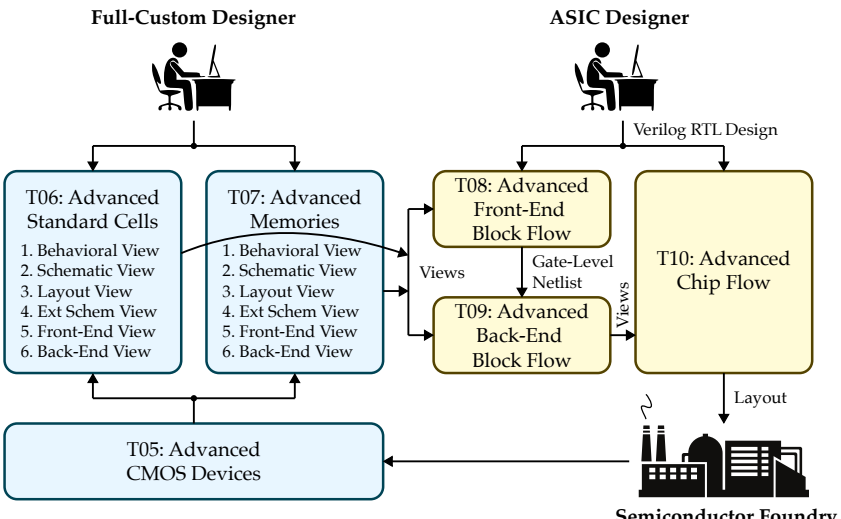
ECE 6745 Complex Digital ASIC Design

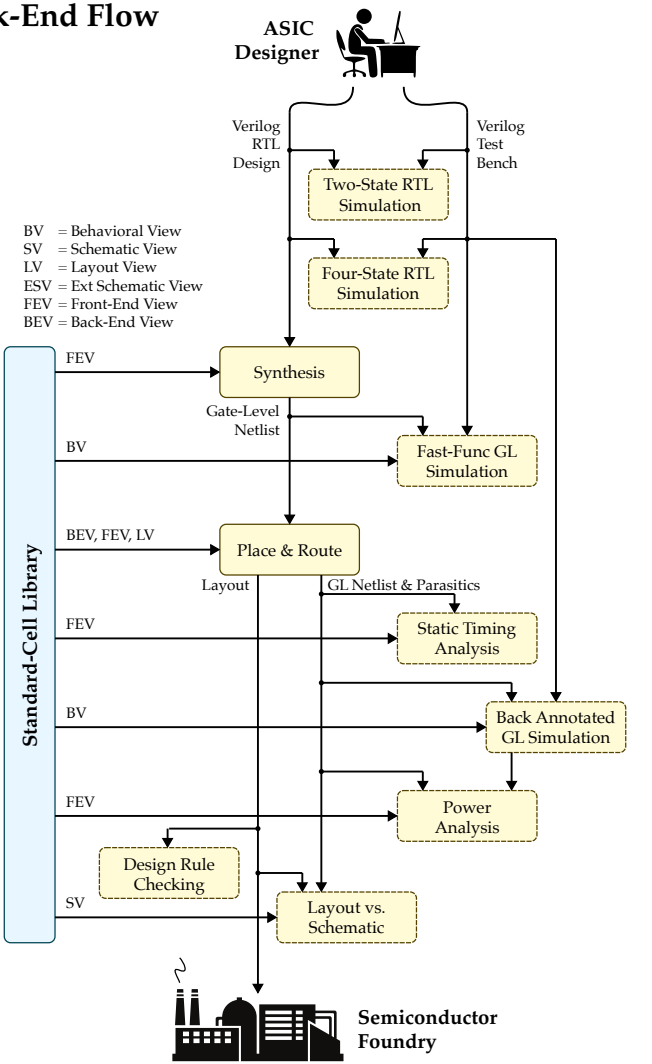
Topic 9: Advanced Back-End Flow

School of Electrical and Computer Engineering
Cornell University

revision: 2026-03-26-12-57

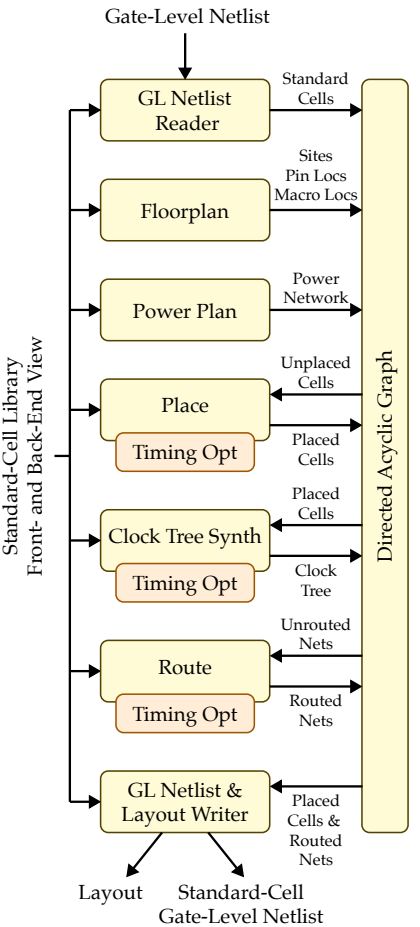


1. Back-End Flow



Place-and-Route

- Place-and-route takes as input a standard-cell gate-level netlist, places cells, routes nets, and produces the final layout
- Place-and-Route Data Structure
  - Graph of standard-cell nodes
- Place-and-Route Algorithms
  - GL Netlist Reader: Parse gate-level netlist into graph of standard-cell nodes
  - Floorplan: Creates grid of sites and position macros and pins
  - Power plan: Create power distribution network
  - Place: Places each standard cell on the grid of sites
  - Clock Tree Synthesis: Create clock tree buffers and routing
  - Route: Routes each net on the routing grid
  - Timing Optimization: Size gates and insert buffers to meet timing constraints
  - Layout & GL Netlist Writer: Outputs the final layout along with an updated standard-cell gate-level netlist



1.1. *Algorithm:* Floorplan

- Goal is to create a grid of sites, position input/output pins, and place macros
  - Each *site* is a one standard cell height tall and one routing track wide
  - Standard cells must be placed such that they align to sites
  - Must also place macros (e.g., SRAM macros)

Fixed Floorplan

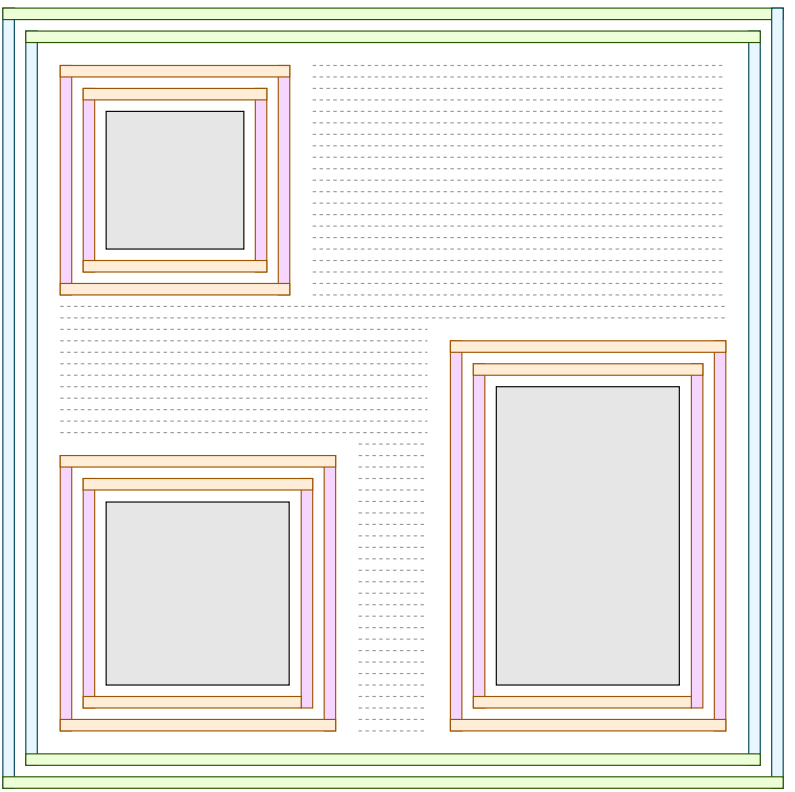
- Width, height, input/output pins, and macro placements are given

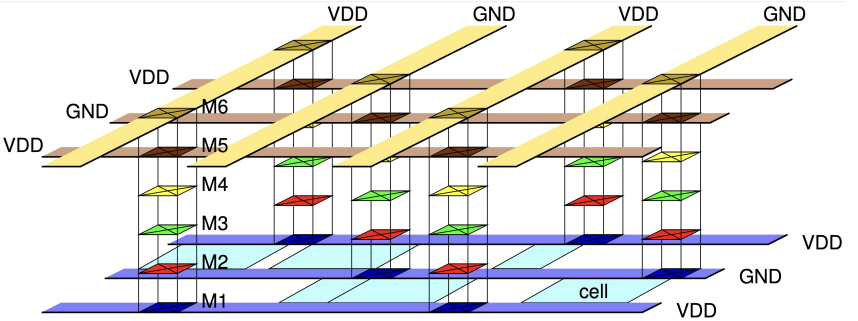
Automatic Floorplan

- Standard cell density and aspect ratio are given
- Calculate total area from gate-level netlist + macro area
- Divide by cell density to get total floorplan area
- Use aspect ratio to determine width and height
- Automatically position input and output pins along perimeter
- Automatically place macros minimizing wire length estimate

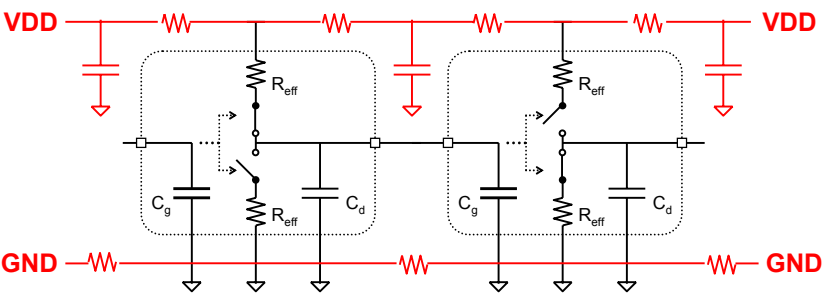
1.2. *Algorithm:* Power Plan

- Each macro has its own local power ring
- Create global power ring around the outside of the block
- Create metal 1 power straps across standard cell rows
- Create power grid on top metal layers

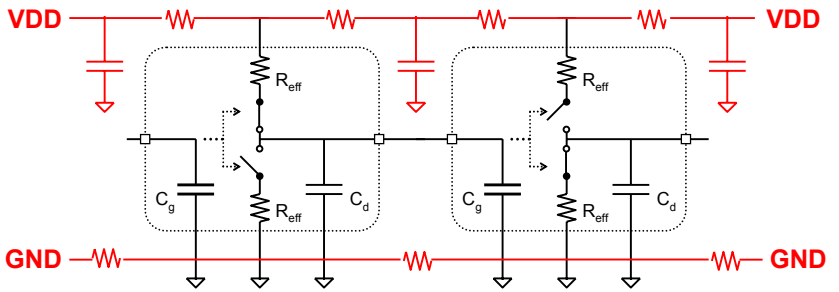




Static IR Drop



Dynamic IR Drop



Impact of Static and Dynamic IR Drop

- Lower effective voltage across logic gates
  - Propagation delays increase
  - Noise margins shrink

1.3. *Algorithm:* Place

- Global Placement: Use hierarchical and interative algorithms to roughly place standard cells on chip without worrying about exact legal palcement restrictions
- Detailed Placement: Ensure every standard cell is in a legal location and then perform local placement optimizations

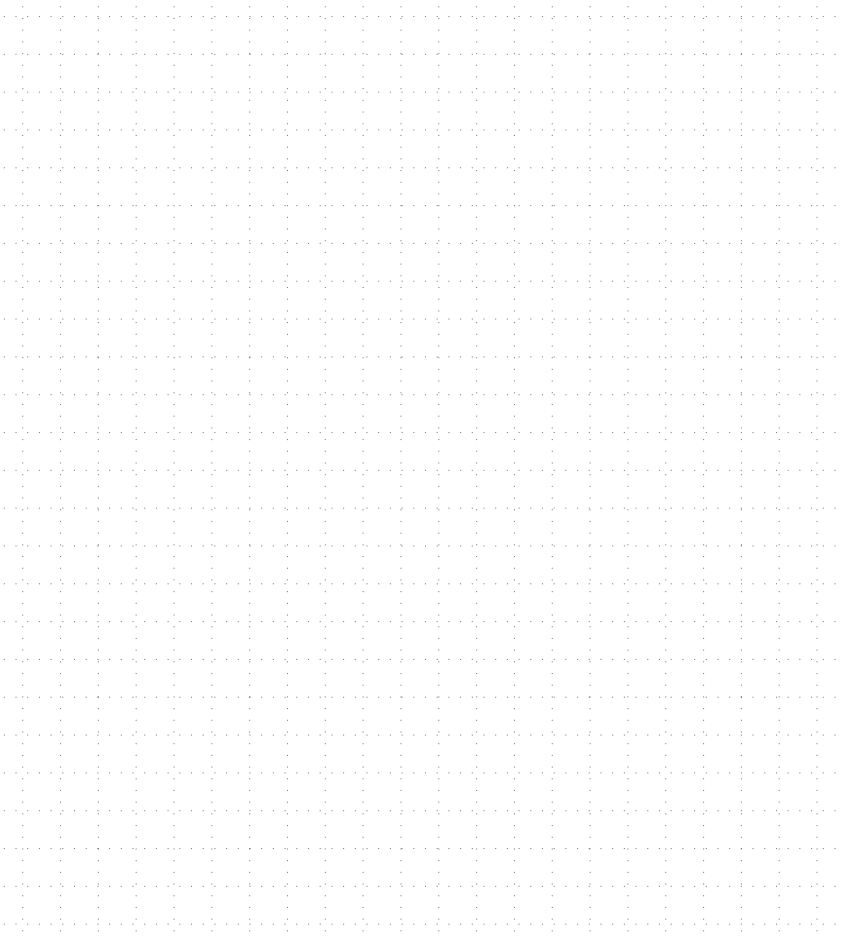
Global Placement

- Analytical approaches express a cost function and the constraints as analytical functions of the coordinates the standard cells and then use optimization techniques to reduce the overall cost
- Quadratic placement algorithm uses the sum of the squared Euclidean distances between standard cells and fixed pins

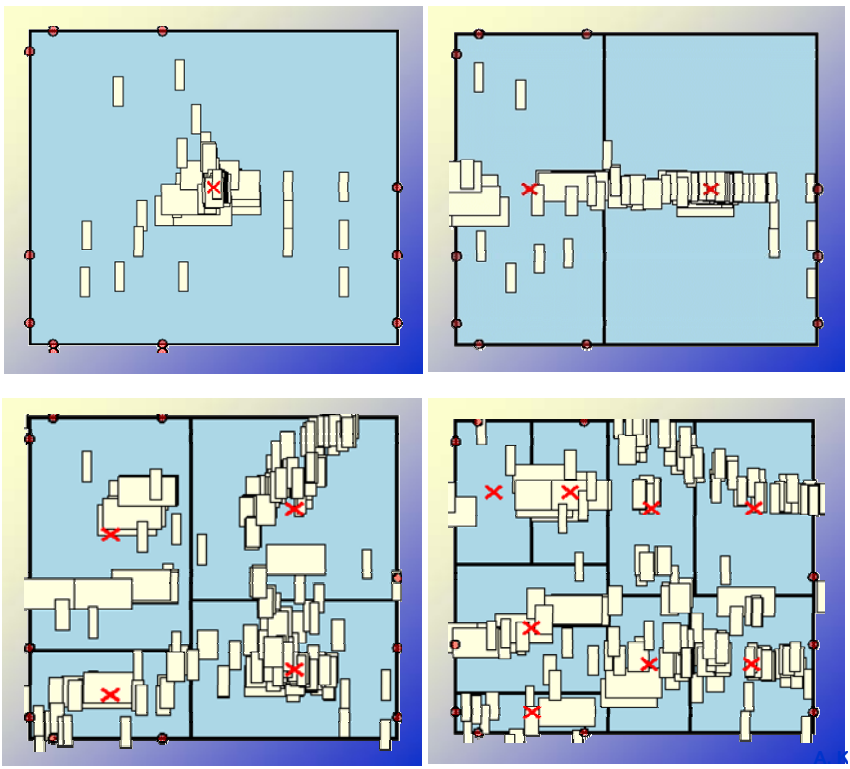
$$\text{Cost} = \sum_{(i,j)\in E} \left((x_i - x_j)^2 + (y_i - y_j)^2 \right)$$

- Consider netlist with two fixed pins and two INVX1 standard cells

- Consider netlist with three fixed pins, NAND2X1, and INVX1 standard cells



- Recursive analytical placement helps spread out cells
  - Initial analytical placement
  - Sort cells by x-coord, first half in left region, second half in right region
  - Add fixed anchor at center of left region for every cell in left region
  - Add fixed anchor at center of right region for every cell in right region
  - Redo analytical placement across all cells

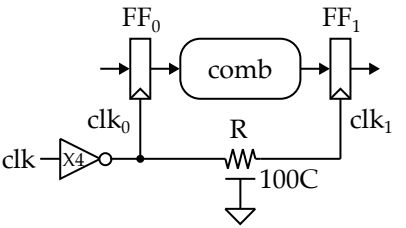


Detailed Placement

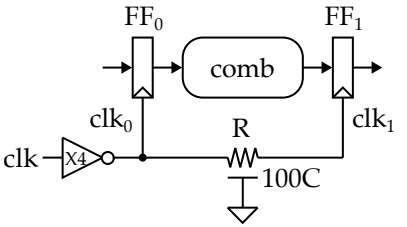
- Legalization
  - Convert global placement into valid standard-cell layout
  - Snap cells to sites
  - Remove overlaps
  - Keep cells close to original positions
- Local Optimization
  - Small cell swaps or reorder cells within a row
  - Fix wirelenth or minor congestion

1.4. *Algorithm:* Clock Tree Synthesis

- Analyze propagation delay of the clock signal in the following post-placement gate-level netlist



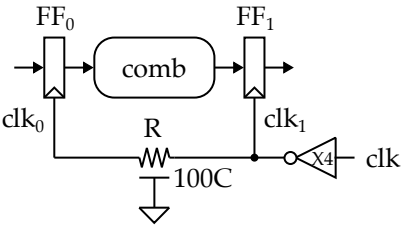
- Analyze the setup and hold timing constraints given the following delay model and flip-flop timing parameters



	t_{pd}	t_{cd}
Comb	10τ	3τ
FF (t_{cq})	9.6τ	9.6τ

FF (t_{setup})	10.6τ
FF (t_{hold})	-1.6τ
T_C	40τ

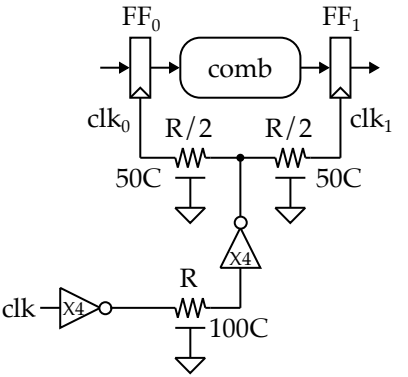
- Analyze the setup and hold timing constraints given the following delay model and flip-flop timing parameters



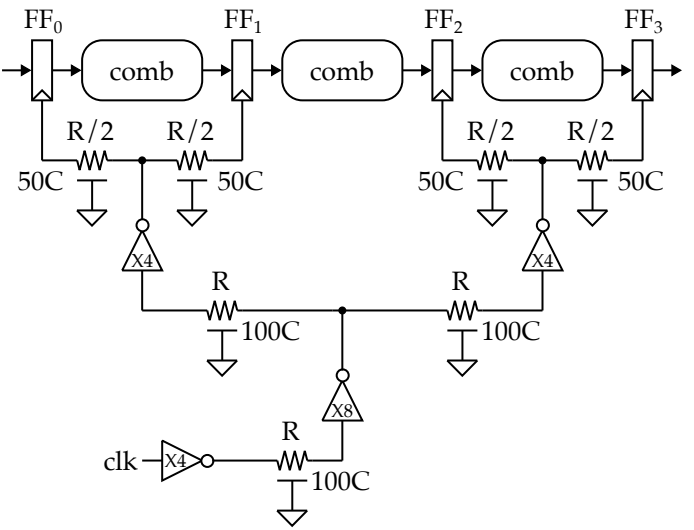
	t_{pd}	t_{cd}
Comb	10τ	3τ
FF (t_{cq})	9.6τ	9.6τ

FF (t_{setup})	10.6τ
FF (t_{hold})	-1.6τ
T_C	40τ

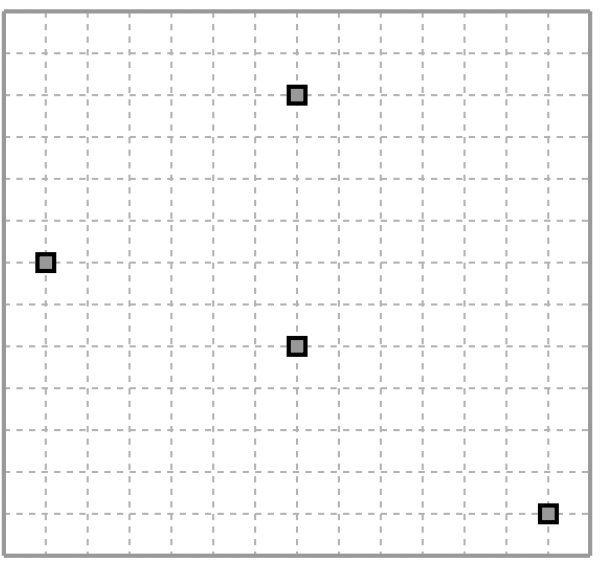
- Analyze clock skew in following post-placement gate-level netlist



- Clock tree synthesis will create a clock tree topology and carefully size the clock buffers to reduce rise/fall times while at the same time minimizing area



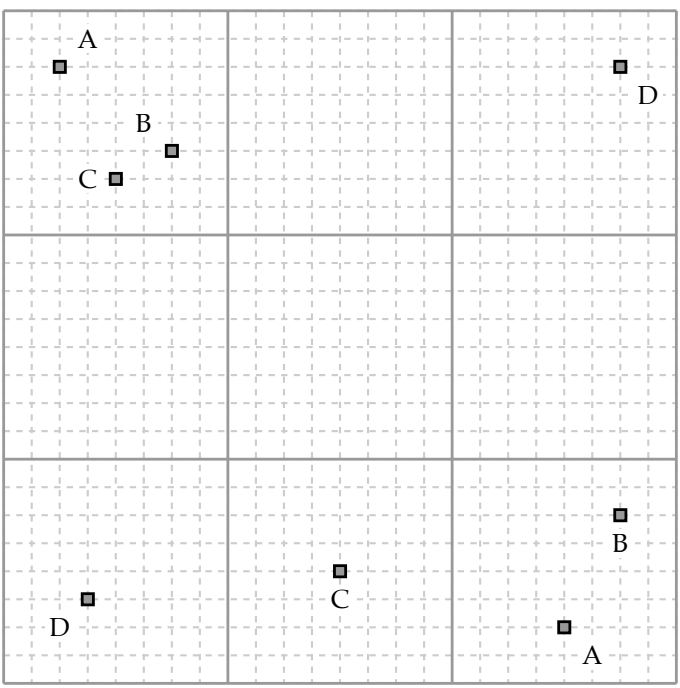
- Clock tree synthesis will also route the clock tree network to balance the delay at each level of the tree perhaps using the deferred-merge embedding (DME) zero-skew routing algorithm
- Bottom-up phase
  - For pairs of leaves find a *merging segment* (MS) which is a set of points equidistant to both children
  - For pairs of merging segments find *merging regions* which is a set of points equidistance to both merging segments
  - Continue until one region remains at the root
- Top-down phase
  - Start from root and pick point inside merging region
  - Route to children ensuring equal delay
  - Continue until reach leaves

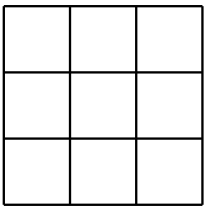
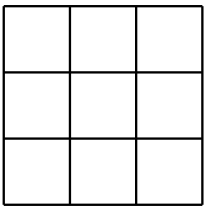
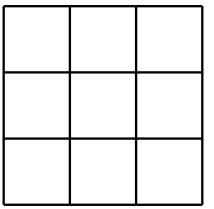
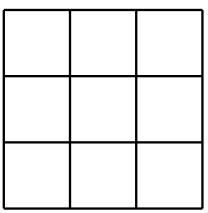
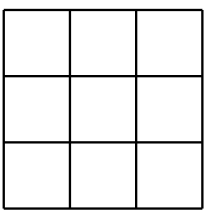
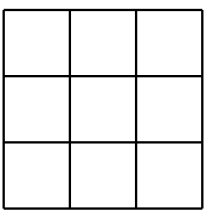
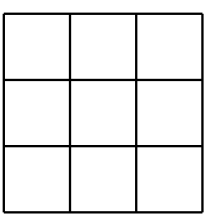
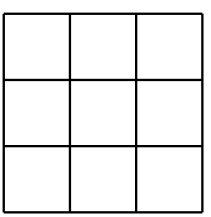
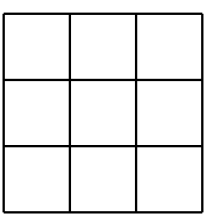
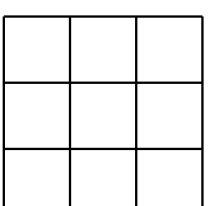
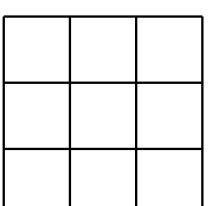
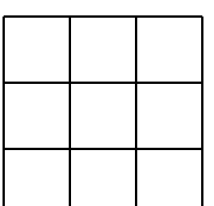


1.5. *Algorithm:* Route

- Global Routing: Partition block into global-routing tiles (gtiles) then create a global routing grid; route in the global routing grid to help avoid congestion and guide detailed routing
- Detailed Routing: Route using metal wires and vias on the low-level routing grid

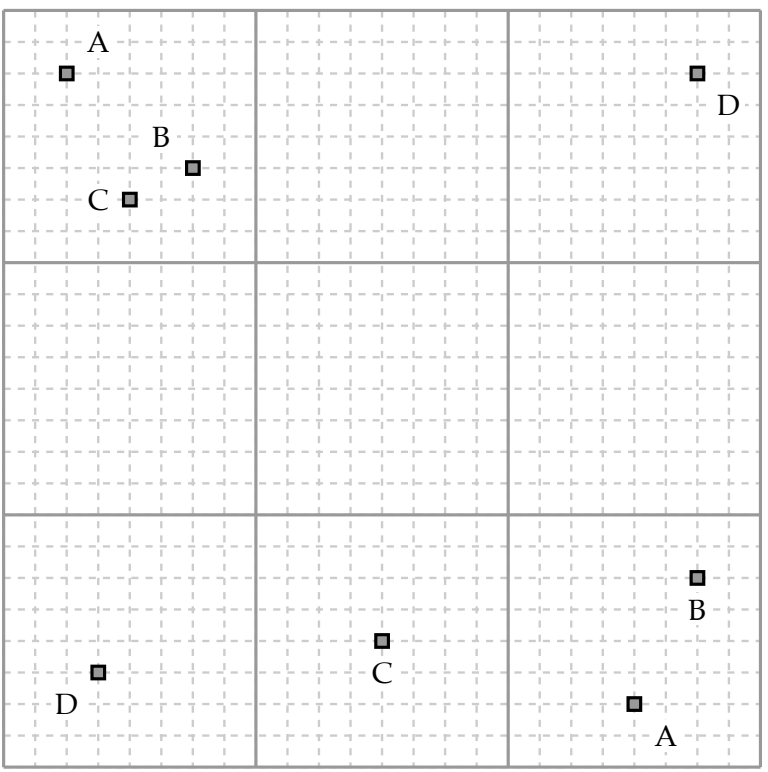
Global Routing



Net	Route	New Cost
		
		
		
		

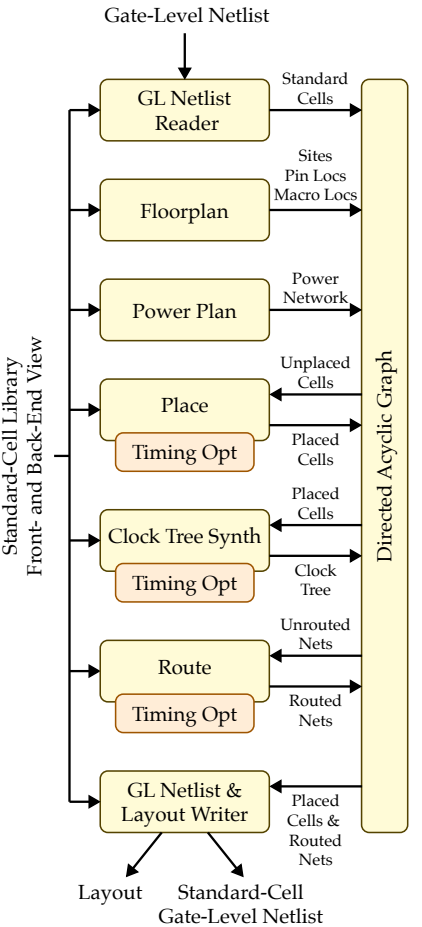
Detailed Routing

- For each net, use global route to mark some global-routing tiles as obstacles then route using maze routing within the tiles of the global-routing path
- If cannot route a net, *rip-up* just a few of the already routed nets, route the blocked net, then *reroute* the ripped-up nets



1.6. *Algorithm:* Timing Optimization

- 1. Static Timing Analysis
  - Run static timing analysis
  - Identify violating endpoints and paths with worst slack
- 2. Focus on Local Region
  - Trace back from violating endpoints
  - Select gates with low slack, high criticality (appear on many bad paths)
- 3. Generate Candidate Optimizations
  - Increase gate size
  - Insert buffers
  - Restructure gates
- 4. Estimate Slack Impact
  - Use simple delay models or local sensitivity heuristics
  - Only keep candidates which improve slack
- 5. Validate with Incremental STA
  - Recompute timing only in affected region
  - Get accurate slack change for each optimization
- 6. Apply Best Timing Optimization
  - Choose optimization with best slack improvement, lowest cost (area/power)
- 7. Iterate
  - Rerun STA
  - Critical paths shfit
  - Repeat until meet timing


Gate-Level Netlist
GL Netlist Reader
Standard Cells
Floorplan
Sites
Pin Locs
Macro Locs
Power Plan
Power Network
Place
Timing Opt
Unplaced Cells
Placed Cells
Clock Tree Synth
Timing Opt
Placed Cells
Clock Tree
Route
Timing Opt
Unrouted Nets
Routed Nets
GL Netlist & Layout Writer
Placed Cells & Routed Nets
Directed Acyclic Graph
Standard-Cell Library
Front- and Back-End View
Layout
Standard-Cell Gate-Level Netlist