Fast Buffering for Optimizing Worst Slack and Resource Consumption in Repeater Trees

Christoph Bartoschek, Stephan Held, Dieter Rautenbach, Jens Vygen

Research Institute for Discrete Mathematics
University of Bonn

March 30, 2009

Outline

- Problem Description
- Preparation
- Buffering Algorithm
- Results

The Repeater Tree Problem

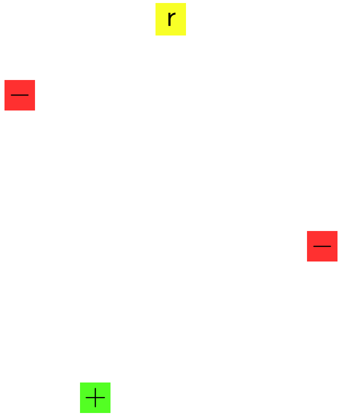
A repeater tree instance consists of

- a root r with position $Pl(r)$ and estimated arrival time AT_r

The Repeater Tree Problem

A repeater tree instance consists of

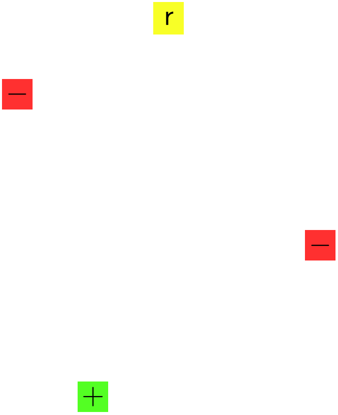
- a root r with position $Pl(r)$ and estimated arrival time AT_r
- a set S of sinks and for each sink s its polarity $+$ or $-$, the location $Pl(s)$ the input capacitance $icap(s)$ and estimated RAT_s



The Repeater Tree Problem

A repeater tree instance consists of

- a root r with position $Pl(r)$ and estimated arrival time AT_r
- a set S of sinks and for each sink s its polarity $+$ or $-$, the location $Pl(s)$ the input capacitance $icap(s)$ and estimated RAT_s
- a library L of repeaters and for each repeater t the timing rules and input capacitance $icap(t)$
- physical information



The Repeater Tree Problem

A feasible repeater tree

- connects the root to the sinks using wires and placed repeaters from the library such that
- the signal arrives at the sinks with correct parity and
- capacitance and slew limits are obeyed

Objectives

- maximinze the worst slack
- reduce power consumption

Topology Generation and Buffering

A common simplification of the problem is to divide the problem into two steps:

- topology generation
- buffering along a given topology

In this talk we only consider buffering and therefore the topology is part of the input

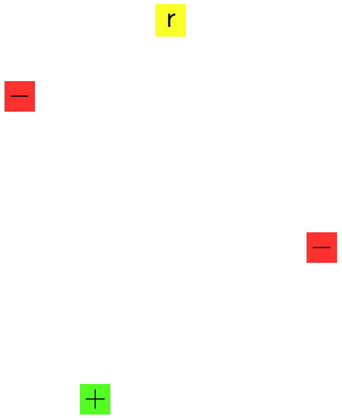
Previous Work

An overview of buffering algorithms can be found in the *Handbook of Algorithms for Physical Design Automation* [AMS08].

The Repeater Tree Problem

A repeater tree instance consists of

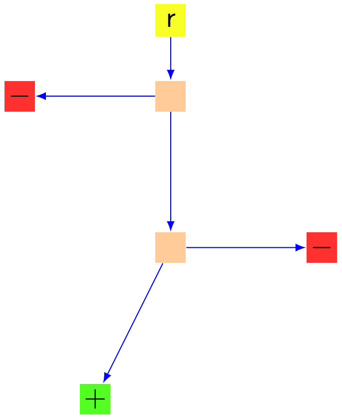
- a root r with position $Pl(r)$ and estimated arrival time AT_r
- a set S of sinks and for each sink s its polarity $+$ or $-$, the location $Pl(s)$ the input capacitance $icap(s)$ and estimated RAT_s
- a library L of repeaters and for each repeater t the timing rules and input capacitance $icap(t)$
- physical information



The Repeater Tree Problem

A repeater tree instance consists of

- a root r with position $Pl(r)$ and estimated arrival time AT_r
- a set S of sinks and for each sink s its polarity $+$ or $-$, the location $Pl(s)$ the input capacitance $icap(s)$ and estimated RAT_s
- a library L of repeaters and for each repeater t the timing rules and input capacitance $icap(t)$
- physical information
- a topology



Preparation – Topologies

A topology T is a directed tree rooted at r with $\delta^+(r) = 1$ and $\delta^+(u) \in \{1, 2\}$ for all internal nodes u.

The set of leaves is a subset of S.

$Pl(u)$ gives the position of each internal node u.

Preparation – Sources of Topologies

Topologies can be generated by several algorithms. (See also [AMS08])

- Fast Topology Generation [BHRV06]
- C-Tree [AHHKLLQSS02]
- Global Routing

Preparation – Delay Model

The algorithm depends on a delay model that is able to calculate slacks for a given topology.

Used Delay Model [BHRV06]

The delay from r to a sink s in a given topology is modeled as:

$$\begin{aligned} AT(s) = & AT_r + rootdelay + sinkdelay(s) + \\ & \sum_{(u,v) \in E(T_{[r,s]})} d_{node} + d_{wire} \cdot dist(Pl(u), Pl(v)) \end{aligned}$$

- d_{node}: Delay penalty for bifurcation
- d_{wire}: Delay per unit length

Other delay models:

BELT [AHSS04]

Preparation – Slew Degradation Factor ν

The required arrival time at a sink depends on the slew that arrives at the sink.
We approximate this function linearly:

$$RAT_s(slew) = RAT_s + \nu(slew - target\_slew) \tag{1}$$

Preparation – Repeater Selection by inv(load, slew)

The function inv gives us the smallest inverter that can drive the given *load* and achieves the given *slew* if a target_slew arrives at its input pin.

Algorithm – Basic Properties

The basic properties of the algorithm are:

- It works bottom-up.
- It is driven by capacitance limits.
- It changes the topology.

Algorithm

1. While there is a leaf in the topology:
    1.1 Choose a leaf x
    1.2 If $PI(x) = PI(parent(x))$ then $Merge(x)$
    1.3 else $Move(x)$
    1.4 Update slacks in the tree

Algorithm – Limits

According to the parameter ξ the following limits are choosen:

- A load limit for any driver *maxcap*
- A wire load limit for any net *maxwcap*
- A slew target

Algorithm – Clusters

Each node of the topology is associated with a pair of clusters (one for each parity).

Cluster

A *cluster* C is a triple $(S(C), W(C), M(C))$ where

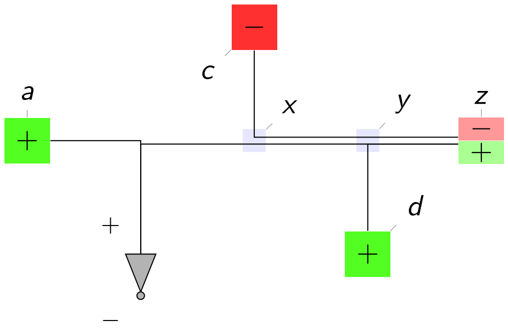
- $S(C)$ is a set of sinks/repeaters below the cluster.
- $W(C)$ is a capacitance estimate for the wires between the cluster and its sinks.
- $M(C) \in \mathbb{R}^2$ is the merge point of the cluster.

Additional properties

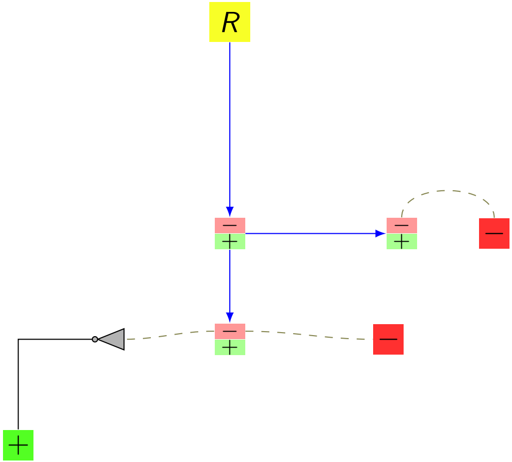
- $Pl(C)$ is the position of the cluster.
- $Cap(C)$ is the total capacitance behind the cluster.

$$Cap(C) = W(C) + \sum_{s \in S(C)} icap(s) \tag{2}$$

Algorithm – Cluster Example



Algorithm – Invariant of the Tree



Algorithm – Timing in the Tree

Timing in the three parts of the tree is computed.

- For the realized subtrees we have a RAT for the slew_target.
- For the topology we use the delay model.
- A RAT can be computed for the clusters by using the topology information.

Algorithm – Moving Clusters

The *Move* operation moves a pair of clusters towards their parent as far as the capacitance limits allow. If the move does not reach the parent then a repeater is inserted by using *inv*.

Algorithm – Merging Clusters

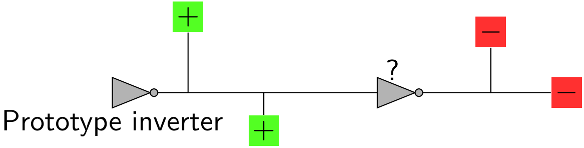
The *Merge* operation merges a cluster pair of a node with the clusters of the parent node.

One of the following options is choosen:

- Merging without adding a repeater
- Inserting a repeater in front of one of the clusters

Algorithm – Inserting a Repeater during Merging

Repeaters are inserted in front of the sinks of a cluster at the merge point of the cluster or at its position.
The new repeater is added as a sink into a given cluster.



Algorithm – Merge Evaluation

Each feasible option gives a slack and a power value.

- The slack can be computed by the RATs of the clusters and the delay in the delay model.
- We use the slack at the root node for evaluation.
- The power value are the costs associated with a repeater.

We choose the candidate that maximizes:

$$\xi \mathit{slack} - (1 - \xi)\mathit{power} \tag{3}$$

Running Time

The running time of the algorithm is in $O(|S||L| + k \log |L|)$ where $k \in O(\frac{l}{l^*})$ and l^* is the length of a wire with capacitance *maxwcap*.

Example – Step 1

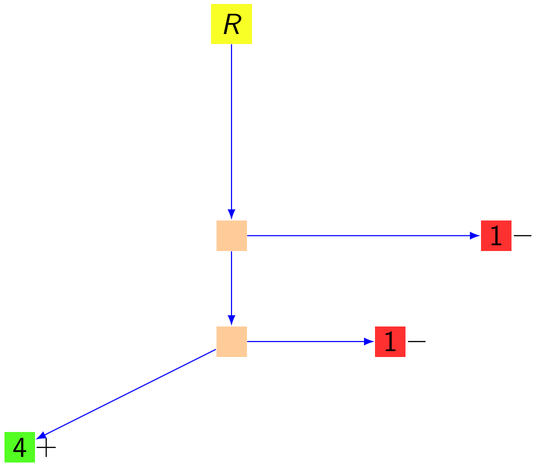


Figure: The input topology

Example – Step 2

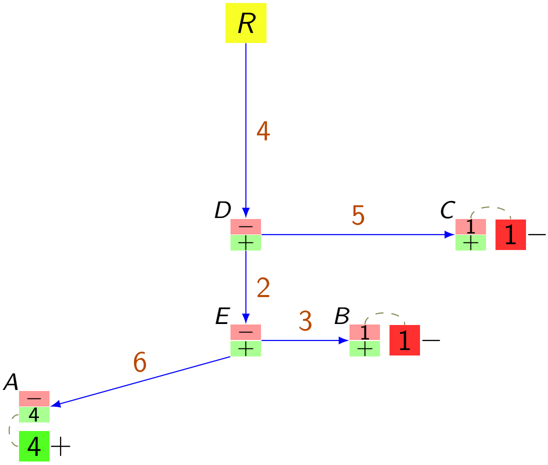


Figure: The preliminary topology after initializing the cluster pairs. Dashed lines indicate which sinks belong to a cluster

Example – Step 3

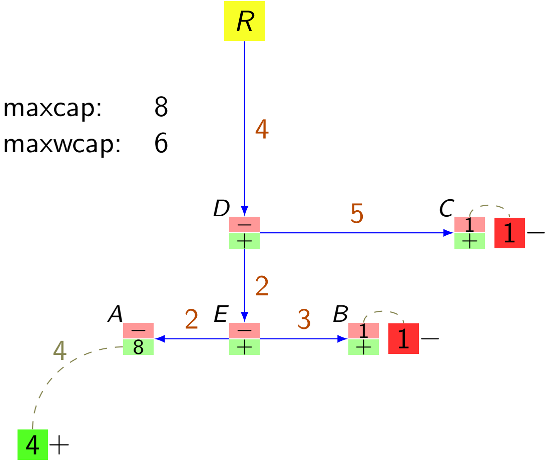


Figure: The preliminary topology after moving A along (E, A)

Example – Step 4

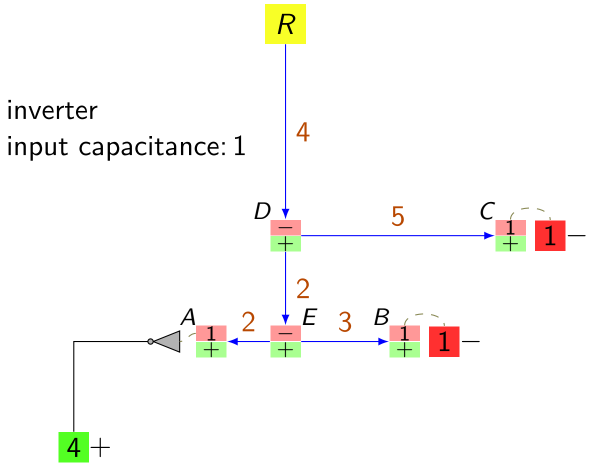


Figure: Creating the first inverter and updating the clusters

Example – Step 5

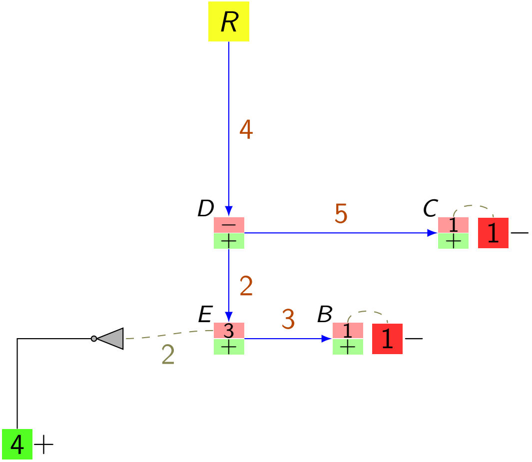


Figure: Moving A and merging A with E

Example – Step 6

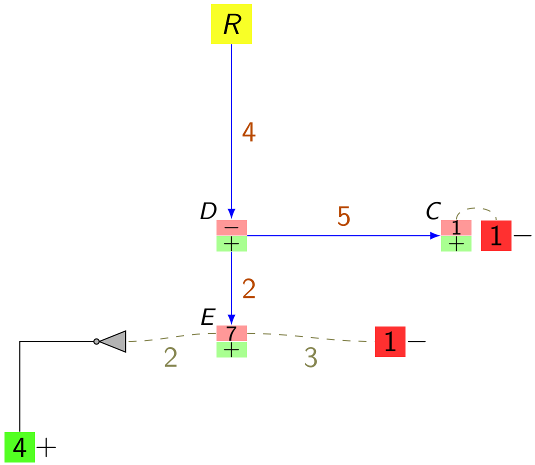


Figure: Moving B and merging B with E

Example – Step 7

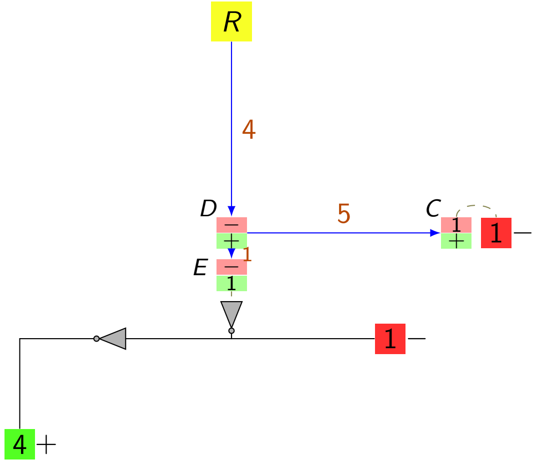


Figure: Moving E and creating the second inverter

Example – Step 8

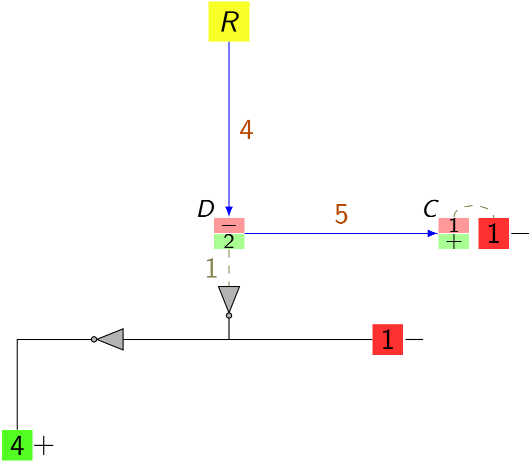


Figure: Moving E further and merging E and D

Example – Step 9

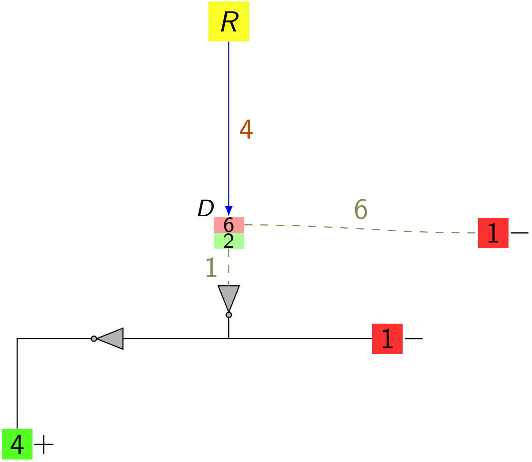


Figure: Moving C and merging C and D

Example – Step 10

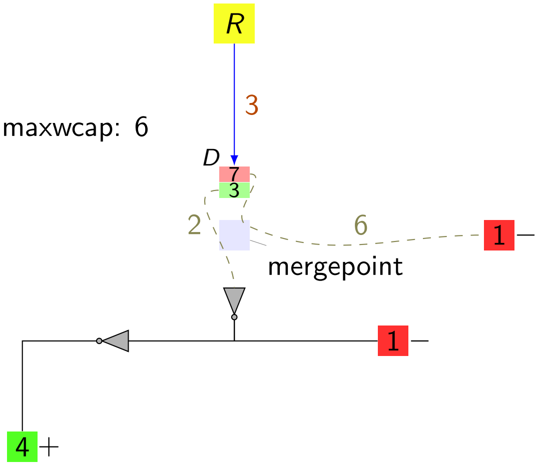


Figure: Moving D one unit along (R, D)

Example – Step 11

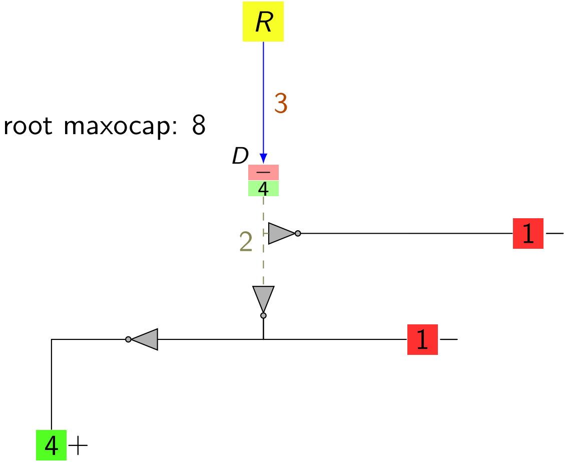


Figure: Creation of the third inverter and updating the clusters

Example – Step 12

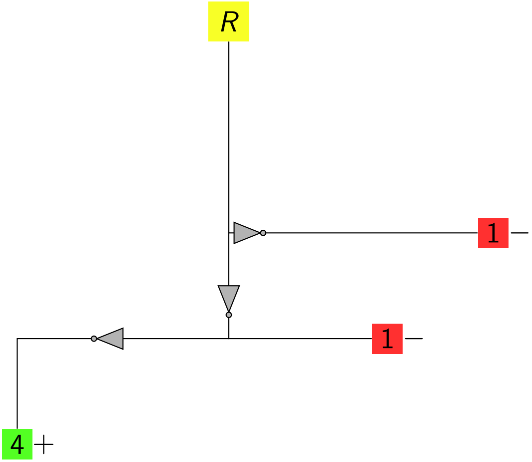


Figure: The final repeater tree

Experimental Results

- 2.1 million instances were taken a current 65 nm design and a 45 nm design.
- All experiments were done on Intel Xeon E7220 processors with 2.93 GHz.

Results - Instances

Optimization Goal:		**Power ($\xi = 0$)**	**Slack ($\xi = 1$)**
Number of Sinks	Number of Instances	Runtime (in seconds)	Runtime (in seconds)
1	1 401 791	149	152
2	337 610	111	119
3	137 825	82	82
4	102 775	90	91
5	49 999	59	58
6	21 375	34	31
7	20 317	39	39
8	17 472	40	38
9	10 549	28	28
10	6 350	19	20
11–20	30 905	152	151
21–30	5 828	56	52
31–50	7 958	124	108
51–100	6 474	173	146
101–200	1 870	106	87
201–500	549	108	74
501–1000	143	59	36
> 1000	6	0	0
Total	2 159 796	1 438	1321

Results - Used Repeaters

Opt. Goal	**Lower bound**	**Power** ($\xi = 0$)	**Slack** ($\xi = 1$)
# sinks	# inverters	# inverters	# inverters
1	116 127	132 721	503 315
2	81 245	111 095	710 066
3	69 531	90 557	355 090
4	68 334	97 094	320 590
5	33 423	47 688	181 005
6	16 057	23 158	74 243
7	20 268	30 514	92 980
8	14 678	19 508	67 406
9	11 318	16 425	47 574
10	7 350	11 162	29 530
11–20	39 312	62 044	188 586
21–30	8 110	14 964	46 982
31–50	12 365	21 354	75 093
51–100	16 697	30 235	123 686
101–200	10 831	19 137	65 968
201–500	4 960	9 654	33 532
501–1000	1 476	2 832	11 677
> 1000	323	579	1 423
Total	532 405	740 721	2 928 746

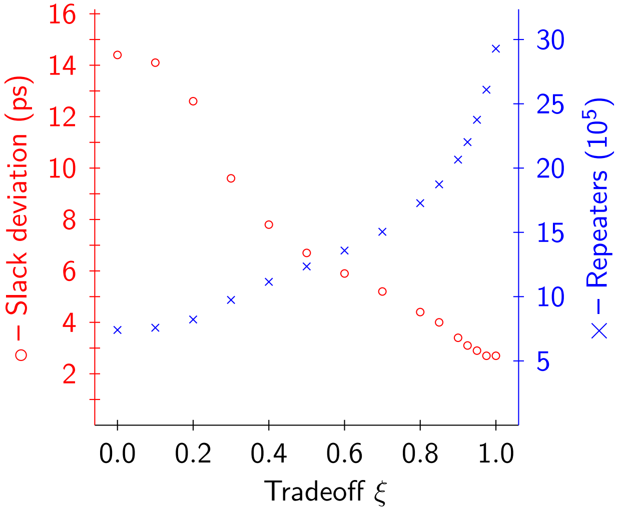
Results - Slack Deviation

Opt. Goal	**Power** ($\xi = 0$)		**Slack** ($\xi = 1$)	
# Sinks	Average	Maximum	Average	Maximum
1	4.9	465.7	0.4	89.7
2	17.1	431.0	2.0	101.5
3	27.8	465.5	4.8	271.5
4	34.8	375.3	7.3	116.9
5	52.6	598.3	10.5	106.3
6	49.3	277.7	11.3	112.6
7	55.9	506.2	12.6	155.8
8	73.2	777.7	14.9	139.0
9	58.0	1594.3	17.1	147.2
10	62.0	565.2	14.4	105.7
11–20	58.1	2236.8	16.7	135.3
21–30	68.1	703.4	24.8	179.4
31–50	86.1	1999.6	53.4	264.9
51–100	106.3	2245.1	71.1	230.7
101–200	77.2	518.6	31.6	240.3
201–500	178.1	1140.2	61.3	274.0
501–1000	514.7	1021.2	172.2	422.1
> 1000	198.5	304.4	89.2	118.9
Total	14.4	2245.1	2.7	422.1

Results - Comparison To Dynamic Programming

# Sinks	Dynamic Programming			New Buffering ($\xi = 1$)		
	Slack Deviation		Runtime	Slack Deviation		Runtime
	Avg	Max	(in s)	Avg	Max	(in s)
1	0.1	39.3	4118	0.4	89.7	152
2	1.2	121.1	2555	2.0	101.5	119
3	2.3	166.3	1538	4.8	271.5	82
4	4.5	93.6	1620	7.3	116.9	91
5	4.3	119.5	1100	10.5	106.3	58
6	6.1	115.4	481	11.3	112.6	31
7	5.7	102.0	650	12.6	155.8	39
8	9.7	100.7	589	14.9	139.0	38
9	9.5	106.2	408	17.1	147.2	28
10	7.2	118.7	255	14.4	105.7	20
11–20	9.2	168.5	1891	16.7	135.3	151
21–30	17.2	167.2	610	24.8	179.4	52
31–50	43.8	227.9	1436	53.4	264.9	108
51–100	44.8	163.3	2114	71.1	230.7	146
101–200	16.1	145.0	1105	31.6	240.3	87
201–500	29.0	152.6	752	61.3	274.0	74
501–1000	87.1	229.3	425	172.2	422.1	36
> 1000	21.5	54.3	65	89.2	118.9	0
Total	1.5	229.3	21711	2.7	422.1	1321

Results - Parameter ξ



References

- C. J. Alpert, D. P. Mehta, S. S. Sapatnekar (editors), "Handbook of Algorithms for Physical Design Automation", Auerbach Publishers Inc. (2008)
- C. J. Alpert, M. Hrkic, J. Hu, A. B. Kahng, J. Lillis, B. Liu, S. T. Quay, S. S. Sapatnekar and A. J. Sullivan, "Buffered Steiner trees for difficult instances", IEEE Transactions on Computer-Aided Design 21, pages 3–14, 2002
- C. J. Alpert, J. Hu, S. S. Sapatnekar, and C. N. Sze, "Accurate Estimation of Global Buffer Delay within a Floorplan". In Proc. of ICCAD, pages 706–711, 2004.
- C. Bartoschek, S. Held, D. Rautenbach, and J. Vygen, "Efficient generation of short and fast repeater tree topologies", Proceedings of the International Symposium on Physical Design (2006), 120–127

Thank you