Variation-Aware Variable Latency Design

Saket Gupta and Sachin S. Sapatnekar, *Fellow, IEEE,*

***Abstract*—Although typical digital circuits are designed so that the clock period satisfies worst-case path delay constraints, the average input excitation often completes computation in less than a clock cycle. Variable latency units (VLUs) allow for improved throughput by allowing one clock cycle for some computations, and two clock cycles for others, using hold logic to differentiate between the two cases. However, they may experience significant throughput losses due to the effects of process variations. We develop a combined presilicon-postsilicon technique for variation-aware VLU design that ensures high throughputs across all manufactured chips. We achieve this by identifying path clusters at the presilicon stage, such that each element of a path cluster is likely to be similarly critical in a manufactured part. We use sensors to determine which path clusters is critical at the postsilicon stage and then activate the appropriate hold logics. Practically, for a small number of path clusters, significant improvements in throughput are achievable. On a set of 32nm PTM-based ISCAS89 circuits, our scheme offers 15.1% throughput enhancements with only 3.3% area overhead.**

I. Introduction

The traditional paradigm of optimizing a chip for the worst-case delay can lead to significant inefficiencies. This has led to the investigation of circuits that are optimized for "better-than-worst-case," rather than the worst-case, computations of a circuit. Better-than-worst-case computations occur frequently and incur less than the worst-case delay. Recent efforts have targeted an associated computation model under the synchronous paradigm [1]–[8]. Two classes of techniques have been proposed: telescopic or variable-latency units (VLUs) [1]–[5], and error detection-correction units [6]–[8].

Although these methods must contend with the increasing process, voltage, and temperature variations associated with shrinking feature sizes, such issues have not been addressed to date. We focus on the design of process variation-aware VLUs.

A. Variable Latency Units

Unlike conventional circuits that complete operations within one clock cycle, VLUs [1] allow the computation of the circuit to be completed in a variable, integer, number of clock cycles. By allowing high-probability operations to complete in a single cycle, but allowing rarer events to use multiple (typically two) cycles, the average cycle time may be shorter than that of the conventional implementation, implying that the circuit throughput for a VLU may be significantly larger.

The idea can be illustrated through the example of a 6-bit ripple carry adder (RCA) shown in Fig. 1 [5]. With unit gate delays, the conventional single-cycle fixed-latency circuit has a cycle time, $T_{clk} = 13$ units, equal to the delay of its longest path, corresponding to a throughput, $\eta_1 = 1/13$. The VLU implementation of this adder operates at a reduced cycle time, $T_{clk} < 13$. For $T_{clk} = 9$, assuming that all primary input signals are mutually independent and have signal probabilities of 50%, 18.75% of the input patterns violate T_{clk}, and the VLU allows these to complete execution in two cycles. Under the 50% assumption above, since each pattern is equiprobable, the average VLU delay is $0.8125 \times 9 + 0.1875 \times 18 = 10.69$ units, and the throughput $\eta_2 = 1/10.69$ is 21.6% better.

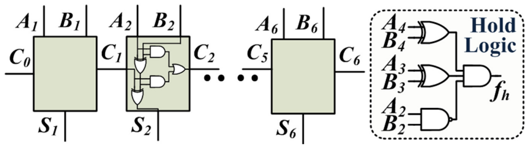


Fig. 1. A VLU implementation of a 6-bit ripple carry adder.

VLUs require dedicated circuitry for identifying the input patterns that require two cycles for completion, to prompt each output flip-flop to hold its current value at the next clock transition (rather than clocking in a new value). This is referred to as "hold logic" and its output is called the "hold signal." For a circuit with N_g gates, hold logic can be generated using a block-based algorithm with O(N_g) complexity [1], [4].

The hold logic for the RCA is small and is shown in Fig. 1. For general circuits, this area overhead is small compared to the circuit's area [1], [4], [5]. Such approaches can be applied to arithmetic units with little overhead in the overall chip area, since these units form a small fraction of the chip area, which is dominated by caches and other units.

In general, the throughput η of a VLU can be evaluated as:

$$\eta = \frac{1}{P_{hold} \cdot 2T_{clk} + (1 - P_{hold}) \cdot T_{clk}} = \frac{1}{(1 + P_{hold}) \cdot T_{clk}} \quad (1)$$

where P_{hold} is the hold logic activation probability.

B. Variation-Aware VLUs

Process variations refer to the variations in the values of CMOS device parameters, such as the transistor width (W), effective channel length (L_{eff}), and oxide thicknesses (t_{ox}) [9], that occur during the process of fabrication. Due to process variation, performance-related metrics such as the delay, slew, and power in circuits and architectures are severely affected. While the error detection-correction units employ techniques for variation tolerance [7], [8], and VLUs have been designed to compensate for aging [2], [5], none of the above research works investigate process variations in VLUs.

VLUs are characterized by the property that the hold logic is highly dependent on distribution of delay amongst the various

This work was supported in part by the NSF under awards CCF-1017778 and CCF-1162267 and the SRC under grant 2012-TJ-2234.

Saket Gupta is with Broadcom Corporation, Edina MN, 55435, USA, and Sachin S. Sapatnekar is with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis MN, 55455, USA (e-mail: saket.gupta@broadcom.com, sachin@umn.edu)

paths in the circuit. If the choice of one-cycle and two-cycle paths in a VLU is based on nominal delay estimates from the presilicon stage, it is quite possible that process variations during the fabrication can change this distribution and hence affect the functional correctness of VLUs. This can be overcome using pessimistic delay estimates; however, due to pessimism, there may be many paths that are identified as two-cycle paths that can operate in one cycle in a manufactured part. As a result, such manufactured parts are unable to achieve the best possible throughput.

In this paper, we propose a combined presilicon-postsilicon solution for constructing variation-aware variable latency designs, for ensuring near-maximal throughputs. At the presilicon stage, we employ a novel clustering scheme to capture paths that are likely to be simultaneously critical, due to similarities arising from structural and spatial correlations. This scheme is based on an initial node clustering scheme, followed by path clustering, based on a simple yet effective metric for measuring the closeness of path delays. We build hold logic for each such cluster. At the postsilicon stage, we use a set of ring oscillator based sensors to determine which clusters are critical: the hold logics for these clusters are activated, and the others are power-gated, implying low power overhead. We show that our clustering scheme provides large improvements over a conventional approach, which would pessimistically mark any potentially critical path as a two-cycle path, even if it operates faster in a manufactured part, with low area and power overhead. We refer to our scheme as *variation-aware hold logic* (VAHL), and show that it maintains functional correctness as well as high throughput by being tailored to individual manufactured parts.

The rest of the paper is organized as follows: Section II first presents some background on statistical timing and postsilicon delay estimation. We then introduce our approach of variation-aware VLUs in Section III. We overview our novel hold logic scheme in Section IV, and then present our approaches for node and path clustering in Sections V and VI. Finally, we present our results and conclusions in Sections VII and VIII, respectively.

II. Background

A. Statistical Timing Analysis

The timing analysis engine used in this work is based on standard block-based statistical static timing analysis (SSTA), the formulation and algorithms for which are described in [10]. Variations in process parameters are modeled as Gaussian distributions, resulting in delay quantities also being Gaussian. SSTA accounts for the spatial correlations of parameters for delay calculations. For devices and wires that are spatially closely located, the electrical parameters will also tend to vary in a more similar manner, resulting in their delays experiencing similar shifts in different manufactured dies. Block-based SSTA is based on a principal components decomposition of the underlying variational parameters, performed once for a process, and a topological SSTA traversal, performed for each design, where the delay is propagated in the following canonical form:

$$d = d_0 + \sum_{i=1}^{n} k_i p_i + k_{n+1} p_{n+1} \tag{2}$$

Here, d is the delay of interest, with a mean of d_0, p_1 through p_n are a set of principal components, p_{n+1} is a lumped variable that represents the independent component, and k_1 through k_{n+1} are the corresponding coefficients. The unit operations during the topological traversal are the sum and max operations; further details are available in [10]. In this paper, it is implicit that all delays are computed statistically. In the interest of bringing out our main ideas and reducing notational clutter, details of the statistical operations are not shown here; however, they rely on what are, by now, standard operations that have been extensively explored in prior work.

We define the slack, ST[G], at an output node of gate G as ST[G] = RT[G] $-$ AT[G], where RT[G] is the Required Time AT[G] is the Arrival Time at the node. We define the slack time ST[P] for a path P as the slack at the output of the final gate on the path. Note that AT[G], RT[G], and ST[G] are all expressed in canonical form, similar to Equation (2). For simplicity, we assume that flip-flop setup times are zero; nonzero setup times can be easily incorporated.

We define criticality in terms of the ability to violate the timing constraint, i.e., a path is critical if the its delay exceeds T_{clk}, the clock constraint. In the presence of variations, a path may be critical in some chips and noncritical in others. As a practical measure, we define *potentially critical* paths as those for which the $\mu + 3\sigma$ value of the presilicon statistical path delay exceeds T_{clk}, where μ is the mean value of the delay, and σ is its standard deviation. Under Gaussian delay approximations, this 3σ condition implies path noncriticality in 99.73% of the manufactured dies, which practically means that the path will never be critical in a manufactured part. Note that this assumption is especially reasonable since real path delay distributions are truncated Gaussians (e.g., delays cannot be negative, although a full Gaussian allows for this possibility). Under this definition, a gate G is *potentially critical* if the $\mu - 3\sigma$ value of ST[G] is less than 0.

B. Postsilicon Delay Estimation

In order to determine the precise value for circuit delay in a particular die, we adopt a methodology seated between SSTA and full-chip testing [11], which uses measurements from a small number of on-chip ring oscillators (ROs) as sensors, located in different parts of the die. The essential idea is that due to spatial correlations, delay shifts in a surrogate sensor based on ROs can be used to predict delay shifts in any on-chip logic.

In this methodology, SSTA is used at the presilicon stage to obtain a delay vector $\mathbf{d_t}$ = $[d_1, d_2, ..., d_k]$ of k such sensors. At the postsilicon stage, delay measurements are performed for a particular die to obtain their *real delay* vector $\mathbf{d_r}$. Using $\mathbf{d_t}$ and $\mathbf{d_r}$, a conditional delay PDF is evaluated for narrowing the spread (variance) of the SSTA based circuit delay PDF, to yield a close estimate of the real circuit delay in the fabricated die.

III. Variation-Aware Hold Logic

As outlined earlier, critical paths in a fabricated die may differ for different manufactured parts. In order to optimize the throughput for each specific part, the hold logic must be specific to the part. This motivates the need to develop a combined presilicon-postsilicon procedure for the generation of a hold logic (or a set of hold logics) that can be (a) functionally correct across all chips at the postsilicon stage, (b) ensure high throughput, and (c) incur low area and power overhead. We will now proceed to describe several approaches for designing VAHL, showing tradeoffs between the design overhead and the throughput benefits.

A. The Conventional Pessimistic Approach

Timing estimates over the entire population of manufactured die may be generated through either a corner-based method or by using 3σ delay points from an SSTA-based presilicon analysis. Using the results of such an analysis, we can generate a single hold logic by applying the hold logic generation algorithm to all potentially critical paths. After manufacturing, some parts may operate faster than this estimate, but since the presilicon hold logic is associated with a pessimistic presilicon delay estimate, it is guaranteed to operate correctly.

B. The Enumerative Approach

An alternative approach that enumerates paths is recognizably impractical, but will serve to motivate our eventual solution. Under this scheme, we build a separate hold logic corresponding to each of N potentially critical paths in the circuit, $P_1, P_2, \ldots, P_N$, as identified by presilicon SSTA.

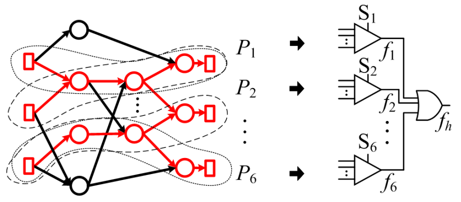


Fig. 2. The enumerative approach for VAHL generation.

If P_i is indeed critical in the manufactured part, then its hold logic, f_i, is activated; otherwise, it is power-gated and f_i is set to logic 0. This operation is controlled by a sleep signal S_i that is set to 1 when f_i is to be power-gated, and is 0 otherwise. Fig. 2 illustrates this idea for an example where N = 6. It shows the directed acyclic graph (DAG) representation of a small circuit and its corresponding enumerative hold logic. The convention used in this graph will be repeated throughout the paper: potentially critical nodes and edges in the graph are marked in red, all others are black in color, and primary inputs (PIs) and primary outputs (POs) are represented as square vertices.

The generation of the sleep signals is based on the prediction of the postsilicon delays of each of these paths, and a crucial ingredient of this solution is to determine whether a path is critical or not in each manufactured part. We achieve this through the scheme in [11] described in Section II-B. Given a small set of measurements on k RO sensors, the problem of prediction of the real delay, D_{P_i}, of each path P_i, is framed as an evaluation of a conditional probability; we leave out the statistical details in this presentation and refer the reader to [11] for a complete description. Based on the results of this determination, the values of f_i and S_i are set according to the criteria described above.

The enumerative method is impractical for two related reasons. First, path enumeration can be prohibitively time-consuming. Second, the hardware overhead, in terms of area and power, of all hold logics over all paths can be large.

C. A Clustered Approach for VAHL

The two methods described earlier define two extremes: the pessimistic approach has low overhead since it generates a single hold logic, but sacrifices throughput; the enumerative approach achieves the best possible throughput, but may have a large overhead since the number of hold logics required, N, can be very large. We adopt a strategy inspired by the notion of the "Middle Way" [12] to find a solution for VAHL generation that is a happy medium between the two extremes.

A key observation that drives our approach is that since delay variations in a circuit can show significant amounts of correlation, for two reasons. First, structural correlations imply that many potentially critical paths may share common subpaths. Second, spatial correlation of random variations can result in correlations in the delays of physically nearby paths, even if they do not have many gates in common. We define the notion of a *path cluster*. Membership within a path cluster is defined according to the following criterion: if any path in the cluster is critical (or noncritical) at the postsilicon stage for a particular chip, all other paths in the set are deemed to have the same property. Therefore, instead of treating these paths separately, as in the enumerative approach, we may use a combined hold logic, driven by a single sleep signal, for a path cluster. By construction, this VAHL will be pessimistic over a cluster, i.e., it will activate the hold logic for the cluster if even one path within a cluster is critical.

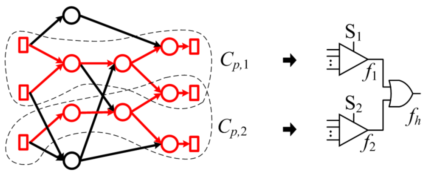


Fig. 3. Path clustering based approach for VAHL generation.

Using this principle, the N potentially critical paths in the circuit may be grouped into m path clusters, with m hold logics in all, which together will constitute the VAHL.[1] Each path cluster, denoted by C_p, will contain a subset of all N potentially critical paths, such that none of these N paths is left unclustered. Further, by definition, each of these clusters will have paths whose delays are similar and correlated, and will vary in the same way in different chips. We will discuss

[1]Note that grouping paths together in a cluster implies that the resulting hold logic is the logical OR of the individual hold logics. This provides for the possibility of logic minimization through logic sharing. However, VAHL *does not* allow any logic sharing amongst the hold logics of different path clusters, since we need the ability to put some of these hold logics to sleep if the corresponding path cluster is noncritical in the manufactured part.

computationally efficient algorithms for the generation of path clusters in Sections V and VI.

This scheme can be illustrated through Fig. 3, which shows m = 2 separate path clusters for the circuit of Fig. 2, which had $N = 6$ paths. Consequently, only two hold logics are required, instead of six hold logics in the enumerative case in Fig. 2. Based on this clustered scheme, we make the following observations:

- The pessimistic case is a special extreme case, where $m = 1$. In general, whenever $m > 1$, this approach will involve less pessimism than the pessimistic approach outlined above, and hence result in higher throughput.
- The enumerative case corresponds to the opposite extreme, where $m = N$.

Therefore, with appropriate path clustering, the number of outputs m can be tuned to be much smaller than enumeration, yielding (as will be shown in Section VII) significantly better throughput than the pessimistic case, but with much less overhead than enumeration.

IV. Implementing Clustered VAHL: Overview

We now present a more systematic description of our clustered approach through Algorithm 1, including further details for generating the select signals. The procedure consists of a presilicon stage (lines 1 through 7) where path clusters are generated, followed by a postsilicon stage (lines 8 through 14) where the criticality of the path clusters is determined.

Algorithm 1: VAHL Generation Framework

```
1  /* Presilicon Stage */
   Input: $T_{clk}$, Circuit: a levelized circuit
2  Perform SSTA
3  $\{C_{p,1}, ..., C_{p,m}\}$ = GeneratePathClusters(Circuit)
4  for each path cluster $C_{p,i}, 1 \leq i \leq m$ do
5      Compute $f_i$ = hold logic for $C_{p,i}$
6      $D^{SSTA}_{C_{p,i}} \leftarrow$ maximum delay $\{\forall$ paths in $C_{p,i}\}$
7  Determine $\mathbf{d_t}$ = $[d_1, d_2, ..., d_k]$
8  /* Postsilicon Stage */
9  for each manufactured chip do
10     $\mathbf{d_r} = [d_1, d_2, ..., d_k]$ from measurements of RO sensors
11     for each path cluster $C_{p,i}$, $1 \leq i \leq m$ do
12         Compute $D^{cond}_{C_{p,i}}$ = PDF $(D^{SSTA}_{C_{p,i}} \mid \mathbf{d_t} = \mathbf{d_r})$ using the
           method in [11]
13         $D^{r}_{C_{p,i}} \leftarrow (\mu + 3\sigma)$ of $D^{cond}_{C_{p,i}}$
14         $(D^{r}_{C_{p,i}} > T_{clk})$ ? {S$_i \leftarrow 0$} : {S$_i \leftarrow 1; f_i \leftarrow 0$}
```

At the presilicon stage, given a T_{clk} specification at the POs of the circuit, we first perform an SSTA (line 2), and generate m path clusters $\{C_{p,1}, ..., C_{p,m}\}$ (line 3) using a procedure that will be described in Sections V and VI. For each of these path clusters, $C_{p,i}$, we then evaluate the corresponding hold logic (line 5). We also compute, in canonical SSTA form [10], the delay PDF of the maximum delay over all paths in the path cluster, $D^{SSTA}_{C_{p,i}}$ (line 6). Finally, we determine, also in canonical form, the statistical presilicon delay vector $\mathbf{d_t}$ = $[d_1, d_2, ..., d_k]$ of the k RO-sensors on the chip (line 7).

At the postsilicon stage, we aim to identify which of the m path clusters will be critical in a particular chip, so that we can decide appropriate subset of these that should be active on that chip. We therefore perform postsilicon measurements for the RO sensors (line 10) for each manufactured chip to obtain the resultant RO sensor delay vector sample, $\mathbf{d_r} = [d_1, d_2, ..., d_k]$. This information is used to predict the postsilicon delays of each path cluster $C_{p,i}$ based on a conditional PDF evaluation [11], $D^{cond}_{C_{p,i}}$, of $C_{P,i}$ (line 12). Next, the real delay, $D^{r}_{C_{p,i}}$, of $C_{p,i}$ is estimated (line 13) to be the $(\mu + 3\sigma)$ point of its conditional PDF. In practice, this value is very close to its mean, as the conditional PDFs have a small variance [11].

Having estimated the real delays, we then determine (line 14) if $C_{p,i}$ is critical in the specific manufactured chip by comparing the values of T_{clk} and $D^{r}_{C_{p,i}}$. If the estimated delay is larger than T_{clk}, then the hold logic is left active; otherwise it is put to sleep and the corresponding f_i is set to zero. The hardware implementation, shown in Fig. 4(b), performs a floating point addition of $D^{r}_{C_{p,i}}$ and the 2's complement of T_{clk}, and checks the sign bit of the result. If this sign bit is 0, then the clock period has been exceeded; otherwise not.

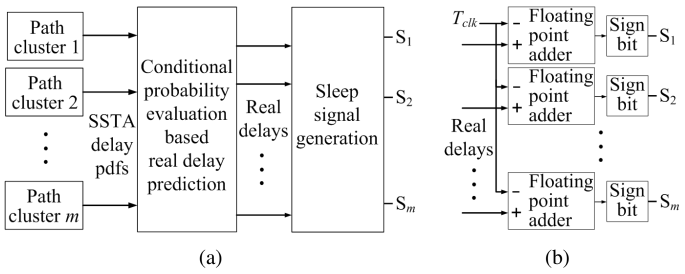


Fig. 4. Postsilicon processing for determining the sleep signal values: (a) overall flow, and (b) hardware for sleep signal generation.

We now evaluate the overhead required to generate the sleep signals. All postsilicon steps described above are a one-time computation for each manufactured part. The conditional PDF evaluation for all m path clusters can be performed by a simple function; the runtime for one such evaluation is reasonable [11]. Practically, as we will find in Section VII, the number of path clusters generated by our scheme is typically less than 7. The calculation for sleep signal generation described in line 14 is also only a one-time computation that can leverage hardware (hardware comparators or adders, as well as registers) that already exist on-chip in many designs. Hence, very little extra hardware is required for generation of these sleep signals, other than a few multiplexers to appropriately route data to these units.

V. Enabling Practical Path Clustering

A. Qualitative Criteria for Path Clustering

A set of paths in the circuit should be clustered together, with the hold logic controlled by a single sleep signal, if they all have a high probability of together being critical/noncritical after fabrication. This means that they should experience similar shifts in their delays from similar mean values. The similarity of mean values is an important consideration, as depicted in Fig. 5, which shows the PDFs of the delays D_{P_i} and D_{P_j} of paths P_i and P_j. Both PDFs are observed to have a high correlation, but since their mean values differ

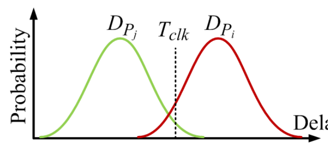


Fig. 5. Importance of comparing the mean of the two path delays.

significantly, in most manufactured dies, path P_j may have $D_{P_j} < T_{clk}$, whereas the path P_i may have $D_{P_i} > T_{clk}$.

We therefore define the *path closeness* of any two paths based on two criteria: (a) the correlation between the path delays, and (b) the mean values of the path delays. Based on this closeness metric, if any two paths are "close enough", they can be clustered together.

B. Reducing the Expense of Path Clustering

The direct use of a path closeness metric, to perform $O(N^2)$ pairwise comparisons of N potentially critical paths, is impractical as N can be very large since it involves a form of path enumeration. This leads us to the need of a procedure to reduce the computational expense associated with path clustering. In pursuit of this, we first coarsen the graph by generating *node clusters* in polynomial time. This coarsening step effectively reduces the number of paths to be enumerated to a practical number. As will be shown in Section VII-D, this enumeration takes even less computational time than a topological traversal.

At a high level, our path clustering scheme works as follows: given a levelized circuit, the function GENERATEPATHCLUSTERS(Circuit) invoked by Algorithm 1 is divided into two parts: first we call a function, GENERATENODECLUSTERS(Circuit), to be described in Algorithm 2, to generate a list of node clusters, L_{C_n}, for all critical nodes of the circuit. This step will be shown to be performed in a block-based manner. Next, we use these node clusters to extract the list of m path clusters, as described in Section VI-B, such that this step requires minimal enumeration.

C. Node Cluster Generation: Concept

1) The Node Closeness Metric: Formally, a node cluster refers to a critical connected subgraph of the original circuit graph that forms a cluster of potentially critical nodes and edges of the original circuit under a specified *node closeness* metric. The inputs of this connected subgraph come either from the PIs or from the outputs of some other node cluster, and whose outputs go either to the critical POs or to the inputs of some other node cluster. The node cluster is abstracted as a set of input-to-output connections connected by *node cluster arcs*; by definition, a node cluster arc represents a subpath of some potentially critical path of the circuit. An illustration of a node cluster is shown in Fig. 6.

We first begin with a definition of *node closeness* of two critical nodes, which if sufficiently high, allows for the two nodes to be clustered together. Since our goal is to reduce path clustering computation by development of node clusters, this definition should be constructed in a way that, although the metric is defined for nodes, it also somehow reflects on the closeness of the set of paths that pass through these two nodes. In other words, the node closeness metric must capture information about path delays of all potentially critical paths passing through the node. In this context, we define the metric $\mathcal{M}$ for the output port of a potentially critical gate G as:

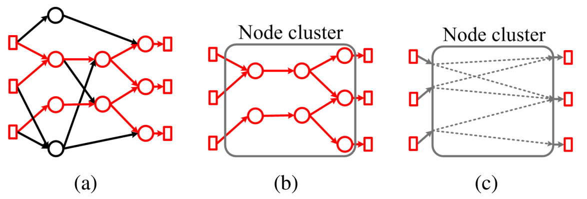


Fig. 6. The concept of a node cluster: (a) the original circuit, (b) node cluster formed with critical nodes and edges, and (c) node cluster with internal interconnections as node cluster arcs.

$$\mathcal{M}[\text{G}] = \text{AT}[\text{G}] + (T_{clk} - \text{RT}[\text{G}]) = T_{clk} - \text{ST}[\text{G}] \qquad (3)$$

where, as usual, ST[G], RT[G], and AT[G] are respectively, the slack, required time, and arrival time at the output of G. As before, these are statistical quantities represented in canonical SSTA form.

The term, AT[G], represents the statistical maximum of the delays of all paths from PIs up to G, while the term, $(T_{clk} - \text{RT}[\text{G}])$, represents the statistical maximum of the delays of all paths from G upto the POs. Therefore, $\mathcal{M}$[G] captures the statistical maximum delay over all paths passing through G.

It is important to note that unlike the deterministic case, in which $\mathcal{M}$[G] will be able to capture *only* the longest path passing through G, in the statistical case, this formulation captures delays over *all* potentially critical paths through G.

We now use this metric to define the closeness between two nodes, in terms of (a) the correlation, and (b) the mean of $\mathcal{M}$ for the two nodes. A high correlation of $\mathcal{M}$ for the two nodes implies that the delays of paths passing through the two nodes may also be well correlated. Further, similar mean values of $\mathcal{M}$ implies that these paths may also have similar mean values. This is important in the light of our previous discussion in Section V-A.

For nodes n_i and n_j, we formulate the closeness metric as:

$$\text{Closeness } \mathcal{C}(n_i, n_j) = 1, \text{ } \textit{if} \qquad (4)$$

$$\rho(\mathcal{M}[n_i], \mathcal{M}[n_j]) \geq \rho_{th}, \qquad (5)$$

$$\text{and, } f_\mu(n_i, n_j) \geq f_{\mu,th} \qquad (6)$$

where ρ is the correlation coefficient, ρ_{th} and $f_{\mu,th}$ are user-defined thresholds. By definition, both ρ and $f_{\mu,th}$ lie in the interval $[0, 1]$. Further,

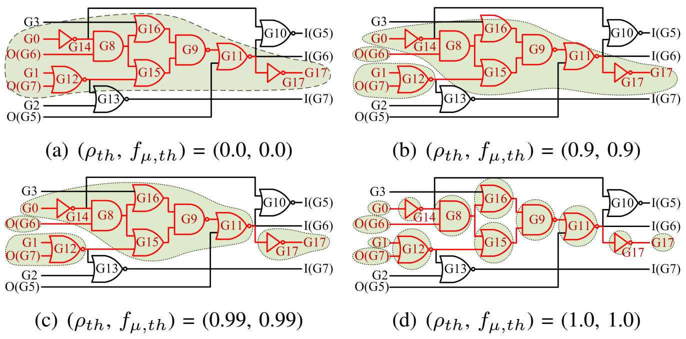
$$f_\mu(n_i, n_j) = \frac{\min\{\mu(\mathcal{M}[n_i]), \mu(\mathcal{M}[n_j])\}}{\max\{\mu(\mathcal{M}[n_i]), \mu(\mathcal{M}[n_j])\}} \qquad (7)$$

The above node closeness metric states that two nodes are close enough to be clustered if the correlation between $\mathcal{M}$ for the two nodes is sufficiently high (Equation (5)), and if the mean value of $\mathcal{M}$ for the two nodes are similar (Equation (7)).

With $\rho_{th} = f_{\mu,th} = 0$, the conditions in Equations (5) and (6) will always be satisfied: the minimum correlation between any two $\mathcal{M}$ values can be 0, and the mean of $\mathcal{M}$ can have a minimum value of 0 (since $\mathcal{M}$ represents path delay, which cannot be negative). Hence, all potentially critical nodes are forced to be clustered in a single node cluster, corresponding to the pessimistic approach in Section III-A.

For ρ_{th} = 1 and $f_{\mu,th}$ = 1, the condition of Equations (5) and (6) will be most likely violated by every pair of gates in a practical circuit, as due to variations, it is very unlikely that two nodes will have $\mathcal{M}$ to be exactly of the same value on the same die. Each node cluster may therefore be classified as a node cluster in itself. This reaches to the other extreme of path enumeration discussed in Section III-B.

Any value in (0, 1), therefore, for both ρ_{th} and $f_{\mu,th}$ corresponds to the clustered solution with different degrees of pessimism. We elaborate in more detail on the considerations for an appropriate choice of threshold values in Section VII.



G3
G0
O(G6)
G1
O(G7)
G2
O(G5)
G14
G8
G16
G15
G12
G13
G9
G11
G10
G17
I(G5)
I(G6)
I(G7)
(a) $(\rho_{th}, f_{\mu,th})$ = (0.0, 0.0)
(b) $(\rho_{th}, f_{\mu,th})$ = (0.9, 0.9)
(c) $(\rho_{th}, f_{\mu,th})$ = (0.99, 0.99)
(d) $(\rho_{th}, f_{\mu,th})$ = (1.0, 1.0)

Fig. 7. Results of node cluster generation for ISCAS89 benchmark s27, for four different values of $(\rho_{th}, f_{\mu,th})$.

2) Example: The above observations are depicted by the results of the application of this metric on an ISCAS89 benchmark circuit, s27, as shown in Fig. 7 (the clustering algorithm will be described in Section V-D). We observe that:

- With $(\rho_{th}, f_{\mu,th})$ = (0.0, 0.0), all potentially critical gates are clustered into a single node cluster.
- With $(\rho_{th}, f_{\mu,th})$ = (0.9, 0.9), the pessimism decreases, and the result is three node clusters in all.
- As the threshold values reach closer to 1.0, the pessimism further decreases and we obtain four node clusters with $(\rho_{th}, f_{\mu,th})$ = (0.99, 0.99).[2]
- When both thresholds are set to 1.0, we see that each node in itself becomes a node cluster, confirming our earlier observation.

D. A Block-Based Algorithm for Node Cluster Generation

Based on the above metric and a measure of node closeness, we now illustrate our block-based nonenumerative procedure for generating node clusters through an example. We then formalize this procedure into an algorithm.

Our approach iteratively grows a cluster by topologically traversing the circuit graph backwards from the POs to the PIs. An atomic operation consists of examining an already clustered node G (as illustrated by an example in Fig. 8), and determining whether the fanins nodes, F1, F2, and F3, of G are close enough to be clustered with G, based on Equation (4). In this example, both F2 and F3 satisfy the criterion, and the node cluster is grown to include F2 and F3 in the cluster. Next, the most recently added nodes are compared with their as-yet-unclustered fanins, and the process continues until the cluster grows no further. Any inputs that could not be added to the current cluster are used to seed new clusters, and the method continues until all clusters have been grown.

[2]With larger circuits, the variations are enhanced, and results in large overhead as will be shown in Section VII-B.

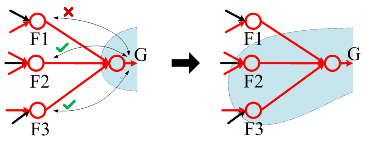


Fig. 8. Illustration of node cluster growth from a particular node in the circuit, which is already in a cluster.

Since at every step we examine only the fanins of a node, this computation can be easily performed using a block-based manner over all nodes in the circuit. This procedure is a heuristic and we do not claim it to be exact or optimal. For instance, we may test the fanout nodes for closeness, and grow the cluster. Second, our cluster growth approach is based on comparisons between a gate output and its fanin nodes, but not between the fanin nodes and existing nodes in the cluster. In general, if node n_1 is close to n_2 and n_2 is close to n_3, there is no guarantee that transitivity applies, making n_1 close to n_3. Therefore, it is possible that our method may cluster nodes more than necessary: the consequence of this is a loss in throughput. *As will be shown in Section VII, this loss is not significant under this fast heuristic.*

Algorithm 2 presents a formalized description of the node cluster generation function for a circuit: given a levelized circuit, the function generates a list, L_{C_n}, of all node clusters of the circuit, along with its input-output connections. For simplicity, a node cluster is denoted as C_n in the algorithm.

The algorithm begins with the potentially critical POs of the circuit and cluster them into node clusters based on their closeness $\mathcal{C}$ in lines 2 to 10. After this initialization, the function then grows the existing clusters in lines 11 to 34. First, all unclustered potentially critical nodes are initialized as unvisited in line 13 (a node is marked visited to indicate that it has been processed for node cluster growth). The algorithm then repeatedly picks up each unvisited potentially critical node (only once), including the POs, in an existing node cluster, in a reverse topological manner (from POs to PIs) in lines 14 to 21, and initializes a new node cluster with it if the node is not already clustered. It then examines all its unclustered, potentially critical fanin nodes in lines 22 to 33, to check if they can be included in this node cluster by performing the closeness test in line 26, effectively growing the cluster. Along with such computation, the external input and output connections of the node clusters are also updated. An input/output connection for a node cluster is created when some of its nodes are connected to either a potentially critical PI, a potentially critical PO, or to a potentially critical gate in some other node cluster.

Having obtained all the node clusters from Algorithm 2, the node cluster arc connections and arc delays for each node cluster (connections from its inputs to outputs as depicted in Fig. 6(c)) are determined using the all-pairs input-to-output delay calculation algorithm of [13], which trades off space for a very small runtime using level queues.

The complexity of Algorithm 2 is O($K^2 + N_g$): K was defined as the number of POs in Algorithm 2, and N_g are the number of gates in the levelized circuit. The first term, K^2

Algorithm 2: GENERATENODECLUSTERS

```
/* Algorithm for node cluster generation */
Input: Circuit: a levelized circuit without node clusters
Output: $LC_n$: list of node clusters with input-output
        connections
$K \leftarrow$ number of POs in the circuit
$LC_n$.Clear() for each critical unclustered PO$_i$, $1 \le i \le K$ do
    $C_n \leftarrow$ new node cluster initialized with PO$_i$
    $LC_n$.Insert($C_n$)
    PO$_i \leftarrow$ output connection of $C_n$
    for each critical unclustered PO$_j$, $i < j \le K$ do
        if closeness(PO$_i$, PO$_j$) = 1 then
            $C_n$.Insert(PO$_j$)
            PO$_j \leftarrow$ output connection of $C_n$

// Node cluster growth, beginning with POs
$l \leftarrow$ number of topological levels in the circuit
G.visited $\leftarrow 0 \;\forall$ critical nodes G
for each critical node G at level $l \ge 1$ do
    if G.visited = 0 then
        G.visited $\leftarrow 1$
        if G is not already clustered then
            $C_n \leftarrow$ new node cluster with G
            $LC_n$.Insert($C_n$)
            G $\leftarrow$ output connection of $C_n$
        else $C_n \leftarrow$ G's node cluster
        for each critical fanin F of G do
            if $F \in C'_n \neq C_n$ then
                F $\leftarrow$ input connection of $C_n$, output
                connection of $C'_n$
            else
                if closeness $\mathcal{C}$(G, F) = 1 then
                    $C_n$.Insert(F)
                    if F is critical PI then
                        F $\leftarrow$ input connection of $C_n$
                else if all critical fanouts of F have been
                visited then
                    $C'_n \leftarrow$ new node cluster with F
                    $LC_n$.Insert($C'_n$)
                    F $\leftarrow$ input connection of $C_n$, output
                    connection of $C'_n$
    $l = l - 1$;
return $LC_n$
```

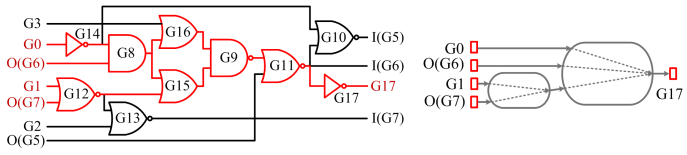
comes from performing the pairwise closeness test of all K POs. The second term is the complexity of visiting each node in the levelized circuit in a reverse topological manner.

VI. GENERATING PATH CLUSTERS AND VAHL

Having presented the theory of node clusters in Section V, we now present the link between node clusters and path clusters in this section.

A. The Relation Between Node Clusters and Path Counts

Our starting point is a coarsened circuit with node clusters, illustrated in Fig. 9(b), where the blocks represent node clusters as in Fig. 6. These node clusters are externally connected to other node clusters through their input-output ports (determined by Algorithm 2), through external interconnections between these node clusters, which are simply a subset of all interconnections of the original uncoarsened circuit (the rest of the interconnections are present within the node clusters). If we begin from the PIs, we can traverse the coarsened circuit along the external connections of the node clusters to reach to the POs. The coarsened circuit of Fig. 9(b) is formed from the original s27 circuit in Fig. 9(a), with (ρ_{th}, $f_{\mu,th}$) = (0.9, 0.9). Note that all nodes and edges of the coarsened circuit are potentially critical, since node clusters only include potentially critical nodes.



(a) Uncoarsened s27 circuit (b) Coarsened s27 circuit

Fig. 9. Illustration of a coarsened circuit.

Each internal arc of a node cluster from an input port I$_k$ to an output port O$_l$ of that node cluster, captures in itself, potentially many ($\geq$ 1, to be more specific) partial paths from I$_k$ to O$_l$ of the uncoarsened circuit, that lie within the node cluster.[3] A path, $P_{c,i}$, in the coarsened circuit, being constituted by multiple such arcs, therefore, encapsulates $1 \le n_i \le N$ potentially critical paths of the uncoarsened circuit in itself, where N is the total number of potentially critical paths in the uncoarsened circuit. *Therefore, each path, $P_{c,i}$, in the coarsened circuit is a cluster of n_i potentially critical paths of the uncoarsened circuit, with $1 \le n_i \le N$.*

If each of the arcs lying on $P_{c,i}$ encapsulates only one partial path of the uncoarsened circuit, it implies that $P_{c,i}$ is a path cluster that contains only one potentially critical path of the uncoarsened circuit, resulting in $n_i = 1$. This was observed to occur in Fig. 7 for all paths $P_{c,i}$ with one of the extremes: ρ_{th} = 1 and $f_{\mu,th}$ = 1: each node in the uncoarsened circuit becomes a node cluster in itself. For the other extreme, ρ_{th} = 0 and $f_{\mu,th}$ = 0, there exists only one node cluster, as discussed in Section V-C, and the number of paths is upper-bounded by the product of the number of PIs and the number of POs.

In other words, the number of paths in the node clusters is no more than, and often substantially less than, the original number of paths, N. For our example of the small s27 circuit shown in Fig. 7, this corresponds to reducing 6 potentially critical paths in the uncoarsened circuit to 4 paths of the coarsened circuit. This reduction seems to be small for s27; in larger circuits, this reduction is also large, as will be shown in Section VII.

B. Path Clustering

The number of such coarsened circuit paths, even if they are fewer than those in the uncoarsened circuit, are still seen to be appreciably large in many circuits. This can result in a large area and power overhead of the hold logics. In such a case, we proceed further to apply our second step in the reduction of the number of coarsened circuit paths, reducing

[3]Every node cluster input may not necessarily have an arc to every node cluster output.

also the number of hold logics that must be generated: if the delays of any two such paths are close enough, then they can be further clustered together.

For this purpose, we can now use and mathematically formulate a path closeness metric, similar to that introduced earlier in Equation (4), for two *paths* $P_{c,i}$ and $P_{c,j}$ in the coarsened circuit:

$$\text{Closeness } \mathcal{C}\big(P_{c,i}, P_{c,j}\big) = 1, \ \textit{if} \tag{8}$$

$$\rho\big(\text{delay}[P_{c,i}], \text{delay}[P_{c,j}]\big) \geq \rho_{th} \tag{9}$$

$$\text{and } f_\mu\big(P_{c,i}, P_{c,j}\big) \geq f_{\mu,th} \tag{10}$$

where

$$f_\mu\big(P_{c,i}, P_{c,j}\big) = \frac{\min\big\{\mu(\text{delay}[P_{c,i}]), \mu(\text{delay}[P_{c,j}])\big\}}{\max\big\{\mu(\text{delay}[P_{c,i}]), \mu(\text{delay}[P_{c,j}])\big\}} \tag{11}$$

In practice, given the reduction in the number of the paths as we move from the original circuit to the coarsened circuit, it is practical to enumerate the paths without much computational overhead and perform pairwise comparisons to cluster them.[4]

The path clustering step can therefore be formalized as follows: given the list of node clusters, L_{C_n}, from Algorithm 2, we first enumerate all paths in the coarsened circuit through a simple depth-first traversal of the coarsened circuit. Next, a pairwise comparison is performed between these paths to check if they can be clustered together by applying the closeness test of Equation (8), resulting in a greatly reduced number of clusters of coarsened circuit paths.

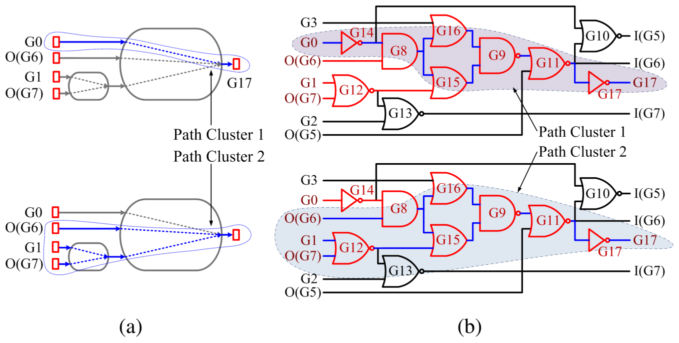


Fig. 10. Path clusters for circuit s27 in (a) the coarsened circuit and (b) the original circuit.

The application of this scheme is shown in Fig. 10 for s27 circuit. Path coarsening in the second step reduces 4 paths in the coarsened circuit further down to 2 clusters of these paths. Hence node and path clustering allows for the number of hold logics to be reduced by 3× as compared to the enumerative scheme discussed in Section III-B (which generated 6 hold logics). Again, this reduction will be seen in Section VII to be much more in larger circuits.

C. Generation of VAHL

Having obtained the coarsened circuit path clusters, we now generate a separate hold logic corresponding to each of these path clusters, by applying the hold logic generation algorithm [1], [4]; each hold logic corresponds to the sensitization criterion of all paths within the path cluster.

[4]Note that while this is computationally feasible in the presilicon phase, node clustering has only solved a part of the problem. Path clustering is still essential to reduce the hardware overhead of hold logic.

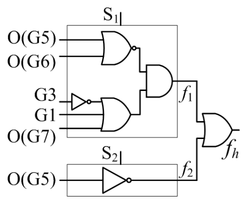


Fig. 11. VAHL generated for circuit s27.

For our running example of circuit s27, the two-output VAHL along with the sleep signals is shown in Fig. 11. These sleep signals can be selectively exercised in the postsilicon stage for various dies.

VII. Experimentation and Results

The proposed algorithms were implemented in C++, using the *MinnSSTA* [14] software for SSTA under 32nm PTM [15] models. The methods were exercised on the ISCAS89 benchmark circuits on a 3.0GHz CPU with 8GB RAM. The SSTA grid size for each circuit is taken from [10], [11]. We first present the baseline for comparison and a heuristic for suitable choice of the threshold parameters. Next, we provide a tabulation of all results, followed by a detailed analysis.

A. Baseline

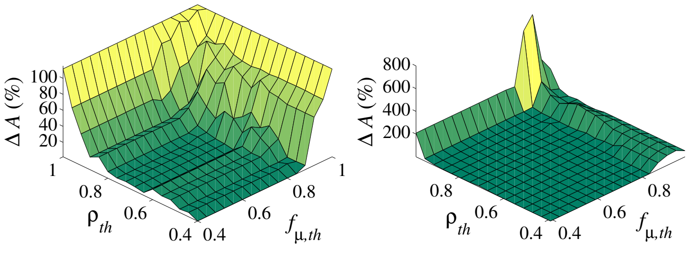
In order to evaluate the effectiveness of our clustering approach, for comparison purposes, we choose the pessimistic VLU design (Section III-A, $\rho_{th} = f_{\mu,th} = 0$) as the baseline. Recall that the pessimistic VLU has only one hold logic, and identifies all potentially critical paths as two-cycle paths in the manufactured chip if the clock constraint is violated.

In the discussion to follow, we will discuss the results for overhead in area and power (denoted as ΔA (%) and ΔP (%), respectively) as compared to this baseline. These overhead arise due to a higher number of hold logics (less pessimism) in the clustered approach. Further, the resultant throughput enhancements are denoted as $\Delta\eta$ (%).

B. Effects of Varying Threshold Values

The process of node and path cluster generation is quite sensitive to the values of ρ_{th} and $f_{\mu,th}$, as discussed in Sections V-C and VI. We recall that a good choice of $\rho_{th}, f_{\mu,th} \in (0,1)$ should give us $\Delta\eta$ value to be as close to (or as high as) the enumerative VLU design, and the ΔA values to be as close (or as low) as the pessimistic VLU design.

To investigate such a possibility, we plot the trends of ΔA against $(\rho_{th}, f_{\mu,th})$ for two ISCAS89 circuits, s1196 and s9234, in Fig. 12, with $(\rho_{th}, f_{\mu,th})$ being varied from (1, 1) to (0.4, 0.4). The value of ΔA is observed to be very high at (1, 1), confirming that the enumerative approach is very expensive. The overhead decreases and becomes very small when $\rho_{th}, f_{\mu,th}$ values are decreased to 0.6 and further down to 0.4, implying that at this point, the design has become as pessimistic as the baseline. However, some "knee" points may be identified, where ΔA is low (from 5% to 15%). This is observed to be true for all the circuits in general (the threshold values at which such knee points are observed may differ).



(a) s1196 (b) s9234

Fig. 12. Variation of ΔA with $(\rho_{th}, f_{\mu,th})$ for s1196 and s9234.

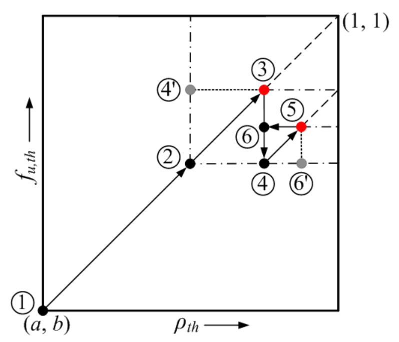


Fig. 13. Bisection based approach for a suitable $(\rho_{th}, f_{\mu,th})$ choice.

The trend of ΔA with $(\rho_{th}, f_{\mu,th})$ is not strictly monotonic; its value may be little higher for lower threshold values. VAHL area depends not only on the number of hold logics, but also their logical complexity. A single hold logic may contain a large number of terms in its (minimized) logical expression, hence requiring a large number of gates to realize such logic. Compared to this, a number of hold logics with only a few terms in their (minimized) logical expressions may need only a small hardware for logic implementation.

For threshold values less than 0.65, ΔA becomes very small. It is therefore useful to explore potential points in the search space contained within the points (0.65, 0.65) and (1, 1).

C. Choice of Threshold Values: Bisection-Based Heuristic

A naive approach to determine a $(\rho_{th}, f_{\mu,th})$ point, that is as close to (1, 1) as possible and also results in a low area overhead, can be to evaluate ΔA and the number of path clusters (separate hold logics) for multiple combinations of $(\rho_{th}, f_{\mu,th})$ between (0, 0) and (1, 1). This approach, however, involves evaluations at many $(\rho_{th}, f_{\mu,th})$ points, and is therefore expensive in runtime (we use the term "evaluation" to refer to the generation of node and path clusters, and the corresponding hold logics).

We therefore employ a fast bisection-based heuristic (described shortly) to determine a *feasible point* that satisfies the following criterion for feasibility: (a) offers less than 15% area overhead compared to the baseline, (b) results in generation of less than 10 hold logics in total. Criteria (a) enables the search to reach close to the "knee" points observed in Fig. 12, maintaining a low area overhead. Criteria (b) ensures that the number of path clusters are within a certain bound, as larger number of path clusters imply increased runtime. Further, we add another restriction to the heuristic, of allowing a maximum of 6 iterations, again, to keep the runtime within bounds.

As will be shown in Section VII-D, this heuristic has a very small runtime. We do not claim this to give us an optimal point with lowest area overhead and high throughput, but is observed to yield good results.

The overall idea of the heuristic is simple, and is shown through the example in Fig. 13. The search begins with an initial evaluation at point (a, b) (chosen as (0.65, 0.65) for our method, following our discussion in Section VII-B), marked as ① in Fig. 13. The search then tries to move closer to (1, 1) by bisecting the diagonal and performing an evaluation at the midpoint of the diagonal (marked as point ② in our example). Once a midpoint is reached, the following options are considered to proceed further:

- The midpoint is feasible (according to the criterion described above). For example, point ② is a feasible point in Fig. 13. In that case, the search proceeds further towards (1, 1) by again bisecting the remaining part of the diagonal, reaching to point ③.
- The midpoint is infeasible. For example, after reaching to point ③, the search finds it infeasible. In that case, the search goes off-diagonal to either of the corner points ④ or ④’ (the choice is random). In the running example, the search moves to point ④.
  If point ④ is feasible (as is so in this case), the search then again proceeds in a 45° direction along a new diagonal of the reduced square, towards point ⑤, in exactly the same way as it had proceeded from point ①. If point ④ is infeasible, the search goes to the other choice: point ④’ and tries to proceed in the same way as from point ④.
- The midpoint and both the off-diagonal corner points (points ③, ④, and ④’) are infeasible. Then the last encountered feasible point (in this example, point ②) will be reported as the suitable $(\rho_{th}, f_{\mu,th})$ point.

D. Runtime

We now present the runtime for the above heuristic in Table I for ISCAS89 circuits listed in C1. This tabulation helps us to evaluate the runtime of node and path clustering steps (each iteration of the heuristic involves node and path clustering). For each of these circuits, the runtime for SSTA is listed in C2, for comparison with the runtime of the node and path clustering steps, as discussed shortly.

Columns C3–C10 present the results for heuristic. C3 and C4 list the final values of $(\rho_{th},\ f_{\mu,th})$ obtained from the heuristic. C5–C7 then list the time taken by the heuristic for *all* iterations to arrive at these threshold values, shown separately for node and path clustering steps in C5, C6, and also their sum total in C7 (within roundoff errors).

Further, to get an estimate of the runtime for each iteration (a single node and path clustering step), C8–C10 list the average runtimes (overall runtime / number of iterations), again separately for the node and paths clustering steps in C8, C9, and the summation in C10.[5]

The heuristic needed 6 iterations for each of the circuits. For circuits other than s5378, this happens as only a maximum of 6 iterations are allowed (otherwise the heuristic would have a

[5]The exact runtime for node and path clustering step depends on the values of $(\rho_{th},\ f_{\mu,th})$; higher thresholds generally result in a higher runtime. Evaluating the average gives a balanced estimate, and is therefore useful.

TABLE I
RUNTIMES (IN SECONDS) FOR THE BISECTION-BASED HEURISTIC

Circuit	SSTA	Heuristic							
		ρ_{th}	$f_{\mu,th}$	Runtime for Node and Path Clustering					
				Overall			Average		
				Nodes	Paths	Total	Nodes	Paths	Total
C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
s27	0.0	0.87	0.91	0.0	0.0	0.0	0.0	0.0	0.0
s1196	0.7	0.78	0.87	0.1	2.7	2.8	0.0	0.4	0.5
s5378	7.4	0.91	0.91	0.4	0.5	0.8	0.1	0.1	0.1
s9234	14.6	0.82	0.91	4.0	32.1	36.1	0.7	5.4	6.0
s13207	74.3	0.91	0.87	6.4	151.6	158.0	1.1	25.3	26.3
s15850	89.6	0.87	0.91	4.5	24.6	29.2	0.8	4.1	4.9
s35932	229.8	0.96	0.96	2.3	0.9	3.2	0.4	0.1	0.5
s38417	207.2	0.82	0.91	7.4	111.6	119.0	1.2	18.6	19.8
s38584	233.0	0.91	0.96	1.8	0.2	2.0	0.3	0.0	0.3

higher number of iterations and hence, a higher runtime). For s5378, the case of midpoint and both the off-diagonal corner points being infeasible is encountered, and the heuristic needed exactly 6 iterations.

It is observed that the overall runtime of the heuristic (in C5–C7) is in general very small; less than half a minute for all the circuits, except s13207 and s38417. For both these circuits, the sizes of node and path clusters are large. Thus, both the node and path clustering steps take more runtime for these circuits compared to other circuits.

The small runtimes of the path clustering step indicate that our intermediate node clustering step (which also has a very small runtime) is very effective in reducing the number of paths that need to be enumerated in the coarsened circuit. Direct path enumeration in the uncoarsened circuit can otherwise be very expensive.

Further, the path clustering step involves path enumeration in the coarsened circuit, and hence, in general, contributes more to the overall runtime than the node clustering step. However, it is smaller for s35932 and s38584 due to a small number of coarsened circuit paths (the exact numbers are described shortly). For 35932, this happens as there are only a few potentially critical paths. For s38584, the degree of correlation is quite high, which enables the encapsulation of a large number of potentially critical paths of the uncoarsened circuit in only a small number of coarsened circuit paths (this follows from our discussion in Section VI-A). The runtime of clustering steps, therefore, does not necessarily depend on the size of the circuit.

The average runtimes listed in C8–C10 give an estimate of the runtime per iteration, i.e., for a single node and path clustering step. For all the circuits, the node clustering step takes less than 1-2 seconds on an average. The path clustering step takes less than half a minute for all circuits, (less than 10 seconds in most cases). Further, the total average runtime in C10 is significantly smaller than the runtime for SSTA listed in C2 for all the circuits. This is intuitive as the node clustering step involves a topological traversal only on the potentially critical gates (a subset of all the gates traversed in SSTA). Also, extending our previous discussion, even though the path clustering step involves a path enumeration, the intermediate node clustering step makes it even cheaper than a topological traversal of the SSTA.

E. Tabulation of Results

Having obtained a suitable (ρ_{th}, $f_{\mu,th}$) point, we use this for our clustering approaches and present the results in Table II. For convenience, we have marked the k^{th} column in the table as Ck. We first discuss the details of Table II, and then analyze the results.

TABLE II
RESULTS FOR VAHL UNDER THE CLUSTERED (ρ_{th}, $f_{\mu,th} \neq 0$) AND ENUMERATIVE (ρ_{th} = $f_{\mu,th} = 1$) APPROACHES.

Circuit	ρ_{th}	$f_{\mu,th}$	Results from Clustering				VAHL Results				
			$\lvert L_{C_n} \rvert$	N	N_C	m	ΔA	$\Delta P(\%)$		$\Delta\eta(\%)$	
							(%)	μ	σ	μ	σ
C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
s27	0.87	0.91	4	12	8	2	11.7	6.6	5.3	12.1	11.5
	1	1	16		12	12	105.3	21.5	19.5	12.4	9.3
s1196	0.78	0.87	16	656	558	4	14.5	10.8	2.3	25.5	15.4
	1	1	337		656	656	109.2	41.2	22.5	38.5	9.5
s5378	0.91	0.91	7	520	121	6	-5.6	-2.2	1.2	27.3	20.9
	1	1	157		520	520	47.5	22.5	27.5	29.5	20.5
s9234	0.82	0.91	42	119556	628	5	4.7	2.1	1.6	6.8	8.6
	1	1	1368		119556	–	–	–	–	–	–
s13207	0.91	0.87	12	189666	689	2	0.3	-0.2	0.3	24.3	15.0
	1	1	1126		189666	–	–	–	–	–	–
s15850	0.87	0.91	3	2.6e7	612	2	1.4	-1.0	0.5	2.8	0.7
	1	1	1383		2.6e7	–	–	–	–	–	–
s35932	0.96	0.96	10	174	60	4	-0.7	-0.3	0.3	1.0	1.0
	1	1	1185		174	174	100.5	37.4	28.1	1.2	0.9
s38417	0.82	0.91	13	38337	1931	5	2.6	-1.0	2.1	16.0	6.3
	1	1	1513		38337	–	–	–	–	–	–
s38584	0.91	0.96	4	12172	14	3	0.2	-0.2	0.1	20.1	6.5
	1	1	383		12172	–	–	–	–	–	–
Average	Clustered						3.3	1.6	1.5	15.1	9.5
	Enum.*						90.6	30.6	24.4	20.4	10.0

* Enumerative scheme becomes prohibitive for larger circuits; the average for this method is taken only over circuits which allow this.

1) Clustering parameters: Columns C2 and C3 list the values of the threshold parameters, ρ_{th} and $f_{\mu,th}$, that are used in determining the closeness of nodes and paths. For comparison, we show two sets of results for each circuit:

- Using clustering (Section III-C, $\rho_{th}, f_{\mu,th} \neq 0$), obtained through the heuristic of Section VII-C.
- Using enumeration (Section III-B, $\rho_{th} = f_{\mu,th} = 1$).

2) Clustering results: The results for our clustering algorithms are shown in C4–C7. Column C4 presents the number of node clusters, $|L_{C_n}|$, generated in Algorithm 2. Columns C5 and C6 list, respectively, the number of potentially critical paths, N, in the uncoarsened circuit[6] and the number of paths, N_C, in the coarsened circuit. The number of coarsened circuit path clusters, m, generated in Section VI-B, are then listed in C7 (for m path clusters, we need m separate hold logics).

Computations for enumerative approach become prohibitive as the number of paths increases, and hence the results could not be obtained for circuits with large value of N.

3) VAHL area, power, and throughput: In C8 to C12, we show the results for overhead in area and power (ΔA (%) and ΔP (%)) compared to the baseline, as discussed earlier in Section VII-A. We recall that for the clustered and enumerative approaches, only a subset of m hold logics will be active in postsilicon stage in a particular chip. Therefore, although ΔA

[6]While the critical path enumeration can be exponential in the number of nodes, the number of critical paths can be computed in linear time.

remains the same for all chips, ΔP will be proportional to the area overhead of the active hold logics in a particular chip, and may differ for different chips. ΔP can therefore be captured as a distribution over L samples of Monte Carlo simulations. Choosing $L = 10000$ as in [10], we tabulate the mean (μ) and standard deviation (σ) of this distribution in C9 and C10. By the same logic, $\Delta\eta$ will also depend on the set of active hold logics in a particular chip, and can be captured as a distribution over the L Monte Carlo samples. C11 and C12 list the μ and σ values for $\Delta\eta$ distribution.

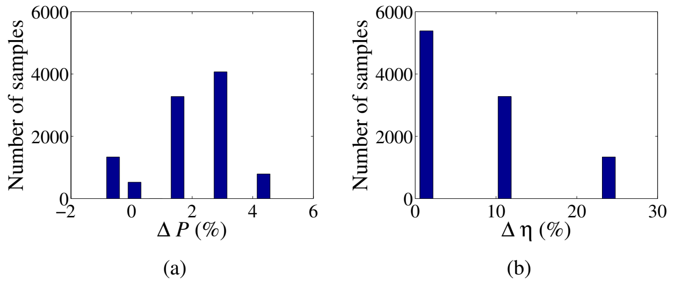


Fig. 14. The distribution of percentage (a) ΔP and (b) $\Delta\eta$, over all the L = 10000 Monte Carlo samples for circuit s9234.

In interpreting these results, it is important to note that the distributions of ΔP and $\Delta\eta$ are ***not*** Gaussian, but can be quite skewed, as shown for an example benchmark, s9234, in Fig. 14. This is because a variation may cause a hold logic to be activated or disactivated; large bars correspond to frequently-activated hold logics, that correspond to paths that are frequently critical. Even more skewed distributions are seen for other benchmarks, e.g., s38417. Therefore, the fact that $\mu - 3\sigma$, for example, is negative, should not be misinterpreted to mean that the distribution necessarily shows samples with negative power or throughput overhead. In fact, we observed for these circuits, $\Delta\eta$ *is positive over all chips.*

We may also see many samples having zero ΔP, $\Delta\eta$ values; in such samples, hold logic is not exercised as T_{clk} is not violated. Further, Fig. 14 shows some samples with negative values of ΔP; the reason for this will be discussed in Section VII-F3.

F. Analysis and Discussion

1) Gains through clustering: We first discuss the results of clustering algorithms presented in C4–C7. As discussed in Section III-A. The pessimistic baseline will always be characterized by one node cluster. Compared to this, the clustered scheme has a slightly higher number of node clusters, and the enumerative scheme has substantially higher numbers, indicating progressively reduced degrees of pessimism.

Comparing C5 and C6, we find that node cluster generation reduces a large number of potentially critical paths in the uncoarsened circuit (N), up to even 1000×, resulting in a small number of paths in the coarsened circuit (N_C). This shows that our node clustering technique is very effective in reducing the number of paths. The enumerative scheme offers no reduction and becomes prohibitive for large values of N.

For the clustered scheme, N_C is still sufficiently high for most circuits, indicating that it is impractical to maintain a separate hold logic for each path in the node-coarsened circuit. Our second step of path coarsening (Section VI-B) further reduces this number, and generates a set of m hold logics for VAHL listed in C8: it is seen that for our clustered approach, $m \leq 6$ over all the circuits. This is again a significant reduction and hence very beneficial in gaining low area and power overhead, as discussed next. On the other hand, the enumerative scheme does not allow any path clustering.

2) Area overhead: Our first observation is that for the clustered VLU, the increase in area over the pessimistic VLU is appreciably small: 3.3% on an average. This is a significant reduction compared to 90.6% overhead incurred in the enumerative scheme (in fairness, it should be pointed out that the average area overhead for the four circuits where enumeration is feasible is 4.9%). We further notice that for some circuits (such as s5378, s35932), ΔA is negative, implying less area than the pessimistic VLU. As discussed earlier in Section VII-B, this can happen as VAHL area depends not only on the number of hold logics (which are higher in the clustered VLU compared to the baseline), but also their logical complexity.

For the clustered approach, a few circuits (s35932, s38584) give an appreciably low area overhead even with high threshold values. This means that there is a high degree of correlation in such circuits, offering greater potential for the use of our clustering method.

3) Power overhead: As stated earlier, ΔP in C9, C10 is proportional to the area overhead of the active hold logics, and is therefore observed to be always less than or equal to the total ΔA in C8. Over all the circuits, ΔP is only about 1.6% on an average.

We also observe that although ΔA is positive for clustered scheme, ΔP is negative for nearly half of the circuits. This implies a small power savings with clustered VLU design, and attributes itself to reduced pessimism: for the baseline pessimistic case, all of the hold logic is active in a particular chip when the T_{clk} constraint is violated, whereas in the clustered scheme, only a subset of the hold logics is active, which may dissipate less power than the pessimistic VAHL. For different circuits therefore, sometimes the pessimistic schemes wins, and sometimes the clustered scheme wins (and sometimes the one-cycle case may win, when T_{clk} is met).

For the enumerative case, both area and power overhead are quite high even for small circuits.

4) Throughput enhancements: We now analyze the throughput values in C11, C12. On an average, the clustered scheme offers high throughput enhancements, with a mean value of 15.1% across all the chips, approaching quite close to the 20.4% value for the enumerative case. Given the low area/power overhead of the clustered scheme, our "Middle Way" approach for clustering is therefore very beneficial in offering the desirable aspects of the pessimism and enumeration extremes: low overhead and close-to-maximum throughputs.

An exception occurs for circuit s35932: $\Delta\eta$ values are low and similar for both approaches; this indicates that this circuit has most of its potentially critical paths with near-critical delays (similar observations are also discussed in [3], [5]), and all such paths will be moved to second cycle. Clustering (or even path enumeration) therefore will not show much benefit.

G. Validation of Our Scheme

While we attempt to be less pessimistic than the pessimistic approach, a functional error is possible in the postsilicon stage if a two-cycle path is wrongly predicted as a one-cycle path by our postsilicon processing described in Section IV. In this section, we validate that our algorithms have *pessimism indeed*: they do not cause any functional errors.

We first notice that a two-cycle path is identified as a one-cycle path only when at least one of the critical ports in any of the critical gates in the circuit is wrongly identified as noncritical, as such computations rely completely on the set of critical ports.

To investigate if such a scenario may arise, we perform Monte Carlo simulations with $L = 10000$ samples, and follow these steps for validation:

1) Each Monte Carlo sample corresponds to a case where we know the exact delay distribution of the circuit. We perform an STA over the circuit and determine the exact set of critical ports, and term it as $\mathrm{C}_{S,exact}$.
2) Next we simulate the clustered scheme: for each of the Monte Carlo samples generated above, the postsilicon processing steps described in Section IV are performed to identify all critical path clusters. We then determine the set of critical ports as ports that belong to gates lying on the paths of critical path clusters, termed as $\mathrm{C}_{S,clus}$.
3) In order to be functionally correct, we must have $\mathrm{C}_{S,clus} \supseteq \mathrm{C}_{S,exact}$, indicating that one-cycle paths may be predicted as two-cycle paths, but not the other way.

We test for this supersetting property for each of the circuits listed in Table II, and find that among the L samples, zero samples caused such an error, thus validating our scheme.

VIII. Conclusion

In this paper, we develop a high-performance scheme for generation of variation-aware hold logics. Delays of paths in a circuit are correlated due to spatial correlations in the circuit. Exploiting this fact, we present algorithms to generate node and path clusters in a circuit, which offer a “Middle Way” solution in between to the pessimistic and enumerative extensions of the deterministic hold logic generation scheme. We show through our results that we obtain near-maximum throughputs with very low area/power overhead.

References

[1] L. Benini, G. D. Micheli, A. Lioy, E. Macii, G. Odasso, and M. Poncino, “Automatic synthesis of large telescopic units based on near-minimum timed supersetting,” *IEEE Transactions on Computers*, vol. 48, pp. 769–779, August 1999.

[2] Y. Chen, H. Li, J. Li, and C. K. Koh, “Variable-latency adder (VL-adder): new arithmetic circuit design practice to overcome NBTI,” in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 195–200, 2007.

[3] J. Cong and K. Minkovich, “Mapping for better than worst-case delays in LUT-based FPGA designs,” in *Proceedings of the International Symposium on Field Programmable Gate Arrays*, pp. 56–64, 2008.

[4] Y. S. Su, D. C. Wang, S. C. Chang, and M. S. Malgorzata, “Performance optimization using variable-latency design style,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 19, no. 10, pp. 1874–1883, 2011.

[5] S. Gupta and S. S. Sapatnekar, “BTI-aware design using variable latency units,” in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 775–780, 2012.

[6] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge, “Opportunities and challenges for better than worst-case design,” in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 2–7, 2005.

[7] X. Liang, G.-Y. Wei, and D. Brooks, “ReVIVaL: a variation-tolerant architecture using voltage interpolation and variable latency,” in *Proceedings of International Symposium on Computer Architecture*, pp. 191–202, 2008.

[8] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, “RazorII: in situ error detection and correction for PVT and SER tolerance,” *IEEE Journal of Solid-State Circuits*, vol. 44, pp. 32–48, January 2009.

[9] S. Sapatnekar, “Overcoming variations in nanometer-scale technologies,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, pp. 5–18, March 2011.

[10] H. Chang and S. S. Sapatnekar, “Statistical timing analysis under spatial correlations,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 9, pp. 1467–1482, 2005.

[11] Q. Liu and S. S. Sapatnekar, “A framework for scalable postsilicon statistical delay prediction under process variations,” *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 28, pp. 1201–1212, August 2009.

[12] http://en.wikipedia.org/wiki/Buddhism#Middle_Way.

[13] S. S. Sapatnekar, “Efficient calculation of all-pairs input-to-output delays in synchronous sequential circuits,” in *Proceedings of the International Symposium on Circuits and Systems*, pp. IV520–IV523, 1996.

[14] H. Chang, Q. Liu, and S. S. Sapatnekar, 2009. *MinnSSTA* : http://www.ece.umn.edu/users/sachin/software/MinnSSTA/.

[15] Predictive Technology Model, 2008. http://www.eas.asu.edu/∼ptm.