XXVII IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, OCTOBER 5, 2009

Symmetrical Buffer Placement in Clock Trees for Minimal Skew Immune to Global On-chip Variations

Renshen Wang[1]
Takumi Okamoto[2]
Chung-Kuan Cheng[1]

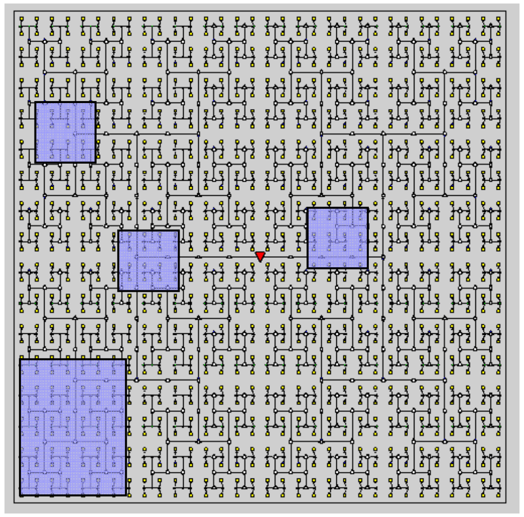
[1]University of California San Diego

[2]NEC Corporation

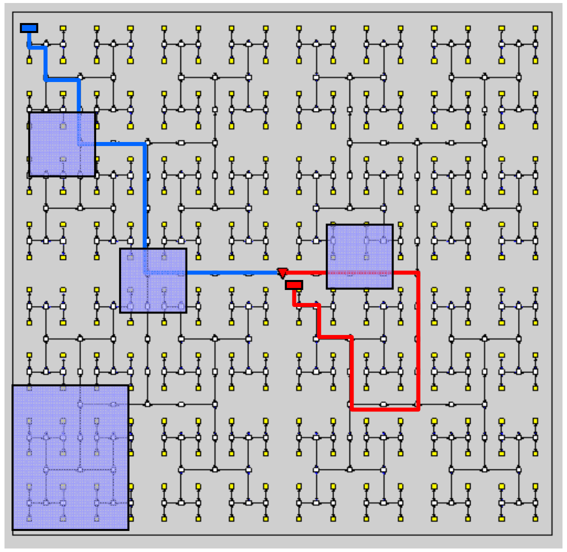
Synopsis

- Zero skew clock trees by symmetry
  - Zero in theory/simulation
- Buffer insertion for minimum delay (previous work)
- Dynamic programming under the constraint of symmetry
  - Difference from general buffer insertion
  - Analysis
- Implementations and test cases
- Conclusion and future works

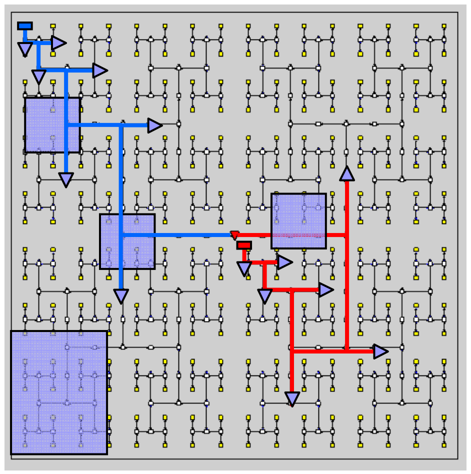
Introduction



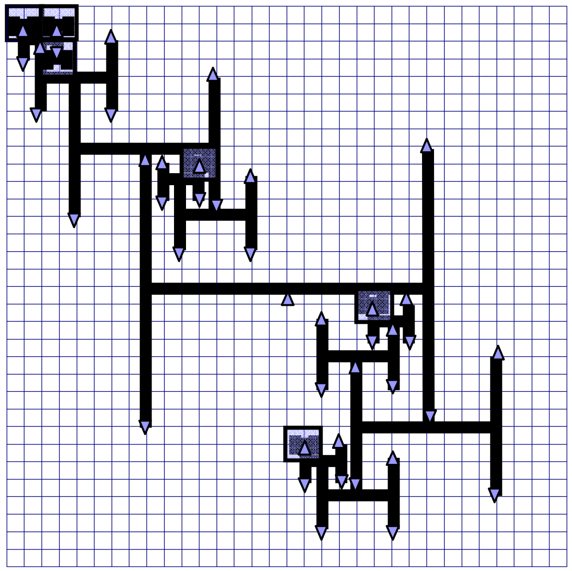
- Minimize clock skew
  - Various strategies for high performance VLSI design
- Symmetrical structures can achieve zero-skew in theory
  - H-trees, each level of tree branches are in "H" shape
  - Buffers are inserted reduce delay from the source to sinks



Introduction (cont.)

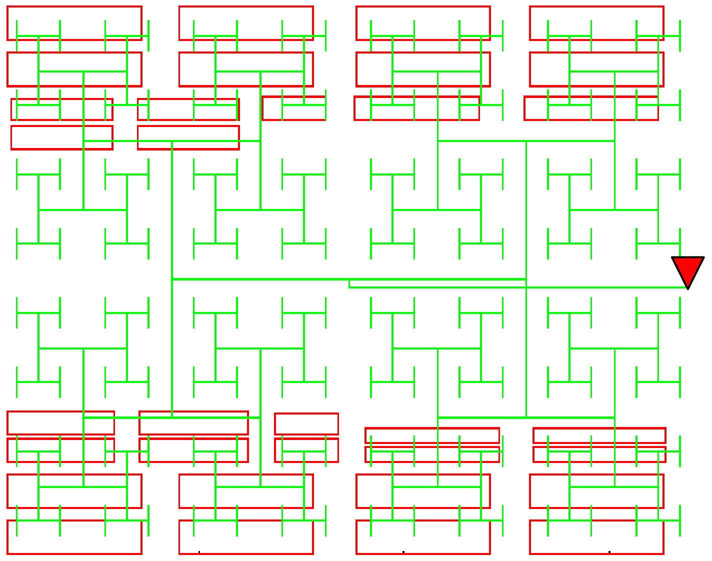


- Buffer placement constraints
  - Hard macros on the chip – buffers cannot be placed
  - Buffers are placed in a **symmetrical pattern** to preserve zero-skew
- Strong symmetry constraint
  - Remove unused paths / redundant sub-trees
  - Buffer placement in the remaining tree



Problem Formulation

- Given a clock tree with symmetrical structure, given a set of hard macros
- Symmetrically insert buffers in the clock tree (so that the clock skew at the sinks is always zero)
- Buffers cannot be place in hard macro areas
- Tree leaves covered by hard macros are not considered as sinks
- Minimize delay



Elmore delay

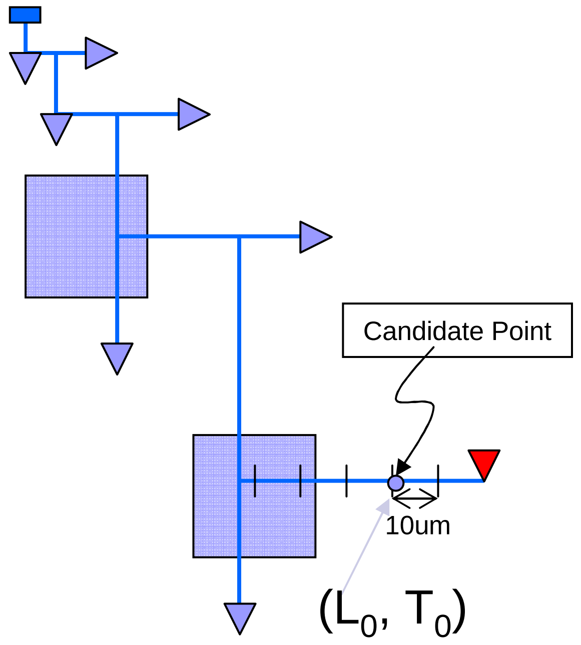
- Delay calculation: Elmore delay model
- For an RC tree network, the Elmore delay on the path from root to sink *i* is a sum over nodes along the path,

$$D_i = \sum_{k \in \pi(i)} R_k L_k$$

  - π (*i*) is the set of nodes from root to node *i*
  - L_k is the total capacitive load in the sub tree rooted at node k
  - E.g. the delay on a uniform wire segment is ½RC

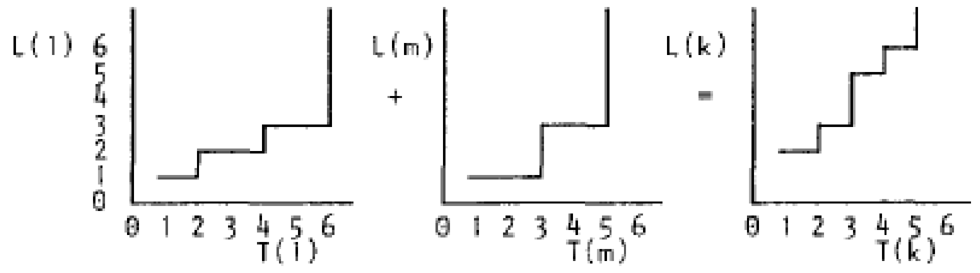
Buffer Placement for Min Delay

- General problem observes “Principle of optimality” in dynamic programming
  - Once the capacitive load at a node is determined, the partial solution can be extended from the node
  - At node N_0, the minimum delay at load L_0 is T_0
  - At the next node, the delay can be computed as T_0+Delay(L_0, R, buf, …)



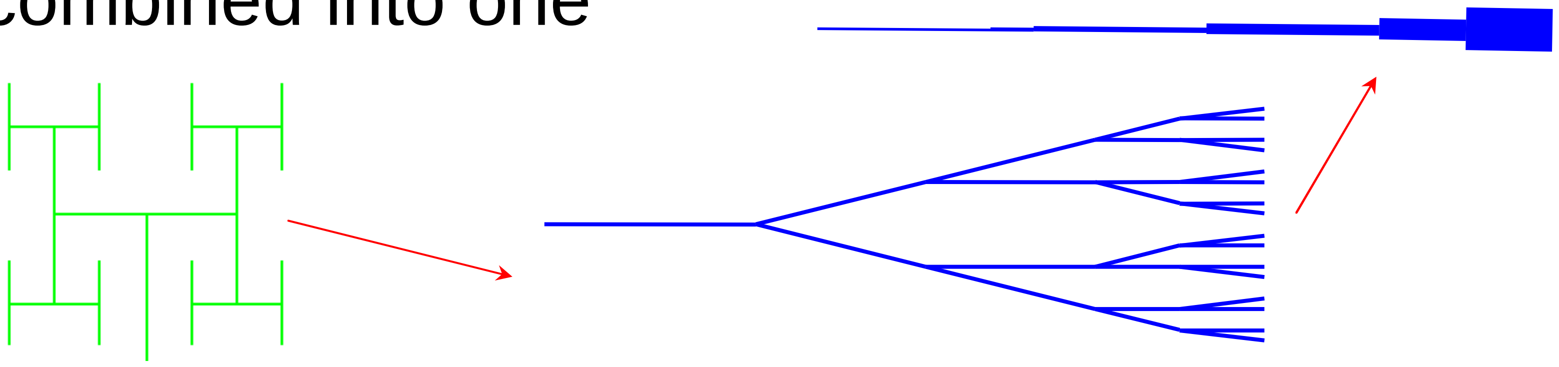
Dynamic Programming

- Ginneken's algorithm: delay minimization in general RC tree networks
  - Keep a curve of (load, time) states at each node
  - The curves can be computed from the sinks to the source
  - At a node with multiple branches, the curves can be added as (L_1+L_2, T)



Symmetrical Buffer Insertion

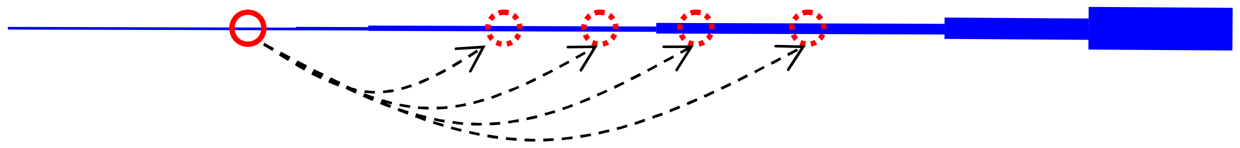
- With symmetry constraints, the paths can be combined into one



- The RC-tree network is equivalent to a segmented path from the source to the sinks
- At each branching point, line width doubled
  - Capacitance rate doubled
  - Resistance rate halved

Different Dynamic Programming

- Instead of sweeping the (L, T) curve, we sweep the buffer candidate points

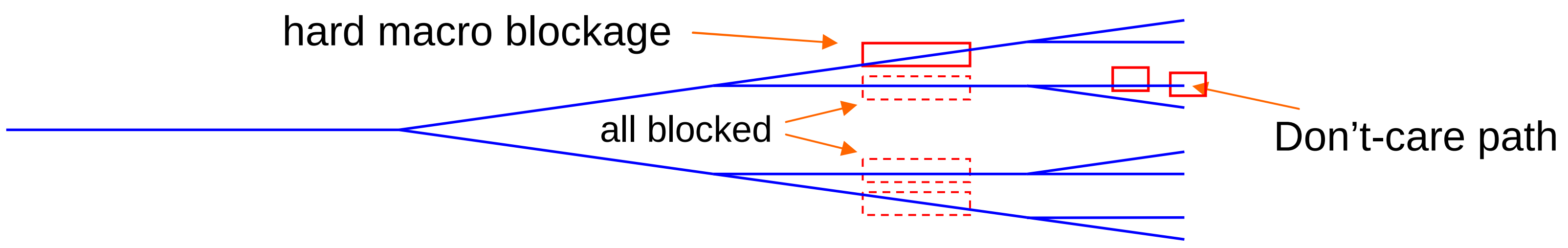


$$T[k] = \min\{T[i] + \mathrm{Delay}(k,i) : k < i < B\}$$

  - T[k] is the delay from node k with a buffer inserted on the node
  - Passing each branching point, R halved and C doubled (for wires and buffers)
- Same effect as Ginneken’s algorithm

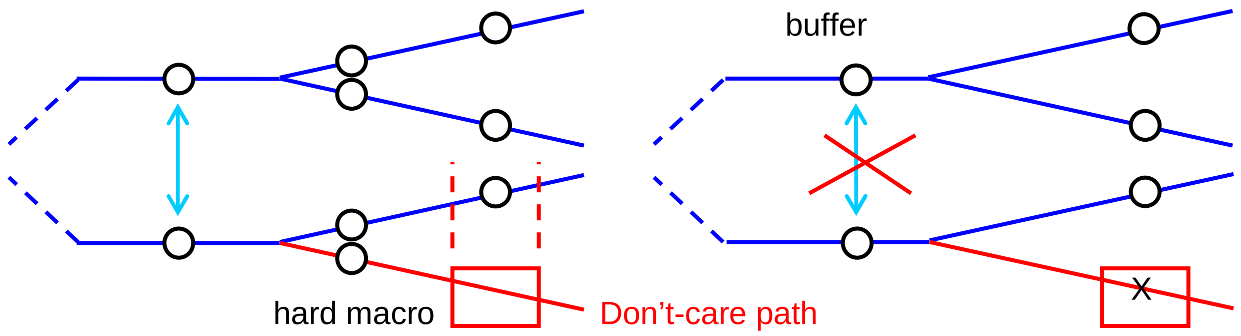
Blockage by Hard Macros

- Many candidate points are blocked due to
  - Non-overlapping constraint
  - Symmetry constraint



- Blockage may depend on path
  - A "don't-care" path does not lead to any sink
  - Macros covering a "don't-care" path does not always affect other paths

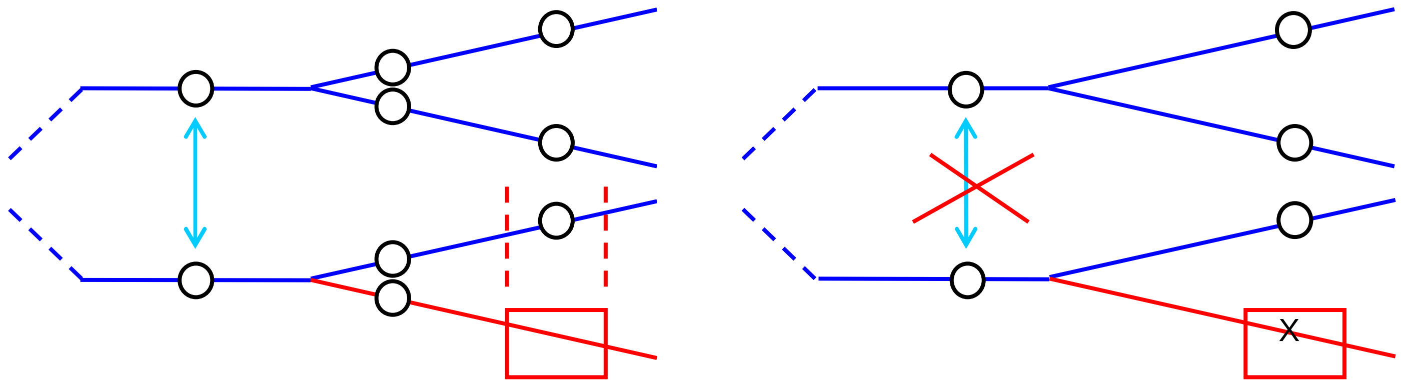
Blockage with Don’t-care Paths



- 2 cases for the symmetry of buffers
  - ✓ On left, all paths are identical
  - ✗ On right, the upstream buffer of the don’t-care path is on a higher level, and two upper level buffers have different loads
- Symmetry depends on the upstream buffer
- Level-dependent blockages

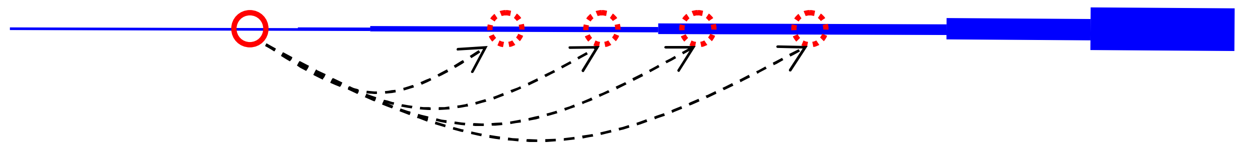
Why Different Way?

- Ginneken's algorithm is not convenient to handle symmetry constraints
  - Level of upstream buffers affects the blockage of downstream buffers
  - State of (load, time) at N_0 not sufficient to describe the partial solution beyond this node
  - Cannot distinguish the two cases



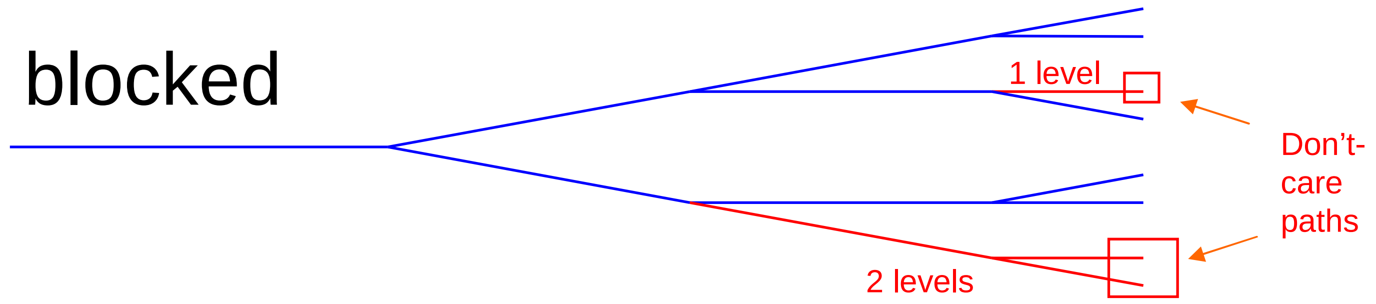
Dynamic Programming for Symmetrical Buffer Insertion

- Our algorithm



$$T[k] = \min\{T[i] + \text{Delay}(k,i) : k < i < B \text{ and notBlocked}(\text{level}(k), i)\}$$

- notBlocked(l, n) is pre-computed
  - =1: with the upstream buffer at level l, node n is not blocked
  - =0: blocked



Dynamic Programming for Symmetrical Buffer Insertion (cont.)

- Bit operations for storing notBlocked(l, n)
  - For n-node tree, number of levels is log$n$
  - Integer operations are performed in unit time O(1)
- The formula can be modified as

$$T[k] = \min\{T[i] + \text{Delay}(k,i) : k < i < B \text{ and } \text{Blocked}(i) \equiv 0 \bmod 2^{level(k)}\}$$

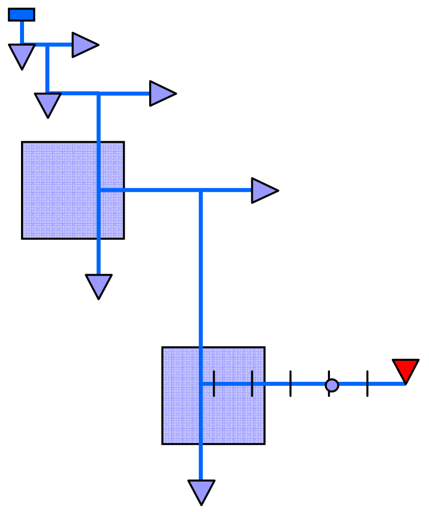
Blocked[k]

0/1	Level 6
0/1	Level 5
0/1	Level 4
0/1	Level 3
0/1	Level 2
0/1	Level 1

- The overall time complexity of the dynamic programming is O(n^2)
- Space complexity O(n)

Brief analysis

- On time complexity
  - Basically same as Ginneken's algorithm
  - Series of states at each node (Ginneken) vs. Series of nodes
  - Capable of handling symmetry constraints
- On space complexity
  - Ginneken's alogrithm needs to keep every curve in order to track the optimal solution, overall $O(n^2)$
  - Our algorithm only needs O(1) space at each node, overall O(n)



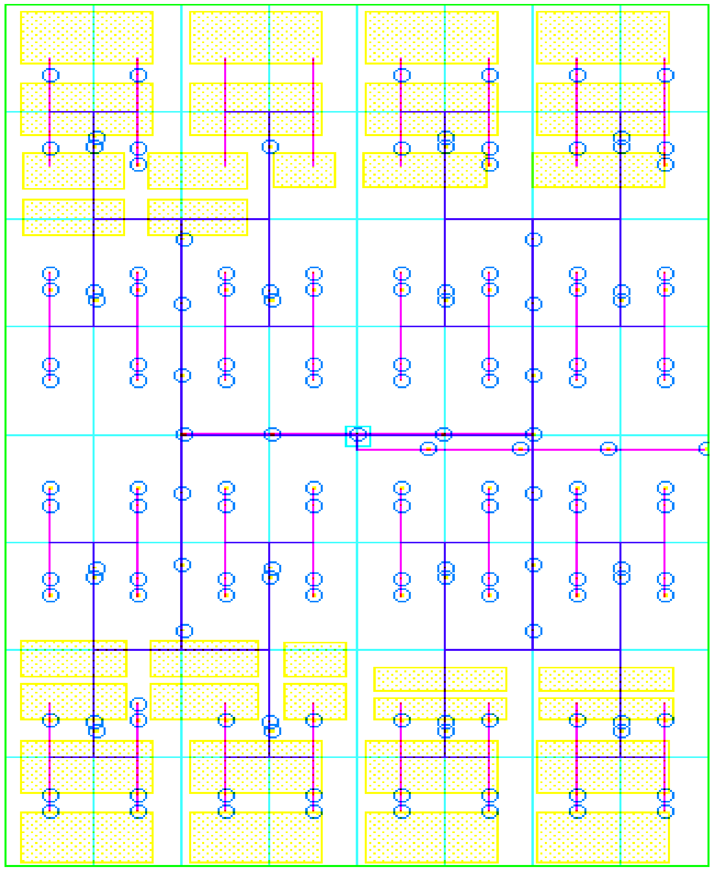
Implementations

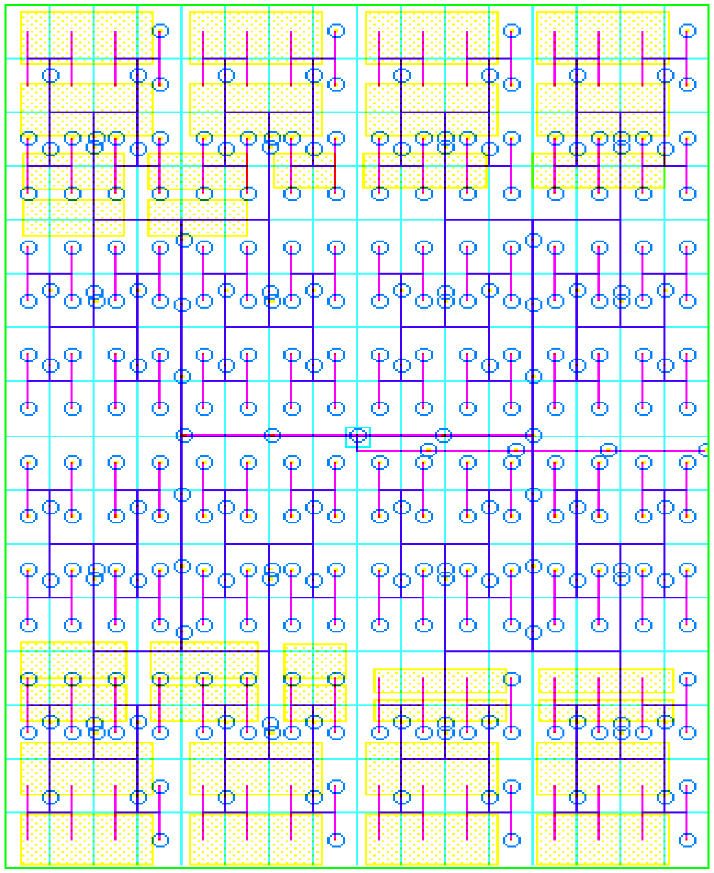
- In real implementations, some more issues
  - Different types of buffers to choose
  - Inverters, total number along a path must be even
  - Wire sizing: each net may have unit width, 2X, …, wX width
- For buffer types, we add another dimension in array T, i.e. T[k,b]
  - the delay from node k with a buffer (type b) inserted on the node
  - The load on this node is the input capacitance of the buffer
  - To add inverters, add another dimension (1,-1):

    T[k,b,1] is the state with clock signal

    T[k,b,-1] is the state with switched signal
- For wire sizing, we add a loop of choosing different wire size in Delay(k, i), from 1X to maximumX

Test Cases



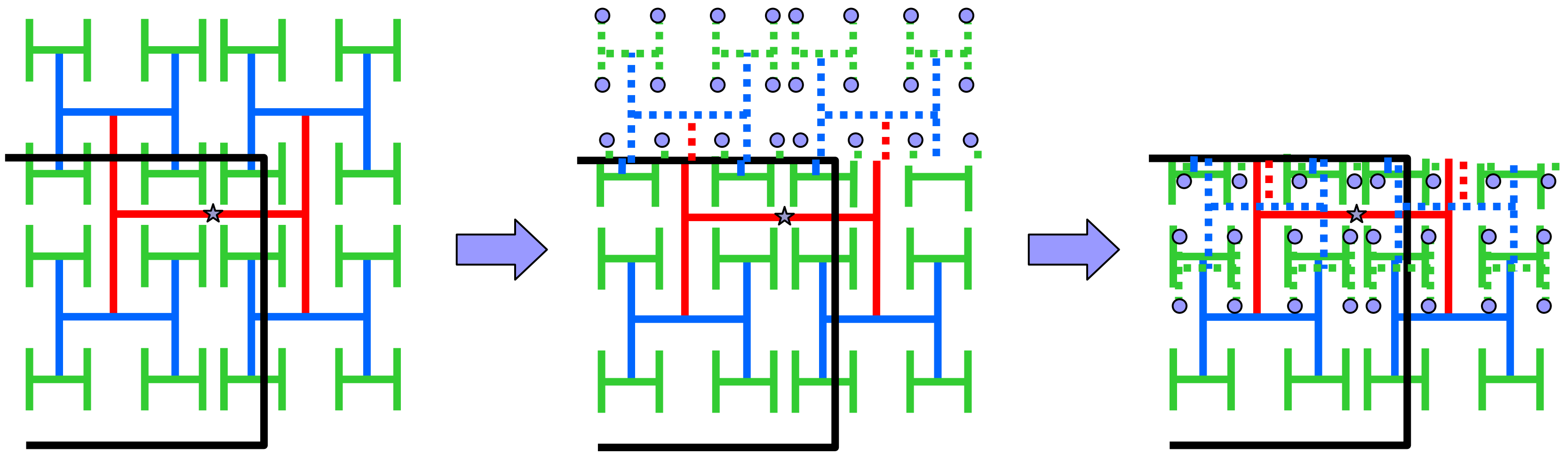


- 8x8 case
- Program feedback:1.07ns
- Simulation result: 1.16ns

16x16 case
Program feedback:1.18ns
Simulation result: 1.28ns

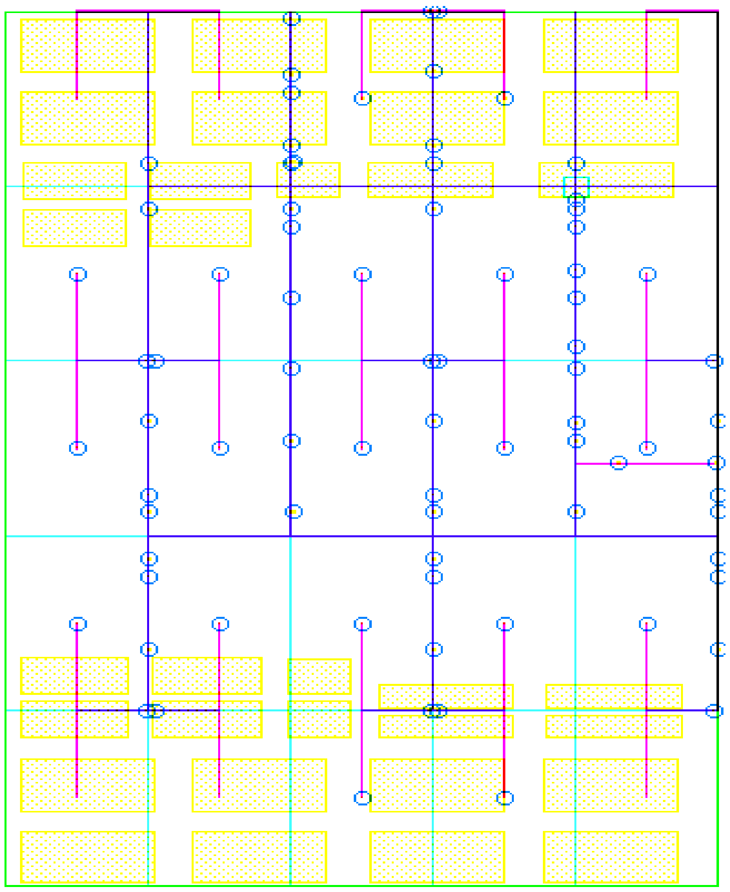
Folded Structures

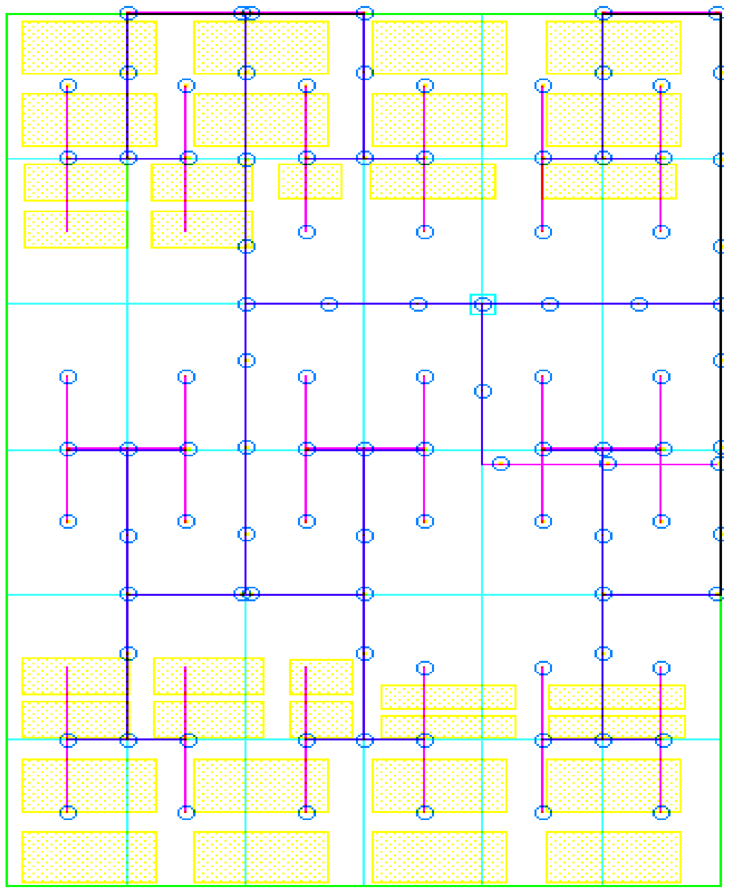
- Sometimes the chip size is not exactly 2^p times of LCD area size



  - Fold the outside part of the H-tree into the chip area
  - Symmetrical topology unchanged
  - The folded wires outside are don't-care paths

Test Cases (cont.)





- 5x5 case
- Program feedback:1.67ns
- Simulation result: 1.69ns

6x6 case
Program feedback:1.12ns
Simulation result: 1.26ns

Different Objectives

- Not only to minimize delay, also
  - Buffers, wires, total capacitance, etc
- Each resource can be optimized separately by dynamic programming, but joint optimization is harder
  - Add dimensions in dynamic programming, e.g.

    T[k, b, 0/1, #buf]

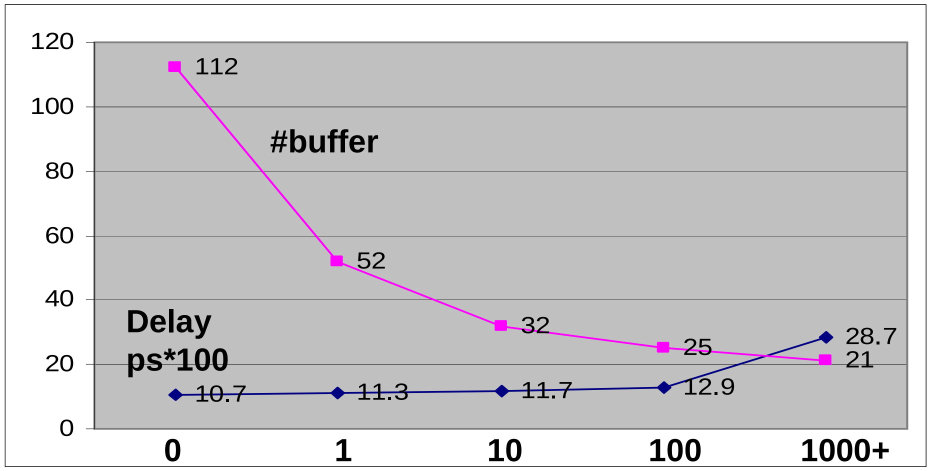
    May result in intractable complexity
  - ✓ Only modify cost function, e.g. minimize

    $\text{Delay} + k_1 \text{ \#buf} + k_2 \text{ wirelength}$

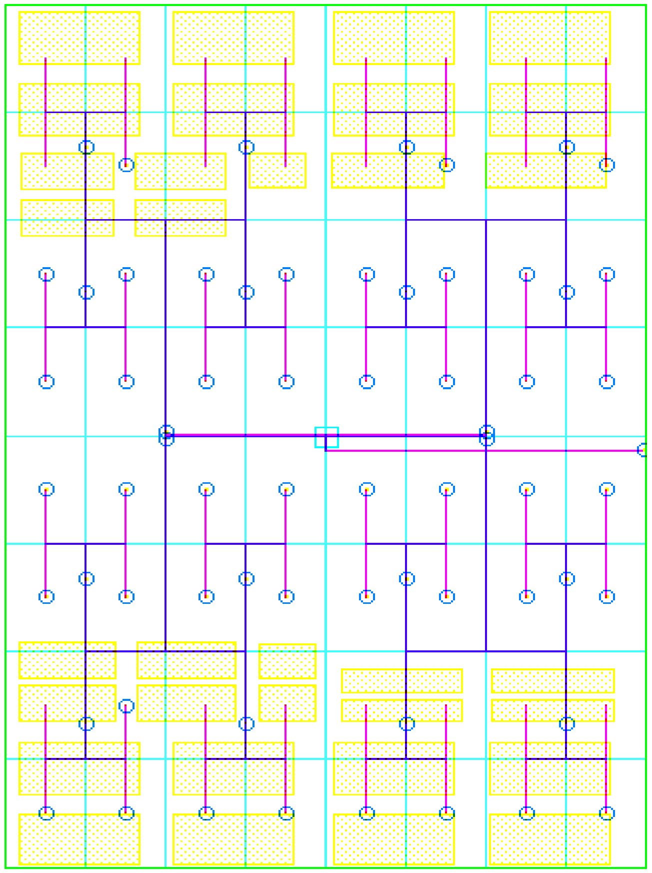
    overall problem does not satisfy "Principle of Optimality"

Test Cases (cont.)

- Joint optimization of delay and number of buffers
  - Trade of between time and resource
  - Optimality is not guaranteed



Cost function = delay (ps) + #buffer * X



Minimum #buffer: 21

Conclusions

- Dynamic programming for symmetrical buffer insertion in zero-skew clock trees
- The basic techniques similar to Ginneken's algorithm, but in a different form
  - Guarantees optimality for single objective
  - Provides approximate solutions for joint optimization
- The advantage of sweeping along the nodes
  - Capable of handling symmetry constraints
  - Space requirement is lower
- Time complexity is $O(n^2)$
  - For usual cases with chip size under $10x10mm^2$, the running time is low (below 10 sec)

Future Works

- Other formulation / optimization objectives
  - Routing resource: wire length, wire length x wire sizing
  - Total capacitance: buffer input capacitance + wire capacitance
- Pre-assigned buffers
  - Optimize buffer insertion based on existing buffers assigned by designers
- Multi clock domain, multi clock trees
  - Zero intra-tree skew & zero inter-tree skew

Q&A

- Thank you!