# Power Optimal Dual-Vdd Buffered Tree Considering Buffer Stations and Blockages


## ABSTRACT

This paper presents the first in-depth study on applying dual $V_{dd}$ buffers to buffer insertion and multi-sink buffered tree construction for power minimization under delay constraint. To tackle the problem of dramatic complexity increment due to simultaneous delay and power consideration and increased buffer choices, we develop a sampling-based sub-solutions (i.e. options) propagation method and a balanced search tree based data structure for option pruning. We obtain 17x speedup with little loss of optimality compared to the exact option propagation. When dual $V_{dd}$ buffers are considered, our algorithm reduces power by 23% at the minimum delay specification compared to single $V_{dd}$ buffer insertion. Moreover, compared to the delay-optimal tree using single $V_{dd}$ buffers, our power-optimal buffered tree reduces power by 7% and 18% at the minimum delay specification when single $V_{dd}$ and dual $V_{dd}$ buffers are used respectively.


## 1. INTRODUCTION

Aggressive scaling of VLSI circuits makes interconnects the performance bottleneck, and buffer insertion is used extensively to reduce interconnect delay at the expense of more power dissipation. [1] develops a power-optimal buffer insertion algorithm to meet the delay specification. In addition, the buffered tree construction problem for multi-sink nets has also been studied in [2, 3], without considering *buffer stations* (*BS*) or blockages. [4, 5] present two construction approaches to account for blockage avoidance and *BS* and quickly explore a few alternative routes for the purpose of delay minimization. [6] presents an optimal delay routing algorithm that also considers *BS* and blockages while [7] enhanced it with several speed-up techniques. All the buffered tree construction methods do not consider power explicitly.

Recently, $V_{dd}$-promgrammble buffers have been used to reduce FPGA interconnect power [8]. As buffers are preplaced, the dual $V_{dd}$ buffer routing is simplified to dual $V_{dd}$ assignment. However, dual-Vdd buffer insertion and buffered tree construction for ASIC designs is more complicated and have not been studied. We present the first in-depth study on applying dual $V_{dd}$ buffers to buffer insertion (*DVB* ) and multi-sink buffered tree generation (*D-Tree* ) considering both *BS* and blockages for power minimization under delay constraint. We develop a sampling-based sub-solutions (i.e. options) propagation method and a balanced search tree based data structure for option pruning to reduce computational complexity. We obtain a 17x speedup with little loss of optimality over [1] which has exact option propagation. Experimental results show that our *DVB* algorithm reduces power by 23% over [1] at the minimum delay specification. Moreover, our *D-Tree* algorithm reduces power by 7% and 18% at the minimum delay specification over [6, 7] when single $V_{dd}$ and dual $V_{dd}$ buffers are used respectively.

Section 2 discusses the dual $V_{dd}$ buffer modeling, the *DVB* and the *D-Tree* problem formulations. Section 3 and 4 give the details of the algorithms for solving the *DVB* and the *D-Tree* problems and their respective experimental results. We conclude the paper in Section 5.

## 2. MODELING AND PROBLEM FORMULATION

### 2.1 Delay, Slew Rate and Power Model

We use distributed Elmore delay model as in [6, 4, 7, 5]. The delay due to a piece of wire of length $l$ is given by

$$d(l) = \left(\frac{1}{2} \cdot c_w \cdot l + c_{load}\right) \cdot r_w \cdot l \tag{1}$$

where $c_w$ and $r_w$ are the unit length capacitance and resistance of the interconnect and $c_{load}$ is the capacitive loading at the end of the wire. We also use Elmore delay times $\ln 9$ as the slew rate metric [9]. The delay of a buffer is given by

$$d_{buf} = d_{int} + r_o \cdot c_{load} \tag{2}$$

where $d_{int}$, $r_o$ and $c_{load}$ are the intrinsic delay, output resistance and capacitive loading at the output of the buffer respectively. We obtain $r_o$ and $d_{int}$ for both high $V_{dd}$ and low $V_{dd}$ buffers, and we observe that both values are higher for low $V_{dd}$ buffers.

In the context of buffer insertion with upper slew rate constraint, we observe that slew rates at the buffer inputs and the sinks are always within up to a few tens $ps$ of the upper bound. Therefore we model buffer delay with negligible error by assuming input slew rate close to the upper bound To illustrate, we consider a rather loose slew rate bound $\hat{s} = 100ps$, which is 20% of the clock period for a 2Ghz clock. Using the formula for delay optimal buffer insertion length $l_{opt}$ [10] for a single interconnect wire and the information of the $65nm$ technology node [11] that is summarized in Table 1, we get $l_{opt} = 3924\mu m$. On the other hand, from Equation (1) and assuming $c_{load} = 0$ and infinite size driver at the input of the wire, we have $l_{slew} = 3073\mu m$, which is the largest length of an unbuffered interconnect without slew rate violation and is much smaller than $l_{opt}$. Considering power optimality increases the optimal insertion length

$l_{opt}$ [10], while considering non-zero wire loading $c_{load}$, additional delay due to finite driver strength and a tighter slew rate bound further decreases $l_{slew}$. Therefore we tend to insert buffers in order to the satisfy the slew rate bound, which results in close-to-bound slew rates.

| Settings | Values |
|---|---|
| Simulators | QuickCap [12] (interconnect)<br>BSIM 4 + SPICE [13] (device) |
| Interconnect | $r_w = 0.186\Omega/\mu m$, $c_w = 0.0519fF/\mu m$<br>(65nm, global, min space and width) |
| Buffer<br>(min size) | $c_{in} = 0.47fF$,<br>$V_{dd} = 1.2V$: $r_o = 4732\Omega$, $d_{int} = 72.0ps$<br>$V_{dd} = 0.9V$: $r_o = 5364\Omega$, $d_{int} = 97.7ps$ |

**Table 1: Settings for the 65nm global interconnect.**

Note that more accurate slew rate and delay models that support bottom up (i.e. sink-to-source) calculation such as [14] can be used instead. The use of simpler delay and slew metrics here is for the ease of implementation and explanation only.

We measure interconnect power by energy per switch. The energy per switch for an interconnect wire of length $l$ is

$$E_w = 0.5 \cdot c_w \cdot l \cdot V_{dd}^2 \quad (3)$$

We collapse per switch short-circuit and dynamic power consumed by a buffer into a single value $E_{buf}$, which is a function of both $V_{dd}$ and buffer size. We observe that low $V_{dd}$ buffers have a much smaller $E_{buf}$ than the same-sized high $V_{dd}$ counterparts. On the other hand, since the leakage component of the total energy consumption depends very much on the operating frequency and the switching activity, we choose not to include leakage power in our study.

## 2.2 Dual $V_{dd}$ Technique

Dual $V_{dd}$ buffering uses both high $V_{dd}$ and low $V_{dd}$ buffers in interconnect synthesis. Designs using low $V_{dd}$ buffers consume less buffer $E_{buf}$ and interconnect (Equation 3) power. Applying this technique to non-critical paths, we achieve power saving without worsening the delay of the overall interconnect tree.

We only allow high $V_{dd}$ buffers followed by low $V_{dd}$ buffers but not the reverse except at right before the sinks. A high $V_{dd}$ buffer can drive a low $V_{dd}$ buffer, but a low $V_{dd}$ buffer driving a high $V_{dd}$ one may cause a large leakage power. Therefore, a $V_{dd}$-level converter must be inserted between the low $V_{dd}$ buffer and its high $V_{dd}$ fanout buffers. We assume that the driver at the source operates at high $V_{dd}$ and $V_{dd}$-level converters are placed *only* at the sinks driven by low $V_{dd}$ buffers. We do not consider $V_{dd}$-level converters explicitly in our algorithm, which reduces power and delay overhead that is introduced by using a large number of $V_{dd}$-level converters. Considering a simple case in Figure 1, the configuration in (a) must have a larger power than that in (b) due to the the level converter and the fact that the low $V_{dd}$ buffer instead of the high $V_{dd}$ buffer is driving the load $C_l$. To have the delay of case (b) larger than that of (a), we make

$$(R_b^L - R_b^H) \cdot C_l + R_b^H \cdot C_b^L - R_b^L \cdot C_{LC} - R_{LC} \cdot C_b^H - d_{LC} \geq 0 \quad (4)$$

where $d_{LC}$ is the intrinsic delay of the level converter and all other parameters are shown in Figure 1. By trying all combinations of buffer sizes (16x, 32x, 64x in our study) and by using the parasitic values for a properly sized level converter, we have $C_l$ to be at least $0.5pF$, or equivalently a $9mm$ long global interconnect worth of capacitance, for Equation (4) to become true, which is extremely unlikely in any buffered interconnect design. Therefore (b), which has no level converter, is very likely to be a superior design than (a). This provides us with the insight about excluding level converters in our study, which saves runtime by considering a smaller and more productive solution space.

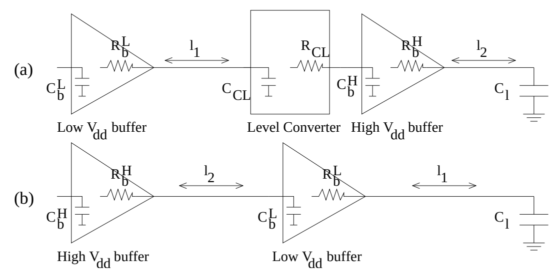


**Figure 1: Demonstrating level converter overhead.**

## 2.3 Dual $V_{dd}$ Buffer Insertion Problem

We assume that the loading capacitance and the required arrival times ($RAT$) $q_n^s$ are given at all sink terminals $n_s$. We assume that the driver resistance at the source node $n_{src}$ is given. We also assume that all types of buffers can be placed only at the buffer candidate nodes $n_b^k$. We use the $RAT$ at the source $n_{src}$ to measure delay performance. Our goal is to minimize power of the interconnect subject to the $RAT$ constraint at the source $n_{src}$.

*Definition 1.* The required arrival time ($RAT$) $q_n$ at node $n$ is defined as

$$q_n = \min_{n_s \forall s} (q_n^s - d(n_s, n))$$

where $d(n_s, n)$ is the delay from the sink node $n_s$ to $n$.

**Dual $V_{dd}$ Buffer Insertion (*DVB*) Problem** – Given an interconnect fanout tree which consists of a source node $n_{src}$, sink nodes $n_s$, Steiner nodes $n_p$, candidate buffer nodes $n_b$ and the connection topology among them, the $DVB$ Problem is to find a buffer placement, a buffer size assignment and a $V_{dd}$ level assignment solutions such that the $RAT$ $q_n^{src}$ at the source $n_{src}$ is met and the power consumed by the interconnect tree is minimized, while slew rate at every input of the buffers and the sinks $n_s$ are upper bounded by $\hat{s}$.

## 2.4 Dual $V_{dd}$ Buffered Tree Construction

We measure the delay and power performance using the same metric as in the $DVB$ formulation. Assuming that a floorplan of the layout is available, we can identify the locations and shapes of rectangular blockages, which allows wiring on top but forbid buffer insertion, and the locations of the buffer station ($BS$) which are the allocated space for buffer insertion. Therefore we have the following problem formulation.

**Dual $V_{dd}$ Buffered Tree Construction (*D-Tree*) Problem** – Given the locations of a source node $n_{src}$, sink nodes $n_s$, blockages and $BS$, the *D-Tree* problem is to find the minimum power embedded rectilinear spanning tree with the buffer placement, the buffer sizes and the $V_{dd}$ assignment on the floorplan that satisfy the $RAT$ $q_n^{src}$ constraint at the source $n_{src}$ and the slew rate bound $\hat{s}$ at every input of the buffers and the sinks $n_s$.

In the *D-Tree* problem, we have alternative tree topologies as an extra dimension over the $DVB$ problem for optimization. Two *D-Tree* solutions are shown in Figure 2. The large rectangle and the black dots are the blockage and the $BS$ respectively. Both cases achieve the same $RAT$ at the source $n_{src}$. However, (a) has to go across a wide blockage and therefore has to rely on running a long high $V_{dd}$ net. An alternative route is shown in Figure 2(b) in which it chooses to go around the blockage so that it can insert more buffers to achieve the same delay while keeping the long route at low $V_{dd}$, which turns out to save power compared to (a).

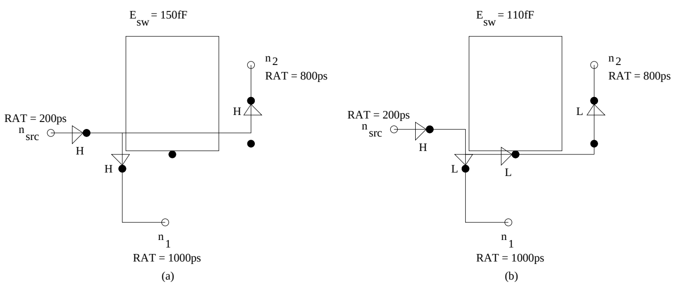


**Figure 2: Routing as a design freedom for power.**

# 3. BUFFER INSERTION

Power-optimal solutions are constructed from partial solutions from the subtrees. We call them as options, which are defined below.

*Definition 2.* An option $\Phi_n$ at the node $n$ refers to the buffer placement, size and $V_{dd}$ assignment for the subtree $T_n$ rooted at $n$. To perform delay and power optimization, the option is represented as a 4-tuple $(c_n, p_n, q_n, \theta_n)$, where $c_n$ is the downstream capacitance at $n$, $p_n$ is the total power of $T_n$, $q_n$ is the $RAT$ at $n$ and $\theta_n$ signifies whether there exists any high $V_{dd}$ buffer at the downstream. The option with the smallest power $p_n^{src}$ at the source node $n_{src}$ is the power-optimal solution.

Our algorithm is based on [1] with a few improvements. We add support for dual $V_{dd}$ buffer insertion without level converters. We also improve the runtime by introducing uniform sampling of the options under each capacitance value to reduce the number of options generated with negligible loss of optimality. To facilitate explanation, we define the concept of option dominance here.

*Definition 3.* An option $\Phi_1 = (c_1, p_1, q_1, \theta_1)$ dominates another option $\Phi_2 = (c_2, p_2, q_2, \theta_2)$ if $c_1 \le c_2$, $p_1 \le p_2$ and $q_1 \ge q_2$.

## 3.1 Baseline Algorithm

We enhance the dynamic programming framework in [1] to accomodate the introduction of dual $V_{dd}$ buffers, which is summarized in Table 2. We use the same notation as in Definition 2 to denote options $\Phi$ and their components. Moreover, we use $c_b^k$, $E_b^k$, $V_b^k$ and $d_b^k(c_{load})$ to denote the input capacitance, the power, the $V_{dd}$ level and the delay with output load $c_{load}$ of the buffer $b_k$. $d_{n,v}$ and $E_{n,v}(V)$ are the delay and the power of the interconnect between nodes $n$ and $v$ operating at voltage $V$. The set of available buffers $Set(B)$ contains both low $V_{dd}$ and high $V_{dd}$ buffers. We first call $DP$ at the source node $n_{src}$, which recursively visits the children nodes and enumerates all possible options in a bottom up manner until the entire interconnect tree $T_n^{src}$ is traversed.

Algorithm: $DP(T_n, Set(B))$

0. $Set(\Phi_n) = (c_n^s, 0, q_n^s, false)$ if $n$ is a sink<br>else $(0, 0, \infty, false)$<br>
1. for each child $v$ of $n$<br>
2. $Set(\Phi_v) =$ sampled $DP(T_v)$<br>
3. $Set(\Phi_{temp}) = Set(\Phi_n)$<br>
4. $Set(\Phi_n) = \emptyset$<br>
5. for each $\Phi_i \in Set(\Phi_v)$<br>
6. for each $\Phi_t \in Set(\Phi_{temp})$<br>
7. for each buffer $b_k \in Set(B)$<br>/* also contains the no buffer option $\phi$ */<br>
8. if $b_k = \phi$<br>
9. $V_n = V_H$ if $\theta_i$ or $\theta_t$ is true, else $V_L$<br>
10. $\Phi_{new} = (c_i + c_t, p_i + p_t + E_{n,v}(V_n),$<br>$\min(q_t, q_i - d_{n,v}), \theta_i$ or $\theta_t)$<br>
11. else if i. $V_b^k$ is high; or<br>ii. $V_b^k$ is low and $\theta_i$ is false<br>
12. $\Phi_{new} = (c_b, p_i + p_t + E_{n,v}(V_b^k) + E_b^k,$<br>$\min(q_t, q_i - d_{n,v} - d_b^k(c_i + c_{n,v}),$<br>$\theta_t$ or $(if\ V_b^k = V_H))$<br>
13. else goto line 7<br>
14. if i. slew rate violation at downstream buffers; or<br>ii. $\Phi_{new}$ *dominated* by any $\Phi_z \in Set(\Phi_n)$<br>
15. drop $\Phi_{new}$<br>
16. else<br>
17. remove all $\Phi_z \in Set(\Phi_n)$ dominated by $\Phi_{new}$<br>
18. $Set(\Phi_n) = Set(\Phi_n) \cup \{\Phi_{new}\}$<br>
19. return $Set(\Phi_n)$

**Table 2: Dynamic programming for buffer insertion.**

There are several new features in our algorithm in order to support the insertion of dual $V_{dd}$ buffers. Line 10 and 12 of Table 2 produce the new options $\Phi_{new}$ for the cases of no buffer insertion and inserting buffer $b_k$ respectively between node $n$ and $v$. In the case of no buffer insertion, we set $V$ to either $V_H$ for high $V_{dd}$ or $V_L$ for low $V_{dd}$ at line 9 according to the downstream high $V_{dd}$ buffer indicators $\theta_i, \theta_j$, and line 10 makes use of $V$ to update the power consumed by the interconnect. Note that when $\theta = false$ (ie. there is no high $V_{dd}$ buffers in the downstream), only the low $V_{dd}$ option has to be created since the high $V_{dd}$ counterpart is always inferior. In the case of buffer insertion, we simply add $E_{n,v}(V_b^k)$ according to the operational voltage of buffer $b_k$ to $p_{new}$ and update $\theta$ accordingly. Also note that we use line 11 to guard against low $V_{dd}$ buffers driving high $V_{dd}$ buffers to avoid the need of level converters, as explained in Section 2.1.

## 3.2 Power-Delay Sampling

We apply the technique of sampling to the options to reduce the growth of options, which can go to the order of billions for large nets if uncontrolled. The idea is to pick only a certain number of options among all options for upstream propagation (line 2 of Table 2) in the algorithm $DP$. Figure 3 shows (a) the pre-sample and (b) the after-sample option sets under the same capacitance. Each black dot corresponds to an option. We divide each side of the bounding box of all options into equal segments such that the entire power-delay domain are superposed by a grid. For each grid square in Figure 3(a), we retain only one option if there is any. By also including the smallest power option and the largest $RAT$ option, we obtain the sampled non-dominated option set in Figure 3(b).

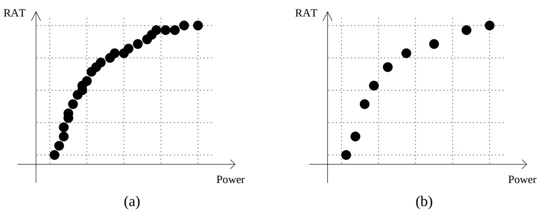


**Figure 3: Sampling the non-dominated options.**

Note that we do not sample on to capacitance values. The capacitance value in an option is for the purpose of accurate calculation of power and delay in the upstream of the tree. Moreover, the number of capacitance values is relatively small due to the upper bound slew rate constraint, which means that sampling on capacitance value has little effect anyway.

## 3.3 Experiment

We test our algorithm on 10 testcases $s1 \sim s10$ generated by randomly placing source and sink pins in a $1cm$ x $1cm$ box. We use a rectilinear Steiner tree generation package [15] to generate the connection between the source and the sink pins. We also break interconnect between nodes longer than $500\mu m$ by inserting degree-2 nodes. In this experiment we assume that every non-terminal nodes are candidate buffer nodes. We set the $RAT$ at all sinks to 0 so that the objective becomes minimizing the maximum delay from the source to any sink. Table 1 lists all the technology related settings. The slew rate bound $\hat{s}$ is set to $100ps$. We have made buffers using an inverter cascaded with another inverter which is four times larger. Buffer sizes used in the experiment are 16x, 32x and 64x. We compare three algorithms, which are i. power-optimal buffer insertion ($PB$) algorithm [1] considering only single (high) $V_{dd}$ buffers; ii. $SVB$ for our $DVB$ algorithm considering only high $V_{dd}$ buffers; and iii. $DVB$ for our $DVB$ algorithm considering dual $V_{dd}$ buffers. In both $SVB$ and $DVB$ we set the sampling grid to 20 x 20.

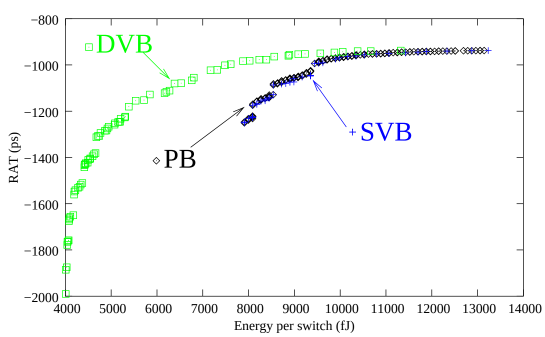


**Figure 4: Non-dominated solutions of $s4$.**

Figure 4 shows all non-dominated options at the source node $n_{src}$ (i.e. valid solutions) of the testcase s4. We observe that the sampling approximation introduced by our $DVB$ algorithm has almost no impact on the power-delay optimality, as the options from $SVB$ follow those from $PB$ very closely. We also see that introduction of dual $V_{dd}$ buffers in $DVB$ significantly improves the power optimality by pushing all option to the left of the graph.

Table 3 shows the experimental results for the three algorithms that we consider. Since the power and delay values are extremely similar between $PB$ and $SVB$, we omit those for $PB$ to save space. $RAT$* is the maximum achievable $RAT$ at the source. The percentages in the brackets show the relative change of power from $SVB$ to those in $DVB$. Runtime is measured on a Intel Xeon 1.9Ghz Linux workstation with 2Gb of memory. We see that on average using dual $V_{dd}$ buffers reduces power by 23% compared to the case when only high $V_{dd}$ buffers are considered at $RAT$*. When we relax the $RAT$ at the source to 105% of $RAT$*, the dual $V_{dd}$ buffer solution saves 26% of power compared to the high $V_{dd}$ buffer-only solutions. Also notice that $SVB$ is 17x faster than $PB$ on average.

# 4. BUFFERED TREE CONSTRUCTION

Using the sampling technique in Section 3.2, we attempt to extend the algorithms in [6, 7] to handle dual $V_{dd}$ buffered tree construction with power minimization as the objective. The *D-Tree* problem is an NP-Hard problem. In fact, in the case of no $BS$ and blockages, the *D-Tree* problem is essentially the optimal rectilinear Steiner tree problem and is known to be NP-Complete [16]. The artifact of the NP-hardness is the exponential growth of the number of options, which is complicated by considering power in addition to delay. We find that if we sample options using a very sparse grid (eg. 2 x 2 grid), we end up losing power optimality by dropping too many options. However, a denser grid causes catastrophic increase in runtime if we perform a linear scan for pruning each time the algorithm creates a new option. Therefore, solving the *D-Tree* problem requires a very efficient way of managing options, which has not been considered in [6, 7].

The data structure in [1] which uses an augmented orthogonal search tree for option pruning is a good starting point. The authors use a hash table labeled by power values as a container for search trees of capacitance and delay. In their algorithm they always add the options into the tree in the order of increasing capacitance. When combined with their dominance detection scheme, the algorithm adds only non-dominated options into the tree.

However, we cannot directly apply the data structure and operations described in [1] to solving the *D-Tree* problem. In this problem the order of node traversal is not known as a priori due to the combinatorial nature of the path searching problem. Therefore we can no longer guarantee the order by which options are added to the search tree. This may cause dominated options residing in the search tree, which causes the tree to take $O((\log m)^2)$ time (where $m$ is the number of options in the tree) per addition of option to update if balanced trees are used. Moreover, keeping redundant options also worsens the space requirement. Therefore, we need a way to efficiently prune options from the tree in order to retain option non-redundancy.

## 4.1 Dynamic Pruning

We propose an improved data structure, as shown in Fig-

| Testcase | | | runtime (s) | | | *SVB* | | *DVB* | |
|---|---|---|---|---|---|---|---|---|---|
| net | # nodes | # sinks | PB (s) | SVB (s) [x] | DVB | power @ *RAT** (fJ) | power @ 105% *RAT** (fJ) | power @ *RAT** [x] (fJ) [%] | power @ 105% *RAT** (fJ) [%] |
| s1 | 86 | 19 | 3 | 2 [1.5] | 6 | 4669 | 4127 | 3980 [-15%] | 3277 [-21%] |
| s2 | 102 | 29 | 4 | 3 [1.3] | 9 | 5476 | 4844 | 4785 [-13%] | 3750 [-23%] |
| s3 | 142 | 49 | 17 | 7 [2.5] | 20 | 8123 | 6316 | 6930 [-15%] | 4804 [-24%] |
| s4 | 226 | 99 | 224 | 33 [6.8] | 64 | 13232 | 9440 | 11322 [-14%] | 7876 [-17%] |
| s5 | 375 | 199 | 719 | 86 [8.4] | 212 | 18699 | 15275 | 13808 [-26%] | 11376 [-26%] |
| s6 | 515 | 299 | 2121 | 139 [15] | 371 | 23443 | 20117 | 17239 [-26%] | 14703 [-27%] |
| s7 | 784 | 499 | 33419 | 393 [85] | 635 | 33552 | 28336 | 23804 [-29%] | 20221 [-29%] |
| s8 | 1054 | 699 | - | 598 | 1072 | 38351 | 33686 | 25799 [-33%] | 22985 [-32%] |
| s9 | 1188 | 799 | - | 853 | 1859 | 40228 | 36358 | 26646 [-34%] | 23045 [-37%] |
| | | | | [17] | | | | [-23%] | [-26%] |

**Table 3: Experimental result of single and dual $V_{dd}$ buffer insertion.**

ure 5, similar to the one in [1] but also support solution pruning from the search trees. We label the hash table using capacitance instead of power and keep the power and $RAT$ portion of options in the tree instead. The slew rate upper bound tends to tightly upper bound maximum value of capacitance and therefore the hash table tends to be smaller, which results in less search trees.

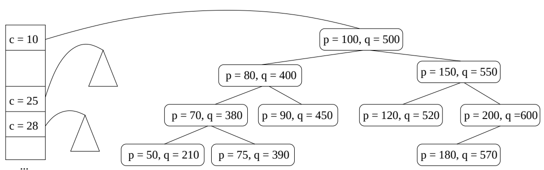


**Figure 5: Data structure for option pruning.**

The search trees are ordered so that at each node the power value is larger (smaller) than those in the nodes of the left (right) subtree respectively. We always maintain the tree so that no option dominates any other. Following from this, we immediately see that all $RAT$ $q$ are in the same order as power $p$, i.e. the $q$ values in the left (right) subtree of the node $n$ are smaller than (larger) than the $RAT$ $q$ of $n$. Therefore, we do not require explicit maintanance of the largest $RAT$ in the left subtree as in [1].

Our algorithm to prune dominated options from the tree is summarized in Table 4. $Set(\Phi_n)$, which contains the options at node $n$, are organized in the data structure mentioned above. In the pseudo-code we treat any option $\Phi_{cur}$ as a node in the search tree, and therefore $\Phi_{cur} \rightarrow left$ refers to the left child of the node storing the option $\Phi_{cur}$. We use $T_\Phi$ to denote the subtree rooted at $\Phi$. For each capacitance value that is larger than that in the new option $\Phi_{new}$, line 2$\sim$7 look for the first option $\Phi_{cur}$ in the tree that $\Phi_{new}$ domiantes. If one is found, line 8$\sim$13 prune the left subtree of $\Phi_{new}$ with a single downward pass of the tree, which takes only $O(\log m)$ time for $m$ options in the tree, by making use of the special tree ordering. The right subtree of $\Phi_{cur}$ is also pruned in a similar fashion. Note that after this step, options in the $Set(\Phi_{junk})$ can be removed and $\Phi_{new}$ can be inserted as usual in a balanced tree in $O(\log m)$ time. Rotation, which helps balancing the tree, requires no label updating as long as no option in the tree is dominated.

## 4.2 The *D-Tree* Algorithm

Table 5 summarizes the *D-Tree* algorithm. Each option now stores the "sink set" $\mathcal{S}$ and "reachability set" $\mathcal{R}$ to keep track of the sinks and the other nodes that the current option covers. The algorithm starts by building a grid using the "escape node algorithm" in [7]. Line 1$\sim$4 create the candidate buffer insertion nodes $n_b^k$ by looking for intersection points between $BS$ and the grid lines $(n_i, n_j)$. The core process of creating new options $\Phi_{new}$ considering dual $V_{dd}$ buffers is the same as that in the $DVB$ algorithm (refer to line 8-18 of Table 2) with additional book keeping to track the routability. The new pruning data structure in Section 4.1 is applied at line 17 for pruning options from $Set(\Phi_j)$.

Algorithm $CleanDominate(\Phi_{new}, Set(\Phi_n))$

0. $Set(\Phi_{junk}) = \emptyset$
1. for each distinct capacitance $c > c_{new}$ in $Set(\Phi_n)$
2. $\quad\Phi_{cur}$ = option at the root of the search tree under $c$
3. $\quad$while $\Phi_{cur} \neq \phi$
4. $\quad\quad$case 1: $p_{new} < p_{cur}$, $q_{new} < p_{cur}$, $\Phi_{cur} = \Phi_{cur} \rightarrow left$
5. $\quad\quad$case 2: $p_{new} < p_{cur}$, $q_{new} > q_{cur}$, goto line 2
6. $\quad\quad$case 3: $p_{new} > p_{cur}$, $q_{new} < q_{cur}$, goto line 9
7. $\quad\quad$case 4: $p_{new} > p_{cur}$, $q_{new} > q_{cur}$, $\Phi_{cur} = \Phi_{cur} \rightarrow right$
8. $\quad Set(\Phi_{junk}) = Set(\Phi_{junk}) \cup \{\Phi_{cur}\}$
9. $\quad\Phi_{dom} = \Phi_{cur} \rightarrow left$
10. $\quad$while $\Phi_{dom} \neq \phi$
11. $\quad\quad$case 1: $p_{new} < p_{dom}$, $Set(\Phi_{junk}) = Set(\Phi_{junk}) \cup \{\Phi_{dom}, T_{\Phi_{dom} \rightarrow right}\}$ $\Phi_{dom} = \Phi_{dom} \rightarrow left$
12. $\quad\quad$case 2: $p_{new} > p_{dom}$, $\Phi_{dom} = \Phi_{dom} \rightarrow right$
13. $\quad$repeat line 9$\sim$12 with modifications:
$\quad$i. exchange 'left' and 'right';
$\quad$ii. replace $p_{new}$ and $p_{dom}$ with $q_{new}$ and $q_{dom}$; and
$\quad$iii. exchange '<' and '>'

**Table 4: Dynamic tree update.**

## 4.3 Experiment

We create 5 testcases g1$\sim$g5 by randomly generating source and sink pins in a $1cm$ x $1cm$ box. We also randomly generate blockages so that it consumes approximately 30% of the total area of the box. Horizontal and vertical $BS$ are randomly scattered in the box so that the average distance between two consecutive $BS$ is about $1000\mu m$. The scales of these testcases as a result are similar to those in [6]. We use 32x and 64x buffers. We set the $RAT$ of all sinks to 0 so that maximizing $RAT$ at the source corresponds to minimizing the maximum delay from the source to any sink. The slew rate bound $\hat{s}$ is set to $100ps$. We again refer to Table 1 for technology related settings. We compare three cases, Algorithm $DTREE(n_{src}, Set(n_s), Set(BS), Set(Blockage))$

$\{Set(n_p), \aleph(Set(n))\} = Grid(Set(n), Set(Blockage))$
for each node $n_i \in Set(n)$
  for each neighbour node $n_j \in \aleph(n_i)$
    $Set(n) = Set(n) \cup \{n_p$ created by edge $(n_i, n_j) \cap Set(BS)\}$
    $\aleph(n_p) = \{n_i, n_j\}$; update $\aleph(n_i), \aleph(n_j)$
$Q(\Phi_n^{cur}) = \bigcup_{n_s \in Set(n_s)} Set(\Phi_n^s)$
while $Q(\Phi_n^{cur}) \neq \emptyset$
  $\Phi_n^{cur}$ = pop $Q(\Phi_n^{cur})$
  for each neighbour $n_j \in \aleph(n_{cur})$
    for each option $\Phi_n^j \in$ sampled $Set(\Phi_n^j)$
      if $(\Phi_n^j.\mathcal{R}) \cap (\Phi_n^{cur}.\mathcal{R}) = \emptyset$
        (form $\Phi_{new}$ similar to line 7∼14 in Table 2)
        $\Phi_{new}.\mathcal{R} = (\Phi_n^j.\mathcal{R}) \cup (\Phi_{new}.\mathcal{R})$
        $\Phi_{new}.\mathcal{S} = (\Phi_n^j.\mathcal{S}) \cup (\Phi_{new}.\mathcal{S})$
        if i. slew rate violation at downstream buffers; or
           ii. $\Phi_{new}$ *dominated* by any
              $\{\Phi_n^j : (\Phi_{new}.\mathcal{S}) \subseteq (\Phi_n^j.\mathcal{S}), \Phi_n^j \in Set(\Phi_n^j)\}$
          drop $\Phi_{new}$
        else
          remove $\{\Phi_n^j : (\Phi_{new}.\mathcal{S}) \supseteq (\Phi_n^j.\mathcal{S}), \Phi_n^j \in Set(\Phi_n^j)\}$
          dominated by $\Phi_{new}$
          $Set(\Phi_n^j) = Set(\Phi_n^j) \cup \{\Phi_{new}\}$
          push $\Phi_{new}$ into $Q(\Phi_{cur})$ if $n_j \neq n_{src}$

**Table 5: Dual $V_{dd}$ buffered tree generation.**

which are i. *RMP* in [6] for timing-aware buffered tree generation; ii. *S-TREE* for our *D-Tree* algorithm considering single (high) $V_{dd}$ buffers; and iii. *D-TREE* for *D-Tree* algorithm considering dual $V_{dd}$ buffers. Note that in the original implementation of [6] only options with the smallest capacitance under each reachable set are kept, which the authors claim has minimal impact on *RAT* optimality through experiment. However, we have found that the validity of this claim has strong correlation with the positions and density of the buffer candidate nodes. Therefore we choose to exclude this speed up heuristic to avoid losing optimal *RAT*.

| Testcase | | *RMP* | *S-TREE* | *D-TREE* | |
|---|---|---|---|---|---|
| # node | # sink | power @ *RAT** (pJ) | power @ *RAT** (pJ) [%] | power @ *RAT** (pJ) [%] | run-time (s) |
| 97 | 2 | 1.6 | 1.6 [0%] | 1.5 [-7%] | 1 |
| 165 | 3 | 3.4 | 3.4 [0%] | 3.2 [-4%] | 35 |
| 137 | 4 | 3.9 | 3.5 [-10%] | 2.9 [-23%] | 66 |
| 261 | 5 | 4.9 | 4.4 [-13%] | 3.1 [-37%] | 937 |
| 235 | 6 | 4.2 | 3.8 [-10%] | 3.4 [-18%] | 1391 |
| | | | [-7%] | [-18%] | |

**Table 6: Experimental result of timing-aware and dual $V_{dd}$ low power buffered tree generation.**

Table 6 shows the experimental results for the five test cases. We compare the power consumption at the maximum achievable *RAT* of each net. The percentages in the brackets show the reductions of power from the *RMP* to the *D-Tree* formulation with high and dual $V_{dd}$ buffers respectively. We observe a 7% reduction through power-minimization using high $V_{dd}$ buffers. Using dual $V_{dd}$ buffers gives 18% of power reduction over *RMP*. Note that power-optimal solution considering high $V_{dd}$ alone may not yield a better power as shown in the first two testcases, but the extra optimization dimension provided by using dual $V_{dd}$ always helps achieve power savings.

# 5. CONCLUSION AND FUTURE WORK

This paper presents the first in-depth study on applying dual $V_{dd}$ buffers to buffer insertion and multi-sink buffered tree construction for power minimization under delay constraint. We develop a sampling-based sub-solutions (i.e. options) propagation method and a balanced search tree based data structure for option pruning to cope with the increased complexity due to simultaneous delay and power consideration and increased buffer choices. We obtain 17x speedup with little loss of optimality compared to the exact option propagation [1]. Extensive experimental results show that when dual $V_{dd}$ buffers are considered, our algorithm reduces power by 23% at the minimum delay specification compared to [1]. Moreover, compared to the delay-optimal tree using single $V_{dd}$ buffers [6, 7], our power-optimal buffered tree reduces power by 7% and 18% when single $V_{dd}$ and dual $V_{dd}$ buffers are used respectively.

This work does not consider the issues regarding dual $V_{dd}$ power distribution network – we assume that all buffer stations support dual $V_{dd}$ buffers, which is valid only if the design uses dual $V_{dd}$ circuits extensively throughout the chip. In a lot of multiple $V_{dd}$ ASIC designs, designers assign voltages to regions ($V_{dd}$ bays). Co-optimization of dual $V_{dd}$ buffered routing and placement of these $V_{dd}$ bays is an interesting problem to pursue.

# 6. REFERENCES


[1] J. Lillis, C. Cheng, and T. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," in *ICCAD*, Nov. 1995.

[2] T. Okamoto and J. Cong, "Buffered Steiner tree construction with wire sizing for interconnect layout optimization," in *ICCAD*, Nov. 1996.

[3] J. Lillis, C. Cheng, and T. Lin, "Simultaneous routing and buffer insertion for high performance interconnect," in *GLVLSI Symp.*, 1996.

[4] C. Alpert, G. Gandham, J. Hu, J. Neves, S. Quay, and S. Sapatnekar, "Steiner tree optimization for buffers, blockages and bays," in *ISCAS*, May 2001.

[5] J. Hu, C. Alpert, S. Quay, and G. Gandham, "Buffer insertion with adaptive blockage avoidance," *TCAD*, vol. 22, no. 4, pp. 492–498, 2003.

[6] J. Cong and X. Yuan, "Routing tree construction under fixed buffer locations," in *DAC*, Jun 2000.

[7] W. Chen, M. Pedram, and P. Buch, "Buffered routing tree construction under buffer placement blockages," in *ASP-DAC*, Jan 2002.

[8] F. Li, Y. Lin, and L. He, "Vdd programmability to reduce fpga interconnect power," in *ICCAD*, Nov 2004.

[9] H. Bakoglu, *Circuits, Interconnects and Packaging for VLSI.* Addison-Wesley, 1990.

[10] K. Banerjee and A. Mehrotra, "A power-optimal repeater insertion methodology for global interconnects in nanometer designs," *TCAD*, vol. 49, no. 11, pp. 2001–2007, 2002.

[11] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, 2003.

[12] "Quickcap user manual," in *http://www.magma-da.com/.*

[13] "Berkeley predictive technology model," in *http://www-device.eecs.berkeley.edu/ ptm.*

[14] C. Alpert, D. Devgan, and C. Kashyap, "RC delay metrics for performance optimization," *TCAD*, vol. 20, no. 5, pp. 571–582, 2001.

[15] D. Warme, P. Winter, and M. Zachariasen, "Geosteiner," in *http://www.diku.dk/geosteiner*, 2003.

[16] W. Shi and C. Su, "The rectilinear steiner arborescence problem is NP-complete," in *ACM-SIAM Symp. on Discrete Algorithms*, 2000.