ABSTRACT

Due to the need of low power methodology in VLSI and SoC designs, voltage island architecture is attracting attentions in design community. However, the corresponding EDA tools development for voltage-island-aware buffered routing is still very few. Recent related studies focused on applying dual $V_{dd}$ buffers in routing tree construction, however it cannot be applied on a design using voltage island architecture due to the restriction on the ordering of low and high $V_{dd}$ buffers and the lack of level converter consideration. This paper presents approaches to solving the buffer insertion and level converter assignment problem in the presence of voltage island in a low-power design, especially under the fixed buffer locations. We have implemented and modified one state-of-the-art graph-based approach for this specific routing problem and applied our efficient heuristics (one of them is based on the selection of Steiner points) to further improve the performance, considering the assignment of buffers and level converters/shifters simultaneously. The experimental results show that we can obtain massive speedup over modified prior approach, and even with lower power and delay. Furthermore, as the number of sinks increases, our approach can effectively find feasible solutions, while modified prior approach cannot find solutions within a reasonable runtime.

Categories and Subject Descriptors
B.7.2 [Design Aids]: Placement and routing

General Terms
Algorithms, Design

Keywords
Low Power, Voltage Island Architecture, Buffer Insertion

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

1. INTRODUCTION

In CMOS digital circuits, power dissipation mainly consists of dynamic and static components. Dynamic and static power both have direct relationship with supply voltage $V_{dd}$. One of the techniques to reduce power consumption is voltage island methodology, proposed by IBM [8]. A voltage island is a group of on-chip cores powered by the same voltage source, independently from the chip-level voltage supply. Voltage island architecture can achieve power saving and this technique becomes more and more popular [6, 7, 8, 18, 19, 10, 14]. However, there are very few development in corresponding EDA tools regarding voltage-island-aware routing tree construction, which is very important in designing low power applications.

Based on the state-of-the-art buffered routing tree methodologies, we can put them into two categories. The first type is tree-based algorithms [5, 1, 15, 12, 11, 13]. They first generate routing tree topology and applied our efficient heuristics (one of them is based on the selection of Steiner points) to further improve the performance, considering the assignment of buffers and level converters/shifters simultaneously. The experimental results show that we can obtain massive speedup over modified prior approach, and even with lower power and delay. Furthermore, as the number of sinks increases, our approach can effectively find feasible solutions, while modified prior approach cannot find solutions within a reasonable runtime.

Figure 1: For a voltage island based design, we always need level converters/shifters to maintain signal level. Our approach treats a level converter as a kind of buffer, inserts buffers and places level converters on better locations than conventional tree-based algorithms do.

If we use one of the tree-based algorithms to insert buffers
and level converters, there will be two concerns. First, the generated routing tree may not have feasible level converter locations. As a result, the correct function cannot be performed. Second, if we treat a level converter as a standard cell and place it during placement stage, signal level can be maintained though, the location of level converter will affect total wirelength, delay and power. On the other hand, if we regard a level converter as a kind of buffer, a graph-based algorithm will provide better solutions. But [3, 2, 16] have presented that the complexity of graph-based algorithms grows exponentially when the number of sinks increases. Recent publication [16] uses RMP (Recursively Merging and Pruning) approach from [3] to seek for low power buffered routing solutions, but it cannot be applied on the designs containing voltage islands.

In order to apply the previous approach in voltage island based designs, we first modify the methodology in [16] (called modified RMP in this paper) to find solutions. However, to deal with the complexity problem, we adopt further greedy heuristics and pruning techniques in the proposed algorithm ViaBuf (Voltage-island aware Buffered tree construction) to improve modified RMP approach. This algorithm deals with both buffer insertion and level converter assignment problem, and also improves the complexity problem in graph-based algorithm. To the best of our knowledge, this paper is the first work on the buffered routing tree construction in the presence of voltage islands. The contributions presented in this paper are as follows:

- Since current dual-V\textsubscript{dd} buffer insertion approaches cannot be performed on the designs which contains voltage islands, we have modified the RMP approach in [16] so that it can be applied on those designs.
- Our method ViaBuf has provided massive speedup over modified RMP, and even produced lower power buffered trees. Furthermore, as number of sinks increases, our approach can find feasible solutions with quality efficiently.

The rest of this paper is organized as follows. Section 2 summarizes some related previous works and our problem formulation. Section 3 shows the modified RMP algorithm and Section 4 introduces our improved methodology ViaBuf to apply on this modified RMP algorithm for runtime reduction (and better solutions). Section 5 shows our experimental results and Section 6 concludes the paper.

2. BUFFERED ROUTING TREE CONSTRUCTION CONSIDERING VOLTAGE ISLANDS FOR LOW POWER

The algorithms about buffer insertion problem have been studied very extensively, such as [5, 1, 15, 12, 11, 13, 16, 20, 3, 2, 17, 4, 9]. Some of these methods (tree-based) have the routing tree as an input, and buffer insertion algorithm intends to find a minimum delay for this routing tree. On the other hand, others methods (graph-based) simultaneously build a routing tree and perform buffer insertion during building routing trees through graphs. Although these kinds of approaches always lead to less delay than tree-based buffer insertion algorithms, most of these algorithms need to spend more time to obtain a better result. Here we briefly introduce some of the algorithms, including the DVB algorithm in [16] for dual-V\textsubscript{dd} buffer insertion. Then we describe our power models used in this paper, followed by our problem formulation.

2.1 Previous Works

Almost all tree-based algorithms are based on the algorithm proposed by [5], van Ginneken’s algorithm deals with a routing tree which has multiple sinks and considers delay minimization only. It traverses a tree with a bottom up approach, and calculates delay with Elmore delay model. A pruning technique is adopted such that there will be only \( k \) kinds of buffer tree and result in a time complexity of \( O(n^2) \). Recently, [12] has improved the time complexity of a buffer insertion algorithm for a 2-pin net from \( O(n^2) \) to \( O(n\log n) \), and to \( O(n\log^2 n) \) for multi-pin nets. This kind of routing algorithms usually intends to find a Minimum Recliner Steiner Tree (MRST) with minimum total wirelength. Therefore, if the routing tree has less feasible buffer locations, the buffer insertion algorithm can only improve the delay with these locations and will result in a poor delay while comparing with a graph-based buffer insertion algorithm.

For a design with voltage island, if the MRST has none of feasible level converter location, the signal level can not be maintained and a leakage current will be generated.

For a 2-pin net, [9] proposes an elegant formulation of the maze routing with buffer insertion and wire sizing problem as a theoretic shortest path problem. But it still needs to consume much time on solving both maze routing and buffer insertion problem for a multiple sink net. In [17, 4], the maze routing with buffer insertion problem is converted into a graph collection problem. The buffered routing tree is constructed through a dynamic programming approach with combining buffered routing tree subsets. Although the dynamic programming approaches consume less time, the algorithms still need much time on table computation. RMP [3] simultaneously builds the routing topology with buffer insertion at the same time. RMP first creates a grid graph and solutions with factors of capacitance \( cap \), required arrival time \( RAT \), reachable sink set \( RE \), and \( buf \) for stating whether a buffer which had been placed is filled into each node in the created graph. With defining the formulation of solution propagation, various kinds of solutions could be generated during solution propagation, and each solution corresponds to a buffer routing tree.

DVB (Dual Vdd Buffer insertion) algorithm in [16] is the first in-depth study on applying dual V\textsubscript{dd} buffers to buffer insertion. Their algorithm is realized with both tree-based and graph-based methods. The experimental result shows that their algorithm reduces 18-26% power consumption while comparing with using single V\textsubscript{dd} buffers for buffer insertion. The DVB algorithm with graph-based method, their algorithm is similar to RMP algorithm, therefore exponential growth complexity is still a problem.

The reasons why the methodology in [16] cannot be applied in the designs which contain voltage islands are as follows. As shown in Figure 2, which DVB algorithm states that there are some unusual conditions when the delay in case (b) is larger than that in case (a). Therefore, they restrict the ordering of buffers by only high V\textsubscript{dd} buffers driving low V\textsubscript{dd} buffers, and thus none of level converter is needed in their algorithm. This reduces the complexity of a dual V\textsubscript{dd} buffer insertion problem, and this assumption makes the algorithm not realistic. First, for Figure 2 (b), if \( C_l \) is a high...
resistance, $L$ switch is defined as:

$$V_{dd} = \text{unit length capacitance}, \quad r_w = \text{unit length wire resistance}, \quad L = \text{downstream wirelength}, \quad V_{dd} = \text{voltage level of the device or the signal on the wire}, \quad R_b = \text{output resistance of a buffer}, \quad C_{load} = \text{downstream load capacitance}. $$

The interconnect power consumption $P_w$ measured by energy per switch is defined as:

$$P_w(L) = 0.5 * c_w * L * V_{dd}^2$$

The power model for high and low voltage buffers and level converters is also from the buffer library in [16].

2.2 Our Power and Delay Estimation Models

For consistency, we adopt the power and delay model introduced in [16]. The delay model is a distributed Elmore delay model. The delay of a wire $D_w$ and delay of a buffer $D_{buf}$ are defined as:

$$D_w(L) = (0.5c_w * L + C_{load}) * r_w * L$$

$$D_{buf} = D_b + R_b * C_{load}$$

2.3 Problem Formulation

We assume that there are two types of buffers (buffers with high and low $V_{dd}$) and one type of level converter with various kinds of size in the buffer library; level converters must be driven by a high $V_{dd}$ supply voltage so as to raise voltage level from low to high. Hence both level converters and high $V_{dd}$ buffers can only be placed within a high $V_{dd}$ region, low $V_{dd}$ buffers can only be placed within a low $V_{dd}$ region. And a voltage island (low $V_{dd}$ region) might be turned off while the chip is operating at power saving mode.

The problem of buffer insertion and level converter assignment on a dual $V_{dd}$ voltage island design can be specified as follows:

*Given a design with voltage island(s), a net with a source node, multiple sink node with RAT (required arrival time) at each sink, feasible buffer location, buffer library and wire obstacles (such as hard IPs), we want to construct buffered routing tree with buffer insertion and level converter assignment under the following constraints:*

- RAT at each sink should be met.
- The design works during power saving mode.
- Signal levels are maintained for all devices.

3. MODIFIED RMP APPROACH FOR DUAL-VDD VOLTAGE ISLAND ROUTING

Since there are currently no works which can be used in routing tree construction for designs with voltage islands, we modify the RMP in [3, 16] to apply on voltage island designs, and try to speed up the algorithm as well. [3, 16] performs reverse-maze-routing-like method to propagate the solutions from sinks to source in routing tree construction, and it chooses the solution with maximal RAT among all tentative solutions to propagate. This modified approach needs to use a container called wave to store the solutions saved in routing grid nodes. The solutions are stored based on their reachable sinks, which are the farthest sinks the solutions contain. During the process, new generated solutions are stored in a new wave. The following subsections show four parts of this modified approach in details: routing grid construction, initial solution fill, solution propagation, and solution pruning.

3.1 Routing Grid Construction

We first create the bounding box with the minimum rectangle covering all the source and sink nodes. Then we keep enlarging the bounding box such that there exists none of wire obstacles being cut from the box. During the enlargement of the bounding box, if the source node of the target net is not inside voltage island, we also treat the voltage island as an obstacle. We then partition the bounding box into a grid graph by using the vertical and horizontal lines intersecting at source and sink nodes, buffer locations, and four corners of the wire obstacle. An example of the grid graph is shown in Figure 3.
3.2 Initial Solution Fill

During this step, we fill initial solutions for each cross point in the grid graph in order to perform solution propagation. There are ten items (cap, rat, pow, rn, rs, B, signalV, Cbl, bend, totLength) in each solution, listed as follows:

- **cap**: The accumulated capacitive load.
- **rat**: Required arrival time.
- **pow**: The accumulated power consumption.
- **rn** (reachable node): The nodes that passed through to prevent from traversing the same path.
- **rs** (reachable sink): The farthest sink that the solution contains.
- **B**: The location where a buffer is placed and the size of the buffer placed at this location.
- **signalV** (signal voltage): Signal voltage level. If it is high, the solution can only be driven by a high $V_{dd}$ device. If it is low, both high and low $V_{dd}$ device can drive this solution.
- **Cbl**: When two solutions with different sinks are merged at buffer location, Cbl states the extra load capacitance that the buffer needs to drive.
- **bend**: The accumulated number of bends in current solution. It is for solutions pruning to prevent more vias.
- **totLength**: The accumulated wirelength.

According to the type of the node, one or some of the initial solutions are filled for each node. The initial solution at sink node is the starting point for the propagation, while the solutions at the other nodes are formed through the propagation toward the source node. We assume that there are $n_H$ high $V_{dd}$ buffers, $n_L$ low $V_{dd}$ buffers, $m$ voltage level converters in the buffer library. The rules for filling the initial solution are described below:

- If node $i$ is a sink node, fill the following solution:
  \[(cap, rat, 0, i, \emptyset, signalV, 0, 0, 0)\]
  
  This contains a buffered routing sub-tree with zero wirelength and it contains only the sink node $i$. The cap and rat state the load capacitance and requirement arrival time of this sink $i$ separately.

- If node $i$ is a source node, fill the following solution:
  \[(0, \infty, 0, i, \emptyset, b_j, Low, 0, 0, 0)\]
  
  where $b_j$ input capacitance=0, delay=0, output resistance = driving resistance of source node. This solution contains a buffer with the driving resistance of the device at source node is placed.

- If node $i$ is not a buffer location, fill the following solution (no buffer is placed here):
  \[(0, \infty, 0, i, \emptyset, Low, 0, 0, 0)\]

- If node $i$ is a buffer location and within the voltage island (low $V_{dd}$ region), fill 1 + $n_L$ following solutions:
  \[(0, \infty, 0, i, \emptyset, b_{iL}, Low, 0, 0, 0)\]  \[(0, \infty, 0, i, \emptyset, b_{iL}, Low, 0, 0, 0)\]  \[(0, \infty, 0, i, \emptyset, b_{iL}, Low, 0, 0, 0)\]  
  This shows no buffer is placed or various kinds of low $V_{dd}$ buffers are placed.

- If node $i$ is a buffer location and outside the voltage island, fill 1 + $n_H$ + $m$ following solutions:
  \[(0, \infty, 0, i, \emptyset, Low, 0, 0, 0)\]  \[(0, \infty, 0, i, \emptyset, b_{iH}, Low, 0, 0, 0)\]  
  This shows no buffer is placed or various kinds of high $V_{dd}$ buffers are placed, or various kinds of level converters are placed. Since some of the level converters needs both of the high and low $V_{dd}$ voltages and it is easier to transmit a signal from high $V_{dd}$ to low $V_{dd}$, we only allow the converters placed in high $V_{dd}$ regions.

During the fill of initial solutions, each solution at each sink node is stored in a container (wave) separately. The solutions in the waves will be propagated in next step.

3.3 Solution Propagation

Here we choose a wave $w$ and propagate all solutions in $w$ to neighboring nodes. While a solution $s_A$ at node A propagates to solution $s_B$ at the neighboring node B, there are three conditions regarding voltage islands to follow:

- If any sink voltage in solution $s$ is high and source voltage is high, we cannot put any buffer within voltage island.

- If $signalV$ in solution $s$ is high, low $V_{dd}$ buffer cannot be placed at the neighboring node.

- The paths should not be overlapped for the solutions in nodes A and B, which is $\exists n_A \bigcap n_B = \emptyset$.

Due to these conditions, we generate a new solution $s_{new}$ and store it in node B. We also store the solution in a wave with its rs. The new solution $s_{new}$ is:

- If $B_B = \emptyset$ (No buffer is placed at node B):
  
  \[
  cap_{new} = cap_B + cap_A + Cw
  \]
  \[
  rat_{new} = min(rat_B; rat_A - Dw)
  \]
  \[
  pow_{new} = pow_A + pow_B + P_w
  \]
  \[
  r_{new} = r_{n_A} \bigcup r_{n_B}
  \]
  \[
  s_{new} = s_{A} \bigcup s_{B}
  \]
  \[
  C_{bl_{new}} = 0
  \]
  \[
  bend_{new} = bend_A + bend_B + ((turn direction)?1 : 0)
  \]
  \[
  totLength_{new} = totLength_A + totLength_B + (distance between A and B)
  \]

If $B_B \neq \emptyset$ (Assume that buffer $j$ is placed at node B with output resistance $R_j$, delay $D_j$, capacitance $C_j$)

- \[
  cap_{new} = C_j
  \]
  \[
  rat_{new} = min(D_1, D_2); where D_1 = rat_B - R_j * (C_w + cap_A); D_2 = rat_A - (D_w + D_B + R_j * C_{bl_{new}})
  \]
  \[
  pow_{new} = pow_A + P_w(V_{dd} based on driver) + P_B
  \]
  \[
  r_{new} = r_{n_A} \bigcup r_{n_B}
  \]
3.4 Solution Pruning

In order to reduce runtime and the memory usage, we have defined two intuitive pruning conditions in modified RMP approach. Assume that there are two solutions $s_A$ and $s_B$ in the same node, we prefer to prune the solutions with more bends and more wirelength, and with power and capacitance dominance:

- Bends and wirelength pruning: If $b_{A} > b_{B}$ and $t_{A} \geq t_{B}$, then $s_A$ is dominated and can be pruned.
- van Ginneken style pruning: If $s_{A} < s_{B}$; $p_{A} < p_{B}$; $c_{A} < c_{B}$; $r_{A} \geq r_{B}$, then $s_B$ is dominated and can be pruned.

4. VOLTAGE ISLAND AWARE BUFFERED TREE CONSTRUCTION (VIABUF)

The modified RMP algorithm deals with a net that has $n$ sink nodes, and the routing grid graph size $N \times M$ and $K$ solutions with the same $rs$ (reachable sink) being propagated in each node. Since $n$ sink nodes has $2^n$ kinds of combinations of selection, the modified RMP algorithm has $O(2^n \times N \times M \times K)$ solutions during propagation. It is obvious that computation time will grow exponentially as number of sinks increases. If we only process one sink at a time, the complexity can be reduced to $O(N \times M \times K)$. Rather than handling $2^n$ kinds of sink combinations, this idea performs modified RMP algorithm with only one sink in each iteration. In other words, because the propagation allows to merge solutions with different sinks at each node in the grid graph, the modified RMP algorithm treats each node as a Steiner point. The solutions in the grid graph corresponding to various kinds of Steiner trees are generated. Instead of treating each node as a Steiner point, a solution is selected and then we retrace the graph to erase the solutions that do not relate to this solution. This approach can efficiently reduce Steiner points.

Our ViaBuf algorithm performs modified RMP algorithm on each sink, until all the sinks are processed. After one source-sink pair has been performed, a lower power solution is selected. In order to find lower power solution with RAT constraints, ViaBuf keeps useful solutions in the routing grid graph as follows (assuming that it is after performing modified RMP to sink $t_1$ and a desired solution $sol_D$ has been chosen):

- We keep the initial solutions filled in every node of the routing grid. Initial solutions are kept so that a buffered routing tree can be generated at the source node. We need those initial solutions to generate the buffered routing tree/branch for the sinks in next iteration.
- We keep the solutions on the nodes whose $rs = t_i$ ($t_i$ is one of the sink nodes of the net) in $sol_D$, and the solutions related to various kinds of buffer insertion on the path of $sol_D$. The reason of keeping these two kinds of solutions is that we want to have more Steiner points for selection during constructing the routing Steiner tree.

![Figure 4: An example of performing ViaBuf to a net with three sink nodes. The light circles are buffer locations. After generating the buffered routing tree for $t_1$, the solutions with $rs = t_1$ need to be kept for the node on the desired path since that node could be a Steiner point while performing modified RMP on next sink node.](image)

We use an example to illustrate the second condition. While performing ViaBuf for a net with three sink nodes (shown in Figure 4) where $t_1$ is first processed node with modified RMP algorithm since it is the nearest node to source node, a desired path between source node and $t_1$ is then selected. The solutions of nodes on the desired path with $rs = t_1$ are kept. The reason is that the nodes on this path could be Steiner points while performing modified RMP for next sink $t_2$. While performing modified RMP for $t_2$, shown in Figure 5, $t_2$ could share with the former buffered routing tree by using node A as a Steiner point since solutions with $rs = t_2$ are kept at node A. After performing modified RMP on $t_2$, solutions of nodes on the desired path with $rs = t_3$ should be kept. Furthermore, solutions of nodes on the desired path with $rs = t_1$, $t_2$ should also be kept so that they can be used (nodes as Steiner points) when the path is possibly shared by handling next sink. Finally $t_3$ is processed, if there exists another sink which needs to be processed, the useful solutions of the nodes on the desired path should be kept, shown in Figure 6.

The ViaBuf algorithm is shown in Figure 7 to generate solutions for routing tree topology. Note that the waves in ViaBuf are different from the ones in the modified RMP. In order to obtain the desired wave with target sink nearest to source during each iteration, ViaBuf only marks each wave with its target sink, while modified RMP marks each wave with the reachable sinks of solutions. Therefore, solutions in a wave might have different reachable sinks in ViaBuf algorithm, but all these solutions are intended to build a tree for this target sink. After a wave with target sink nearest to source is processed, solutions of nodes in this wave will be propagated to its neighbors (line 3-8). Once we have propa-
gated the solution to one of its neighboring node, we prune the redundant new generated solutions and store the non-redundant solutions to the neighboring nodes. When one or more than one new generated solutions are stored, we put the neighbor to a temporary wave (line 9-12). Whenever we complete a wave propagation, we check whether the source node is visited or not. If visited, we choose a desired solution which has least power consumption. If not visited, we again store the new generated wave (line 13-19). The algorithm stops when all the sinks have been processed.

5. EXPERIMENTAL RESULTS

We have implemented modified RMP and our ViaBuf in C++ and the platform is on AMD Dempron 1.75GHz with 1GB memory. We randomly generate several test cases. All these cases have six obstacles, one voltage island, and ten buffer feasible locations, but the number of sinks, pin location assignment, and buffer location assignment are different. For each of these cases, the size of the grid graph is about 25*25 on a 17*17mm design, and the number in each test case name shows the total number of source and sink pins.

The experimental results for modified RMP and our ViaBuf are shown in Table I. Since both modified RMP and our ViaBuf are heuristic methods, the results present advantages in both algorithms for cases net4, net5, and net6. Our approach also shows massive speedup over modified RMP algorithm. Moreover, when the number of sinks is larger than six, modified RMP cannot find solutions in six hours. There are several reasons that modified RMP cannot perform well. First, as we mentioned in previous section, it has high time complexity. Second, when performing solutions pruning, only solutions with the same voltage level can be pruned. Third, level shifters are not considered in original RMP algorithm, we need to add this part to work on voltage island design, thus increasing the complexity.

In Table I, there is one column to describe the specification of the benchmarks, which states the source node of the net is within voltage island or not. If the source node is within the voltage island, level shifter is a must to nodes that are out of voltage island. If there is no buffer locations between the source node and a sink node, it is necessary to detour to find a location for placing level shifter. That is, we need more waves for finding this detour, which results in more runtime. This explains the long runtime of net10 in finding solutions.

The buffered routing trees of net5 and net6 are is shown in Figure 8 and Figure 9. Figure 8 presents the case that the source node is within the voltage island. In order to avoid leakage current and to maintain signal level, level converter must be placed somewhere between the sink node out of voltage island and source node. Figure 9 presents the case that the source node is out of the voltage island. Since the voltage island might be turned off, none of the buffers is placed inside the voltage island. If we examine those buffered routing trees in Figure 8 and Figure 9, we can find that those trees are not MRSTs, which most routing algorithms intend to obtain. This means that our specified buffered routing tree must consider the signal level maintenance and the leakage current problems, i.e. there must be a level shifter somewhere between low voltage source node and high voltage sink node.

Algorithm ViaBuf
Input: A routing grid graph and a wavePool W
Output: Solutions at source node, each one corresponds to a buffered routing tree topology
1 While (W is not empty) {
2 get a wave w with sink nearest to source node
3 for each node ni in w {
4 for each solution si in ni {
5 if node nk which is a neighbor of ni {
6 propagate si to the solutions at neighbor node nk
7 store new generated solutions in temporary container Q
8 prune redundant solutions in Q
9 if Q is not empty {
10 store new generated solution from Q to nk
11 put nk to a temporary wave wtemp
12 } }
13 if wtemp contains source node {
14 choose a desired solution with least power consumption
15 erase useless solutions in the routing grid graph
16 } else {
17 W = W ∪ wtemp
18 } }

Table 1: Comparison between modified RMP and ViaBuf algorithm. The results present advantages in both algorithms for cases net4, net5, and net6, however our approach shows massive speedup over modified RMP algorithm. Our approach also achieves lower power with slightly worse phase delay (but conforming RAT constraints).

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>source out</th>
<th>vol island</th>
<th>modified RMP</th>
<th>ViaBuf</th>
</tr>
</thead>
<tbody>
<tr>
<td>net4</td>
<td>no</td>
<td></td>
<td>delay (ps)</td>
<td>1162</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>power (fJ)</td>
<td>9253</td>
</tr>
<tr>
<td>net5</td>
<td>no</td>
<td></td>
<td>CPU time (sec)</td>
<td>66.7</td>
</tr>
<tr>
<td>net6</td>
<td>yes</td>
<td></td>
<td>delay (ps)</td>
<td>1199</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>power (fJ)</td>
<td>12504</td>
</tr>
<tr>
<td>net10</td>
<td>no</td>
<td></td>
<td>&gt;6hr</td>
<td></td>
</tr>
<tr>
<td>net15</td>
<td>yes</td>
<td></td>
<td>&gt;6hr</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU time (sec)</td>
<td></td>
</tr>
</tbody>
</table>

6. CONCLUDING REMARKS

In this paper, we have implemented modified RMP algorithm (from [3, 16]) to deal with the designs in the presence of voltage islands. By using our efficient observation and pruning techniques, ViaBuf is much faster and can deal with multiple sinks net as the number of sinks increases. With RAT constraints, we can produce lower power buffered routing tree suitable for voltage island designs. Since our approach still has complexity limitation on the location of source node which is inside the voltage island, our future work plans to find a better approach for this specification to obtain solutions more efficiently.

7. REFERENCES


