Buffering Interconnect: From Basics to Breakthroughs

Charles Alpert
Jiang Hu
Noel Menezes
Weiping Shi
Outline

- Van Ginneken’s algorithm
- Enhancements
- Buffered Steiner tree
- Handling buffer blockages
Buffers Reduce Wire Delay

\[ t_{\text{unbuf}} = R(cx + C) + rx(cx/2 + C) \]
\[ t_{\text{buf}} = 2R(cx/2 + C) + rx(cx/4 + C) + t_b \]
\[ t_{\text{buf}} - t_{\text{unbuf}} = RC + t_b - rcx^2/4 \]
Buffers Improve Slack

\[ Slack_{\text{min}} = -50 \]

\[ Slack_{\text{min}} = 50 \]

RAT = Required Arrival time
Slack = RAT - Delay

Decouple capacitive load from critical path

### Example Cases:

- RAT = 300
  - Delay = 350
  - Slack = -50

- RAT = 700
  - Delay = 600
  - Slack = 100

- RAT = 300
  - Delay = 250
  - Slack = 50

- RAT = 700
  - Delay = 400
  - Slack = 300
Problem Formulation

- Given
  - A Steiner tree
  - RAT at each sink
  - A buffer type
  - RC parameters
  - Candidate buffer locations

- Find buffer insertion solution such that the slack_{min} is maximized
Candidate Buffering Solutions
Candidate Solution Characteristics

Each candidate solution is associated with:
- $v_i$: a node
- $c_i$: downstream capacitance
- $q_i$: RAT

$v_i$ is a sink
$c_i$ is sink capacitance

$v$ is an internal node
Van Ginneken’s Algorithm

- Start from sinks
- Candidate solutions are generated

Candidate solutions are propagated toward the source
Solution Propagation: Add Wire

\[ c_2 = c_1 + cx \]
\[ q_2 = q_1 - rcx^2/2 - rcx_1 \]

- \( r \): wire resistance per unit length
- \( c \): wire capacitance per unit length
Solution Propagation: Insert Buffer

- $c_{1b} = C_b$
- $q_{1b} = q_1 - R_b c_1 - t_b$
- $C_b$: buffer input capacitance
- $R_b$: buffer output resistance
- $t_b$: buffer intrinsic delay
Solution Propagation: Merge

\[ c_{\text{merge}} = c_l + c_r \]

\[ q_{\text{merge}} = \min(q_l, q_r) \]
Solution Propagation: Add Driver

- \( q_{0d} = q_0 - R_d c_0 = \text{Slack}_{\text{min}} \)
- \( R_d \): driver resistance
- Pick solution with max \( \text{Slack}_{\text{min}} \)
Example of Solution Propagation

- \( r = 1, c = 1 \)
- \( R_b = 1, C_b = 1, t_b = 1 \)
- \( R_d = 1 \)

1. \((v_1, 1, 20)\) - Add wire
   - \((v_2, 3, 16)\) - Insert buffer
   - \((v_3, 5, 8)\) - Add wire
     - Slack = 3 - Add driver
   - \((v_2, 1, 12)\) - Add wire
   - \((v_3, 3, 8)\) - Add wire
     - Slack = 5 - Add driver
Example of Merging

Left candidates

Right candidates

Merged candidates
Solution Pruning

- Two candidate solutions
  - $\langle v, c_1, q_1 \rangle$
  - $\langle v, c_2, q_2 \rangle$
- Solution 1 is inferior if
  - $c_1 > c_2$: larger load
  - and $q_1 < q_2$: tighter timing
They have the same load cap $C_b$, only the one with max $q$ is kept.
Basic Data Structure

Sorted list such that

- $c_1 < c_2 < c_3$
- If there is no inferior candidates $q_1 < q_2 < q_3$
Prune Solution List

Increasing $c$

$\begin{align*}
(c_1, q_1) & \rightarrow (c_2, q_2) & \rightarrow (c_3, q_3) & \rightarrow (c_4, q_4) \\
q_1 < q_2 \quad & \text{Prune 2} & q_1 < q_3 \quad & \text{Prune 3} & q_1 < q_4 \\
q_2 < q_3 \quad & \text{Prune 3} & q_2 < q_3 \quad & \text{Prune 3} & q_2 < q_4 \\
q_3 < q_4 \quad & \text{Prune 4} & q_3 < q_4 \quad & \text{Prune 4} & \text{Prune 4}
\end{align*}$
Pruning In Merging

Left candidates

- \((c_{l1}, q_{l1})\)
- \((c_{l2}, q_{l2})\)
- \((c_{l3}, q_{l3})\)

Right candidates

- \((c_{r1}, q_{r1})\)
- \((c_{r2}, q_{r2})\)
- \((c_{r3}, q_{r3})\)

Merged candidates

- \((c_{l1} + c_{r1}, q_{l1})\)
- \((c_{l2} + c_{r1}, q_{l2})\)
- \((c_{l3} + c_{r1}, q_{r1})\)
- \((c_{l3} + c_{r2}, q_{l3})\)

- \((c_{l1}, q_{l1})\)
- \((c_{l2}, q_{l2})\)
- \((c_{l3}, q_{l3})\)

- \((c_{r1}, q_{r1})\)
- \((c_{r2}, q_{r2})\)
- \((c_{r3}, q_{r3})\)

Conditions:

- \(q_{l1} < q_{l2} < q_{r1} < q_{l3} < q_{r2}\)
Van Ginneken Recap

- Generate candidates from sinks to source
- Quadratic runtime
  - Adding a wire does not change #candidates
  - Adding a buffer adds only one new candidate
  - Merging branches additive, not multiplicative
  - Linear time solution list pruning
- Optimal for Elmore delay model
Outline

- Van Ginneken’s algorithm
- **Enhancements**
  - Wire segmenting
  - Buffer library
  - Inverter/ polarity
  - Slew/ cap constraints
  - Cost-slack tradeoff
  - Wire sizing
  - Driver sizing
- Buffered Steiner tree
- Handling buffer blockages
Wire Segmenting

Faster runtime

Better solution quality
Multiple Buffer Types

- \( r = 1, c = 1 \)
- \( R_b = 1, C_b = 1, t_b = 1 \)
- \( R_{b2} = 0.5, C_{b2} = 2, t_{b2} = 0.5 \)
- \( R_d = 1 \)
Using Inverters

Less cost
Handle Polarity

Positive

Negative
Slew Constraints

- When a buffer is inserted, assume ideal slew rate at its input
- Check slew rate at downstream buffers/sinks
- If slew is too large, candidate is discarded
Capacitance Constraints

- Each gate $g$ drives at most $C(g)$ capacitance.
- When inserting buffer $g$, check downstream capacitance.
- If $> C(g)$, throw out candidate.

Total cap = 500 ff
Consider Cost/Power

- A solution is also characterized by cost $w$
- A solution is inferior if it is poor on all of $c$, $q$ and $w$
- At source, a set of solutions with tradeoff of $q$ and $w$
- $w$ can be
  - total capacitance
  - or the number of buffers
Cost-Slack Trade-off

Slack (ps)

# of Buffers
Data Organization

Sorted in ascending order of \((c, q)\)

0: \((c_1, q_1)\) → \((c_2, q_2)\) → \((c_3, q_3)\)
1: \((c_4, q_4)\) → \((c_5, q_5)\) → \((c_6, q_6)\)
2: \((c_7, q_7)\) → \((c_8, q_8)\)
3: \((c_9, q_9)\) → \((c_{10}, q_{10})\)
4: \((c_{11}, q_{11})\)

#buffers inserted
Continuous Wire Sizing

Min delay wire shape: \( w(x) = a(e^{-bx}) \)
Two Types of Wire Sizing

- Uniform Wire Sizing (UWS)
- Wire Tapering (TWS)
TWS versus UWS

\[
\frac{Signal\_Velocity(TWS)}{Signal\_Velocity(UWS)} = \frac{2 + \sqrt{2}}{2\sqrt{e}} \approx 1.0354
\]
Why Uniform Wire Sizing?

- Empirically, UWS almost as good as TWS
- Tapering info hard to give to router
- Better congestion and space management
- Extraction, detailed routing, verification?
- Estimated Steiner anyway
- Can do it simultaneously with buffering
Wire Sizing in a Tree

- Monotone property: ancestor edges cannot be narrower than downstream edges
Optimal Wire Sizing (OWS)

- Maximum width solution
  - Each edge starts with max width
  - Greedy iterative improvement
- Minimum width solution
  - Each edge starts with min width
  - Greedy iterative improvement
- Enumerate possibilities between min and max width solutions
Simultaneous Buffer Insertion and Wire Sizing
Driver Sizing
Combine Buffering and Driver Sizing Directly?

Min delay
Impact To Previous Stage

Previous stage

Small load

Large load

Current stage

Large delay

Small delay

$C_d$
Input Load Penalty

- Penalty = delay of min delay buffer chain driving $C_d$
Penalty Computation

- For a min delay buffer chain \( B_1, B_2, \ldots, B_k, C_{B1} < C_{B2} < \ldots < C_{Bk} \)
- To drive capacitance \( C_{Bi} \), combine optimal chain driving \( C_{Bj} \) with buffer \( B_j \)
- \( D(C_{B1}) = 0 \)
- \( D(C_{Bi}) = \min_{0 < j < i} \{D(C_{Bi}) + \text{Delay}(B_j, C_{Bi})\} \)
- \( D(C_d) = \min_{1 < i < k} \{D(C_{Bi}) + \text{Delay}(B_i, C_d)\} \)
Driver Sizing Considering Impact to Previous Stage

Previous stage

Small load

Current stage

Large delay

Large penalty

Large load

$C_d$

Small delay
Driver Sizing in Van Ginneken’s Algorithm

Treat the buffer chain as a part of the net

Run van Ginneken’s algorithm with fixed driver and min sized buffer
Outline

- Van Ginneken’s algorithm
- Enhancements
- **Buffered Steiner tree**
- Handling buffer blockages
Dependence on Steiner Tree

Timing critical

Timing critical
Timing Driven Steiner Trees

- Area-radius trade-off, Prim-Dijkstra trade-off
- Rectilinear Steiner Arborescence (RSA), buffered A-Tree
- P-Tree, buffered P-Tree
- C-Tree
- S-Tree
Area or Radius?

• Prim’s minimum spanning tree
• Small total wire length
• Long path to sinks

• Dijkstra’s shortest path tree
• Short path to sinks
• Large total wire length
Area Radius Trade-off

- Find a solution in middle
  - Not too much area
  - Not too long radius
- How to find an ideal point?
Prim’s and Dijkstra’s Algorithms

- $d(i,j)$: length of edge $(i, j)$
- $p(i)$: length of path from source to $i$
- Prim: min $d(i,j)$   Dijkstra: min $d(i,j) + p(j)$
The Prim-Dijkstra Trade-off

- Prim: add edge minimizing $d(i,j)$
- Dijkstra: add edge minimizing $p(i) + d(i,j)$
- Trade-off: $c \cdot p(i) + d(i,j)$ for $0 \leq c \leq 1$
- When $c=0$, trade-off = Prim
- When $c=1$, trade-off = Dijkstra
Spanning Tree $\rightarrow$ Steiner Tree
Rectilinear Steiner Arborescence (RSA)

- Every source-sink path is the shortest
- Minimum total wire length
RSA Heuristic

- Assume all sinks in first quadrant
- Initially, each sink is a subtree
- Iteratively merge or grow subtrees toward the source
RSA Diagonal Line Sweep
Iteratively

- Find subtrees rooted at \( p \) and \( q \) maximizing \( \min(x_p, x_q) + \min(y_p, y_q) \)
- Merge them to a new subtree rooted at \( r = (\min(x_p, x_q), \min(y_p, y_q)) \)
RSA Example

Merge

Grow
Buffered A-Tree
P-Tree: Abstract Tree

Obtain sink permutation

Optimal tree $T_{opt}\{a, b, c, d\}$

$T_{opt}\{a\} + T_{opt}\{b, c, d\}$

$T_{opt}\{a, b\} + T_{opt}\{c, d\}$

Dynamic programming $T_{opt}\{a, b, c\} + T_{opt}\{d\}$

Traveling salesman
P-Tree: Embedding

Hanan grid
Buffered P-Tree

Optimal tree $T_{opt}\{a,b,c,d\}$

$T_{opt}\{a\}+T_{opt}\{b,c,d\}$

$T_{opt}\{a\}+T_{buf\_opt}\{b,c,d\}$

$T_{buf\_opt}\{a\}+T_{opt}\{b,c,d\}$

$T_{opt}\{a,b\}+T_{opt}\{c,d\}$

$T_{buf\_opt}\{a,b\}+T_{opt}\{c,d\}$

$T_{opt}\{a,b,c\}+T_{opt}\{d\}$

$T_{buf\_opt}\{a,b,c\}+T_{opt}\{d\}$

$T_{buf\_opt}\{a,b\}+T_{buf\_opt}\{c,d\}$

$T_{buf\_opt}\{a,b\}+T_{buf\_opt}\{c,d\}$
Sink Criticality

Timing critical

Timing critical
Polarity

Diagram of polarity with positive and negative symbols arranged in a sequence.
Clustering
Clustering Distance Metric

- $\text{pDist}(i,j) = |\text{polarity}(i) - \text{polarity}(j)|$
- $\text{sDist}(i,j) = (|x_i - x_j| + |y_i - y_j|)/\text{diam}$
- $\text{tDist}(i,j)$ scaled between 0 and 1, 0 for equal criticalities, 1 for opposite criticalities

Final distance metric $d(i,j) = \text{pDist}(i,j) + \beta\text{sDist}(i,j) + (1-\beta)\text{tDist}(i,j)$
Clustering – Finding Centers
Clustering – Group to Centers
S-Tree

Given Sink partition \{a,c\}, \{b,d\}

Given topology

Promote new topology according to partitioning
Outline

- Van Ginneken’s algorithm
- Enhancements
- Buffered Steiner tree
- Handling buffer blockages
Buffer Blockage

- Macro, IP core, cache ...
- Wires are allowed
- Buffers are not allowed
Wire Detour?
Rerouting to Avoid Blockages

Reroute each 2-path overlapping with blockages
Reroute a 2-path

Grid graph

Targets

- - - -

Edge cost = edge_len

- - - -

Edge cost = A \cdot edge\_len
A > 1

• Find the shortest path from the source to a target

• Careful selection of value A
Buffered Path

- For 2-pin nets
- Simultaneous routing and buffer insertion
- Consider buffer blockages
- Candidate solution
  - Characterized by \((c, q)\)
  - Generated from the sink
  - Propagated over the grid graph
Solution Propagation

Candidate solutions are pruned at each node
Graph Based Buffered Path

- **Observation**: a buffer decouples path delay into additive parts
- One buffer at each node except source and sink
- No buffer in middle of an edge
- Edge weight = edge delay
- Obtain min delay path via Dijkstra’s algorithm
Example of Graph Based Buffered Path
Multi-pin Nets

- Simultaneous Steiner tree construction and buffer insertion
- Ex. Buffered P-Tree
- Label nodes in blockages
- Candidate solutions
  - Generated at sinks
  - Propagated and merged at nodes over the grid
Adaptive Tree Adjustment

- Obtain an initial timing driven Steiner tree
- Adjust tree topology only if necessary
- Buffer insertion
Tree Adjustment Procedure

- Candidate solutions are propagated along given tree
- Alternative Steiner node is considered if blockage is met
- Candidate tree adjustments are also propagated
Buffer Insertion References


Steiner Tree References

- Prim-Dijkstra tradeoffs for improved performance-driven routing tree design

- Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to VLSI physical design
  Cong, J.; Kahng, A.B.; Kwok-Shing Leung

- Interconnect layout optimization by simultaneous Steiner tree construction and buffer insertion
  Okamotot, T.; Cong, J.; ACM Physical Design Workshop, April 1996, Page(s): 1-6

- Simultaneous routing and buffer insertion for high performance interconnect
  Lillis, J.; Chung-Kuan Cheng; Ting-Ting Y. Lin

- Buffered steiner trees for difficult instances
  International Symposium on Physical Design, April 2001 Page(s): 4-9

- Buffer tree synthesis with consideration of temporal locality, sink polarity requirements, solution cost, congestion and blockages
  Hrkic, M.; Lillis, J.;
Blockage Avoidance

References