Buffered Routing Tree Construction under Buffer Placement Blockages

Wei Chen, Massoud Pedram
Dept. EE – Systems
Univ. of Southern California
Premal Buch
Magma Design Automation

Outline

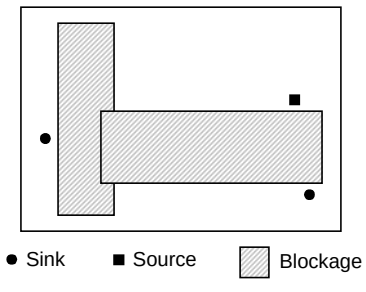
- Introduction
- Delay Models and Data Structures
- Algorithm
- Complexity Analysis
- Experimental Results
- Conclusions

Previous Work

- Insert buffers into a pre-determined net topology
  - Van Ginneken ’90
- Insert buffers while generating the net topology (using a set of buffer stations)
  - Two-pin nets: Zhou ’99, Jagannathan ’00
  - Multi-pin nets: Cong ’00

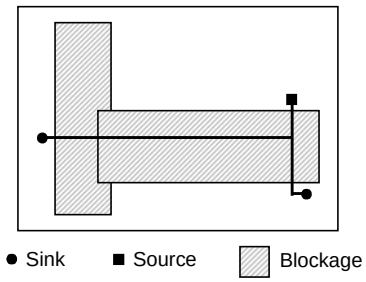
Motivation

- Placement blockages in the circuit restrict the areas on the chip where the buffers can be placed
- These blockages are easily determined after the global placement



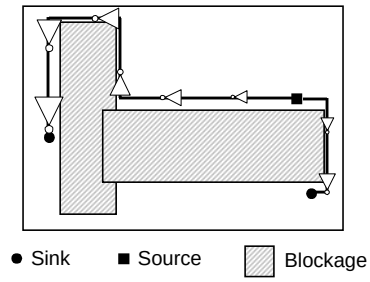
Solution I

- Conventional flow
  - Build the Steiner Tree first and then insert buffers
  - There may be no opportunity to insert buffers due to placement blockages



Solution II

- Alternative Flow
  - Force the global router to go around all the blockages and then insert buffers
  - This scheme tends to insert too many buffers



Problem Definition

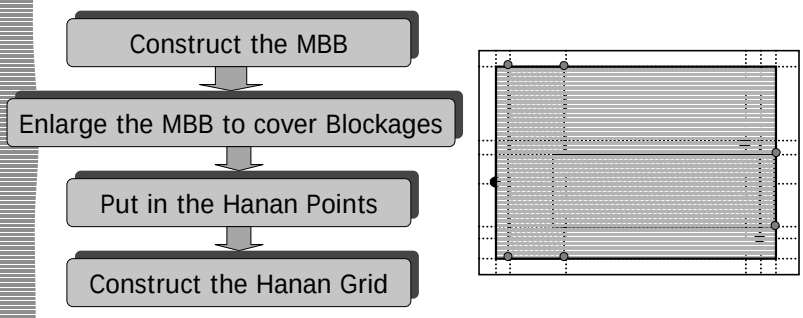
- Given (1) a set of placement blockages, where routing is allowed but no buffers can be placed, and (2) locations of the source pin and the sink pins of a net, simultaneously build the net topology and insert sized buffers/inverters at places where they are permitted to maximize the required time at the source

Delay Models

- Buffer delay: $d = \tau \cdot (p + gh)$
  - p : parasitic delay
  - g : logical effort
  - h : electrical effort or gain
- Interconnect delay: $d_w = r_w \cdot (0.5c_w + C)$
  - r_w : wire resistance
  - c_w : wire capacitance
  - C : capacitive load

Hanan Grid Graph

- Iteratively enlarge the boundary of the Hanan graph to include the source pin, the sink pins, and the corner points of the blockages



Key Data Structures

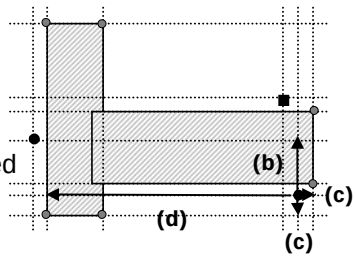
- Solution
  - Root node: *root*
  - Capacitive load: *cap*
  - Required time: *req*
  - Reachable set: *reachable_set*
  - Repeater type and size: *repeater*
- Priority queue
  - The solution that is ejected from the queue is one with the maximum required time (i.e. the least critical solution)

Initializing the Base Solutions

- Sink node *p*:
  - *sol(p, cap, req, {p}, 0}*
- Source node *p* or Hanan point *p:*
  - If the location does not allow a buffer, then
    sol(p, 0, +∞, {p}, 0}
  - Otherwise
    sol1(p, 0, +∞, {p}, 0}
    sol2(p, 0, +∞, {p,}, {buffer,0}}
    sol3(p, 0, +∞, {p,}, {inverter,0}}

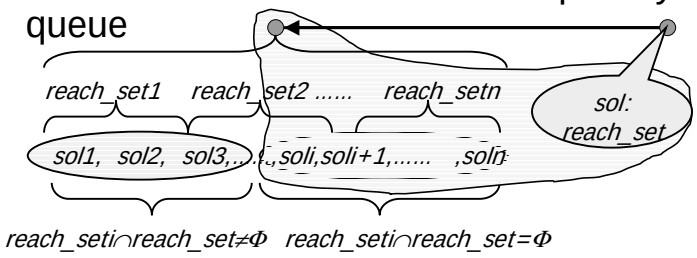
Destination Nodes

- Eject a solution from the queue
- Expand it using line probes;
  Stop expansion if we reach:
  - (a) The source or a sink node
  - (b) A Hanan point formed by two lines passing through the source and a sink node;
  - (c) A Hanan point that is not covered by a blockage covering the *root*
  - (d) The first unblocked Hanan point after the probe line extends beyond the blockages



How to Combine Solutions

- When a current solution arrives at its destination node, combine it with any solution that is rooted there and has a reachable set that does not share a node with reachable set of the current solution
- Insert the new solution into the priority queue

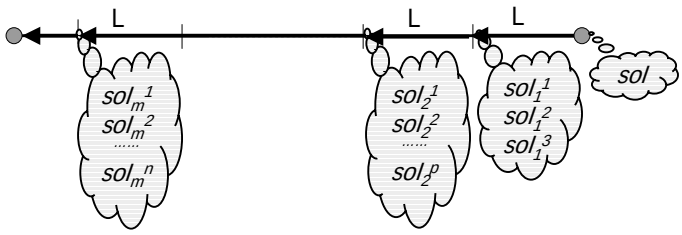


Data Member of a High Level Solution

- *Root :* is the destination node
- *reachable_set* : union of the reachable sets of the two child solutions
- *cap*, *req* : as is done in previous works
- *Repeater :* the repeater that is placed at the root node

Long Edge Buffering

- Because of the existence of long edges in the Hanan grid, buffer insertion should not be limited to the Hanan points
- Split a long edge into a set of segments
- Buffer the segments



Pruning Operation

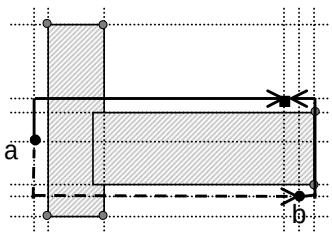
- For solutions *sol1* and *sol2,* which are rooted at the same node and have the same sink pin set, if *req1* ≤ *req2* and *cap1* ≥ *cap2*, then *sol1* is inferior to *sol2*
- Optimal pruning: Delete the inferior solutions
- Non-optimal pruning: Delete non-inferior solutions that are sufficiently close to other unpruned solutions in order to reduce the problem size

Pruning Cases

- When buffering a long edge
  - Prune after a fixed number of segments are processed
- When combining a solution with the solutions rooted at its destination node
  - Delete the inferior solutions
- Only non-dominated solutions are injected into the priority queue

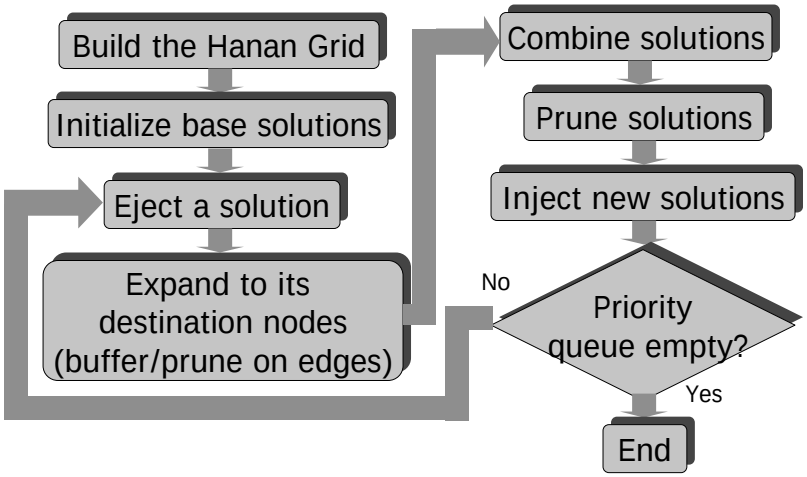
Heuristic to Reduce the Problem Size

- When a solution reaches the source pin, all other solutions with the same sink pin set are removed
  - Reduces the problem size
  - Improves the net topology



req_a>>req_b

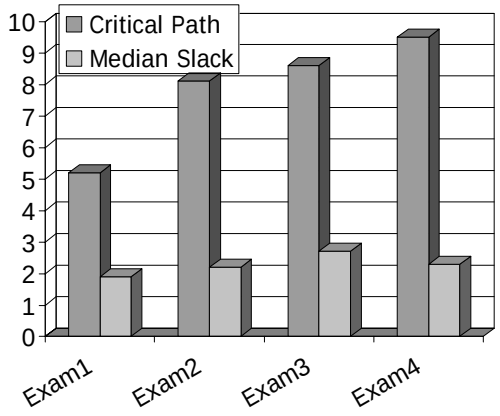
Algorithm Flow



Complexity Analysis

- Assume *N*M* Hanan grid, *n* sinks
- Space complexity
  - $O(N \cdot M \cdot 2^n \cdot K)$
  - *K:* Keep at most *K* solutions after pruning
- Time Complexity
  - $O(N^2 \cdot M^2 \cdot 2^n \cdot K)$
  - At most *N*M* steps are needed to propagate a solution to the source
  - Never revisit an edge for any solution

Experimental Results



	Cell
Exam1	8087
Exam2	38127
Exam3	62187
Exam4	767982

Conclusions

- Proposed an algorithm for concurrent global routing and buffer insertion in the presence of placement blockages
- Used line probe-based maze routing and solution pruning to speed up the algorithm and reduce the problem size
- Results on four industrial circuits show 6-10% reduction in the circuit delay