Power Minimization of Pipeline Architecture through 1-Cycle Error Correction and Voltage Scaling

Insup Shin[1], Jae-Joon Kim[2], and Youngsoo Shin[1]
[1]Department of Electrical Engineering, KAIST, Daejeon 305-701, Korea
[2]Department of Creative IT Engineering, POSTECH, Pohang 790-784, Korea

***Abstract*— We present a new 1-cycle timing error correction method, which enables aggressive voltage scaling in a pipelined architecture. The proposed method differs from the state-of-the-art in that the pipeline stage where the timing error occurs can continue to receive input data without halting to avoid data collision. The feature allows the pipeline to avoid recurring clock gating when timing errors happen at multiple stages or timing errors continue to occur at a certain stage. Compared to a state-of-the-art method, the proposed method shows 2-6% energy reduction for a 5-stage pipeline and 7-11% reduction for a 10-stage pipeline. In addition, the proposed logic to propagate clock gating signal is much simpler than that of the previous method [1] by eliminating reverse propagation path of clock gating signal.**

I. Introduction

Adding timing guardband has been an effective solution to address delay deviations due to process variations. Designers need to add suitable amount of timing guardband to critical path delay only to make the circuit operate correctly even in the worst case scenario. However, as the variation has become a major concern in nanoscale circuits, the amount of timing guardband increases significantly. This has led to growing interest in the design methodology to eliminate or reduce the timing guardband.

Timing speculation [2]–[6] is one of circuit techniques to eliminate the timing guardband. The key idea is to detect and correct timing errors; Razor [2] is a representative example. In this method, the output of a logic stage is sampled by a main flip-flop with nominal clock as well as by a shadow latch with the delayed clock. A timing error is checked by comparing the data from flip-flop and latch. The error is corrected by restoring the data from shadow latch into main flip-flop.

Another benefit of timing speculation is significant power saving when aggressive voltage scaling is employed. Since errors can be detected and corrected, supply voltage can be lowered beyond the minimum voltage that leads to zero error. The effectiveness of this voltage scaling depends on timing error rate below the minimum voltage. Various techniques [7]–[11] have been proposed to reduce minimum allowable voltage level given target throughput by reshaping distribution of path delays. Methods for adding slack to the frequently violated paths have been proposed in [7] (Blueshift). A cell sizing method was proposed in [8]; cell upsizing on critical and frequently exercised paths to extend voltage scaling and down sizing in noncritical and infrequently exercised paths to reduce leakage power. In [9], a dual-Vt assignment algorithm (Dynatune) was proposed to increase throughput based on the behavior curve of a circuit. The technique proposed in [10] changes the circuit structure by re-ordering fan-ins of some gates during logic optimization. The technique proposed in [11] uses new cost function which takes timing error probability into account during logic synthesis.

However, these techniques have limited impact due to the existence of critical operating point [12]. Since timing slacks are spent to optimize such objectives as area and power, many paths have almost same path delay as the critical one, which is called slack wall. Thus, there exists a critical voltage V_c for a fixed ambient temperature such that any voltage below V_c causes massive errors [12]. The techniques in [7]–[11] cannot handle such a large number of errors. Hence, critical operating point is the major challenge for the timing speculation based voltage overscaling.

Massive errors require very large timing penalty for error correction. Therefore, a challenge of timing speculation based voltage overscaling lies in reducing overall timing penalty for error corrections. There have been several error correction methods [1], [2], [5], [6], [13], [14]. Among them, instruction replay [5] has the smallest design overhead. When the instruction for which an error has been detected reaches the last stage, the processor flushes all stages and issues the failing instruction again. During re-execution of the failing instruction, clock frequency is reduced by half. Thus it has large timing penalty per error correction. Hence, the instruction replay has limitation for timing speculation based voltage overscaling technique.

Micro-rollback [13], [14] saves previous data of each pipeline stage to backup storage in each cycle. When the error signal reaches the last stage, the backup storage injects the last known correct data to each pipeline stage. However, since this method increases flip-flop energy consumption by 15% [13], it is also not suitable for voltage overscaling design.

In counterflow pipelining method [2], error is corrected by restoring shadow latch data into main flip-flop in the next cycle of the error detection. Then, to maintain proper operation, the stage which detected the error sends a bubble to the last stage and a flush signal to the first stage. When the flush signal

978-1-4799-2816-3/14/$31.00 ©2014 IEEE

reaches the first stage, the next-cycle instruction to the failed instruction is issued again. If an error is detected at the k-th stage, the completion time of the next-cycle instruction is delayed by $2k$. It has smaller timing penalty than that of instruction replay but the timing penalty is still too large to use for voltage overscaling scenario.

Recently, two 1-cycle error correction methods have been proposed [1], [6]. The first method [6] targets two-phase transparent latch based designs and the second method [1] is toward flip-flop or pulsed-latch based design. However, they also have limitation in handling massive errors. Both methods can correct multiple errors by one cycle penalty but only limited in scope.

In this paper, we propose new error correction method which has 1-cycle penalty with simpler control logic. More importanlty, the proposed method can handle massive errors (and even permanent error) with only a small number of timing penalties.

The remainder of this paper is organized as follows. Our proposed method is presented in Section II. Its extension to handle complex pipeline structure is addressed in Section III. Experimental results are provided in Section IV and Section V concludes the paper.

II. Proposed Architecture

The main difficulty in error correction is loss of new input data while shadow latch data is restored to main flip-flop. Counterflow pipelining solves this problem by flushing all the subsequent instructions from the one that caused error and re-executing them after the error has been corrected. Instruction replay [5] avoids this problem by re-executing the failed instruction. This method mitigates the circuit complexity by using simple recovery signal so that it requires very small overhead in control logic. However many clock cycles are needed to flush instructions and re-executes them. A recently proposed error correction method [1] overcomes this problem by stalling each input stage in a wave-like fashion. However this method requires relatively complex control logic.

We propose a simpler method to overcome the problem. Shadow latch receives wider clock pulse than that main latch receives. Thus, after sending its data to main latch for error correction, shadow latch can still capture new input data. Our key idea is to alter the clock toward shadow latch in such a way that shadow latch opens after main latch closes. Then, shadow latch can send previous correct data to main latch for error correction and also capture new input data in the same cycle. Thus, the data conflict problem can be avoided.

Fig. 1(a) illustrates conventional Razor latch. We denote the clock for main and shadow latch as clk_m and clk_s, respectively. Pulse width of clk_s, denoted by W_s, is larger than that of clk_m, denoted by W_m. When an error occurs, restore signal becomes 1 so that shadow latch data SQ is restored into main latch. Fig. 1(b) illustrates modified Razor latch for the proposed method. It also uses shadow latch to capture the data when the error occurs, and send the stored data to main latch in the next cycle when the restore signal

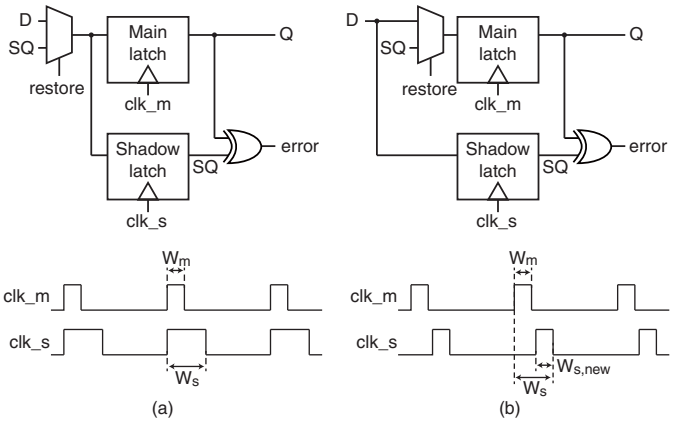

Fig. 1. Conceptual schematics of (a) conventional Razor latch and (b) modified Razor latch for our method.

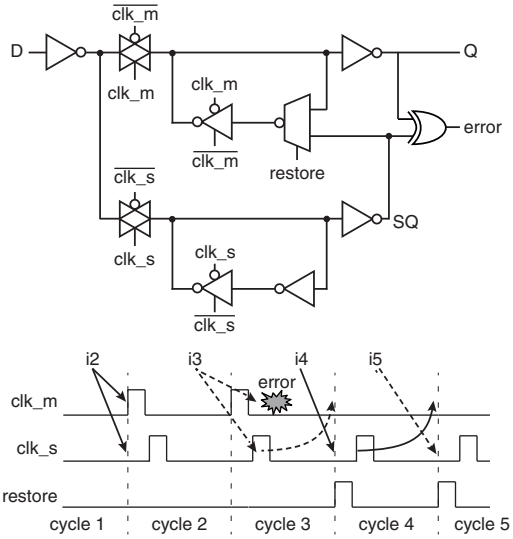

Fig. 2. Circuit level schematic of modified Razor latch.

becomes 1. The main difference of the modified Razor latch is to use non-overlapping clock signals to drive main and shadow latches. As shown 1(b), clk_s has been altered from conventional Razor latch such that shadow latch will start capturing input data after main latch stops capturing the data. Note that the window of timing speculation does not change from that of conventional Razor latch although clk_s has more narrow width. Also the input data is directly fed into the shadow latch so that it can capture the data even when main latch is gated; we will explain in the later section how this can help reduce control logic during error correction.

Fig. 2 shows circuit level schematic of modified Razor latch. The multiplexer at the input of Razor latch is moved to the feedback path of the main latch to reduce delay and power overhead. Assume that instruction $i3$ fails at cycle 3. After the error detection, clk_m is gated at each following cycle. In cycle 4, restore signal becomes 1 for the duration of W_m so that correct $i3$ is transmitted from shadow latch to main latch. Instruction $i4$ is captured by shadow latch after the restoration so that no data is lost.

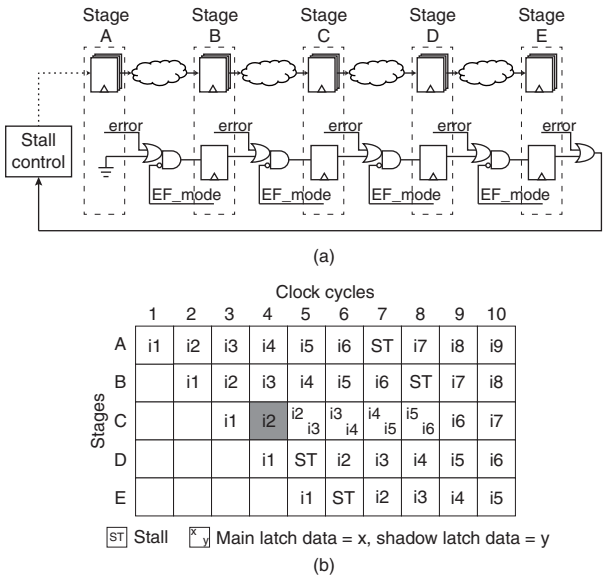

Fig. 3. (a) Modified pipeline organization for propagation of CG signal and (b) error correction example for our method when an error occurs at stage C in cycle 4.

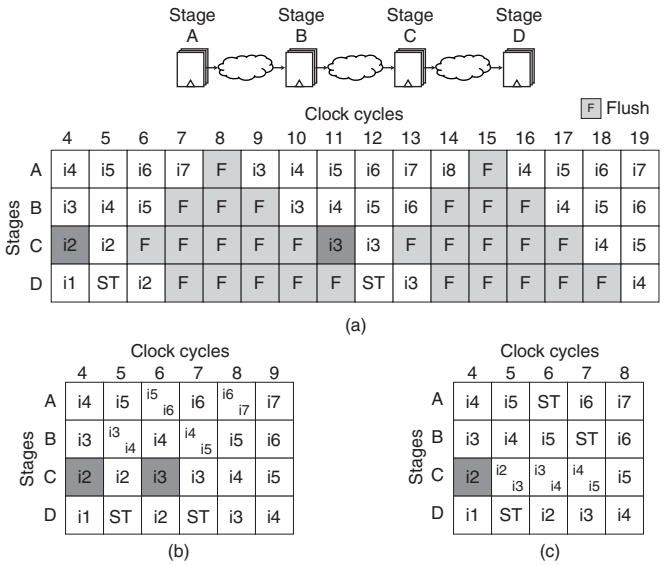

Fig. 4. Error correction examples for (a) counterflow pipelining, (b) previous 1-cycle error correction [1], and (c) our method when instructions $i2$ and $i3$ will be failed at stage C.

From cycle 4 on, input data is stored into shadow latch and then is transmitted to main latch in the next cycle. This state will be retained until a stall is propagated to this stage. In order to send a stall to the stage where an error occurred, we add stall control logic as shown in Fig. 3.

Similar to the previous error correction methods, our method also uses clock gating control signal, called CG signal, to prevent incorrect data propagation through the pipeline due to the timing error at a stage. When an error occurs, a CG signal is propagated to output stages from the stage where the error occurred. The CG signal gates both clk_m and clk_s for one cycle. CG signal is propagated in a wave-like fashion such that it is transmitted to the next stage at each cycle. When the CG signal reaches the last stage, the stage sends stall signal to the stall control so that a stall is issued to the pipeline.

A. Error-Free Mode

Once an error occurs at a particular stage, input data is received by shadow latch alone and the data is transmitted back to main latch in the next cycle as mentioned before. No late timing error occurs in this operation mode, which we call error-free mode. Fig. 3 shows a conceptual pipeline organization for our method and error correction example. Assume that instruction $i2$ fails at stage C in cycle 4 as shown in Fig. 3(b). Stage C enters error-free mode and CG signal is propagated to last stage, stage E. Since stage E receives CG signal in cycle 5, it sends the CG signal to stall control in the next cycle. Thus, a stall is issued to the pipeline in cycle 7. When the stall is propagated to stage C, stage C gets out the error-free mode as shown in Fig. 3(b). Timing penalty per error correction is only one cycle.

One advantage of error-free mode is to offer the opportunity that multiple errors at the same stage are corrected with only one cycle. Consider a pipeline architecture that has four stages as shown in Fig. 4 and assume that instructions $i2$ and $i3$ will fail at stage C. In counterflow pipelining, the completion time of the next-cycle instruction is delayed by $2k$ when an error occurs at k-th stage. Therefore, the completion time of instruction $i4$ is delayed by $2(3)+2(3) = 12$ cycles as shown in Fig. 4(a). 1-cycle error correction method [1] requires two cycles to correct errors as shown in Fig. 4(b).

Compared to the previous error correction methods, our method can handle multiple errors at the same stage with less cycle penalty. Once an error occurs, the stage enters into error-free mode until a stall signal is propagated back to the stage. Thus, our method requires only one cycle delay to handle the case shown in Fig. 4(c). Furthermore, our method can handle a permanent error in the piepline more effectively. If a pipeline stage has a permanent timing error, that stage will simply stay in error-free mode; the pipeline stage will re-enter into error-free mode in the next cycle when the stall signal is propagated back to the stage and this will repeat as long as an error persists.

B. Multiple Timing Errors

CG signal is propagated from the stage where an error occurred to its output stages to prevent the propagation of incorrect data and then returns to the stage to bring it out of error-free mode. However, CG signal does not necessarily need to propagate back to the stage where is was issued since the propagation of CG signal will stop when it arrives at any stage already in error-free mode. Therefore, multiple errors occurring at different stages can be corrected simultaneously. In such cases, the number of stalled cycles is smaller than the number of errors.

In Fig. 5, timing errors occur at stage D in cycle 5 and at stage B in cycle 6. Propagation of CG signal which started from stage B stops at stage D in cycle 8 because stage D is already in error-free mode. Propagation of CG signal which

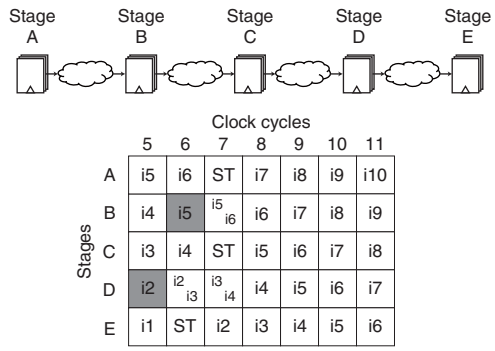

Fig. 5. Propagation of CG is stopped when it is propagated to the stage which has already entered into error-free mode.

started from stage D also stops at stage B in cycle 8 for the same reason. In this case, only one cycle is needed to correct both errors.

Previous 1-cycle error correction methods [1], [6] can also correct multiple errors by one cycle penalty but it is possible only when clock gating signals are met. Thus, our method has more opportunity to correct multiple error with one cycle penalty.

III. Extension to General Pipeline Architecture

A practical pipeline may have multiple fan-out and multiple fan-in stages or a loop. In order to handle more general pipeline architecture, additional control signal is needed. The main problem with a multiple fan-in stage lies in the fact that the pipeline fails after the multiple fan-in stage receives CG signals from part of its input stages alone. Assume that stage D has multiple fan-in as shown in Fig. 6. When an error occurs at stage A in cycle 4, CG signals are propagated to stage B and stage D in cycle 5. Stage D is therefore stalled in cycle 5 but stage C sends instruction $i2$ to stage D in the same cycle. Since instruction $i2$ cannot be captured in stage D, the pipeline fails after cycle 5.

The basic rule to maintain proper operation is to forbid all of its input stages from sending their data to the stage when a stage is stalled. Thus, the problem can be resolved by generating virtual errors at all the stages that did not send CG signals to a multiple fan-in stage. In Fig. 6(b), we generate a virtual error at stage C in cycle 4.

A new control signal VE is employed to realize the concept. When a stage receives a CG signal from any of its input stage, the stage sends VE signals back to all of its input stages to check whether each stage sends CG signal. If a stage which receives VE signal is already clock-gated or has generated an error in early part of the same cycle, the VE signal does not affect the stage. Otherwise, the stage gets into error-free mode. Fig. 6(c) shows a timing diagram for CG signal and VE signal. Since stage A sends CG signals to stage B and stage D in cycle 4, stage B sends VE signal to stage A, and stage D sends VE signals to its input stages (stage A and C) in the same cycle. The VE signal does not affect stage A because stage A already experiences a timing error in the earlier part of the cycle 4. Stage C gets into error-free mode immediately.

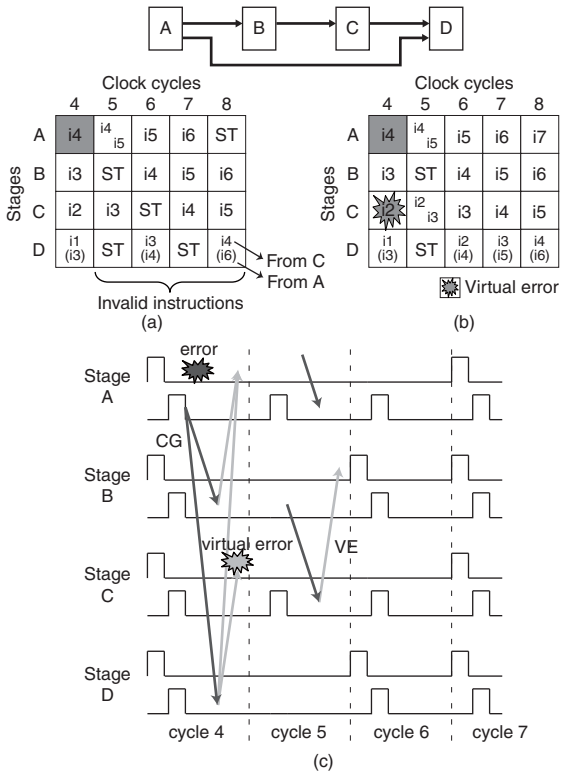

Fig. 6. A pipeline circuit which has multiple fan-out stage and multiple fan-in stage. (a) When VE control signal is not used and (b) when VE control signal is used and (c) its timing diagram.

In cycle 5, stage B sends CG signal to stage C and then stage C sends VE signal back to stage B. Due to the CG signal being propagated to stage C, the stage C gets out of error-free mode so that clk_m is not gated after cycle 7. Since stage B sends CG signal to stage C, the VE signal received by stage B is nullified.

Our method also correctly works in the loop condition by using VE control signal. The main challenge in such a condition is to prevent infinite looping. The infinite looping of CG propagation does not occur regardless of the location where the error happens. Note that the error can occur before the loop, in the loop, or after the loop. Consider a pipeline architecture that has a loop and assume that three errors occur at stage A, stage C, and stage E, respectively, as shown in Fig. 7. When an error occurs before the loop (stage A in cycle 5), the CG signal is inserted into the loop and generates a virtual error at stage D due to VE signal. Due to this virtual error, CG signals are propagated from stage D to its output stages (stage B and E) in cycle 6. The CG signal which was propagated to stage E is sent to stage A through the stall control so that stage A gets out of error-free mode in cycle 7. The CG signal which was sent to stage B is propagated to stage D and stage D also gets out of error-free mode in cycle 8. Timing penalty for the error correction is one cycle even in this case.

When an error occurs within the loop (stage C in cycle 9), CG signal is propagated from stage C to stage D in cycle 10. In cycle 11, stage D sends CG signal to its output stages (stage

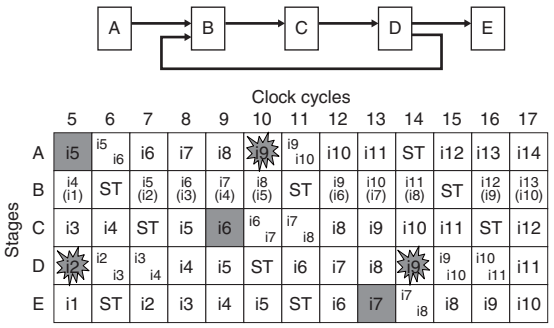

Fig. 7. Error correction example for loop case when errors occur before the loop, in the loop, and after the loop.

B and E). When stage B receives CG signal, VE signals are sent to stage A and stage D. Since stage D is already gated in cycle 10, the VE signal which was sent to stage D is nullified. Stage A is not gated nor detects an error in cycle 10. Thus, a virtual error is generated at stage A as shown in Fig. 7. Due to this virtual error, CG signal is propagated to stage C and thus stage C gets out of error-free mode in cycle 12. Stage A gets out of error-free mode by the CG signal which was propagated from stage C.

When an error occurs after the loop (stage E in cycle 13), CG signal is propagated to stage A through the stall control in cycle 14. When stage B receives CG signal, it sends VE signals to stage A and stage D. Due to this VE signal, a virtual error is generated at stage D in cycle 14. Stage E gets out of error-free mode in cycle 15 due to CG signal generated from stage D and stage D also gets out of error free mode in cycle 17 due to CG signal generated from stage E.

The control logic for proposed error correction method is illustrated in Fig. 8. All sequential elements in the figure are transparent latches. Once an error occurs, EF node becomes 1; CG_out node is also asserted to 1. EF node remains at 1 until CG signal arrives (CG_in node becomes 1). When CG signal arrives from any input stage, it is transmitted to all the output stages through CG_out node and VE signal is sent to all the input stages through VE_out node. However, the propagation of CG signal works only during normal operation mode (when both EF and EF_pre nodes are 0). When a stage receives VE signal, it generates a virtual error and then EF_node becomes 1 if CG_pre node is 0.

IV. Experimental Results

To assess the effectiveness of our error correction method, we compiled nine pipelined circuits with 45-nm Open Cell Library [15]. Three different number of pipeline stages (5, 8, and 10) was tried; three different circuits (c1908, c3540, and c6288 from ISCAS benchmark) were assumed for each pipeline stage. The combinational circuits are synthesized using Design Compiler [16] with timing constraint set to 90% of critical path delay of the circuit synthesized without any contstraint. Clock periods of resulting c1908, c3540, and c6288 were 1.2 ns, 1.6 ns, and 2.4 ns, respectively. All pipelined circuits are pulsed-latch based Razor circuit. The pulse width of main latch is 105 ps, which is the minimum

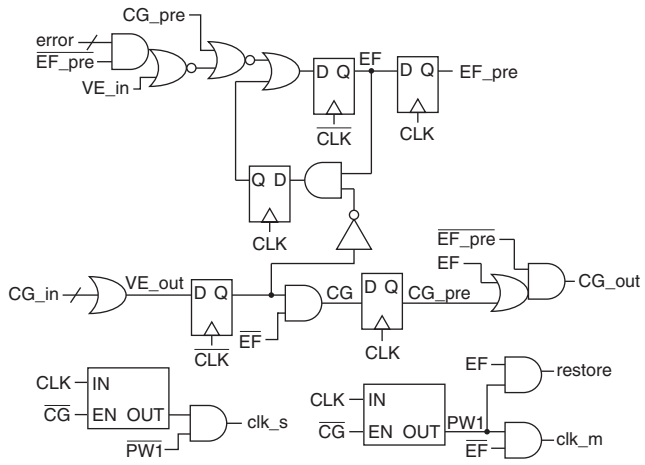

Fig. 8. Schematic of control logic.

width at 0.75 V under nominal process corner, and that of shadow latch is 400 ps. In all the circuits, extra delay buffers are inserted to fix hold violations.

We applied 100 random vectors to each pipelined circuit and determined its throughput via fast SPICE simulation [17]. The simulation was performed at the initial voltage (1.1 V) and the voltage was gradually reduced by an increment of 0.02 V until the throughput met a target; total energy dissipation was then measured. We performed this experiment for two target throughputs, 0.9 and 0.7, as shown in Table I and Table II. We used the same experimental setting to collect data for counterflow pipelining [2] and 1-cycle timing error correction (1-CTEC) [1] for comparison.

Since counterflow pipelining has the largest timing penalty per error correction ($2k$ cycles, where k is the order of the stage that an error was detected), its target voltage has the highest value as expected. Moreover, it requires the largest design overhead to implement flushing logic; this adds extra overhead to the total energy dissipation. Note that the normalized total energy dissipation of counterflow pipelining increases with more pipeline stages. This is because of two shortcomings compared to both 1-CTEC and our method: 1) its timing penalty per error correction depends on the number of pipeline stages and 2) it cannot correct multiple errors simultaneously.

Both our method and 1-CTEC have 1 cycle penalty per error correction. However, thanks to error-free mode, our method can handle multiple errors with less cycle penalties, which is why the target voltage level of our method is lower than that of 1-CTEC. Compared to 1-CTEC, our method reduces the total energy dissipation by 2% and 6% when target throughputs are 0.9 and 0.7, respectively, for 5-stage pipeline. Note that the benefit is more as the number of pipeline stage increases; for 10-stage pipeline, our method reduces the total energy dissipation by 7% and 11% when target throughputs are 0.9 and 0.7, respectively. This can be understood from the fact that the number of cycles that each stage runs in error-free mode increases with the number of pipeline stages; this increases the opportunity for correcting multiple errors occurred at different

TABLE I
EXPERIMENTAL RESULTS WHEN TARGET THROUGHPUT IS 0.9

#Stages	Base circuit	Counterflow		1-CTEC		Ours	
		Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]
5	c1908	0.92	783	0.84	707	0.84	716
	c3540	0.94	2107	0.88	1816	0.86	1751
	c6288	0.98	5108	0.90	4307	0.90	4221
	Avg.		1.16		1.00		0.98
8	c1908	0.92	1279	0.86	1156	0.84	1108
	c3540	0.94	3598	0.88	3056	0.86	2868
	c6288	0.98	8267	0.90	6774	0.90	6803
	Avg.		1.17		1.00		0.97
10	c1908	0.94	1591	0.88	1362	0.86	1316
	c3540	0.96	4931	0.90	3991	0.88	3692
	c6288	0.98	10449	0.90	8489	0.88	7596
	Avg.		1.21		1.00		0.93

TABLE II
EXPERIMENTAL RESULTS WHEN TARGET THROUGHPUT IS 0.7

#Stages	Base circuit	Counterflow		1-CTEC		Ours	
		Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]
5	c1908	0.90	751	0.78	614	0.76	576
	c3540	0.92	1912	0.82	1594	0.80	1515
	c6288	0.98	5108	0.86	3994	0.84	3706
	Avg.		1.23		1.00		0.94
8	c1908	0.92	1279	0.80	967	0.76	914
	c3540	0.92	3435	0.82	2325	0.80	2045
	c6288	0.98	8267	0.86	6227	0.84	5696
	Avg.		1.38		1.00		0.91
10	c1908	0.92	1540	0.84	1254	0.80	1153
	c3540	0.96	4931	0.84	3411	0.80	2906
	c6288	0.98	10449	0.86	7457	0.84	6625
	Avg.		1.36		1.00		0.89

stages with less cycle penalties.

V. CONCLUSION

Fast error correction method is vital to timing error detection/correction mechanism especially when targeting for low voltage operation. Our main contributions in this paper include a simplified logic to propagate clock gating signal and the reduction of timing penalty when large number of errors occur at critical operation point voltage. Compared to previous fast (1-cycle) error correction methods, the number of stalled clock cycles can be reduced significantly by allowing input data to continuously flow into the pipeline stage where timing error has occurred.

ACKNOWLEDGMENT

The work of Jae-Joon Kim was supported by the MSIP (Ministry of Science, ICT, and Planning), Korea under the "IT Consilience Creative Program"(NIPA-2013-H0203-13-1001) supervised by NIPA (National IT Industry Promotion Agency).

REFERENCES

[1] I. Shin *et al.*, "A pipeline architecture with 1-cycle timing error correction for low voltage operations," in *Proc. Int. Symp. on Low Power Electronics and Design*, Sept. 2013, pp. 199–204.
[2] D. Ernst *et al.*, "Razor: a low-power pipeline based on circuit-level timing speculation," in *Proc. Int. Symp. on Microarchitecture*, Dec. 2003, pp. 7–18.
[3] S. Das *et al.*, "Razor II: in situ error detection and correction for PVT and SER tolerance," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 32–48, Jan. 2009.
[4] K. A. Bowman *et al.*, "Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 49–63, Jan. 2009.
[5] K. A. Bowman *et al.*, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 194–208, Jan. 2011.
[6] M. Fojtik *et al.*, "Bubble Razor: an architecture-independent approach to timing-error detection and correction," in *Proc. Int. Solid-State Circuits Conf.*, Feb. 2012, pp. 488–490.
[7] B. Greskamp *et al.*, "Blueshift: designing processors for timing speculation from the ground up," in *Proc. Int. Symp. High Performance Computer Architecture*, Feb. 2009, pp. 213–224.
[8] A. B. Kahng *et al.*, "Slack redistribution for graceful degradation under voltage overscaling," in *Proc. Asia South Pacific Design Automation Conf.*, Jan. 2010, pp. 825–831.
[9] L. Wan and D. Chen, "Dynatune: circuit level optimization for timing speculation considering dynamic path behavior," in *Proc. Int. Conf. on Computer Aided Design*, Nov. 2009, pp. 172–179.
[10] Y. Liu *et al.*, "On logic synthesis for timing speculation," in *Proc. Int. Conf. on Computer Aided Design*, Nov. 2012, pp. 591–596.
[11] J. Cong and K. Minkovich, "Logic synthesis for better than worst-case designs," in *Proc. Int. Symp. on VLSI Design, Automation & Test*, Apr. 2009, pp. 166–169.
[12] J. Patel, "CMOS process variations: a critical operation point hypothesis," Online Presentation, 2008.
[13] D. Ernst *et al.*, "Razor: circuit-level correction of timing errors for low-power operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, Nov. 2004.
[14] J. Crop *et al.*, "Error detection and recovery techniques for variation-aware CMOS computing: a comprehensive review," *Journal of Low Power Electronic and Applications*, vol. 1, no. 3, pp. 334–356, 2011.
[15] "Nangate 45nm open cell library," Available: http://www.nangate.com/.
[16] *Design Compiler User Guide*, Synopsys, Sept. 2011.
[17] *NanoSim User Guide*, Synopsys, Sept. 2011.