Layout Compaction

Prof. Kurt Keutzer
Prof. A. R. Newton
Prof. Sanjit Seshia
UC Berkeley
Prof. M. Orshansky
UCB → U of Texas
RTL Design Flow
Physical Design: Overall Flow

Input
- Read Netlist
- Floorplanning

Floorplanning
- Initial Placement
- Routing Region Definition
- Global Routing
- Cost Estimation
- Routing Region Ordering
- Detailed Routing
- Cost Estimation
- Placement Improvement
- Routing Improvement

Placement
- Routing
- Output
- Compaction/clean-up
- Write Layout Database
Layout Compaction

Input

Floorplanning

Placement

Routing

Output

Read Netlist

Floorplanning

Initial Placement

Routing Region Definition

Global Routing

Cost Estimation

Routing Region Ordering

Detailed Routing

Cost Estimation

Compaction/clean-up

Write Layout Database

Placement Improvement

Routing Improvement
Compaction: Introduction

- After P&R, the layout is functionally complete
- Some vacant space may still be present in the layout
  - Due to non-optimality of P&R
- Compaction = removing vacant space
  - Improves cost, performance, and yield
- Key for high-performance full-custom layouts
- Standard cells – only channel heights may be minimized
  - But channel compactors are near-optimal
Layout Compaction
Compaction: Introduction

- Compaction tries to minimize total layout area while
  - Respecting design rules and designer-specified constraints

- Three ways to minimize the layout area
  - Reducing inter-feature space
    - Check spacing design rules
  - Reducing feature size
    - Check size rules
  - Reshaping features
    - Electrical connectivity must be preserved
Cross-Section of CMOS Technology

Jan Rabaey
Design Rules

- Interface between designer and process engineer
- Guidelines for constructing process masks
- Unit dimension: Minimum line width
  - scalable design rules: lambda parameter
  - absolute dimensions (micron rules)
# CMOS Process Layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Color</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well (p,n)</td>
<td>Yellow</td>
<td><img src="yellow_image" alt="Yellow" /></td>
</tr>
<tr>
<td>Active Area (n+,p+)</td>
<td>Green</td>
<td><img src="green_image" alt="Green" /></td>
</tr>
<tr>
<td>Select (p+,n+)</td>
<td>Green</td>
<td><img src="green_image" alt="Green" /></td>
</tr>
<tr>
<td>Polysilicon</td>
<td>Red</td>
<td><img src="red_image" alt="Red" /></td>
</tr>
<tr>
<td>Metal1</td>
<td>Blue</td>
<td><img src="blue_image" alt="Blue" /></td>
</tr>
<tr>
<td>Metal2</td>
<td>Magenta</td>
<td><img src="magenta_image" alt="Magenta" /></td>
</tr>
<tr>
<td>Contact To Poly</td>
<td>Black</td>
<td><img src="black_image" alt="Black" /></td>
</tr>
<tr>
<td>Contact To Diffusion</td>
<td>Black</td>
<td><img src="black_image" alt="Black" /></td>
</tr>
<tr>
<td>Via</td>
<td>Black</td>
<td><img src="black_image" alt="Black" /></td>
</tr>
</tbody>
</table>
Intra-Layer Design Rules

- **Well**
  - 0 or 6
  - 10

- **Active**
  - 3
  - 3
  - 2

- **Select**
  - 2
  - 2

- **Polysilicon**
  - 2
  - 2

- **Metal1**
  - 3
  - 3

- **Metal2**
  - 4
  - 3

- **Contact or Via Hole**
  - 2

Jan Rabaey
Transistor Layout – Inter Layer Rules
Via’s and Contacts

Jan Rabaey
Different Approaches to Compaction

- One dimensional vs. two-dimensional compaction
- 1-D compaction
  - Components moved only in x- or y-direction
  - Efficient (nearly linear-time) algorithms available
- 2-D compaction
  - Components may be moved both in x- and y-direction
  - More effective compaction
  - NP-hard
    - Determining how x and y should interact to reduce area is hard!
- Historical interest: Constraint-graph based compaction vs. virtual grid based compaction
  - Virtual grid methods are fast and simple. Results in larger area.
Historical interest: Virtual Grid Compaction

- Every feature is assumed to lie on a virtual grid line.
- Required to stay at grid locations during compaction – no distortion of topology.
- Compact by finding the min possible spacing between each adjacent pair of grids.
  - Min spacing is given by the worst-case design-rule for any feature on the grid.
- Advantage: algorithm is simple and fast.
Need something different: Constraint-based Compaction

Early tools: Floss, Cabbage, SLIP, SLIM, Sticks
Constraint-based Compaction

- Features represented as rectangular shapes
  - Left-most edge becomes “origin” rectangle
- Layout constraints are imposed by design rules
  - Min spacing (separation) design rules
- Want to compact to the left

Want to minimize width
Two Steps in Constraint-based Compaction

1. Constraint generation
   - Constraints are naturally represented as inequalities
     - Example: if $c$ is to the right of $b$, and they have to be at least 2 units apart: $c \geq b + 2$
   - These can be represented in a graph

2. Constraint solving
   - Use graph algorithms
Constraint Generation: A Scan-Line Approach
Constraint Generation: A Scan-Line Approach

\[ d \geq \text{origin} + 1 \]
Constraint Generation: A Scan-Line Approach

\[ d \geq \text{origin} + 1 \quad e \geq d + 3 \]
Constraint Generation: A Scan-Line Approach

\[
\begin{align*}
    d & \geq \text{origin} + 1 \\
    d & \geq b + 2 \\
    e & \geq d + 3 \\
    b & \geq \text{origin} + 1
\end{align*}
\]
Constraint Generation: A Scan-Line Approach

\[ d \geq \text{origin} + 1 \quad e \geq d + 3 \]
\[ d \geq b + 2 \quad b \geq \text{origin} + 1 \]

No more constraints involving \(d\)
All Generated Constraints

\[ b \geq \text{origin} + 1 \quad c \geq b + 2 \]
\[ d \geq \text{origin} + 1 \quad d \geq b + 2 \]
\[ e \geq c + 1 \quad e \geq d + 3 \]
Constraint Generation Summary

- Constraint generation is the most time-consuming part of constraint-based compaction
  - Getting harder in current technologies
  - In the worst case, there may be a design rule between every shape of layout
    - In reality only a small local subset of constraints are needed
  - Approached naively $O(n^2)$

- Approach:
  - First generate connectivity (grouping) constraints
  - In generating separation constraints, need to define a set of non-redundant constraints

- Scan-line algorithm
- Shadow-propagation algorithm
How do we solve these constraints?

For now, assume we only have separation constraints.

What is a mathematically efficient way to solve these constraints?
Use graphical structure

Constraint graph represents all constraints

\[ b \geq \text{origin} + 1 \]
\[ c \geq b + 2 \]
\[ d \geq b + 2 \]
\[ e \geq c + 1 \]
Problem formulation - 1

- Use a labeled directed graph
- \( G = \langle V, E \rangle \)
- Vertices layout objects
- Edges represent constraints between vertices
- Labels represent constraint values
- Now what do we do with this?

Hint: What problem does this remind you of?
Problem formulation - 1

- Use a labeled directed graph
- $G = \langle V, E \rangle$
- **Vertices** layout objects
- **Edges** represent constraints between vertices
- **Labels** represent constraint values
- Now what do we do with this?
Goal of 1-D compaction is to generate a minimum width layout

Determination of minimum width is equivalent to solving a longest path problem

- Calculate longest path from origin to each vertex
  - Longest path to a vertex is its x-coordinate
  - Longest of these is the width of the layout

Theoretically, run time is $O(|V|+|E|)$
- Why?

Practically, run time is close to linear in $|V|$, the size of the layout!
Compute the longest path in a graph \( G = <V,E,\text{constraints},\text{Origin}> \) (\text{constraints} is a set of labels, \text{Origin} is the super-source of the DAG)

\[
\text{Forward-prop}(W)\
\quad\text{for each vertex } v \text{ in } W\
\quad\quad\text{for each edge } <v,w> \text{ from } v\
\quad\quad\quad\text{value}(w) = \max(\text{value}(w), \text{value}(v) + \text{value}(w) + \text{constraint(<v,w>>))}
\quad\text{if all incoming edges of } w \text{ have been traversed}
\quad\quad\quad\text{add } w \text{ to } W
\]

\text{Longest path}(G)
\quad\text{Forward\_prop}(\text{Origin})

\text{Are we done?}
DAG $\rightarrow$ Arbitrary Graph

- Separation constraints yield inequalities, all in the same direction
  - Corresponding graph is a DAG

- What about grouping constraints?
  - These are typically equalities

- Sliding ports
  - Yield inequalities in both directions (upper and lower bounds)

- $O(|V| + |E|)$ algorithm only works on DAGs, but the constraint graph can contain cycles
First Elaboration - Equality Constraints -1

- **Grouping constraints**
  - Features from same circuit component
  - Need to be moved together
- Described by *equality constraints*

\[ e = c + 1 \]
First Elaboration - Equality Constraints -2

e = c + 1

\[ e \geq c + 1 \]
\[ c \geq e - 1 \]
Reflecting Equality Constraints

What challenges does this pose to our algorithm?

e \geq c + 1

c \geq e - 1
Dealing With Legitimate Cycles

- Topological longest path algorithm can’t handle graphs with cycles, only DAGs
- Can use a Bellman / Moore algorithm for general graphs
  - Run time is O(|V|*|E|)
  - Run time of longest path algorithm on DAG is O(|V|+|E|)
Alternative Approach to Equality Constraints

Handle equality constraints independently
Build independent graph for equality constraints
Express inequality constraints between designated representatives

\[ e = c + 1 \]
Handling Equality Constraints

\[
\begin{align*}
\text{origin} &= x + 1 \\
\text{origin} &= y + 1 \\
z &= y + 1 \\
\text{origin} &= z + 2 \\
v &= w + 2 \\
w &= u - 2 \text{ i.e. } u = w + 2 \\
u &= t - 3 \text{ i.e. or } t = u + 3
\end{align*}
\]
Handling Equality Constraints

Generally, this is known as the union-find algorithm.

- $v = w + 2$
- $w = u - 2$ or $u = w + 2$
- $u = t - 3$ or $t = u + 3$
- $v = x + 2$
- $w = (v - 2) = x (\text{origin} - 1) + 2$
- $v - 2 = \text{origin} - 1 + 2$
- $v = \text{origin} + 3$
- $\text{origin} = x + 1$
- $\text{origin} = y + 1$
- $y = z + 1$
- $\text{origin} = y + 1$
- $y = z + 1$
- $\text{origin} = y + 1$
- $y = z + 1$
Constraint-Based Compaction: Constraint Solving Summary

Build constraint graph
  if equality constraint
    add to equality constraint graph
  if inequality constraint
    find distinguished representatives of each vertex in constraint
    add constraint between distinguished representatives
Check for cycles in inequality constraint graph
  If cycles exist terminate with error
Solve inequality constraint graph
Solve equality constraint graph
Compaction Enhancements

◆ 1-D constraint-based compaction problem can be formulated optimally and computationally efficiently
◆ In real circuits what we want is often more complex than can be captured in simple linear inequalities of the form:
  ♦ \( e \geq c + 1 \)
◆ Or equalities of the form:
  ♦ \( u = t - 3 \)
◆ For example:
  ♦ Spacing, slack distribution
  ♦ Wirelength minimization
  ♦ Jog introduction
◆ Improving area minimization using:
  ♦ 1 ½ D compaction
  ♦ 2D compaction
Enhancements: Sliding/Spacing Terminals

Python, Sparks
(requires use of upper as well as lower-bound constraints)
Wire-Length Minimization

- Features not on the critical path will be pulled towards a layout edge because they are given their minimal legal spacing
- May lead to increased wire length and slower circuit performance
- Can re-distribute ‘slack’ (the available empty space) to the features not on the critical path
Jogs in “Wires”

Cabbage, SLIP, Dumbo
One-and-a-half Dimensional Compaction

If we could just bump C over

C compacted down

C compacted up

K. Keutzer/R. Newton/S. Seshia
One-and-a-half Dimensional Compaction

◆ Key idea: provide enough lateral movements to blocks during compaction to resolve interferences
◆ Algorithm starts with a partially compacted layout (two applications of 1-D compaction)
◆ Maintain two lists – floor and ceiling
◆ Floor is a list of blocks visible from the top, ceiling is the list of blocks visible from the bottom
◆ Select the lowest block in the ceiling and move it to the floor maximizing the gap between floor and ceiling.
◆ Continue until all blocks have been moved from ceiling to floor.
1-Dimensional Compaction in 2D
1-Dimensional Compaction in 2D

X then Y 1D Compaction

Y then X 1D Compaction
True Two-Dimensional Compaction

- Two dimensional compaction is NP Hard (C. K. Wong, 1984)
- Choosing how two dimensions should interact to produce optimal is hard
- Can formulate as integer-linear programming problem
  - Worst-case complexity is exponential
What makes 2-D compaction hard?

- Two dimensional compaction is NP Hard (C. K. Wong, 1984)
- Choosing how two dimensions should interact to produce optimal is hard
- Can formulate as integer-linear programming problem
  - Worst-case complexity is exponential
Choices -- Vertical or Horizontal Overlap?

\[ a \geq b + 1 \]
Choices -- Vertical or Horizontal Overlap?

\[ a \geq b + 1 \]
Next Month: Optimization, Testing, Verification

Library

HDL

RTL Synthesis

Manual Design

Module Generators

logic optimization

netlist

physical design

layout

K. Keutzer/R. Newton/S. Seshia
Background Material

Handout in Reader:

Algorithms background:
  - shortest paths (read selectively)
  - union-find algorithm (disjoint forest implementation)