Statistical Dual-Vdd Assignment for FPGA Interconnect Power Reduction

Yan Lin and Lei He
Electrical Engineering Department
University of California, Los Angeles
{ylin, lhe}@ee.ucla.edu, http://eda.ee.ucla.edu

ABSTRACT

Field programmable dual-Vdd interconnects are effective to reduce FPGA power. However, the deterministic Vdd assignment leverages timing slack exhaustively and significantly increases the number of near-critical paths, which results in a degraded timing yield with process variation. In this paper, we present two statistical Vdd assignment algorithms. The first greedy algorithm is based on sensitivity while the second one is based on timing slack budgeting. Both minimize chip-level interconnect power without degrading timing yield. Evaluated with MCNC circuits, the statistical algorithms reduce interconnect power by 40% compared to the single-Vdd FPGA with power gating. In contrast, the deterministic algorithm reduces interconnect power by 51% but degrades timing yield from 97.7% to 87.5%.

1. INTRODUCTION

Modern VLSI designs see a large impact from process variation as devices scale down to nanometer technologies. Similar to ASICs, FPGAs are subject to variations in the operation of transistors comprising the logic functionality and the switching muxes. Unique in FPGAs, the guard-banded timing model will be applied to designs unknown in advance, and can be arbitrarily conservative or aggressive [1]. Statistical analysis on timing or power is necessary under the presence of variations. [2] presents closed form formulae for FPGA leakage yield, and performs architecture and device co-evaluation considering process variation. It has been shown in [2] that the FPGA on-chip delay and leakage variations can be up to 1.9X and 3X, respectively. [1] develops a stochastic placement leveraging statistical static timing analysis (SSTA) for FPGA timing yield optimization with process variation.

Meanwhile, field programmable dual-Vdd (VddP) techniques have been used for FPGA interconnect power reduction [3]. A Vdd-level converter is needed when a low-Vdd (VddL) interconnect switch drives a high-Vdd (VddH) switch to avoid excessive leakage. It has been shown that the fine-grained level converter insertion in interconnects consumes large leakage [3]. [4] proposes a formulation in which no VddL switch drives VddH switches within a routing tree such that each tree may have two Vdd levels but without level converter in interconnects. The proposed Vdd assignment algorithm in [4] is based on timing slack budgeting with linear programming (LP) formulation for FPGAs with uniform wire length, and is extended to mixed wire lengths in [5]. [6] further re-formulates the LP formulation to a min-cost network flow (netflow) formulation and achieves an average of 8X overall speedup.

However, the deterministic Vdd assignment leverages timing slack on non-critical paths exhaustively for power reduction. Although the critical path delay does not increase, the portion of near-critical paths increases significantly. With process variation, any near-critical paths may become statistically timing critical. Therefore the increased number of near-critical paths may result in a larger mean and variance of circuit delay, and a much degraded timing yield with variation. As shown in Figure 1, the usage of programmable Vdd increases the percentage of near-critical paths (with delay larger than 90% critical path delay) from 0.26% to 4.7%, which degrades the timing yield from 97.7% to 89.2%.

Figure 1: (a) Path-length distribution; (b) delay distribution with variation (circuit: s38417).

A similar issue has been addressed in statistical gate sizing for ASICs [7]. [8] presents statistical gate sizing based on iterative greedy algorithm with SSTA evaluated in each iteration. Other work studies non-linear programming based gate sizing. [9] presents a robust LP formulation with timing yield estimated while this estimation may be highly pessimistic with local variation considered. [10] presents an iterative Lagrangian Relaxation (LR) based sizing algorithm. However, the mean and nominal delay are implicitly assumed as the same which may result in an over-optimistic timing yield estimation. There is no existing work in the literature for statistical Vdd assignment for FPGAs. Moreover, the existing circuit tuning techniques in ASICs [8, 9, 10] cannot be directly applied to FPGAs due to the Vdd-level constraint, i.e. no VddL switch should drive VddH switches without level converter in interconnects.

This paper is partially supported by NSF grant CCR-0306682 and Actel under UC MICRO program. Address comments to lhe@ee.ucla.edu.

1 The mean delay may be larger than the nominal one due to max operation with variations [11].
In this paper, we assume the same Vdd-programmable interconnects as [5, 6] and study statistical Vdd assignment for FPGA interconnect power reduction. Two statistical algorithms are proposed. The first greedy algorithm is based on sensitivity, namely, greedy-s. The second one is based on timing slack budgeting with netflow formulation, namely netflow-s. The budgeting stage in netflow-s is deterministic and identical to that in EdTLC-NW [6] (called as netflow in this paper). Different from [9, 10], our algorithm netflow-s guarantees that the near-critical path number does not increase to avoid timing yield degradation without a large number of SSTAs. In addition, Vdd-level constraint is considered in both greedy-s and netflow-s. Compared to single-Vdd FPGA with power-gating, the deterministic algorithm, netflow, reduces interconnect power by 51.2% but degrades timing yield from 97.7% to 87.5%. In contrast, greedy-s and netflow-s reduce interconnect power by 40% and 39.5% respectively but without degrading timing yield. Moreover, netflow-s runs 3.6X faster than greedy-s.

The rest of the paper is organized as follows. Section 2 introduces modeling and problem formulation. Section 3 presents statistical Vdd assignment algorithms. Section 4 discusses the experimental results. We conclude the paper in Section 5. Note that our technique to bound the number of near-critical paths can be extended to circuit tuning with a variety of design freedoms for ASICs.

2. PRELIMINARIES

2.1 Process Variation Model

Process variation can be classified as global, affecting all aspects of a given chip, spatial/regional, affecting geographic areas of the chip, or local, randomly affecting each individual transistor. Delay and leakage of a circuit element (e.g., an LUT or a routing switch) are random variables with process variation.

The delay of a timing edge, \( d \), is modeled as a normal distributed (Gaussian) random variable and can be captured by the following model in the first-order canonical form

\[
d = d_0 + a \cdot p_d + a \cdot p_s + a \cdot p_r,
\]

where \( d_0 \) is the mean value of \( d, a \) is the delay sensitivity to this variation source, and \( p_d, p_s \), and \( p_r \) model the global, spatial, and local variations, respectively. Leakage power is exponentially affected by variation such as that in effective channel length. Same as [2], the leakage current of one circuit element, \( i \), is modeled as a lognormal variable as

\[
i = i_0 \cdot e^{p_d \cdot p_s + p_s + p_r},
\]

where \( i_0 \) is the nominal value of leakage current and \( s \) is the leakage sensitivity to this variation source.

We assume that \( p_d, p_s \), and \( p_r \) all follow a standard normal distribution \( N(0, 1) \). Moreover, \( p_d, p_s \), and \( p_r \) are mutually independent, as their causing mechanisms are different by definition. To model spatially correlated variation, an FPGA chip is partitioned into \( m \) grids and perfect correlation among the devices is assumed in the same grid. We adopt the methods from [12] to generate the covariance matrix and use principle component analysis (PCA) [13] to transform a set of correlated random variables to an uncorrelated set. We use SPICE to extract delay and leakage variations for basic circuit elements under different Vdd-levels in FPGA, considering process variation in effective channel length \( L_{eff} \), threshold voltage \( V_T \), for delay and further gate oxide thickness \( T_{ox} \) for leakage [2].

2.2 Statistical Timing and Leakage Analysis

Statistical static timing analysis (SSTA) has recently been proposed to analyze timing with process variation [14, 13]. The probabilistic equivalents of the “max”, “min”, “add” and “subtract” operations are involved in SSTA. The delay in the canonical form, addition and subtraction are performed easily [14]. The max or min of two Gaussians is not a Gaussian, but is modeled as a Gaussian [11] and then expressed in the canonical form, which allows us to propagate the correlations due to global and spatial variations. With forward and backward traversals of the timing graph, the distribution of the arrival and requested arrival time for each node can be obtained. Given a cut-off delay \( T_{cut} \), the timing yield is defined as the probability that the critical path delay is no longer than \( T_{cut} \) considering variation, and can be calculated using the cumulative density function (CDF) of circuit delay.

Statistical leakage analysis has been presented in [15] for ASICs and in [2] for FPGAs. We extend the method from [2] to consider spatial variation for chip-level leakage analysis under variation. The chip-level leakage power is the sum of the leakage of each circuit element, where each is a lognormal distributed random variable. This sum of the lognormals is then modeled as another lognormal. The leakage yield is defined as the probability that the chip-level leakage is no larger than a given cut-off leakage, \( P_{leak} \), considering variation. By transforming the lognormal distributed chip-level leakage into its corresponding normal random variable, leakage yield can be analytically calculated given a specified cut-off leakage \( P_{leak} \).

2.3 Problem Formulation

In this paper, we assume the VddP interconnects identical to [5, 6]. A mix of Vdd-levels within one routing tree is allowed while no VddL switch should drive VddH switches. The deterministic Vdd assignment formulation in [5] leverages timing slack exhaustively for power reduction. The increased portion of near-critical paths results in a significantly degraded timing yield with variation. The statistical Vdd assignment problem is to assign Vdd level to interconnect switches such that the power is minimized and timing yield is not degraded compared to that using single-Vdd.

3. STATISTICAL VDD ASSIGNMENT

In this section, we present two statistical Vdd assignment algorithms. The first greedy algorithm, greedy-s, is based on sensitivity. The second algorithm, netflow-s, is based on timing slack budgeting with netflow formulation, namely, netflow-s and netflow-s are motivated by their deterministic counterparts, dTLC-S in [4] (called as greedy ) and EdTLC-NW in [6] (called as netflow ), respectively. Both greedy-s and netflow-s minimize power under timing yield constraint considering process variation. The details of these two algorithms are presented below.

3.1 Sensitivity Based Algorithm greedy-s

We extend the deterministic Vdd assignment algorithm greedy [4] to consider timing yield constraint under process variation. greedy initializes VddH to all switches and iteratively assigns VddL to candidate switches in non-increasing
order of power sensitivity. The assignment is accepted if timing constraint is not violated and vice versa. In either case, this assigned switch will not be visited again. The iteration terminates when there is no candidate switch. A switch is a candidate switch if firstly it has not been visited, and secondly it does not drive any switch or VddL has been assigned to all of its fanout switches. Power sensitivity is defined as the power reduction when VddL is assigned to a switch with both dynamic and leakage power considered.

There are two main differences between greedy-s and greedy. The first difference is that SSTA instead of STA is performed to analyze delay distribution in each iteration in greedy-s. The assignment is accepted if timing yield requirement is met and vice versa. Secondly, statistical criticality is considered in sensitivity. For each switch u, the sensitivity sens(u) is calculated as \( \frac{\partial P_c}{\partial u} \), where ps is the power sensitivity and scrit is the statistical criticality. Statistical criticality is the probability for a timing edge/node to be statistically critical under variation [14]. Essentially, we try to assign VddL first to the switch with larger power saving but with a less probability to be timing critical.

Algorithm 1 greedy-s

1: ReverseTopologicSort(SW, &parent, &fo);
2: \( L \leftarrow \emptyset \);
3: for each \( u \in SW \) do
4: \( ps(u) \leftarrow \Delta P_c; \ flag(u) \leftarrow false; vdd(u) \leftarrow VddH; \)
5: if \( \text{fo}(u) = \emptyset \) then
6: \( L \leftarrow L \cup u; \)
7: SSTA();
8: While \( L \neq \emptyset \) do
9: \( v \leftarrow \text{GetMaxSensSwitch}(SW, ps, scrit); \)
10: \( L \leftarrow L \setminus \{v\}; \ flag(v) \leftarrow true; vdd(v) \leftarrow VddL; \)
11: SSTA and update criticality
12: \( Y \leftarrow CDF(T_{\sigma - T_{\mu}}); \)
13: if \( Y \leq Y_{\text{target}} \) then
14: \( vdd(v) \leftarrow VddH; \)
15: while \( \text{parent}(v) \neq \emptyset \) do
16: \( v \leftarrow \text{parent}(v); \ flag(v) \leftarrow true; vdd(v) \leftarrow VddH; \)
17: else if \( \text{parent}(v) \neq \emptyset \land \text{flag}([\text{parent}(v)]) = false \) then
18: \( x \leftarrow \text{parent}(u); f \leftarrow true; \)
19: for each \( c \in \text{fo}(u) \) do
20: \( \text{if} \ \text{flag}(c) = false \land \text{vdd}(c) = VddH \) then
21: \( f \leftarrow false; \) break;
22: if \( f = \text{true} \) then
23: \( L \leftarrow L \cup x; \)

The greedy-s algorithm (shown in Algorithm 1) first sorts all the switches in the reverse topological order (line 1). parent and fo save the parent switch and fanout switch set for each switch, respectively. Lines 2-6 initialize power sensitivities ps, Vdd levels, and visit flags for all switches. In addition, all leaf switches, i.e., a switch without fanout switch, are put into candidate switch list L. Line 7 performs an initial SSTA and calculates statistical criticality for each switch. greedy-s enters a loop (lines 8-23) and terminates the iteration after no candidate switch exists (line 8). Lines 9-10 remove the candidate switch with the largest sensitivity from L and assign VddL to it. Lines 11-12 perform SSTA with statistical criticality updated and calculate the current timing yield. The assignment is rejected and VddH is assigned to all of its upper stream switches if yield constraint is violated (lines 13-16). Otherwise, if its immediate upper stream (parent) switch is a candidate switch, we put this parent switch into list L (lines 17-23). It is clear that the complexity of greedy-s is dominated by SSTA with \( O(m \cdot (|V| + |E|)) \) complexity, where m is a constant and is decided by the number of variation sources and the number of grids in spatial variation model, \( |V| \) and \( |E| \) are the number of vertices and edges in the timing graph respectively. greedy-s requires \( n + 1 \) SSTA in the worst case and therefore has the worst case complexity of \( O(n \cdot m \cdot (|V| + |E|)) \), where \( n \) is the number of switches.

3.2 Slack Budgeting Based Algorithm

netflow-s

 greedy implicitly allocates timing slack first to interconnect switches with higher power sensitivity to reduce more power. On the other hand, netflow explicitly allocates timing slack to minimize power. There are three phases in netflow-s (see Figure 2 (a)). Timing slack is first allocated to each routing tree by re-formulating the linear programming (LP) formulation [4, 5] as a network flow (netflow) problem to minimize chip-level nominal power without increasing critical path delay. A bottom-up assignment algorithm is then performed to achieve the optimal solution within each tree given the allocated slack. Finally, a refinement step is performed to leverage the surplus timing slack.

Motivated by netflow, we develop its statistical counterpart, netflow-s, to consider timing yield constraint under process variation. netflow-s has three steps in the similar fashion (see Figure 2 (b)). The first step in netflow-s is the same as that in netflow, i.e. deterministically allocating timing slack using netflow formulation. In the second step, we perform an iterative robust bottom-up assignment considering process variation. In each iteration, we adjust the threshold of near-critical path delay and guarantee that the number of near-critical paths with delay larger than this threshold does not increase. Finally, a statistical refinement step is performed to leverage the surplus timing slack. Below, we only focus on the second and third steps but skip the timing slack budgeting based on netflow formulation. Interested readers can refer to [4, 5, 6] for more details on the timing slack budgeting formulation.

Figure 2: (a) netflow (b) netflow-s

Observed that the usage of VddP increases the number of near-critical paths significantly and degrades timing yield, we develop the iterative robust bottom-up assignment algorithm with binary search on static criticality, crit, such that the number of near-critical paths with delay larger than \( \text{crit} \cdot T_{\text{spec}} \) does not increase in each iteration. Given a path
P and critical path delay $T_{\text{spec}}$, static criticality, crit, is defined as $(T_{\text{spec}} - \text{delay}(P))/T_{\text{spec}}$, where delay(P) and $T_{\text{spec}} - \text{delay}(P)$ are the delay and slack of P, respectively. Given a timing edge or node, the static criticality and slack are calculated based on the longest path passed through this edge or node. Slack of each edge can be analyzed by STA, i.e., the difference between the requested arrival time and the arrival time. A range of 0 to 1 for crit covers all the solution space for timing yield and power tradeoff. crit = 0 corresponds to the VddH case, i.e., no path delay is increased. In this case timing yield is maximized with no power reduction. On the other hand, crit = 1 corresponds to the deterministic VddP case, i.e., any near-critical path delay can be increased as long as the critical path delay does not increase. In this case, power reduction is maximized while timing yield may be significantly degraded with process variation.

To guarantee that the number of near-critical paths with delay larger than crit · $T_{\text{spec}}$ does not increase with VddP, the allocated slack for each edge is adjusted as follows,

$$ \text{adjusted slack} = \begin{cases} \text{ slack } & \text{if } \text{ slack } < (1 - \text{crit}) \cdot T_{\text{spec}} \\ \text{allocated slack} & \text{otherwise} \end{cases} $$(3)

where slack is the VddH slack, i.e., when the whole circuit is using VddH, slack is the VddP slack, i.e., when all the allocated slacks are consumed. allocated slack and adjusted slack are the allocated and adjusted slack, respectively. Note that the slack and allocated slack of an edge are two different concepts. Given a $T_{\text{spec}}$, STA can be performed to analyze slack, slack, for each edge under VddH. After timing slack budgeting, the delay of each edge can be increased by allocated slack but without increasing the critical path delay $T_{\text{spec}}$. With the delay of each edge increased by allocated slack, STA can be performed again to analyze slack, for each edge under VddP. If the longest path through an edge under VddH has delay larger than $(1 - \text{crit}) \cdot T_{\text{spec}}$, i.e., the first case in (3), the allocated slack is set to 0 without further increasing the near-critical path delay. In the second case in (3), the longest path delay through an edge under VddP has delay smaller than $(1 - \text{crit}) \cdot T_{\text{spec}}$, i.e., this longest path is not near-critical even using VddH. In this case, the allocated slack remains the same. In the third case in (3), where the longest path through the edge is not near critical under VddH but becomes near critical under VddP, the allocated slack of each edge along this path is scaled with a factor such that this path becomes non-near critical. The scaling factor for each edge is,

$$ \text{scale} = \frac{\text{slack}_{\text{vddh}} - (1 - c) \cdot T_{\text{spec}}}{\text{slack}_{\text{vddh}} - \text{slack}_{\text{vddp}}} $$

Figure 3 shows an example for adjusting slack. For the purpose of simplicity, we assume no branch for the path shown in this figure. The path consists of three edges with 2ns delay for each edge. The critical path delay $T_{\text{spec}}$ is 10ns resulting in a vddh slack of 4ns, which is the same for three edges in this example. After slack budgeting, each edge is allocated with a slack of 1ns resulting in a vddp slack of 1ns. This path then becomes a near-critical path with delay of 0.9$T_{\text{spec}}$. Suppose that we set crit to 0.8, i.e., the number of near-critical paths with delay larger than 0.8$T_{\text{spec}}$ should not increase. The allocated slack for each edge is then scaled by a factor of $(4ns - (1 - 0.8) \cdot 10ns)/(4ns - 1ns) = 2/3$. After scaling, the vddp slack becomes 2ns and the path delay becomes no greater than the threshold crit · $T_{\text{spec}}$.

**Figure 3:** (a) Original path (b) Path with allocated slack (c) Path with adjusted slack.

**Theorem 1.** Given a static criticality crit, (3) guarantees that the number of near-critical paths with delay larger than crit · $T_{\text{spec}}$ does not increase.

**Sketch of proof:** The proof is straightforward for near-critical paths under VddH or non-near critical paths under VddP. For near-critical paths due to VddP, the proposal can be proved by induction. It has been shown that this proposal stands for a path without branches. If a branch is added, slack and slack, either remain the same or decrease for each edge in the path. It can be verified that (4) must decrease or remain the same. Therefore, the path delay under VddP must not increase with branches added. This proof can be applied to any path in the circuit.

**Theorem 2.** Given a static criticality, crit, (4) is the maximum scaling factor to prevent the increase of near-critical path number with delay larger than crit · $T_{\text{spec}}$.

**Sketch of proof:** For the example in Figure 3 and a crit of 0.8, any scaling factor larger than 2/3 may result in a near-critical path with delay larger than 8ns.

**Algorithm 2 IterativeRobustBottomUpAssign**

1: STA();
2: $Y \leftarrow 0; c_0 \leftarrow 1; c_i \leftarrow 0; c \leftarrow 0.9$
3: \{adjust allocated slack\}
4: while $Y < Y_{\text{target}}$ do \{c = $\text{crit}$ \}
5: if each edge $\in$ routing edges do
6: if slack $< (1 - \text{crit}) \cdot T_{\text{spec}}$ then
7: adjusted slack $= 0$
8: else if slack $< (1 - \text{crit}) \cdot T_{\text{spec}}$ then
9: adjusted slack $= \text{scale} \cdot \text{slack}$
10: \{perform bottom up assignment for each net\}
11: for each routing tree $i$ do
12: \{adjust static criticality factor\}
13: if $Y > Y_{\text{target}}$ then
14: $c_i \leftarrow c_i \cdot c \leftarrow \frac{1}{c_i}$
15: else
16: $c_n \leftarrow c_i \cdot c \leftarrow \frac{1}{c_n}$

We present the iterative robust bottom-up assignment in Algorithm 2. Line 1 performs STA and calculates slack.
and slack \(vddp\) for each edge. Line 2 initializes static criticality to 0.9, and its upper and lower bound to 1.0 and 0.0, respectively. Line 3 enters a loop (lines 3-16) and terminates when the current timing yield is larger than the target timing yield and the difference between the lower and upper bounds of \(crit\) is small enough. Lines 4-9 adjust allocated slack for each edge based on \(crit\). Lines 10-11 perform deterministic bottom-up assignment for each routing tree given the adjusted slack. This step is the same as the bottom-up assignment in netflow, where VddL is iteratively assigned to all switches in one routing tree in a bottom-up fashion. The iteration in the bottom-up assignment terminates when no more switch in this tree can be applied with VddL. Line 12 performs SSTA and analyzes timing yield. Lines 13-16 adjust \(crit\) and its lower and upper bounds based on the current timing yield. Note that this iterative adjustment on \(crit\) is a heuristic and conservative approach, which trades power reduction with runtime (the number of SSTAs) compared to greedy-s (as shown in Section 4).

After the iterative robust bottom-up assignment, we may further reduce power by leveraging the surplus slack. To leverage the surplus slack, we mark all the VddH switches as ‘untried’ (flag ← false) but keep the VddL switches as ‘tried’ (flag ← true), and then perform greedy-s algorithm (see Algorithm 1) to achieve more VddL switches and further reduce power but without degrading timing yield.

Solving the netflow based timing slack budgeting formulation has the worst case complexity of \(O(|V|^2|E|\log|V|+K)\), where \(K\) is a constant. However, it has been shown in \[6\] that the netflow based budgeting runs \(50X\) faster than the LP based budgeting in practice and the speedup is larger for larger circuits. For the robust iterative bottom-up assignment, the complexity is \(O(H \cdot m \cdot (|V| + |E|))\), where \(H\) is the number of iterations and \(m \cdot (|V| + |E|)\) is the SSTA complexity. For the statistical refinement, the complexity is \(O(\eta \cdot n \cdot m \cdot (|V| + |E|))\), where \(\eta\) is the number of SDDH switches after the iterative robust bottom-up assignment. The overall runtime of netflow-s is dominated by the statistical refinement step and netflow-s runs 3.6X times faster than greedy-s (as shown in Section 4).

4. EXPERIMENTAL RESULTS

In this section, we conduct the experiments on the largest MCNC designs [16]. We use the Berkeley predictive device model [17] at TTRS [18] 65nm technology node with VddH 1.0v and VddL 0.7v. The VPR FPGA toolset [19] implements an island style FPGA architecture resembling Altera’s Stratix device [20] with 10 4-LUT clusters, and 60% length-4 and 40% length-8 wires. 1.2X of minimum routing channel width is used for each design. VddH is applied to all logic blocks. The nominal power is calculated using the power model in [21]. Single-Vdd with power gating is used as the baseline for power comparison. We implement an SSTA based on [14] for timing yield analysis. To model spatial correlation, each FPGA chip is partitioned into grids such that each grid contains five tiles in one dimension (around 0.5mm in 65nm technology). The correlation covariance coefficient decreases to 0.1 at 2mm distance. We assume a variation in each of \(L_{coll}, V_{dc}, T_{mu}\) and \(T_{sigma}\) of 10%, 10% and 6% at 3σ (i.e. a 99.7% chance that variation is within ±/− 10% or 6% deviated from the nominal value) for global, spatial and local variations, respectively. \(T_{mu}\) + 2\(T_{sigma}\) under single-Vdd is used as the cut-off delay (i.e. 97.7% yield with single-Vdd) for timing yield analysis. 1.2X of leakage power using single-Vdd is used as the cut-off leakage for leakage yield analysis.

Table 1 compares the timing yield between single-Vdd, netflow, greedy-s and netflow-s. The mean, \(T_m\), and standard deviation, \(T_s\), of circuit delay in netflow are presented in the difference compared to their counterparts in single-Vdd, respectively. The deterministic assignment netflow uses the critical path delay in single-Vdd as the timing constraint, i.e. the critical path delay does not increase after assignment. However, netflow increases \(T_m\) and \(T_s\) by 8.8% and 14.8% respectively due to the significantly increased number of near-critical paths with VddP. The average timing yield, \(Y_T\), in netflow therefore degrades from 97.7% to 87.5%. In contrast, the statistical assignment algorithms do not increase \(T_m\) and \(T_s\), and maintain the timing yield.

Table 1: Timing yield comparison between single-Vdd, netflow, greedy-s and netflow-s.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>albu2</td>
<td>19.44</td>
<td>3.26</td>
<td>3.45</td>
<td>19.92</td>
</tr>
<tr>
<td>apex2</td>
<td>22.62</td>
<td>3.24</td>
<td>3.45</td>
<td>19.92</td>
</tr>
<tr>
<td>apex4</td>
<td>19.33</td>
<td>2.52</td>
<td>2.12</td>
<td>19.92</td>
</tr>
<tr>
<td>bigeye</td>
<td>11.15</td>
<td>1.28</td>
<td>1.28</td>
<td>19.92</td>
</tr>
<tr>
<td>close</td>
<td>35.96</td>
<td>5.21</td>
<td>5.21</td>
<td>19.92</td>
</tr>
<tr>
<td>dso</td>
<td>22.02</td>
<td>3.12</td>
<td>3.12</td>
<td>19.92</td>
</tr>
<tr>
<td>eaddi</td>
<td>26.16</td>
<td>5.04</td>
<td>5.04</td>
<td>19.92</td>
</tr>
<tr>
<td>dep</td>
<td>10.36</td>
<td>1.41</td>
<td>1.41</td>
<td>19.92</td>
</tr>
<tr>
<td>elab</td>
<td>30.55</td>
<td>5.08</td>
<td>5.08</td>
<td>19.92</td>
</tr>
<tr>
<td>ex1010</td>
<td>26.8</td>
<td>3.21</td>
<td>3.21</td>
<td>19.92</td>
</tr>
<tr>
<td>g164</td>
<td>20.03</td>
<td>2.67</td>
<td>2.67</td>
<td>19.92</td>
</tr>
<tr>
<td>frise</td>
<td>41.78</td>
<td>7.54</td>
<td>7.54</td>
<td>19.92</td>
</tr>
<tr>
<td>micas3</td>
<td>18.38</td>
<td>2.48</td>
<td>2.48</td>
<td>19.92</td>
</tr>
<tr>
<td>pcle</td>
<td>27.91</td>
<td>3.54</td>
<td>3.54</td>
<td>19.92</td>
</tr>
<tr>
<td>s298</td>
<td>38.69</td>
<td>5.9</td>
<td>5.9</td>
<td>19.92</td>
</tr>
<tr>
<td>s38417</td>
<td>30.13</td>
<td>4.16</td>
<td>4.16</td>
<td>19.92</td>
</tr>
<tr>
<td>s38584</td>
<td>20.93</td>
<td>4.04</td>
<td>4.04</td>
<td>19.92</td>
</tr>
<tr>
<td>seq</td>
<td>18.51</td>
<td>2.51</td>
<td>2.51</td>
<td>19.92</td>
</tr>
<tr>
<td>sp18</td>
<td>25.08</td>
<td>3.11</td>
<td>3.11</td>
<td>19.92</td>
</tr>
<tr>
<td>sp37</td>
<td>23.7</td>
<td>4.21</td>
<td>4.21</td>
<td>19.92</td>
</tr>
</tbody>
</table>

We then compare interconnect power, leakage yield and VddL switch number between single-Vdd, netflow, greedy-s and netflow-s in Table 2. Columns 2-3 present the interconnect power and leakage yield using single-Vdd. When using 1.2X leakage power as the cut-off leakage, the average leakage yield is 60%. Columns 4, 7 and 10 present the VddL switch number (in the percentage of total switch number) achieved by the three assignment algorithms. On average, netflow, greedy-s and netflow-s achieve 95.6%, 75.7% and 75.7% VddL switches, respectively. Columns 5, 8 and 11 present the interconnect power reduction achieved by the three algorithms. On average, netflow, greedy-s and netflow-s reduce interconnect power by 51.2%, 40% and 39.5%, respectively. We also present the leakage yield achieved by the three algorithms in columns 6, 9 and 12. With the same cut-off leakage, netflow, greedy-s and netflow-s have an average leakage yield of 95.7%, 88.6% and 88.5%, respectively. Compared to netflow, the statistical algorithms achieve slightly less power reduction but without degrading timing yield with variation. The two statistical algorithms, greedy-s and netflow-s, achieve similar power reduction. In addition, we present the VddL switch number and power reduction due to statistical refinement for netflow-s in columns 10 and 11. The statistical refinement achieves 25.7% (out of 75.7%) VddL switches and 12.3% (out of 39.5%) interconnect power reduction. The statistical refinement is effective in power reduction due to the surplus timing slack. Since we scale the allocated slack conservatively without increasing any near-critical path delay, some near-critical paths may
be slightly increased under timing yield constraint which results in the surplus slack. This conservative adjustment trades power reduction with runtime.

Table 3 compares the runtime between netflow, greedy-s and netflow-s. Geometric mean is based on 20 MCNC designs.

Table 3: Runtime (s) comparison between netflow, greedy-s and netflow-s. Geometric mean is based on 20 MCNC designs.

Table 6: REFERENCES