Advanced Routing Techniques for Nanometer IC Designs

Organizer:
Jason Cong - Univ. of California, Los Angeles, CA

Speakers:
Jason Cong - Univ. of California, Los Angeles, CA
Tong Gao - Synopsys, Inc., Mountain View, CA
Rob A. Rutenbar - Carnegie Mellon Univ., Pittsburgh, PA

Outline

• Introduction
• Basic routing algorithms and scalable routing paradigms (Jason Cong)
• Challenges and solutions to large-scale IC routing in nanometer designs (Tong Gao)
• Challenges and solutions to analog and mixed signal routing (Rob Rutenbar)
Part I
Basic Routing Algorithms and Scalable Routing Paradigms

Jason Cong

Outline of Part I

• Introduction to the VLSI routing problem
• Basic routing algorithms
  – Global routing
  – Detailed routing
• Scalable routing paradigm
  – Hierarchical routing
  – Multilevel routing
Introduction to VLSI Routing Problem

- **Input**
  - Routing region: multi-layer rectangle
  - Obstacles: size/location
  - Pins: location
  - Netlist
- **Output**
  - Routed paths for all nets
- **Constraints**
  - Routing resources
  - Connection rules
  - Design rules
- **Objectives**
  - Total wirelength
  - Timing
  - Temperature
  - Manufacturability
  - Others

A routing example of four layers: poly, m1, m2, m3 and three nets: N1, N2 and N3

Challenges to Nanometer Routing

- **Sheer complexity**
  - > 1B transistors
  - > 100M signals to be routed
- **Complex design rules**
  - And the number increases rapidly each process generation
- **Many constraints and optimization objectives**
  - Routability
  - Timing
  - Noise
  - Manufacturability and yield
  - ...
Traditional Two Level Routing Flow

Floorplan/Placement Result

- Sequential routing
- Negotiation-based routing
- Iterative deletion
- Multicommodity flow-based

GR

- Grid-based
- Gridless
  - shape based
  - tile based
  - non-uniform grid graph

DR

Final Layout

Global Routing

- Global Routing Problem Formulation
- Single Net Routing
  - Spanning Tree
  - Steiner Tree
  - Rectilinear Steiner Tree
- Routing All Nets
  - Iterative Improvement
  - Negotiation Based Routing
  - Iterative Deletion
  - Multi-commodity Flow Based Routing
Global Routing Formulation

Given
(i) Placement of blocks/cells
(ii) channel capacities

Determine
Routing topology of each net in terms the
channels or routing regions it goes through

Optimize
(i) max # nets routed
(ii) min routing area (for variable die design)
(iii) min total wirelength

• In general cell or standard cell designs,
  we are able to move blocks or cell rows, so
  we can guarantee connections of all the
  nets.
• In gate array designs, exceeding channel
  capacity is not allowed.

Minimum Spanning Trees

Given a weighted graph
Find a spanning tree whose weight is minimum

Prim’s algorithm
start with an arbitrary node S
T ← {s}
while T is not a spanning tree
  find the closest pair x ∈ V-T, y ∈ T
  add (x,y) to T
runs in O(n^2) time
very simple to implement
always gives a tree of minimum cost
The Graph Minimal Steiner Tree Problem

• Input:
  – Undirected Graph G=(V,E)
  – A set of vertices N which is a subset of V
  – A function cost(e)>0 defined on the edges

• Output:
  – A tree T(V’,E’) in G, such that
    • N is a subset of V’, V’ is a subset of V
    • E’ is a subset of E

• Objective:
  – Minimize the sum of cost(e) for each e∈E’

• NP Complete
  – 1972, R. Karp formulated a reduction from Exact Cover.
  – 1979, S. Even formulated a reduction from Exact Cover by 3-sets (X3C).

Graph Steiner Tree Approximate Algorithms

• History
  – From 1980 to now
  – Approximate Ratio from 2 to 1.55

• Typical flow
  – Construct distance graph G’ (N, N×N),
    • cost(e_{ij}) = cost of shortest path between n_i and n_j
  – Construct Minimum Spanning Tree on G’, MST(G’)
  – Improve MST(G’)
KMB Heuristic

- [Kou, Markowsky and Berman, Acta Informatica 1981]
- **Approach**
  - Construct distance graph $G'$
  - Compute MST($G'$), expand each edge to the corresponding shortest path, yielding $G''$
  - Compute MST($G''$) and delete pendant edges from MST($G''$) until all leaf nodes are in $N$
- **Approximate ratio**
  - $2(1-1/L)$, where $L$ is the maximum number of leaves in any optimal solution
- **Complexity:** $O(|E| + V \log |V|)$

Iterative Improvement

- Alexander and Robins [TCAD96]
- **Take any Graph Steiner Tree and improve**
- **Definition**
  - Given a set of Steiner candidate node $S \subseteq V-N$, define the cost savings of with respect to $H$
    - $\Delta H(G,N,S) = \text{cost}(H(G,N)) - \text{cost}(H(G,NUS))$

**Iterated Graph Minimal Steiner Tree (IGMST) Algorithm.**

| Input: A weighted graph $G = (V,E)$, a set $N \subseteq V$, and a GMST heuristic $H$ |
| Output: A low-cost tree $T' = (V',E')$ spanning $N$, where $N' \subseteq V$ and $E' \subseteq E$ |

| Do forever |
| $T = \{ v \in V - N \mid \Delta H(G,N,S \cup \{v\}) > 0 \}$ |
| If $T = \emptyset$ then Return $H(G,N \cup S)$ |
| End if $T$ with maximum $\Delta H(G,N,S \cup \{t\})$ |

$S = S \cup \{v\}$
Rectilinear Steiner Trees

Given a set of points on the plane

Determine a Steiner tree using only horizontal and vertical wires (lines)

Manhattan distance:
\[ \text{cost}(v_1, v_2) = |x_1 - x_2| + |y_1 - y_2| \]
\[ v_1 = (x_1, y_1), \quad v_2 = (x_2, y_2) \]

Steiner points (Hanan grid)
Draw a horizontal and a vertical line through each point.
Need to consider only grid points as Steiner points

Prim-based algorithm:
Grow a connected subtree by iteratively adding the closest points

- It gives $3/2$-approximation, i.e. $\text{cost}(T) \leq \frac{3}{2}\text{cost}(T_{opt})$

Steiner Tree Heuristics

- Observation: MST approximation can be easily improved

- Difficulty: where to add Steiner points to maximize sharing??
**L-Shaped MST Approach**

*Ho, Vijayan and Wong, “A new approach to the rectilinear steiner tree problem”, DAC’89, pp. 161-166*

**Basic Idea:** Each non-degenerated edge in MST has two possible L-shaped layouts. Choose one for each edge in MST to maximize overlap.

![Diagram showing degenerated edges, non-degenerated edge, and two L-shaped layouts.

**Problem:** Compute the best L-shaped mapping.

**Key Ideas in L-RST Approach**

*Separable MST:* bounding boxes of every two non-adjacent edges don’t intersect or overlap

*Theorem:* Every point set has a separable MST

*Theorem:* Each node is adjacent to at most 8 edges (6 non-degenerate edges) in a rectilinear MST

*Theorem:* We can compute an optimal L-shaped implementation of an MST in $O(2^{d}n)$ time. (Dynamic Programming Approach).

Note that $d \leq 8$
FLUTE (1)

- First proposed for wirelength estimation [Chu, ICCAD04]
- Then also used for rectilinear Steiner minimal tree generation [Chu and Wong, ISPD 2005]
- Accurate and fast tree generation for low degree nets
- Optimal for nets up to degree 9
- Lookup table for low degree nets only, and partition high degree nets to low degree nets.

FLUTE (2)

- Lookup Table based Steiner Tree Generation
  - with techniques to reduce table size
- Net Representation by Vertical Sequence
  - index from sorted x position
  - sequence from sorted y location
  - Nets with the same vertical sequence share the same optimal tree solution
- Wirelength Representation
  - linear combination of Hanan grid length
  - Wirelength vector: vector of the coefficients
  - Potentially optimal wirelength vector (POWV): a vector that can potentially produce the optimal wirelength
  - Different nets can be represented by the same wirelength vector

Vertical sequence = 3142

Wirelength vector = (1,2,1,1,1,2)
Global Routing

- Global Routing Problem Formulation
- Single Net Routing
  - Spanning Tree
  - Steiner Tree
  - Rectilinear Steiner Tree
- Routing All Nets
  - Iterative Improvement
  - Negotiation Based Routing
  - Iterative Deletion
  - Multi-commodity Flow Based Routing

Iterative Improvement

- Route all nets independently, allowing possible design rule violation
- Iterative ripup and reroute for some or all nets
  - For both global routing and detailed routing
- Penalty function adjustment before each iteration
Negotiation Based Routing


- Iterative framework that allow resource sharing during intermediate iterations
- Signals negotiate with each other to determine which one needs the resource most
- Cost of resource adjusted with sharing and historical congestion information

Negotiated Cost Function

- Cost of using each routing resource given by 
  \[ c_n = (b_n + h_n) \times p_n \]
  - \( b_n \) is base cost
  - \( p_n \) denotes how many signals share the routing resource during current iteration
  - \( h_n \) denotes how congested the routing resource was during previous iterations
- \( p_n \) is increased with each iteration to deal with routing order
- \( h_n \) is increased with each iteration to deal with ripup and reroute order
- NC converges for bipartite graph matching by
  - Only rematch vertexes that have resource conflict with others
  - Or match all the vertexes and give priority to unconflicted resource when matching
**Negotiated Congestion Algorithm**

While shared resources exist

For each signal \( S_i \)

Rip up routing tree \( RT_i \)

Construct routing tree \( RT'_i \) using breadth-first search

Update the cost of nodes on \( RT'_i \)

End

End

---

**Iterative Deletion for Standard Cell Global Routing**  
*[Cong/Preas, ICCAD’88]*

- Assuming feedthroughs have been inserted -- chip width is fixed.
- \( V \): fixed. \( E \): connections within each channel.
- **Goal**: Build a spanning forest of \( G \) to minimize the total channel density.

Weight of an edge \( e = (p_i, p_j) \)

\[
W(e) = \alpha \frac{d(e)}{d} + \beta |x_i - x_j|
\]

\( d(e) \) is the density over \( e \). \( a >> b \) --- use wire length to break tie.
Basic Idea of Iterative Deletion

Start with all possible connections. Repeatedly delete the edges from G until we obtain a spanning forest.

\[ S := E; \]
\[ \text{repeat} \]
\[ \quad \text{Remove the max weighted edge in } S \text{ on a cycle;} \]
\[ \quad \text{Update edge weights for the affected edges;} \]
\[ \text{until } S \text{ is a spanning forest;} \]

**Advantages**

- Knows the congested area, since we start with all the possible edges (superior to iterative addition).
- Considers all the edges in every net, each net 'shrinks' to a spanning tree in parallel.
- There exists a deletion sequence which leads to the optimal spanning forest.

Simplified Net Connection Graph

SG=(V’, E’) is a subgraph of G.

V’ =V.

E’ : connections of adjacent pins of the same net in the same channel.
**Simplified Net Connection Graph**

(Cont’d)

Theorem: \( m=|E'|, n=|V'| \)
1. \( m \leq 1.5n \).
2. SG can be constructed in \( O(n \log n) \) time.
3. SG contains an optimal spanning forest.

Consequences:
1. \( \leq 0.5n \) steps of edge deletion.
   - Runs faster;
   - Predicts congested areas more accurately.
2. SG can be constructed efficiently.
3. SG is as good as G.

The algorithm starts with SG instead of G to go through iterative deletion.

---

**Multi-Commodity Flow (MCF)**

Based Global Routing

- More global view of all nets
- Does not have the net-ordering problem
- Can prove if a design does not have a feasible routing solution
- Original formulation: NP hard
- Relaxation: integer flow \( \Rightarrow \) fractional flow
  - relaxed problem is equal to LP and can be solved optimally
  - rounding to get integer results
- Formulations can be adjusted to handle
  - Performance
  - Coupling
  - Power
History of MCF Based Global Routing

- 1987, Shragowitz & Keel, Integration,
  - first usage of MCF in 2-pin nets global routing
- 1990, Meixuer & Lauther, ICCAD
  - Approximation using single-commodity flow (for rip-up)
- 1991, Raghavan and Thompson, Algorithmica,
  - first usage of MCF in multi-pin nets global routing, find optimal fractional global routing results
- 1996, Carden, Li and Cheng, TCAD
  - Speedup using LP approximate algorithm to solve MCF
- 2001, Albrecht, TCAD
  - Further speedup the approximate algorithm by application of Gargand Konemann’s fast LP approximation

MCF Based Global Routing Formulation [Albrecht, ISPD’00]

- Global Routing Problem can be formed as a mixed integer linear programming (NP-hard) problem : assuming there are \( l_i \) candidate Steiner tree for each net \( i \)

\[
\begin{align*}
\min & \quad \lambda \\
\text{subject to} & \quad \sum_{i,j \in T_{i,j}} \lambda(x) \leq \sum_{e \in E} \lambda(e) \\
& \quad \sum_{j=1}^{l_i} x_{i,j} = 1 \quad \text{for } i = 1, \ldots, k \\
& \quad x_{i,j} \in \{0,1\} \quad \text{for } i = 1, \ldots, k; \\
& \quad j = 1, \ldots, l_i
\end{align*}
\]

\( \lambda \) – maximum congestion, \( T_{i,j} \) – the \( j \)th Steiner tree for net \( i \), \( e \) – edge of global routing graph, \( w_{i,e} \) – cost of net \( i \) to go through \( e \), \( c(e) \) – capacity of \( e \), \( k \) – number of nets to be routed,
\( x_{i,j} \) – 0 or 1, indicating whether \( P_{i,j} \) is selected for net \( i \), \( l_i \) – candidate tree number of net \( i \)
MCF Based Global Routing Formulation

- linear programming relaxation \(\rightarrow\) fractional global routing problem

\[
\begin{align*}
\min \lambda \\
\text{subject to} \\
\sum_{i,j \in \mathcal{E}_{i,j}} w_{i,j} x_{i,j} &\leq \lambda c(e) \quad \text{for } e \in E \\
\sum_{j=1}^{k_i} x_{i,j} &= 1 \quad \text{for } i = 1, \ldots, k \\
x_{i,j} &\geq 0 \quad \text{for } i = 1, \ldots, k; j = 1, \ldots, \ell_i
\end{align*}
\]  

- can be solved optimally by fast matrix multiplication: slow
- approximate, combinatorial algorithms: faster, with error bound

Approximation Algorithm for Fractional Global Routing

- originally used as approximation for multi-terminal multi-commodity flow problem
- associate each edge with a length \(y_e\), which is related with the congestion at \(e\)
- at any step, route a unit flow along the minimum Steiner tree
- then multiply every edge on the tree with \(1 + \epsilon \frac{w_{i,j}}{c(e)}\)
- long edges \(\leftrightarrow\) congested edges
- after sufficient many steps, say \(X\), there is a flow number \(X_{i,j}\), assigned to the \(j\)th candidate tree of net \(i\), and \(X_{i,j}/X\) is the fractional flow for net \(i\) on the \(j\)th tree.
Approximation Algorithm for Fractional Global Routing

1. Set $y_e := \frac{c(e)}{\Delta}$ for all $e \in E$.
2. Set $x_{i,j} := 0$ for $i = 1, ..., k$, $j = 1, ..., h$.
3. Set $z_k := 0$ for $i = 1, ..., k$.
4. While $\left( \sum_{e \in E} c(e)y_e \right) < 1$
5. begin
6. For $i := 1$ to $k$
7. begin
8. If $z_k = 0$ or $\sum_{e \in T_{i,j}} w_{i,e}y_e > (1 + \gamma)e_i$
9. begin
10. Find a minimal Steiner tree $T_{i,j} \in T_i$ for net $i$ with respect to lengths $w_{i,e}y_e$, $e \in E$.
11. Set $z_i := \sum_{e \in T_{i,j}} w_{i,e}y_e$.
12. end
13. end
14. Set $x_{i,j} := x_{i,j} + 1$.
15. Set $y_e := \left( 1 + \frac{e_i}{\Delta} \right)y_e$ for all $e \in T_{i,j}$.
16. end

- $x_{i,j}$ is the flow on $T_{i,j}$
- $y_e$ is the length of edge $e$
- $z_k$ is the current total cost for net $I$
- $W_{i,e}$ is the width of net $i$ at edge $e$
- $\delta$, $\gamma$, $\epsilon$ are parameters of the algorithm.
- Implementation: $\delta$ can be 1 (related to the error bound), $\gamma$ between 7 and 10, $\epsilon$ between 0.6 and 2.0

Outline of Part I

- Introduction to the VLSI routing problem
- Basic routing algorithms
  - Global routing
  - Detailed routing
    - Grid-based routing
      - Maze Routing
      - Line Search
    - Gridless routing
      - Implicit Routing Graph-Based Routing
    - Between grid-based and gridless routing
      - Subgrid-Based Router
- Scalable routing paradigm
  - Hierarchical routing
  - Multilevel routing
Maze Routing

Basic idea -- wave propagation method (Lee, 1961)

- Breadth-first search
- Backtracking after finding the shortest path
- Guarantee to find the shortest path

Connecting a Multi-Terminal Net

- Connect one terminal at a time
- Use the entire connected paths as source to expand.
- Improve the quality of the solution (remove a segment and re-connect)
Problems with Maze Routing

- Slow: for each net, we have to search a $N \times N$ grid.

**Improvements**
- Simple speed-up
- Line search (Mikami & Tabuchi, 1968; Hightower, 1969)
- Minimum detour algorithm (Hadlock, 1977)
- Fast maze algorithm (Soukup, 1978)

- Net ordering: we have to route net by net, but it is difficult to determine the best net ordering!

**Improvement**
- Use other routers
  - channel/switchbox routers
  - hierarchical routers
- Rip-up and re-route

Line Searching Algorithms

*Mikami & Tabuchi IFIPS Proc, Vol H47, pp 1475-1478, 1968*

*Hightower, IFIP Proc. 6th Design Automation Conf. pp 1-24, 1969*

**Mikami & Tabuchi’s algorithm**
- Generate search lines from both the source and the target (level-0 lines)
- From every point on the level-i search lines, generate perpendicular level-(i+1) search lines
- Stop until a search line from the source meet a search line from the target
- Guarantee to find the shortest path
Hightower’s algorithm

Difference: generate level-(i+1) search lines which are extendable beyond the obstacle.

Faster, but not guarantee a connection

Minimum Detour Algorithm

Hadlock, F.O. “A shortest path algorithm for grid graphs” Networks, vol 7, 1977

Let P be a path connecting A and B
\[ \text{dist}(A,B)=\text{Manhattan distance between A and B} \]
\[ \text{detour}(p): \text{# points away from the target (detour number)} \]

Then \[ \text{length}(p): \text{dist}(A, B)+2x \text{detour}(p) \]
Minimum Detour Algorithm (cont’d)

**Algorithm**
- each cell stores the detour number so far from the source expand the cell with the least detour number

**Result**
- guarantee to find a shortest path
- expand fewer points in general
  (similar to the A* search algorithm)

---

Cells Searched Before Target is Reached

(a) original Lee algorithm
(b) minimum detour algorithm
(c) fast maze algorithm
Line Search with Optimal Wirelength [Hetzel DATE 98]

- **Existing Path Searching Algorithms**
  - Node-oriented labeling algorithms
    - original maze search, Lee 1961, A* maze search, Rubin 1974, etc.
    - Pros: general cost function/ optimal solution
    - Cons: runtime/memory consumption
  - Line search
    - Mikami & Tabuchi 1968, Hightower 1969
    - Pros: runtime/memory consumption
    - Cons: can not guarantee optimal

XRouter Detailed Router

- Shortest Manhattan length paths in a grid graph
  - Suitable for detailed routing
- Adoption of Rubin’s algorithm (A* search) to interval labeling
  - Node cost = current_cost + potential cost
- Expand using intervals
- Runtime/memory consumption: similar to line search
  - Can handle huge detailed routing grids
A routing example

\begin{align*}
s &= (4, 1, 0), \quad t = (7, 5, 0), \quad ||s - t||_1 = 7
\end{align*}

(1). Initialization: all nodes, \( \delta (v) = \infty \), \( \delta (s) = ||s - t||_1 = 7 \)
(2). For \( \delta = 7 \), label \( G_1 \) with 7, label \( G_0 \) with 7
(3). Next largest \( \delta = 9 \), label \( G_1 \) with 9, label \( G_0 \) with 9
(4). Next largest \( \delta = 11 \), label \( G_1 \) with 11, label \( G_0 \) with 11
(5). Next largest \( \delta = 13 \), label \( G_1 \) with 13, label \( G_0 \) with 13, \( \delta (t) = 13 \), success,
(6). retrieve routing path

\#Labeling planes = \( 4 \leq L - ||s - t||_1 + 1 = 13 - 7 + 1 = 6 \)
**A routing example**

![Routing Example Diagram]

- Theoretically fast for simple paths with a small detour
- Guarantees optimality

---

**Gridless Detailed Routing**

- **Gridless Routing**
  - More flexible
  - Longer runtime due to complex data structure
- **Gridless Detailed Routing Algorithms**
  - Shape (Tile) based routing [Sato, et al., ISCS87, Margarino, et al., TCAD87, Dion, et al., WRL Research Report 95/3, Liu, et al., ISPD98]
  - Graph-based routing [Wu, et al., TC87, Ohtsuki, ICCAS85, Cong, et al., Zheng, et al., TCAD96, ICCAD'99]
Basic Operation:
Obstacle Expansion in Gridless Routing

- In order to route a wire with width $w$ and spacing $sp$
  - Obstacles are expanded by $w/2 + sp$
- Reduced the problem to finding a zero-width routing path
  - [Schiele, et al., DAC 90]
  - [Dion, et al., WRL Research Report 95/3]
  - [Cong, et al., ICCAD’99]

DUNE [Cong, et al., ICCAD’99]

- Gridless routing engine [Cong, et al., ICCAD’99]
  - Non-uniform grid graph
  - Implicit grid graph
  - Path-based Maze Searching
Rectangle-based Query

- Given a set of rectangles and a query point q
- Query: if the query point is contained by any of the given rectangles

Rectangle-based Query Algorithms

- K-D tree
- Quad-list quad tree
- Multiple storage quad tree
- HV/VH tree
- 1-D and 2-D indexing
2-D Query Data Structure in Dune

Data Structure

Query

Is q in free space

Caching

Cache is an array that stores previous query results

Caching Obstacles

Caching Empty Area
Subgrid Based Router

- Handle Complicated Wire Widths/Spacing in Grid-Based Router
- Finer Routing Grids (e.g. 16× the conventional detailed router)
- Each Grid Contains 4×4 Subgrids
  - Bit patterns used in each grid to accelerate the point query

Finer Routing Grids

- Conventional detailed router – Routing on a fixed grid
- Magma detailed router – Expansion on the coarser grid, but implement the path on the finer subgrid
Step 1: Build Subgrid Map

• Expand obstacles by proper width and spacing
• Covering subgrid points by expanded Obstacles (e.g. 14I, 14J, 14K)

Step 2: Make Every Grid Map Reachable

• A grid map is reachable: iff every subgrid with “1” can be reached by other subgrid with “1”s
• Dropping some “1”s might be necessary

Reachable bit patterns
Step 3: Path Expansion by AND Operation

- On adjacent subgrids of two neighboring grids

![Diagram showing path expansion by AND operation]

Outline of Part I

- Introduction to the VLSI routing problem
- Basic routing algorithms
  - Global routing
  - Detailed routing
- **Scalable routing paradigm**
  - Hierarchical routing
  - Multilevel routing
Hierarchical Wire Routing


- Top-down refinement
- Can be used for both global routing and detailed routing

The Basic Approach

Use recursive 2x2 routing
2x2 Routing

- Given
  - Edge capacity constraints
  - Via constraints (if detailed routing)
- Each net is one of the following 11 types
- Determine routing for all the nets

Types of 2-terminal nets

Types of 3-terminal nets

Types of 4-terminal nets

Solution method: integer Linear programming
Routing Configuration of Each Type of Nets

**Routing Configuration of Each Type of Nets (Cont'd)**

<table>
<thead>
<tr>
<th>Type</th>
<th>Configuration</th>
<th>X(n,1), X(n,2), X(n,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td><img src="image1" alt="Type 1 Configuration" /></td>
<td>X(7,1), X(7,2), X(7,3)</td>
</tr>
<tr>
<td>Type 2</td>
<td><img src="image2" alt="Type 2 Configuration" /></td>
<td>X(11,1), X(11,2), X(11,3), X(11,4)</td>
</tr>
</tbody>
</table>
Integer Linear Programming for 2x2 Routing

\[ \begin{align*}
  k(i) &: \text{# nets of type } i \quad 1 \leq i \leq 11 \\
  h_1, h_2, v_1, v_2 &: \text{capacity constraints.} \\
  x(i) &: \text{# unconnected nets of type } i \quad 1 \leq i \leq 11 \\
  x(i,j) &: \text{# nets of type } i \text{ connected using the } j\text{-th possibility} \\
  \min & \sum_{i=1}^{11} x(i) \\
  & \begin{cases} \\
    x(i) \geq 0 \\
    x(i,j) \geq 0 \\
    x(i) + \sum_{j} x(i,j) = k(i) & 1 \leq i \leq 11 \\
    \sum_{(i,j) \in \Theta_1} x(i,j) \leq v_1 \\
    \sum_{(i,j) \in \Theta_2} x(i,j) \leq v_2 \\
    \sum_{(i,j) \in \Theta_1} x(i,j) \leq h_1 \\
    \sum_{(i,j) \in \Theta_2} x(i,j) \leq h_2 \\
  \end{cases}
\end{align*} \]

Integer Linear Programming for 2x2 Routing (Cont’d)

\[ \begin{align*}
  V_1 & = \{(i,j) | P(i,j) \text{ crosses left horizontal boundary}\} \\
  & = \{(1,1), (2, 2), (3, 2), (4, 2), (5, 2), (6, 1), (7, 2), (7, 3), (8, 1), (8, 3), (9, 2), (9, 3), (10, 2), (10, 3), (11, 1), (11,3), (11, 4)\} \\
  V_2, H_1, H_2 & \text{ defined similarly}
\end{align*} \]
ILP Approach for 2x2 Routing (Cont’d)

- 39 variables
  \[
  \begin{align*}
  11 & \quad x(i) \\
  28 & \quad x(i,j) \\
  11 & \quad = k(i)
  \end{align*}
  \]
- 15 linear equation
  \[
  \begin{align*}
  \quad & \leq h_1, h_2, v_1, v_2
  \end{align*}
  \]

(19 equations, if we consider via constraints since we have 4 more equations for each super cell)

- Can be solved efficiently
- Map a net to a routing configuration using heuristic (we only know the number of nets for each configuration)

Multilevel Routing Framework (MARS [TCAD05])

- Fine routing tile generation
- Coarsening
- Implicit graph gridless routing
- Initial routing
- Refinement
- History-based iterative refinement
- Multicommodity flow based algorithm
- Detailed routing
Starting Point: Finest Tile Generation + Capacity Estimation

- 3-D routing graph generation
- Resource estimation: use the technique in [Cong, et al., ISPD’00]

Planning Graph Construction

Congestion Estimation

$$C = W_1 \times D_1 / D + W_2 \times D_2 / D + W_3 \times D_3 / D$$

Downward Pass

Fine routing tile generation

Detailed routing

Coarsening

Refinement

Initial routing
Downward Pass — Tile Coarsening

- Estimate resources on the coarser tiles from finer tiles

\[ T_{i,j} \quad T_{i,j+1} \quad T_{i+1,j} \quad T_{i+1,j+1} \]

\[ G_i \quad G_{i+1} \]

Downward Pass — Resource Reservation*

- Local net effect
  - Congested region
  - Waste planning efforts
Initial Routing at Coarsest Level

- Fine routing tile generation
- Coarsening
- Initial routing
- Detailed routing
- Refinement

Multicommodity Flow-Based Initial Routing

- Start the Planning at the Coarsest Level
- Advantages of Multicommodity Flow-based Algorithm
  - Fast enough for coarse grids
  - More global view, proved error bound to optimal for fractional routing
  - Can be integrated with performance optimization by including high-performance topologies, such as A-Tree, BA-Tree, and P-Tree
- Implemented the Algorithm in [Albrecht, ISPD'00]
  - Minimize the overall congestion
  - Randomized rounding
Congestion-Driven Graph Based Steiner Tree

• Steiner Tree Approach
  – Simplistic approach
    • Starting from a minimum spanning tree
    • Fast and utilize the maze search engine
  – Congestion driven construction
    • Avoid congested area and big obstacles
  – Whole tree refinement
    • Tree topology can change at every refinement level

Congestion Driven Graph Based Steiner Tree

• Tree Construction
  – Starting from a geometric MST
  – Start with the shortest edge
  – Hit and stop maze searching

• Steiner tree refinement
  – Input edge ordering
  – Connect newly appeared nodes first
  – Refine the remaining edges according to the ordering
Refinement

Fine routing tile generation

\[ G_0 \]

Coarsening

\[ G_1 \]

\[ \ldots \]

\[ G_k \]

Initial routing

Detailed routing

\[ G_0 \]

\[ G_1 \]

\[ \ldots \]

\[ G_1 \]

Incremental Refinement

- Refine the coarser level results at the finer level

\[ L_1 \]

\[ L_2 \]

Local nets

Global nets

Preferred region for \( N_3 \)

Lower cost

Higher cost

Routing graph for \( N_3 \)

- Use A* algorithm to find the path for each net
History-Based Iterative Refinement

- History Based Multi-Iteration Refinement
  - First proposed in [Nair, TCAD’87], later used in PathFinder [McMurchie et al, FPGA Symp’95]
  - Iteratively update each edge’s cost with the consideration of historical congestion information
  - Reroute all the nets based on the new edge cost functions

- Cost Function Used in MARS
  \[ cost(e, i) = \alpha \times \text{congestion}(e, i) + \beta \times \text{history}(e, i) \]
  \[ \text{history}(e, i) = \text{history}(e, i - 1) + \gamma \times \text{congestion}(e, i - 1) \]

Hierarchical vs. Multilevel Routing

- No local net view during coarse level routing
- Coarse-level decisions constrain the fine-level solution
- Resource reservation for local nets
- Coarse-level decisions only guide the fine-level solution
Part II
Challenges and Solutions to Large-Scale IC Routing in Nanometer Designs

Tong Gao

Outline of Part II

• Objectives and new challenges for industrial routers
• Techniques for run time challenges
• Techniques for capacity challenges
• Techniques for design rules challenges
• Techniques for DFM/DFY challenges
Objectives

- **Traditional objectives**
  - QoR – Via count, wire length, DRCs, and timing/crosstalk
  - Via count and wire length cause congestion and affect yield
  - DRCs increase tapeout time, and possibly chip cost
  - Timing/crosstalk affects performance and post routing optimization efforts
- **Run time**
  - Always one of the most important objectives
  - Closely related to QoR
- **Memory**
  - Very important for 32bit machines
  - Still important for 64bit machines
    - Hardware is expensive
    - May lead to more run time

New Challenges – Design Rule Explosion

- **Example: end of line spacing rule**

<table>
<thead>
<tr>
<th>Description</th>
<th>Rule (µm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum spacing (S) between a metal and the end-of-line of the metal whose edge width (W) &lt;= 0.2 µm</td>
<td>0.14</td>
</tr>
<tr>
<td>Otherwise, minimum spacing</td>
<td>0.11</td>
</tr>
</tbody>
</table>

This rule is applied only inside the shaded area
### New Challenges – Design Rule Explosion

**Example: end of line spacing rule (cont.)**

<table>
<thead>
<tr>
<th>Description</th>
<th>Rule (µm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum spacing (S) between two end-of-line metals whose edge width W &lt;= 0.2. The end-of-line of the metal is searched from corner to distance K</td>
<td>0.14</td>
</tr>
<tr>
<td>Otherwise, default minimum spacing (minSpacing)</td>
<td>0.11</td>
</tr>
</tbody>
</table>

If a metal has width of W <= 0.2 and there is NO connecting metal within minimum width 0.16 (W_{min}), has neighboring metal along two adjacent edges, then one of the spacings (S_1 or S_2) should be >= stubSpacing; Neighboring metal is searched from corner to distance 0.035 (K)
New Challenges – Design Rule Explosion

- Example: end of line spacing rule (cont.)

<table>
<thead>
<tr>
<th>Description</th>
<th>Rule (µm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>If a metal of width (W) &lt;= 0.2 has neighboring metals along three adjacent edges, then one of the spacings (S₁ or S₂ or S₃) should be &gt;= 0.14</td>
<td>0.14</td>
</tr>
</tbody>
</table>

- Progressively becomes more complicated
  - Need to support each intermediate form
- Spacing rules involves a lot more than 2 shapes
  - Analysis challenge – multiple neighbors with pattern
  - DRC book keeping challenge – no more two shape DRCs
  - Optimization challenges – many alternatives to resolve DRCs, and many new ways to create to DRCs
- Benefit from polygon
  - Routing is rectangle based
New Challenges – Design Rule Explosion

• Min edge rules
  – Min edge – polygon edge less than given threshold length

CONCAVE corner
The concave corner is formed by two adjacent edges if both are minimum length.

CONVEX corner
The two or more minimum length edges form the convex corner, but no concave corner adjacent to it.

New Challenges – Design Rule Explosion

• Max number of min edge rule

<table>
<thead>
<tr>
<th>Description</th>
<th>Rule (μm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min edge rule threshold</td>
<td>α</td>
</tr>
<tr>
<td>Violation if number of consecutive min edges is greater than this value</td>
<td>β</td>
</tr>
</tbody>
</table>

Edge A, B, and C are all shorter than α μm
Max number of min edge rule is violated if β < 3
New Challenges – Design Rule Explosion

• Total min edge length rules

<table>
<thead>
<tr>
<th>Description</th>
<th>Rule (µm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>If there is at least one edge less than the minimum edge length,</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>Violation if the sum of the minimum edge lengths is greater than this value</td>
<td>$\beta$</td>
</tr>
</tbody>
</table>

Edge A, B, and C are all shorter than $\alpha$ µm
Total minimum edge length rule is violated if length of $(A + B + C) > \beta$ µm

New Challenges – Design Rule Explosion

• Min edge length rules
  - If $\text{minEdgeMode} = 0$, a concave corner is needed
    - $A, B, C < \alpha$ and $A + B + C > \beta$ Total minimum edge length rule is violated
    - $B, C < \alpha$ and $B + C > \beta$ Total minimum edge length rule is not violated

$\text{minEdgeMode} = 0$
New Challenges – Design Rule Explosion

• Min edge length rules (cont.)
  – If \( \text{minEdgeMode} = 1 \), a concave corner is not needed

    - If \( A, B, C < \alpha \) and \( A + B + C > \beta \)
      Total minimum edge length rule is violated

    - If \( B, C < \alpha \) and \( B + C > \beta \)
      Total minimum edge length rule is violated

![Min Edge Length Rules Diagram](image)

New Challenges – Design Rule Explosion

• Min edge rule
  – Analysis challenge – Totally polygon based while routing shapes are rectangles
  – DRC book keeping challenge – for multiple shapes along edges
  – Optimization challenges – many different ways to fix the DRCs
    • Patching
    • Shifting
    • Via rotating
    • Rerouting

![Min Edge Rule Diagram](image)
New Challenges – Design Rule Explosion

- **Number of design rule exploding**
  - Synopsys router already added more than 40 new 45nm rules
  - A lot of development efforts
  - Analysis can be very time consuming
  - Impractical to support in search core
  - More design rules means more DRCs to resolve, which again leads to more run time

New Challenges – Design Rule Explosion

- **Design rule complexity explosion**
  - Design rules are to enhance yield - difficult to model with rules
    - Need to be conservative
    - Large number of complex rules to reduce conservatism
      - More polygon based (versus rectangle based)
      - Very difficult to model in search core
        - Need to bring in design rule analysis to block search graph for existing shapes
        - Might be impossible to model their blockage onto search graph for to be routed shapes
New Objectives – Design Rules

- Design rule number and complexity, and large design size compound with each other, causing major implementation, quality, runtime, and memory challenges
- New objective: have the ability to add large number of new complex design rules in short period of time, while keeping run time/memory under control

New Challenges - DFM
New Challenges - DFM

- **New DFM/DFY requirements**
  - Yield becomes a major issue in 90nm/65nm
  - Directly related to manufacturing cost – very important
  - Largely determined by routing – natural place to consider
  - Might be difficult or impossible to fix post routing

- **New challenges**
  - Yield and rules are not very compatible (e.g., end of line rule)
    - Simple rules do not correlate well to yield – need to be conservative
    - Large number of complex rules are needed to reduce conservatism
    - Most yield related rules are soft – a new concept
    - Model based approaches give much more accurate results – never before
    - Independent rules affect yield in non-monotonic way
      - Example, double via enhance yield for vias, but increase critical area, and cause small edges, which hurts yield

New objectives - DFM

- **New objectives**
  - Soft rule support - Multiple rules simultaneously with different weight (e.g., multiple spacing requirements) – major change to routing core
  - Model based approach instead of rule based approach
    - Yield simulation – run time?
    - Simulation results driving routing – how?
  - Unified yield analyzer to drive router
    - Answer if a routing decision improve yield
    - Run time need to be adequate for router
    - Analyzed results need to be able to drive routing decisions – how?
Outline of Part II

- Objectives and new challenges for industrial routers
- Techniques for run time challenges
- Techniques for capacity challenges
- Techniques for design rules challenges
- Techniques for DFM/DFY challenges

Techniques for Run Time

- More efficient routing algorithms
  - As efficient as possible algorithms and implementations
    - Dijkstra’s shortest path algorithm is not enough
      - Only work for simple cost function with no constraints
      - Modern search cores consider constraints – e.g., via staging rule (“Via design rule consideration in multi-layer maze routing algorithms”, Jason Cong etc.)
        » Need to keep multiple search front at the same point
      - Carefully tuned heuristics make a huge difference
    - Implementation make a huge difference

< stager distance

<table>
<thead>
<tr>
<th>M2</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src</td>
<td>tgt</td>
</tr>
<tr>
<td>Blocked</td>
<td></td>
</tr>
</tbody>
</table>

< stager distance

<table>
<thead>
<tr>
<th>M2</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src</td>
<td>tgt</td>
</tr>
<tr>
<td>Blocked</td>
<td></td>
</tr>
</tbody>
</table>

Single front fails

<table>
<thead>
<tr>
<th>M2</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src</td>
<td>tgt</td>
</tr>
<tr>
<td>Blocked</td>
<td></td>
</tr>
</tbody>
</table>

Multi-front succeeds

<table>
<thead>
<tr>
<th>M2</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src</td>
<td>tgt</td>
</tr>
<tr>
<td>Blocked</td>
<td></td>
</tr>
</tbody>
</table>
Techniques for Run Time

• More efficient routing algorithms (cont.)
  – Stay away from more time consuming algorithms
    • Shape based, gridless routers
    • Can achieve gridless routing effect with gridded router
      – Gridless routing cause space fragmentation, not good for early iterations
      – Can achieve gridless effects by using finer grids – good enough in practice
  – Search cores support few basic rules
    • Incorporating new rules directly into search core will kill the run time
    • Keep new complex design rules out of search core
      – More later
    • Only keep most commonly supported rules in search core
      – Spacing between different nets
      – Staggering distance
      – Antenna layer hopping
      – ...
  – DRC convergence has a huge effect on run time
    • Multiple iteration DRC convergence
    • Run time is determined by how fast DRC converge
      – Resolving DRC too fast cause longer wires, more vias, and entangled routes
        » Bad quality and longer run time
      – Resolving DRC too slow leads to many iterations – longer run time
      – It is an art to balance the speed of DRC convergence
Techniques for Run Time

• Hierarchical routing – break up the complexity
  – More routing stages – global routing/track assign/detailed routing
  – Hierarchical global routing
  – Multilevel routing
  – Partition/corridor based iterative routing

Techniques for Run time

• Take advantage of the latest hardware development
  – Linux multi-processor computer farms are everywhere
    • Multithreading for multi-processor machines
    • Distributed computing for computer farm
    • Combined for both
Techniques for Run Time

- Threading versus distributed computing

<table>
<thead>
<tr>
<th></th>
<th>Threading</th>
<th>Dist. Comp.</th>
</tr>
</thead>
<tbody>
<tr>
<td># avail proc</td>
<td>Fewer</td>
<td>More</td>
</tr>
<tr>
<td>Memory usage</td>
<td>More work mem</td>
<td>Less subtask mem</td>
</tr>
<tr>
<td>Data structure req.</td>
<td>Modular, clean</td>
<td>No requirement</td>
</tr>
<tr>
<td>New router difficulty</td>
<td>More, but for better programming</td>
<td>Less</td>
</tr>
<tr>
<td>retrofit difficulty</td>
<td>Very high</td>
<td>Little</td>
</tr>
<tr>
<td>New proc cost</td>
<td>Cheap</td>
<td>Very expensive</td>
</tr>
<tr>
<td>Proc comm. cost</td>
<td>Cheap, easy</td>
<td>Expensive, difficult</td>
</tr>
<tr>
<td>Parallel style</td>
<td>Smaller, interacting, fast changing subtasks</td>
<td>Larger, non-interacting, slow changing subtasks</td>
</tr>
</tbody>
</table>

Techniques for Run Time

- Multithreading
  - Hardware readiness
    - Dual-core processors are common nowadays
    - Multi-processor machines are common also
      - 2 – 4 processor machines are cheap main stream machines
  - Offers significant scalable speedup with relatively low efforts
    - Much easier to obtain scalable speedup compared to algorithm improvement
Techniques for Run Time

• Multithreading (cont.)
  – Shared memory processing (SMP)
    • Different processors access and communicate through
      shared memory
    • Conflicting concurrent access to memory is protected by
      good modular programming, clean task division, and
      locking

Techniques for Run Time

• Multithreading (cont.)
  – Modular/well designed data structure – good practice
    anyway
    • No or few global variables
      – Exception: data that do not change in threads
    • Identify global data structures shared by threads
      – Can they run into contentious situation? Minimize
        contention
      – Minimize contention at partition level
        » Do not pick overlapping partitions
        » Avoid bin lock by schedule partitions that are far
          enough
        » …
Techniques for Run Time

• Multithreading (cont.)
  – Modular data
    • Group data to minimize contentious data structures
      – Separate contentious data from non-contentious data in global data structure
      – Choose thread specific data structure over global data structure
    – Different levels of data caching to reduce dependency on global data
      • Two tie data – global persistent data and thread specific working data (DRC)
        – Thread specific data is checked out at beginning, and checked in at the end
        – Great for memory usage also
        – Example - DRCs
  – Contention prevention – partition to break interactions
    • Routing is partition based – design rules are mostly area based
    • Pick non-adjacent partitions to multithread
      – No area conflicts, less other conflicts
      – Still desirable to expand out continuous partition front for uniform partitions – less misalignments
    • Break shapes across partitions, or avoid partitions sharing shapes
Techniques for Run Time

- Multithreading (cont.)
  - Contention prevention – lock design
    - Design data structures to minimize lock needed for frequently accessed data
    - Balance between run time, memory, complexity
      - Place lock at the lower level to minimize contention, at the cost of run time, memory, and more complicated control
      - Place lock at the higher level to trade off above
      - Example – global binning structure for geometry query

![Lock Diagram]

- Use scheduler to reduce waiting for lock
  - Example: need net lock for antenna
    - Use a round robin scheduler in each thread to schedule nets
    - Reduce the amount of lock due to different threads working on the same net

![Scheduler Diagram]
Techniques for Run Time

• Multithreading (cont.)
  • Non-determinism
    – Unless tasks are totally independent, will have non-determinism
      » Could be challenging for debugging
      » Will not always produce the same results, but should produce similar results
    – Reduce non-determinism
      » No hash on pointer
      » Thread specific random number generator
      » Use algorithms that are as order independent as possible
      » …

• Distributed computing
  – Divide routing problems into (almost independent) multiple subtasks, and send the subtasks to different processes on different processors and/or machines with minimum communication
  – Has more processors available
  – Subtask overhead is high – smaller number of larger subtasks
  – Subtasks need to be as independent as possible
    • Communication between processes is difficult and expensive
    • Certain rules such as antenna rule is not localized, therefore difficult with distributed computing
  – As a result, the scalability and quality using distributed computing is usually not as good as for multithreading
Outline of Part II

- Objectives and new challenges for industrial routers
- Techniques for run time challenges
- Techniques for capacity challenges
- Techniques for design rules challenges
- Techniques for DFM/DFY challenges

Techniques for Capacity

- Better infrastructure design
  - Think of memory as your own money – be stingy
  - Go after every bit in highly repeated data structures
  - Use bit fields
Techniques for Capacity

• Two tiered in memory data storage
  – Store non-derivable persistent data in as lean form as possible
  – e.g., use center line to represent routing shapes
  – Derive partition level data in more run time friendly ways – e.g., fully instantiate routing related shape information
  – Best balance between data size and run time

Routing

M1 wire

M2 wire

M1M2 via

Detailed representation

M1/M2 wire: x1, y1, x2, y2, layer
Low surround/cut/high surround: x1, y1, x2, y2, layer

Abstract representation

(x1, y1, lay1, widIdx1)

(x2, lay2, widIdx2)

(y2, lay3, widIdx3)

Total: 7 words

Total: 25 words

Techniques for Capacity

• Child process
  – Very useful to break 32bit 4G limit
  – Might still help memory caching for better speed for 64bit

• Distributed computing
  – Smaller distributed subtasks, which consume less memory per subtasks
Outline of Part II

• Objectives and new challenges for industrial routers
• Techniques for run time challenges
• Techniques for capacity challenges
• Techniques for design rules challenges
• Techniques for DFM/DFY challenges

Techniques for Design Rule

• DRC analysis
  – Trend - polygon based
    • Past rules are rectangular based
      – Less complexity
    • No polygon generation time
    • More and more rules are polygon based
    • Routing shapes are rectangles
    • Difficult and inefficient to convert polygon based rules to rectangle based rules
    • Balance tipping towards polygon manipulations
    • Bite the bullet and maintain polygons along rectangles

![Diagram of Metal1]
Techniques for Design Rule

• DRC analysis (cont.)
  – DRC annotation
    • Routing shapes are still rectangles
    • Need to map DRCs from polygon to relevant rectangles

Techniques for Design Rule

• Search core
  – Search graph (maze map): only blocked by basic spacing rules
    • Heavy development needed if introduce new rules
    • Significant run time increase is expected for new rules
    • Very difficult if possible to block maze map for rules depending on routing pattern of to be routed wires
  – Search core: only consider as few constraints as possible besides maze map blockage
    • Very difficult to introduce new rules in the middle of search
    • Significant run time increase is expected for new rule
    • Changes will cause stability issues in routing core in continuous way
Techniques for Design Rule

- **Search core (cont)**
  - Search core avoid resolve DRCs by avoiding DRC areas
    - DRC areas are mapped into maze map
    - Extra cost are added for DRC areas during routing
    - Extra DRC cost decays with a carefully designed schedule
      - Slow decay causes massive over blockage
      - Fast decay leads to DRC oscillation
  - Advantages – scalable search core, no development, memory, and run time penalty for routing search, work well for less frequent DRCs
  - Disadvantages – Requires more search and repair, expensive and does not work well for high frequency DRCs

Techniques for Design Rule

- **Complex rule DRC fixing example – end of line spacing rule**

![Diagram of end of line spacing rule](image)
Techniques for Design Rule

- Non-reroute techniques
  - Techniques
    - Patching
    - Shifting
    - Rotating
  - Advantages – fast, converging, and easy
  - Disadvantages – greedy, limited improvement, possibly more routing resources required
- Example – min edge rule

Outline of Part II

- Objectives and new challenges for industrial routers
- Techniques for run time challenges
- Techniques for capacity challenges
- Techniques for design rules challenges
- Techniques for DFM/DFY challenges
Techniques for DFM/DFY

- **CAA/wire spreading/wire widening**
  - Critical area - the region where, if the center of a random defect with certain size falls on, it will cause circuit failure (yield loss)
    - A good metric for yield
    - Reduction of Critical Area increases defect-limited yield

  \[ A_{cr} = \int_{x_0}^{\infty} A_{cr}(x) f(x) dx \]

  - Conductive Defect Causing Short
  - Non-Conductive Defect Causing Open

  \( A_{cr} \): average critical area
  \( x_0 \): smallest particle size
  \( x \): defect size (diameter)
  \( A_{cr}(x) \): critical area for defect size \( x \)
  \( f(x) \): defect size distribution function

- **CAA/wire spreading/wire widening (cont.)**
  - Critical area (cont.)
    - Critical area value varies with defect size
    - For a given layout, the larger the defect size, the larger the critical area
    - Average critical area is usually used

  [Diagram showing conductive and non-conductive defects causing short and open failures]
Techniques for DFM/DFY

• CAA/wire spreading/wire widening (cont.)
  – Current flow

    Design Ready for Signal Routing
    ↓
    Density-Driven Global Route
    ↓
    Density-Driven Track Assign
    ↓
    Detail Route and S&R
    ↓
    Wire Spreading/widening
    ↓
    Critical Area Analysis

Techniques for DFM/DFY

• Density driven global routing – distribute unused space more evenly across design
  – Reduce congestion overflow threshold
    • May cause significant wire/via increase – careful tuning
    • May interact with real routing congestion
      – Non-constant/non-linear over-congestion cost
      – Reduce conservatism as iteration goes
  – Better approach – have another congestion map for wire spreading
    • Better separation of real congestion and wire spreading
    • Tune wire spreading congestion cost against real congestion cost
Techniques for DFM/DFY

- Post DR wire spreading
  - Sub-pitch tracks for more continuous wire spreading
  - Ripup and reroute with bigger spacing requirements
- Better approach – wire spreading during detailed routing with softer spacing rules on wires together with regular spacing
  - Up to this point, each wire has one spacing rule
  - No tool does this yet

Techniques for DFM/DFY

- Via doubling - double via improves yield during chip manufacturing
  - It fails 10X-100X less than single via
Techniques for DFM/DFY

• **Via doubling**
  – Rotates and swaps line via arrays to best fit into available space

![Diagram showing via doubling process]

• Via doubling (cont.)
  – Mostly done as a post routing process
    • Pros: Does not affect overall DRC convergence
    • Cons: limited by routing results, timing variance
  – Newer approaches
    • Support soft spacing rules around vias to reserve space
    • Double via before post route timing closure, and keep doubling via after timing optimization

![Before and After Via Optimization images]
Techniques for DFM/DFY

- Litho aware routing
  - Many routing rules to compensate for lack of simulation
    - Via proximity
    - Line-end
    - Length based
  - Need to consider litho-effects w/o exploding routing rules

Techniques for DFM/DFY

- Litho hot spot fixing
  - Run litho compliance check (LCC), identify hot spots and replacement patterns
  - Replace with patterns suggested by LCC
  - Fix possible resulting DRCs
Techniques for DFM/DFY

Pattern dependent effects dictate a need for correct type and amounts of metal fill.

• Metal fill
  – Density driven metal fill is not good enough
Techniques for DFM/DFY

- **Model based CMP**
  - Driven by thickness simulation
  - Many patterns to choose from for least thickness variation

- **Rule-Based**
- **CMP-Aware Model-Based**

Pattern selection based on simulation

- **Density Only**
- **Density and Thickness**

---

**Techniques for DFM/DFY**

- **Future works**
  - New area in routing, a lot of on going projects
  - Need to have a unified yield analyzer and cost function to drive optimization

- **Example**
  - via doubling improve yield
  - Critical area decrease yield
  - Complex geometries decrease yield
  - Is via doubling good for yield?

After Via Optimization
(double vias)
Part III
Analog and Mixed Signal Issues

Rob A. Rutenbar
Professor, Electrical & Computer Engineering
rutenbar@ece.cmu.edu

And Now, For Something Completely Different...
Why Analog Matters: Many “Mixed-Signal” SoCs

- Telecom
- Automotive
- Consumer
- Medical
- Computers & Networks

% Digital Chips with Analog Content

- 12% in 2000
- 30% in 2003
- 75% in 2006

(Source: IBS 2003)

Routing in the Digital World: Summary

- Capacity issues
  - 1-10 million placed instances
  - Millions of wires and pins

- Nanometer issues
  - Increasingly complex DRC rules
  - More (and conflicting) DFM rules

- Complexity issues
  - Billions of shapes
  - Coupling, timing closure, yield and manufacturability iterations
  - Don’t want to spend CPU months

- Problems look like this
  - IBM network switch
  - IP blocks + N million gates

(Courtesy Juergen Koehl, IBM)
**Is Analog/Mixed-Signal Problem Basically Same?**

Are we just routing a big set of analog pins with a million min-width wires?

**NO.**

---

**Backing Up: What Exactly Gets Routed, Digital-Side?**

- Gates (standard cells) and IP blocks (memory, core, etc)
  - Gates in rows, with large interspersed macro blocks
  - Wires over the top of everything (except a few very sensitive macros)

- Soft IP: CPU Core
- Random Logic
- Hard IP: Memory, etc
- More random logic

---

**© R.A. Rutenbar 2006**
What Do We Route on Analog/Mixed-Signal Side?

- **Device-level designs**
  - Unique problems for large, geometrically complex devices

- **Circuit-level designs (cells)**
  - Typically 10 – 100 devices
  - Analog: “like a library element”

- **System-level designs**
  - Block level designs, looks more like digital-side problems

About This Talk

- Walk “up” the routing hierarchy for analog side
- Point out salient differences from “big digital” routing
- Mention some approaches for solutions – and the many open problems here
Background: Low-Level Routing

- **First question:**
  - Why are we worrying about routing problems at these seemingly “low” levels of design hierarchy?

- **Said differently**
  - Isn’t this what libraries are supposed to hide from system designers?

Role of Digital Cells in Digital System Design

- **Digital ASIC design**
  - Usually **starts** from assumed library of cells (usually some cores too)
  - Supports changes in cell-library; assumed part of methodology
  - Cell libraries heavily **reused** across different designs

Diagrams showing the flow of digital HDL, Logic Synthesis, Tech Mapping, and Physical Design, with a Gate-Level Cell Library.
Where Do Digital Cells Come From?

- Foundries: Optimized for this fab
- 3rd Party IP: Emphasize portability, quick use
- Manual, Custom Design: Proprietary or custom library

Where Do Analog Cells Come From?

- From analog designers
  - Mainly manual design
  - Often, manual redesign
  - Almost no reuse

- Why is this?
  - Analog exploits, rather than abstracts, low-level physics of devices
  - Individual devices designed for precision
  - Circuits sensitive to all aspects of device and interconnect and environment
Why No Analog Libraries: Dimensionality

- Problem: many continuous specs for analog cells

\[ \begin{array}{c}
\text{Spec=LOW} \\
\text{Spec=HIGH}
\end{array} \times \begin{array}{c}
\text{variants for ALL combinations}
\end{array} = \sim 1000 \text{ variants for just this cell} \]

- Can't just build a practical-size, universal analog library
  - Note, people still do “library” some useful cells as hard IP (layouts), but still expect most cells you need will not be in your average library

About This Talk

- Routing at device level
Device-Level Routing Issues

- **Focus is always on precision**
  - Want precise electrical characteristics, or matching among several devices, or precise ratios among devices

- **Central issues**
  - Analog devices are often larger; e.g., a 4000/4 FET is not unusual
  - Analog devices are often designed and laid out as a careful connection of many small, well-matched unit-size devices
  - M-factors: 1 device → $M$ matched, inter-digitated devices/fingers in layout
  - Guard-ring(s) common for electrical isolation

- **Result**
  - Even 1 device may end up with a complex, large geometric layout

---

Example of Digital vs Analog Geometry Disparity

<table>
<thead>
<tr>
<th>Digital FET</th>
<th>Analog FET</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Digital FET" /></td>
<td><img src="image" alt="Analog FET" /></td>
</tr>
</tbody>
</table>

*Device-level routing*
Device-Level Layout Precision Example

- Consider a resistor which uses a resistive poly layer

Low-precision R, poly snake resistor
- Resistive material
- Metal-strapped pins

High-precision R, add dummy bars at ends, well and guard ring

Higher-precision R, poly bars with all-metal interconnect

Interdigitated pair of precise-ratioed 2:1 resistors

Industrial Example: Large Resistor Array

- New problem: who creates this intra-device wiring?
  - Could be procedural (eg, SKILL, PCELL), ie, it’s not routed, it’s placed
  - Could be a real router: a general router, or one specifically adapted to this
    - Small problems (100-1000 wires), not many layers (poly + few metals)
    - Must deal with analog-centric matching/balance/symmetry requirements

Courtesy Neolinear

© R.A. Rutenbar 2006
## Intra-Device Routing Issues

- **Layers**
  - You are going to have to route on poly, and deal with all the unpleasant device-level shapes rules associated with poly in scaled CMOS

- **Pins**
  - On digital side, people take great pains to make pins “nice” = “little metal boxes”
  - On analog side – *not* always true. May have to hit messy device shapes

- **Wire widths**
  - Much *more* about this later, but – often, *not* minimum width
  - Wires are carrying more current (analog biasing, transducer signals, etc)
  - Means they get sized up for (1) ohmic drop and (2) electromigration rules
  - Also, designers get very fussy about via shapes, # of cuts, etc, for these wires

---

## About This Talk

- **Routing at circuit/cell level**
Routing in the Circuit/Cell Level Design Flow

- Basic tasks

  - From sized schematic
  - Design cell footprint & floorplan
  - Design individual device geometries
  - Place & route devices, optimize area, coupling, etc.

Problems Look Like This: **Route This Placement**

- **Concern 1: Congestion**
  - Wire-to-wire and wire-to-device
  - Do we have enough "white space" and "over device space" to embed all the wires?
  - Can the wires all take short, straight "natural" paths (designers get way upset if not)

- **Concern 2: Constraints**
  - Have I met all the analog-specific geometric constraints?
  - Have I messed up any subtle electrical constraints?
**Congestion: Geometric Complexity**

- Inside of an analog cell is a dense, complex place to do wiring
  - Dense design rule interactions – getting much worse as we scale
  - Many wires need to be wide(r) to carry analog current levels
  - Want to use few metal layers, but many devices may have pins strapped with metals, or be restricted for routing over in lower metals: obstructions galore
  - Difficult, tight interactions with placement to ensure routability

**Congestion: Contrast With Digital Routing**

- We use *hierarchy* in digital routing: **Global Routing Grid**
Global Routing for Circuit Level Analog?

- Not such an obvious idea here
  - GBOXes in big digital design can be 50, 100, 200 wire tracks across
  - The whole analog circuit may be on the order of several such GBOXes
  - Handling wide range of wire widths is also challenging here

- Severe “aesthetic” concerns
  - Nobody really cares exactly where the wires go in a big digital chip
  - But when humans route analog, most wires are short, straight, minimal
  - Designers hate it when routers don’t produce similar visual results, ie, big penalties for even small “kinks”

Big Digital Routes: Nobody Looks At Them All

Gosh, is it just me, or does wire #1,034,237 look odd…? Oh Brad – I was just thinking the same thing!
Other Side: Analog Designers Obsess Over All Wires

Hey, why is that bend in that wire, right there?

…and I really don’t like the look of that via!

This Is The “It All Fits On One Screen” Problem

- Even big cells (100+ devices) may fit on one editor screen
  - ...which means, it's easy to go and look at every single wire
  - This is a level of aesthetic scrutiny most digital routes never get
Another, Rather Dense “Fits On A Screen” Example

Circuit-Level Routing Issues

- Need negotiation-based, ripup-reroute, iterative routing
  - Cannot just route each wire once and assume they all go down “nice”
  - Severe density, congestion issues even in small cells

- Need to accommodate a wide range of wire widths (+ via cuts)
  - It just never happens that they all go down at min width
  - Either need a fully shape-based engine, or a very fancy gridded router

- But, also wide range of analog-specific geometric features...
**Analog-Specific Geometric Features**

- Unique attribute of analog is need to balance wiring
  - Support mirror-symmetric routing, cross-symmetric routing, varieties of incomplete/partially symmetric routing… etc…
  - Guarantee that all routing is exactly geometrically mirrored

**A Few of the Options for Symmetric Nets**

- **Mirror symmetry**
  - Complications
    - There are lots of forms of symmetries, letting designers specify them easily is tough
    - Sometimes, the pins are “not quite symmetric” or there are a few extra non-symmetric pins on the net. Still need to route “most” of the net as symmetrically as possible

- **Cross symmetry**
Symmetric Routing: Basic Trick

- Only route one wire, but reflect obstacles from other side across symmetry line, into one shared left-right model of space.

Balanced Routing

- Symmetry is the geometrically easy form of “balance.”
  - Sometimes, you don’t have the option, if pins not symmetric.
  - In these cases, routing solutions usually look like channels, with extra wiring, and very carefully controlled vias+stubs to balance (capacitance) on nets.

Want nets 1-2 to have same length on each layer, same # vias.

Ditto for nets 3-4, 5-6.
Detailed Solution to Balanced Route Example

Poly M1 Poly-M1 Via

M2 M1-M2 Via

Nets 1, 3, 5... matching 2, 4, 6

Each net pair has ~same length on each layer, same num and type of vias

Observations
- Not every dense arrangement of pins (with obstacles) can be routed
- Much of this problem is getting the placement right, with space reserved
- Routing here much more like channel-ed problems, with more constraints
- Can attack these as routing problems, or as “wire placement” problems
About This Talk

- Routing at system-level

What Does System-Level Routing Look Like?

- Mostly, like a big version of the circuit-level problem
  - Routing 10s – 100s of basic cells together
  - 1K – 10K nets, roughly, connecting ~25K analog transistors + digital stuff
  - Very few min-width nets, lots of balance constraints + avoidance issues
- Also, surprisingly, like the device-level problem
  - Lots of repeated structures (e.g., bits of converter), often want a highly stylized, patterned kind of routing, just like for device-level tasks

Ex: 14-bit 150-Ms/s 0.5um CMOS DAC


Courtesy Georges Gielen, K.U. Leuven
Small System Ex: Dual-Tone Multi-Frequency Decoder

- RAM (256 x 16)
- RAM (128 x 16)
- RAM Compiler
- PLL Clock
- Results Converter (FFT)
- Std Cell Place/Route
- ROM (512 x 16A)
- ROM Compiler
- I/O pads
- Glue Logic
- DSP Core
- Std cell place/route

Pushing Inside the PLL

- Looks like a macroblock digital design – without all glue logic
- Counter (3-bit)
- Divider (2-bit)
- Phase Detector
- Charge Pump
- Voltage-Controlled Oscillator
- Cadence Generic PDK 0.18um 6LM Generic Process

[Courtesy Artisan, Cadence]

[Courtesy Cadence]
Bigger Example: Industrial ADC

[Gradient et al, IEEE Electronic Design Proc Workshop, EDP2002]

What’s Different? Coupling Avoidance Issues

- Digital: A small set of relatively simple, *discrete* fix-it options

- Analog: Not so easy.
  - Much closer attention to each critical wire’s parasitics, crossings, neighbors, etc.
  - Still use spacing / shields a lot, but more detailed analysis of parasitic impacts
What’s Different: Power Distribution

- **Digital:** Grid is not really routed
  - Core rings, around whole chip, around individual macroblocks
  - Stripes to bring power to inside
  - Do DC drop analysis, if you don’t like, add more power stripes

- **Analog:** Grid is really routed
  - Maybe not all of it, but lots of it
  - No nice row/col pattern structure
  - Also, need to deal with sizing for ohmic drop and electromigration

---

Summary

**Digital routing**
- Capacity: 1-10M nets/pins
- Scalability: huge data, CPU time
- Route mainly system level
- Negotiation-based rip/reroute
- Rising DFM complexity hurts
- More **gridded** than shape based
- Mostly min width nets
- Simple coupling fix-its

**Analog routing**
- Capacity: ~100–10K nets, ~25K devices
- Scalability: it’s **electrical** complexity
- Route **devices, circuits, & systems**
- Negotiation-based rip/reroute
- Rising DFM complexity hurts
- More **shape-based** than gridded
- Mostly **not** min width nets
- **Not** simple coupling fix-its
- Analog-specific symmetry/balance/etc
- Power grid routing / sizing
To Learn More: Mixed-Signal CAD

- **Computer-Aided Design of Analog Integrated Circuits and Systems**
  - Rob A. Rutenbar, Georges G. E. Gielen, Brian A. Antao, Editors
  - Hardcover: 768 pages
  - Publisher: IEEE
  - Published: April 2002
  - ISBN: 047122782X

- **Book** is a collection of essential papers on all aspects of analog and mixed signal synthesis, modeling, layout, etc. Many of the results shown here appear in these papers.