On Modeling and Evaluation of Logic Circuits Under Timing Variations

Mehdi Dehbashi[*]
[*]Institute of Computer Science
University of Bremen
28359 Bremen, Germany
dehbashi@informatik.uni-bremen.de

Görschwin Fey[*†]
[†]Institute of Space Systems
German Aerospace Center
28359 Bremen, Germany
goerschwin.fey@dlr.de

Kaushik Roy[‡], Anand Raghunathan[‡]
[‡]School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907, USA
{kaushik, raghunathan}@purdue.edu

***Abstract*—This paper presents a methodology to model and analyze the functional behavior of logic circuits under timing variations. In the framework, first a *Time Accurate Model* (TAM) of the circuit is constructed. The TAM represents the behavior of the circuit in the functional domain under a discrete time model. Afterwards, *Variation Logic* is inserted to apply the timing variations. Moreover, the circuit TAM is enhanced by *Time Control* (TC) logic to model the circuit frequency. We apply the proposed methodology to analyze a circuit or an approximate circuit under timing variations as well as to analyze a circuit under timing-induced errors for approximate computing.**

***Keywords*—logic circuits, timing variations, formal verification**

I. Introduction

As *Integrated Circuit* (IC) technology continues to scale down, variability is recognized to be a major challenge in analyzing the circuits. In this case, delay deviations are imposed by process variations such as uncertainty in the parameters of fabricated devices and interconnects, and by environmental variations such as temperature and voltage [1] [2] [3].

Recently, there is a range of works that considers timing analysis of circuits under variations. A survey of the works focusing on *Statistical Static Timing Analysis* (SSTA) is given in [1]. The statistically-critical paths under process variations are extracted by a bound-based method in [4]. The extracted paths have the highest probability to fail the timing constraint. The effects of process variations on the delays of logic gates and timing errors are analyzed in [2] [5] [3]. Timing error detection and correction has been proposed as an approach to bridge the gap between typical case and worst case design, by allowing circuits to operate without any margins [6] [7]. The work in [8] presents a framework to evaluate how microarchitectural techniques can trade off variation-induced errors for power and processor frequency.

On the other hand, in the recent years, significant progress in areas such as approximate and probabilistic computing has been achieved. Computations typically addressed in these areas focus on good enough or bounded results but not necessarily exact results [9]. In these techniques, the requirement of exact numerical or Boolean equivalence is relaxed to yield performance or energy efficiency [10] [11] [12].

An approximate implementation of a circuit does not exactly match the specification because of timing-induced errors or functional approximations [9]. Timing-induced errors can be produced by voltage over-scaling or overclocking.

Systematic synthesis of approximate circuits is exploited in [13] [14] [15] to reduce circuit area and delay as well as to increase yield. The work in [16] develops a logic optimization procedure that utilizes multi-V_t (threshold voltage) libraries to optimize a circuit for higher frequency and throughput under timing error detection and correction. The work in [17] uses a power-aware slack redistribution that shifts the timing slack of frequently-excercised, near-critical timing paths in a power- and area-efficient manner. The work in [18] presents an *Error-Resilient System Architecture* (ERSA) which combines unreliable cores with a small fraction of reliable processor cores for running system software, controlling application flow, and recovering from errors generated on unreliable cores. Scalable effort hardware design is proposed in [19] to identify mechanisms at each level of design abstraction (circuit, architecture, and algorithm) which can be used to vary the computational effort expended for generating the exact results. These scaling mechanisms are utilized to improve energy efficiency while maintaining an acceptable result.

A systematic methodology for the modeling and analysis of circuits for approximate computing is proposed in [12]. The methodology is utilized to analyze a circuit under timing-induced approximations as well as functional approximations using multiple metrics. However, timing variations are not considered. Since variations can significantly perturb the timing of various paths in a circuit, it is natural to expect that they will also significantly impact which paths fail under timing-induced approximations, and therefore, the functional behavior of approximate circuits. For this purpose considering the timing behavior of variations during the analysis of approximate circuits is essential.

In this paper, we propose a unified framework that can be used to analyze *how a circuit behaves under timing variations*, *how a circuit behaves under timing-induced approximations*, and *how an approximate circuit behaves under timing variations*. By considering the functional domain, our approach is complementary to SSTA. In the approach, we first convert the timing behavior of a circuit into the functional domain according to a time unit model. The newly constructed circuit is called the *Time Accurate Model* (TAM) of the circuit. The TAM represents the functional behavior of the circuit with respect to the circuit delay and a precision of an arbitrarily fine-grained but discrete time unit. Afterwards, *Variation Logic* (VL) is inserted in the TAM to apply the timing variation. The VL is applied at each gate. The cumulative slowdown or speedup normalized to a time unit may affect the correct behavior of the circuit. The behavior of the VL is determined by a *Variation Control* (VC) component. Moreover, the circuit is enhanced by *Time Control* (TC) logic. TC is a flexible model to control the frequency at the inputs. TC models timing-induced approximations like overclocking. We use *Boolean Satisfiability* (SAT) as an underlying reasoning engine to analyze the circuits.

The rest of this paper is organized as follows. Section II introduces preliminary information. Section III describes our approach to construct the TAM, to insert VL, and to enhance a circuit by TC and VC. Then, experimental results on arithmetic units are presented in Section IV. Section V concludes the work.

This work has been funded in part by the German Research Foundation (DFG, grant no. FE 797/6-1).

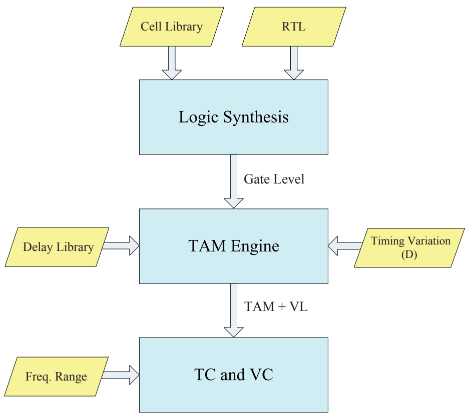

Fig. 1. Overview of proposed methodology

II. Preliminaries

A. Timing Parameters

Delay of a component in a circuit is the amount of time that a signal needs to propagate from the component inputs to its outputs. A change of the component delay is called *timing variation*. Timing variation may increase the component delay called $slowdown$, or may decrease the component delay called $speedup$.

We consider a *time unit* which is an arbitrarily fine-grained but discrete unit of delay. The delays of gates and interconnects are assumed to be an integer multiple of one time unit[1]. In a circuit where the shortest path delay is D_s time units, and the longest path delay is D_l time units, the current output O_t depends on the inputs of I_{t-D_s}, I_{t-D_s-1}, $\ldots$, I_{t-D_l} where indices denote the times of input with a step of one time unit. Each index is also called *time step*.

A *clock period* is defined as T time units. In synchronous circuits, the input to the combinational logic changes only once every *clock cycle*. Clock cycles denote the times of inputs with a step of one clock period. If the circuit has a clock period of T, the output at time step t depends on the inputs of the following clock cycles:

$$\forall i, a \leq i \leq b : [I_{t-iT-1},\ \ldots,\ I_{t-(i+1)T}] \tag{1}$$

$$a = \lceil D_s/T \rceil - 1\,, \quad b = \lceil D_l/T \rceil - 1$$

This formula partitions the times of input according to the clock period T such that in each clock cycle, the inputs are assumed to be fixed. For example, when $D_s = 1$, $D_l = 5$, $T = 5$, O_t depends on input values from time steps that fall within the previous clock cycle $[I_{t-1}, I_{t-2}, I_{t-3}, I_{t-4}, I_{t-5}]$, and in this clock cycle, the inputs do not change: $I_{t-1} = I_{t-2} = I_{t-3} = I_{t-4} = I_{t-5}$. When $T = 2$, O_t depends on input values from time steps that fall within the following clock cycles: $[I_{t-1}, I_{t-2}]$, $[I_{t-3}, I_{t-4}]$, $[I_{t-5}, I_{t-6}]$. Overall, when $T < D_l$, the clock is overscaled, i.e., the current output depends on the inputs of multiple previous clock cycles. This case is also called *overclocking*. We note that, for our purpose, voltage overscaling has the same effect as overclocking, since the delays of the gates will be scaled up based on the lower voltage, while the clock period remains the same.

When the clock is overscaled, the longer paths fail because the input does not have enough time to propagate to the output. In this case, the current output result depends not only on the input of one previous clock cycle but also on the inputs of multiple previous clock cycles. The "older" inputs (the inputs of the clock cycles more distant from current time t) influence the output through longer paths and the "newer" inputs (the inputs of the clock cycles closer to the current time t) affect the output through shorter paths.

[1]At the end of Section III-C, we discuss how more complex timing models can be handled by our approach.

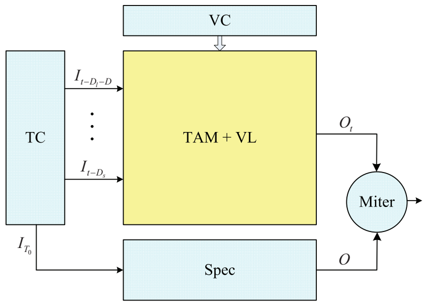

Fig. 2. Overall model created by the framework

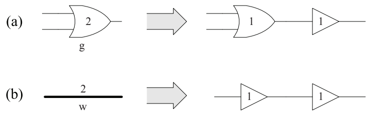

Fig. 3. Converting original gates and wires to untimed gates and wires

III. Methodology

The fine-grained timing behavior of a circuit is converted into the functional domain. Having the behavior of a circuit according to a fine-grained time unit allows us to utilize it for modeling races, glitches, etc. The fine-grained timing model is also utilized to control and to modify the frequency of a circuit during our analysis. This fine-grained timing model of the circuit is called *Time Accurate Model* (TAM). When the timing behavior of a circuit is available in the functional domain, formal verification methods can comprehensively analyze the timing effects of the circuit.

Figure 1 shows the overview of our approach. Firstly, the gate level circuit (synthesized netlist) is generated according to a cell library. Afterwards, the TAM engine creates the time accurate model of the circuit which models the fine-grained timing behavior of a circuit in the functional domain. The TAM is generated according to a fine-grained time unit. The time unit specifies the granularity of the analysis. The delays of all gates in the circuit are normalized according to the time unit. *Variation logic* (VL) is inserted in the TAM according to the maximum timing variation (D).

In the final step, *Time Control* (TC) and *Variation Control* (VC) are added. TC includes some constraints on the inputs to control the clock period according to Formula 1. VC is a constraint to control the timing variation. By this, the model can be used to analyze a circuit with respect to different frequencies.

Figure 2 shows the overall model created by our framework for the analysis. The main components of the model are: TAM and VL, TC, VC. Also the model includes two side components: *spec* and *miter*. These two components serve different tasks in different applications. *Spec* can be a golden specification or golden properties of the ideal circuit behavior. *Miter* measures the deviation of the circuit output result against its specification. Here, we use a SAT solver as an underlying engine to measure the deviations.

By using our framework, a designer can answer the following questions while varying the frequency:

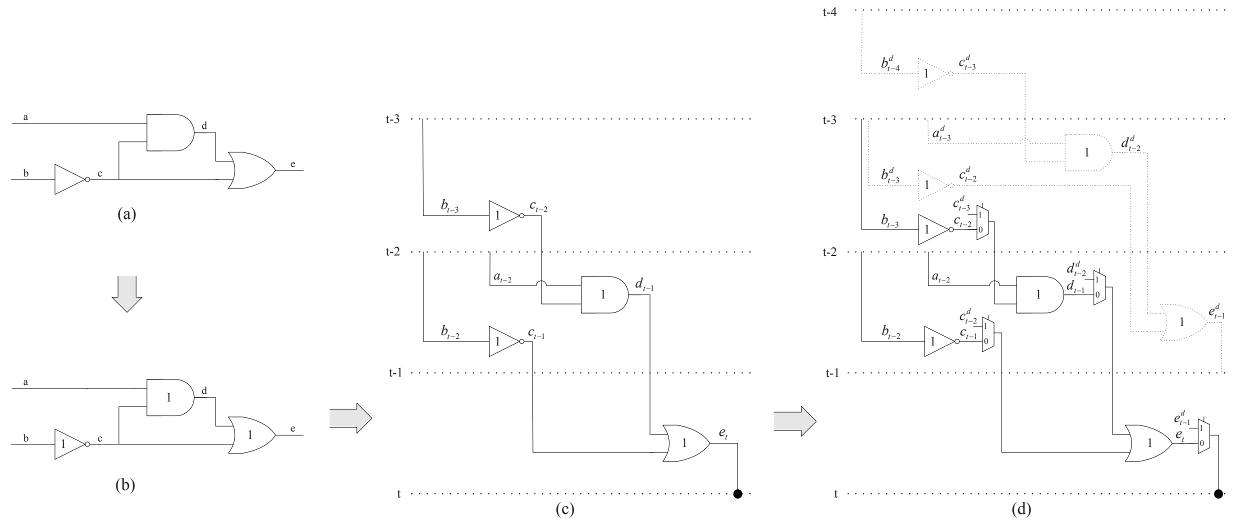

Fig. 4. (a) Original circuit and original gates (b) Untimed circuit and untimed gates (c) TAM circuit and TAM gates (d) TAM and VL

```
1  function $TAM(In: untimed\ circuit,\ Out: TAM\ circuit)$
2  $time = 0$
3  $SIG = PO$
4  while $SIG \neq \emptyset$ do
5  {
6     $SIG\_temp = \emptyset$
7     foreach $sig \in SIG$ do
8     {
9        $gate = predecessor(sig)$
10       $copy(gate, i_{t-time-1}, o_{t-time})$
11
12       foreach $input \in I(gate)$ do
13          if $input \notin SIG\_temp$ and $input \notin PI$ then
14             $SIG\_temp = SIG\_temp \cup input$
15    }
16    $SIG = SIG\_temp$
17    $time++$
18 }
19 end function
```

Fig. 5. Creation of Time Accurate Model

- How does a circuit behave under timing variations?
- How does a circuit behave under timing-induced approximations, i.e., under overclocking?
- How does an approximate circuit (overclocked circuit) behave under timing variations?

In the following, Section III-A describes how the TAM of a circuit and its VL are created. The TC and VC components are explained in Section III-B. Section III-C discusses how speedup variations are considered in our framework.

A. TAM Engine

The TAM algorithm was inspired from the algorithm presented in [12]. However, there are key differences necessitated by the need to handle variations - we consider a fine-grained time unit and also the variation logic is added into the model.

The underlying idea is that a signal s_t represents a signal s of the original circuit at time step t. In the TAM engine, first the delays of the original gates and wires are converted into the functional domain according to the chosen time unit. An *original gate* g with delay n is converted to n successive *untimed gates*: $(g, Buf_{n-1}, \ldots, Buf_1)$ modelling the n time steps required to propagate a value from the input of g to the output of g (Figure 3(a)). The equivalent untimed gates show the behavior of the original gate with an accuracy of one time unit. Also a wire w with delay n is converted to n successive buffers: $(Buf_n, \ldots, Buf_1)$ (Figure 3(b)). The circuit with the untimed gates and wires is called the *untimed circuit*. In [20], untimed gates are used for ATPG of crosstalk faults.

After untiming the original gates and wires, the second step of the TAM engine starts to convert the timing behavior of the overall circuit into the functional domain. Figure 5 describes the algorithm in pseudo code. The input data of this algorithm is an untimed circuit. For each gate the algorithm creates as many copies as values of the gate at different time steps may be relevant to determine the output.

The algorithm starts from *Primary Outputs* (PO) and traverses the untimed circuit graph backward (line 3). Given the delay of one time unit at each gate in the untimed circuit, the output of an untimed gate depends on its corresponding inputs one time step ago. These inputs are given by the predecessor node of the output in the untimed circuit graph (line 9). Given the output and its driving inputs, a new gate is created in which the output and the inputs have the timing difference of one time unit (line 10). The newly created gates are called *TAM gates* and constitute a new circuit called the *TAM circuit*. The inputs of the current untimed gates are collected in the set SIG_temp to be used for the next backward traversal step (lines 12-14). If the input is a *Primary Input* (PI) or already exists in the set SIG_temp (fanout case), it will not be added to the set SIG_temp (line 13).

We explain the algorithm using the example original circuit of Figure 4(a). For sake of simplicity, the delays of original gates are considered to be 1. Therefore, the untimed circuit (Figure 4(b)) is the same as the original circuit. In the first step, the algorithm starts from primary output e and copies the OR gate: $OR(c_{t-1},\ d_{t-1},\ e_t)$. The copied gates (TAM gates) are shown in Figure 4(c). The dotted lines visualize the time steps. $SIG\_temp = \{c, d\}$ is created. The second step is backward traversal of the untimed circuit from c and d. The NOT and AND gates are copied: $NOT(b_{t-2},\ c_{t-1})$, $AND(a_{t-2},\ c_{t-2},\ d_{t-1})$. Having $SIG\_temp = \{c\}$, the third step starts and copies the NOT gate: $NOT(b_{t-3},\ c_{t-2})$. As explained in Section II-A, O_t depends on the inputs of $I_{t-D_s}, I_{t-D_s-1}, \ldots, I_{t-D_l}$ where $D_s = 2$ and $D_l = 3$ in this example.

For investigating the behavior of a circuit under timing variations, the *Variation Logic* (VL) is inserted in the TAM. The VL models the slowdown of a signal by d time units. The slowdown is modeled for each gate independently by activating the select line of the variation multiplexer of the corresponding gate. Therefore, in addition to the value of a signal in the corresponding time, the value of the signal d

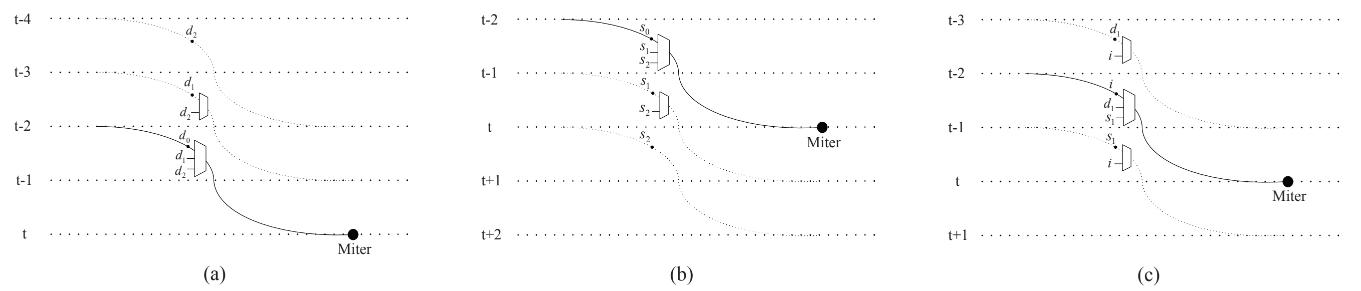

Fig. 6. (a) Slowdown (b) Speedup (c) Slowdown and Speedup

```
1   function $TAM + VL()$
2   $time = 0$
3   $SIG = PO$
4   while $SIG \neq \emptyset$ do
5   {
6       $SIG\_temp = \emptyset$
7       foreach $sig \in SIG$ do
8       {
9           $gate = predecessor(sig)$
10          for $0 \leq d \leq D$ do
11              $copy(gate_d, i_{t-time-1-d}, o_{t-time-d})$
12
13          if $sig$ is $Original\_Gate\_Output$ then
14              for $0 \leq d < D$ do
15                  $Insert\_Variation\_Mux_d(in : gate_d, \cdots, gate_D)$
16
17          foreach $input \in I(gate)$ do
18              if $input \notin SIG\_temp$ and $input \notin PI$ then
19                  $SIG\_temp = SIG\_temp \cup input$
20      }
21      $SIG = SIG\_temp$
22      $time++$
23  }
24  end function
```

Fig. 7. TAM + VL

time steps ago is also needed. Figure 7 describes this algorithm which extends the algorithm of Figure 5.

In lines 10-11 of Figure 7, when $d = 0$, the gate related to the corresponding time is copied. This gate is called TAM gate. After that, the behavior of the gate up to d time steps ago is also copied (lines 10-11, when $d \neq 0$). These gates are called *slowdown gates*. Parameter d is limited by a maximum timing variation D specified by the user. Given a TAM gate ($gate_0$) and its corresponding slowdown gates ($gate_1, \cdots, gate_D$), a variation multiplexer is added (lines 14-15). The variation multiplexer can skew the normal behavior of a gate by a slowdown up to D time units. The timing variation of each original gate is modeled by its corresponding variation multiplexers (line 13). Additional slowdown in the input cone of slowdown gates may accumulate to an overall slowdown up to D time units (lines 14-15, when $d \neq 0$).

Figure 4(d) shows the circuit generated by the algorithm where $D = 1$. The dashed gates are the slowdown gates. For each original gate, one variation multiplexer is inserted at the output of its corresponding TAM gate. A variation multiplexer either selects the normal behavior of a signal or the signal value one time step ago.

Overall, the generated circuit is formulated as follows:

$$\Phi = C(IN(t)) \cdot \prod_{d=1}^{D} C^d(IN(t-d)) \cdot S \tag{2}$$

$$IN(t): \; I_{t-D_s}, \, I_{t-D_s-1}, \, \ldots, \, I_{t-D_l}$$

Φ has three main parts: TAM circuit (C), slowdown circuit (C^d), and variation multiplexers (S). The parts S and C^d together are called variation logic.

The size of the TAM depends on the topology and the delay of reconverging paths of the original circuit. In the worst case, the size of the TAM may be exponentially larger than the original circuit.

B. Time Control (TC) and Variation Control (VC)

Frequency and clock period are denoted by f and T, respectively ($f = 1/T$). The task of the TC is applying a clock period T on the inputs with the accuracy of one time unit. According to Formula 1, inputs are constrained to have a constant value throughout each clock cycle.

The task of the VC is controlling the select lines of the variation multiplexers. The VC applies the maximum timing variation by the following constraint:

$$\sum_{i=1}^{n} s_i \leq D \tag{3}$$

where s_i denotes the integer value of the select lines related to one variation multiplexer.

Alternative constraints can also be added to model more complex variations. For example to apply and to control block-based variation models [1], variations of each region can be controlled by a constraint. In a hierarchical manner, the variations of different regions in each level of hierarchy (like quadtree partitioning [21]) can be correlated by additional constraints in each level.

Our approach can also be extended to consider clock skew. This can be done by adding some units for the delay of the clock network and the sequential elements.

Our model is a conservative model in the sense that it overapproximates slowdown induced under timing variations. The model allows timing variations to be activated independently in every location of a circuit. The model can be used to evaluate the worst case of a circuit functional deviation under timing variations. Here, a SAT solver is used to compute the maximum or the worst-case error under induced variations.

C. Slowdown versus Speedup

In the previous sections, we showed how the slowdown induced under timing variations is modeled. Here, we discuss how to analyze a circuit under slowdown, speedup or both.

We consider an original gate g having a specified delay n where its maximum *slowdown* and maximum *speedup* are y and x, respectively. The total *timing variation* is $D = x + y$. Therefore, the delay value of an original gate is bounded: $n - x \leq Gate\_Delay \leq n + y$.

When $x = 0$ and $y > 0$, the algorithm of Figure 7 is invoked to apply the slowdowns. Figure 6(a) shows an example path

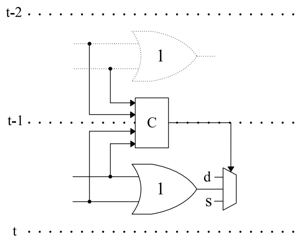

Fig. 8. Transition monitoring

and a gate with $x = 0$ and $y = 2$ such that the paths including the slowdown gates are denoted by d_1 and d_2. The variation multiplexer can skew the normal behavior (d_0) by selecting a slowdown of one time unit (d_1) or a slowdown of two time units (d_2). Also the variation multiplexer on the delay path d_1 can skew the gate behavior by another time unit such that the overall applied skew is at most 2. Here, the reference time is t, and the TC controls the frequency of previous times ($t-1$, $t-2$, ...).

When we evaluate the effect of speedup ($x > 0$ and $y = 0$), again the algorithm of Figure 7 can be reused with a minor modification. Firstly, the position of the variation multiplexer changes. Figure 6(b) shows this case by an example when $x = 2$ and $y = 0$. Now the variation multiplexer can skew the normal behavior (s_0) by selecting a speedup of one time unit (s_1) or a speedup of two time units (s_2). Also the variation multiplexer on the speedup path s_1 can skew the gate behavior by another one time unit such that the overall applied skew is at most 2. Here, also the position of the reference time t changes. In this case, TC controls the frequency of the previous times ($t-1$, $t-2$, ...) as well as the next times (t, $t+1$, $t+2$, ...).

When $x > 0$ and $y > 0$, there are both slowdown and speedup. In this case, the algorithm of Figure 7 is invoked with the input $D = x+y$. For example, when $x = 1$ and $y = 1$, the algorithm of Figure 7 is invoked with $D = 1+1 = 2$. Here, the position of the variation multiplexer as well as the reference time t change. Figure 6(c) shows this example. In this case, the variation multiplexer can skew the normal behavior (i) by a delay of one time unit (d_1) or by a speedup of one time unit (s_1). Also the variation multiplexer on the delay path d_1 can skew the gate behavior by a speedup of one time unit (i). The variation multiplexer on the speedup path s_1 can skew the gate behavior by a delay of one time unit (i).

Using VC, more complex delay models can also be handled. For modeling the behavior of a gate where the gate delay depends on the transitions of its inputs, additional constraints are needed to monitor the input's transitions and to activate the corresponding slowdown or speedup. In this case again, we consider each gate having a specified delay. But this delay may increase or decrease dependent on the input's transitions. To monitor a transition, the behavior of each gate during two consecutive time steps is needed. Therefore, in addition to the normal behavior of a gate, another copy of the gate one time step ago is created. As Figure 8 shows, one additional gate is copied. Then, the *transition constraint* C activates the select lines of the multiplexer (slowdown or speedup) dependent on the transitions.

IV. Experimental Results

We apply the proposed approach to analyze arithmetic circuits under timing variations. The experiments are carried out on a Quad-Core AMD Phenom(tm) II X4 965 Processor (3.4 GHz, 8 GB main memory) running Linux. We synthesize our circuits using Synopsys Design Compiler with Nangate 45nm Open Cell Library [22]. The techniques described in this paper are implemented using C++ in the WoLFram environment [23]. For the experiments, one time unit is $0.01\,ns$. MiniSAT is used as underlying SAT solver [24].

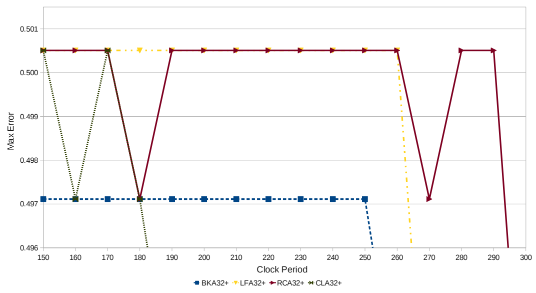

Fig. 9. Maximum error for 32-bit adders when D = 0

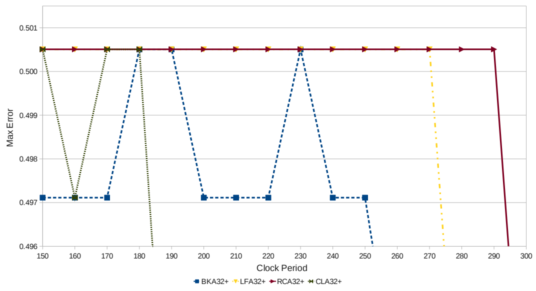

Fig. 10. Maximum error for 32-bit adders when D = 1

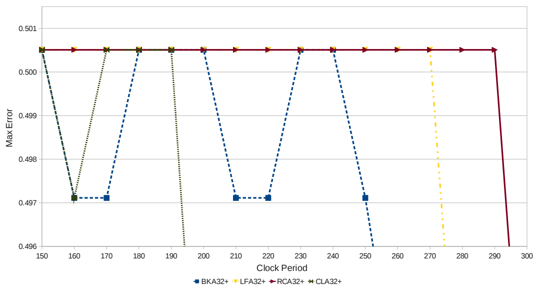

Fig. 11. Maximum error for 32-bit adders when D = 2

We evaluate *Ripple Carry Adder* (RCA), *Carry Look-ahead Adder* (CLA), *Brent-Kung Adder* (BKA), and *Ladner-Fischer Adder* (LFA) benchmarks.

As Figure 2 showed, our framework includes two side components: *spec* and *miter*. In the experiments, the original circuit is considered as a specification. The inputs of the most recent clock cycle are applied to the specification. For arithmetic circuits, we use a miter on the outputs of the specification and the TAM to measure the output deviation as the numerical difference. This miter is an integer subtractor followed by a comparison operation. In order to measure the *maximum positive error*, the miter subtracts the specification output (O) from the TAM output (O_t): ($O_t - O \geq L$). To compute the *maximum negative error*, the following miter is used: ($O - O_t \geq L$).

Here, a binary search is used to determine the maximum error. First an interval $[a, b)$ is selected such that when $L = a$, the CNF is satisfiable and when $L = b$, the CNF is unsatisfiable, i.e., the maximum error is more equal than a and less than b. In this case, the interval is divided into two smaller intervals by selecting $c = a + ((b-a)/2)$ as the middle point of the last interval. If the CNF is satisfiable with $L = c$, the next interval will be $[c, b)$, otherwise the next interval will be

TABLE I
SIZE AND TIME

Circ Name	Size when Time Unit = 0.01 ns			Time when D = 2
	Original Circ	Untimed Circ	TAM Circ	Time (s)
BKA32	136	637	7561	747
LFA32	135	634	7982	1428
RCA32	128	608	7712	868
CLA32	153	710	9154	1402

$[a, c)$. This procedure is repeated until a sufficient accuracy is obtained. Considering $[a, b)$ as the final interval, The accuracy k is specified by the length of the final interval i divided by maximum 2^n (n = number of output bits), i.e, $k = i/2^n$. In this case, b is considered as an upper bound for the maximum error.

Figure 9, 10, and 11 show the maximum positive error computed for 32-bit adders when $D = 0$, $D = 1$, and $D = 2$. For 32-bit adders, the accuracy is $k = 0.0007$ and the maximum observed error in the diagrams is 0.500. Among the 32-bit adders, CLA has the shortest delay. After CLA; BKA, LFA, and RCA have the shortest delay. Figure 9 shows the maximum error computed along overclocking. In the determined clock periods, the maximum error of LFA is 0.500, while the maximum error of BKA is 0.497. At $T = 270$ and $T = 180$, the maximum error of RCA decreases. For CLA, at $T = 180$ and $T = 160$, a decreased error is observed. While the clock period decreases, sometimes the maximum error also decreases. This is due to failing output bits that are functionally correlated as they may share certain paths.

In Figure 10, the timing variation is activated ($D = 1$). In Figure 10, at the clock period $T = 180$, CLA maximum error increases in comparison to the non-varied CLA of Figure 9. When $D = 1$, RCA maximum error increases at the clock period $T = 270$ and $T = 180$. But the errors of LFA do not increase. For BKA, at the clock period $T = 230$, $T = 190$, and $T = 180$, an increased error is observed.

When $D = 2$ (Figure 11), at the clock period $T = 190$, the CLA error increases. Also, at the clock period $T = 240$, $T = 200$, and $T = 150$, the BKA error increases. In this case, the maximum errors of RCA and LFA do not change in comparison to their errors in Figure 10 (when $D = 1$). There is always an increase if there is at least one path "about to become critical". But this may not always be the case and we have a measurement error of $k = 0.0007$. Therefore if an error increase is less than k, it is not visible in the results. As the diagrams showed, the used adders are not resilient against overclocking and timing variations. To have an improved robustness, other kinds of approximate circuits can be utilized to bound the error [25] [26].

The number of gates in the original circuit, the untimed circuit, and the TAM circuit as well as the average run times required to compute the maximum error at one clock period for 32-bit adders are shown in Table I. The number of original gates for BKA is 136, while the number of untimed gates is 637. It indicates that 501 (637 - 136) additional buffers have been added to the original circuit to create the untimed circuit. CLA has the largest number of TAM gates. This shows that in CLA there are more paths than in others which consequently increase the size of the TAM.

V. CONCLUSION

This paper introduced a methodology to model and to analyze the functional behavior of circuits under timing variations. The framework includes the following main components: *Time Accurate Model* (TAM) and *Variation Logic* (VL), *Time Control* (TC), and *Variation Control* (VC). Our framework is utilized to analyze a circuit under timing variations as well as an approximate circuit under timing-induced errors.

REFERENCES

[1] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer, “Statistical timing analysis: From basic principles to state of the art,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 589–607, 2008.

[2] M. Alioto, G. Palumbo, and M. Pennisi, “Understanding the effect of process variations on the delay of static and domino logic,” *IEEE Trans. VLSI Syst*, vol. 18, no. 5, pp. 697–710, 2010.

[3] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, “VARIUS: A model of process variation and resulting timing errors for microarchitects,” *IEEE Trans. Semiconductor Manufacturing*, vol. 21, no. 1, pp. 3–13, 2008.

[4] L. Xie and A. Davoodi, “Bound-based statistically-critical path extraction under process variations,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 30, no. 1, pp. 59–71, 2011.

[5] M. Gao, Z. Ye, Y. Peng, Y. Wang, and Z. Yu, “A comprehensive model for gate delay under process variation and different driving and loading conditions,” in *Int’l Symp. on Quality Electronic Design*, 2010, pp. 406–412.

[6] D. Ernst, N. S. Kim, S. Das, S. Pant, R. R. Rao, T. Pham, C. H. Ziesler, D. Blaauw, T. M. Austin, K. Flautner, and T. N. Mudge, “Razor: A low-power pipeline based on circuit-level timing speculation,” in *Int’l Symposium on Microarchitecture*, 2003, pp. 7–18.

[7] J. Tschanz, K. A. Bowman, C. Wilkerson, S.-L. Lu, and T. Karnik, “Resilient circuits - enabling energy-efficient performance and reliability,” in *Int’l Conf. on CAD*, 2009, pp. 71–73.

[8] S. R. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas, “EVAL: Utilizing processors with variation-induced timing errors,” in *Int’l Symposium on Microarchitecture*, 2008, pp. 423–434.

[9] M. A. Breuer, “Hardware that produces bounded rather than exact results,” in *Design Automation Conf.*, 2010, pp. 871–876.

[10] N. R. Shanbhag, R. A. Abdallah, R. Kumar, and D. L. Jones, “Stochastic computation,” in *Design Automation Conf.*, 2010, pp. 859–864.

[11] S. T. Chakradhar and A. Raghunathan, “Best-effort computing: rethinking parallel software and hardware,” in *Design Automation Conf.*, 2010, pp. 865–870.

[12] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, “MACACO: Modeling and analysis of circuits for approximate computing,” in *Int’l Conf. on CAD*, 2011, pp. 667–673.

[13] D. Shin and S. K. Gupta, “Approximate logic synthesis for error tolerant applications,” in *Design, Automation and Test in Europe*, 2010, pp. 957–960.

[14] ——, “A new circuit simplification method for error tolerant applications,” in *Design, Automation and Test in Europe*, 2011, pp. 1566–1571.

[15] A. Lingamneni, C. Enz, J.-L. Nagel, K. Palem, and C. Piguet, “Energy parsimonious circuit design through probabilistic pruning,” in *Design, Automation and Test in Europe*, 2011, pp. 764–769.

[16] L. Wan and D. Chen, “Dynatune: Circuit-level optimization for timing speculation considering dynamic path behavior,” in *Int’l Conf. on CAD*, 2009, pp. 172–179.

[17] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, “Slack redistribution for graceful degradation under voltage overscaling,” in *ASP Design Automation Conf.*, 2010, pp. 825–831.

[18] L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra, “ERSA: Error resilient system architecture for probabilistic applications,” in *Design, Automation and Test in Europe*, 2010, pp. 1560–1565.

[19] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, “Scalable effort hardware design: exploiting algorithmic resilience for energy efficiency,” in *Design Automation Conf.*, 2010, pp. 555–560.

[20] K. P. Ganeshpure and S. Kundu, “On ATPG for multiple aggressor crosstalk faults,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 29, no. 5, pp. 774–787, 2010.

[21] A. Agarwal, D. Blaauw, and V. Zolotov, “Statistical timing analysis for intra-die process variations with spatial correlations,” in *Int’l Conf. on CAD*, 2003, pp. 900–907.

[22] *Nangate 45nm Open Cell Library*, http://www.nangate.com.

[23] A. Sülflow, U. Kühne, G. Fey, D. Groβe, and R. Drechsler, “WoLFram – a word level framework for formal verification,” in *IEEE/IFIP Int’l Symposium on Rapid System Prototyping*, 2009, pp. 11–17.

[24] N. Eén and N. Sörensson, “An extensible SAT solver,” in *SAT 2003*, ser. LNCS, vol. 2919, 2004, pp. 502–518.

[25] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, “Design of voltage-scalable meta-functions for approximate computing,” in *Design, Automation and Test in Europe*, 2011, pp. 950–955.

[26] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, “IMPACT: imprecise adders for low-power approximate computing,” in *ISLPED*, 2011, pp. 409–414.