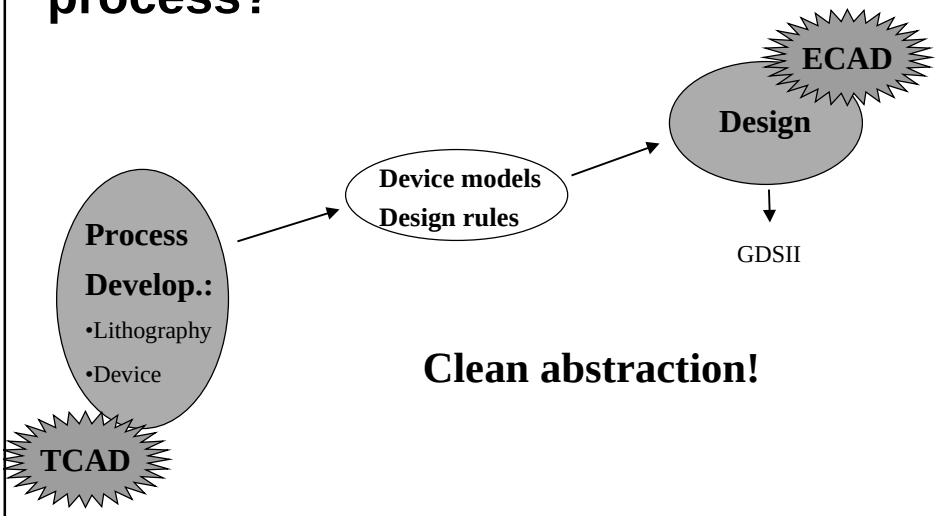
Modern Physical Design:
Algorithm
Technology
Methodology
(Part V)

Stan Chow Ammocore

Andrew B. Kahng UCSD

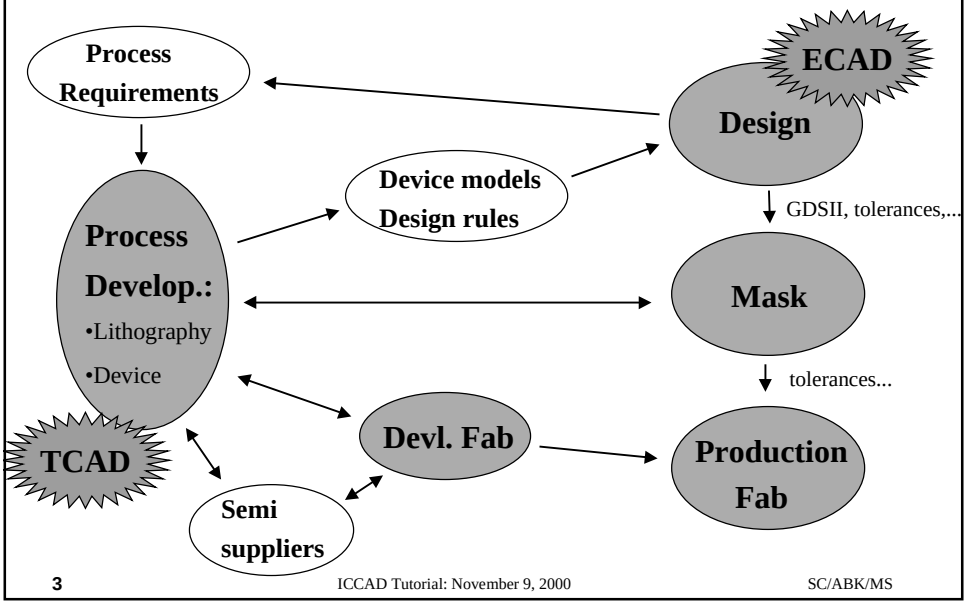
Majid Sarrafzadeh UCLA

What does EDA know about process?



Clean abstraction!

Developmental fab in tight loop



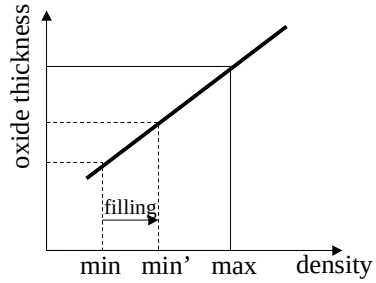
Density Control for CMP

Density Control for CMP

- Chemical-mechanical polishing (CMP)
  - applied to interlayer dielectrics (ILD) and inlaid metals
  - polishing pad wear, slurry composition, pad elasticity make this a very difficult process step
- Cause of CMP variability
  - pad deforms over metal feature
  - greater ILD thickness over dense regions of layout
  - "dishing" in sparse regions of layout
  - huge part of chip variability budget used up (e.g., 4000Å ILD variation across-die)

Min-Variation Objective

- Relationship between oxide thickness and local feature density



- Minimizing variation in window density over layout preferable to satisfying lower and upper density bounds

Density Control for CMP

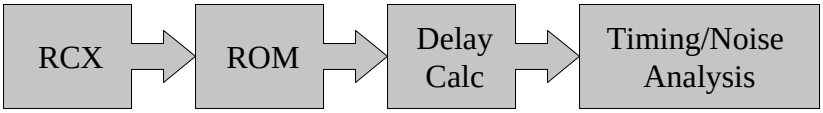
- Layout density control
  - density rules minimize yield impact
  - uniform density achieved by post-processing, insertion of dummy features
- Performance verification (PV) flow implications
  - accurate estimation of filling is needed in PD, PV tools (else broken performance analysis flow)
  - filling geometries affect capacitance extraction by > 50%
  - is a multilayer problem (coupling to critical nets, contacting restrictions, active layers, other interlayer dependencies)

Density Rules

- Modern foundry rules specify layout density bounds to minimize impact of CMP on yield
- Density rules control local feature density for $w \times w$ windows
  - e.g., on each metal layer every 2000um $\times$ 2000um window must be between 35% and 70% filled
- Filling = insertion of "dummy" features to improve layout density
  - typically via layout post-processing in PV / TCAD tools
    - boolean operations on layout data
  - affects vital design characteristics (e.g., RC extraction)
  - accurate knowledge of filling is required during physical design and verification

Need for Density Awareness in Layout

- Performance verification flow:



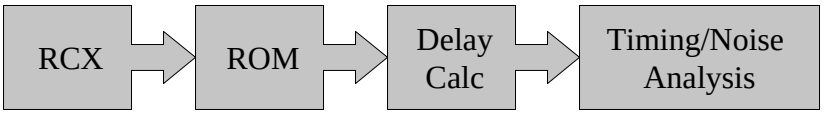
- Filling/slotting geometries affect RC extraction

VICTIM LAYER TOTAL CAPACITANCE (10^{-15} F)			
Same layer-i neighbors?	Fill layers i-1, i+1?	$\varepsilon = 3.9$	$\varepsilon = 2.7$
N	N	2.43 (1.0)	1.68 (1.0)
N	Y	3.73 (1.54)	2.58 (1.54)
Y	N	4.47 (1.84)	3.09 (1.84)
Y	Y	5.29 (2.18)	3.66 (2.18)

Up to 1% error in extracted capacitance
Reliability also affected (e.g. slotting of power stripes)

Need for Density Awareness in Layout

- Performance verification flow:



- Can be considered as ``single-layer'' problem

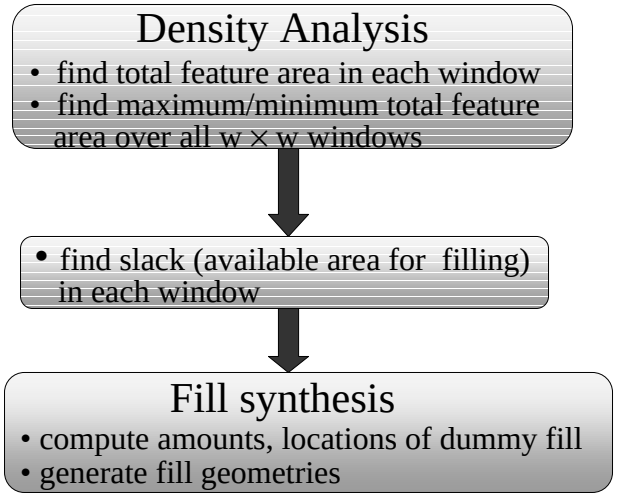
Middle Victim Conductor Total Capacitance (10^{-15} F)			
Fill layer offset	Fill geometry	$\varepsilon = 3.9$	$\varepsilon = 2.7$
N	10×1	3.776 (1.0)	2.614 (1.0)
N	1×1	3.750 (0.99)	2.596 (0.99)
Y	10×1	3.777 (1.00)	2.615 (1.00)
Y	1×1	3.745 (0.99)	2.593 (0.99)

- Caveat: contacting, active+gate layers, other layer interactions

Limitations of Current Techniques

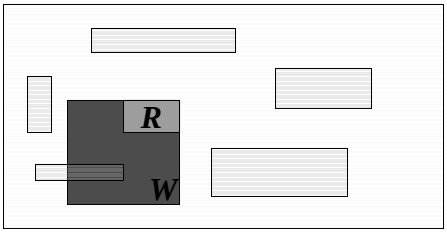
- Current techniques for density control have three key weaknesses:
  - (1) only the average ***overall*** feature density is constrained, while local variation in feature density is ignored
  - (2) density analysis does not find ***true*** extremal window densities - instead, it finds extremal window densities only over fixed set of window positions
  - (3) fill insertion into layout does not minimize the maximum variation in window density

Layout Density Control Flow



Exact Max-Density Window Analysis

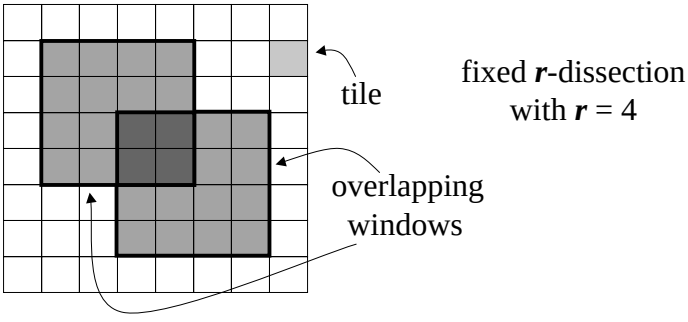
- For each pivot rectangle ***R*** do
  - find density of ***w*** × ***w*** window ***W*** that abuts ***R*** on top and right
  - while ***W*** intersects ***R*** do
    - slide ***W*** right till intersection with other rectangle edge
    - record changes in density



- ***O***(***k***^{2}) algorithm for ***k*** rectangles

Fixed *r*-Dissection Regime

- Feature area density bounds enforced only for ***fixed*** set of ***w*** × ***w*** windows
- Layout partitioned by r^2 distinct fixed dissections
- Each ***w*** × ***w*** window is partitioned in r^2 ***tiles***

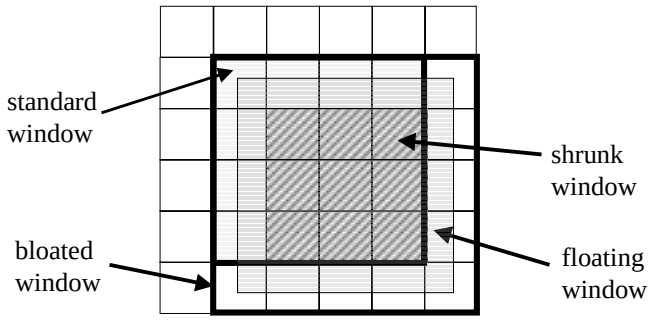


Drawbacks of Fixed *r*-Dissection Analysis

- If all $w \times w$ windows of fixed *r*-dissection have density $\leq U$, there may be ***floating*** $w \times w$ window with density $\min\{1, U + 1/r - 1/(4r^2)\}$
- Fixed-dissection algorithm is ***inaccurate***
- Exact algorithm is ***slow*** = $O(k^2)$

Shrunk and Bloated Windows

- ***Standard window*** = fixed *r*-dissection $w \times w$ window
- ***Floating window*** = arbitrary $w \times w$ window
- ***Bloated window*** = standard window bloated by one tile
- ***Shrunk window*** = standard window shrunk by one tile
- Any ***floating window*** is contained in ***one bloated window*** and contains ***one shrunk window***



Multilevel Approach

- ***Estimation***:
  - max floating window density ≤ max bloated window density
  - min floating window density ≥ min shrunk window density
- ***Zooming***:
  - remove standard windows in underfilled bloated windows
  - subdivide remaining tiles and find area of new bloated windows
- Terminate subdivision when either:
  - # of rectangles is small (run exact density analysis), or
  - (max bloated density)/(max standard density) ≤ ε ***(say, ε=1%)***

Multilevel Algorithm

Tiles = list of all windows (r =1)
Accuracy = ∞
While ***Accuracy*** > 1+ ε
    find are in each bloated and standard window
        MAX = max area of standard window
        BMAX = max area of bloated window
    refine ***Tiles*** = list of tiles from bloated windows of area ≥ MAX
    subdivide each tile in ***Tiles*** into 4 subtiles
    Accuracy = BMAX / MAX
Output max standard window density = MAX/ w^2

Filling Problem

- **Given** design rule-correct layout
  of k disjoint rectilinear features in $n \times n$ region
- **Find** design rule-correct **filled** layout
  - no fill geometry is added within distance B of any layout feature
  - no fill is added into any window that has density $\geq \boldsymbol{U}$
  - minimum window density in the filled layout is maximized (or has density $\geq$ lower bound $\boldsymbol{L}$)

Filling Problem in Fixed-Dissection Regime

- **Given**
  - fixed $\boldsymbol{r}$-dissection of layout
  - feature ***area***[T] in each tile T
  - ***slack***[T] = area available for filling in T
  - maximum window density $\boldsymbol{U}$
- **Find** total fill area p[T] to add in each T s.t.
  any $\boldsymbol{w} \times \boldsymbol{w}$ window W has density $\leq \boldsymbol{U}$ and
  $\min_W \sum_{T \in W}$ (area[T] + p[T]) is maximized

Fixed-Dissection LP Formulation

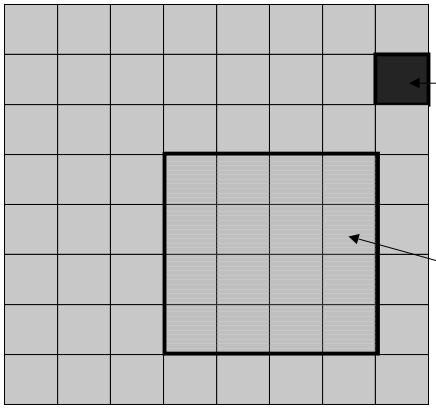
- Maximize M (= lower bound on window density)
- Subject to:
  - For any tile T: $0 \le p[T] \le \textbf{\textit{pattern}} \times slack[T]$
  - For any window W:

$$\sum_{T \in W} p[T] + area[T] \le U \times w^2$$

$$M \le \sum_{T \in W} (p[T] + area[T])$$

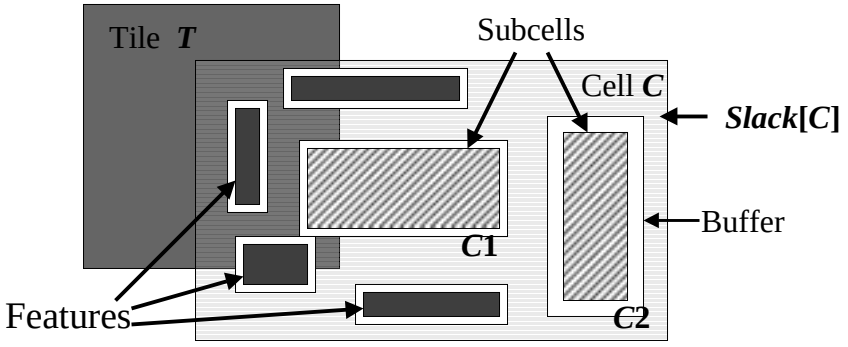
(***pattern*** = max achievable pattern area density)

Fixed-Dissection LP Formulation



Hierarchical Density Control

- Hierarchical filling = master cell filling



Hierarchical LP Formulation

- For any cell instance *C* of master cell **C** and tile T, $\gamma[C,T]$ is portion of slack[**C**] in intersection of *C* with T:

  $$\gamma[C,T] = \text{slack}(C \cap T)/\text{slack}[\mathbf{C}]$$

- New variable d[**C**] per each master cell **C:**

  d[**C**] = ***filling per master cell*** **C**

- New constraints:
  - for total amount of filling added into tile T:

    $$p[T] = \sum_{C \cap T} d[\mathbf{C}] \cdot \gamma[C,T]$$

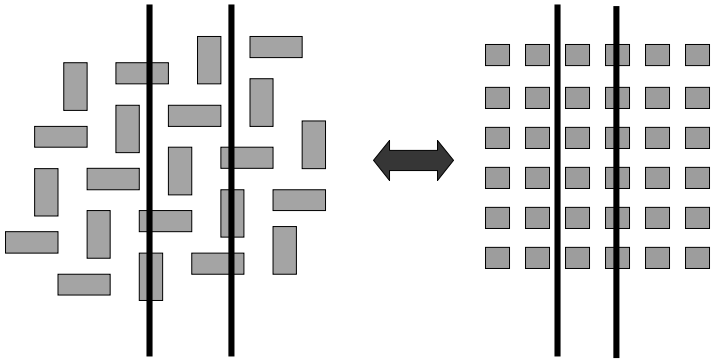
  - for amount of filling added into each master cell C:

    $$0 \le d[\mathbf{C}] \le \textbf{\textit{pattern}} \times \text{slack}[\mathbf{C}]$$

Synthesis of Filling Patterns

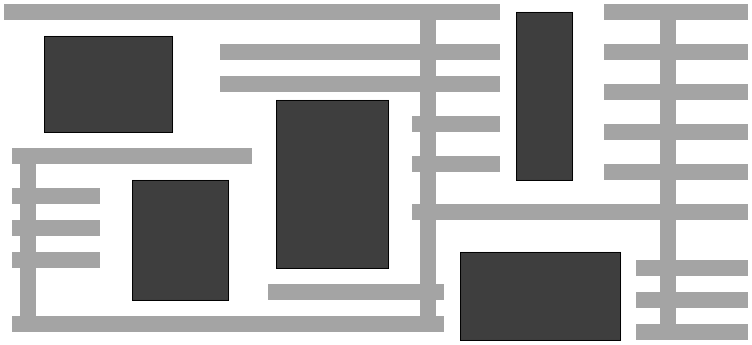
- Given area of filling pattern p[i,j], insert filling pattern into tile T[i,j] ***uniformly*** over available area
- Desirable properties of filling pattern
  - uniform coupling to long conductors
  - either grounded or floating

Basket-Weave Pattern



Each vertical/horizontal crossover line has same overlap capacitance to fill

Grounded Pattern



Fill with horizontal stripes,
then span with vertical lines

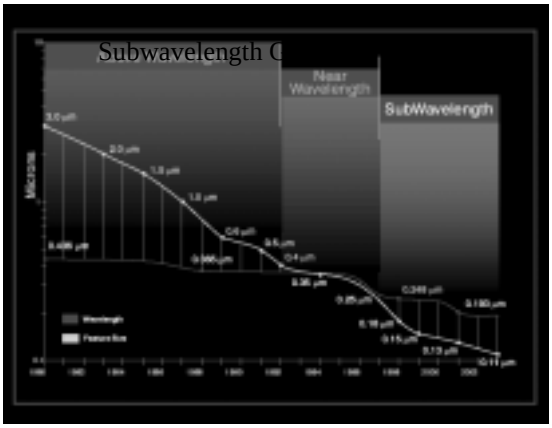
OPC and PSM

Subwavelength Optical Lithography — Technology Limits

- Implications of Moore's Law for feature sizes
- Steppers not available; WYSIWYG (layout = mask = wafer) fails after .35μm generation
- Optical lithography
  - circuit patterns optically projected onto wafer
  - feature size limited by diffraction effects
  - Rayleigh limits
    - resolution *R* proportional to λ / *NA*
    - depth of focus *DOF* proportional to λ / NA^2
- Available knobs
  - amplitude (aperture): OPC
  - phase: PSM

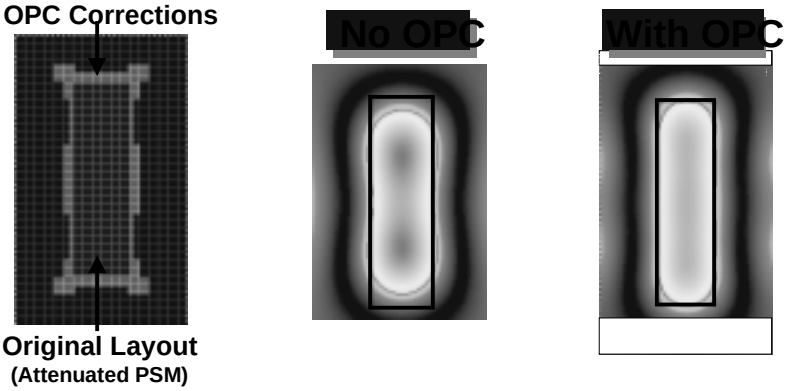
Next-Generation Lithography and the Subwavelength Gap

- EUV
- X-rays
- E-beams
- All at least 10 years away; require significant R&D, major infrastructure changes
- > 30 years of infrastructure and experience supporting optical lithography



Optical Proximity Correction (OPC)

- Corrective modifications to improve process control
  - improve yield (process latitude)
  - improve device performance



Optical Proximity Correction (OPC)

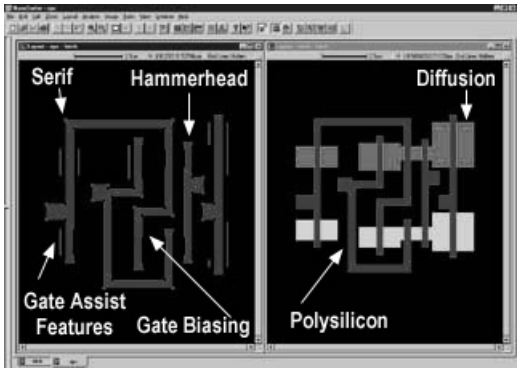
- Mostly cosmetic corrections; complicates mask manufacturing and dramatically increases cost (with little benefit?)
- Post-design verification is essential

Rule-based OPC
- apply corrections based on a set of predetermined rules
- fast design time, lower mask complexity
- suitable for less aggressive designs

Model-based OPC
- use process simulation to determine corrections on-line
- longer design time, increased mask complexity
- suitable for aggressive designs

OPC Features

- Serifs - for corner rounding
- Hammerheads - for line-end shortening
- Gate assists (subresolution scattering bars) - for CD control
- Gate biasing - for CD control
- Issues for custom, hierarchical and reuse-based layout methodologies

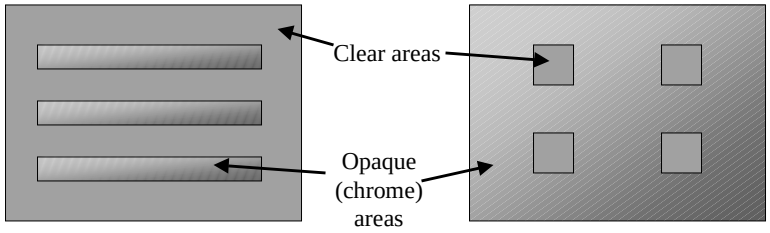


OPC Issues

- WYSIWYG broken $\rightarrow$ (mask) verification bottleneck
- Pass functional intent down to OPC insertion
  - make corrections that win $$$, reduce performance variation
  - OPC insertion is for predictable circuit performance, function
- Pass limits of manufacturing up to layout
  - don't make corrections that can't be manufactured or verified
  - Mask Error Enhancement Factor, etc.
- Layout needs models of OPC insertion process
  - geometry effects on cost of required OPC to yield function
  - costs of breaking hierarchy (beyond known verification, characterization costs)

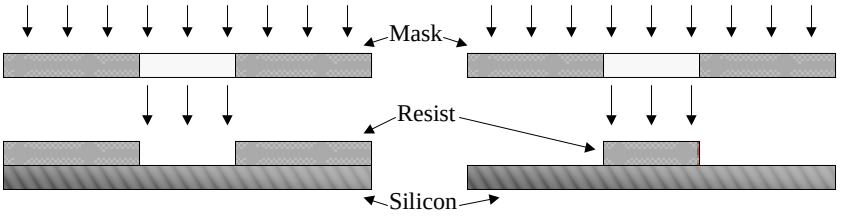
Mask Types

- Bright field masks
  - opaque features defined by chrome
  - background is transparent
  - used, e.g., for poly and metal
- Dark field masks
  - transparent features defined
  - background is opaque (chrome)
  - used, e.g., for contacts
  - used also for damascene metals



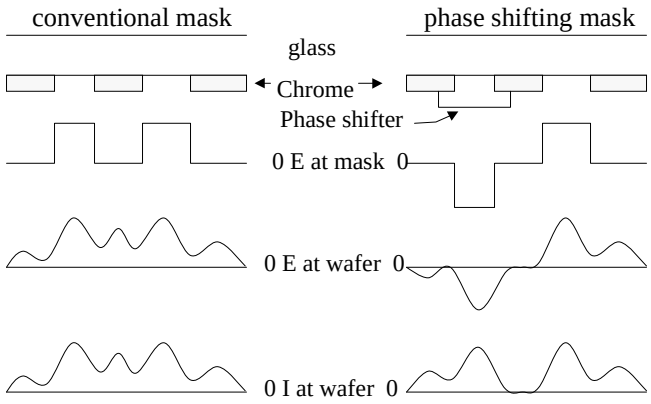
Photoresist Types

- Positive resists
  - material is removed from *exposed* areas during development
  - most widely used
- Negative resists
  - material is removed from *unexposed* areas during development
  - less mature



Post development profile for positive and negative photoresists

Phase Shifting Masks

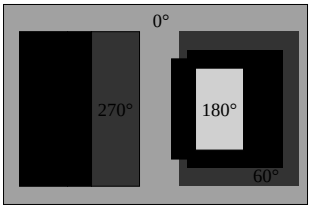


Phase Shifting Masks

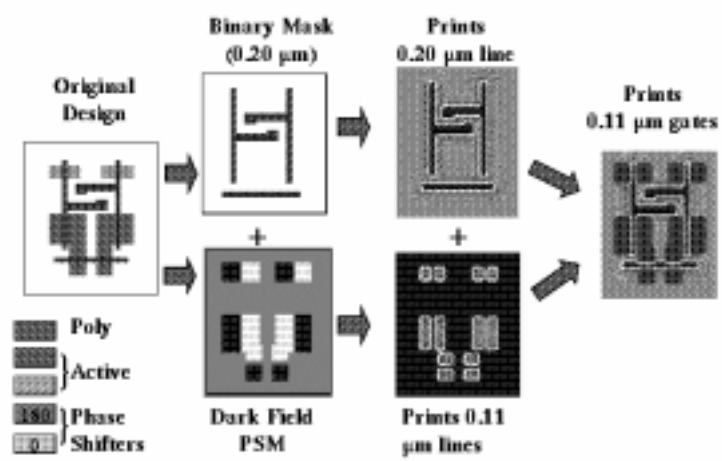
- no phase shifting: poor contrast due to diffraction
- phase shifting by 180°: reverse electric field on mask, destructive interference yields zero-intensity on wafer (high contrast)
- Background
  - invented in 1982 by Levenson at IBM
  - interest in early 1990s, but near wavelength → no pressing need
- Many forms of phase-shifting proposed
- Key issues: manufacturability, design tools
- Today: subwavelength gap forces PSM into every process (example: Motorola 90nm gates using 248nm stepper, announced in early 1999)

Forms of PSM

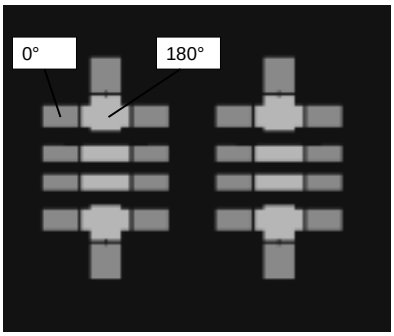
- Bright Field Phase-Shifting
  - single exposure
    - phase transitions required, e.g., 0-60-120-180 or 90-0-270 to avoid printing phase edges
    - throughput unaffected
    - limited improvement in process latitude
    - mask manufacturing difficult, mask cost very high
  - double exposure
    - PSM with 0 and 180 degree phase shifters
    - define only critical features ("locally bright-field"), rest of mask is chrome
    - second exposure with clear-field binary mask protects critical features, defines non-critical features as well
    - excellent process latitude
    - decrease in throughput (double exposure)



Gate Shrinking and CD Control Using Phase Shifting



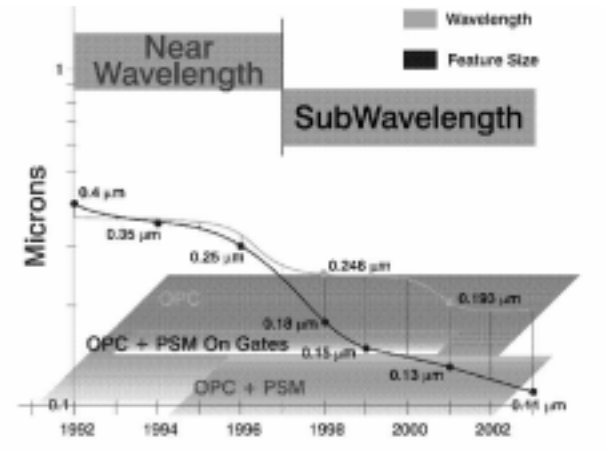
Double-Exposure Alternating PSM



1. Alternate PSM Mask

2. Trim Mask (COG)

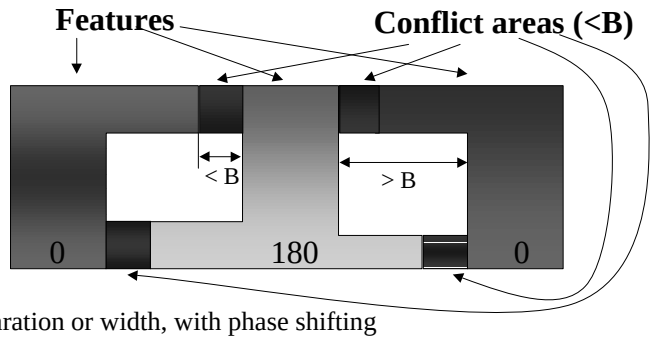
Applicability of OPC and PSM



The Phase Assignment Problem in PSM

Assign 0, 180 phase regions such that

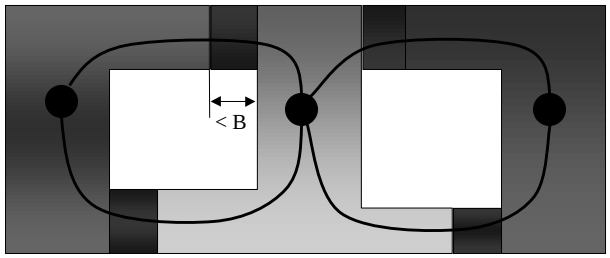
- (dark field) feature pairs with separation < B have opposite phases
- (bright field) features with width < B are induced by adjacent phase regions with opposite phases



b ≡ minimum separation or width, with phase shifting
B ≡ minimum separation or width, without phase shifting

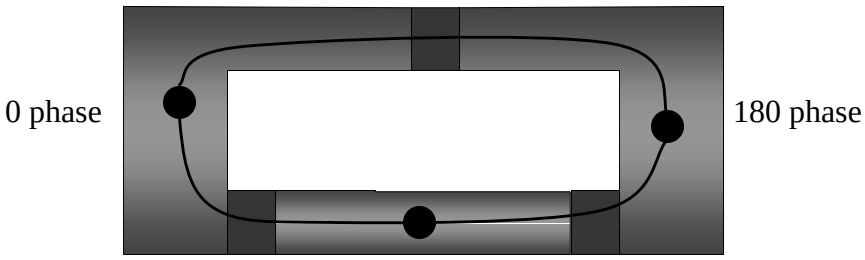
Phase Conflict and the Conflict Graph

- Vertices: features (or phase regions)
- Edges: “conflicts” (necessary phase contrasts)
  (feature pairs with separation < B)



Odd Cycles in Conflict Graph

- Self-consistent phase assignment is not possible if there is an odd cycle in the conflict graph
- Phase-assignable ≡ bipartite ≡ no odd cycles



Phase Conflict and the Conflict Graph

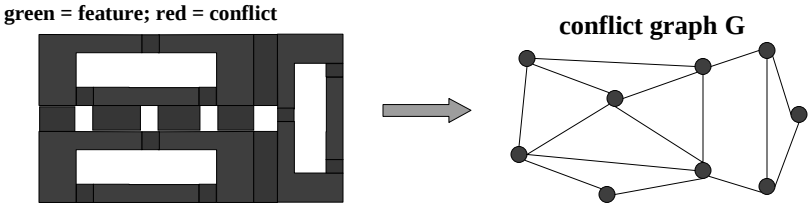
- Self-consistent phase assignment is not possible if there is an odd cycle in the conflict graph
- Phase-assignable = bipartite = no odd cycles
  - this is a global issue!
  - features on one side of chip can affect features on the other side
- Breaking odd cycles: must change the layout!
  - change feature dimensions, and/or change spacings
  - degrees of freedom include layer reassignment for interconnects

Conflict Graph

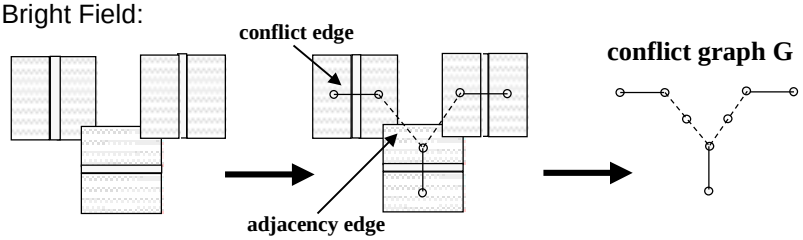
- Dark Field: build graph over feature regions
  - edge between two features whose separation is < B
- Bright Field: build graph over shifter regions
  - two edge types
  - adjacency edge between overlapping phase regions : endpoints must have same phase
    - essentially, these regions must be "merged" into single phase shifter
    - DRC-like (gap, notch type) local rules must likely be applied to such "merging"
  - conflict edge between shifters on opposite side of critical feature: endpoints must have opposite phase
  - Step 3: simple reduction to previous (dark-field) T-join solution: each dotted edge becomes a 2-chain (introduce one extra vertex)

Conflict Graph

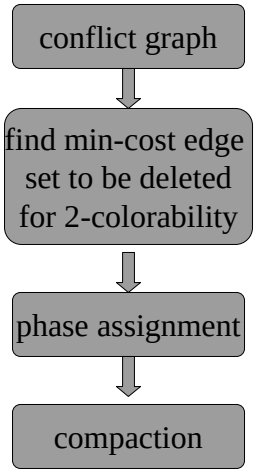
- Dark Field:



- Bright Field:

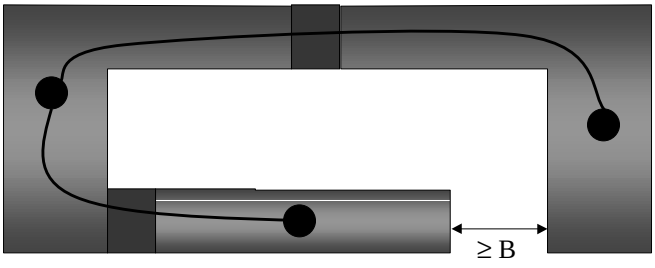


One-Shot Phase Assignment



Breaking Odd Cycles

- Must change the layout:
  - change feature dimensions, and/or
  - change spacings
  - PSM phase-assignability is a layout, not verification, issue

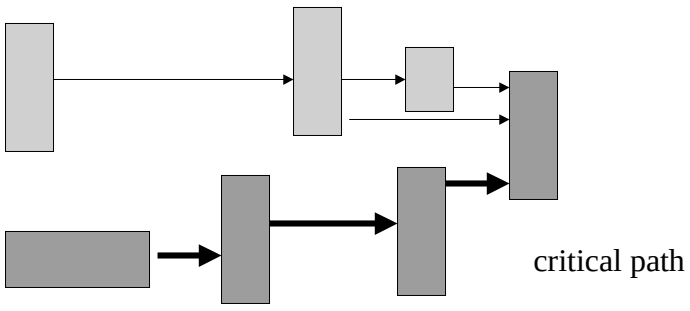


Phase Assignment: Key Technologies

- Key technologies: incremental gridless routing, incremental compaction
- Issues for custom, hierarchical and reuse-based layout methodologies

Conflict Edge Weight

- Which conflict edges are cheapest to break?
- Critical paths (e.g., in compactor) in x- and y-directions define layout area
- Conflict edges not on critical path: break for free

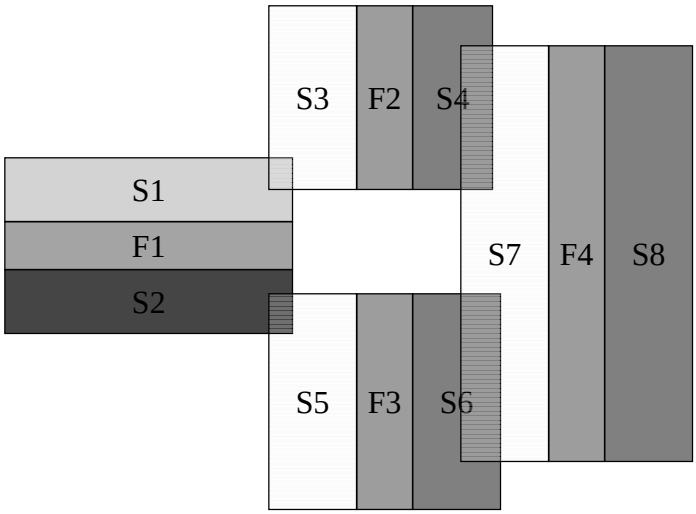


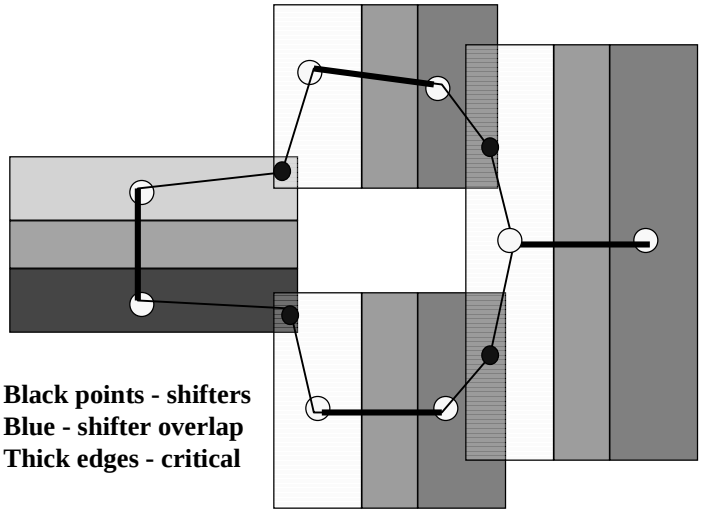
Minimum-Perturbation PSM Layout

- Input: layout in, e.g., .25um design rules
- Goal: adjust feature sizes and spacings to new PSM design rules, e.g., .15um
  - keep topology as close to original as possible
- Application to new design and to migration
  - assumes existence of a starting layout
  - hope to attain fewer violations in verification, require less manual cleanup of output layout

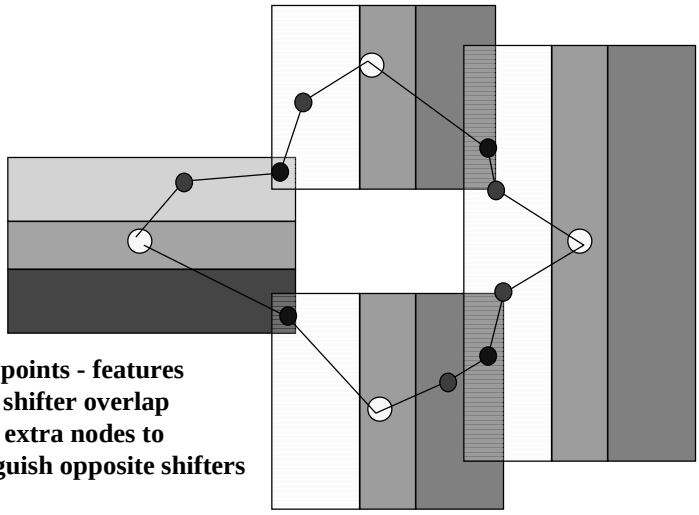
Compaction-Oriented Approach

- Analyze input layout
- Determine constraints for output layout
  - new PSM-induced (shape, spacing) constraints
- Compact (e.g., solve LP) with min perturbation objective
  - e.g., minimize sum of differences between old and new positions of each edge
- Key: Minimize the set of new constraints, i.e., break all odd cycles in conflict graph by deleting a minimum number of edges.



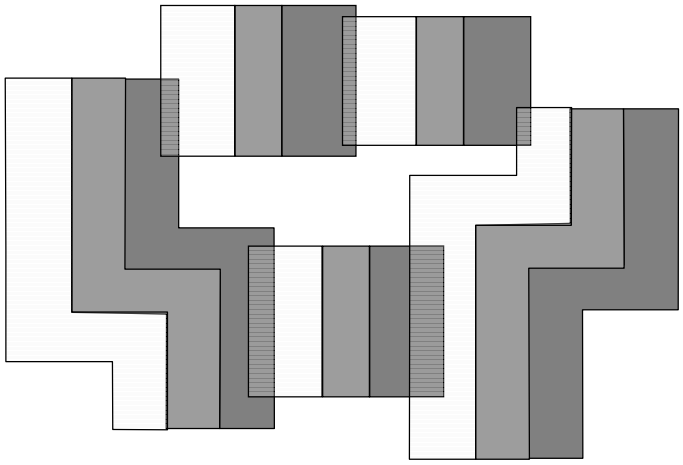


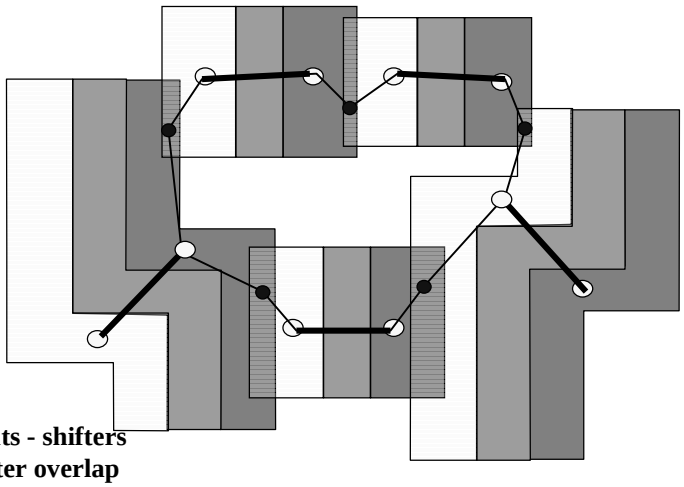
Bipartization Problem: delete min # of thin edges to make graph bipartite



Black points - features
Blue - shifter overlap
Pink - extra nodes to distinguish opposite shifters

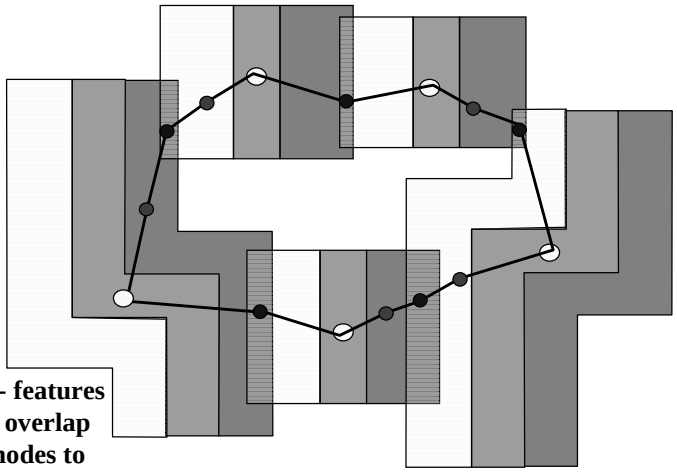
Bipartization Problem: delete min # of nodes (or edges) to make graph bipartite





Black points - shifters
Blue - shifter overlap
Thick edges - critical

Bipartization Problem: delete min # of thin edges to make graph bipartite



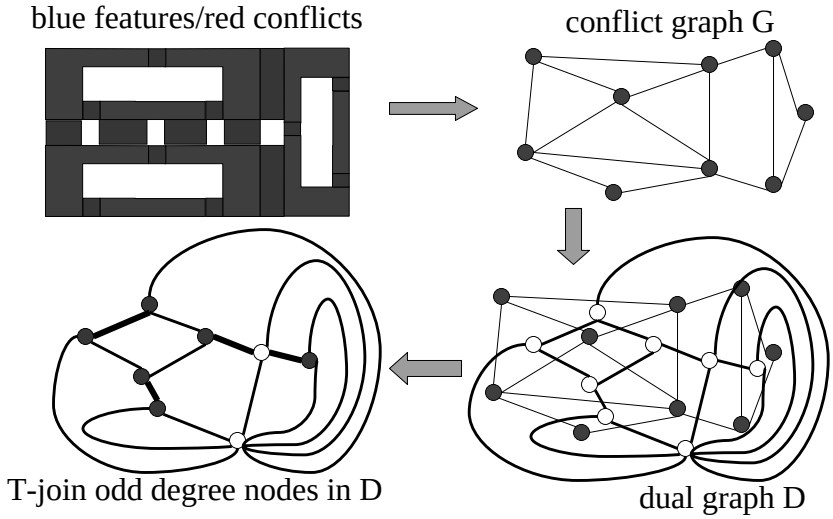
Black points - features
Blue - shifter overlap
Pink - extra nodes to distinguish opposite shifters

Bipartization Problem: delete min # of nodes (or edges) to make graph bipartite

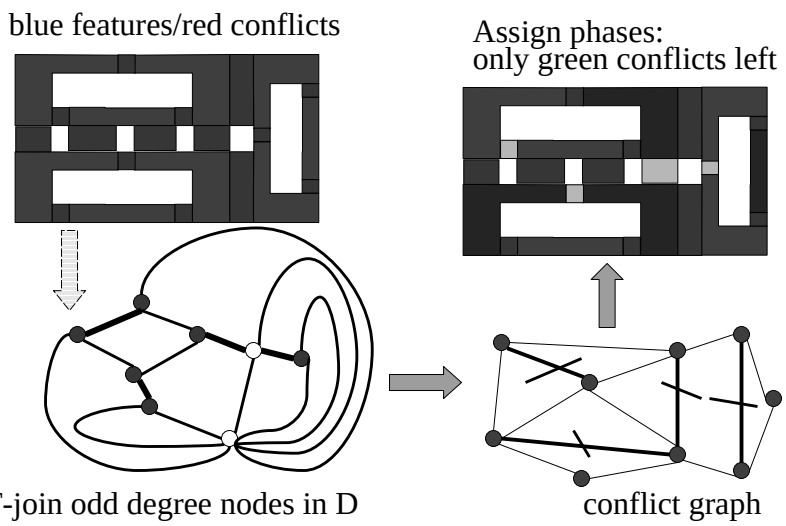
The T-join Problem

- How to delete **minimum-cost** set of edges from conflict graph G to eliminate odd cycles?
- Construct geometric dual graph D=dual(G)
- Find odd-degree vertices T in D
- Solve the **T-join problem** in D**:**
  - find min-weight edge set J in D such that
    - all T-vertices has odd degree
    - all other vertices have even degree
- Solution J corresponds to desired min-cost edge set in conflict graph G

Optimal Odd Cycle Elimination



Optimal Odd Cycle Elimination



T-join Problem in Sparse Graphs

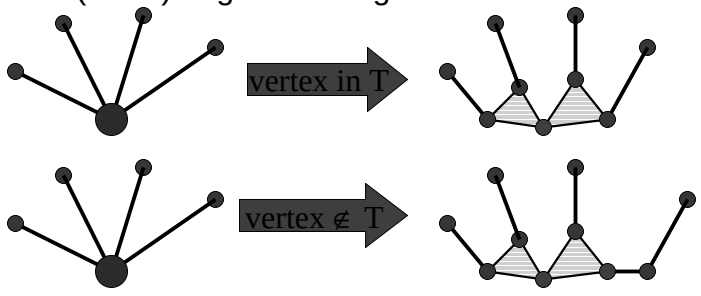
- Reduction to matching
  - construct a complete graph T(G)
    - vertices = T-vertices
    - edge costs = shortest-path cost
  - find minimum-cost perfect matching
- Typical example = sparse (not always planar) graph
  - note that conflict graphs are sparse
  - #vertices = 1,000,000
  - #edges $\approx 5 \times$ #vertices
  - # T-vertices $\approx$ 10% of #vertices = 100,000
- Drawback: finding all shortest paths too slow and memory-consuming
  - #vertices = 100,000 $\rightarrow$ #edges = 5,000,000,000

T-join Problem: Reduction to Matching

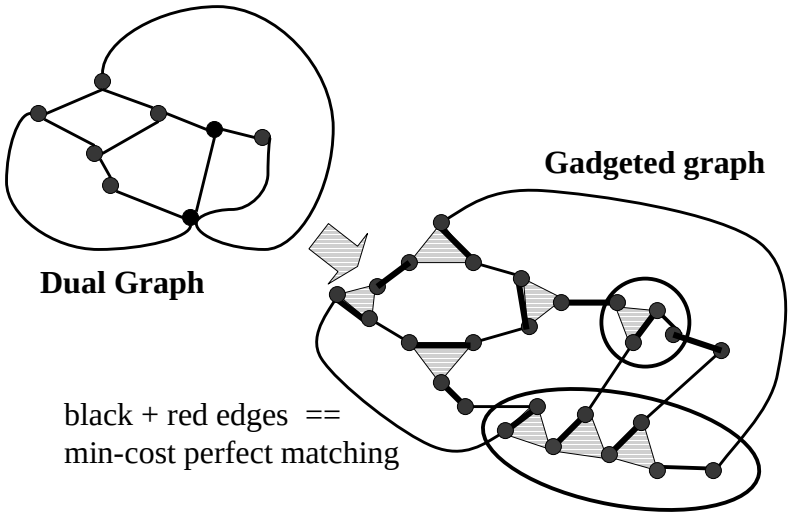
- **Desirable properties of reduction to matching:**
  - exact (i.e., optimal)
  - not much memory (say 2-3Xmore)
  - results in a very fast solution
- **Solution: gadgets!**
  - replace each edge/vertex with gadgets s.t.
    matching all vertices in gadgeted graph
    $\Leftrightarrow$ T-join in original graph

T-join Problem: Reduction to Matching

- replace each vertex with a chain of triangles
- one more edge for T-vertices
- in graph D: m = #edges, n = #vertices, t = #T
- in gadgeted graph: 4m-2n-t vertices, 7m-5n-t edges
- cost of red edges = original dual edge costs
  cost of (black) edges in triangles = 0



Example of Gadgeted Graph



Results

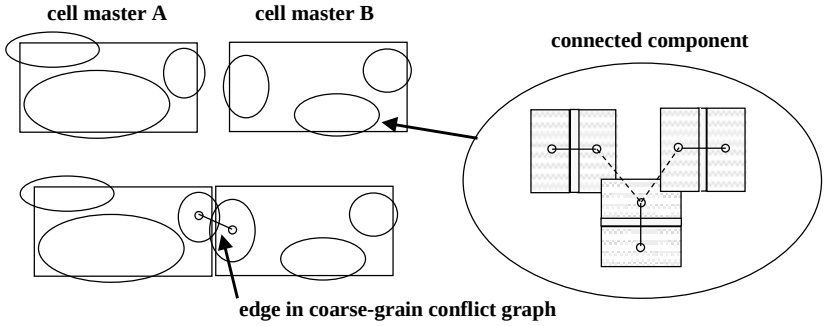
	Layout1		Layout2		Layout3	
Testcase	polygons	edges	polygons	edges	polygons	edges
	3769	12442	9775	26520	18249	51402
Algorithm	edges	runtime	edges	runtime	edges	runtime
Greedy	2650	0.56	2722	3.66	6180	5.38
GW	1612	3.33	1488	5.77	3280	14.47
Exact	1468	19.88	1346	16.67	2958	74.33

- Runtimes in CPU seconds on Sun Ultra-10
- Greedy = breadth-first-search bicoloring (similar to Ooi et al.)
- GW = Goemans/Williamson95 heuristic
- Cook/Rohe98 for perfect matching

Latest improved gadgets: runtimes decrease by factor of 6

Conflict Graph for Cell-Based Layouts

- Coarse view: at level of connected components of conflict graphs within each cell master
  - each of these components is independently phase-assignable
  - can be treated as a single "vertex" in coarse-grain conflict graph



Auto-P&R Flow Issues

Constraints

- PSM must be "transparent" to ASIC auto-P&R
  - "free composability" is the cornerstone of the cell-based methodology!
  - focus on poly layer → we are concerned with placer, not router
- Competitive context for placer
  - extremely competitive runtime regimes (e.g., 10^6 cells detail-placed in 20 min); faster runtimes needed in RTL-planning methodologies (Nano/PKS, Tera)
  - any nontrivial cost of checking placement phase-assignability is unacceptable
- Iteration between placer and a separate tool is unacceptable
  - interface to auto-P&R tools is bulky (e.g., 100s of MB for DEF), slow
  - no known convergent method for post-P&R phase-assignability checks to drive P&R to guaranteed correct solution (very difficult!)
- P&R tool MUST deliver guaranteed phase-assignable poly layer

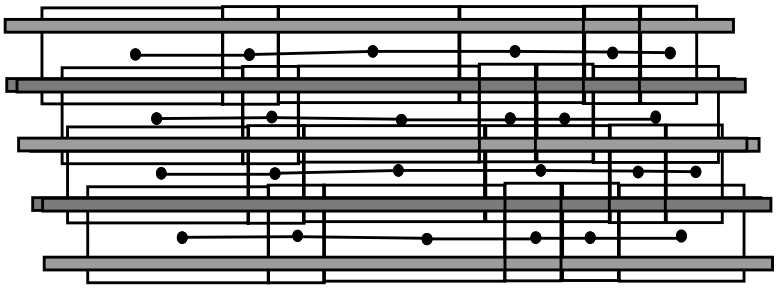
Guidelines

- Placer
  - no re-entry into placer from an external tool
    - any needed extra functionality must be built directly into QP
  - placer must guarantee a phase-assignable poly when finished
  - polygon layout information currently not in placement vocabulary
    - available relevant abstractions: pin EEQs/LEQs, overlap layer geometries
    - side files or LEF extensions needed for, e.g., capturing versioning or phase shifters near left/right cell boundaries
- Cell layout
  - cell layouts and phase shifters are assumed fixed during library creation
    - on-the-fly cell layout synthesis or layout perturbations generally not allowed
  - $2^{k'}$ possible versions (i.e., distinct phase bindings) are available for a given master cell with k connected components in its phase conflict graph, $k' < k$ of which contain critical poly at cell boundary
    - impractical to use EEQs to capture versioning within iterative improvement

Types of Composability

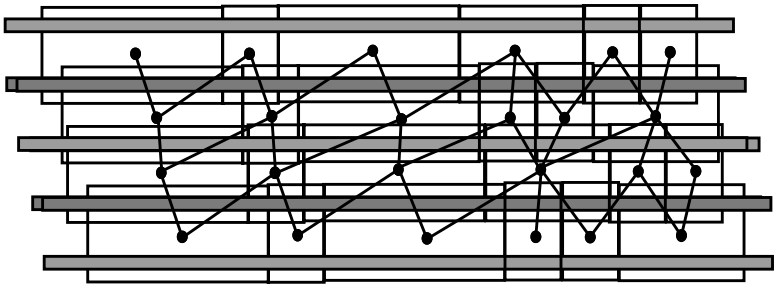
- Same-row composability
  - any cell can be placed immediately adjacent (in the same row) to any other cell
- Adj-row composability
  - any cell can be placed in an adjacent cell row to any other cell, with the two cells having intersecting x-spans
- Four cases of cell libraries (G = guaranteed; NG = not guaranteed)
  - Case 1: adj-G, same-G
    - most-constrained cell layout; most transparent to placer
  - Case 2: adj-G, same-NG
  - Case 3: adj-NG, same-G
  - Case 4: adj-NG, same-NG
    - least-constrained cell layout; least transparent to placer

Case 2: Adj-G, Same-NG



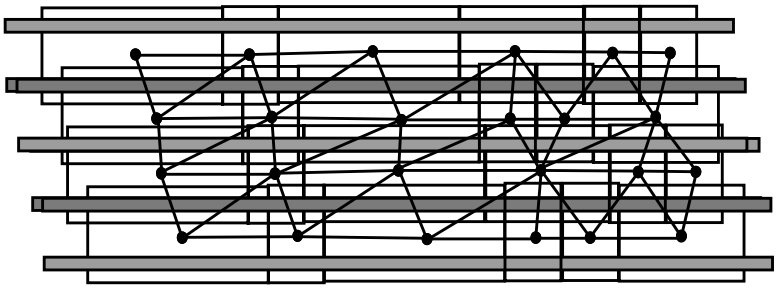
Blue vertices, edges = graph of phase assignment "dependencies"

Case 3: Adj-NG, Same-G



Blue vertices, edges = graph of phase assignment "dependencies"

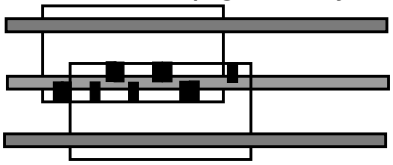
Case 4: Adj-NG, Same-NG



Blue vertices, edges = graph of phase assignment "dependencies"

Overlap Layer Abstraction in LEF

- Like "teeth of a broken comb" defined for each master cell
- Placer makes sure that the teeth don't collide when the cells are placed, i.e., the two "broken combs" interlace
- Available today in LEF standard; placer understands overlap layer
  - current heuristics may not scale well if many instances have overlap geometry



Traditional picture of overlap geometries

Case 1: Adj-G, Same-G

- Solution 1: "no restrictions on the cell layout"
  - create cell abstractions such that placer runs in "normal" mode
    - e.g., pre-bloat (by 1 site) cells that have critical poly near left/right boundary
    - e.g., create overlap layer obstacles corresponding to critical poly near top/bottom boundary
- Solution 2: smart rules to restrict cell layout
  - e.g., every pair of boundary-CP features from the same cell must be non-interfering
    - **definition:** two features are non-interfering if they are in different connected components of the cell's phase conflict graph
  - no boundary-CP feature is "near" two different sides of its cell
  - these two restrictions $\rightarrow$ composability guaranteed (no odd cycles possible)
- Solution 3: dumb rules to restrict cell layout
  - all cells have 250nm-wide 0-phase boundary (IBM style)

Notation

- M = number of master cells in library
- C_i = i^{th} master cell, i = 1, …, M
- w_i = width of i^{th} master cell, i = 1, …, M
- V_i = number of versions of the i^{th} master cell, i = 1, …, M
- C_{ik} = k^{th} version of i^{th} master cell, i = 1, …, M and k = 1, …, V_i
- N = number of movable cells in the row of interest
- R_h = h^{th} cell in the row of interest
- S_h = master cell corresponding to the h^{th} cell in the row of interest
- boundary-CP = critical poly feature "near" the cell boundary

Cases 2,4: Same-NG

- Each (sub)row checked separately, post-placement
- Basic tool: <u>cell compatibility table</u>
  - library is precharacterized by M^2 two-dimensional arrays A_{ij}, one array for each possible pairing of cells with C_i to the left of C_j
  - A_{ij}<p,q> = minimum site separation at which C_{ip} can be placed adjacent to C_{jq} (p = 1, …, V_i and q = 1, …, V_j)
    - example: M = 500 with 16 versions of each master cell → < 30 MB storage
- <u>Goals:</u>
  - (1) if phase assignment possible, return set of versions for each of the cell instances
  - (2) if not possible, return set of versions plus set of inserted feedthroughs (extra sites) such that minimum perturbation is achieved

Cases 2,4: Same-NG Example Sol'n

- Shortest-path finding in a simple graph (actually, a DAG) :
  - for each version j of each cell R_i, create node $<R_i,j>$, i = 1, …, N and j = 1, …, V_{r_i}
  - create source node $<R_0,0>$ and termination node $<R_{N+1},0>$
  - create directed edges $(<R_i,j>,<R_{i+1},k>)$ for all versions j of cell R_i and versions k of cell R_i+1 (weight = cost of perturbing placement to achieve minimum allowed site separation)
  - create zero-weight directed edges $(<R_0,0>,<R_1,j>)$ for all versions j of cell R_1 and $(<R_N,j>, <R_{N+1},0>)$ for all versions j of cell R_N
- Minimum-perturbation solution (specifies compatible versions as well as required changes in cell positions) given by shortest path from $<R_0,0>$ to $<R_{N+1},0>$

Cases 3,4: Adj-NG Example Sol'ns

- Basic cause of problem: horizontal poly near shared rails
  - complex cells that push the cell height (#pitches), e.g., latch/FF, adder, mux
- Solution 1: partial amelioration by layout constraints
  - e.g., for horizontal critical poly near power rail, the outside shifter must be 0-phase (NTI style)
  - can be done silently by version compatibility, etc.
- Solution 2: abstract w/existing LEF overlap layer construct

Manufacturing Process Issues

- Clean abstractions of process
  - OPC: a “rectangle” has 26 edges in 3 polygons
  - PSM: cell layouts and legal routes not freely composable

Manufacturing Process Issues

- Mask inspection / validation bottleneck
  - if Layout $\neq$ Silicon but the circuit is still functional… is it really an error ?
  - must the Design $\leftrightarrow$ Mask boundary change ?

Manufacturing Process Issues

- Tool flow
  - line end extensions in the router
  - hammerheads in the library generator
  - OPC insertion in physical verification
  - RC extraction for timing signoff in P&R
  - post-P&R density control (filling and cheesing)

Manufacturing Process Issues

- “Life changes” for circuit and layout designers
  - only particular geometries and pitches ?
  - self-adapting circuits ?
  - huge verification guardbands ?
  - **reuse-based design at risk -- at all levels ?**
  - **again, relative cost of custom decreases !**

Implications of Technology

- Hard IP reuse is difficult
  - divergent foundry processes
  - design- and context-specific variants of cells, macros, cores
    - filling densities
    - thermal, noise sensitivity contexts
    - layer usage and local region porosity constraints, physical access
    - incompatibility of separate phase solutions, or phase solutions + local routing
    - tool-specific variants (e.g., for different auto-routers)
    - diffusion sharing, continuous device sizing, tuning (dual Vt, multiple supply voltages (thermal, IR drop contexts), different input arrival times/slews,…)
- Hard-reuse: An ideal that must be tempered (abandoned?)
- Custom-on-the-fly is natural consequence of tuning, perf opts, migration, soft- and firm-IP reuse