Application-Specific Cross-Layer Optimization Based on Predictive Variable-Latency VLSI Design

VIVEK K. DE, Intel Labs
ANDREW B. KAHNG, University of California, San Diego
TANAY KARNIK, Intel Labs
BAO LIU, MILAD MALEKI, and LU WANG, University of Texas at San Antonio

Traditional synchronous VLSI design requires that all computations in a logic stage complete in one clock cycle. This leads to increasingly pessimistic design as technology scaling introduces increasingly significant parametric variations that result in an increasing performance variability. Alternatively, by allowing computations in a logic stage to complete in a variable number of clock cycles, variable-latency design provides relaxed timing constraints for average performance, area, and power consumption optimization. In this article, we present improved variable-latency design techniques including: (1) a generic minimum-intrusion variable-latency VLSI design paradigm, (2) a signal probability-based approximate prediction logic construction method for minimum misprediction rate at minimum cost, and (3) an application-specific cross-layer analysis methodology. Our experiments show that the proposed variable-latency design methodology on average reduces the computation latency by 26.80%(14.65%) at cost of 0.08%(3.4%) area and 0.4%(2.2%) energy consumption increase for the interger (floating point) unit of an open-source SPARC V8 processor LEON2 synthesized with a clock-cycle time between 1.97ns(3.49ns) and 5.96ns(13.74ns) based on the 45nm Nangate open cell library, while an automotive application-specific design further achieves an average latency reduction of 41.8%.

Categories and Subject Descriptors: B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; C.4 [**Performance of Systems – Fault Tolerance**]: General

General Terms: Performance, Reliability

Additional Key Words and Phrases: VLSI, CAD, optimization, better-than-worst-case VLSI design, VLSI statistical timing analysis and optimization

ACM Reference Format:
Vivek K. De, Andrew B. Kahng, Tanay Karnik, Bao Liu, Milad Maleki, and Lu Wang. 2015. Application-specific cross-layer optimization based on predictive variable-latency VLSI design. ACM J. Emerg. Technol. Comput. Syst. 12, 3, Article 21 (September 2015), 19 pages.
DOI: http://dx.doi.org/10.1145/2746341

1. INTRODUCTION

VLSI technology scaling has been the driver of the semiconductor industry for decades. However, in recent years, technology scaling has introduced increasingly significant parametric variations from the manufacturing process and at the system runtime which result in subtle signal propagation delay variations at each component, and possibly significant accumulated performance variation at the system level

This work is supported by the National Science Foundation, under grant CISE SE-1117975.
Authors' addresses: V. K. De, Intel Labs, Hillsboro, OR; A. B. Kahng, Computer Science and Engineering Department, University of California, San Diego, CA; T. Karnik, Intel Labs, Hillsboro, OR; B. Liu (corresponding author), M. Maleki, and L. Wang, Electrical and Computer Engineering Department, University of Texas, San Antonio, TX; email: bliu@utsa.edu.

DOI: http://dx.doi.org/10.1145/2746341

[Alam 2008; Benini and Micheli 2004; Borkar 2005; Ghosh and Roy 2010; West and Harris 2011]. This has created an unprecedented reliability challenge, as traditional synchronous VLSI design requires that all computation in a logic stage complete in a single clock cycle. As a result, cutting-edge VLSI systems are subject to degraded reliability (with an increasing timing error rate) and design quality (in terms of area and power consumption).

A number of adaptive or resilient techniques have been proposed for VLSI design in the presence of performance variability for timing error resilience, average performance improvement, and power consumption reduction. For example, "better-than-worst-case" design allows the critical-path delay of a combinational logic network to exceed the clock cycle time [Austin et al. 2004, 2005; Blaauw et al. 2008; Bowman et al. 2008, 2011; Das et al. 2006, 2009; Ernst et al. 2003, 2004; Fojtik et al. 2012; Kondo et al. 1997], while logic correctness is still guaranteed as long as all timing errors are identified. An identified timing error can be corrected by computation replay, for example, at an elevated supply voltage or reduced clock frequency [Blaauw et al. 2008; Bowman et al. 2011; Das et al. 2009], or by local stalling allowing a slow logic computation to complete in two clock cycles [Fojtik et al. 2012]. Such a variable-latency VLSI design provides relaxed timing constraints and enables further improvements in average performance, area, and power consumption optimization.

A critical problem is timing error identification, which can be achieved by detection techniques which check logic outputs such as in Razor logic [Austin et al. 2004, 2005; Blaauw et al. 2008; Das et al. 2006, 2009; Ernst et al. 2003, 2004; Fojtik et al. 2012] or prediction techniques based on logic inputs such as in a telescopic unit [Neres et al. 2009; Benini et al. 1998, 1999; Su et al. 2011]. Timing error detection has certain performance cost as it is performed after logic completion. The existing timing-error-prediction-based variable-latency design techniques have only been applied to small benchmark circuits such as ISCAS'89 and SIS [Neres et al. 2009; Benini et al. 1998, 1999; Su et al. 2011].

Our contributions in this article are as follows.

(1) We present a generic variable-latency design paradigm including a minimum prediction logic network, a small FSM which resets the timing error signal in the subsequent clock cycle once it rises, and a clock gate.
(2) We present an approximate timing error logic construction algorithm considering timing-critical-path side input signal probabilities for minimum misprediction rate at minimum cost.
(3) We present a cross-layer statistical analysis methodology based on behavior/architecture-level simulation and gate-level *signal-probability-based statistical timing analysis* (SPSTA) [Liu 2008].
(4) Our experiments show that the proposed variable-latency design methodology on average reduces the computation latency by 26.80%(14.65%) at the cost of 0.08%(3.4%) area and 0.4%(2.2%) energy consumption increase for the interger (floating point) unit of an open-source SPARC V8 processor LEON2 synthesized with a clock-cycle time between 1.97ns(3.49ns) and 5.96ns(13.74ns) based on the 45nm Nangate open cell library, while an automotive application-specific design further achieves an average latency reduction of 41.8%.

The rest of this article is organized as follows. We briefly review the nanoscale VLSI design reliability challenge and the existing resilient VLSI design techniques in Section 2. We propose our cross-layer variable-latency design methodology in Section 3, and present our experimental results in Section 4. We conclude in Section 5.

2. BACKGROUND

2.1. Nanoscale VLSI Design Challenge

Nanoscale VLSI systems are subject to increasingly prevalent manufacturing defects, process variations, and system runtime parametric variations such as on-chip temperature, power supply voltage, aging effects, and uncertainties such as radiation and cosmos ray strikes. Catastrophic manufacturing defects lead to permanent faults. System runtime uncertainties may create soft errors. More subtle process variations and system runtime parametric variations lead to system performance variation which may accumulate and result in timing errors. Parametric variations may also degrade reliability and reduce system lifetime [Alam 2008; Benini and Micheli 2004; Borkar 2005; Ghosh and Roy 2010; West and Harris 2011].

Such parametric variations cannot be reduced below certain levels due to the uncertainty principle in quantum physics which dominates at nanometer scale. As a result, improvement in manufacturing technology cannot solve the problem alone, and design techniques are necessary to achieve reliable nanoscale VLSI systems. Of particular interest are timing errors resulting from performance variations, which have become a common problem in recent technology nodes. As technology scales, average performance improves while the performance variation increases. The traditional guardbanding-based synchronous design methodology requires a smaller worst-cost critical-path delay in a combinational logic network than the clock-cycle time, which is increasingly pessimistic, leading to an increasing performance cost without logic correctness guarantee.

2.2. Resilient VLSI Design Techniques

Asynchronous design achieves timing error resilience and performance scaling [Furber and Day 1996; Muller and Bartky 1959; Singh and Nowick 2007; Sutherland 1989]. However, lack of design automation tools prevents its practical use [Hauck 1995; Sparso and Furber 2001]. Alternatively, variable-latency design is a relatively simple synchronous design paradigm wherein logic computation is allowed to complete in a flexible number of clock cycles based on a timing error signal. This allows a clock-cycle time to be less than the worst-case signal propagation path delay in a combinational logic network, that is, achieving a better-than-worst-case design. With a tiny probability of timing error occurrence, variable-latency design further achieves average performance or throughput improvement.

A variable-latency design may include a logically incomplete or fast approximate logic network besides a complete or slow exact logic network, and a multiplexer which selects one of the two logic networks based on a timing error signal [Kondo et al. 1997; Lu 2004]. Or, a logically complete logic network can be simply over-clocked (operate under a higher clock frequency than allowed by the critical-path delay), forming a temporally incomplete logic network in which logic correctness is guaranteed by clock gating based on a timing error signal [Kelly and Phillips 2005; Liu et al. 2012]. Based on a timing error signal, a microprocessor may flush its instruction pipeline and rerun the computation at an elevated power supply voltage [Blaauw et al. 2008; Das et al. 2009] or a reduced clock frequency [Bowman et al. 2011]. Or, a timing error signal can be included in pipeline control logic [Fojtik et al. 2012].

A critical problem is to generate a timing error signal. This problem can be reduced to that of generating a logic computation completion signal in asynchronous design. The logic computation completion signal needs to be generated as soon as possible after the logic computation is complete while, in generating a timing error signal, one only needs to make a binary decision on whether the current logic computation leads to a

timing error, and the timing error signal needs to be generated within the current clock cycle.[1]

The existing logic computation completion signal generation techniques in asynchronous design can be leveraged for generating a timing error signal in variable-latency design, which is, however, too costly. Except for certain specific arithmetic components, logic computation completion is detected in asynchronous circuits by encoding the logic outputs in a delay-insensitive code [Bose and Rao 1982; Jha and Wang 1993; Verhoeff 1988] such as *m*-hot code or dual-rail code, constructing a logic network which allows only unidirectional (e.g., rising) signal transition (as in Domino logic), and checking whether the logic outputs a legal codeword [Mago 1973]. This leads to more than double gate count [Liu et al. 2012].

A timing error can be detected more cost efficiently based on time domain redundancy. In Razor logic [Austin et al. 2004, 2005; Blaauw et al. 2008; Das et al. 2006, 2009; Ernst et al. 2003, 2004; Fojtik et al. 2012] and Intel *error detecting sequential* (EDS) design [Bowman et al. 2008, 2011], logic outputs are sampled at two different time spots for each clock cycle by a flip-flop and a latch, respectively. Any mismatch between the flip-flop and latch signals indicates a timing error. This, however, comes with a performance cost because no signal transition is allowed in the error detection timing window.

While timing error detection is based on checking the combinational logic outputs, timing error prediction is based on the combinational logic inputs [Benini et al. 1998, 1999; Su et al. 2011]. For cost, part of the existing logic network may be reused, that is, timing error prediction can be based on signals at some internal nodes in an existing logic network (which is alternatively called timing error detection in Neres et al. [2009]).

For example, in a ripple-carry adder, the carry propagation chain is the timing-critical path. The number of bits that carry propagation crosses determines the computation latency, which is based on the inputs. For each bit i of the input operands a and b, the carry propagation signal is given by $p_i = a_i + b_i$, while the carry generation signal is given by $g_i = a_i \cdot b_i$. A carry signal propagates across k bits if $g_i \cdot p_{i+1} \ldots p_{i+k} = 1$. A variable-latency adder may include a fast and incomplete adder including only carry chains of a length no more than k, and a slow and complete adder with a predictor selecting one of the two adders [Kondo et al. 1997; Lu 2004]. There is only a tiny probability to select the slow and complete adder, which leads to improved average performance. This variable-latency logic design technique can be extended to other arithmetic units such as multipliers [Olivieri 2001] and random logic blocks [Ghosh et al. 2006, 2007].

CRISTA is a generic variable-latency VLSI design methodology which includes three steps as follows [Ghosh et al. 2006, 2007].

(1) Recursively apply Shannon expansion

$$\begin{aligned} & f(x_1, \ldots, x_i, \ldots, x_n) \\ & \quad = x_i \cdot f(x_1, \ldots, x_i = 1, \ldots, x_n) \\ & \qquad + \bar{x}_i \cdot f(x_1, \ldots, x_i = 0, \ldots, x_n) \\ & \quad = x_i \cdot CF_1 + \bar{x}_i \cdot CF_2 \end{aligned} \tag{1}$$

and partition the combinational logic network into a timing-critical sub-network and other non-timing-critical sub-networks whose outputs are joined by multiplexers (Figure 1). A typical CRISTA implementation has $n = 4$ selected input variables, and partitions a combinational logic network into $2^4 = 16$ sub-networks. Assuming

[1]Because a flip-flop can be in a metastable state, the problem of generating a timing error signal, especially timing error detection, may not be cleanly Boolean.

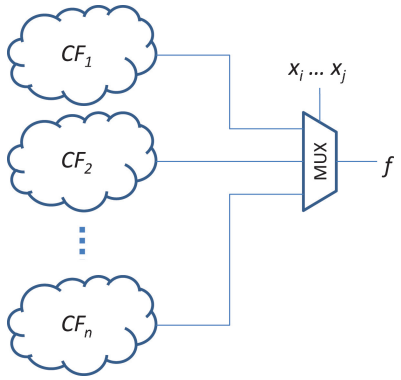

Fig. 1. CRISTA logic.

50% signal (logic one occurrence) probability for the logic inputs and $2^4 = 16$ sub-networks, the critical sub-network which contains the timing critical path has an activation probability of $2^{-4} = 6.25\%$.

(2) Optimize the combinational logic blocks by gate sizing.
(3) Scale down the power supply voltage and the clock-cycle time while meeting a given timing yield requirement. Logic computation in the critical sub-network completes in two clock cycles, while logic computation in the other sub-circuits completes in one clock cycle.

In variable-latency design by function speculation [Neres et al. 2009], timing error detection is based on selecting those internal nodes (as speculation points) in a logic network which cut all timing-critical paths, and comparing their logic values with their respective approximate logic values. For cost, the approximate logic networks are achieved by reusing part of the existing logic network, that is, some of the existing internal nodes are selected for approximate logic. This is similar to generalized bypass transform [McGeer et al. 1991] or generalized select transform [Berman et al. 1990], wherein a multiplexer selects between a fast and a slow signal based on a select logic, with the fast signal coming from part of the existing logic network.

An approximate logic network of a minimal misprediction rate for a speculation point may not be present in an existing logic network. An alternative timing error prediction method is (to construct a "hold logic function" for a "telescopic unit") based on the critical-path sensitization condition, that is, the side inputs need to take their respective non-controlling logic values [Benini et al. 1999]. Benini et al. show that constructing the exact hold logic is NP-complete (as it is equivalent to path sensitization), and present algorithms for constructing approximate hold logic of zero false negative misprediction rate. They further present heuristic algorithms which select a subset of the critical-path side inputs from the critical gates which form the minimum cut of the given timing-critical paths [Benini et al. 1999]. Su et al. [2011] presented an exact hold logic recursive formula for netlists including overlapping critical paths, and proposed to construct an approximate hold logic module by omitting the hold logic components for which the probability of assertion exceeds a certain threshold.

3. VARIABLE-LATENCY VLSI DESIGN

We formulate the variable-latency VLSI design problem as follows.

Problem 1 (*Variable-Latency VLSI Design*). Given a traditional synchronous system operating under a clock frequency, construct a synchronous system under a higher

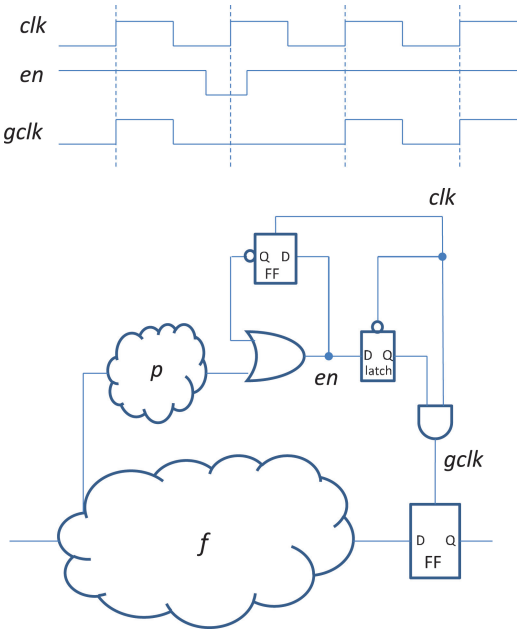

Fig. 2. Minimum-intrusion variable-latency VLSI design.

clock frequency achieving the same computation output in a given timeframe, while allowing the logic computations in a logic stage to complete in a flexible number of clock cycles such that average performance is maximized or the area/power consumption is minimized for a given average performance requirement.

We propose a generic minimum-intrusion variable-latency VLSI design paradigm, a signal-probability-based approximate prediction logic construction method for minimum misprediction rate at minimum cost, and an application-specific cross-layer analysis methodology as follows.

3.1. Minimum-Intrusion Variable-Latency Design Paradigm

For a given gate-level netlist f, we achieve minimum-intrusion variable-latency design by constructing a logic computation completion prediction unit and a clock gating mechanism (Figure 2). In normal operation, the prediction logic p outputs logic one, drives the *en* signal to logic one, and enables the clock signal to reach the D flip-flops. For any slow logic computation which requires more than one clock cycle, the prediction logic p outputs logic zero, drives the *en* signal to logic zero, and blocks the clock signal from reaching the D flip-flops. In the next clock cycle, the inverting D flip-flop and the OR gate restore the *en* signal to logic one such that the slow logic computation takes two clock cycles, and the circuit restores to normal operation. The latch and the AND gate form a clock gate such that an updated *en* signal takes effect in the next clock cycle and the output *gclk* signal has a fixed (e.g., 50%) duty cycle.

This variable-latency design paradigm (Figure 2) allows logic computation in a logic stage to complete in one or two clock cycles. To allow logic computation in a logic stage to complete in $n > 2$ clock cycles, the OR gate and the flip-flop in Figure 2 need to be replaced by an FSM which returns to output one in n cycles after p falls.

We predict a timing error occurrence in the current clock cycle and apply global clock gating before the next clock cycle such that no timing error occurs and no data is

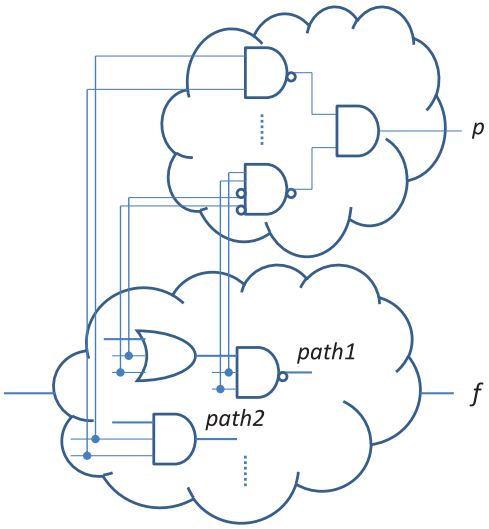

Fig. 3. Timing-critical-path side input-based prediction logic.

corrupt (e.g., the inputs of a slow logic computation hold until the slow logic computation completes). In comparison, Razor and Intel EDS designs detect a timing error in the next clock cycle and recover the timing error by moving back to a previous state, such as by computation replay [Blaauw et al. 2008; Das et al. 2009; Bowman et al. 2011], or holding the previous state in a latch-based design [Fojtik et al. 2012]. For further performance improvement, the prediction logic module can be integrated into a pipeline control logic module.

We construct the prediction logic p as follows. For a given set of critical paths, we predict activation of any of them based on the critical-path side inputs. Given a critical path, for each gate in the critical path other than the on-path gate input, the other gate inputs are off path and are called side inputs. For a signal to propagate through a path, the side inputs need to take the non-controlling logic value of the gate, for instance, logic one for an AND gate or logic zero for an OR gate. For multiple critical paths, the prediction logic p outputs logic zero if any of the critical paths is activated (Figure 3).

$$p = 0 \; if \bigvee_{\pi_j \in \Pi_c} \bigwedge_{s_i \in S(g_i), g_i \in \pi_j} s_i = ncv(g_i), \tag{2}$$

where $\pi_j \in \Pi_c$ is a timing critical path, $g_i \in \pi_j$ is a gate in path π_j, $s_i \in S(g_i)$ is an off-path (side) input of gate g_i, and $ncv(g_i)$ is the non-controlling logic value of gate g_i.

Note that we have *if* instead of *iff* (if and only if) in (2). We do not need to take all critical-path side inputs for the prediction logic p. We only need to take at least one side input for each critical path. Taking a subset of the critical-path side inputs gives an approximation prediction logic. An approximation prediction logic needs to identify all the given timing-critical paths (giving a zero false negative misprediction rate) such that function correctness is guaranteed for all computations while allowing some noncritical paths be identified as critical paths (giving a nonzero false positive misprediction rate), leading to suboptimal performance. Taking a few critical-path side inputs for prediction logic meets these requirements.

The advantages of this design paradigm include: (1) we apply minimum intrusion to an existing gate-level netlist, and (2) we achieve minimum cost increase by reusing the existing logic.

For correctness of this design paradigm, let us take a closer look at signal propagation across a logic gate. Take as an example a two-input AND gate with an on-path input a and an off-path side input b. When a rises (transits from a controlling to a non-controlling logic value), then the following occurs.

(1) If b is 0 (controlling logic value), the path through a is not enabled, which p correctly predicts.
(2) If b is 1 (non-controlling logic value), the path through a is enabled, which p correctly predicts.
(3) If b rises (transits from a controlling to a non-controlling logic value) before a rises, the path through a is enabled, which p correctly predicts.
(4) If b rises after a rises, the path through a is not enabled, while another path through b is more timing critical and needs to be included in the set of timing-critical paths, and p can correctly predict the path through b based on a taking a non-controlling logic value.
(5) If b falls (transits from a non-controlling to a controlling logic value) before a rises, the gate output remains logic zero and no path is enabled across the gate, which p correctly predicts.
(6) If b falls after a rises, there is a glitch at the gate output which may propagate through the path. When we reduce the clock-cycle time in variable-latency design, such a glitch may be in the logic outputs. The prediction logic p predicts such a glitch by including the path through b in the set of critical paths, and predicting activation of the path through b based on a taking a non-controlling logic value.

There is further a timing requirement for the prediction logic. The prediction logic needs to drive the *en* signal to logic zero, allowing a setup time for the latch before the clock signal rises. This timing requirement can be easily met by selecting critical-path side inputs which are closer to the logic inputs with a simple prediction logic p.

3.2. Prediction Logic Construction Methodology

Given a traditional synchronous system operating with a clock-cycle time T_{CC}, a variable-latency system operates with a reduced clock-cycle time T'_{CC} and a clock gating probability $P_0(p) = \Pr(p = 0)$. For a computation which takes n clock cycles in the traditional synchronous system, running in a variable-latency system takes $n(1 + P_0(p))$ clock cycles. The latency reduction is given by

$$\frac{T'}{T} = \frac{n(1 + P_0(p))T'_{CC}}{nT_{CC}} = \frac{T'_{CC}}{T_{CC}}(1 + P_0(p)). \tag{3}$$

By timing analysis, we achieve a ranked list of timing-critical paths. For each timing-critical path π_j, we select a subset $S'(\pi_j) \subseteq S(\pi_j)$ of the side inputs as the inputs of a prediction logic p_j, where $p = \wedge_j p_j$. The prediction logic p_j outputs zero if all the selected side inputs $s_i \in S'(\pi_j)$ take their respective non-controlling logic values $ncv(g_i)$, where s_i is an off-path input of gate g_i.

$$P_0(p_j) = \Pr(p_j = 0) = \Pr\left(\bigwedge_{s_i \in S'(\pi_j)} s_i = ncv(g_i)\right) \tag{4}$$

ALGORITHM 1: Prediction Logic Construction

Input: gate-level netlist G, clock cycle time T_{CC}, timing-critical paths Π_c, signal probabilities, cost limit C
Output: prediction logic p for a variable-latency design of average latency T within cost limit C

```
1.  p = ∅, T = T_CC
2.  foreach (critical path π_j ∈ Π_c) in descending order of D(π_j) {
3.     T'_CC = D(π_{j+1})
4.     foreach (side input s_i ∈ S(π_j)) in ascending order of P_ncv(s_i) {
5.        construct prediction logic p' including s_i by (2)
6.        if (C(p') > C) return p
7.        calculate P_0(p') by (5) and T' = T'_CC(1 + P_0(p'))
8.        if (T' < T) break
9.     }
10.    update prediction logic p = p', T = T'
11. }
```

Assuming stochastic independence between the side inputs,[2] the preceding equation is simplified as

$$P_0(p_j) = \prod_{s_i \in S'(\pi_j)} \Pr(s_i = ncv(g_i)) = \prod_{s_i \in S'(\pi_j)} P_{ncv}(s_i), \tag{5}$$

while the activation probability for path π_j is given by

$$\Pr(\pi_j) = \prod_{s_i \in S(\pi_j)} P_{ncv}(s_i), \tag{6}$$

Clearly, we have

$$P_0(p_j) \geq \Pr(\pi_j) \tag{7}$$

since we have zero false negative and allow false positives. To closely approximate $\Pr(\pi_j)$ by $P_0(p_j)$, we select the side inputs of minimum probability to take their respective non-controlling logic values.

We further develop an algorithm for selecting side inputs and constructing prediction logic at minimum cost as follows. Given a ranked list of timing-critical paths, at each step, we have the choice of improving the runtime speedup based on Eq. (3) by: (1) reducing the clock-cycle time T'_{CC} by selecting a side input of the next timing-critical path, or (2) reducing the clock gating occurrence probability $P_0(p)$ by selecting another side input of the current timing-critical path or one of the processed timing-critical paths. In a greedy algorithm, we compare these choices and select the side input that gives the maximum runtime speedup improvement at each step. However, there are circumstances that we can only improve the runtime speedup by including multiple side inputs in a critical path (because including any single side input would make $P_0(p)$ too large). An improved algorithm (Algorithm 1) is as follows. If, by selecting a side input in the current critical path, we improve the runtime speedup, we proceed to the next critical path. Otherwise, we continue to select more side inputs in the current critical path for a smaller $P_0(p)$, until we improve the runtime speedup or we reach the cost limit.

[2]More accurate signal probability analysis techniques considering signal correlations can be found, for example, in Liu [2008].

3.3. Application-Specific Cross-Layer Analysis

A couple of methodologies are available for timing-critical paths and their side input signal probabilities, including simulation, STA, SSTA, and signal switching analysis as in power estimation.

For accuracy, we obtain signal probabilities by simulation, as signal probabilities strongly depend on workload. Different workloads lead to different signal probabilities and potentially to different variable-latency designs. Subsequently, variable-latency design is application specific and best applied to systems of a few specific application programs such as automotive electronics or embedded systems as part of the Internet of Things. For a general-purpose system, we may construct a variable-latency system based on the average workload, which may not be the optimal design for any specific application program.

For efficiency, we run simulation only at architecture/behavior level for given workload and collect signal probabilities at sequential elements and primary inputs. At gate level, we apply signal-probability-based statistical methods for timing-critical path delays, activation probabilities, and side input signal probabilities.

We have developed a combined statistical timing analysis and signal probability analysis method, namely *signal probability-based statistical timing analysis* (SPSTA) [Liu 2008].

SPSTA leverages the signal probability analysis techniques in VLSI power estimation [Najm 1993]. For example, for an AND gate, its output signal logic one occurrence probability is given by the product of the signal logic one occurrence probabilities at the gate inputs; for an OR gate, its output signal logic zero occurrence probability is given by the product of the signal logic zero occurrence probabilities at the gate inputs. Given signal probabilities at the logic inputs, signal probability analysis calculates signal probabilities for all the nodes in a gate-level netlist with a runtime complexity linear to the size of the netlist. That is, the toggling rate $\rho(y)$ of a signal at the output of a simple logic gate is given by

$$\rho(y) = \sum_i \prod_{j \neq i} \mathrm{P}_{nc}(x_j)\rho(x_i), \tag{8}$$

where $\mathrm{P}_{nc}(x_j) = \Pr(x_j = v_{nc}(x_j))$ is the probability for a side input x_j to take its non-controlling logic value $v_{nc}(x_j)$. SPSTA extends the signal toggling rate in power estimation to a signal transition occurrence probability (top) function in the time domain. The time domain integral (area under the curve) of a top function is the signal toggling rate. The extremes of a top function give the minimum and maximum signal arrival times, respectively. A top function is computed as follows, similar to a toggling rate.

$$\phi(y) = \sum_i \prod_{j \neq i} \mathrm{P}_{nc}(x_j)\phi(x_i) \tag{9}$$

Compared with traditional STA and SSTA techniques, SPSTA gives not only timing-critical path delays but also their activation probabilities. In particular, SPSTA excludes false paths (which have zero activation probabilities) by integrating some of the existing ATPG techniques, such as backtracing and justification, which find any logic conflict in assigning side inputs to their respective non-controlling logic values.

Further, SPSTA finds timing-critical signal propagation networks besides timing-critical paths. A timing-critical signal propagation network includes multiple converging signal propagation paths. This is because, in the presence of performance variabilities, the maximum of multiple input signal arrival times gives the worst-case signal arrival time at the output of a gate. For example, for an AND gate, two rising input signals give a larger or equal signal arrival time at the gate output compared with

any single rising input signal, while two falling input signals give a smaller or equal signal arrival time at the output of the AND gate compared with any single falling input signal.

4. EXPERIMENTS

In this section, we present our experimental results in evaluating the proposed minimum-intrusion predictive variable-latency VLSI design methodology.

4.1. Experimental Setup

We run behavioral-level simulation of the design using VCS and C language programs and generate an SAIF file. We later calculate input signal activation probabilities from signal switching activity and toggling rates listed in the SAIF file. In application-specific system design in which the targeted design would operate on a set of determined programs, running VCS on such programs would be sufficient. However, in the case of a general-purpose system, a set of benchmark programs characterizing a workload of a such system can be used (e.g., SPEC). Alternatively, as the embedded application domain becomes the fastest expanding market segment in the semiconductor industry, a number of benchmarks have been developed which characterize embedded systems workloads. For this purpose we make use of the freely available SPEC MiBench benchmark [Guthaus et al. 2001]. MiBench contains an Automotive and Industrial Control category of programs that characterize typical automotive processes of applications such as air-bag controllers, engine performance monitors, and sensor systems. From the Automotive and Industrial Control category we use *basicmath*, *bitcount*, and *stringsearch* benchmarks. Since networking is becoming more vital as the number of MCUs increases in automotives, we consider the *dijkstra* benchmark from the Network category as well. In cases where cross-layer system optimization is not pursued, random input signal activation probabilities can be taken into account and this step can be completely disregarded.

Because we still need to bound the worst-case critical-path delay in this predictive variable-latency design methodology, we simplify SPSTA by removing any gate delay variation such that each gate has a fixed minimum/maximum delay based on the fast/slow cell library. We run Synopsys PrimeTime and generate an SDF file. SPSTA reads in the SDF file and a Verilog gate-level netlist. We have verified that our modified SPSTA program outputs the same critical-path delays as PrimeTime.

4.2. Test Cases, Results and Observations

Our first test case is the integer unit of an open-source 32-bit SPARC V8 processor LEON2 [Gaisler 2015], which includes a simple 5-stage instruction pipeline for fetch, decoder, execute, memory, and writeback without a multiplier or divider. The integer unit circuit consists of approximately 10,000 gates. We first perform timing-driven logic synthesis by running Synopsys Design Compiler based on the Nangate 45nm open cell library [Silicon Integration Initiative (SI2) 2015] and achieve a gate-level netlist for a given clock-cycle time. We then apply our predictive variable-latency design methodology. We have developed a modified SPSTA program for timing analysis and signal probability analysis. We have verified that our modified SPSTA program reports the same critical-path delays as PrimeTime. We have also verified that our modified SPSTA program and VCS logic simulation report the same clock gating probability for variable-latency designs. We run our modified SPSTA program and print out the critical-path delays, their activation probabilities, and their side input non-controlling value signal probabilities for random inputs. For each critical path we find a side input of minimum non-controlling value signal probability, and calculate an estimated variable-latency design average latency (Algorithm 1). For example, for each

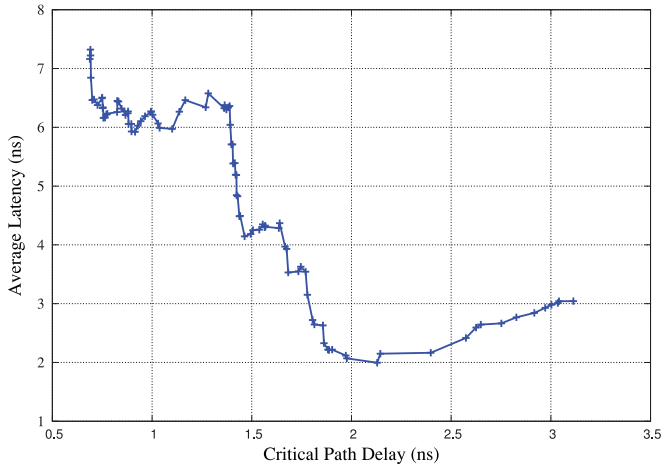

Fig. 4. Critical-path delay vs. estimated variable-latency design average latency for the top 97 critical paths in a LEON2 integer unit gate-level netlist synthesized by Synopsys Design Compiler with a 3.0ns clock-cycle time.

of the first 97 critical paths in a LEON2 integer unit gate-level netlist synthesized by Synopsys Design Compiler with a 3.0ns clock-cycle time, Figure 4 gives delay and average latency $T'_{CC}(1 + P_0(p))$ of a predictive variable-latency design estimated based on the minimum side input non-controlling value signal probabilities. Average latency values on the plot are calculated for the critical paths being predicted up to that critical-path delay. The minimum average latency is achieved by predicting the first 14 critical paths.

We achieve an application-specific predictive variable-latency design in addition to a generic predictive variable-latency design for each of the 9 LEON2 integer unit gate-level netlists synthesized by Synopsys Design Compiler with a clock-cycle time between 1.98ns and 5.96ns. We use random input signal activation probabilities for the generic design. For application-specific design we use average input signal probabilities calculated from architecture/behavior-level simulation over *basicmath*, *bitcount*, *stringsearch*, and *dijkstra* benchmarks [Guthaus et al. 2001]. Figure 5, Table I, and Table II compare these Synopsys Design Compiler logic synthesis results with their corresponding variable-latency design improvements in terms of area, average latency, and energy consumption. The average latency is the maximum true path delay D_{tdd} in a traditional timing-driven design, or the maximum delay D_{vld} of all the true paths excluding the predicted paths in a variable-latency design which are further affected by the clock gating probability $P_{clkgating}$. The energy consumption is the power consumption times the clock-cycle time.

We observe that the proposed predictive variable-latency design methodology provides a quite superior area vs. performance trade-off curve compared with Synopsys Design Compiler timing-driven logic synthesis. While Synopsys Design Compiler timing-driven logic synthesis achieves a minimum average latency of 3.37ns, predictive variable-latency design further reduces it to 2.07ns or by 38.58%. On average, our proposed predictive variable-latency design reduces the average latency by 26.80% at a cost of 0.08% area and 0.4% energy consumption increase for the LEON2 processor interger unit with a clock-cycle time between 1.98ns and 5.96ns. When statistical cross-layer optimization for an automotive electronic embedded system design is applied, improvement in average latency is 41.8%. The prediction unit only requires simple logic such that it introduces minimum area and energy consumption.

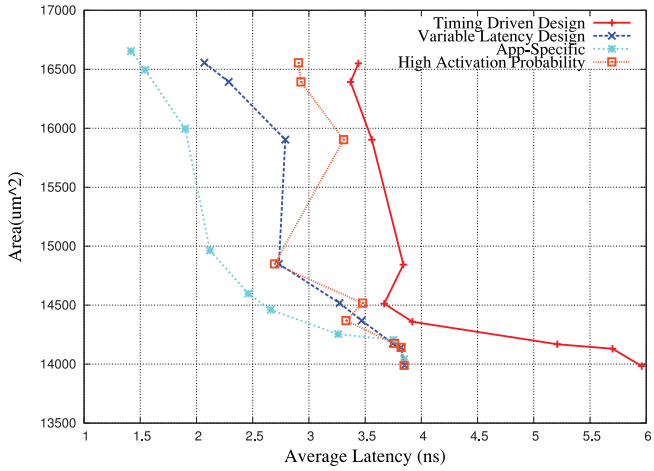

Fig. 5. Area vs. average latency of 9 LEON2 integer unit gate-level netlists synthesized by Synopsys Design Compiler and their corresponding variable-latency improvements of random input activation probability and high activation probability.

Table I. Comparison of LEON2 Integer Unit Gate-Level Netlists Synthesized by Synopsys Design Compiler and their Corresponding Predictive Variable-Latency Improvements in Terms of Area (μm^2), Energy Consumption (fJ), and Average Latency (ns) for Random Logic Inputs

Timing-Driven Design				Generic Variable-Latency Design					Improvements		
CC Const. (ns)	Area (μm^2)	Energy (fJ)	Avg. Lat. (ns)	Area (μm^2)	Energy (fJ)	Avg. Lat. (ns)	Pred. Path (#)	GCK Prob. (%)	Area (%)	Energy (%)	Avg. Lat. (%)
3.0	16549.45	6167.4	3.44	16561.82	6191.51	2.07	14	4.9	0.07	0.39	−39.82
3.1	16392.06	6105.45	3.37	16404.43	6128.10	2.287	10	6.9	0.07	0.37	−32.14
3.4	15647.13	5760.62	3.56	15659.50	5780.14	2.79	9	6.6	0.07	0.34	−21.63
4.0	14844.01	5587.2	3.84	14856.53	5610.22	2.733	8	2.4	0.08	0.41	−28.82
4.5	14512.80	5434.65	3.67	14526.73	5457.85	3.273	8	.25	0.09	0.42	−10.82
5.0	14359.35	5393.0	3.92	14377.49	5413.16	3.469	7	7.6	0.12	0.37	−11.51
5.5	14168.74	5371.57	5.21	14181.11	5394.84	3.746	9	.10	0.08	0.43	−28.10
6.0	14130.67	5310.6	5.70	14144.75	5329.27	3.82	14	.09	0.09	0.35	−32.98
6.5	13983.69	5270.98	5.96	13996.06	5298.48	3.85	8	.09	0.08	0.50	−35.40

Also listed are the number of predicted critical paths and the clock gating probability for each variable-latency design.

Furthermore, to test how our variable latency design performs for a significantly different set of inputs, we provided very high activation probability on all the inputs (75% Stable One and 80% Rising). We gathered new average latency results of our variable-latency design for each of the 9 LEON2 integer unit gate-level netlists synthesized by Synopsys Design Compiler with a clock-cycle time between 1.98ns and 5.96ns. Figure 5 compares the Synopsys Design Compiler logic synthesis results with their corresponding variable-latency design improvements in terms of area and average latency of random input activation probability and high activation probability. Our results show 12.6% average change in average latency. Reduction in average latency of our variable-latency design, compared to time-driven design using the Synopsys tools, decreases from 26.8% to 17.48%.

Our second test case is the floating-point unit of an open-source 32-bit SPARC V8 processor LEON2 [Gaisler 2015], which includes a 5-stage pipeline of opcode stage,

Table II. Comparison of LEON2 Integer Unit Gate-Level Netlists Synthesized by Synopsys Design Compiler and their Corresponding Predictive Variable-Latency Improvements in Terms of Area (μm^2), Energy Consumption (fJ), and Average Latency (ns) for Automotive Applications

Timing-Driven Design				App-Specific Variable-Latency Design					Improvements		
CC Const. (ns)	Area (μm^2)	Energy (fJ)	Avg. Lat. (ns)	Area (μm^2)	Energy (fJ)	Avg. Lat. (ns)	Pred. Path (#)	GCK Prob. (%)	Area (%)	Energy (%)	Avg. Lat. (%)
3.0	16549.45	6167.4	3.44	16654.83	6232.14	1.417	268	4.2	0.63	1.0	−58.8
3.1	16392.06	6105.45	3.37	16494.15	6189.25	1.543	165	4.8	0.61	1.3	−54.2
3.4	15647.13	5760.62	3.56	15723.13	5805.62	1.9	95	4	0.4	0.7	−46.6
4.0	14844.01	5587.2	3.84	14909.11	5619.2	2.12	86	4	0.4	0.5	−44.7
4.5	14512.80	5434.65	3.67	14557.8	5702.37	2.46	56	3.1	0.3	0.5	−32.9
5.0	14359.35	5393.0	3.92	14406.35	5425.56	2.66	45	2.5	0.32	0.5	−32.1
5.5	14168.74	5371.57	5.21	14254.38	5399.54	3.26	16	0.1	0.1	0.5	−37.4
6.0	14130.67	5310.6	5.70	14140.38	5330.68	3.752	18	0.1	0.1	0.3	−34.17
6.5	13983.69	5270.98	5.96	13995.12	5279.38	3.847	10	0.1	0.09	0.51	−35.4

Also listed are the number of predicted critical paths and the clock gating probability for each variable-latency design.

pre-normalization stage, addition/subtraction stage, post-normalization stage, and rounding stage. Once again we perform timing-driven logic synthesis by running Synopsys Design Compiler based on the Nangate 45nm open cell library [Silicon Integration Initiative (SI2) 2015] and achieve a gate-level netlist for a given clock-cycle time. We then apply our predictive variable-latency design methodology. We applied our methodology based on Algorithm 1. After running our modified SPSTA program, we print out the critical-path delays, their activation probabilities, and their side input non-controlling value signal probabilities for random inputs. For each critical path we find a side input of minimum non-controlling value signal probability, and calculate an estimated variable-latency design average latency to the point that no improvement in average latency can be achieved. We start selecting more side inputs to reach minimum average latency (Algorithm 1). For example, for each of the first 95 critical paths in a LEON2 floating-point unit gate-level netlist synthesized by Synopsys Design Compiler with a 12.0ns clock-cycle time, Figure 6 depicts our methodology and gives the delay and average latency of a predictive variable-latency design estimated based on the minimum side inputs' non-controlling value signal probabilities.

We attained a predictive variable-latency design for each of the 13 LEON2 floating-point unit gate-level netlists synthesized by Synopsys Design Compiler with a clock-cycle time between 4.28ns and 15.6ns. Figure 7 and Table III compare these Synopsys Design Compiler logic synthesis results with their corresponding variable-latency design improvements in terms of area, average latency, and energy consumption. Our variable-latency design methodology based on Algorithm 1 on average reduces the average latency by 14.65% at cost of 3.4% area and 2.2% energy consumption for the floating-point unit with a clock-cyle time between 3.49ns and 13.74ns.

4.3. Further Discussion and Comparison to Existing Variable-Latency Design Techniques

Besides comparing with a mainstream logic synthesizer Synopsys Design Compiler, we compare the proposed predictive variable-latency design methodology with a few leading variable-latency design methodologies in literature as follows.

(1) RAZOR and Intel EDS designs are timing-error-detection-based variable-latency design techniques. In comparison, the proposed methodology is based on timing error prediction and still requires a timing analysis method to bound the

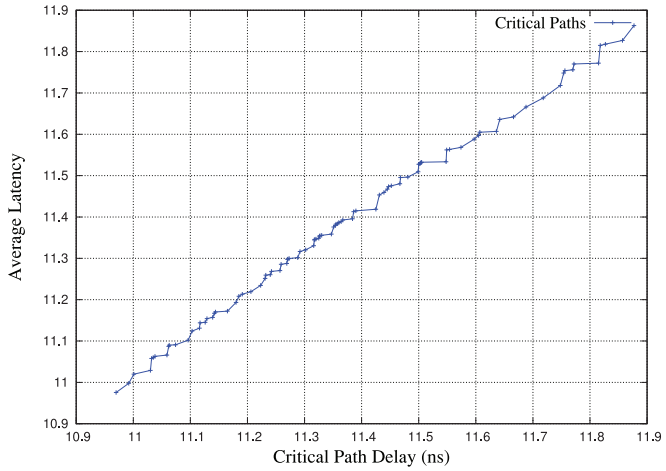

Fig. 6. Critical-path delay vs. estimated variable-latency design average latency for the top 95 critical paths in a LEON2 floating-point unit gate-level netlist synthesized by Synopsys Design Compiler with a 12.0ns clock-cycle time.

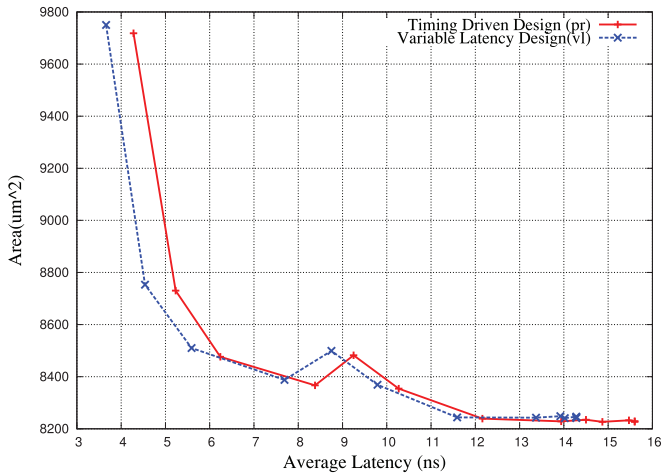

Fig. 7. Area vs. average latency of 13 LEON2 floating-point unit gate-level netlists synthesized by Synopsys Design Compiler and their corresponding variable-latency improvements.

worst-case timing performance under parametric variations, the same as in traditional synchronous design. The proposed methodology gives some of the top critical paths two clock cycles for a signal to propagate through, thus reducing the timing error rate due to performance variability compared with traditional synchronous design. A timing margin ΔT for performance variation tolerance can be taken into account in prediction logic construction as in Su et al. [2011]. The proposed predictive variable-latency design methodology can be extended to combine with a detective variable-latency design methodology, for example, by deploying a timing error detection element at the end of a critical path under prediction. This timing error detection signal is integrated with the timing error prediction logic such that clock gating is applied only if: (1) a critical/near-critical path p is predicted to have a signal transition propagating through, and (2) at the end of the path p

Table III. Comparison of LEON2 Floating-Point Unit Gate-Level Netlists Synthesized by Synopsys Design Compiler and their Corresponding Predictive Variable-Latency Improvements in Terms of Area (μm^2), Energy Consumption (fJ), and Average Latency (ns) for Random Logic Inputs

Timing-Driven Design				Variable-Latency Design					Improvements		
CC Const. (ns)	Area (μm^2)	Energy (fJ)	Avg. Lat. (ns)	Area (μm^2)	Energy (fJ)	Avg. Lat. (ns)	Pred. Path (#)	GCK Prob. (%)	Area (%)	Energy (%)	Avg. Lat. (%)
4.0	9718.4	3040.2	4.28	10045.15	3098.44	3.4	114	0.1	3.3	1.91	−20.56
5.0	8730.35	2818.65	5.23	9015.05	2866.8	4.0	149	0.6	3.2	1.7	−23.52
6.0	8476.35	2749.38	6.24	9214.7	2857.92	5.2	110	0.3	8	3.9	−16.7
8.0	8366.87	2610.72	8.38	8871.07	2716.8	7.17	130	0.2	6	4.0	−14.42
9.0	8482.33	2639.88	9.25	8971.65	2735.82	8.45	92	0.13	5.7	3.6	−8.6
10.0	8354.24	2663.3	10.27	8799.08	2746.5	9.35	86	0.15	5.3	3.1	−9
12.0	8239.08	2616.6	12.16	8801.35	2733.84	10.88	99	0.25	6.8	4.4	−10.53
14.0	8228.62	2620.24	13.94	8447.32	2664.48	12.02	38	0.5	2.6	1.6	−13.77
14.5	8234.69	2636.1	14.50	8355.94	2667.27	12.4	26	0.8	1.4	1.18	−14.48
15.0	8226.79	2619.15	14.87	8315.44	2645.1	12.5	23	0.9	1.07	0.9	−15.94
15.5	8232.48	2633.91	15.47	8317.36	2660.73	12.43	24	1.0	1.03	0.9	−19.65
16.0	8226.80	2617.44	15.6	8294.87	2641.76	13.75	18	0.5	0.8	0.9	−11.67
16.5	8229.32	2632.08	15.6	8284.5	2655.34	13.78	16	0.5	0.6	0.8	−11.67

Also listed are the number of predicted critical paths and the clock gating probability for each variable-latency design.

no signal transition is detected. This combined predictive and detective variable-latency design methodology reduces the misprediction rate without missing any timing error in prediction, thus further improving average latency. It further improves design reliability under performance variability in that an accurate timing analysis method giving the worst-case timing performance may not be needed. This combined predictive and detective variable-latency design requires a smaller number of timing error detection elements compared with the existing detective variable-latency designs.

(2) CRISTA is a predictive variable-latency design technique. CRISTA applies Shannon expansion for each logic network and inserts a multiplexer for each output bit which leads to significant area increase. CRISTA has a fixed 6.25% clock gating probability which affects the performance improvement. For a three-stage pipeline wherein each stage is an MCNC benchmark circuit, CRISTA achieves an average of 60% power saving with 18% area overhead and 6% performance degration [Ghosh et al. 2006, 2007].

(3) The telescopic design methods are state-of-the-art variable-latency design methods [Benini et al. 1998, 1999; Su et al. 2011]. Benini et al. proposed to construct a telescopic unit or hold logic module based on the critical-path sensitization condition [Benini et al. 1999]. Benini et al. further proposed to find a minimum number of critical gates which cut all the critical paths, and construct the hold logic based on some of the critical-gate side inputs [Benini et al. 1999]. Su et al. [2011] further presented an exact hold logic recursive formula for netlists including overlapping critical paths, and proposed to construct an approximate hold logic module by omitting those components for which the probability of assertion exceeds a certain threshold. Our proposed technique can be taken as a further improvement of these techniques. We sort the critical-path side inputs by their non-controlling logic value probabilities, and construct a prediction logic based on the critical-path side inputs of the minimum non-controlling logic value probabilities. We further calculate the critical-path activation probabilities and side input signal probabilities

by architecture-level simulation and SPSTA [Liu 2008]. SPSTA is an input-aware statistical timing analyzer which gives signal probabilities and path activation probabilities besides critical-path delays. While the state-of-the-art telescopic design methods in Benini et al. [1999] and Su et al. [2011] achieve, respectively, an average performance improvement of 13.99% and 21.67% with an area overhead of 11.62% and 16.20% for the ISCAS'85 and ISCAS'89 benchmark circuits, we achieve an average performance improvement of 26.80% (14.65%) with an area overhead of 0.08% (3.4%) for the LEON2 processor integer (floating point) unit.

We apply the proposed prediction unit construction method after logic synthesis for accurate timing analysis and a fixed group of timing-critical paths. Subsequent physical design procedures may introduce timing analysis inaccuracy or even new timing-critical paths. This can be handled by predicting a few more near-critical paths or updating the prediction logic in post-route timing optimization or *engineer-change-order* (ECO).

5. CONCLUSION

In this article we present an improved variable-latency design methodology including: (1) a generic minimum-intrusion variable-latency VLSI design paradigm, (2) a signal-probability-based approximate prediction logic construction method for minimum false positive misprediction rate at minimum cost, (3) an application-specific cross-layer analysis methodology. This methodology improves average performance at minimum area and energy consumption overhead because: (1) a critical path typically has a tiny activation probability which is the probability for all its side inputs to have their respective non-controlling logic values, and (2) prediction of critical path activation requires a simple logic and an area that is linear to the number of inputs. Our experimental results based on an open-source SPARC V8 processor LEON2 and the 45nm Nangate open-source cell library show that the proposed variable-latency design methodology, on average, reduces the expected logic computation latency by 26.80%(14.65%) at average cost of 0.08%(3.4%) area and 0.4%(2.2%) energy consumption increase for the interger (floating point) unit with a clock-cycle time between 1.97ns(3.49ns) and 5.96ns(13.74ns), while improvement in average latency for an automotive electronic-specific design is 41.8%. Our ongoing research targets integrating timing error prediction logic into the existing instruction pipeline stall logic for further performance improvement.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable comments and Jiaxin Guo, Abdullah Al Owahid, and Mohammed Quiyaam Farooqui for their contribution to the experimental setup.

REFERENCES

M. Alam. 2008. Reliability- and process-variation aware design of integrated circuits. *Microelectron. Reliab.* 48, 1114–1122.

T. Austin, V. Bertacco, D. Blaauw, and T. Mudge. 2005. Opportunities and challenges for better than worst-case design. In *Proceedings of the Asian and South Pacific Design Automation Conference (ASP-DAC'05).* 2–7.

T. Austin, D. Blaauw, T. Mudge, and K. Flautner. 2004. Making typical silicon matter with razor. *IEEE Comput.* 37, 3, 57–65.

L. Benini, E. Macii, M. Poncino, and G. D. Micheli. 1998. Telescopic units: A new paradigm for performance optimization of VLSI designs. *IEEE Trans. Comput.-Aided Des.* 17, 3, 220–232.

L. Benini and G. D. Micheli. 2004. Networks on chips: A new paradigm for component-based MPSoC design. http://si2.epfl.ch/~demichel/publications/archive/2004/mpsoc.pdf.

L. Benini, G. D. Micheli, A. Lioy, E. Macii, G. Odasso, and M. Poncino. 1999. Automatic synthesis of large telescopic units based on near-minimum timed supersetting. *IEEE Trans. Comput.* 48, 8, 769–779.

C. L. Berman, D. J. Hathaway, A. S. Lapaugh, and L. Trevillyan. 1990. Efficient techniques for timing corretion. In *Proceedings of the IEEE International Symposium Circuits and Systems (ISCA'90).* 415–419.

D. Blaauw, S. Kalaiselvan, K. Lai, W.-H. Ma, S. Pant, C. Tokunaga, S. Das, and D. Bull. 2008. RAZOR-II: In-situ error detection and correction for PVT and SER tolerance. In *Proceedings of the IEEE Solid State Circuits Conference (ISSCC'08).* 400–401.

S. Borkar. 2005. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro* 11, 10–15.

B. Bose and T. R. N. Rao. 1982. Theory of unidirectional error correcting/detecting codes. *IEEE Trans. Comput. C-31,* 6, 521–530.

K. A. Bowman, J. W. Tschanz, N. S. Kim, J. Lee, C. B. Wilkerson, S. L. Lu, T. Karnik, and V. K. De. 2008. Energy-efficient and metastability-immune timing-error detection and instruction-replay-based recovery circuits for dynamic-variation tolerance. In *Proceedings of the IEEE Solid State Circuits Conference (ISSCC'08).* 402–623.

K. A. Bowman, J. W. Tschanz, S.-L. L. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, B. M. Geuskens, C. Tokunaga, C. B. Wilkerson, T. Karnik, and V. K. De. 2011. A 45nm resilient microprocessor core for dynamic variation tolerance. *IEEE J. Solid State Circ.* 46, 1, 194–208.

S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. 2006. A self-tuning dvs processor using delay-error detection and correction. *IEEE J. Solid State Circ.* 41, 4, 792–804.

S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw. 2009. Razorii: In situ error detection and correction for PVT and SER tolerance. *IEEE J. Solid State Circ.* 44, 1, 32–48.

D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner. 2004. RAZOR: Circuit-level correction of timing errors for low-power operation. *IEEE Micro* 24, 6, 10–20.

D. Ernst, N. S. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, and T. Mudge. 2003. RAZOR: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'03).* 7.

M. Fojtik, D. Fick, Y. Kim, N. Pincknet, D. Harris, D. Blaauw, and D. Sylvester. 2012. Bubble razor: An architecture-independent approach to timing-error detection and correction. In *Proceedings of the IEEE Solid State Circuits Conference (ISSCC'12).* 488–490.

S. B. Furber and P. Day. 1996. Four-phase micropipeline latch control circuits. *IEEE Trans. VLSI Syst.* 4, 2, 247–253.

A. Gaisler. 2015. LEON SPARC V8 processors. http://www.gaisler.com/.

S. Ghosh, S. Bhunia, and K. Roy. 2006. A new paradigm for low-power, variation-tolerant circuit synthesis using critical path isolation. In *Proceedings of the IEEE International Conference Computer-Aided Design (ICCAD'06).* 619–624.

S. Ghosh, S. Bhunia, and K. Roy. 2007. Crista: A new paradigm for low-power, variation-tolerant, and adaptive circuit synthesis using critical path isolation. *IEEE Trans. Comput.-Aided Des.* 26, 11, 1947–1956.

S. Ghosh and K. Roy. 2010. Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era. *Proc. IEEE* 98, 10, 1718–1751.

M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE International Workshop on Workload Characterization (WWC'01).* 3–14.

S. Hauck. 1995. Asynchronous design methodologies: An overview. *Proc. IEEE* 83, 1, 69–93.

N. K. Jha and S. J. Wang. 1993. Design and synthesis of self-checking VLSI circuits. *IEEE Trans. Comput.-Aided Des.* 12, 878–887.

D. R. Kelly and B. J. Phillips. 2005. Arithmetic data value speculation. In *Proceedings of the 10th Asia-Pacific Conference on Advances on Computer Systems Architecture (ACSAC'05).* 353–366.

Y. Kondo, N. Ikumi, K. Ueno, J. Mori, and M. Hirano. 1997. An early-completion-detecting alu for a 1GHz 64B datapath. In *Proceedings of the IEEE Solid State Circuits Conference (ISSCC'97).* 418–497.

B. Liu. 2008. Signal probability based statistical timing analysis. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE'08).* 562–567.

B. Liu, X. Chen, and F. Teshome. 2012. Resilient and adaptive performance logic. *ACM J. Emerg. Technol. Comput. Syst.* 8, 3.

S.-L. Lu. 2004. Speeding up processing with approximation circuits. *Comput.* 37, 3, 67–73.

G. Mago. 1973. Monotone functions in sequential circuits. *IEEE Trans. Comput.* 22, 10, 928–933.

P. C. Mcgeer, R. K. Brayton, A. L. Sangiovanni-Vincentelli, and S. K. Sahni. 1991. Performance enhancement through the generalized bypass transform. In *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD'91)*. 184–187.

D. E. Muller and W. S. Bartky. 1959. A theory of asynchronous circuits. In *Proceedings of the International Symposium on the Theory of Switching*. 204–243.

F. N. Najm. 1993. Transition density: A new measure of activity in digital circuits. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* 12, 2, 310–323.

D. B. Neres, J. Cortadella, and M. Kishinevsky. 2009. Variable-latency design by function speculation. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE'09)*. 1704–1709.

M. Olivieri 2001. Design of synchronous and asynchronous variable-latency pipelined multipliers. *IEEE Trans. VLSI Syst.* 9, 2, 365–376.

Silicon Integration Initiative (SI2). 2015. Nangate open cell library. http://www.si2.org/openeda.si2.org/projects/nangatelib/.

M. Singh and S. M. Nowick. 2007. MOUSETRAP: Ultra-high-speed transition signaling asynchronous pipelines. *IEEE Trans. VLSI Syst.* 15, 6, 684–698.

J. Sparso and S. Furber. 2001. *Principles of Asynchronous Circuit Design – A Systems Perspective.* Kluwer Academic.

Y.-S. Su, D.-C. Wang, S.-C. Chang, and M. Marek-Sadowska. 2011. Performance optimization using variable-latency design style. *IEEE Trans. VLSI Syst.* 19, 10, 1874–1883.

I. E. Sutherland. 1989. Micropipelines. *Comm. ACM* 32, 6, 720–738.

T. Verhoeff. 1988. Delay-insensitive codes – An overview. *Distrib. Comput.* 3, 1–8.

N. H. E. West and D. M. Harris. 2011. *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Addison-Wesley.

Received June 2014; revised October 2014; accepted March 2015