Efficient In Situ Error Detection Enabling Diverse Path Coverage

Chia-Hsiang Chen, Yaoyu Tao, and Zhengya Zhang
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor

***Abstract*—Technology scaling continues to improve density, but also reduces the critical charge to hold a logic state, causing devices to become more susceptible to accidental disruptions due to noise and soft errors. Increased process variation adds to the reliability challenge, resulting in over designs and extra timing margins at the cost of power consumption, silicon area and performance degradation. We present efficient in situ error detection techniques to exploit datapath characteristics for monitoring circuit errors: pre-edge checking in non-critical paths without hold time constraints; post-edge checking in critical paths without sacrificing performance; and cross-edge checking in moderate paths for the optimal trade-off. The techniques are all realized using the inherent redundancy within a conventional flip-flop design and do not require any logic or sample duplication as done by most existing methods. The detection-enabled flip-flop is implemented using only 31 transistors as a competitive and low-cost solution.**

I. Introduction

The scaling of device geometries continues to improve the performance of digital integrated circuits, but also leads to growing challenges in reliability and variability [1]. To deal with variations and aging problems, increasingly pessimistic margins have been applied in circuit designs as shown in Fig. 1(a), which results in a performance degradation and energy wasted. The reliability challenge is exacerbated by the shrinking of device dimensions and the ensuing reduction of critical charge, as devices are becoming more susceptible to the external noise sources and soft errors due to high-energy particle strikes [2].

To enhance the robustness of deep-submicron designs against occasional delay errors and soft errors, online circuit techniques have been proposed to detect error occurrences. These techniques can be classified to three groups based on how the checking is performed: post-edge checking that detects error in a window after the sampling edge [3]–[7], pre-edge checking that detects error in a window prior to the sampling edge [8], [9], as in Fig. 1(b), and multi-edge checking that detects errors by upsampling using multiple clock phases [10], [11].

Each existing technique has its own advantage and limitation. The post-edge technique incurs no performance penalty as checking occurs after the sampling edge, but each path under post-edge protection must be carefully tuned to avoid race conditions. The pre-edge technique is free of any hold time constraints, and it can use the delay slack in fast paths for error detection, however it prolongs the clock period if it is used in slow (critical) paths. Note that none of the above techniques alone is well suited to providing coverage of all types of datapaths. The implementation cost is also prohibitive, as each of the above techniques costs more than 40 transistors [3], [4], [8] or requires special clock controls [11], making it very expensive to equip every datapath for a full coverage.

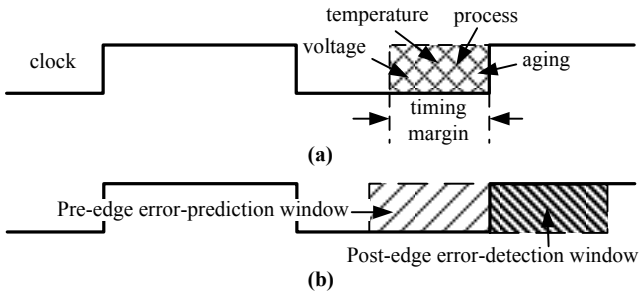

Fig. 1: (a) Timing margin allocated for static and dynamic variations, and (b) pre-edge and post-edge error checking window.

To overcome these challenges, we propose a diverse error detection technique to protect combinational logic paths while minimizing the performance penalty and implementation cost. In particular, we use a new cross-edge technique and its special pre-edge and post-edge versions for moderate, fast, and slow paths, respectively. Our method utilizes the inherent redundancy in a flip-flop design, thereby keeping the cost at only 31 transistors. Furthermore, it can also be tuned by duty cycling the clock signal, offering more flexibility for diverse levels of protection.

II. Prior Work

Pre-edge, post-edge and multi-edge are three classes of in situ timing error detection techniques. Their unique features are listed in Table I for comparison. The post-edge technique performs error detection after the sampling edge, thus eliminating the timing margin that would otherwise be necessary for the occasional variation-induced delay errors. The post-edge technique has been applied in high-performance, low-power designs. It allows the clock period to be reduced for a higher performance or the supply voltage to be reduced for a lower power consumption, as the post-edge checking acts as a safety net to detect the resulting delay errors. Post-edge checking is often implemented by a duplicate shadow latch that holds a post-edge sample that is compared with the primary sample for error detection [3], [5]. Transition detection was recently proposed as an alternative for a more area-efficient implementation. Successful designs including the second-generation RAZOR [4], [7] and DSTB [6] have been demonstrated. However, it requires each path under

978-1-4673-5762-3/13/$31.00 ©2013 IEEE

TABLE I: COMPARISON OF IN SITU ERROR DETECTION TECHNIQUES

Technique	Post-edge checking	Pre-edge checking	Multi-edge checking
Implementations	RAZOR [3], [4] DSTB [6], and TDTB [5]	BISER [8] and aging sensors [9]	PEDFF [10] and TIMBER [11]
Error detection mechanism	Transition detectors [4], [6] Duplicate latch/FFs [3], [5]	Transition detectors [9] Duplicate latch/FFs [8], [9]	Duplicate latch/FFs [10], [11]
Number of clock domains	1	1	2 [10] and 4 [11]
Race conditions	Yes	No	Yes
Checking window	Limited by fast path	Limited by max clock period	Flexible by multi-sampling

post-edge protection to be carefully adjusted to avoid race conditions.

The pre-edge technique has been presented in [8], [9] to monitor errors by detecting transitions in the checking window before the data is registered. The pre-edge technique is free of hold time constraints and is proven to be effective, e.g., in detecting slow changing events such as transistor aging [9], but the checking window occupies a portion of the clock period and degrades the performance in order to guarantee error-free operations for critical paths.

The multi-edge technique in [10], [11] offers more flexibility as the sampling edges for error detection are not necessarily aligned with the primary sampling edge of the datapath. Even though multi-edge improves the performance by adaptive time borrowing, the multi-edge implementation costs more than the transition detection introduced in post-edge and pre-edge checking and an additional clock domain also adds significant design complexity.

Each of the three techniques has its advantages and disadvantages in handling paths of different delays. The output of a critical path makes its final transition close to the sampling edge, while a non-critical path makes its final transition early. In the case of a fast path, the transition can happen shortly after the launching edge, resulting in a tight hold time constraint and a significant idle period prior to the end of the clock period. The ideal error checking window in each case should be placed right after the transition point as any delayed transition is an indication of possible errors. The desirable checking window for each type of path is annotated in Fig. 2(a). The figure also shows the difficulty of designing one checking window that fits all cases: a post-edge checking window causes race conditions from fast path, as in Fig. 2(b); a pre-edge checking window leads to performance penalty due to critical path as the clock period needs to be lengthened, as in Fig. 2(c).

Besides the location of the checking window, the length of the checking window is also important. A fixed length, as in most of the existing pre-edge and post-edge implementations, offers only a fixed protection against non-deterministic errors with variable durations. A soft error can last from a few pico seconds to hundreds of pico seconds [2]. The large variation calls for a tunable scheme to support different levels of protection as needed.

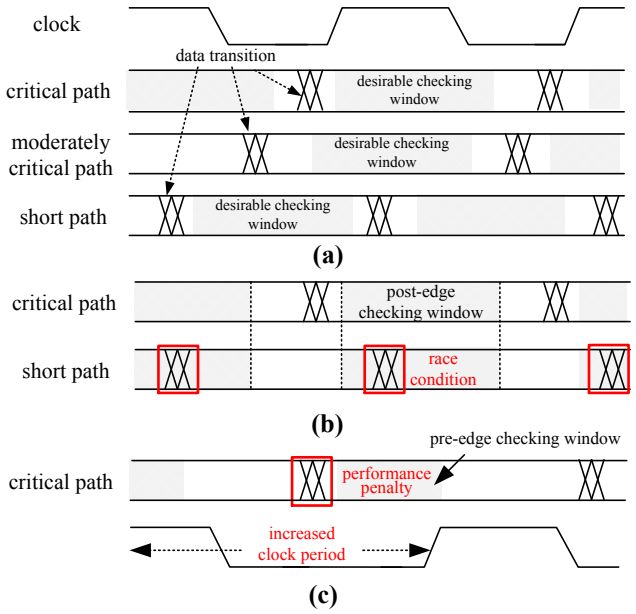

Fig. 2: (a) Checking window positions depending on path criticality, (b) fixed post-edge checking window causing race conditions, and (c) fixed pre-edge checking window leading to performance degradation.

III. Flexible Cross-Edge Checking

We exploit the diverse path delay distribution and design a new cross-edge checking to improve the performance while minimizing race conditions. We implement cross-edge checking based on a low-cost transition detection using a flip-flop. The cross-edge circuitry can be tuned depending on path criticality: pre-edge for fast paths, post-edge for critical paths, and cross-edge for the optimal trade-off between performance and race conditions in moderate paths. We assume that the path criticality is determined in design time for tuning the cross-edge circuitry.

A cross-edge checking window starts prior to the sampling edge and extends after the sampling edge, thus it crosses the sampling edge. A flexible and efficient circuit implementation is shown in Fig. 3(a) based on a conventional transmission gate flip-flop [12]. The flip-flop naturally keeps four copies of an input: D, ND, NM and M ("N" indicates logic inversion) that are phased apart by inverters and a transmission gate. We rely

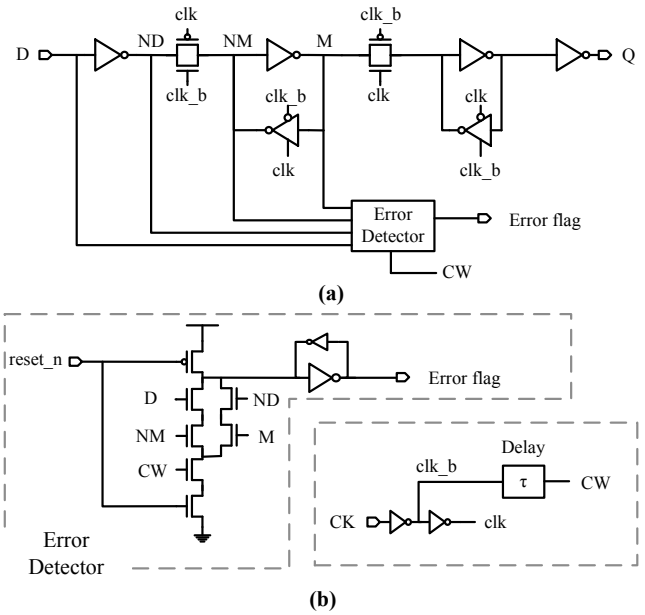

Fig. 3: (a) Cross-edge checking design based on a conventional transmission gate flip-flop [12], and (b) error detector design.

on the inherent redundancy instead of creating duplications, which is the key to achieving lower power and area.

A. Cross-Edge Circuitry

The cross-edge error detector circuit is shown in Fig. 3(b). The checking window CW is based on the inverted clock clk_b phased by an amount τ that is controlled by a local delay element. The delay element can be shared by a group of cross-edge flip-flops to amortize its cost. We can increase τ to push the checking window forward in time towards a post-edge checking for critical paths, or shrink the delay towards a pre-edge checking for fast paths. To reduce the design effort, a number of checking windows can be made based on different local delays. We will select one for a cluster of paths to suit their criticality. In addition to the location-tunability of the checking window, the duration of CW is controlled by duty cycling the clock signal. Thus, the duration is a tunable fraction of the clock cycle, and is dictated by one of the clock phase (i.e., low phase of the clock in Fig. 4).

The error detector circuit in Fig. 3(b) is made of a dynamic gate followed by a latch [4]. The dynamic gate first pre-charges. When CW is high, it checks the agreement of (D and NM) and (M and ND) as an indication of an erroneous transition. More specifically, the cross-edge checking window can be divided into two parts: a pre-edge part and a post-edge part. During the pre-edge part, clock is low and the master latch is transparent (see Fig. 3(a)). Early samples in NM and M are checked against late arriving samples D and ND to accomplish pre-edge checking. The dynamic gate is designed to respond much quicker than the propagation delay between D and NM and the delay between M and ND to guarantee the proper functionality. During the post-edge part of the checking window, the master latch is holding and the stored samples in NM and M are checked against post-edge

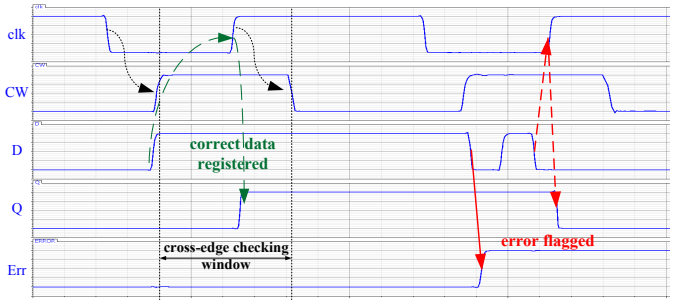

Fig. 4: Waveforms illustrating the operations of the cross-edge circuitry.

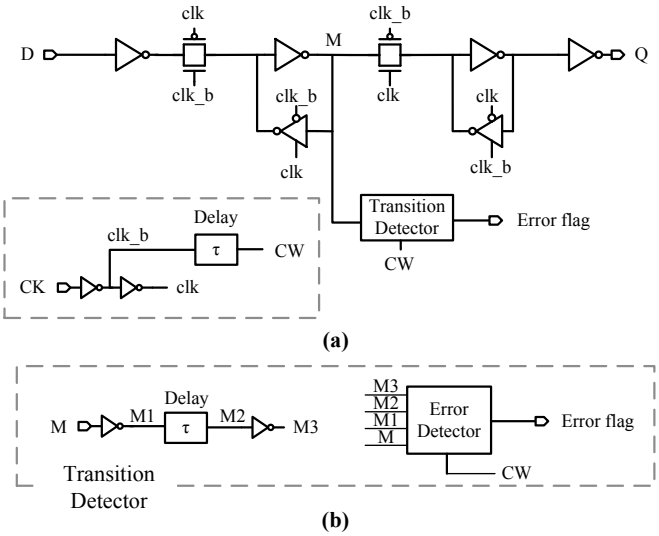

Fig. 5: (a) Pre-edge checking design, and (b) transition detector design.

samples D and ND to accomplish post-edge checking. If an erroneous transition is detected, the dynamic gate pulls down and an error flag is generated to trigger appropriate actions. An error could be corrected through rollback recovery [13], architectural recovery [14] or cross-layer recovery [15]. Once the error is resolved, the detector is reset by the controller.

We implemented the cross-edge circuitry in a 65nm CMOS technology and the SPICE simulation waveforms are provided in Fig. 4. The checking window is generated by delaying clk_b. In the first clock cycle, a clean input D is shown to be correctly registered. While in the second clock cycle, the input D makes an erroneous transition during the checking window and it is detected as an error. The proposed cross-edge checking circuitry adds a detector and extra wiring to the conventional flip-flop design. Our 65nm CMOS circuit simulation shows that the design consumes 13% more static power and 25% more total power than the conventional flip-flop at the highest switching activity of 1.

B. Pre-Edge Circuitry

Pre-edge checking can be made by shortening or eliminating the local delay τ in the cross-edge design, but an alternative design is possible for a guaranteed alignment between the sampling edge and the checking window for a zero hold

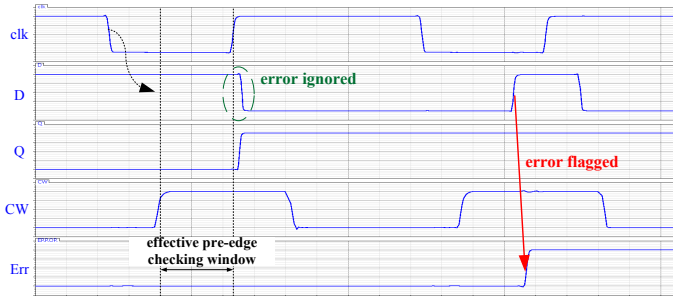

Fig. 6: Waveforms illustrating the operations of the pre-edge circuitry.

time. Fig. 5(a) shows the alternative design. The same error detector circuit is used but it taps only the master latch internal node M and its delayed copies. During the pre-edge part of checking window, the master latch is transparent and the error detector monitors (M and M2) and (M1 and M3) for transition detection, which implements pre-edge checking. When the master latch is holding during the post-edge part of the checking window, M can no longer change and post-edge checking is essentially turned off.

The SPICE simulation waveforms of this circuitry are illustrated in Fig. 6. An erroneous transition made by input D after the sampling edge is blocked by the master latch, thus no error is detected. In this way, an effective pre-edge checking window is created as shown in Fig. 6, and the alignment of this effective pre-edge window with the sampling edge is guaranteed. The design costs 15% more static power and 33% more total power than a conventional flip-flop at the highest switching activity of 1. The slight increase in power compared to the cross-edge design is due to the extra inverters to generate M1, M2 and M3 as in Fig. 5(b).

C. Comparisons

We compare the implementation complexity of the proposed cross-edge technique with state-of-the-art in situ error detection techniques in Table II. All error detection techniques use one delay element to generate the checking window. The transistor count of the proposed cross-edge flip-flop is only 31, much lower than other designs. In particular, the post-edge RAZOR technique uses detection clock generator and transition detector, which increase its transistor count to 47. The pre-edge BISER technique uses two conventional flip-flops and a C-element as a filter, resulting in a transistor count of 52. The transistor count of multi-edge TIMBER flip-flop is 36, but a specially designed clock control circuit is required to generate input signals to the flip-flop, costing more than 30 extra transistors.

IV. Conclusion

We propose a new efficient and in situ error detection technique to enhance system reliability against delay and soft errors. This technique exploits datapath criticality by appropriately adjusting the checking window for a higher performance while minimizing race conditions. The error detection technique is implemented in a flexible cross-edge

TABLE II: IMPLEMENTATION COMPLEXITY OF IN SITU ERROR DETECTION TECHNIQUES

Flip-flop type	Standard [12]	Post-edge [3], [4]	Pre-edge [8]	Multi-edge [11]	Cross-edge (this work)
Transistor count	22	47	52	36	31
Delay elements	0	1	1	1	1
Clock domains	1	1	1	≥ 2	1

checking circuitry that relies on the inherent redundancy without resorting to additional storage or duplication, thus the implementation cost is kept low. By duty cycling the clock signal, the checking window can be adjusted to provide different levels of protection. An alternative pre-edge circuitry is also proposed to guarantee the alignment of the checking window and the sampling edge for a zero hold time. The flexible and low-cost technique provides diverse path coverage for a fully protected error-resilient system.

Acknowledgment

This research was supported in part by Intel Corporation.

References

[1] S. Borkar, "Design perspectives on 22nm CMOS and beyond," in *Design Automation Conference, ACM/IEEE*, Jul. 2009, pp. 93–94.
[2] P. Dodd *et al.*, "Production and propagation of single-event transients in high-speed digital logic ICs," *IEEE Trans. Nuclear Science*, vol. 51, no. 6, pp. 3278–3284, Dec. 2004.
[3] D. Ernst *et al.*, "Razor: a low-power pipeline based on circuit-level timing speculation," in *IEEE/ACM Microarchitecture*, Dec. 2003, pp. 7–18.
[4] S. Das *et al.*, "RazorII: In situ error detection and correction for PVT and SER tolerance," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 32–48, Jan. 2009.
[5] K. Bowman *et al.*, "Energy-efficient and metastability-immune timing-error detection and recovery circuits for dynamic variation tolerance," in *Integrated Circuit Design and Technology and Tutorial*, June 2008.
[6] K. Bowman *et al.*, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 194–208, Jan. 2011.
[7] S. Valadimas *et al.*, "Cost and power efficient timing error tolerance in flip-flop based microprocessor core," in *IEEE European Test Symposium*, 2012, pp. 8–13.
[8] M. Zhang *et al.*, "Sequential element design with built-in soft error resilience," *IEEE Trans. VLSI Systems*, vol. 14, no. 12, pp. 1368–1378, Dec. 2006.
[9] M. Agarwal *et al.*, "Circuit failure prediction and its application to transistor aging," in *VLSI Test Symposium*, May 2007, pp. 277–286.
[10] 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 *Design Automation Conference, ACM/IEEE*, Jun. 2008, pp. 884–889.
[11] M. Choudhury *et al.*, "TIMBER: Time borrowing and error relaying for online timing error resilience," in *Design, Automation Test in Europe Conference Exhibition*, Mar. 2010, pp. 1554–1559.
[12] G. Gerosa *et al.*, "A 2.2 W, 80 MHz superscalar RISC microprocessor," *IEEE J. Solid-State Circuits*, vol. 29, no. 12, pp. 1440 –1454, Dec. 1994.
[13] T. Sakata *et al.*, "A cost-effective dependable microcontroller architecture with instruction-level rollback for soft error recovery," in *Dependable Systems and Networks, IEEE/IFIP*, June 2007, pp. 256–265.
[14] C.-H. Chen *et al.*, "A confidence-driven model for error-resilient computing," in *Design, Automation Test in Europe Conference Exhibition*, Mar. 2011.
[15] N. Carter *et al.*, "Design techniques for cross-layer resilience," in *Design, Automation Test in Europe Conference Exhibition*, Mar. 2010, pp. 1023–1028.