ForTER: A Forward Error Correction Scheme for Timing Error Resilience

Jie Zhang, Feng Yuan, Rong Ye and Qiang Xu
CUhk REliable Computing Laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {jzhang, fyuan, rye, qxu}@cse.cuhk.edu.hk

ABSTRACT

With technology scaling, integrated circuits suffer from increasingly severe static and dynamic variations, which often manifest themselves as infrequent timing errors on circuit speed paths, if a large timing guard-band is not reserved. This paper presents a new forward timing error correction scheme, namely ForTER*, which predicts whether the occurrence of timing errors would propagate to the next level of sequential elements and corrects them without necessarily borrowing timing slack. The proposed technique can be combined with other timing error resilient circuit design techniques to further improve circuit performance, as demonstrated in our experimental results with various benchmark circuits.*

1. INTRODUCTION

With technology scaling, the timing behavior of integrated circuits (ICs) becomes increasingly uncertain due to static process variation and various dynamic variation effects such as voltage/temperature fluctuations [1–3]. Conventional circuit designs tolerate these variations by embedding a large timing guard-band into the design to ensure error-free computing. Unfortunately, such conservative design methodology reduces the benefits provided by technology scaling. Consequently, resilient design methodologies for timing errors have emerged as alternative solutions and have attracted lots of attention.

1.1 Related Work

Generally speaking, prior works achieve timing error resilience with the following three techniques: (i) timing error recovery by restoring the system to a pre-error state and conducting re-execution with more timing margin; (ii) timing error masking by adding redundant logic; (iii) time borrowing by delaying the arrival time of the correct data to the next logic level.

Timing Error Recovery: One well-known solution to achieve timing error resilience is the so-called *timing speculation* technique. Timing-speculative circuits usually employ double-sampling latches to detect timing errors on circuit speed paths (e.g., *Razor* [4]). Once an error is detected, backward error recovery (BER) is conducted to restore the system to a known-good pre-error state. Thanks to the built-in roll-back support in microprocessors, it is quite cost-efficient to implement a BER solution for timing errors occurred on their datapath. That is, we can simply flush the pipeline and replay the offending instructions (usually at a lower frequency) for timing error correction. Recently, Intel presented the measurement results of a timing-speculative microprocessor test chip in [5], showing that the resilient design is able to achieve more than 30% throughput gain over a conventional design that does not allow timing errors. Such huge benefits have motivated a large amount of recent research efforts on design and optimization techniques for timing-speculative circuits (e.g., [6–12]).

However, for general logic circuits (including microprocessor control path), it is very difficult, if not impossible, to implement a BER solution for timing error resilience, due to the high cost to checkpoint error-free states without microarchitecture support.

Timing Error Masking: Instead of rolling back the system to a pre-error state, timing error masking techniques add a redundant logic block to overwrite the outputs of the circuit upon application of inputs that sensitize speed paths [13, 14]. With such exact sensitization constraint, the redundant *error-masking circuit* tends to have more timing slack when compared to the original circuit, and hence is immune to timing errors.

In [13], the so-called *speed-path characteristic function* that represents the set of all speed-path activation patterns is used to synthesize the error-masking circuit, which is quite expensive in terms of runtime and cost overhead. Recently, InTimeFix [14] proposed to add fine-grained redundant *approximation circuit* into the design to provide more timing slack for circuit speed paths.

Time Borrowing: For those flip-flops (FFs) driven by circuit speed paths, denoted as *suspicious FFs* (SFFs), if they are followed by non-critical paths, we can replace them with sequential elements having time-borrowing capability and correct timing errors by delaying the arrival time of the correct data to the next logic level (e.g., [15, 16]).

While effective in many cases, such time-borrowing techniques have the inherent weakness of error effect propagation. That is, even if a suspicious FF can borrow some time from its successive logic level, the timing slack of this level is reduced. Therefore, some initially non-suspicious FFs in this successive level may become suspicious ones and need to be replaced by sequential elements with time-borrowing capability again. [17] proposed a so-called *TIMBER* technique that is able to recover from two-stage timing errors, but it can only mitigate the problem instead of solving it. Moreover, all these works involve rather complicated clock control and analysis, limiting their applicability in real designs.

1.2 Summary of Contributions

In this paper, we propose a new forward error correction (FEC) scheme for timing error resilience, namely *ForTER*, which predicts whether the timing errors occurred on SFFs would propagate to the next level of sequential elements, denoted as *affected FFs* (AFFs). If they do affect a particular AFF and the timing error indeed occurs, we simply invert the original AFF value calculated with the erroneous value in the SFFs. It is important to know that we have an almost full clock period for prediction logic since it has nothing to do with whether there is a timing error or not. Therefore, the proposed FEC technique does not require complex clock control and does not reduce the timing slack of any level, when compared to time borrowing tech-

978-1-4799-1071-7/13/$31.00 ©2013 IEEE.

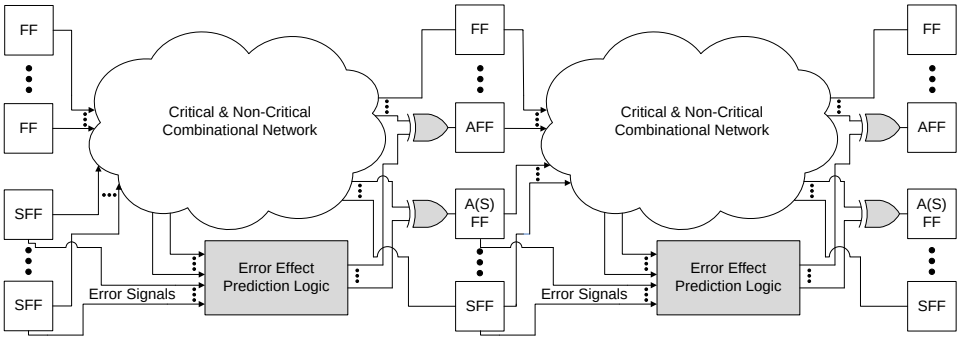

Figure 1: The overall architecture of the proposed forward timing error correction scheme, *ForTER*

niques. Note, however, SFFs need to be equipped with timing error detection capability (e.g., double-sampling latches as in [4, 5]).

The main limitation of the proposed FEC scheme is its associated hardware cost because it needs to be introduced to protect all AFFs of a particular SFF. As a result, it is usually not cost-efficient to serve as a standalone solution to achieve timing error resilience. However, as a general FEC solution, it can be flexibly combined with other resilient design techniques such as timing speculation and timing error masking to further optimize the circuit. The main contributions of this work include:

- we propose a novel FEC scheme, *ForTER*, for timing error resilience, in which we construct the timing error effect prediction logic based on *Boolean Differential Equation* (BDE) and use it to achieve timing error resilience;
- we present a cost-efficient implementation to apply *ForTER* into timing-speculative circuit to achieve better system throughput;
- we show how to apply *ForTER* into timing-speculative circuit to trade-off performance and cost and how to combine *ForTER* with InTimeFix to achieve better timing error resilience for general circuits.

The remainder of this paper is organized as follows. In Section 2, we discuss the hardware architecture of the proposed FEC framework in detail. Next, we show the application of *ForTER* in the timing-speculative circuits and the general circuits in Section 3 and Section 4, respectively. Experimental results on various benchmark circuits are then presented in Section 5. Finally, Section 6 concludes this paper.

2. PROPOSED ForTER ARCHITECTURE

The overall architecture of the proposed FEC framework for timing error resilience, namely *ForTER*, is shown in Fig. 1. As can be observed, all the SFFs are equipped with timing error detection capability (e.g., implemented as Razor flip-flops [4]) and the error signals from SFFs indicate whether timing errors occur or not. *ForTER* protects SFFs by correcting the timing error at their corresponding AFFs. However, it is important to note that the effect of the timing error might be masked by side-inputs on the propagation path. Therefore, the error effect prediction logic is designed to predict the propagation of such error effect. To be specific, for a particular AFF, if the prediction logic indicates that its value is affected by some SFFs which have the timing errors at a particular cycle, then the value of the AFF is reversed to correct the error; otherwise it keeps unchanged. This is achieved by adding a two-input XOR gate before the AFF whose inputs are the original input to the AFF and the error effect prediction logic output.

Let us use the example shown in Fig. 2 to briefly explain how $ForTER$ works. Suppose $FF2$ is an SFF with timing error detection capability, wherein the *Error Signal* indicates the occurrence of the timing error. The timing error effect of $FF2$ would propagate to its AFF, $FF4$, under the condition that a timing error occurs on $FF2$ and the side-input of the *OR* gate is a non-controlling value (i.e., logic

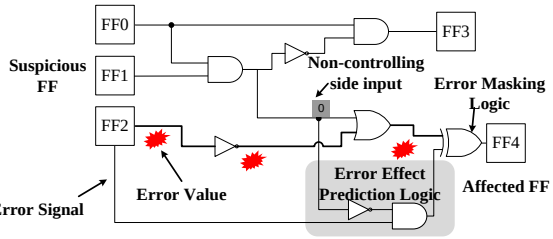

Figure 2: Timing error correction with *ForTER*: an example

'0') at the same time. To correct such error effect, we only need to invert the current value of $FF4$ calculated with the erroneous value of $FF2$. The error prediction logic is constructed accordingly and it outputs logic '1' when the above condition is satisfied. One may argue for this example, simply delaying the clock to $FF2$ could achieve timing error resilience without adding much overhead. This is true. However, for complex industrial designs, time borrowing techniques require to perform the accurate timing analysis and conduct the complicated clock control, which is a challenging task, especially considering the ever-increasing circuit variations. The proposed *ForTER* technique eliminates such needs and thus is a preferred solution.

In the rest of this section, we discuss a general method to construct the error effect prediction logic and the corresponding FEC circuit.

2.1 Timing Error Effect Prediction with BDE

We build our timing error effect prediction logic based on Boolean differential equation. Let us briefly introduce it here first. BDE, as an analytical methodology, is widely used in the synthesis, verification and testing of digital circuits. According to [18], we have the following two definitions:

DEFINITION 1. *The Boolean Difference of a Boolean function* $f(\mathbf{x})$ *with respect to a single variable* x_i *is*

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = f_{x_i} \oplus f_{\bar{x}_i}, \quad \mathbf{x} = (x_0, x_1, ..., x_i, ..., x_n), \tag{1}$$

where $\mathbf{x}$ *denotes input array with n variables,* $f_{x_i} = f(x_0, x_1, ..., x_i = 1, ..., x_n)$, $f_{\bar{x}_i} = f(x_0, x_1, ..., x_i = 0, ..., x_n)$, *and* $\oplus$ *is XOR operation.*

Eq. 1 is called *Simple Boolean Differential Equation*, which has the following property:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \begin{cases} 1, & \text{transition of } x_i \text{ affects the value of } f \\ 0, & \text{transition of } x_i \text{ does not affect the value of } f \end{cases}, \tag{2}$$

and transition could be either $0 \rightarrow 1$ or $1 \rightarrow 0$.

Furthermore, BDE can be also used to check concurrent transitions on multiple variables. Assuming $\mathbf{x_s} = (x_{s0}, x_{s1}, ..., x_{sm})$ is a subset of the entire input variable set $\mathbf{x}$,

DEFINITION 2. *The Boolean Difference of a Boolean function* $f(\mathbf{x})$ *respect to a set of variables* $\mathbf{x_s}$ *is*

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x_s}} = f_{\mathbf{x_s}} \oplus f_{\bar{\mathbf{x}}_\mathbf{s}}, \quad \mathbf{x_s} = (x_{s1}, x_{s2}, \ldots, x_{sm}), \tag{3}$$

$f_{\mathbf{x_s}}$ and $f_{\bar{\mathbf{x}}_\mathbf{s}}$ are functions whose variables in $\mathbf{x_s}$ have opposite values.

Eq. 3 is called *Vertical Boolean Differential Equation*. Since variables in $\mathbf{x_s}$ can have 2^m combinations, Eq. 3 can be expanded as,

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x_s}} = \begin{cases} f_{(x_{s0}x_{s1}\ldots x_{sm})} \oplus f_{(\bar{x}_{s0}\bar{x}_{s1}\ldots\bar{x}_{sm})} \\ f_{(\bar{x}_{s0}x_{s1}\ldots x_{sm})} \oplus f_{(x_{s0}\bar{x}_{s1}\ldots\bar{x}_{sm})} \\ \vdots \qquad\qquad \vdots \\ f_{(x_{s0}x_{s1}\ldots\bar{x}_{sm})} \oplus f_{(\bar{x}_{s0}\bar{x}_{s1}\ldots x_{sm})} \end{cases}. \tag{4}$$

Each equation in Eq. 4 has similar properties to detect the transition of variables in $\mathbf{x_s}$ as shown in Eq. 2.

For a given Boolean function, the basic principle of BDE is to verify whether the output value is affected by the transition of one or some input variables, which perfectly matches our needs on error effect prediction, wherein we would like to know whether the transition of certain SFFs would change their corresponding AFFs.

Without loss of generality, consider the Boolean function of the AFF, f, and timing errors occur concurrently on a set of inputs denoted as $\mathbf{x_s}$. The error effect can then be predicted by one of the equations in Eq. 4, determined by the error state of $\mathbf{x_s}$. Let us consider the following simple example. If inputs x_{s_0} and x_{s_1} have timing errors, and the error state is $(x_{s_0}=0,\ x_{s_1}=1)$, which means that the correct state should be $(x_{s_0}=1,\ x_{s_1}=0)$. This error effect can be predicted by $f_{(\bar{x}_{s_0}x_{s_1})} \oplus f_{(x_{s_0}\bar{x}_{s_1})}$, indicating whether the correct state would lead to the change of the output based on the erroneous state. To give the complete error effect prediction function, the BDE should cover all error states. Suppose the function f has n suspicious inputs. Then, there are C_n^p suspicious input combinations for any p of n suspicious inputs occurring errors concurrently. For each combination, there are 2^{p-1} error states to be handled. Therefore, in all, there are $\sum_{p=1}^{n} C_n^p 2^{p-1}$ possible error states for n suspicious inputs. At last, the error effect prediction function for the AFF is to combine the BDE with suspicious signals and error signals that indicate which error state appears.

2.2 Cost-Efficient ForTER Implementation

To implement the error effect prediction logic for a particular AFF, a straightforward method is to realize it along with the original logic, which, however, may introduce quite high hardware cost. This motivates us to propose a novel cost-efficient implementation scheme.

Before discussing the details, let us take an example shown in Fig. 3 to illustrate the idea of our cost-efficient implementation. Consider one function of AFF, f, that is driven by one SFF, x_{s_0}. According to BDE, its error effect prediction function, f_p, can be given as:

$$f_p = e_{x_{s_0}}(f_{x_{s_0}} \oplus f_{\bar{x}_{s_0}}),$$

where $e_{x_{s_0}}$ denotes the error signal and $e_{x_{s_0}}=1$ if x_{s_0} encounters the timing error. If $f_p=1$, the timing error of x_{s_0} would erroneously change the value of the original function; otherwise not. For the original function, f, it can be re-written according to Shannon's decomposition theorem as:

$$f = x_{s_0} f_{x_{s_0}} + \bar{x}_{s_0} f_{\bar{x}_{s_0}}.$$

Examination of the above two equations together shows that they share two sub-functions, $f_{x_{s_0}}$ and $f_{\bar{x}_{s_0}}$. This motivates us to generate the shared sub-functions first and thereafter construct the original logic and error prediction logic with some additional gates separately to save the hardware cost, as shown in Fig. 3. Moreover, it is worth noting that $f_{x_{s_0}}$ and $f_{\bar{x}_{s_0}}$ can further share some common logics, since they come from the same original function. At last, the error masking logic corrects possible timing error effect by *XOR*ing the outputs of original logic and error effect prediction logic.

When the AFF is driven by multiple SFFs, the error effect prediction logic needs to deal with a number of error states as discussed earlier. Let us start with the Boolean function that contains two suspicious input variables (i.e. $\mathbf{x_s}=(x_{s0},x_{s1})$) as an example. According

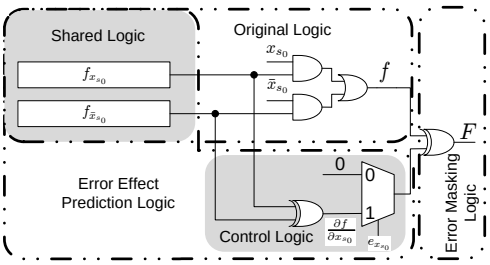

Figure 3: *ForTER* implementation for an AFF with one SFF (x_{s_0}).

to Eq. 1 and Eq. 3, the basic sub-function used in BDEs are $f_{x_{s_0}}$, $f_{\bar{x}_{s_0}}$, $f_{x_{s_1}}$, $f_{\bar{x}_{s_1}}$, $f_{(x_{s_0}x_{s_1})}$, $f_{(x_{s_0}\bar{x}_{s_1})}$, $f_{(\bar{x}_{s_0}x_{s_1})}$, and $f_{(\bar{x}_{s_0}\bar{x}_{s_1})}$, wherein the first four subfunctions can be easily constructed by the last four sub-functions with x_{s_0} and x_{s_1}. In order to save hardware cost, the original function can be decomposed with respect to x_{s_0} and x_{s_1} according to the Shannon's decomposition theorem:

$$f = x_{s_0}x_{s_1}f_{(x_{s_0}x_{s_1})} + \bar{x}_{s_0}x_{s_1}f_{(\bar{x}_{s_0}x_{s_1})} + x_{s_0}\bar{x}_{s_1}f_{(x_{s_0}\bar{x}_{s_1})} + \bar{x}_{s_0}\bar{x}_{s_1}f_{(\bar{x}_{s_0}\bar{x}_{s_1})},$$

and the error prediction function is given as:

$$\begin{aligned} f_p =& e_{x_{s_0}}\bar{e}_{x_{s_1}}(f_{x_{s_0}} \oplus f_{\bar{x}_{s_0}}) + \bar{e}_{x_{s_0}}e_{x_{s_1}}(f_{x_{s_1}} \oplus f_{\bar{x}_{s_1}}) + \\ & e_{x_{s_0}}e_{x_{s_1}}[(x_{s_0} \oplus x_{s_1})(f_{x_{s_0}x_{s_1}} \oplus f_{\bar{x}_{s_0}\bar{x}_{s_1}}) + (x_{s_0} \oplus \bar{x}_{s_1})(f_{\bar{x}_{s_0}x_{s_1}} \oplus f_{x_{s_0}\bar{x}_{s_1}})]' \end{aligned}$$

where four subfunctions can be further shared. In general, for an AFF function with n SFFs, there are 2^n shared subfunctions that can be obtained by decomposing the original function with respect to n suspicious inputs in the same way. After building shared subfunctions, the error effect prediction logic and the original logic can be automatically constructed according to BDEs and the decomposed original function. Note that, the above implementation has impact on the circuit timing and may render a non-suspicious FF becoming a suspicious one. We would stop using *ForTER* to protect AFFs under such circumstances.

2.3 ForTER Cost Analysis

The hardware cost of *ForTER* is composed of two parts: the error detection hardware and the error correction hardware, given as:

$$Cost_{ForTER} = N_{SFF}Cost_{Razor} + \sum_{i=1}^{M_{AFF}} Cost_{AFF_i}, \tag{5}$$

where N_{SFF} and M_{AFF} denote the number of SFFs and AFFs, respectively, and $Cost_{Razor}$ and $Cost_{AFF_i}$ denote the extra cost of Razor-like sequential elements for error detection when compared to standard flip-flop and the cost for error correction at AFFs. Note that, when applying *ForTER* for timing-speculative circuits, only the latter cost exists because such circuits need to have built-in timing error detection capability and are already equipped with error detection sequentials.

$Cost_{AFF_i}$ is related to the number of SFFs driving AFF_i. As shown in Fig. 3, the cost for error correction at a particular AFF mainly comes from two parts, the reconstructed original logic according to the new signal order and the error effect prediction logic. In particular, the cost of the error effect prediction logic for an AFF increases significantly with the number of SFFs that drive it, because the BDE in the error effect prediction logic becomes much more complicated with the increase of SFFs.

From the above, solely applying *ForTER* to achieve timing error resilience may result in quite high hardware cost even with the proposed cost-efficient implementation, especially when the circuit contains many SFFs. Nevertheless, with its unique advantage of being able to mask timing error effects without performing accurate timing analysis and complex clock control, it is an attractive solution to be combined with other timing error-resilient design techniques, as shown in the following two sections.

3. ForTER FOR TIMING-SPECULATIVE CIRCUITS

In this section, we study the application of *ForTER* on timing-speculative circuits. As protecting SFFs would introduce hardware cost, the key issue is to protect a selected set of SFFs under a hardware cost constraint to minimize the timing error rate in such circuits.

3.1 Problem Formulation

Given a circuit with n SFFs, we use $S = (s_1, s_2, ..., s_n)$ to represent the SFF list, wherein $s_i = 1$ indicating SFF_i is chosen to be protected by *ForTER* while $s_i = 0$ indicating it is not chosen. We use $E_S = (e_{s_1}, e_{s_2}, ..., e_{s_n})$ to represent the list of error rates of SFFs which are estimiated according to [11]. With n SFFs, we assume this circuit has m AFFs driven by them.

With the above definitions, our optimization problem is to minimize the timing error rate of the circuit subject to a specified hardware cost constraint, given as:

$$\begin{aligned} &\text{Objective: } \min_{\forall S} E = (I-S)E_s^T, \quad s_i \in \{0,1\} \\ &\text{Constraint: } \sum_{i=1}^{m} Cost_{AFF_i} < Cost_{ex} \end{aligned}, \tag{6}$$

where $Cost_{AFF_i}$ is the cost used by *ForTER* to protect AFF_i, $Cost_{ex}$ is the cost constraint, and $I = (1,1,...,1)_{1\times n}$. Note that, $(I-S)E_s^T$ represents the sum of timing error rates of SFFs, implying that if a certain SFF is chosen to be protected by *ForTER*, it does not contribute to the sum of error rates. As discussed earlier, the cost of *ForTER* mainly results from the reconstructed original logic and the error effect prediction logic, and they are estimated with a weighted value for the extra logic elements introduced by $ForTER$.

3.2 Optimization Algorithm

The above optimization problem can be modeled as the well-known Knapsack problem [19], which has been proved to be NP-complete. SFFs are considered as the items to be packed, with error rate reduction as their profits and protection cost as weights. The cost constraint is considered as bag capacity. If a certain item is put into the bag, that means we select a certain SFF to be protected by *ForTER*. Thus, the optimization objective is to maximize the total profit by selecting items into the bag. It is worth noting that the above problem is not exactly the same with the conventional formulation of Knapsack problem. This is because, in our targeted problem, the profits and weights are correlated, resulting from the fact that SFFs could share the prediction logic of *ForTER*. With a different protection scheme, the protection cost for the same SFF can be different.

To solve the above Knapsack problem, we resort to a branch-and-bound algorithm (BB), which divides the Knapsack problem into sub-Knapsack problems iteratively. At each step, BB tries to protect one SFF, and discards some subproblems based on a bounding function that estimates the lower bound of error rate achieved by the subproblem under cost constraint. The key difference between the original problem and the subproblem is that, in the subproblem, some AFFs have already been equipped with error effect prediction logic that may be shared with some other SFFs later. Thus, it is difficult to accurately estimate the protection costs of SFFs.

To give the bounding function, let us first consider a certain subproblem, in which, $S_r = \{s_i, s_{i+1}, ..., s_k\}$ denotes the list of un-protected SFFs and C_r denotes the remaining cost. All unprotected SFFs are sorted according to their profit-weight ratios, and then we have

$$\frac{e_{s_i}}{c_{s_i}} > \frac{e_{s_{i+1}}}{c_{s_{i+1}}} > ... > \frac{e_{s_k}}{c_{s_k}},$$

where c_{s_i} is the protection cost of SFF s_i, estimated based on the existing constructed error effect prediction logic. The lower bound of the subproblem can be obtained greedily by protecting SFFs in the descending order according to their profit-weight ratios. Assume SFF s_j is the first one that cannot be protected because of the cost constraint. The lower bound of the error rate E_{S_r} for this subproblem is given as:

$$E_{S_r} = e_j \frac{C_r - \sum_{k=i}^{j-1} c_k}{c_j} + e_{j+1} + e_{j+2} + ... + e_k. \tag{7}$$

Interested readers may refer to [19] for the proof.

With above, the optimization algorithm based on BB algorithm is as follows. We first adopt the greedy algorithm to determine the baseline solution, treated as the initial best solution. The greedy algorithm takes items in the sorted order according to their profit-weight ratios until no more items can be taken. Then, BB algorithm starts with no SFFs chosen to be protected. In each search process, BB prunes some SFFs by comparing the lower bound of the subproblem with that of the best solution. If this bound is less than that of the best one, it goes into the subproblem; otherwise it does not. For each searched solution, we record it if it is the best one among all the searched solutions. The BB stops until there is no enough cost to protect any more SFFs.

4. ForTER FOR GENERAL LOGIC CIRCUITS

Timing speculation is an effective technique to achieve timing error resilience for circuits with built-in rollback mechanism, e.g., microprocessor datapath. For general logic circuits, however, it is usually not possible to checkpoint the entire system states and low-cost timing error masking techniques are preferable. In this section, we present how to combine *ForTER* with the low-cost timing error masking solution, InTimeFix [14], to further optimize circuit timing.

4.1 InTimeFix

To get a better understanding of the proposed solution, we first briefly discuss InTimeFix.

For a SFF with Boolean function F driven by speed paths, InTimeFix employs two simplified approximation logics $G0$ and $G1$ to calculate correct results to mask timing errors on it. This is achieved by constructing a functionally-equivalent yet timing-improved circuit. With the approximation logic, circuit speed paths in F become false paths while those non-critical paths in F remain true. Approximation logic functions of $G0$ and $G1$ are defined as: $G0$ is *0-approximation* of F if $G0 = 0 \Rightarrow F = 0$ (i.e., $G0 = 0$ implies $F = 0$); similarly, $G1$ is *1-approximation* of F is $G1 = 1 \Rightarrow F = 1$.

Given a targeted timing slack improvement threshold, InTimeFix proposed a heuristic algorithm to construct approximation logics to cover those minterms that sensitize circuit speed paths. It also guarantees that the delay of the approximation logics do not introduce any new speed paths. While in most cases, it can improve timing slack with low hardware cost, its effectiveness is related to the speed path structure. In the extreme case, for example, if a SFF is driven by one speed path containing only a series of AND gates, the corresponding 1-approximation logic would be the duplication of this circuit path, which cannot provide any timing slack.

4.2 Problem Formulation

For a given SFF, we can either mask the timing error at its fan-in logic cone with InTimeFix, or predict and correct its error effects at the AFFs of its fan-out logic cone with *ForTER*. Consequently, InTimeFix and *ForTER* are two complementary techniques that can be naturally combined to achieve better timing error resilience.

Consider a general logic circuit with n SFFs, $S = (s_1, s_2, \ldots, s_n)$, our optimization problem is to minimize the maximum path delay subject to the hardware cost by selectively protecting a set of SFFs with either InTimeFix or *ForTER*, given as:

$$\begin{aligned} &\text{Objective: } \min_{\forall S}(\text{Max}(d_p)) \\ &\text{Constraints: } Cost_{InTimeFix} + Cost_{ForTER} < Cost_{ex} \end{aligned}, \tag{8}$$

where $\text{Max}(d_p)$ denotes the maximum path delay without timing error protection and $Cost_{ex}$ denotes the cost constraint.

4.3 Optimization Algorithm

We adopt a heuristic algorithm to solve the above optimization problem. In the beginning, all SFFs are sorted in the descending order according to their timing slack. Then, we start from the SFF driven by the most critical speed path and determine to adopt either InTimeFix or *ForTER* to protect it. For each SFF, if InTimeFix can protect it and cost less hardware than *ForTER*, we choose InTimeFix; otherwise we use *ForTER*. Before protecting this SFF by *ForTER*, we determine whether SFFs that have been protected by *ForTER* can be partially protected with InTimeFix in order to save the cost to protect this SFF (to be discussed in the following paragraph). After protecting each SFF, we check whether the cost constraint is satisfied. If it is not, we discard the protection for this SFF and terminate the algorithm; otherwise we try the next SFF.

As 0-approximation logic $G0$ and 1-approximation logic $G1$ used in InTimeFix are separate, we can select to use only one of them. With $G0$, timing error occurs only for $0 \rightarrow 1$ transition; with $G1$, the timing error occurs only for $1 \rightarrow 0$ transition. Based on the above observation, we can achieve further benefits by protecting a SFF jointly with $ForTER$ and InTimeFix for the following reasons: (i) it is likely that the hardware cost and timing slack with $G0$ and $G1$ are quite unbalanced, and selectively using the one with lower cost and leaving the other transition to be protected by $ForTER$ may lead to more benefits; (ii) with only one possible timing error, the error prediction logic for $ForTER$ can be dramatically reduced.

5. EXPERIMENTAL RESULTS

In this section, we demonstrate the effectiveness of the proposed *ForTER* solution by conducting experiments with several large ISCAS'89 and IWLS'05 benchmark circuits. We first use a commercial logic synthesis tool to optimize the circuits for their timing performance. *ForTER* logic is then synthesized into the circuit by adapting an open-source synthesis tool ABC [21].

5.1 Results on Timing-Speculative Circuits

In this subsection, we present the results when applying *ForTER* on timing-speculative circuits[1], which can roll back the system once a timing error is detected. We try to optimize the circuit throughput, given as [12]: $\min_{CP}[(1 + error(CP) \cdot penalty) \cdot CP]$, wherein CP is the operational clock period, $error(CP)$ is the percentage of cycles with timing errors, and $penalty$ is the penalty for error correction. Similar to [12], we assume the penalty to be 10 cycles. To get $error(CP)$, we sweep the operational clock period CP and perform timing simulation with one million random input patterns.

Experimental results are reported in Table 1. *Original* denotes the case for conventional designs that do not allow timing errors to occur. We employ error recovery mechanism without and with *ForTER* and denote these two solution as BER and *ForTER*, respectively. As can be observed from Table 1, when compared to *Original*, BER leads to 7.05% throughput improvement on average, which justifies the effectiveness of timing speculation technique. As for *ForTER*, we present two cases with hardware cost constraints of 5% and 10%, respectively. Under 5% constraint, *ForTER* achieves 20.56% throughput improvement over *Original* and 12.61% additional throughput improvement over BER, which proves the efficiency of *ForTER* on optimizing timing-speculative circuits. Relaxing the cost constraint to 10% leads to an additional 6-7% throughput gain over BER. We can find out that the increase of throughput gain does not keep the same pace as the increase of hardware cost. This is because our algorithm targets those SFFs with relatively high error probability and low cost for correction first. A close examination of experimental results shows that the effectiveness of *ForTER* varies among different benchmark circuits. For example, the throughput improvement for *s38417* with *ForTER* is much higher than that for *wb_conmax*. We attribute such difference to the unique path delay distribution of each benchmark.

[1]We assume these benchmark circuits have built-in error recovery scheme.

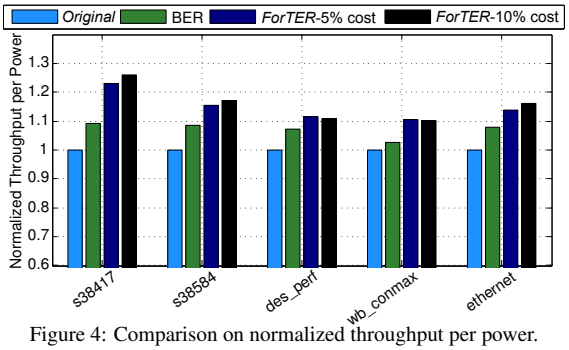

Figure 4: Comparison on normalized throughput per power.

Finally, to evaluate the power impact with *ForTER*, we present experimental results on throughput per power (TPP) in Fig. 4. The reported TPP values are normalized with respect to the case of *Original*. As can be clearly seen from the figure, *ForTER* is able to achieve much higher TPP than both *Original* and BER. In particular, *ForTER* achieves more than 20% improvement over BER for *s38417*. It is worth noting that, in most cases, *ForTER* with 10% constraint outperforms the case with 5% constraint in terms of TPP. This is because, even though we get diminishing returns for throughput gain with the increase of hardware cost, its improvement percentage is still usually higher than the cost increment.

5.2 Results on General Logic Circuits

In this subsection, we present the results when applying *ForTER* in general logic circuits that do not have rollback recovery capability. We combine *ForTER* and InTimeFix (denoted as "*ForTER*") and compare it with the case when only applying InTimeFix (denoted as "*Baseline*"). According to [4], the hardware cost to equip a SFF with timing error detection capability is assumed to be 10 gates.

In Table 2, we report the minimum clock period, the relaxed timing slack with resilient design techniques *Baseline* and $ForTER$, and their associated hardware cost. We set the hardware cost constraint to be only 2%, considering the low cost of InTimeFix. As can be seen, the *Baseline* solution results in 11.81% timing slack compared to the original design (denoted as *Original*) on average, while $ForTER$ has 14.57% timing slack. The additional 2.76% slack justifies the effectiveness of *ForTER*. Note that, to apply *ForTER*, only another 0.48% hardware cost is required when compared to the *Baseline*.

A close examination of Table 2 shows the following interesting observations. Firstly, for *s38417*, $ForTER$ achieves more timing slack with less hardware cost, due to the fact that some of the SFFs in *s38417* require *Baseline* to pay more hardware than $ForTER$. Secondly, for *s38584* and *ethernet*, *Baseline* stops its optimization with very low hardware cost (about 0.1%) because there exist one SFF whose delay cannot be reduced at all. $ForTER$, on the other hand, is able to conduct further optimization with more hardware cost. Thirdly, for *des_perf* and *wb_conmax*, *ForTER*, unfortunately, cannot lead to any further timing slack improvement under the specified 2% hardware cost constraint. Consequently, we relax the cost constraint for these two benchmarks to check whether *ForTER* can lead to any improvement, and results are plotted in Fig. 5. As can be observed, for *des_perf*, *ForTER* starts to show benefits when the hardware constraint is relaxed to be more than 6%; for *wb_conmax*, *Basline* is not effective with more than 4% hardware while $ForTER$ can still be used to dramatically improve timing slack.

Finally, we report the power values in Fig. 6, under hardware constraint 2%. They are normalized with respect to the original design and we can find that both *Baseline* and $ForTER$ only increase circuit power consumption slightly.

6. CONCLUSION

In this paper, we have proposed a novel forward timing error correction scheme, namely *ForTER*, which predicts whether the occurrence of timing errors would propagate to the next level of sequen-

Circuit	Circuit size	*Original*	BER		*ForTER*							
					5% Cost Constraint				10% Cost Constraint			
	(# of gates)	$Th.(MHz)$	$Th.(MHz)$	$\Delta_{BW}(\%)$	$Th.(MHz)$	$\Delta_{FW}(\%)$	$\Delta_{FB}(\%)$	$\phi_F(\%)$	$Th.(MHz)$	$\Delta_{FW}(\%)$	$\Delta_{FB}(\%)$	$\phi_F(\%)$
s38417	24370	148.68	162.41	9.24	191.54	28.83	17.94	4.93	205.97	38.54	26.82	9.87
s38584	21066	142.29	154.43	8.53	173.026	21.60	12.04	5.07*	183.50	28.97	18.83	10.21*
des_perf	154323	150.083	153.81	2.49	174.05	15.97	13.16	5.13*	181.82	21.15	18.21	9.88
wb_conmax	75352	69.34	74.31	7.17	81.11	16.98	9.15	4.85	84.67	22.11	13.94	10.18*
ethernet	157841	79.90	86.16	7.83	95.44	19.44	10.77	4.76	101.66	27.23	17.99	9.51
Average				7.05		20.56	12.61	4.95		27.60	19.16	9.93

$Th.$: throughput; Δ_{BW}: *BER* throughput improvement ratio over *Original*; Δ_{FW}: *ForTER* throughput improvement ratio over *Original*; Δ_{FB}: *ForTER* throughput improvement ratio over *BER*; ϕ_F: *ForTER* hardware cost ratio.
*: These results do not strictly satisfy the constraint because cost is estimated during optimization.

Table 1: Experimental results on the throughput and hardware cost for timing-speculative circuits.

Circuit	Circuit Size	*Original*	Relaxed Slack				Hardware Cost			
			Baseline		*ForTER*		Baseline		*ForTER*	
	(# of gates)	CP (ns)	σ (ns)	$\Delta_{IW}(\%)$	σ (ns)	$\Delta_{FW}(\%)$	# of gates	$\phi_{IW}(\%)$	# of gates	$\phi_{FW}(\%)$
s38417	24370	6.726	0.764	11.36	0.906	13.47	460	1.89	280	1.15
s38584	21066	7.028	0.983	13.99	1.389	19.76	23	0.11	430	2.04
des_perf	154323	14.422	2.784	19.30	2.784	19.30	3059	1.98	3059	1.98
wb_conmax	75352	6.663	0.401	6.02	0.401	6.02	1441	1.91	1441	1.91
ethernet	157841	12.515	1.051	8.40	1.792	14.31	193	0.12	2,057	1.30
Average				11.81		14.57		1.2		1.68

Δ_{IW}: relaxed slack ratio of *Baseline* over *Original* Δ_{FW}: relaxed slack ratio of *ForTER* over *Original*
ϕ_{IW}: hardware cost ratio of *Baseline* over *Original* ϕ_{FW}: hardware cost ratio of *ForTER* over *Original*

Table 2: Experimental results on the relaxed slack and hardware cost: Baseline vs. *ForTER*.

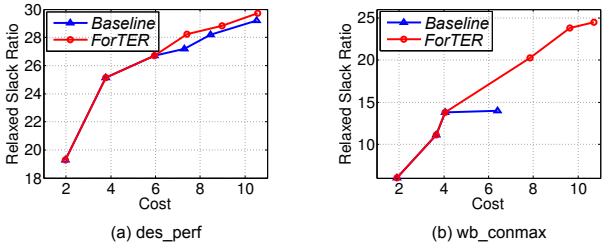

Figure 5: Relaxed slack ratio with respect to cost: des_perf and wb_conmax.

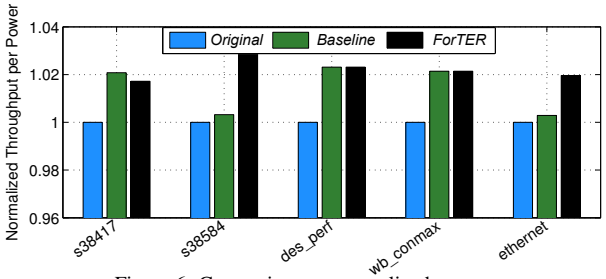

Figure 6: Comparison on normalized power.

tial elements and corrects them without necessarily borrowing timing slack. While *ForTER* itself is associated with moderate hardware cost, it can be effectively combined with other timing error-resilient design techniques and dramatically improve circuit timing slack, as demonstrated in our experimental results.

7. ACKNOWLEDGEMENT

This work was supported in part by the Hong Kong SAR Research Grants Council under General Research Fund No. CUHK418111 and No. CUHK418812, and in part by NSFC/RGC Joint Research Scheme No. N_CUHK444/12.

8. REFERENCES

[1] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, pp 10–16, 2005.

[2] D. Frank, R. Puri, and D. Toma. Design and CAD Challenges in 45nm CMOS and beyond. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, pp. 329–333, 2006.

[3] K. Bowman, et al. Circuit techniques for dynamic variation tolerance. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 4–7, 2009.

[4] D. Ernst, et.al. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proc. IEEE/ACM International Symposium on Microarchitecture*, pp. 7–18, 2003.

[5] K. Bowman, et al. Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *IEEE JSSC*, 44(1):49–63, 2009.

[6] B. Greskamp and J. Torrellas. Paceline: Improving single-thread performance in nanoscale CMPs through core overclocking. In *Proc. International Conference on Parallel Architecture and Compilation Techniques*, pp. 213–224, 2007.

[7] L. Wan and D. Chen. Dynatune: circuit-level optimization for timing speculation considering dynamic path behavior. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, pp. 172–179, 2007.

[8] B. Greskamp, et al. Blueshift: Designing processors for timing speculation from the ground up. In *IEEE International Symposium on High Performance Computer Architecture*, pp. 213–224, 2009.

[9] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori. Slack redistribution for graceful degradation under voltage overscaling. In *Proc. Asia and South Pacific Design Automation Conference (ASPDAC)*, pp. 825–831, 2010.

[10] Y. Liu, F. Yuan, and Q. Xu. Re-synthesis for cost-efficient circuit-level timing speculation. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 158–163, 2011

[11] Y. Liu, et al., On logic synthesis for timing speculation. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 591–596, 2012.

[12] R. Ye, and F. Yuan, and Q. Xu, Online clock skew tuning for timing speculation. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 442–447, 2011.

[13] M. R. Choudhury and K. Mohanram. Masking timing errors on speed-paths in logic circuits. In *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 87–92, 2009.

[14] F. Yuan, and Q. Xu. InTimeFix: a low-cost and scalable technique for in-situ timing error masking in logic circuits. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2013.

[15] M. Kurimoto, et al. Phase-adjustable error detection flip-flops with 2-stage hold driven optimization and slack based grouping scheme for dynamic voltage scaling. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 884–889, 2008.

[16] M. Wieckowski, et al. Timing yield enhancement through soft edge flip-flop based design. In *Proc. IEEE Custom Integrated Circuits Conference (CICC)*, pp. 543–546, 2008.

[17] M. R. Choudhury and K. Mohanram. TIMBER: Time borrowing and error relaying for online timing error resilience. In *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 1554–1559, 2010.

[18] B. Steinbach and C. Posthoff. Boolean differential equations. In *Proc. International Workshops on Post-Binary ULSI Systems*, pp. 46–53, 2011.

[19] S. Kosuch, and A. Lisser, Upper bounds for the 0-1 stochastic knapsack problem and a B&B algorithm. In *Journal of Annals of Operations Research*, pp. 77–93, 2010.

[20] M. R. Choudhury and K. Mohanram. Approximate logic circuits for low overhead, non-intrusive concurrent error detection. In *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 903–908, 2008.

[21] http://www.eecs.berkeley.edu/alanmi/abc/.