

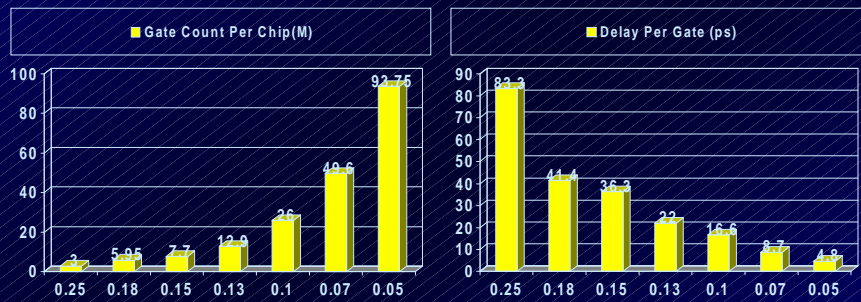
Integrated Logical Physical Design for DSM Circuits

Massoud Pedram
Department of EE - Systems
University of Southern California
Los Angeles, California

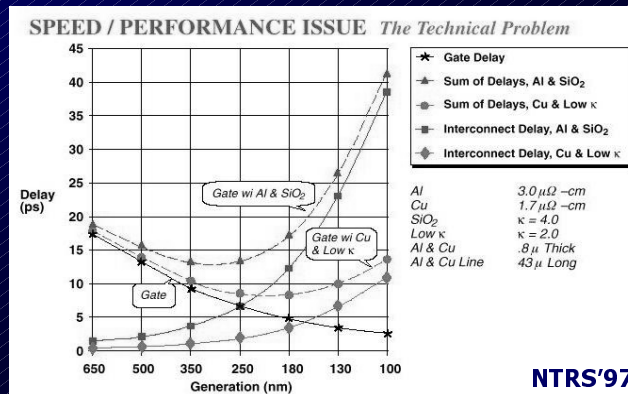
Outline

- **Motivation and Background**
- **FANROUT Algorithm**
- **Results and Discussion**
- **SCD Algorithm**
- **Results and Discussion**
- **Conclusions**

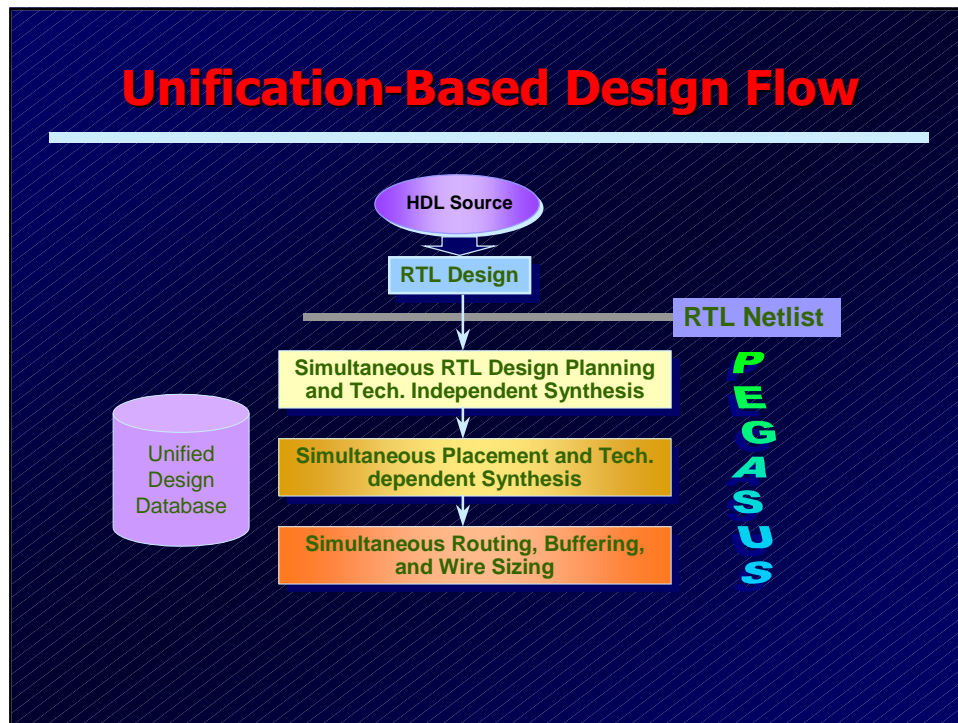
Technology Trends



Interconnect Dominance



Interconnect delays play an increasingly important role in determining the overall performance of VLSI designs



Outline

- Motivation and Background
- **FANROUT Algorithm**
- Results and Discussion
- SCD Algorithm
- Results and Discussion
- Conclusions

Prior Work

■ Fanout Optimization

- Berman`89, Singh`90, Touati`90, van Ginneken`90, Vaishnav`93, Kung`98

■ Performance-driven Routing

- Rao`92, Boese`93, Cong`93, Vittal`94, Lillis`96, Cong`97

■ Concurrent fanout optimization and Steiner tree routing

- Okamoto`96

Fanout Optimization Problem

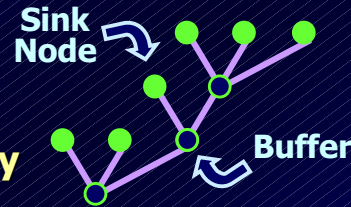
- Distribute a signal to a set of sinks with known loads and required times so as to maximize the required time at the root of the fanout tree
- A logic level operation with little access to (or use of) routing information
- NP-Complete for the general case

LT-Tree: Logical Structure

Sinks: with given required times and loads

Buffers: with given strengths

- The connection topology must be determined



- **LT-Tree Type-I:**
 - Every buffer is connected to at most one other buffer
 - No buffer has a right sibling
- Sinks with larger required times are placed further from the root of the tree

LT-Tree Based Fanout Opt.

- If the sink loads are all equal, there exists an optimal LT-Tree such that the sinks with larger required times are placed further from the root
- The LT-TREE algorithm is based on dynamic programming; Its complexity is $O(n^2)$

Routing Tree Construction

- Route a signal to a set of sinks with known loads, required times, and positions so as to maximize the required time at the driver
- A physical design operation with little power to change the logical structure of the circuit
- NP-Complete for the general case

P-Tree: Physical Structure

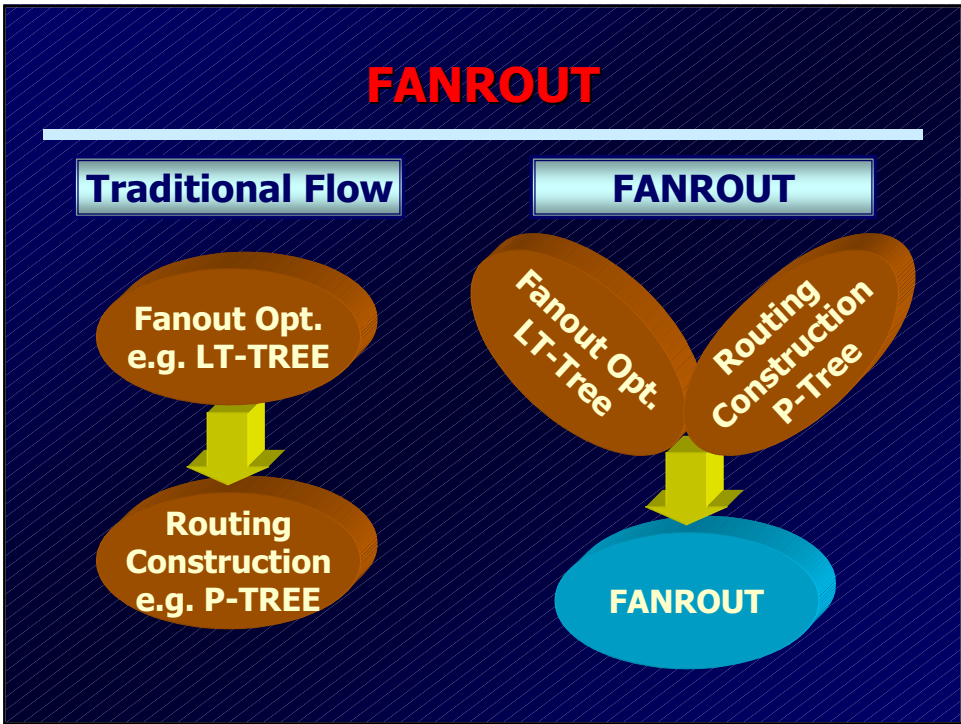
Sinks: with given required times, loads, and positions

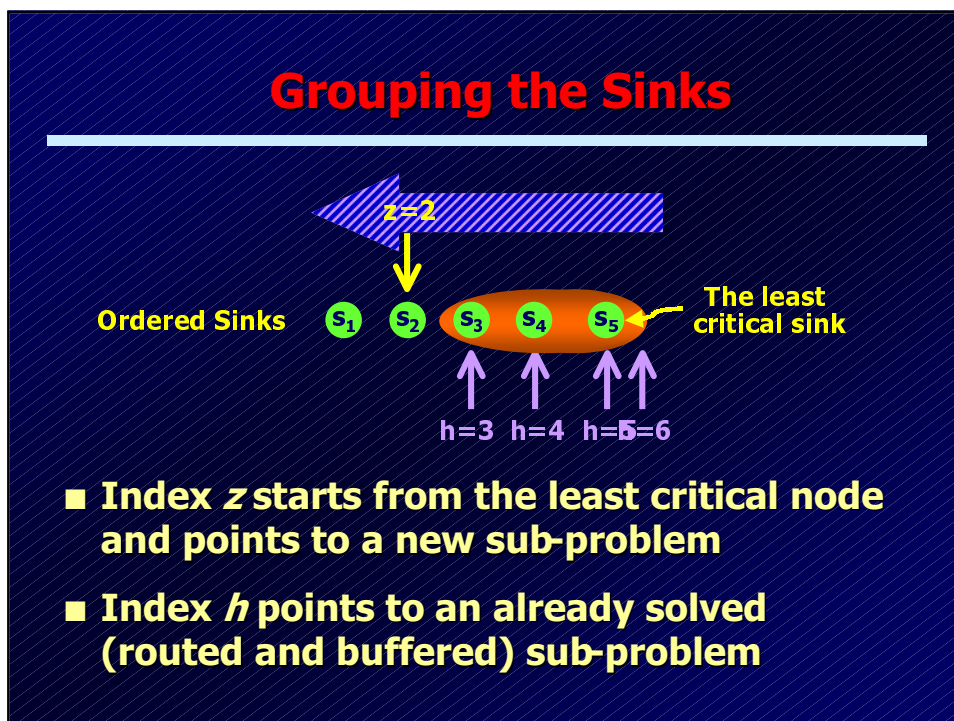
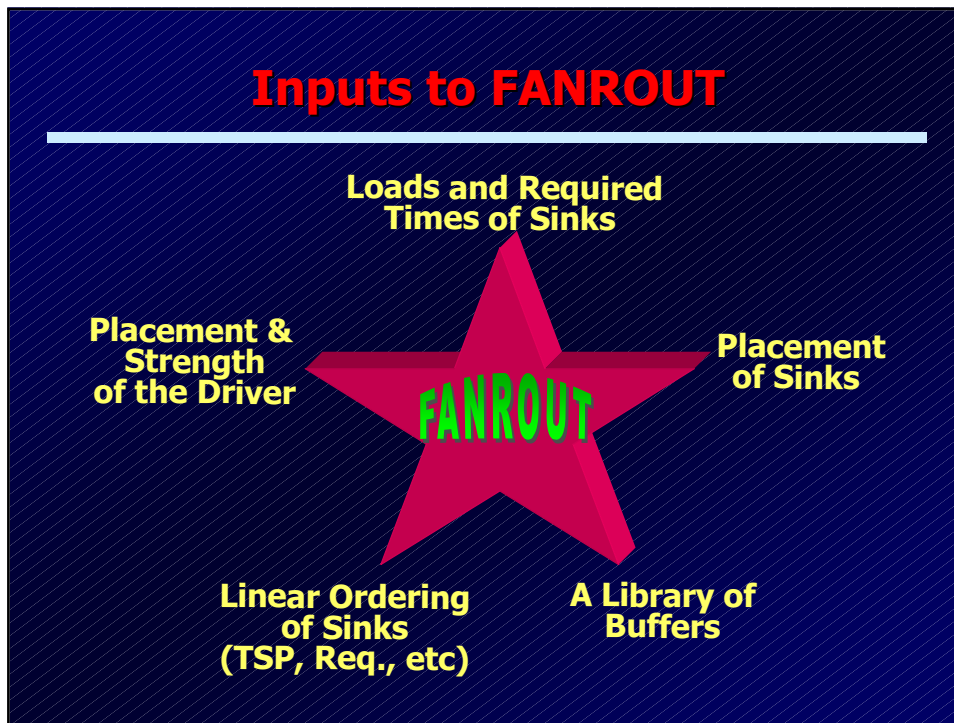
- Lengths and connection topology must be determined

- There exists a RSMT for a terminal set where all Steiner points are located on the Hanan grid graph
- In the P-TREE algorithm, the branching factor at every Steiner point is exactly three

P-Tree Based Routing Opt.

- For a given sink order, the dynamic programming-based P-TREE algorithm computes the set of all RST's with non-dominated required time and total capacitance
- The complexity of the P-TREE algorithm is $O(n^5)$





Solution Curves

- At every step of DP, a two-D solution curve is generated for every sub-problem rooted at some grid point
- Building and pruning the solution curves are the two major operations in FANROUT

A Solution Curve

Flow of FANROUT

Illustration for $z=2$ and $h=4$: d, e, c, a, b

For every Hanan point (v)

For every solution (γ)

P-TREE

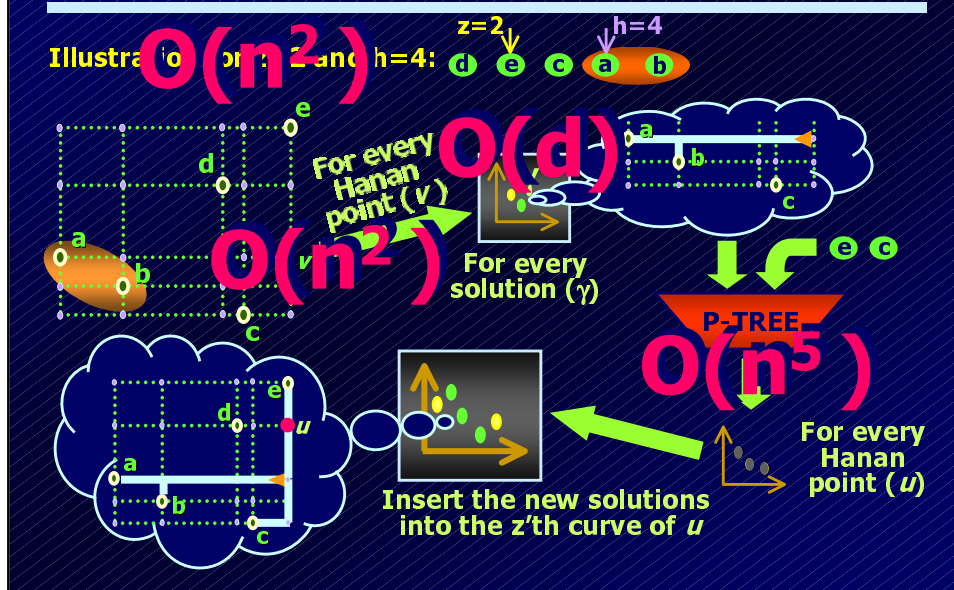
For every Hanan point (u)

Insert the new solutions into the z^{th} curve of u

Properties of FANROUT

- Finds the solution with the maximum required time at the root, subject to the given sink order and the structure of LT-Tree and P-Tree
- Does not depend on a particular gate or wire delay model
- Has polynomial time complexity (albeit it is high)

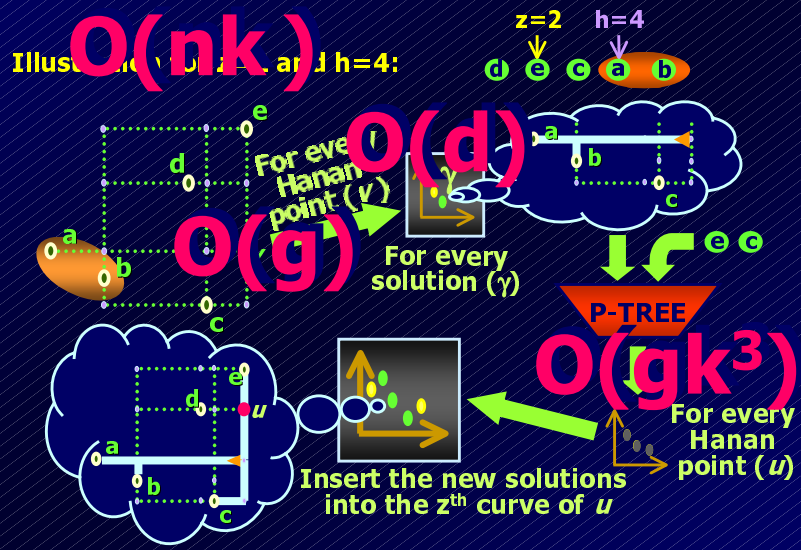
Complexity Analysis



Heuristic FANROUT

- FANROUT uses heuristics to achieve a lower runtime:
 - Limit the number of Hanan points to g points
 - Allow no more than k fanouts for every buffer
 - Use a heuristic implementation of P-TREE algorithm

Complexity of Heuristic FANROUT

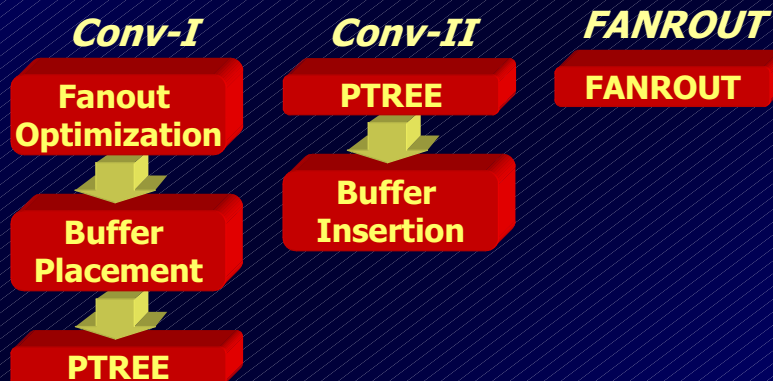


Outline

- Motivation and Background
- FANROUT Algorithm
- **Results and Discussion**
- SCD Algorithm
- Results and Discussion
- Conclusions

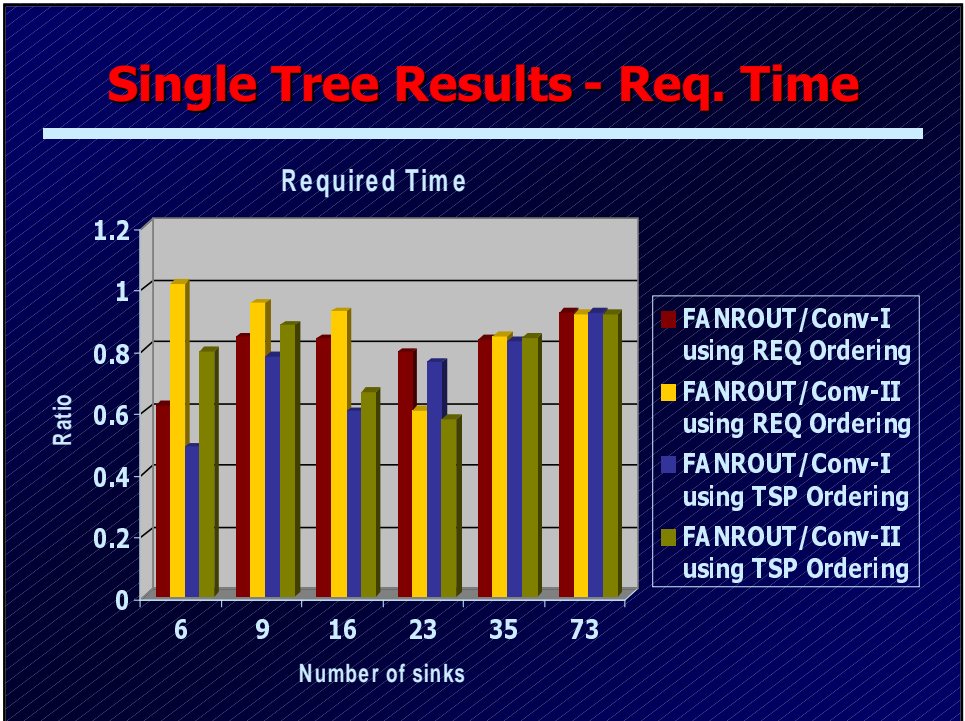
Experimental Setup 1

INPUT: Placed sinks and the driver of a net

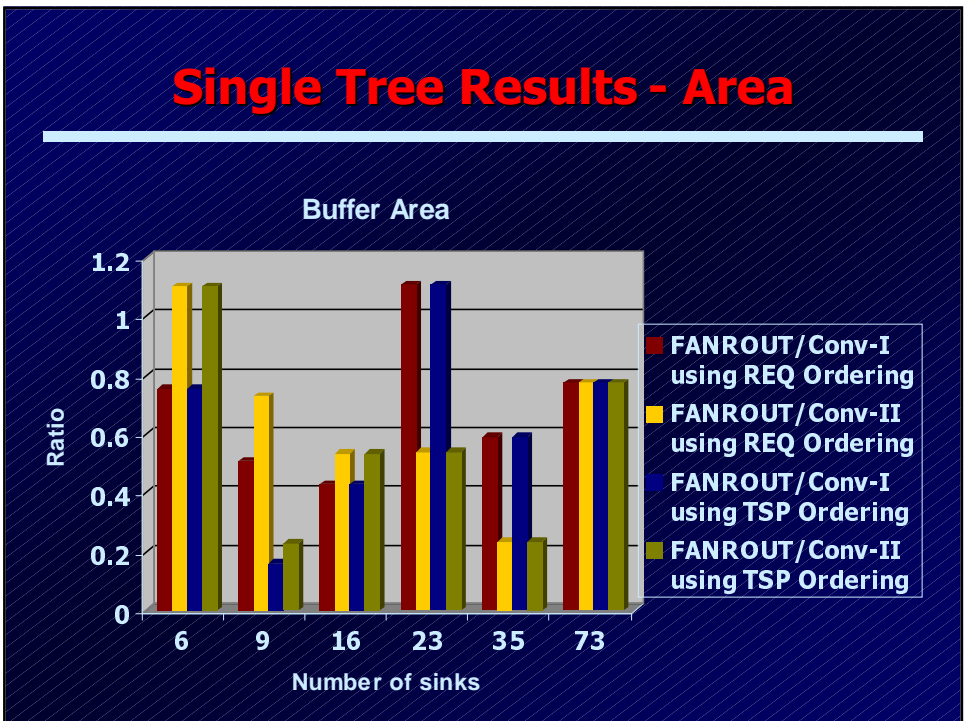


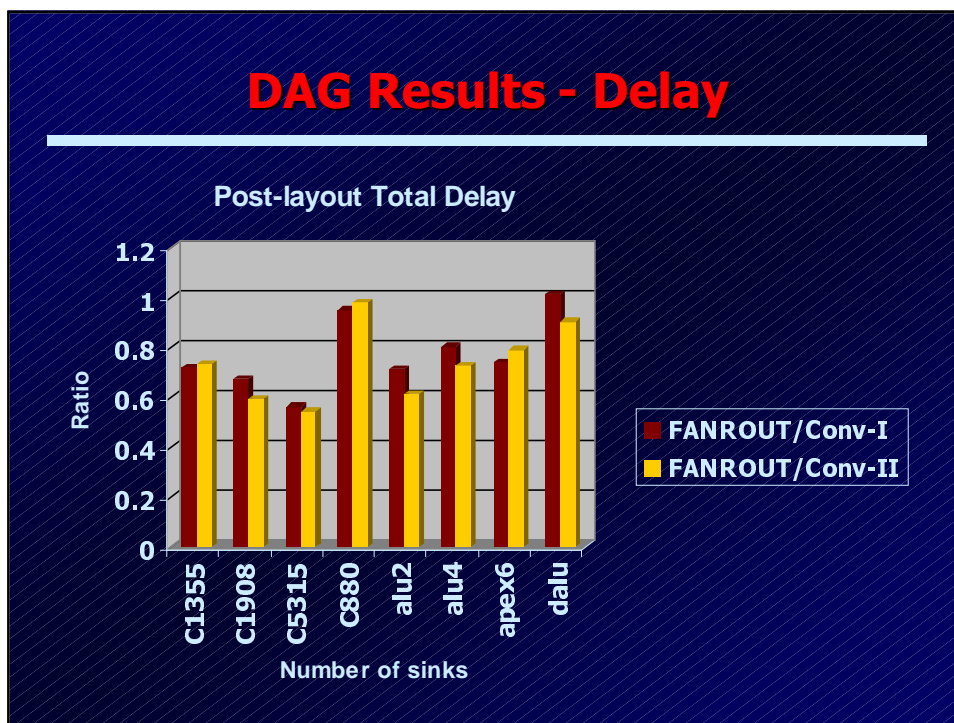
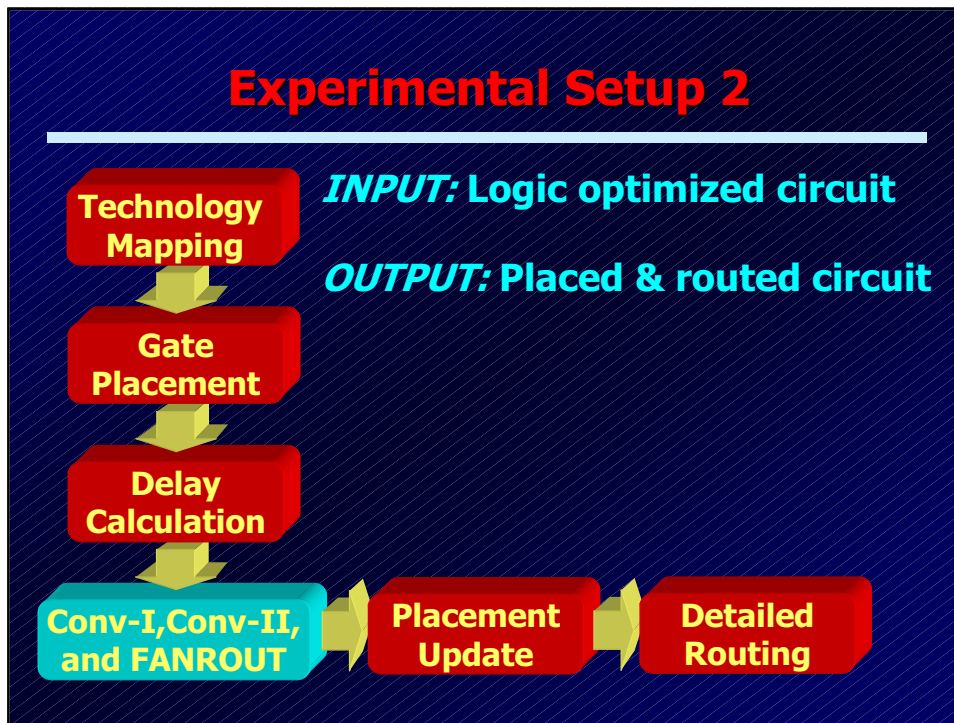
OUTPUT: Buffered routing tree

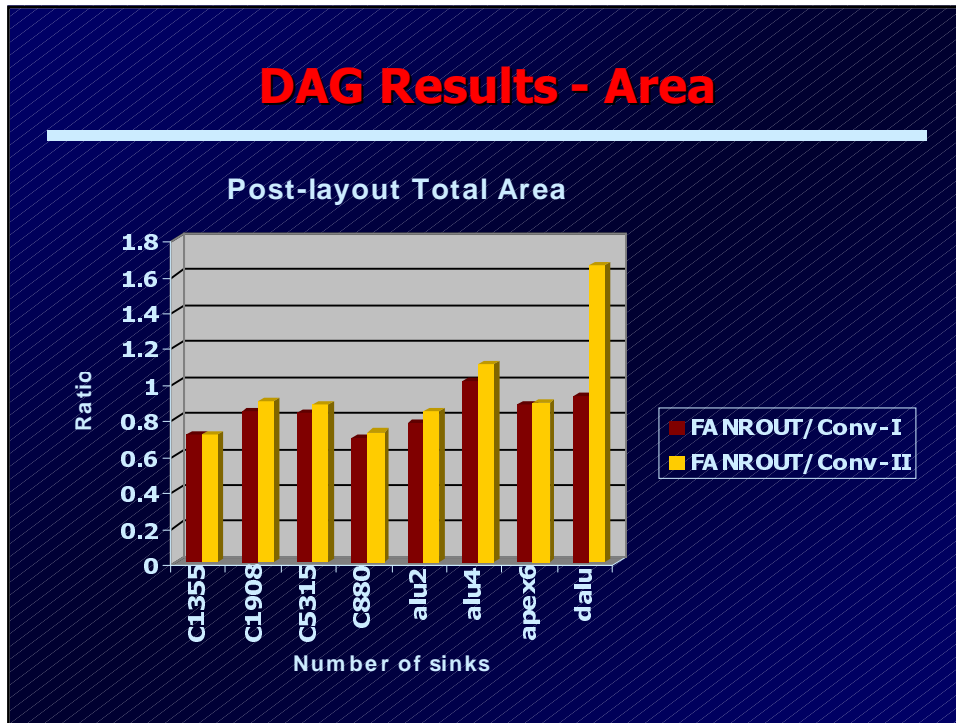
Single Tree Results - Req. Time



Single Tree Results - Area







- ### Outline
-
- Motivation and Background
 - FANROUT Algorithm
 - Results and Discussion
 - **SCD Algorithm**
 - Results and Discussion
 - Conclusions

SCD: Problem Definition

- Given a mapped and placed circuit with allowed range of gate sizes, find the best local displacement and size for each gate in the circuit so as to minimize the circuit delay

Previous Work

- In-Place Continuous Gate Sizing
 - Fishburn '85, Cirit '87, Berkelaar '90, Sapatnekar '93, Kung '96
- In-Place Discrete Gate Sizing
 - Chan '90, Li '93, Coudert '96
- Gate Sizing and Relocation
 - Chuang '94

Delay Model

$$d_{i,j} = d_{int_{i,j}} + r_{dr_{i,j}} \cdot (c_{load_j} + c_{net_j}) + r_{net_j} \cdot c_{load_j}$$

where $c_{load_j} = \sum_{g_k \in fanout(g_j)} c_{in_{j,k}}$

Delay Model, Cont'd

- **Gate Sizing Model**

$$d_{int_{i,j}}(z_j) = \alpha 1_{i,j} \cdot z_j + \beta 1_{i,j}$$

$$r_{dr_{i,j}}(z_j) = \frac{\alpha 2_{i,j}}{z_j} + \beta 2_{i,j}$$
- **Wire Load Estimation**

$$c_{in_{i,j}}(z_j) = \alpha 3_{i,j} \cdot z_j + \beta 3_{i,j}$$

$$c_{net_i} = \rho \cdot [C_{hor}(x_{net_{i,max}} - x_{net_{i,min}}) + C_{ver}(y_{net_{i,max}} - y_{net_{i,min}})]$$

$$r_{net_i} = \rho \cdot [R_{hor}(x_{net_{i,max}} - x_{net_{i,min}}) + R_{ver}(y_{net_{i,max}} - y_{net_{i,min}})]$$

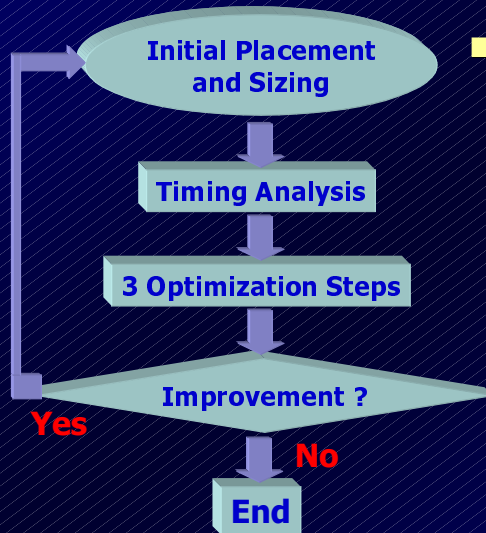
Detailed Delay Model

■ Elmore Delay Model

- non-Convex
- non-linear

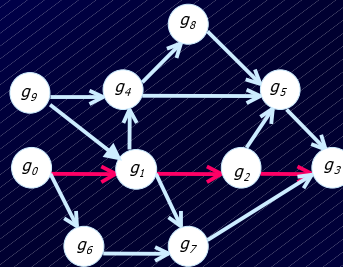
$$\begin{aligned}
 d_{i,j} = & \text{dint}_{i,j}(z_j) + \text{rdr}_{i,j}(z_j) \cdot \{ (\rho \cdot C_{hor}(xnet_{j,max} - xnet_{j,min}) \\
 & + \rho \cdot C_{ver}(ynet_{j,max} - ynet_{j,min}) + \sum_{g_k \in \text{fanout}(g_j)} \text{cin}_{j,k}(z_k) \} \\
 & + \rho \cdot \{ R_{hor}(xnet_{j,max} - xnet_{j,min}) + R_{ver}(ynet_{j,max} - ynet_{j,min}) \} \\
 & \cdot \sum_{g_k \in \text{fanout}(g_j)} \text{cin}_{j,k}(z_k)
 \end{aligned}$$

SCD Flow



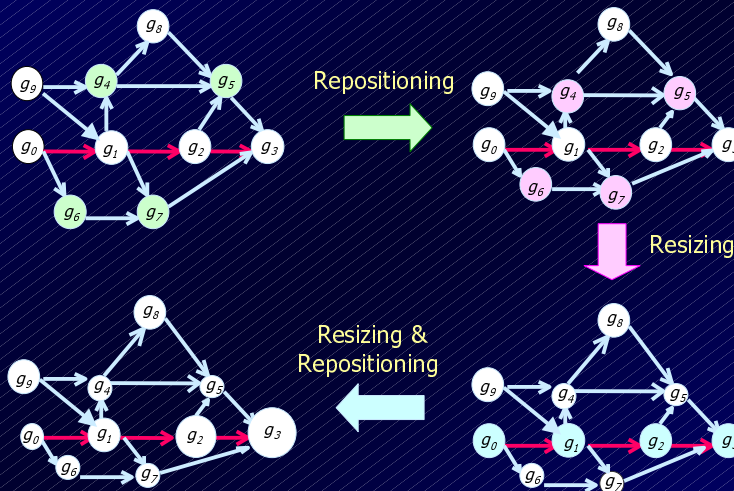
- Iteratively select and optimize gates and their immediate fanouts on the *k* most-critical paths

Motivational Example



- Critical path: $C(1) = \{g_0, g_1, g_2, g_3\}$
- Neighbor(1): $Ne(1) = \{g_4, g_5, g_6, g_7\}$

Process Steps



Three Optimizations

■ Steps and Methods

- Reposition the cells directly driven by the cells on the k most-critical paths
 - Use linear programming (LP)
- Size down the cells directly driven by the cells on the k most-critical paths
 - Use geometric programming (GP)
- Simultaneously size and place the cells on the k most-critical paths
 - Use generalized geometric programming (GGP)

Neighbor Repositioning

■ Linear programming (LP)

$$\begin{aligned}
 & \text{minimize} && t_{\text{cycle}} \\
 & \text{s.t.} && a_j \geq a_i + d_{i,j} \quad \forall (v_i, v_j) \in A \\
 & && a_j \leq t_{\text{cycle}} \quad \forall v_j \in \text{primary outputs and } v_j \in C(k) \\
 & && a_j \leq \gamma T_{\text{critical}} \quad \forall v_j \in \text{primary outputs and } v_j \notin C(k) \\
 & && a_j \geq T_{\text{start}} \quad \forall v_j \in \text{primary inputs} \\
 & && |x_i - x'_i| \leq \Delta_x \quad \forall v_i \in Ne(1) \\
 & && |y_i - y'_i| \leq \Delta_y \quad \forall v_i \in Ne(1) \\
 & && \gamma : \text{constant, } 0 \leq \gamma \leq 1 \\
 & && T_{\text{critical}} : \text{constant, longest path delay before this step}
 \end{aligned}$$

Neighbor Resizing

■ Geometric Programming

$$\begin{aligned}
 & \text{minimize } t_{\text{cycle}} \\
 \text{s.t. } & a_j \geq a_i + d_{i,j} \quad \forall (v_i, v_j) \in A \\
 & a_j \leq t_{\text{cycle}} \quad \forall v_j \in \text{primary outputs and } v_j \in C(k) \\
 & a_j \leq \lambda T_{\text{critical}} \quad \forall v_j \in \text{primary outputs and } v_j \notin C(k) \\
 & a_j \geq T_{\text{start}} \quad \forall v_j \in \text{primary inputs} \\
 & |z_i - z'_i| \leq \Delta_z \quad \forall v_i \in Ne(1) \\
 & \lambda : \text{constant, } 0 \leq \lambda \leq 1 \\
 & T_{\text{critical}} : \text{constant, longest path delay before this step}
 \end{aligned}$$

Critical Path Sizing & Place

■ Generalized Geometric Programming

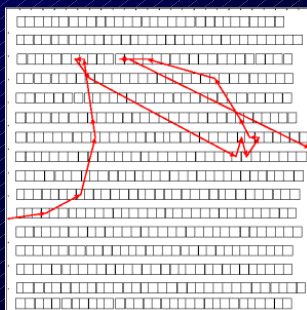
$$\begin{aligned}
 & \text{minimize } t_{\text{cycle}} \\
 \text{s.t. } & a_j \geq a_i + d_{i,j} \quad \forall (v_i, v_j) \in A \quad v_i, v_j \in C(k) \\
 & a_j \leq t_{\text{cycle}} \quad \forall v_j \in \text{primary outputs } v_i \in C(k) \\
 & a_j \geq T_{\text{start}} \quad \forall v_j \in \text{primary inputs } v_i \in C(k) \\
 & |x_i - x'_i| \leq \Delta_x \quad \forall v_i \in C(k) \\
 & |y_i - y'_i| \leq \Delta_y \quad \forall v_i \in C(k) \\
 & |z_i - z'_i| \leq \Delta_z \quad \forall v_i \in C(k)
 \end{aligned}$$

GGP-Algorithm

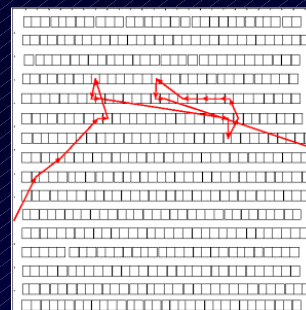
- Transform the original GGP problem into a sequence of (convex) GP problems
- The sequence of optimal solutions to the GP sequence converges to a point satisfying the Kuhn-Tucker necessary conditions for the optimality of GGP

SCD In Action (I)

- Large change allowed



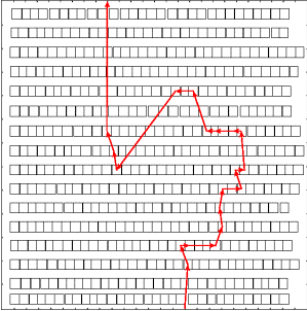
Path delay: 12.43 ns



Path delay: 12.02 ns

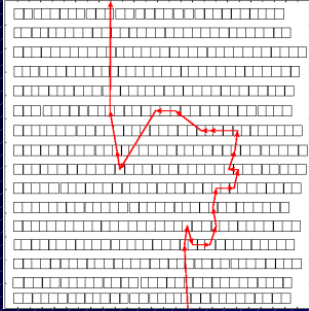
SCD In Action (II)

■ Small change allowed (1)



Path delay: 8.31 ns

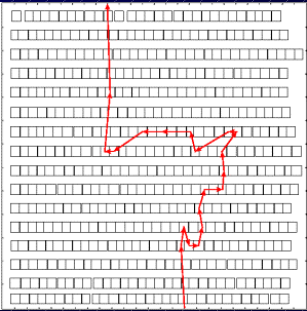
→



Path delay: 8.27 ns

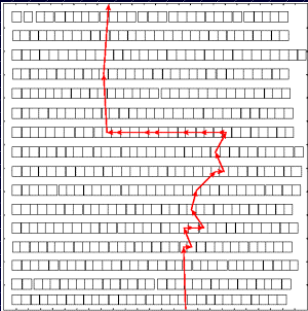
SCD In Action (III)

■ Small change allowed (2)



Path delay: 8.22 ns

→

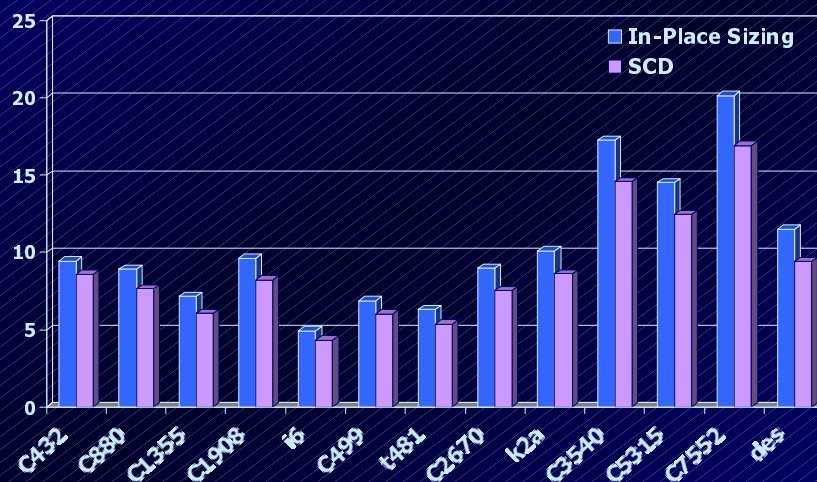


Path delay: 8.17 ns

Outline

- Motivation and Background
- FANROUT Algorithm
- Results and Discussion
- SCD Algorithm
- Results and Discussion
- Conclusions

Experimental Results



Experimental Results

■ Benchmark Circuit C499

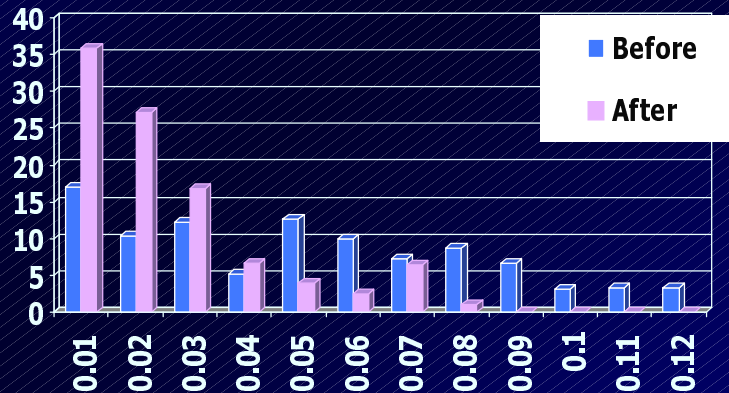
– Critical Path Delay

- Before: 13.91 ns
- In-Place Sizing: 6.89 ns
- SCD: 6.04 ns

Comparison of Slack Values

■ Normalized slack distribution (C499)

- X: ratio of the gate slack compared to the longest path delay
- Y: percentage



Outline

- Motivation and Background
- FANROUT Algorithm
- Results and Discussion
- SCD Algorithm
- Results and Discussion
- **Conclusions**

Conclusions I

- FANROUT builds buffered routing trees with maximum required time at drivers
- The resulting structure is a LT-Tree from the logical viewpoint and a P-Tree from the physical viewpoint
- Future work will focus on derivation of the initial sink order, using relaxed LT-Tree, and employing buffered P-Tree structures

Conclusions II

- SCD improves timing by balancing the path delays, i.e. longer delay paths get shorter at the expense of shorter delay paths getting longer
- 15% improvement compared to in-place gate sizing on average
- Future work will focus on combining other logic optimization techniques in the placement loop