International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

Adaptive Error Correction Techniques in Pipelines For Low Voltage Design

Sujaya Grace. CH. M.Tech., Assistant Professor, ECE Department, *sujaya.grace@gmail.com*
Swami Vivekananda Institute of Technology, Secunderabad, India
Mrs. Savitha T. M.Tech., Assistant Professor, ECE Department, *savitha.daram@gmail.com*
Swami Vivekananda Institute of Technology, Secunderabad, India.

***Abstract*— One of the most aggressive uses of dynamic voltage scaling is timing speculation, which in turn requires fast correction of timing errors. The fastest existing error correction technique imposes a one-cycle time penalty only, but it is restricted to two-phase transparent latch-based pipelines. We perform one-cycle error correction by gating only the main latch in each stage of the pipeline that precedes a failed stage. This new method is applicable to widely used clocking elements, such as flip-flops and pulsed latches. Because it prevents inputs arriving at a stage, which is stalled, it can also be used in pipelines with multiple fan-in, fan-out, and looping. Simulations show an energy saving of 8%–12% with a target throughput of 0.9 instructions per cycle, and 15%–18% when the target is 0.8.**

***Index Terms*— Error correction, low-voltage operation, timing speculation.**

I. Introduction

AS PORTABLE computing has become ubiquitous, energy efficiency has emerged as a critical design requirement in contemporary VLSI circuits. Numerous energy-efficient design techniques have been proposed at various levels of abstraction. Among them, voltage scaling has proved to be one of the most effective ways of reducing energy consumption. This is because switching energy falls quadratically with supply voltage, and leakage energy even more quickly [2].

The drawback of reducing supply voltage is that it increases circuit delays, and this limits the scope of voltage scaling. Several techniques, such as pipelining and parallel processing, have been proposed to allow large reductions in voltage [3]. Pipelining involves the insertion of sequential elements into the data path to reduce the critical path delay. Its disadvantages are an increase in circuit latency, and the area required for the additional sequential elements. In parallel processing, a task is split into N subtasks, which are then executed concurrently on N processors. This allows each processor to operate N times more slowly, thus at a correspondingly reduced voltage. Unlike pipelining, parallel processing does not increase circuit latency, but it has a much larger area overhead.

Another way to increase the scope for voltage scaling is the reduction of timing margins, which is a technique that has attracted a lot of attention in recent years. The standard method of dealing with variability in delay, which is an inevitable consequence of circuit manufacturing tolerances, is to add a timing margin to the nominal cycle times during the design process. As technologies have been repeatedly scaled down, however, the magnitude of this variability has increased significantly, so that timing margins have become very significant. This makes timing margin reduction a potentially profitable approach for increasing speed or reducing supply voltage. Reducing timing margins has the advantage that it involves no increase in latency and incurs a much lower area overhead than pipelining or parallel processing.

A major challenge in timing margin reduction methodologies is the increased probability of timing errors due to variations. In general, the variations can be categorized into two types: 1) spatial and 2) temporal variations. Transistors on a die experience two types of spatial variations: 1) global variation and 2) local variation. Global variation mostly affects the electrical characteristics of the devices on a die in the same way. On the other hand, local variation affects the transistor characteristics in more unpredictable way due to randomness [4]. Temporal variation also has two types. The amount of static variations is decided during the fabrication period and it does not change with time. On the other hand, temporal variation occurs due to environmental changes, such as temperature, supply voltage noise, and aging cause the transistors to experience variability depending on time. To accommodate the potential increase in circuit delay caused by the variations, more timing margin is given in traditional design approaches.

There are two approaches to overcome the limitation in the traditional design practices. One is to predict the occurrence of errors to avoid timing violation (error prediction approach), and the other is to detect actual errors and correct them (error detection approach). The error prediction approach uses sensors or canary circuits to monitor the magnitude of timing variations. On-chip sensors [5]–[10] measure the supply voltage or the temperature of the chip, and canary circuits [11]–[13] measure the delay in critical path replicas of the chip. On-chip sensor and canary circuit must communicate their results to the adaptive circuit so that it can adjust the

In this paper, we propose a new correction method that has only one cycle penality. We can elaborate how the importance of reducing the timing penality for error correction is for low voltage design by looking at the relationships shown in fig.1 between supply voltage and pipeline throughput[instructions per cycle(IPC)], for three different error correction methods: 1. Instruction Replay, 2. Counter Flow Pipelining, and 3. Our Proposed Method.

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

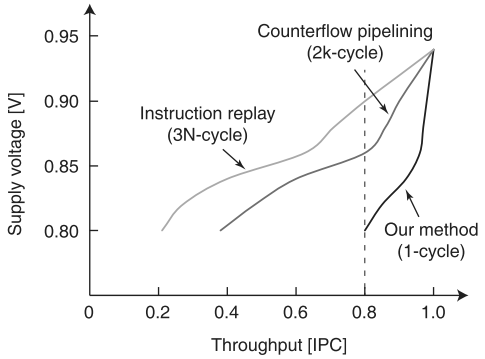


Fig. 1. Relationship between supply voltage and throughput for our method, counterflow pipelining, and instruction replay, in c6288-based five-stage pipeline circuit.

supply voltage or clock frequency, or both, before errors actually occur. However, communication and adjusting operating environment require some time, and, therefore, they cannot respond to fast-changing dynamic variations. In addition, they cannot detect local variations because a limited number of sensors or canary circuits are placed in a chip. Thus, error prediction approach still requires a timing margin for local or dynamic variations.

The error correction approach uses error detection sequential (EDS) circuits [14]–[20] to detect the errors that actually occur and correct them using on-chip correction logic. Razor [15] is a well-known EDS circuit, in which data are captured by a shadow latch with a delayed clock signal, as well as by a main flip-flop with a nominal clock. If the shadow latch data are different from those captured by the main flip-flop, an error is flagged, and then corrected by error correction logic. Since the error correction approach detects changes, which occur on the actual critical path, the timing margin, which would otherwise be required to allow for dynamic and local variations, can be eliminated, as well as the margin for global variations. This makes error correction more effective than error prediction for scaling the supply voltage.

A. Motivation

Error correction reduces throughput because it requires extra cycles. Existing error correction methods have large timing penalties, as given in more detail in Section II. For example, instruction replay and counterflow pipelining, which are the best-known error corrections, have timing penalties of $3N$ and $2k$ cycles, respectively. N is the number of pipeline stages and k is the order of the stage that detects an error.

Hence, there is a pressing need to reduce the timing overhead as much as possible. In this paper, we propose a new error correction method that has only one-cycle penalty. We can illustrate how the importance of reducing the timing penalty for error correction is for low-voltage design by looking at the relationships shown in Fig. 1 between supply voltage and pipeline throughput [instructions per cycle (IPC)], for three different error correction methods: 1) instruction replay; 2) counterflow pipelining; and 3) our proposed method. The curves plotted in this figure are based on a pipeline of five stages ($N = 5$), each of which is c6288 circuit from the ISCAS benchmark. Error correction method with smaller timing penalty has larger maximum tolerable error rate under the same throughput, and thus can be made to operate at a lower supply voltage. If the target throughput is 0.8, in terms of IPC, our method can reduce supply voltage to 0.8 V, but counterflow pipelining and instruction replay can reduce supply voltage to only 0.86 and 0.9 V, respectively. This voltage benefit increases as the number of pipeline stages increases.

Bubble Razor made a breakthrough by enabling one-cycle error correction [20]. However, it can only be used in designs based on two-phase transparent latches. Since edge-triggered flip-flop or pulsed-latch-based design are more popular, we propose a new one-cycle error correction method for the more frequently used clocking elements.

B. Summary of Contributions

Our main contributions are as follows:

1) the first one-cycle error correction method for flip-flop or pulsed-latch EDS circuits;
2) a pulsewidth determination method for main and shadow latches.

The remainder of this paper is organized as follows. We review previous error correction methods in Section II. Our one-cycle error correction method is introduced in Section III. A time borrowing pulsewidth determination methodology is presented in Section IV. Experimental results are provided in Section V. Finally, the conclusion is drawn in Section VI.

II. Review of Error Correction Methods

Among the several published error correction schemes, instruction replay [18], [21] is the most time consuming. If an error occurs at a particular stage, it is allowed to propagate until the last stage, and then all stages in the pipeline are flushed. If there are N pipeline stages, this will require N cycles. The failed instruction is then reissued to the pipeline, with the clock running at half speed, which should ensure that the failing instruction does not cause another error. This rerun takes $2N$ cycles, and so the completion time of the next instruction that follows the error is delayed by $3N$ cycles. An example of instruction replay is given in Fig. 2(a). Instruction $i2$ fails at stage C in cycle 4. The error propagates to stage E. Then, all the pipeline stages are flushed, from cycles 7 to 11, and instruction $i2$ is issued again in cycle 12. Since the clock frequency has now been halved, $i2$ is only completed at cycle 21, and so the completion time of instruction $i3$ is delayed from cycles 7 to 22.

The counterflow pipelining technique [15] has smaller penalty of $2k$ cycles, where k is the position of the stage, which detects an error in the pipeline. The error is corrected in the next cycle after it is detected and instructions are reissued starting from the next instruction. To flush entire pipeline, flush signal is propagated from the stage that detected an error, via its input stages, to a flush control unit. When the flush signal reaches each stage, that stage is flushed. When the

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

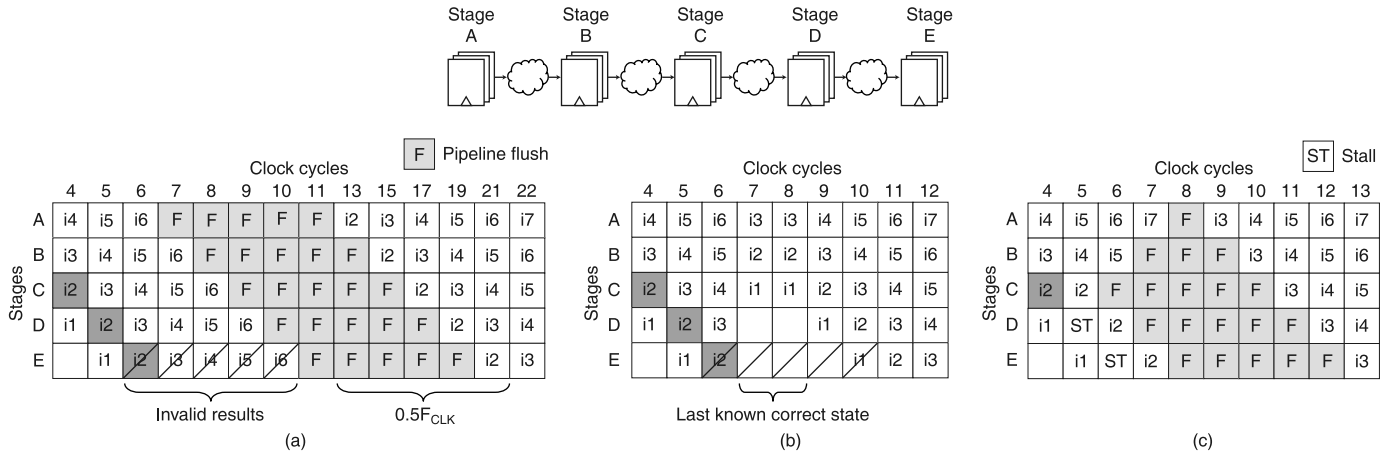


Fig. 2. Examples of error correction after an error occurs at stage C in cycle 4. (a) Instruction replay. (b) Microrollback. (c) Counterflow pipelining.

signal finally reaches the flush control unit, instructions are reissued to the pipeline. In the example shown in Fig. 2(c), instruction $i3$ could have completed in cycle 7 if there had not been an error. However, an error occurs at stage C in cycle 4. Therefore, $i3$ is reissued in cycle 9 and completes in cycle 13; thus the completion time of $i3$ is delayed by $2k = 6$ cycles.

Microrollback [16], [22], [23] has a similar timing penalty to that of counterflow pipelining. At each cycle, the previous state of each pipeline stage is saved to the backing storage. Like instruction replay, this involves no error correction logic. An error signal is issued by the stage at which an error occurs. When this signal reaches the last stage, the state of each stage is rolled back to the last known correct value, provided from backing storage. To correct the error, the backing storages inject the same values to each pipeline twice. The completion time of the instruction that follows the error is delayed by $N - k + 3$. An example of microrollback is given in Fig. 2(b). The error signal reaches the last stage, E, in cycle 6. Then, all the stages are returned to the last states that are known to be correct in cycle 7. Microrollback is not widely used because it increases flip-flop energy dissipation by 15% due to the requirement for backing storage [16].

There are two existing methods of error correction that have a one-cycle timing penalty: 1) global clock gating [15] and 2) Bubble Razor [20]. Global clock gating [15] is conceptually the simplest error correction method of all. When a stage detects an error, all the stages in the pipeline are stalled for one cycle, and shadow latch data are restored to the main flip-flop. However, it may take multiple cycles for the clock-gating signal to be propagated to all the stages in complicated or high-frequency designs, and hence its applicability is limited.

Bubble Razor [20] represents a breakthrough, because it reduces the timing penalty to one cycle based on local stalling, allowing it to be used in complicated and high-frequency designs. However, unlike other methods, Bubble Razor can only be used for two-phase transparent latch-based designs. Therefore, flip-flop datapaths in most existing designs have to be converted to two-phase transparent latch datapaths. This requires extra design effort, as proposed in [20], and the number of stages is doubled, which may lead to an increase of error rate.

Since flip-flop and pulsed latch are more popular clocking elements in current digital circuits than level-sensitive latch, there remains a pressing need for one-cycle error correction in EDS circuits, which use flip-flop or pulsed-latch.

III. New Approach to One-Cycle Error Correction

Since the data in the shadow latch are correct, even in failing stage, the simplest error correction method may be restoring the data from the shadow latch to the main latch. The only problem is that the data coming from the input stage will be lost during the restore cycle. That is the reason why counterflow pipelining reissues the next instruction to the failed instruction after the error correction [15]. Our insight is that the previous stages to the failed one, only main latch needs to be gated, and their shadow latches do not need to be gated. If the main latch of a stage is gated, while its shadow latch is being clocked, that stage can simultaneously capture input data at the shadow latch while retaining its previous data at the main latch. This allows a stage that detects an error to receive the correct data in the very next cycle after error correction.

Fig. 3 shows a circuit-level schematic of our Razor latch. In this design, multiplexer at the input of the Razor latch is placed in the feedback path of the main latch to reduce delay and power consumption. If the instruction $i3$ fails at cycle 3, *clk_m* and *clk_s* are gated at the following cycle after error detection. In cycle 4, the restore signal becomes 1 so that the correct instruction $i3$ is transmitted from the shadow latch to the main latch. Instruction $i4$ is captured at cycle 5.

A. Gating Signal Propagation

To maintain the correctness of the data, each of the stages previous to the one where the error happened must eventually go through a two-cycle process in which its main latch is gated in the first cycle and the data in its shadow latch are restored to its main latch in the second cycle. In addition, we need to prevent the propagation of incorrect data from the stage in which the timing error occurred. To accomplish

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

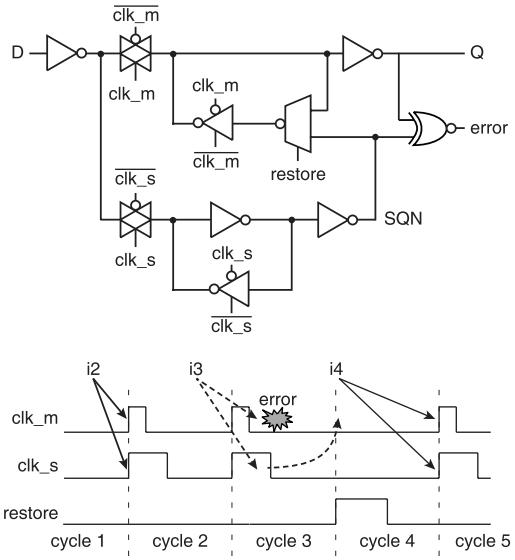
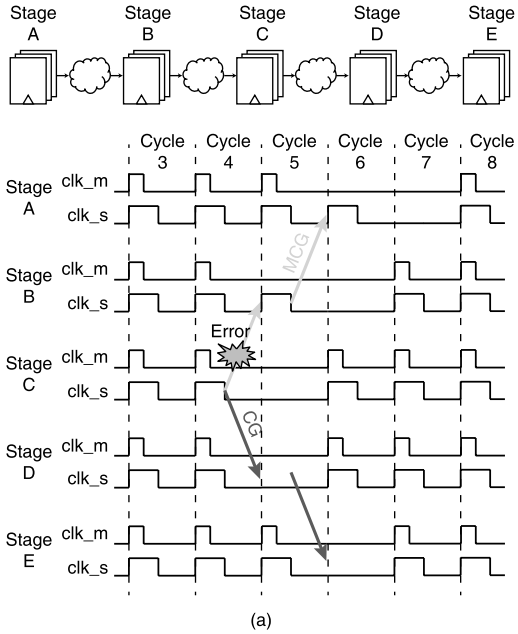


Fig. 3. Circuit-level schematic of our Razor latch.

the requirements mentioned above, we introduce two types of clock gating control signals: 1) CG and 2) MCG. When a stage receives a CG signal, the clock for its main latch, *clk_m*, and the clock for its shadow latch, *clk_s*, are gated for one cycle. The CG signal is propagated from the stage where an error occurs to consecutive output stages. When a stage receives the MCG signal, its *clk_m* clock is gated for one cycle, and then both *clk_m* and *clk_s* are gated in the next cycle. In a similar way to the CG signals, the MCG signals are propagated sequentially to the previous input stages.

An example of the propagation of CG and MCG signals is shown in Fig. 4(a). Suppose that an error occurs at stage C in cycle 4. Each signal is transmitted to the next stage at every cycle, starting at cycle 4. Fig. 4(b) shows the data stored in each stage at each cycle. In cycle 5, instruction $i2$ is restored at stage C by passing the correct data from the shadow latch to the main latch. In the same cycle, the main latch in stage B is gated to retain instruction $i3$, while its shadow latch stores instruction $i4$ sent from stage A. Stage D must be stalled in cycle 3 because its input data from stage C is incorrect. In cycle 6, instruction $i3$, which arrived previously at cycle 5, is captured by stage C.

When multiple errors occur, CG and MCG signals can meet or cross each other. In these cases, the propagation of both signals must be stopped. We introduce two stop conditions to take care of such cases. The first condition is met if CG and MCG are propagated to the same stage; then both signals are stopped. In this case, the main and shadow latches are both gated for one cycle. For example, in Fig. 5, CG and MCG are propagated to stage B in cycle 1. Thus, the main and shadow latches of this stage are both gated in cycle 2, and propagation of CG and MCG is stopped. The second condition is met if a clock-gating signal is propagated to a stage, which has already sent a control signal of the other type in the same cycle, in this case, the propagation of the signal ends at this stage. In Fig. 5, stage B sends a CG signal to stage C, and receives an MCG signal from stage C in cycle 4.



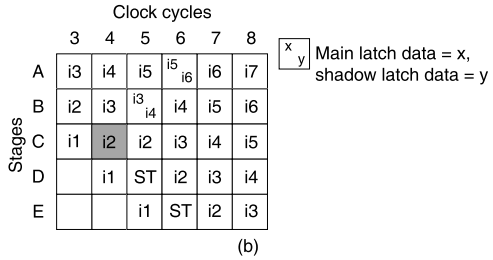


Fig. 4. Correcting an error using our method. (a) Propagation of CG and MCG signals. (b) Instructions stored in each stage during each cycle.

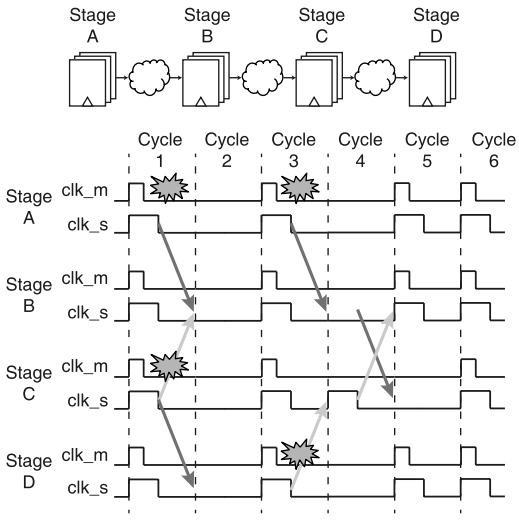


Fig. 5. Propagation of both CG and MCG must be stopped when they meet or cross.

Therefore, the propagation of MCG is stopped at stage B, for the same reason, the propagation of the CG signal is stopped at stage C in cycle 4.

In the linear pipeline circuits, clock gating control signals must arrive at the next stage before rising of clock. This timing constraint for control signal can be expressed as follows:

$$T_{\text{ctrl}} < T_c - W_s \quad (1)$$

International Journal of Research
Available at
https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

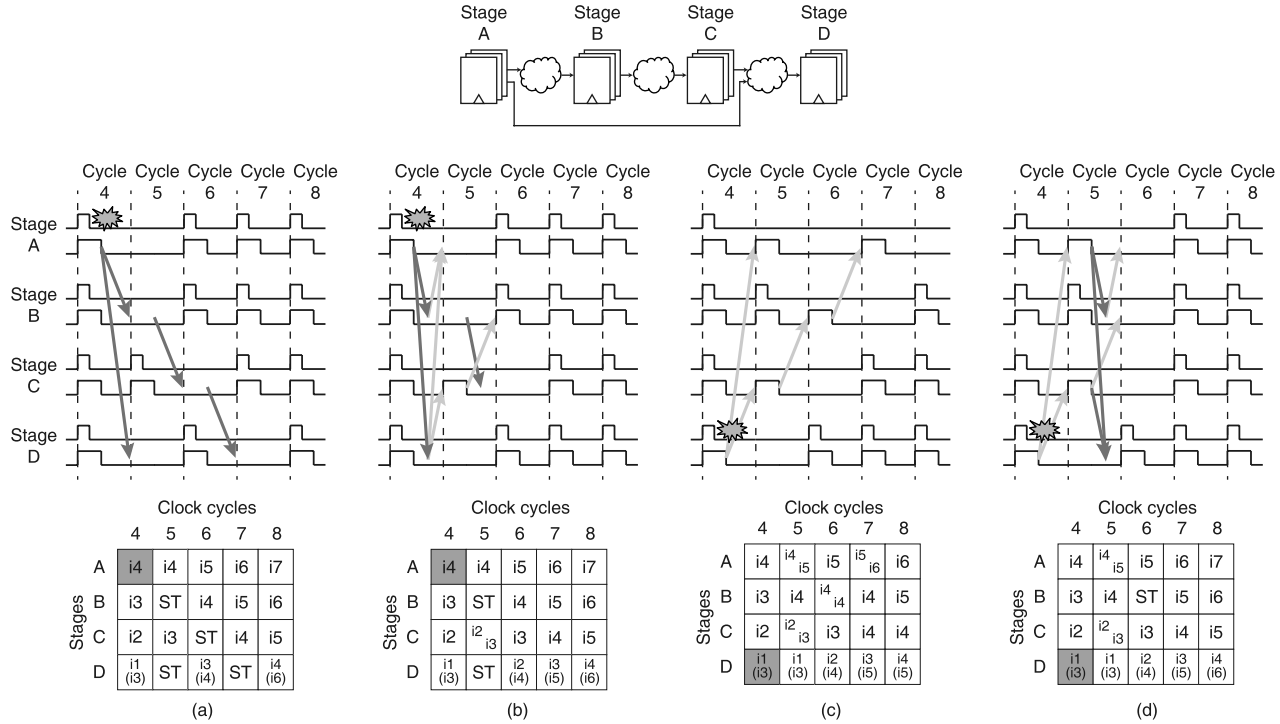


Fig. 6. Pipeline circuit, which has fan-out and fan-in stages. (a) Lost data problem occurs at stage E. (b) Our propagation algorithm is applied. (c) Double-sampling problem occurs at stage C. (d) Our propagation algorithm is applied.

where T_c is clock period and W_s is pulsewidth of shadow latch. If the Razor circuit cannot meet this constraint, W_s must be reduced, which leads to a decrease in window for timing speculation.

B. Extension to General Pipelines

The proposed error correction method can be extended to more general pipelines with multiple fan-in, fan-out, loops, or a combination of these structures. In the case of multiple fan-in and fan-out, there are two problems that need to be addressed.

The first problem is the potential loss of data at a fan-in stage when only some of the input stages send a CG signal. Consider the pipeline architecture with fan-in and fan-out stages shown in Fig. 6. Suppose that an error occurs at stage A, as shown in Fig. 6(a). Then, stage D receives a CG signal from stage A in cycle 4. However, stage C, which is the other input stage of stage D, does not send a CG signal in cycle 4. Therefore, instruction $i2$, sent from stage C, cannot be captured by stage D in cycle 5, and so the pipeline loses instruction $i2$ at stage D. This problem can be solved by modifying the propagation algorithm as follows. If a stage receives a CG signal from any of its input stages, it sends MCG signals to all of its input stages in the same cycle. This is shown in Fig. 6(b), in which stage D sends an MCG signal to its input stages, A and C, in cycle 4. The signal received by stage A is nullified because that stage has already sent a CG signal. Stage C retains instruction $i2$ in cycle 5, and, therefore, instruction $i2$ can be captured by stage D in cycle 6. By modifying the propagation algorithm, each input stage of the fan-in stage will stall for a cycle and propagate the data in the next cycle, and hence data remain synchronized.

The second problem is the double sampling of data at the input stages of a multifan-out stage when not all the output stages have sent an MCG signal. Suppose that an error occurs at stage D, as shown in Fig. 6(c). Then, stage A receives an MCG signal from stage D in cycle 4, but the other input stage, B, does not send an MCG signal to stage A. Therefore, stage B captures instruction $i4$, sent twice from stage A, in cycles 5 and 6. We extend the propagation algorithm to solve this problem as follows. If a stage receives an MCG signal from any of its output stages, it must send a CG signal to all of its output stages in the next cycle. This modification means that each output stage of a fan-out stage will stall for one cycle.

Operation of the modified algorithm is shown in Fig. 6(d). In cycle 5, stage C sends a CG signal to its output stage, D, and stage A also sends CG signal to stages B and D. The signals sent to stage D are nullified because stage D is gated in cycle 5. Propagation of both CG and MCG is stopped because both signals meet at stage B. The MCG signal is nullified but CG is not, and, therefore, stage B sends an MCG signal back to stage A in the same cycle. Stage C is stalled in cycle 6, and thus instruction $i4$ is not double-sampled by stage B.

Our error correction method can also handle loops. The main challenge with loops is usually to prevent infinite looping. However, because CG and MCG are propagated in opposite directions, they meet eventually each other within a loop, and then both signals are stopped. Therefore, infinite looping cannot occur in our scheme. Fig. 7 shows how this works for error occurring before, during, and after a loop.

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

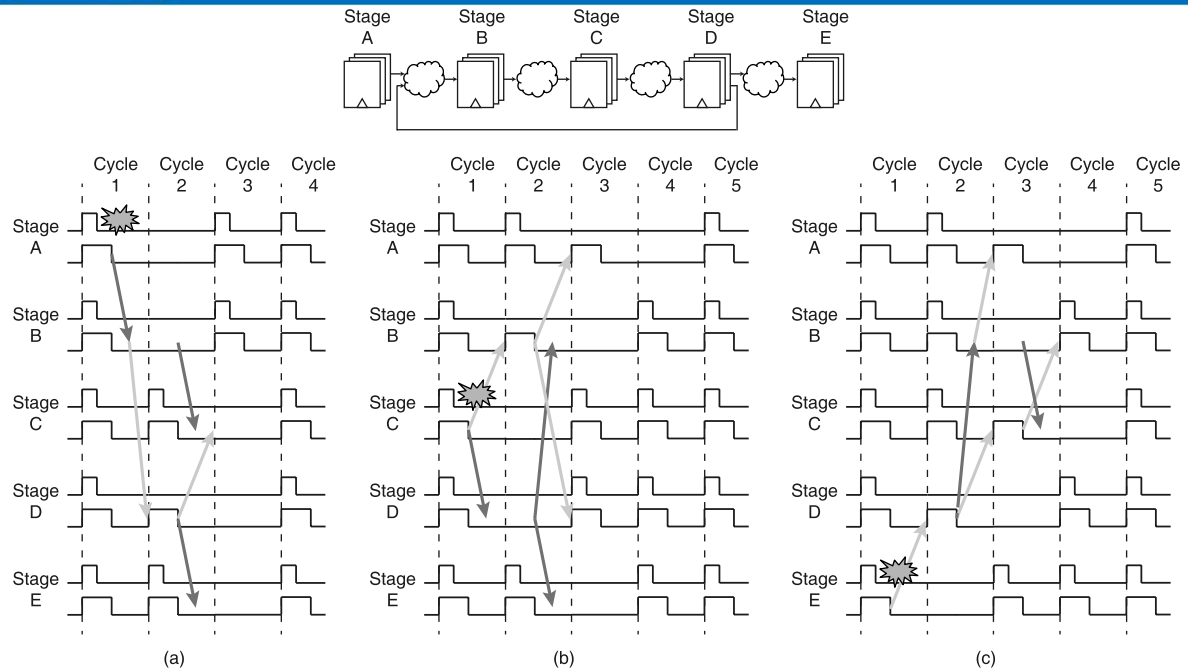


Fig. 7. Different types of loop error. (a) Error occurs before the loop. (b) Error occurs in the loop. (c) Error occurs after the loop.

An error before a loop is shown in Fig. 7(a). Suppose that an error occurs at stage A, and so a CG signal is inserted into the loop. Then, CG and MCG signals are propagated to stages B and D, respectively. In cycle 2, they meet each other at stage C, where the propagation of both signals is, therefore, stopped. In addition, the CG signal is propagated to stage E, which is outside the loop, and then the signal is propagated to the upstream stages.

In the second case, shown in Fig. 7(b), an error occurs at stage C. The CG and MCG signals are propagated round the loop in opposite directions. They cross each other at both stages B and D in cycle 2, and both signals are stopped.

The third case is shown in Fig. 7(c), in which an error occurs at stage E. The MCG signal is propagated back into the loop. Stage D receives MCG, and then sends CG to stage B and MCG to stage C. In cycle 3, the two signals cross each other at both stages B and C, and so both signals are stopped.

The timing constraint for control signal becomes tighter in complicated pipeline circuits, and this can be expressed as follows:

$$T_{\text{ctrl}} < \frac{T_c - W_s}{2}. \quad (2)$$

Hence, care must be taken to prevent clock-gating signal paths from becoming critical in complicated pipeline circuits.

C. SRAM Interfaces

Different from Bubble Razor, which uses two-phase clocking, our proposed scheme does not need to treat SRAM as positive latch. Hence, we do not need to place negative latches around SRAM. Instead, we place flip-flops on the input and output of SRAM and detect the errors at the flip-flips.

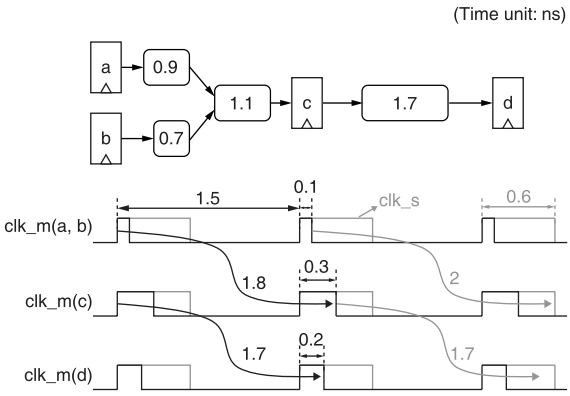


Fig. 8. Time borrowing by means of multiple pulsewidths.

As a result, SRAM array can use the full cycle in contrast to the Bubble Razor in which reading SRAM array needs to be finished in half a cycle [20].

However, our scheme has the same limitation in SRAM write interface as the Bubble Razor because it is not possible to correct the error once the address itself goes wrong. Therefore, the same adjustment as Bubble Razor (clocking write input on the negative clock edge or using the write buffer) is needed [20].

IV. Determination of Pulsewidth

The pulsewidth of the shadow latch is important in all Razor designs, because it determines the timing speculation window. For pulsed-latch-based Razor designs, the pulsewidth of the main latch is also important because some timing violations can be eliminated by time borrowing. Fig. 8 shows four pulsed Razor latches in a circuit with a clock period of 1.5 ns; the minimum pulsewidth is 100 ps, and the setup time,

International Journal of Research
Available at
https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

clock-to-Q, and data-to-Q delays are assumed to be 0 for simplicity. The combinational blocks between latches a and c impose a maximum delay of 2 ns, between latches b and c, the maximum delay is 1.8 ns, and between latches c and d, it is 1.7 ns. All these path delays are larger than the clock period, so that they cause timing errors whenever the corresponding paths are activated. If the width of the clk_m pulse for latch c is set to 300 ps, and that of d is set to 200 ps, as shown in the figure, then the paths from b to c and from c to d do not cause timing errors when the data leave the first latch on the rising edge of clk_m. Note that the activation probability for the top 10% of the most critical paths is usually small [24]. Thus, the input data for a latch often arrive before the rising edge of the clock signal.

Unlike the main latch, the shadow latch must capture the correct data even in the worst case when the data leave the launching latch on the falling edge of clk_m (gray arrows in Fig. 8). The pulsewidths of all the shadow latches are assumed to be 600 ps. If the pulsewidth of $clk\_m(c)$ is increased by 200 ps, then the path from a to c no longer causes an error, thanks to time borrowing. However, it is no longer certain that the shadow latch of Razor latch d always captures the correct data. Thus, we see that care is required to determine a pulsewidth, which minimizes the number of timing violating paths while ensuring that the shadow latches always capture the correct data.

Supposing there is a set of available pulsewidths $\mathcal{W} = \{W_1, W_2, \ldots, W_n\}$, we can state the problem of pulsewidth determination as follows.

Problem 1: Given the netlist of a Razor circuit that uses pulsed-latches together with a set of distinct pulsewidths $\mathcal{W}$ and clock period T_c, the pulsewidth determination problem is to allocate pulsewidths $W_i^m \in \mathcal{W}$ and $W_i^s \in \mathcal{W}$ to main latch and shadow latch in each Razor latch i, respectively, with the objective of minimizing the sum of the activation rates of timing violating paths at the given voltage level $V_{\min}$ and minimizing the sum of the pulsewidths of all shadow latches.

Problem 1 can be solved by formulating it as an integer linear program (ILP). Let $L = \{l_0, l_1, \ldots, l_{N_t}\}$ be a set of Razor latches, where l_0 is a virtual latch launching primary inputs. Let $p_{i,j}^k$ denote the kth longest path from l_i to l_j, let $P_t = \{p_{1,2}^1, p_{1,2}^2, p_{1,3}^1, \ldots, p_{k-1,k}^4\}$ be the set of paths whose delays are longer than $T_c + \min(\mathcal{W})$. The following notation is also used in the ILP formulation.

1) $x_{i,j}^k$: A Boolean variable that indicates whether path $p_{i,j}^k$ violates a timing constraint. If $p_{i,j}^k$ is a violating path then $x_{i,j}^k = 1$; otherwise $x_{i,j}^k = 0$.
2) $AR_{i,j}^k$: The activation rate of path $p_{i,j}^k$.
3) N_l: The total number of latches.
4) $y_{i,k}^m$: A Boolean variable that indicates the pulsewidth of the main latch in l_i.
5) $y_{i,k}^s$: A Boolean variable that indicates the pulsewidth of the shadow latch in l_i.
6) T_{su}: Setup time of Razor latch.
7) T_{dq}: Data-to-Q delay of Razor latch.

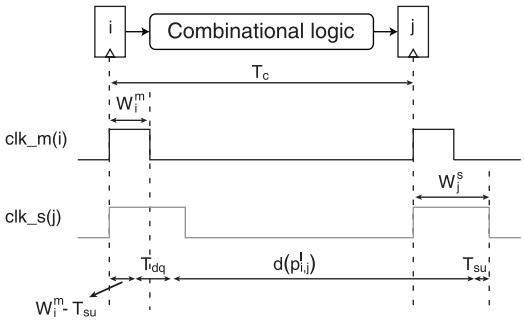


Fig. 9. Timing diagram for (7) in the ILP formulation.

Objective Function: We formulate the following objective for the pulsewidth allocation problem:

$$\min C_u \sum_{(p_{i,j}^k) \in P_t} AR_{i,j}^k x_{i,j}^k + \sum_{i=1}^{N_t} \left(W_i^s + W_i^m\right). \tag{3}$$

Using a large constant for C_u, this objective function minimizes the sum of the path activation rates of violating paths, and then minimizes the sum of the pulsewidths of all Razor latches.

This ILP formulation is subject to pulsewidth and timing constraints, which can be expressed as follows:

$$W_i^m = \sum_{k=1}^{n} y_{i,k}^m W_k, \quad W_i^s = \sum_{k=1}^{n} y_{i,k}^s W_k \quad \forall l_i \in L \tag{4}$$

$$\sum_{k=1}^{n} y_{i,k}^m = 1, \quad \sum_{k=1}^{n} y_{i,k}^s = 1 \quad \forall l_i \in L \tag{5}$$

$$W_i^m \le W_i^s \quad \forall l_i \in L \tag{6}$$

$$d\left(p_{i,j}^l\right) \le T_c + \left(W_j^s - T_{\text{su}}\right) - \left(W_i^m - T_{\text{su}} + T_{\text{dq}}\right) \quad \forall p_{i,j}^l \in P_t \tag{7}$$

$$\frac{T_c + W_j^m - d\left(p_{i,j}^k\right)}{2T_c} + x_{i,j}^k \ge 0 \quad \forall p_{i,j}^k \in P_t. \tag{8}$$

Constraint (7) ensures that the data in the shadow latch are always correct even in the worst case. This worst case scenario is shown in Fig. 9. The latest time at which data can depart from main latch in i is $W_i^m - T_{\text{su}}$, and the latest time at which data can arrive to shadow latch in j is $T_c + W_j^s - T_{\text{su}}$. Constraint (8) states that if $d(p_{i,j}^k)$ is larger than $T_c + W_j^m$, and, therefore, $p_{i,j}^k$ is a violating path, then $x_{i,j}^k$ must be 1. Constraint (6) sets lower and upper bounds on the pulsewidth of the main and shadow latches.

V. Experimental Results

The schematic of control logic for our error correction scheme is shown in Fig. 10. When a stage receives an MCG signal from any of its output stages (MCG_in becomes high), then, cg_m becomes high and EN_clk_m becomes low. Thus, clk_m is gated at that cycle. At the next cycle, MCG_out and CG_out become high. Thus, both EN_clk_m and EN_clk_s become low, MCG signals are propagated to the input stages of the current stage, and CG signals are propagated back to its output stages.

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

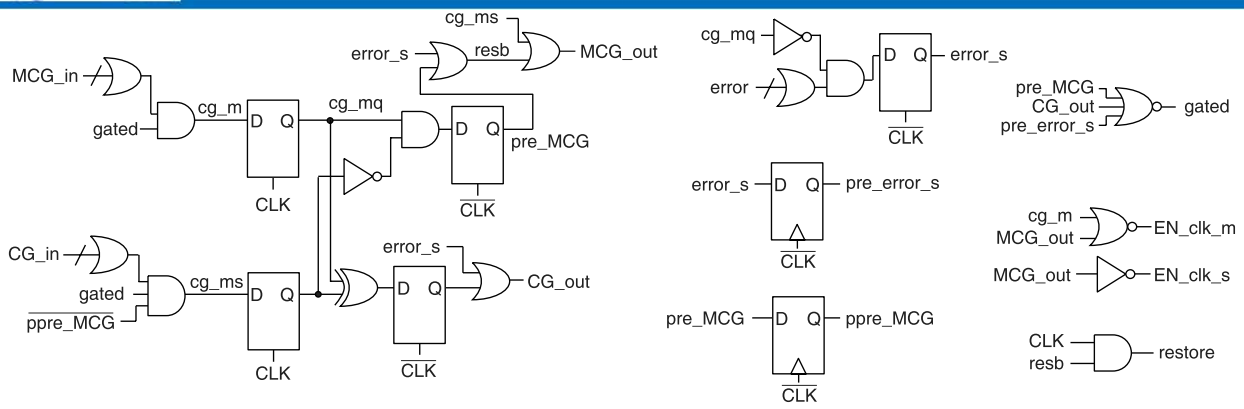


Fig. 10. Schematic of the control logic for our scheme.

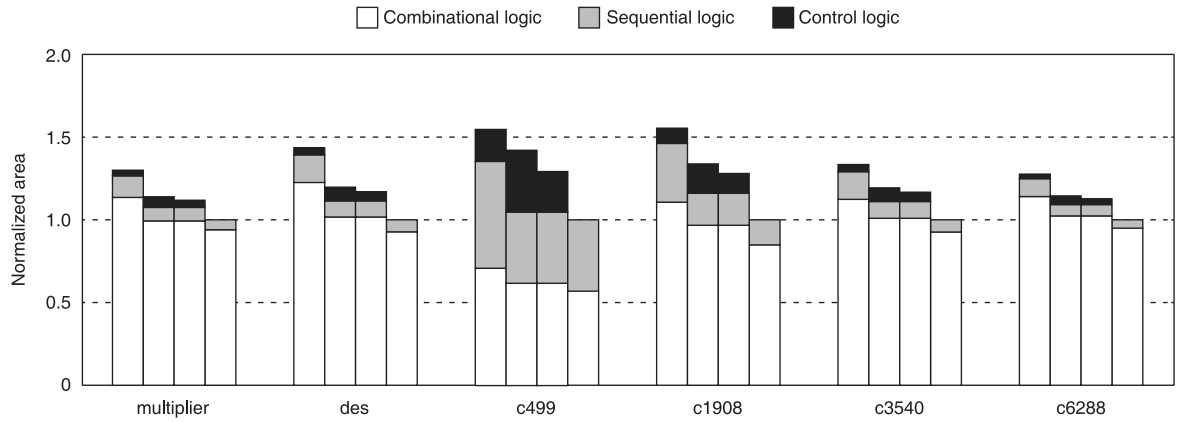


Fig. 11. Comparison of normalized area for our method (the first bars from the left), counterflow pipelining (the second bars from the left), instruction replay (the third bars from the left), and baseline (the fourth bars from the left).

When a stage receives a CG signal from any of its input stages, node *cg_ms* becomes high and then *MCG_out* is high. As a result, both *EN_clk_m* and *EN_clk_s* become low, and MCG signals are propagated back to its output stages in the same cycle.

When an error occurs at a stage, node *error_s* becomes high. Then, *MCG_out* and *CG_out* become high, so that MCG signals are propagated to its input stages, and CG signals are propagated to its output stages. Both clocks are also gated at the current cycle. The node labeled gated represents the stop conditions for both MCG and CG signals.

A. Simulation Setup

To assess the effectiveness of our error correction method, we compiled 15 linear pipelined circuits and ten more complicated pipelined circuits using a 28-nm commercial library. The linear pipelines had five, eight, or ten stages, each of which was one of six different pulsed-latch Razor circuits: 1) c499[1]; 2) c1908; 3) c3540; and 4) c6288 from the ISCAS benchmark, a 32-bit multiplier from Designware from Synopsys, and a modified data encryption standard (DES) from OpenCores. The circuits were synthesized using Design Compiler [25], with the timing constraints set to 90% of the critical path delay of each circuit, when synthesized without any constraint. The most complicated pipeline had multiple fan-in, fan-out, and loop structure with eight stages. The pulsewidth of each main latch was 130 ps, which is the minimum width at 0.8 V under nominal process corner, and that of the shadow latches were 430 ps. Extra delay buffers were inserted into all the circuits to fix hold violations.

Note that there is the possibility of metastability problem in the proposed Razor latches. Since all error correction methods in this paper have same overhead for metastability detector and the main focus of this paper is to reduce the timing penalty of error correction down to one cycle, we did not deal with metastability problem. However, to prevent the metastability problem, metastability detector [15] must be used.

Fig. 11 shows the area of each of the circuits comprising the stages, normalized to the total area of original pulsed-latch-based pipeline with no Razor latch. It is worthwhile to note that our method requires all latches to be replaced by Razor latches because the shadow latch is used for error correction as well as error detection. In contrast, the counterflow pipelining and instruction replay methods only require Razor latches for part of the pipeline, because the shadow latches are only used for error detection. Thus, we replaced all the latches that had negative timing slacks at 0.8 V by Razor latches. This is why the area required for sequential logic by our method is greater

[1]In case of c499, each stage consists of two c499 circuits.

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

TABLE I
RESULTS FROM LINEAR PIPELINES WITH A TARGET THROUGHPUT OF 1

Number of stages	Base circuit	Our method		Counterflow		Replay		Baseline	
		Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]
5	multiplier	0.94	548	0.94	524	0.94	511	1.00	600
	des	0.92	401	0.92	384	0.92	372	1.00	447
	c499	0.92	232	0.92	203	0.92	196	1.00	249
	c1908	0.94	285	0.94	261	0.94	253	1.00	311
	c3540	0.94	493	0.94	470	0.94	452	1.00	544
	c6288	0.94	794	0.94	765	0.94	745	1.00	870
	Avg. normalized		1.00		0.94		0.91		1.10
8	multiplier	0.94	808	0.94	789	0.94	774	1.00	891
	des	0.92	591	0.92	571	0.92	558	1.00	654
	c499	0.92	344	0.92	312	0.92	301	1.00	368
	c1908	0.94	433	0.94	405	0.94	394	1.00	470
	c3540	0.94	746	0.94	718	0.94	692	1.00	825
	c6288	0.94	1165	0.94	1115	0.94	1097	1.00	1264
	Avg. normalized		1.00		0.95		0.93		1.09
10	multiplier	0.94	951	0.94	922	0.94	902	1.00	1034
	des	0.92	698	0.92	672	0.92	649	1.00	773
	c499	0.92	404	0.92	368	0.92	358	1.00	428
	c1908	0.94	493	0.94	464	0.94	443	1.00	534
	c3540	0.94	875	0.94	852	0.94	830	1.00	957
	c6288	0.94	1399	0.94	1323	0.94	1310	1.00	1512
	Avg. normalized		1.00		0.95		0.93		1.09

than that needed by either counterflow pipelining or instruction replay. The area overhead for converting all latches to Razor latches is 13% on average. This overhead increases as the area portion of sequential elements increases; c499 and c1908 have large area portion of sequential elements and, therefore, area overhead of these circuit is 21% on average.

Our method also requires a larger area for combinational logic, as shown in Fig. 11, because it needs more delay buffers. Because of these reasons, our method has 42% large area than original circuit on average.

Counterflow pipelining requires the largest area for control logic. This consists of flip-flops for the ID of the current instruction and for flush ID of instruction where an error occurs. Counterflow pipelining and instruction replay both need these flip-flops, but our method requires none.

B. Linear Pipelines

We applied 300 random vectors to each of the five linear pipelined circuits, set target throughputs of 1, 0.9, and 0.8, and determined the throughput of each circuit by fast SPICE simulation [26]. In each test, the initial supply voltage was set to 1 V and then gradually reduced in 0.02 V increments until the throughput failed to meet the target. Energy consumption was measured at the lowest supply voltage, which achieved the target throughput. We used the same experimental setting to simulate the counterflow pipelining and instruction replay techniques.

Table I shows the lowest successful voltage level and the total energy consumption when the target throughput was 1. The energy consumption of our method was 4%–9% larger than that of counterflow pipelining or instruction replay. That is expected, because our method requires all latches to be replaced by Razor latches, and needs a large number of buffers to accommodate their hold-time violations. Counterflow pipelining and instruction replay require the same number of Razor latches, as shown in Fig. 11. Instruction replay has a simpler control logic than counterflow pipelining, and a lower energy consumption.

Tables II and III have the same layout as Table I, but show the results for target throughputs of 0.9 and 0.8, respectively. Note that our method incurs a timing penalty for error correction, which is significantly lower than those of the other methods. Using our method, the voltage, which corresponds to the minimum total energy consumption, is lower than those for the other methods, as shown in Tables II and III. Compared with counterflow pipelining and instruction replay, our method reduces the total energy dissipated by the five-stage pipelines by 8% and 9%, respectively, when the target throughput is 0.9. Note that the differential increases with the number of pipeline stages: the equivalent reductions for the ten-stage pipelines are 12% and 11%, respectively, when the target throughput is 0.9. This can be explained by recollecting that timing penalty for each error corrected depends on the number of pipeline stages with both of the previous schemes, and they cannot correct multiple simultaneous errors.

Since instruction replay has the largest timing penalty for each error corrected: this is $3N$ cycles, where N is the number of pipeline stages. Thus, it permits the least reduction in supply voltage before it fails to meet the throughput target, as we would expect; and this remains true as the target throughput is reduced.

Our method reduces the energy consumption of the five-stage circuits by 15% and 16%, compared with counterflow pipelining and instruction replay, respectively, when the target throughput is 0.8. This differential increases with more pipeline stages: for the ten-stage circuits, the equivalent figures are 17% and 18%.

C. More Complicated Pipelines

Simulations showed that our method works correctly with multiple fan-in, fan-out, and loops, which counterflow

International Journal of Research
Available at
https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

TABLE II

RESULTS FROM LINEAR PIPELINES WITH A TARGET THROUGHPUT OF 0.9

Number of stages	Base circuit	Our method		Counterflow		Replay		Baseline	
		Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]
5	multiplier	0.84	423	0.90	470	0.92	489	1.00	600
	des	0.84	336	0.88	364	0.90	355	1.00	447
	c499	0.84	188	0.88	194	0.90	189	1.00	249
	c1908	0.86	230	0.92	250	0.92	246	1.00	311
	c3540	0.84	392	0.88	417	0.90	432	1.00	544
	c6288	0.84	638	0.90	709	0.92	707	1.00	870
	Avg. normalized		1.00		1.08		1.08		1.36
8	multiplier	0.88	672	0.92	753	0.94	774	1.00	891
	des	0.86	505	0.90	558	0.92	558	1.00	654
	c499	0.86	294	0.90	304	0.92	300	1.00	368
	c1908	0.90	376	0.92	392	0.94	394	1.00	470
	c3540	0.86	612	0.90	682	0.92	660	1.00	825
	c6288	0.86	942	0.92	1061	0.94	1097	1.00	1264
	Avg. normalized		1.00		1.09		1.09		1.30
10	multiplier	0.88	785	0.92	887	0.94	902	1.00	1034
	des	0.88	581	0.92	672	0.92	649	1.00	773
	c499	0.88	337	0.92	368	0.92	356	1.00	428
	c1908	0.90	420	0.92	464	0.94	443	1.00	534
	c3540	0.88	729	0.92	852	0.94	830	1.00	957
	c6288	0.88	1170	0.92	1301	0.94	1310	1.00	1512
	Avg. normalized		1.00		1.13		1.11		1.30

TABLE III

RESULTS FROM LINEAR PIPELINES WITH A TARGET THROUGHPUT OF 0.8

Number of stages	Base circuit	Our method		Counterflow		Replay		Baseline	
		Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]
5	multiplier	0.82	364	0.86	420	0.90	445	1.00	600
	des	0.80	281	0.86	339	0.88	337	1.00	432
	c499	0.82	169	0.86	180	0.88	177	1.00	249
	c1908	0.82	199	0.88	220	0.90	230	1.00	318
	c3540	0.82	351	0.86	398	0.88	409	1.00	568
	c6288	0.80	545	0.86	641	0.90	650	1.00	884
	Avg. normalized		1.00		1.14		1.16		1.58
8	multiplier	0.84	589	0.90	716	0.92	741	1.00	891
	des	0.84	451	0.88	534	0.90	533	1.00	635
	c499	0.84	270	0.90	304	0.92	300	1.00	368
	c1908	0.86	337	0.90	380	0.92	381	1.00	470
	c3540	0.84	570	0.90	682	0.92	660	1.00	855
	c6288	0.84	865	0.90	1010	0.92	1048	1.00	1255
	Avg. normalized		1.00		1.17		1.18		1.44
10	multiplier	0.86	711	0.90	840	0.94	902	1.00	1034
	des	0.84	520	0.90	647	0.92	649	1.00	773
	c499	0.86	308	0.90	353	0.92	356	1.00	428
	c1908	0.88	392	0.92	464	0.94	443	1.00	534
	c3540	0.86	671	0.92	852	0.94	830	1.00	957
	c6288	0.86	1070	0.92	1278	0.94	1310	1.00	1512
	Avg. normalized		1.00		1.20		1.21		1.42

or instruction replay schemes could not handle. Counterflow pipelining cannot handle multiple fan-in, fan-out, or loops, which fail when a corrected instruction arrives at a stage having multiple fan-in. Instruction replay can only handle fan-in and fan-out. In these pipelines, a stage may receive two or more instructions. When the error signal reaches the last stage, the earliest instruction is replayed at half the clock frequency, and the original clock is only restored when the latest instruction reaches the last stage. As the distance between a fan-in stage and a fan-out stage increases, the number of cycles needed to replay a failed instruction increases. N cycles are needed for pipeline flushing, and $2N + d - 1$ are needed to replay the failed instruction, where d is the distance between the fan-in and the fan-out stages. Thus, the completion of the next instruction is delayed by $3N + d - 1$ cycles. In a loop, the instructions would have to be replayed from the first instruction, incurring too large a timing penalty to be practical.

We constructed three different fan-in and fan-out circuits by modifying linear pipelines, as shown in Fig. 12. Table IV shows the minimum voltage level and total energy dissipated by these circuits when the target throughput is 0.8. The timing penalties for instruction replay are 16, 26, and 34 cycles. Successful voltage is higher than those for the linear pipelines. Our method uses 18%, 20%, and 21% lesser energy than instruction replay, for pipelines

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 04 Issue 05
April 2017

TABLE IV
EXPERIMENTAL RESULTS FOR FAN-IN AND FAN-OUT PIPELINES WHEN THE TARGET THROUGHPUT IS 0.8

Number of stages	Base circuit	Our method		Instruction replay	
		Voltage [V]	Energy [pJ]	Voltage [V]	Energy [pJ]
5	multiplier	0.82	374	0.90	468
	des	0.80	273	0.90	337
	c1908	0.82	186	0.90	219
	c3540	0.82	338	0.90	415
	c6288	0.80	561	0.92	681
	Avg. normalized		1.00		1.22
8	multiplier	0.84	601	0.92	757
	des	0.82	415	0.92	540
	c1908	0.84	317	0.94	391
	c3540	0.82	548	0.92	664
	c6288	0.82		0.94	1029
	Avg. normalized		1.00		1.25
10	multiplier	0.86	684	0.94	932
	des	0.84	503	0.92	631
	c1908	0.86	370	0.94	434
	c3540	0.84	638	0.94	814
	c6288	0.84	993	0.94	1273
	Avg. normalized		1.00		1.27

TABLE V
ILP RESULTS FOR FIVE-STAGE LINEAR PIPELINES

Base circuit					ILP		
Name	#Gates	#Latches	#Violated paths	#Hold buffers	ΔViolated paths	ΔHold buffers	Time (s)
multiplier	3475	160	1440	350	226	86	6252
des	2360	240	2550	425	573	121	12372
c499	1430	160	1360	255	215	51	5538
c1908	1715	165	1955	325	366	91	6798
c1908_c499	2615	165	2040	445	389	129	6918
c3540	2855	110	2520	430	450	113	2706
c6288	5255	160	1530	470	276	121	7284
c6288_c499	6220	160	1615	655	301	153	8540

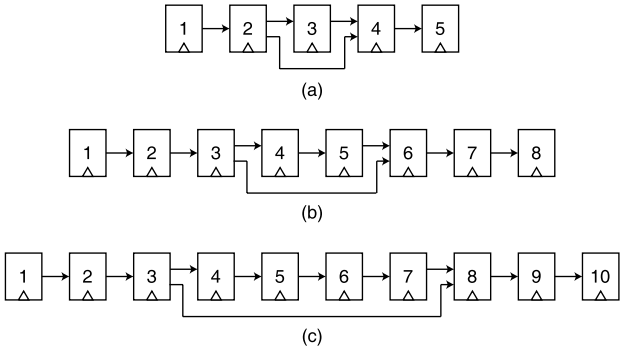


Fig. 12. Multiple fan-in and fan-out pipelines created by modifying (a) five-stage, (b) eight-stage, and (c) ten-stage linear circuits.

with five, eight, and ten stages, respectively. It is reasonable to expect the timing penalty for instruction replay to increase with the length of the pipeline. In addition, our method gives opportunities to correct multiple errors in different stages, with a lower time penalty than would be needed to correct each error individually.

D. Determination of Pulsewidth

We use five different pulsewidths 130, 170, 210, 250, and 430 ps. Table V shows the results of ILP. In the case of c1908_c499 (c6288_c499), each stage consists of c1908 (c6288) and c499 circuits. The columns 2 and 3 show the total number of combinational gates and latches of five-stage pipelined circuits. The column 4 shows the total number of violated paths at 0.75 V when *clk_m* and *clk_s* are 130 and 430 ps, respectively. As the number of latches increases, the number of ILP (4)–(7) increases. Thus, DES circuit, which has the largest number of latches, has the longest runtime. The column 5 indicates the number of hold buffers of each circuit. Since all latches are replaced by Razor latches and *clk_m* is 430 ps, relatively large number of hold buffers is needed. The column 6 indicates the reduction of the number of violated paths after solving the ILP. The number of violated paths is reduced by 19% on average. The DES circuit has the largest reduction (22%) of the violated paths. When a Razor latch at which violated paths terminate drives short paths only, the violated paths can be fixed using time borrowing. The DES circuit has the largest ratio of such Razor latches over timing violated Razor latches. Multiplier has the smallest reduction (16%) of violated paths and it has the smallest ratio of such Razor latches. The column 8 indicates the reduction of the number of hold buffers to be needed. The objective function of ILP minimizes the sum of pulsewidths of all Razor latches. Thus, the shadow latches, which do not have any violated paths, can have short *clk_s*. This leads to the reduction of hold buffers in the results of ILP.

VI. CONCLUSION

Voltage scaling based on timing speculation has become the most promising way to reduce the power consumption

in the nanometer regime, but its effectiveness depends on the overall timing penalty, and, therefore, error correction needs to be completed as fast as possible. The Bubble Razor scheme [20] made a breakthrough by introducing one-cycle error correction, but their method can be applied to two-phase transparent latch designs only. The contribution of this paper is a new one-cycle error correction method that can be applied to more widely used clocking elements, such as flip-flops and pulsed latches. Unlike most of the previous error correction methods except Bubble Razor, this method can handle complicated pipeline architectures, which include multiple fan-in, fan-out, and loops. Results from the simulation of linear pipelines show that our new method consumes 8%–16% lesser energy than existing techniques in a five-stage pipeline, and 11%–18% lesser energy in a ten-stage pipeline. In a pipeline with fan-in and fan-out, the new method uses 18%–21% lesser energy than instruction replay method.

REFERENCES

[1] I. Shin, J.-J. Kim, Y.-S. Lin, and Y. Shin, "A pipeline architecture with 1-cycle timing error correction for low voltage operations," in *Proc. IEEE Int. Symp. Low Power Electron. Design*, Sep. 2013, pp. 199–204.

[2] R. K. Krishinamurthy, A. Alvandpour, V. De, and S. Borkar, "High-performance and low-power challenges for sub-70 nm microprocessor circuits," in *Proc. Custom Integr. Circuits Conf.*, May 2002, pp. 125–128.

[3] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[4] K. Bernstein *et al.*, "High-performance CMOS variability in the 65-nm regime and beyond," *IBM J. Res. Develop.*, vol. 50, nos. 4–5, pp. 433–449, Jul./Sep. 2006.

[5] K. J. Nowka *et al.*, "A 32-bit PowerPC system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1441–1447, Nov. 2002.

[6] A. Muhtaroglu, G. Talyor, and T. Rahal-Arabi, "On-die droop detector for analog sensing of power supply noise," *IEEE J. Solid-State Circuits*, vol. 39, no. 4, pp. 651–660, Apr. 2004.

[7] M. Nakai *et al.*, "Dynamic voltage and frequency management for a low-power embedded microprocessor," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 28–35, Jan. 2005.

[8] M. Agarwal, B. C. Paul, M. Zhang, and S. Mitra, "Circuit failure prediction and its application to transistor aging," in *Proc. 25th IEEE VLSI Test Symp.*, May 2007, pp. 277–286.

[9] J. Tschanz *et al.*, "Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging," in *IEEE Int. Solid-State Circuits Conf., Dig. Tech. Papers*, Feb. 2007, pp. 292–293.

[10] T. Fischer, J. Desai, B. Doyle, S. Naffziger, and B. Patella, "A 90-nm variable frequency clock system for a power-managed itanium architecture processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 218–228, Jan. 2006.

[11] A. Drake *et al.*, "A distributed critical-path timing monitor for a 65 nm high-performance microprocessor," in *IEEE Int. Solid-State Circuits Conf., Dig. Tech. Papers*, Feb. 2007, pp. 398–399.

[12] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De, "Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance," in *Proc. Symp. VLSI Circuits*, Jun. 2009, pp. 112–113.

[13] C. R. Lefurgy *et al.*, "Active management of timing guardband to save energy in POWER7," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2011, pp. 1–11.

[14] A. K. Uht, "Achieving typical delays in synchronous systems via timing error toleration," Dept. Elect. Comput. Eng., Univ. Rhode Island, Kingston, RI, USA, Tech. Rep. 032000-0100, 2000.

[15] D. Ernst *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2003, pp. 7–18.

[16] 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./Dec. 2004.

[17] B. Greskamp and J. Torrellas, "Paceline: Improving single-thread performance in nanoscale CMPs through core overclocking," in *Proc. 16th Int. Conf. Parallel Archit. Compilation Techn.*, Sep. 2007, pp. 213–224.

[18] S. Das *et al.*, "Razor II: *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.

[19] K. A. Bowman *et al.*, "Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 49–63, Jan. 2009.

[20] M. Fojtik *et al.*, "Bubble Razor: An architecture-independent approach to timing-error detection and correction," in *IEEE Int. Solid-State Circuits Conf., Dig. Tech. Papers*, Feb. 2012, pp. 488–490.

[21] K. A. 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.

[22] Y. Tamir and M. Tremblay, "High-performance fault-tolerant VLSI systems using micro rollback," *IEEE Trans. Comput.*, vol. 39, no. 4, pp. 548–554, Apr. 1990.

[23] J. Crop *et al.*, "Error detection and recovery techniques for variation-aware CMOS computing: A comprehensive review," *J. Low Power Electron. Appl.*, vol. 1, no. 3, pp. 334–356, 2011.

[24] M. Choudhury, V. Chandra, K. Mohanram, and R. Aitken, "TIMBER: Time borrowing and error relaying for online timing error resilience," in *Proc. Design, Autom., Test Eur. Conf. Exhibit.*, Mar. 2010, pp. 1554–1559.

[25] *Design Compiler User Guide*, Synopsys, Mountain View, CA, USA, Sep. 2011.

[26] *CustomSim User Guide*, Synopsys, Mountain View, CA, USA, Mar. 2013.