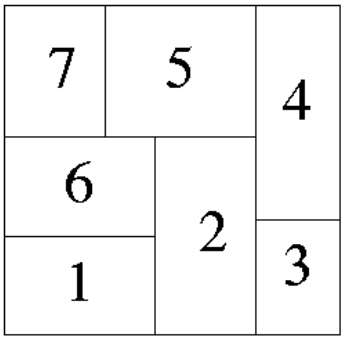


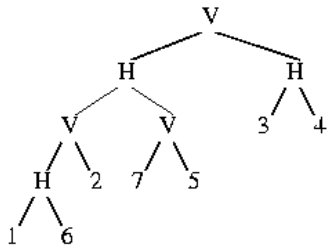
..... VV .....

- **Question:** How to eliminate ambiguous representation?

Normalized Polish Expression

- A Polish expression $E = e_1\ e_2\ \ldots\ e_{2n-1}$ is called **normalized** iff E has no consecutive operators of the same type (H or V), i.e. skewed.
- Given a **normalized** Polish expression, we can construct a **unique** rectangular slicing structure.



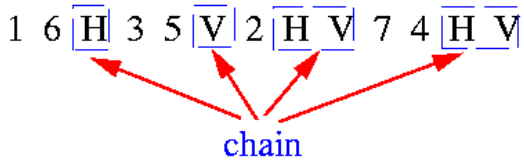


E = 16H2V75VH34HV

A normalized Polish expression

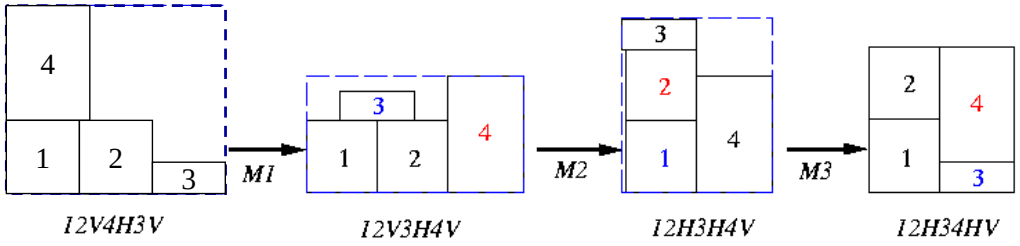
Neighborhood Structure

- **Chain:** *HVHVH* ... or *VHVHV* ...



- **Adjacent:** 1 and 6 are adjacent operands; 2 and 7 are adjacent operands; 5 and *V* are adjacent operand and operator.
- 3 types of moves:
  - *M1* (**Operand Swap**): Swap two adjacent operands.
  - *M2* (**Chain Invert**): Complement some chain ($\overline{V} = H$, $\overline{H} = V$).
  - *M3* (**Operator/Operand Swap**): Swap two adjacent operand and operator.

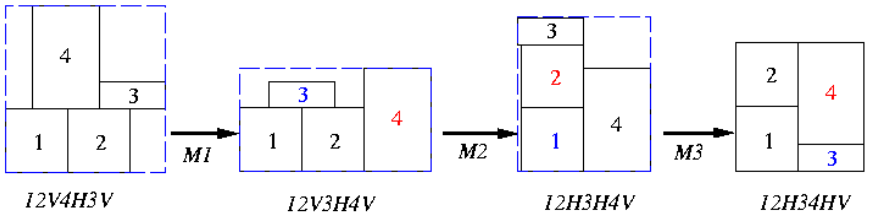
Effects of Perturbation



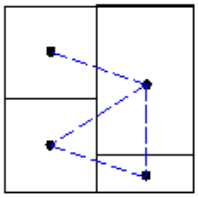
- **Question:** The balloting property holds during the moves?
  - *M1* and *M2* moves are OK.
  - **Check the *M3* moves!** Reject "illegal" *M3* moves.
- **Check *M3* moves:** Assume that the *M3* move swaps the operand e_i with the operator e_{i+1}, $1 \le i \le k-1$. Then, the swap will not violate the balloting property iff $2N_{i+1} < i$.
  - N_k: # of operators in the Polish expression $E = e_1\ e_2\ \ldots\ e_k$, $1 \le k \le 2n-1$

Cost Function

- $\phi = A + \lambda W$.
  - A: area of the smallest rectangle
  - W: overall wiring length
  - λ : user-specified parameter

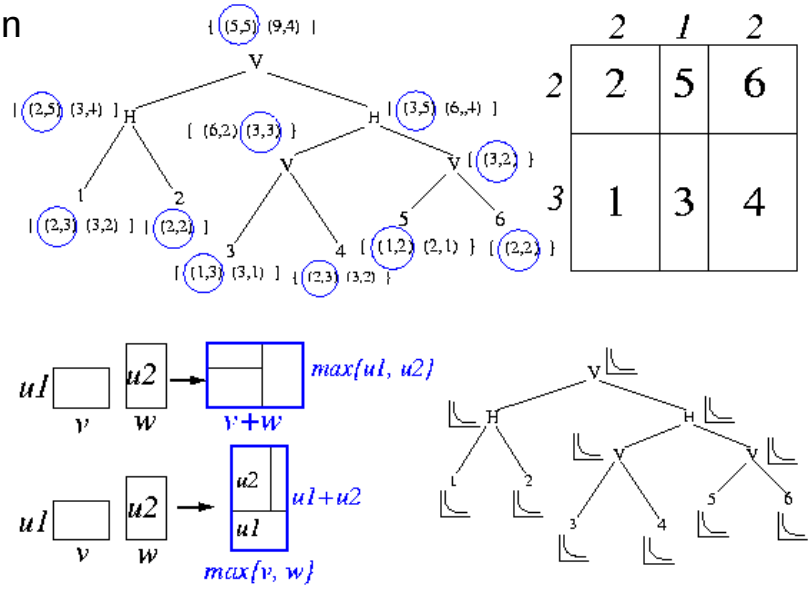


- $W = \sum_{ij} c_{ij}\, d_{ij}$.
  - c_{ij}: # of connections between blocks i and j.
  - d_{ij}: center-to-center distance between basic rectangles i and j.



Area Computation for Hard Blocks

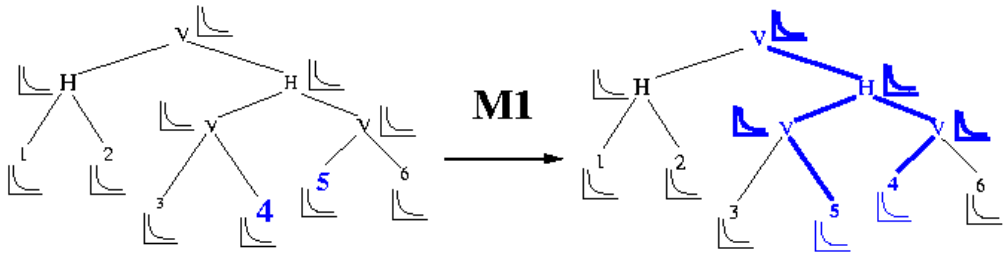
- Allow rotation



- Wiring cost?
  - Center-to-center interconnection length

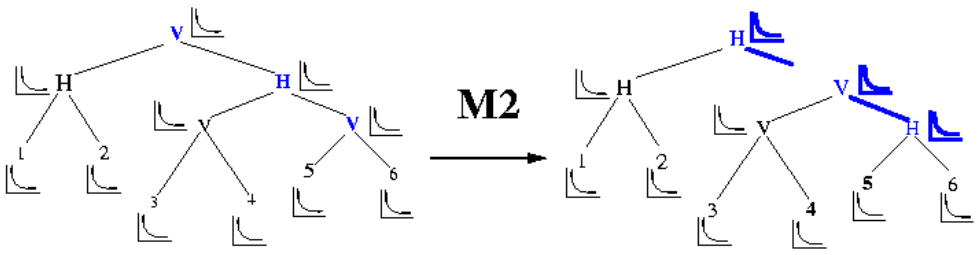
Incremental Computation of Cost Function

- Each move leads to only a minor modification of the Polish expression.
- At most **two paths** of the slicing tree need to be updated for each move.



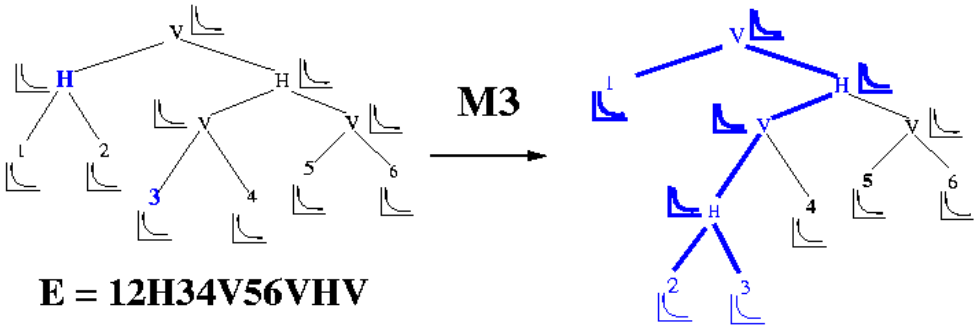
E = 12H34V56VHV **E = 12H35V46VHV**

Incremental Computation of Cost Function (cont'd)



E = 12H34V56VHV

E = 12H34V56HVH



E = 12H34V56VHV

E = 123H4V56VHV

Annealing Schedule

- Initial solution: 12*V*3*V* ... *nV*.

1	2	3		n

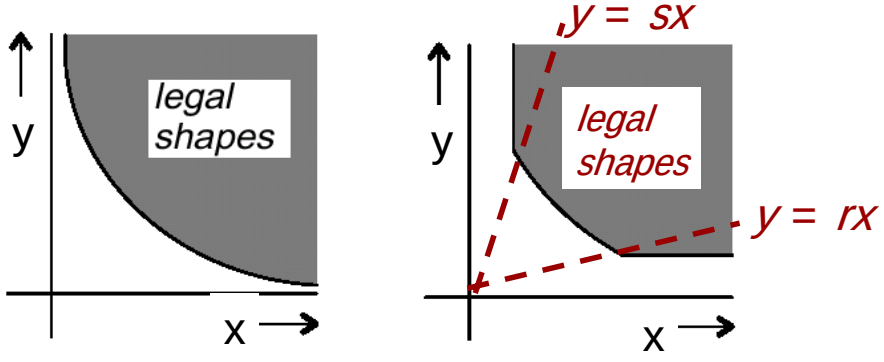
- $T_i = r^i T_0$, i = 1, 2, 3, ...; r =0.85.
- At each temperature, try kn moves (k = 5-10).
- Terminate the annealing process if
  - # of accepted moves < 5%,
  - temperature is low enough, or
  - run out of time.

Wong-Liu Algorithm

```
Input: (P, ε, r, k)
1 begin
2 E ← 12V3V4V ....nV; /* initial solution */
3 Best ← E; T0 ← Δavg / ln(P) ; M ← MT ← uphill ← 0; N = kn;
4 repeat
5   MT ← uphill ← reject ← 0;
6   repeat
7     SelectMove(M);
8     Case M of
9     M1: Select two adjacent operands ei and ej; NE ← Swap(E, ei, ej);
10    M2: Select a nonzero length chain C; NE ← Complement(E, C);
11    M3: done ← FALSE;
12        while not (done) do
13            Select two adjacent operand ei and operator ei+1;
14            if (ei-1 ≠ ei+1)  and (2 Ni+1 < i) then done ← TRUE;
13'           Select two adjacent operator ei and operand ei+1;
14'           if (ei≠ ei+2) then done ← TRUE;
15        NE ← Swap(E, ei, ei+1);
16    MT ← MT+1; Δcost ← cost(NE) - cost(E);
17    if (Δcost ≤ 0) or (Random < e^(-Δcost/T)      )
18     then
19         if (Δcost > 0) then uphill ← uphill + 1;
20         E ← NE;
21         if cost(E) < cost(best) then best ← E;
22      else reject ← reject + 1;
23   until (uphill > N) or (MT > 2N);
24   T ← rT; /* reduce temperature */
25 until (reject/MT > 0.95) or (T < ε) or OutOfTime;
26 end
```

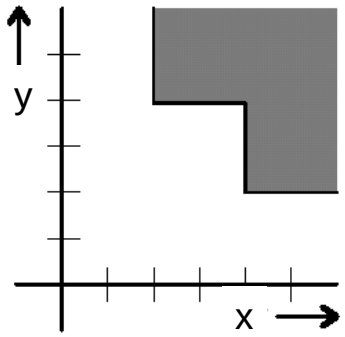

Shape Curve

- Flexible cells imply that cells can have different aspect ratios.
- The relation between the width x and the height y is: $xy = A$, or $y = A / x$. The shape function is a hyperbola.
- Very thin cells are not interesting and often not feasible to design. The shape function is a combination of a hyperbola and two straight lines.
  - Aspect ratio: $r <= y/x <= s$.



Shape Curve (cont'd)

- Leaf cells are built from discrete transistors: it is not realistic to assume that the shape function follows the hyperbola continuously.
- In an extreme case, a cell is rigid: it can only be rotated and mirrored during floorplanning or placement.



The shape function of a 2 × 4 inset cell.

Shape Curve (cont'd)

- In general, a *piecewise linear* function can be used to approximate any shape function.
- The points where the function changes its direction, are called the corner *(break) points* of the piecewise linear function.