Error Detection and Recovery Techniques for Timing Speculative Processors

Omid Assare

University of California, San Diego

Abstract

Variability in various system parameters presents itself as design-time uncertainty in the performance of modern processors. To ensure correct operation, conventional processors are designed pessimistically for the "worst case," leading to large safety margins. By relaxing the strict requirement of error-free operation, timing speculative processors replace these safety margins with additional circuitry to tolerate possible timing errors. In this paper, we review the major error detection and recovery schemes used for timing speculation and discuss the main challenges in the design and analysis of these techniques.

1. Introduction

As the semiconductor technology scales into ever deeper regimes, the amount of variability in manufacturing process parameters, such as transistor threshold voltage and critical dimensions is increased [16]. Also concomitant to the aggressive scaling of transistors is the increased sensitivity of their speed to variations in environmental conditions such as the supply voltage and temperature. Consequently, system performance varies significantly both across the manufactured parts and over time.

The sources of variability can be categorized according to their spatial reach as well as their temporal rate of change. Spatial reach determines whether a source affects all transistors of a chip (global) or only a few transistors in close proximity (local). For instance, process variation has global or die-to-die (D2D) and local or within-die (WID) components [2]. A recent experimental Intel processor shows around 50% performance variation among its 80 cores when operated at $0.8V$ due to WID process variation alone [10]. Both D2D and WID components of process variation, however, are in the same category based on their temporal rate of change. Variations caused by these sources are called static variations since their magnitude remains constant during the lifetime of the chip. Dynamic sources of variation, on the other hand, cause variations that change during the operation of the chip. They include slow variations such as temperature hotspots that are spread over thousands of clock cycles as well as fast variations such as supply voltage fluctuations that occur over the course of several clock cycles. Ambient temperature variations can also be expected to be dynamic. International Technology Roadmap for Semiconductors (ITRS) projects supply power variation to be 10% while the operating temperature can vary from 30 to 175°C resulting in several tens of percent performance change [16]. Finally, it is useful to consider input data (running code and its input in case of processors) as variation sources. A recent study concluded that input data to a single program can cause variations comparable in magnitude to those caused by other variability sources such as process variation [1].

Circuit designers have addressed the growing uncertainty in the timing characteristics of variability-affected systems by continuous widening of timing guardbands. Each source of variability is accounted for by adding more guardbands. While more sophisticated design-time analysis methods such as Statistical Static Timing Analysis (SSTA) have tried to reduce design pessimism, guardbands continue to increase steadily with each technology generation [16], leading to loss of performance and increased costs due to over-design.

1.1. Reducing Timing Guardbands: Predict and Prevent

Traditional approaches specifically aimed at reducing timing guardbands monitor the state of the system and reduce timing guardbands when possible. For instance conventional Dynamic Voltage and Frequency Scaling (DVFS) adjusts the system's operating point based on precharacterized safe values stored in a look-up table [28][27]. Canary circuits are a more sophisticated class of system state monitors that try to monitor the critical path's available slack directly [13][6][19][11]. One such technique, called Active Management of Timing Guardband [13] , is implemented in IBM POWER7 server where Critical Path Monitors (CPM) measure the available timing margin under thermal fluctuations, voltage skewing, workload-induced voltage and temperature variations, etc., and a control unit adjusts processor voltage and frequency to achieve error-free operation while reducing the guardbands. While these techniques provide a low-cost solution for reducing timing guardbands, they are unable to account for local and fast changing dynamic variations including delay variations caused by input data. Therefore, guardbands associated with these variations cannot be removed and the

associated costs remain.

1.2. Eliminating Timing Guardbands: Detect and Recover

Recent research efforts have increasingly focused on a class of processors called *timing speculative* processors. Traditional sequential circuit designs work under the strict condition that voltage and frequency of the circuit should be set such that no timing violations can occur. Timing speculation, on the other hand, allows timing errors by removing timing margins and relies on circuit- and microarchitecture-level techniques to detect and recover from these errors. In this paper, we focus on these systems and present a review of the major techniques published in recent years. While both detection and recovery of timing errors can be performed at various levels of the hardware/software stack, in this paper we focus on circuit level detection and microarchitecture level recovery techniques.

Overview. The remainder of the paper is organized in three sections. We start in section 2 with a review of new circuits designed for detecting timing errors. These circuits which are referred to as Error Detection Sequential (EDS) circuits are designed to add timing error detection capabilities to the conventional sequential circuits (i.e. latches and flip-flops). Next, in section 3, we review a number of microarchitectural methods that are designed to restore the correct system state after a timing error is detected by the EDS circuits. Finally, section 4 presents a brief overview of a timing model designed for performance modeling of timing speculative systems.

2. Error Detection

In this section, we present a review of the major EDS circuits. These circuits are designed to add timing error detection capabilities to the conventional sequential circuits (i.e. latches and flip-flops) of digital systems. If the input data signal arrives late (i.e. after the clock edge) at their input pins, EDS circuits raise an error signal which is used by the error recovery logic to restore the correct system state. Error recovery schemes are reviewed in section 3. The major challenges in the design of EDS circuits are energy overhead and possible occurrence of metastability. An ideal EDS circuit would detect a timing error if and only if the input signal arrives after the clock edge while (i) enabling fast, low-overhead error recovery, (ii) incurring little energy overhead, and (iii) minimizing the probability of metastability. We introduce these issues in describing the first EDS circuit proposal, known as *Razor* [8] and explain how subsequent methods have tried to address them.

In general, EDS circuits take one of the following approaches in detecting timing errors:

Double Sampling: In addition to the conventional flip-flop that samples the input data at the edge of the clock, these EDS circuits sample the data a second time *after* the clock edge. The second sample is guaranteed to be correct using design-time worst-case timing analysis. Consequently, the two samples are compared and a timing error is declared in case of a mismatch. EDS circuits described in sections 2.1 and 2.2 are of this type.

Transition Detection: Instead of a second delayed sampling, these EDS circuits dynamically monitor the input data or some internal node *after* the clock edge. A transition during this period is indicative of a late arriving input signal and a timing error is declared if such a transition is detected. EDS circuits described in sections 2.4, 2.3, and 2.5 take this approach.

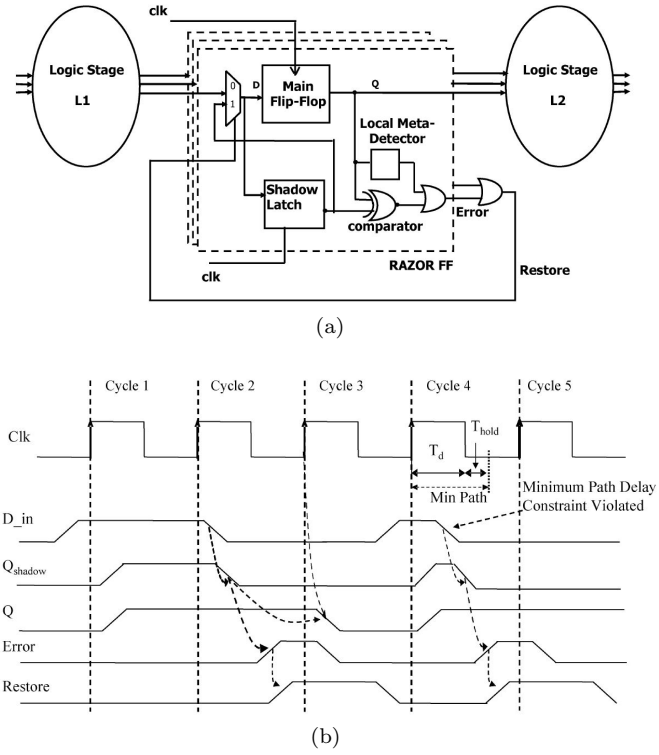


Figure 1: Conceptual representation and timing diagrams of the Razor flip-flop [8].

2.1. Razor

Fig. 1a shows a block diagram of a Razor flip-flop (henceforth referred to as RFF). In addition to the standard edge triggered D-flip-flop (DFF), the RFF includes a *shadow latch*. Here the main flip-flop samples the input data at the rising edge of the clock while the shadow latch is transparent throughout the high clock phase. Therefore, a signal arriving at the input pin of RFF after the rising edge of the clock (cycle 2 in Fig. 1b) causes different values to be captured by the main flip-flop and the shadow latch. This raises the *error* signal which then initiates the recovery mechanism. Note that the shadow latch always captures the correct value as long as the input data arrives no later than the falling clock edge. For this reason,

the high clock phase is referred to as the *detection*, *sampling* or *speculation window* of the RFF. The length of the detection window determines the maximum allowable delay of the incoming timing paths, and consequently the maximum amount of increase in frequency or decrease in supply voltage that the RFF can tolerate. The maximum path delay constraint is therefore defined as [3]

$$T_{max} \leq T_{cycle} + T_w - T_{setup} \quad (1)$$

where T_{max} is the maximum path delay, T_{cycle} is the clock cycle time, T_w is the detection window length, and T_{setup} is the setup time of the shadow latch. Eq. 1 illustrates how a Razor flip-flop *relaxes* the maximum path delay constraint of a conventional DFF by the amount equal to the length of the detection window.

But, what if the input signal toggles during the high clock phase not due to a *slow* path from last cycle but because of a *fast* path in the current cycle? The RFF has no way of differentiating between these two cases and would indicate a false error if the former occurs. This is illustrated in Fig. 1b where the input signal toggles too soon during the high clock phase of cycle 4. In fact, the length of the detection window lies at the heart of a fundamental trade-off in the design of EDS circuits. While a wide detection window is desirable to achieve maximum throughput and energy improvement, its length is limited by the minimum delay of fast paths. In order to eliminate the possibility of fast paths incurring false errors, Razor requires all paths to have best-case delays larger than the length of the detection window plus the hold time of the shadow latch. The minimum path delay constraint is therefore written as [3]

$$T_{min} \geq T_w + T_{hold} \quad (2)$$

where T_{min} is the minimum path delay, T_w is the detection window length, and T_{hold} is the hold time of the shadow latch. This constraint is met by inserting delay buffers in fast paths to increase their propagation delay. This incurs a power overhead to the design, working against the original goal of reducing it. Hence, the length of the detection window should be configured to maximize power savings through voltage reduction while minimizing the power overhead from the insertion of delay buffers.

Eq. 2 illustrates two important facts about a Razor flip-flop. First, RFF *restricts* the minimum path delay constraint of a conventional DFF by the amount equal to the length of the detection window. This is in contrast to the maximum path delay constraint of RFF which is *relaxed* by the same amount (See Eq. 1). The maximum and minimum path delay constraints of an RFF are, therefore, linked together by the detection window. Energy efficiency improvements which are achieved by relaxing the maximum path delay constraint come at the cost of additional restriction on the minimum path delay constraint, which is met by adding delay buffers leading to diminishing energy efficiency gains. Second, the minimum path delay constraint only limits the maximum length of the high clock phase and imposes no restrictions on the clock frequency. Hence, the operating frequency can be chosen arbitrarily while the duty cycle of the clock is tuned for the minimum path delay constraint to be met.

Another important issue in the design of EDS circuits arises from the fact that by allowing the input signal to arrive after the rising edge of the clock, setup and hold time constraints of the main flip-flop are not respected. Therefore, if the input signal is only slightly late and toggles "too close" to the rising clock edge, there is a possibility that the main flip-flop becomes metastable. Razor handles this issue by taking two measures. First, a local metastability detector is placed at the output of the main flip-flop and the RFF raises its error signal in the case it becomes metastable. This ensures that metastability-causing errors do not go undetected while, again, incurring additional energy overhead. Second, Razor requires at least two successive non-critical pipeline stages immediately before storage to eliminate the possibility of metastable signals being committed to memory. Margining two pipeline stages runs contrary to the design motivation of Razor and can limit its ability to achieve optimal energy and/or throughput improvements.

For a timing error to go undetected, the resolution time (i.e. the time it takes for the RFF to resolve its state) must satisfy two timing constraints. First, the data-path signal must become stable early enough so that a low-logic is sampled as the error signal. Therefore, the resolution time must satisfy

$$T_r < T_{cycle} - T_{error-path} - T_{setup} \quad (3)$$

Second, the additional delay incurred on the data-path by the resolution time must cause the subsequent RFF to fail its maximum path delay timing constraint. In other words

$$T_r + T_{datapath} > T_{cycle} + T_w - T_{setup} \quad (4)$$

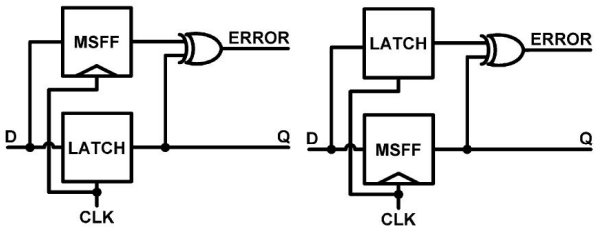
where $T_{datapath}$ is the propagation delay of the data-path. Hence, for these conditions to be met, the data-path propagation delay must approximately fall in the following range

$$T_{min} + T_{error-path} - T_{hold} < T_{datapath} < T_{max} \quad (5)$$

Since propagation delay of the error path ($T_{error-path}$) is typically a small fraction of the cycle time, a large number of data-paths would satisfy these conditions and result in undetected timing errors. As a result, it is imperative for an RFF to include a metastability detector in its data-path to prevent such an event.

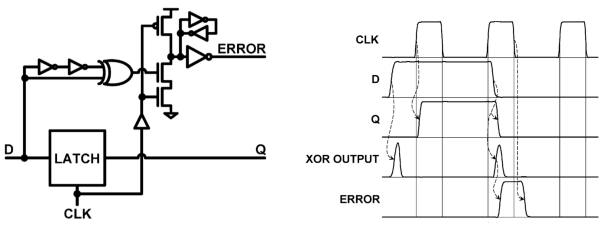
2.2. Double Sampling with Time Borrowing(DSTB)

The second EDS circuit discussed is called Double Sampling with Time Borrowing (DSTB). The design of DSTB



(a) DSTB (b) RFF

Figure 2: Conceptual representation DSTB and Razor flip-flop [3].



(a) (b)

Figure 3: Conceptual representation and timing diagrams of the TDTB EDS Circuit [3].

is very similar to that of the RFF. Fig. 2a shows a conceptual view of a DSTB next to that of an RFF in Fig. 2b. In DSTB, relative to RFF, the main flip-flop and the shadow latch have swapped places. Recall that since the setup and hold time constraints of the main flip-flop is not respected in RFF, there is always the possibility of it becoming metastable. The shadow latch, in contrast, would never be metastable as long as the maximum path delay constraint of the RFF (Eq. 1) is satisfied. In RFF, the problematic DFF feeds both the data path and the error path. Hence, signals on both paths can become metastable.

As discussed in section 2.1, Razor requires a metastability detector to prevent undetected errors as well as two successive non-critical pipeline stages immediately before the memory to ensure that metastable signals do not corrupt the state of the processor, requirements that limit the benefits of timing speculation offered by Razor. In contrast, in DSTB, the data-path is driven by the shadow latch and the metastability issue is limited to the error path. An analysis of the metastability in DSTB (See [3] for more details) reveals that limiting metastability to error path causes the *Mean Time Between Failures* (MTBF) to be orders of magnitude larger for DSTB than for RFF, to the point that the metastability detector can be safely omitted from the EDS circuit, reducing the energy overhead of timing speculation.

2.3. Transition Detector with Time Borrowing (TDTB)

The next EDS circuit is called Transition Detection with Time Borrowing (TDTB) [3]. Fig. 3a and 3b show the circuit level implementation and sample timing diagrams of the TDTB EDS circuit. In contrast to previous EDS circuits, TDTB implements a *dynamic* error detection scheme where timing errors are identified not by delayed resampling, but by dynamic monitoring of the input data using a transition detector. The XOR gate continuously compares the input data with its delayed version and produces a pulse when they are not equal, effectively detecting input data transitions. During the low phase of the clock, the *error* signal is disconnected from the transition detector and is driven by the top transistor to a logic low. As the data-path latch is opaque during this period, the EDS circuit behaves like a conventional flip-flop. However, if a late-arriving signal causes the input data to transition during the high clock phase, the *error* signal transitions to logic high and remains in this state until the falling clock edge brings it to a logic low again. Similar to DSTB EDS circuit, while time borrowing is potentially enabled by employing a level-sensitive latch in the data-path, it is suppressed by the activated *error* signal, effectively enforcing conventional edge-triggered flip-flop operation.

The above analysis shows that a TDTB design implements the same functionality as double sampling EDS circuits, including limiting the metastability issue to the error path. At the same time, TDTB improves the performance and energy efficiency of DSTB EDS circuits in the following ways. First, both size and clock energy of TDTB is significantly smaller than that of DSTB, and even the conventional master-slave flip-flop. Granted, conventional master-slave flip-flops can always be replaced with pulse-latches at the expense of additional design complexity to achieve lower clock energy. Second, both the propagation delay (CLK-to-Q) and setup time (defined as the minimum D-to-CLK delay prior to the rising clock edge such that an error signal is not generated) are improved in a TDTB EDS circuit, resulting in potential performance improvements. These benefits, however, come at the price of increased design complexity as well as sensitivity to within-die process variation. Moreover, the issues of error path metastability and minimum path delay constraint remain in TDTB.

2.4. RazorII

The next version of Razor flip-flop is called RazorII [9]. Fig. 4a and 4b show the block diagram and sample timing diagrams of the RazorII EDS circuit. Similar to the TDTB design, RazorII uses a single latch and employs a transition detector to monitor dynamic behavior of the input signal and detect timing errors. Unlike the TDTB transition detector that directly monitors the input data pin, the transition detector in RazorII is connected to the internal latch node and essentially detects transitions on the latch output rather than the latch input. This design decision trades off some of EDS circuit timing error coverage to achieve full soft error coverage. TDTB EDS circuit has a detection window equal to the high phase of clock for both timing and soft errors. RazorII, on the other hand,

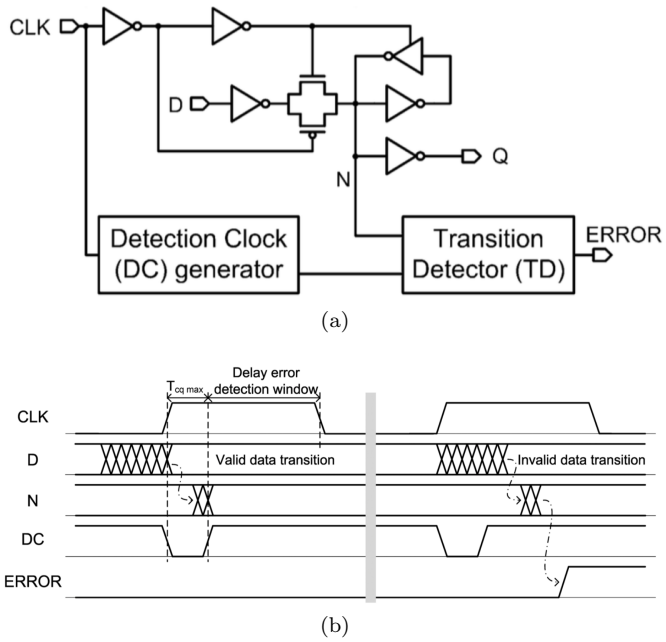


Figure 4: Conceptual representation and timing diagrams of the RazorII flip-flop [9].

can detect soft errors during the entire clock cycle [1], while its detection window for timing errors is reduced. The reason for this will be clear after an analysis of RazorII's operation which follows.

During the low phase of the clock, the latch is opaque and no transitions happen at the internal latch node denoted by N in Fig. 4a. The EDS circuit, therefore, operates similar to a conventional flip-flop during this time. A transition caused by a late-arriving signal during the high clock phase, on the other hand, toggles N as the latch is transparent at this time. The transition is detected by the transition detector and flagged as a timing error. However, even a legitimate transition at the EDS circuit input before the rising edge of the clock takes the time equal to the CLK-to-Q delay of the latch to appear at its output after the rising edge of the clock. Therefore, the transition detector must be disabled at the beginning of the clock cycle for a time greater than this delay to prevent flagging such transitions as timing errors. This is accomplished by the detection clock generator block that produces a negative pulse which deactivates the transition detector. In order to ensure correct operation, the length of this pulse must be guaranteed to be greater than the CLK-to-Q delay of the latch across all PVT corners. Hence, it is required that the *minimum* width of the negative pulse is greater than the *maximum* CLK-to-Q delay of the latch. While post-manufacturing tuning of the detection clock pulse width can be used to account for process variation, the width of the pulse should still be suitably margined. This causes the detection pulse to be longer than the CLK-to-Q delay

[1] Analysis of soft error tolerance is beyond the scope of this paper. See [9] for details.

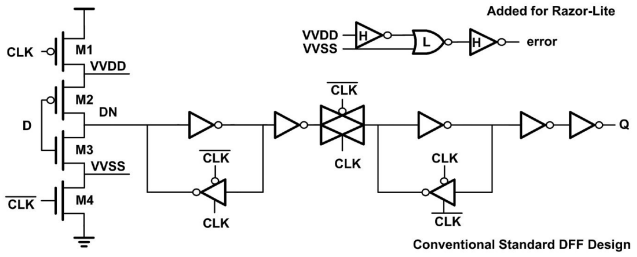


Figure 5: Circuit level implementation of a conventional flip-flop and added Razor-Lite EDS circuit [18].

of the latch. As a result, the transition detector remains disabled for an additional period equal to the difference between the pulse width and CLK-to-Q delay. During this time, transitions on the internal latch node are not detected and no timing errors are declared, thereby reducing the detection window width of the EDS circuit. If the adjoining path in the next pipeline stage has ample timing slack, correct operation is maintained through time borrowing, resulting in even more performance improvements. However, there is always the possibility of system failure if multiple stage time borrowing is accumulated beyond the ability of RazorII flip-flops error tolerance. Complex design-time timing analysis of time-borrowing is, therefore, required to guarantee correct system operation.

Similar to DSTB and TDTB EDS circuits, RazorII successfully limits the metastability issue to the error path while the minimum path delay constraint problem remains. It accomplishes improved energy efficiency compared to double sampling EDS circuits at the expense of additional design complexity and sensitivity to process variation. Compared to TDTB, soft error coverage is extended to the entire clock cycle, but timing error detection window is shortened and sensitivity to process variation is increased.

2.5. Razor-Lite

All EDS circuits discussed so far implement error detection capabilities at the expense of, among other things, significantly increasing the energy consumption of conventional sequential circuits. Razor-Lite [18] is a more recent EDS circuit designed to mitigate this problem. Fig. 5 shows circuit level implementation of a conventional flip-flop along with the error detection circuit added by Razor-Lite. The detection circuit uses two internal nodes of the flip-flop's input buffer, called *virtual rails* and denoted by VVDD and VVSS in Fig. 5, to detect timing errors. To understand how this is accomplished, it is useful to analyze how these two nodes (VVDD and VVSS) behave during the absence and presence of timing errors.

During the low phase of the clock, M1 and M4 are both on and VVSS and VVDD are driven (by ground and VDD) to logic-low and logic-high, respectively. DN is connected to one of the virtual rails based on the value of the input data D by turning on either M2 or M3. During this time, transitions on D change the value of DN but

VVDD and VVSS remain constant. At the rising edge of the clock, both M1 and M4 are turned off and the virtual rails start floating. Note, however, that either VVDD or VVSS remains connected to DN through the M2 or M3. If no timing errors occur and D does not transition during the high clock phase, VVDD and VVSS remain high and low, respectively. However, a transition on D caused by a late-arriving signal changes the states of M2 and M3, disconnecting the previously connected virtual rail from and connecting the other one to DN. Since the newly connected virtual rails has the opposite state as DN, the feedback inverter of the master latch pull it toward the value of DN. In other words, an input data transition during the high clock phase causes either a low-to-high transition on VVSS or a high-to-low transition on VVDD. Such transitions do not occur in the absence of timing errors, and can, therefore, be used to detect timing errors.

The detection circuit consist of two high-skewed inverters and a low-skewed OR gate. Skewed logic is used to take advantage of the unidirectionality of the transitions to speed up the error path. An important property of these error-indicating transitions in Razor-Lite is that they never occur during error-free operation, neither in the low nor in the high phase of the clock. This is in contrast to the error-indicating transitions monitored by previous EDS circuits with transition detection that indicated timing errors only during the detection window. As a result, unlike previous transition detectors that needed to employ the clock signal, Razor-Lite EDS circuit implements a static transition detector, incurring no additional clock load. Extra data-path loading is also avoided. As a bonus, setup and hold times of the circuit are slightly reduced. Energy overhead is consequently limited to less than 3% as only 8 transistors are used to implement error detection. As a point of comparison, the most light-weight previous EDS circuit, TDTB, uses 15 transistors and incurs around 10% of energy overhead for error detection while also incurring extra clock and data-path loadings that add larger overheads to the performance (CLK-to-Q) of the EDS circuit.

In addition to the energy-efficiency of the EDS circuit, Razor-Lite is inherently more resilient to metastability. While the possibility of metastability in data-path is present similar to Razor, the use of skewed logic for error detection ensures with a high level of confidence that long-term metastable events are identified as timing errors. Short-term metastable events are either detected as timing errors (if they resolve to the correct state) or add a small delay to the CLK-to-Q delay of the EDS circuit. It is reasonable to assume that this small additional propagation delay can be absorbed by the next pipeline stage.

Finally, Razor-Lite mitigates the problem of minimum path delay constraints by implementing a duty cycle controller to dynamically adjust the duty cycle of the clock. An algorithm for initial and run-time calibration of duty cycle is proposed that minimizes false error detections due to fast paths. Initial calibration is performed at half-frequency and by reducing the duty cycle from 50% until no fast path errors are detected. At run-time, duty cycle is tuned during error recovery where again the processor works at half-frequency and replays the errant instruction. If another timing error is detected during this time, both errors are assumed to be the result of a fast path violation and the duty cycle is reduced to suppress them. Duty cycle is increased when too many slow path violations (i.e. true timing errors) are detected. This strategy mitigates the energy overhead incurred by the delay buffers added to fast paths to satisfy minimum path delay constraints of EDS circuits and further improves the energy efficiency of Razor-Lite.

3. Error Recovery

When a timing error is detected by EDS circuits, they raise an *error* signal that initiates error recovery. The error recovery logic is responsible for restoring the correct system state by (i) preventing the propagation of the error and (ii) maintaining the correct order of instruction executions. Recovery penalty is defined as the average additional time spent for an errant instruction to correctly finish execution compared with the error-free case. A large recovery penalty significantly limits the potential energy/performance improvements that timing speculation can achieve. Various error recovery mechanisms can be characterized, among other things, by how they choose to balance the trade off between the recovery penalty and the energy overheads of EDS circuits and the recovery logic. Other desirable properties include manageable design complexity including architecture independence, unintrusiveness, and automatic design and analysis using CAD tools.

The error recovery techniques discussed in this section take one of the following approaches to accomplish these tasks.

Local Error Correction: This recovery technique relies on the ability of the error detection circuit to correct its own state. Therefore, this approach cannot be realized using the EDS circuits discussed in section 2 except first Razor EDS circuit reviewed in section 2.1 which implements local error correction. The techniques discussed in sections 3.1 and 3.2 are of this type.

Instruction Replay: It is not strictly necessary to correct the timing error as long as it is prevented from corrupting the architectural state of the processor. A number of recovery techniques take advantage of this observation to enable the use of simpler EDS circuits that have smaller energy overheads. The errant instruction is simply replayed until it completes without experiencing errors. This, of course, results in a larger recovery penalty. These methods are discussed in section 3.3.

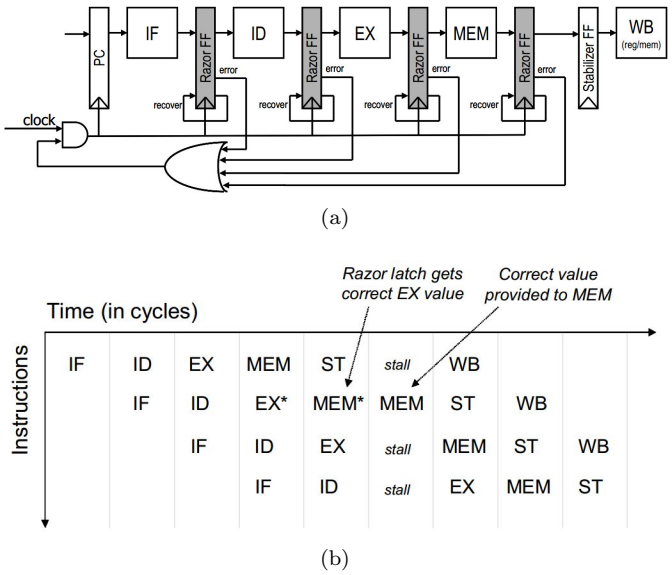


Figure 6: Microarchitecture design of clock gating and pipeline timing during error recovery [12].

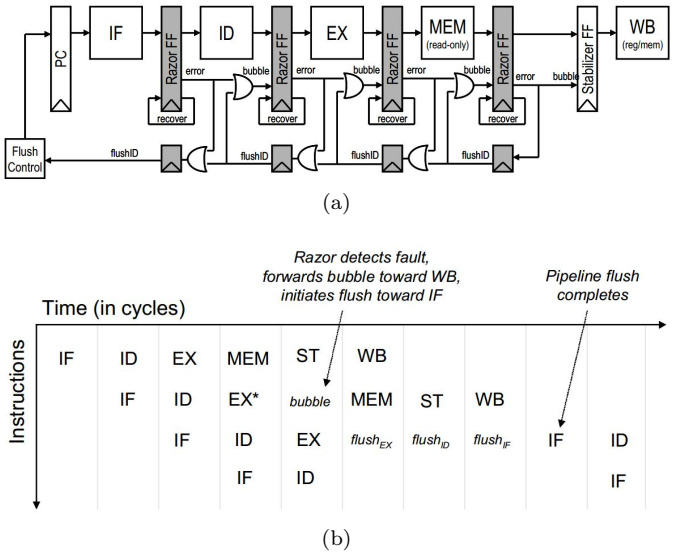


Figure 7: Microarchitecture design of counterflow pipelining and pipeline timing during error recovery [8], [12].

Error Masking: Data corruption can be avoided by using EDS circuits that have a latch in their data-path, by taking advantage of their ability to mask timing errors using time borrowing. This approach incurs some additional design complexity but achieves a smaller recovery penalty. Techniques discussed in sections 3.4 and 3.5 take this approach.

3.1. Clock Gating

Fig. 6a and 6b show the microarchitecture design of clock gating and a timing diagram of the pipeline during the recovery of a timing error in the EX stage. This technique relies on the capability of the EDS circuits to perform *local* or *in situ* error correction. The Razor EDS circuit reviewed in section 2.1 (RFF) implements this functionality. Error signals of all RFFs are simply OR-ed together and produce a *recover* signal which is used as the global clock gating control signal. When a timing error is detected, the entire pipeline is stalled for one cycle during which all RFFs reload their main flip-flops with the value in their shadow latches. Normal execution continues with the next clock cycle. The recovery penalty is only one clock cycle, even when multiple errors occur during one clock cycle.

Other than reliance on an EDS circuit design with local error correction capability, this scheme meets all design goals of error recovery. Recovery penalty is only one clock cycle and the recovery logic incurs small area and energy overheads. Moreover, it can be implemented independent of the processor architecture using existing CAD tools. The main drawback arises from the strict requirement on the error path delay. For correct operation, the timing error must be detected, translated into the recover signal, and routed to all flip-flops, in less than a clock cycle. This is impractical in high-performance processors where the chip-wide wire delays alone are more than one clock cycle.

3.2. Counterflow Pipelining

Fig. 7a and 7b show the microarchitecture design of counterflow pipelining and a timing diagram of the pipeline during the recovery of a timing error in the EX stage. Similar to clock gating, this technique relies on EDS circuits with local error correction capability. Using the counterflow pipelining concept [26] of instruction results (error signals in this case) moving backwards in the pipeline, this technique eliminates the problematic timing constraint of the error path in clock gating at the expense of increasing the recovery penalty.

The detection of a timing error initiates two actions. First, a *bubble* is sent to downstream pipeline stages nullifying the computation in the stage following the errant stage. This prevents the propagation of the error. Second, a *flush* train carrying the stage identifier of the failing stage is launched backwards. The flush train inserts a bubble as it passes through upstream stages nullifying the succeeding instructions. The errant instruction is corrected by the EDS circuits in the following clock cycle and continues execution. Once the flush train reaches the start of the pipeline, execution at the instruction following the errant one. The recovery penalty consists of the time for the flush train to reach the start of the pipeline and the time for the re-executed instructions to return to their corresponding stages before the timing error. Counterflow pipelining, therefore, incurs a recovery penalty of $2N$ cycles where N is the maximum (in case of multiple errors) depth of the errant stages in the pipeline. This value ranges from 2 to $2S$ with an average value of $S+1$ (assuming uniform error probabilities across the stages), where S is the number of pipeline stages.

Counterflow pipelining remains reliant on the error correction ability of the EDS circuits while it successfully eliminates the error path delay constraint of the clock gating scheme at the expense of increased recovery penalty, area and energy overheads, and design complexity.

3.3. Instruction Replay

The design of instruction replay error recovery techniques is motivated by the observation that energy and/or performance gains from aggressive voltage scaling or overclocking is mitigated by the exponential increase in the rate at which timing errors occur. Therefore, the processor should be operated near the first point of failure where error rates are low. At this point, the energy of the EDS circuits and recovery logic rather than the recovery penalty is the main contributor to the energy/performance overhead of timing speculation. As a result, the recovery mechanism is designed to trade off the recovery penalty for the energy overhead of EDS circuits and recovery logic. This is accomplished in two steps. First, no error correction mechanism is implemented, neither by the EDS circuit nor the microarchitecture. Error signals are simply passed along the pipeline and serve in the write-back stage as write enable controls for the register file and memory to prevent the corruption of the architectural state. Second, the recovery mechanism is very similar to the logic already present in most processors for branch miss-prediction and can share resources with it to decrease the energy overhead.

In order to ensure that replaying instructions resolves the timing errors, one of the following approaches, or a combination of them is taken. For a timing error to occur on a path, the clock cycle should be smaller than its propagation delay and transitions at the start of the clock cycle should sensitize the path (i.e. toggle all its nets). The two approaches correspond to these two requirements and seek to reduce or eliminate their probability.

Slow Execution: In this approach, clock frequency is substantially reduced (typically halved) such that the clock cycle is longer than worst case delay of all the paths, thereby eliminating the possibility of timing errors. Minimum path delay constraints are guaranteed to be satisfied by maintaining the high phase delay of the clock. In addition to doubling the penalty for flushing the pipeline from S cycles to $2S$ cycles where S is the number of stages in the pipeline, this approach incurs an instruction replay penalty of $2S$ cycles.

Path Sensitization Reduction: This approach reduces, and ultimately eliminates, the probability of the failing paths being sensitized. It is accomplished by N back-to-back executions of the errant instruction. In a pipeline with S stages, the probability of failing paths being sensitized by the Nth execution reduces as N is increased and reaches zero for $N = S + 1$. The first $N - 1$ executions set register input values without changing the architectural state and the last execution is the only valid one. Each increment of N sets more register input values and reduces path sensitization probabilities, eventually preventing all signal transitions at $N = S+1$. While the worst-case total recovery penalty of this approach which equals $3S$ cycles ($2S$ cycles for instruction replay plus S cycles for flushing the pipeline) is smaller than the penalty of halving the frequency which equals $4S$ cycles ($2S$ cycles for instruction replay plus $2S$ cycles for flushing the pipeline), careful selection of N can further reduce the total penalty to $2S+N-1$. If the selected N is too small, however, the effective recovery penalty would be larger than the first approach as the entire recovery mechanism must be repeated to ensure correct execution.

A RazorII implementation [9] uses a combination of the above techniques to recover from a timing error as follows. First, the recovery mechanism is initiated by the error signals. Next, the entire pipeline is flushed. Then, the errant instruction is replayed for a maximum of N_{max} times, called the replay limit. If the error persists through all the replays, a final replay is issued at half frequency to guarantee correct execution. Experimental results indicate that around 60% of the errant instructions do not require frequency reduction when $N_{max} = 2$. With $N_{max} = 2$, errant instructions are replayed once at the current frequency and, in case of a repeated error, once at the half frequency.

An Intel research processor [4] implements two slightly different policies. The first one, called instruction replay at half frequency, essentially implements the above technique for $N_{max} = 1$. This policy has also been implemented in an ARM processor using RazorII EDS circuits [5]. The second policy proposed in [4] does not reduce the frequency and relies on the second approach only. First the errant instruction is replayed N times with only the Nth execution allowed to change the architectural state. If the error persists even in the Nth execution, the instruction is replayed with $N = S + 1$ to ensure correct execution.

Selecting the optimal policy requires a comprehensive analysis of the system's error behavior while the effect of the typical workload is taken into account. This highlights the need for fast simulation/emulation platforms that enable extensive design space exploration.

3.4. Bubble Razor

With the exception of clock gating which is architecture-independent, all the recovery mechanisms discussed so far must be implemented during the design of the microarchitecture at the register transfer level. This increased design complexity substantially limits their scalability to large high-performance processors. Bubble Razor [14] proposes a distributed and architecture-independent error recovery mechanism to provide improved scalability. The

basic idea is to convert a conventional flip-flop based design into a two-phase latch based design. The flip-flops are broken into their constituent master and slave latches and the master latch is moved to backwards in the data-path to achieve a new balanced pipeline with twice the number of stages as the original pipeline. Any of the EDS circuits discussed in section 2 with a latch in its data-path can be used. A latch clustering scheme is used to mitigate the overhead incurred by the additional latches.

A key property of the new two-phase latch based design is that consecutive latches operate out of phase. In other words, when a latch is transparent, all its neighbors are opaque and vice versa. This property is extremely beneficial to a timing speculative system by producing two main consequences. First, it restores the minimum path delay constraints of the design to their conventional forms by breaking their link to the detection window length and minimum path delay constraints. Recall that the minimum path delay constraints of EDS circuits is substantially more restricted than conventional flip-flops due to the requirement of differentiating between slow paths from the last clock cycle and fast paths from the current clock cycle. This problem does not exist in a two-phase latch based pipeline. An input signal arriving during the transparency of period of a latch is guaranteed to be a slow path from the last clock cycle because the neighboring latches are closed during this time and do not launch new signals. Recall that the minimum path delay constraints of EDS circuits substantially limit the energy and performance benefits of timing speculation. Bubble Razor eliminates this limitation.

Second, since latches operate out of phase, they can be stalled one after the other without losing data. In flip-flop based pipelines, flip-flops must be stalled simultaneously to avoid the loss of data as launching and capturing operations are performed at the same time. A two-phase latch based pipeline, on the other hand, interleaves the launch and capture operations. Therefore, when a latch stalls, data is not launched towards it for a period of one clock phase. This time difference can be used to communicate the stall signal to all neighbors, causing them to stall one clock phase later as necessary. Since neighbors are by definition less than one clock phase apart (otherwise normal data communication would have been impossible), stall signals are guaranteed to reach their destinations in time. Therefore, it is possible and practical to implement a scalable clock gating scheme to achieve a one-cycle error recovery penalty.

Fig. 8 illustrates the error recovery mechanism of Bubble Razor. Once a timing error is detected by a latch, it communicates the error to the next stage, causing it to stall by skipping its next transparent phase. This provides additional time for the late-arriving signal to reach the next latch. This mechanism essentially implements a *controlled* time borrowing scheme where the amount of borrowed time is always equal to one clock cycle. While a latch based design such as the one used by Bubble Razor can mask timing errors by *continuous* time borrowing without paying any recovery penalty, a failure is possible if the time borrowing compounds through multiple consecutive failing stages. The discrete time borrowing scheme implemented by Bubble Razor avoids the complex analysis required for verification against this effect at the expense of paying a one cycle recovery penalty for all timing errors including the ones that would not induce failures. Going back to the recovery mechanism, when a stage receives an error signal, it starts the bubbling process by sending bubbles to all its neighbors. A latch receiving bubbles from one or more of its neighbors stalls and sends bubbles to the neighbors it did not receive bubbles from.

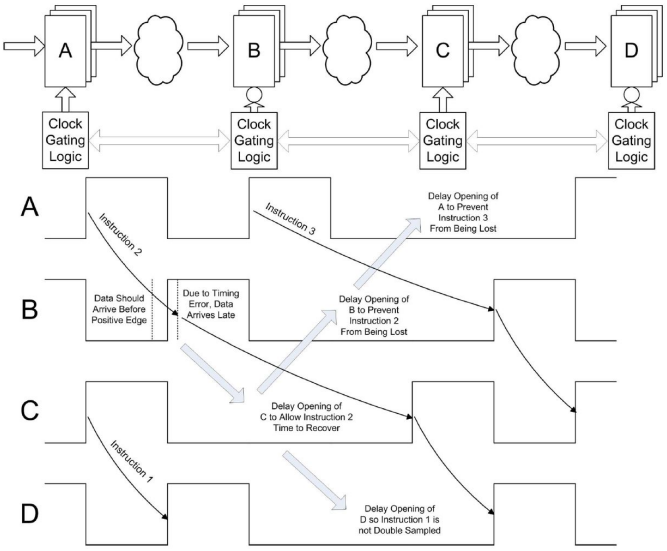


Figure 8: Two-phase latch based pipeline used for error recovery in Bubble Razor [14].

This recovery mechanism (with a small modification not discussed here) guarantees error recovery and forward progress even in the face of multiple timing errors by paying a constant one-cycle recovery penalty. This comes at the expense of an increased number of latches compared to a conventional latch-based design that is required with instruction replay recovery scheme. The increased energy overhead incurred by these extra latches is compensated by the low-overhead recovery mechanism and the elimination of large energy overheads as a result of more relaxed minimum path delay constraints.

3.5. TIMBER

TIMBER [7] introduced a pair of new EDS circuits (TIMBER flip-flop and TIMBER latch) based on the concept of double sampling. Unlike other EDS circuits that are designed to be accompanied by some error correction scheme to recover from errors, TIMBER EDS circuits enable a different recovery mechanism based on *time borrowing* and *error relaying*. Design of TIMBER is motivated by two observations. First, as frequency is increased (or voltage is decreased) past the point of first failure, the rate

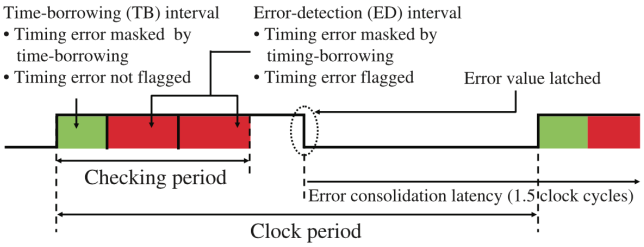


Figure 9: Conceptual representation TIMBER operation [7].

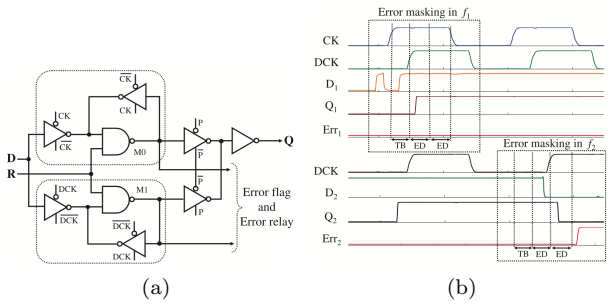


Figure 10: Circuit level implementation and timing diagrams of the TIMBER flip-flop [7].

at which timing errors occur increases exponentially, and benefits of aggressive overclocking (or voltage scaling) are exceeded by the large cost of error recovery. As a result, a large fraction of wide detection windows remain unused by the error-rate-aware DVFS operation of the processor. Accordingly, TIMBER is designed with a narrow detection window. Second, while a processor may have a large number of critical paths, only a small fraction them are connected together by flip-flops. In other words, most critical paths are preceded and followed by non-critical paths. The narrow detection window of TIMBER allows it to *mask* timing errors in a pipeline stage by borrowing time from the next stage. Indeed, this comes at the expense of large recovery costs for multiple-stage timing errors (i.e. errors spanning multiple successive pipeline stages).

Fig. 9 illustrates design concept of TIMBER. Checking period is the time after the rising clock edge during which possible late signals may arrive at the EDS circuits. This period is divided into a time borrowing and two error detection intervals.The length of the time borrowing interval determines the amount of overclocking allowed such that a single-stage timing error is guaranteed to arrive during the time borrowing period. This timing error is masked by borrowing the time borrowing interval of the next pipeline stage. An error signal is relayed to the next stage so that incoming signal is sampled later, but no errors are flagged to the central control unit. If a timing error occurs in the next stage as well (i.e. a two-stage error), the signal to the endpoint of the second path is similarly guaranteed to arrive in the first error detection window. This error is similarly masked by borrowing the first error detection interval of the next stage. The number of error detection intervals determines the number of additional successive errors after the first timing error that can be tolerated. In order to prevent borrowed times from accumulating, in addition to the error relay signal, an error flag is raised after detection of a timing error in the first error detection interval that causes a reduction in clock frequency. Because no error correction mechanism is present, the frequency must be reduced to a safe level so that a non-maskable error does not occur. The second error detection interval ensures that a possible additional error in the next stage can also be masked to account for the latency in error consolidation and frequency reduction.

Fig. 10a and shows the circuit level implementation of the TIMBER flip-flop. Design concepts of the TIMBER latch is similar and are not discussed here (see [7] for details). Latches M0 and M1 take turns in driving the inputs to the next stage by setting signal P such that inputs are driven by M0 during the time borrowing interval and by M1 during the rest of the clock cycle. Each TIMBER flip-flop includes additional logic (not shown) for producing the delayed clock (DCK) and relaying error signals to the next stage. Incoming error relay signals are used to generate DCK by delaying the clock signal appropriately. If the previous stage has not relayed error signals, the delay is equal to the length of time borrowing interval. It is increased by one or two error detection interval lengths if one or two previous stages have experienced errors. Fig. 10b the timing diagrams for a two-stage error occurring at flip-flops f_1 and f_2. M0 samples the input data on the rising edge of the clock and drives next stage input signals during the time borrowing interval. M1 samples the input data again on the rising edge of the delayed clock, guaranteeing that the correct value is stored. If no timing error occurs, M1 starts driving next stage input signals, starting at the end of the time borrowing period for the rest of the clock cycle, with the same value as in M0 and the EDS circuit operates similar to a conventional flip-flop. In case of a timing error, the correct value is launched a time borrowing interval late. An error signal is generated at the falling edge of the clock and relayed to the next stage flip-flops to inform them of the late-arriving signal.

Minimum path delay constraints are improved in TIMBER due to the smaller detection window. However, metastability remains an issue and a metastability detector must monitor the output of M0 to detect metastable signals in case of a multiple-stage timing error. Since correct operation relies on accurate generation of the delayed clock, guardbands are required for the clock control logic. Moreover, the complex design of this EDS circuit suffers from high sensitivity to process variation. While TIMBER flip-flop incurs a large energy overhead compared to previous EDS circuits, accurate comparison of their relative efficiency requires detailed information about single as well as multiple-stage error rates.

4. Performance Modeling

As the efficiency of timing speculative systems such as the ones reviewed in this paper improves and researchers consider them as viable alternatives to traditional always-correct systems, development of performance models for these systems becomes more important. In order to perform effective design space exploration as well as reliable accuracy evaluation, efforts at various levels of the hardware/software stack directed towards improving the performance of timing speculative systems need platforms that can emulate the behavior of the underlying system. However, development of performance models for timing speculative systems is significantly more complicated than conventional error-free systems. In this section, we review a timing model and its implementation as a simulator for analysis of software error behavior in presence of process variation.

4.1. Related Work

The need for fast and accurate timing models for estimation of the system's error behavior is most evident in the 'Experimental Results' sections of papers describing variability management techniques of erroneous systems (i.e. systems that allow timing errors) where researchers face the long simulation times of variability effects at the architecture level [20][24][30][17][25]. The lack of fast simulation platforms for these systems is becoming a bottleneck and a key challenge for erroneous systems research [16], [30]. As a result, most works limit their analysis in both time (analyzing only small parts of the application software) and space (analyzing only a few components of the system), adversely affecting optimization opportunities and/or accuracy of evaluation. The following are some examples of variability management techniques in which the analysis of dynamic error behavior has been limited. Trifecta [20] focuses only on the ALU adder and selection logic. Roy and Chakraborty [24] only consider ALU errors and limit their simulations to 100 instructions that most frequently exercise the ALU. Xin and Joseph [30] focus on ALU, LSU, and Shift/Branch units. Hoang et al. [17] reduce the size of benchmark input sets due to extremely long run- times associated with gate-level simulations. Similarly, Sartori and Kumar [25] only study the LSU and IQ and do not simulate the benchmarks for their entire duration.

At the same time, efforts specifically directed at development of error models have also been limited. Rehman et al. [23] assume that the area of functional units determine their error rate which, while acceptable in case of soft errors, is not expendable to errors caused by variability. Rahimi et al. propose Instruction [21] and Sequence [22] Level Vulnerability to predict the error probabilities, but their models do not specifically take input data dependence into account. A number of recent works [24][30] have proposed to use the Program Counter (PC) to predict errors, suggesting that various dynamic instances of an instruction behave similarly due to locality of input data. Such techniques, even if effective, do not result in timing models that can be used at various phases of the design of these systems. Finally, an emulation testbed for variability-aware software called VarEMU is presented in [29]. While VarEMU provides a fast and easy-to-use platform for implementation of error models and may be useful for variability-aware software power analyses, it cannot handle the cycle-by-cycle nature of variability in causing errors due to the functional (and not cycle-true) operation of the underlying virtual machine.

4.2. Clustered Timing Model

The timing model presented in this section [1] maps the low-level effects of process variation and path sensitization onto the level of microarchitecture. The result is then used in a microarchitecture simulator to enable the analysis of error behavior as a function of the running code. A processor's Clustered Timing Model (CTM) estimates pipeline-stage-level instruction error probabilities by (i) functionality-based clustering of its timing paths and (ii) using architecture level information at simulation/run time.

A CTM consists of a set of *Register Clusters (RC)* and a set of *hyperpaths* connecting them. Fig. 11 shows a depiction of a CTM for LEON3 [15], an in-order RISC processor. RCs are clusters of registers (i.e. flip-flops or input/output pins) whose logic levels at each clock cycle constitute a *value* for the RC. A hyperpath represents a collection of timing paths that connect its *origin* and *destination* RC registers together. A random variable called *delay* is associated to a hyperpath representing the *effective delay* of its constructing timing paths. The effective delay of a set of timing paths combines their delay and functionality and is defined as the maximum of the delays of the subset of paths that are sensitized. A hyperpaths' delay is, therefore, a function of the current and previous values of its origin RC. By utilizing architecture-level information of the processor to determine hyperpaths used by an instruction, the problem of estimating error probability of instructions can be reduced to the problem of modeling this functionality, mapping value transitions of a hyperpath origin RC to its delay.

While we defined a hyperpath as an object that abstracts the timing characteristics of its timing paths, it can equivalently be viewed as a collection of *functional paths*. The correlation among timing paths is abstracted into correlations among functional paths and the delay of a hyperpath is calculated as the maximum of the delays of its activated functional paths rather than the activated timing paths. In accordance, the set of possible transitions of the value of the origin RC is mapped into a set of *primary transitions*, such that a primary transition consists of all transitions that sensitize a specific functional path. The training step (not discussed here) involves characterization of the delays and pair-wise correlations of functional paths. The model can then estimate hyperpath delays simply by detecting primary transitions of the RCs.

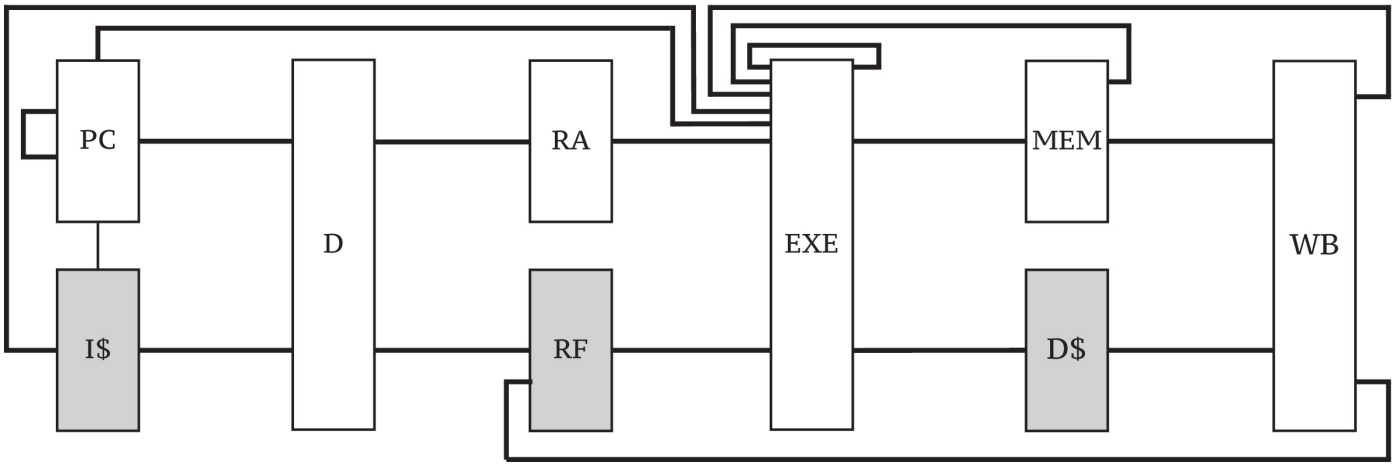


Figure 11: Clustered Timing Model for LEON3 pipeline [1].

Without going into more detail, when constructing a CTM, registers are clustered in a way that each resulting hyperpath performs a certain operation. In the case of an in-order RISC pipeline, hyperpaths can be divided, based on their operation, into four types: data transfer, addition, logic, and hybrid hyperpaths. Each type has a certain set of functional paths, chosen in a way to enable easy identification of activated functional paths from the origin RC value transition. Let us take the example of addition hyperpaths. These hyperpaths perform an addition operation on the contents of their origin RC. In LEON3 pipeline, hyperpaths EXE-PC, EXE-I$, EXE-D$, and EXE-MEM (when the active instruction in the execution stage is `add` or `sub`) are addition hyperpaths. In a multi-bit addition, a bit in the result may be changed due to either a local path activation (i.e. a 1-0 or 0-1 situation at that bit position in addition inputs) or a carry-chain activation from any lower order bit. These paths, in fact, constitute the set of functional paths of an addition hyperpath.

With these abstractions in place, [1] describes how A CTM of LEON3, a representative in-order RISC processor, is developed, verified for accuracy, and implemented in a microarchitecture-level simulator. A pipeline analysis module is added on top of the simulator to provide stage-level transition information of the running code. The timing model is implemented in a separate module which takes an instruction trace and stage-level transition information from the simulator and produces PoEs for each hyperpath every clock cycle. These hyperpath PoEs are then combined with architecture level information that determines the activated hyperpaths of instructions to derive instruction error rates at different pipeline stages. The resulting simulation platform is much faster than a timing analysis framework implemented using low-level timing models and enables the simulation of actual programs.

References

[1] Assare, O., and Gupta, R. Timing analysis of erroneous systems. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2014 International Conference on* (Oct 2014), pp. 1–10.

[2] Blaauw, D., Chopra, K., Srivastava, A., and Scheffer, L. Statistical timing analysis: From basic principles to state of the art. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 27*, 4 (April 2008), 589–607.

[3] Bowman, K., Tschanz, J., Kim, N. S., Lee, J., Wilkerson, C., Lu, S., Karnik, T., and De, V. Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *Solid-State Circuits, IEEE Journal of 44*, 1 (Jan 2009), 49–63.

[4] Bowman, K., Tschanz, J., Lu, S., Aseron, P., Khellah, M., Raychowdhury, A., Geuskens, B., Tokunaga, C., Wilkerson, C., Karnik, T., and De, V. A 45 nm resilient microprocessor core for dynamic variation tolerance. *Solid-State Circuits, IEEE Journal of 46*, 1 (Jan 2011), 194–208.

[5] Bull, D., Das, S., Shivashankar, K., Dasika, G., Flautner, K., and Blaauw, D. A power-efficient 32 bit arm processor using timing-error detection and correction for transient-error tolerance and adaptation to pvt variation. *Solid-State Circuits, IEEE Journal of 46*, 1 (Jan 2011), 18–31.

[6] Burd, T., Pering, T., Stratakos, A., and Brodersen, R. A dynamic voltage scaled microprocessor system. In *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International* (Feb 2000), pp. 294–295.

[7] Choudhury, M., Chandra, V., Aitken, R., and Mohanram, K. Time-borrowing circuit designs and hardware prototyping for timing error resilience. *Computers, IEEE Transactions on 63*, 2 (Feb 2014), 497–509.

[8] Das, S., Roberts, D., Lee, S., Pant, S., Blaauw, D., Austin, T., Flautner, K., and Mudge, T. A self-tuning dvs processor using delay-error detection and correction. *Solid-State Circuits, IEEE Journal of 41*, 4 (April 2006), 792–804.

[9] Das, S., Tokunaga, C., Pant, S., Ma, W.-H., Kalaiselvan, S., Lai, K., Bull, D., and Blaauw, D. Razorii: In situ error detection and correction for pvt and ser tolerance. *Solid-State Circuits, IEEE Journal of 44*, 1 (Jan 2009), 32–48.

[10] Dighe, S., Vangal, S., Aseron, P., Kumar, S., Jacob, T., Bowman, K., Howard, J., Tschanz, J., Erraguntla, V., Borkar, N., De, V., and Borkar, S. Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor. *Solid-State Circuits, IEEE Journal of 46*, 1 (Jan 2011), 184–193.

[11] Drake, A., Senger, R., Deogun, H., Carpenter, G., Ghiasi, S., Nguyen, T., James, N., Floyd, M., and Pokala, V. A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE*

International (Feb 2007), pp. 398–399.

[12] Ernst, D., Kim, N. S., Das, S., Pant, S., Rao, R., Pham, T., Ziesler, C., Blaauw, D., Austin, T., Flautner, K., and Mudge, T. Razor: a low-power pipeline based on circuit-level timing speculation. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on* (Dec 2003), pp. 7–18.

[13] Floyd, M., Allen-Ware, M., Rajamani, K., Brock, B., Lefurgy, C., Drake, A., Pesantez, L., Gloekler, T., Tierno, J., Bose, P., and Buyuktosunoglu, A. Introducing the adaptive energy management features of the power7 chip. *Micro, IEEE 31*, 2 (March 2011), 60–75.

[14] Fojtik, M., Fick, D., Kim, Y., Pinckney, N., Harris, D., Blaauw, D., and Sylvester, D. Bubble razor: Eliminating timing margins in an arm cortex-m3 processor in 45 nm cmos using architecturally independent error detection and correction. *Solid-State Circuits, IEEE Journal of 48*, 1 (Jan 2013), 66–81.

[15] Gaisler, A. Tsim erc32/leon simulator.

[16] Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R., Kumar, R., Mitra, S., Nicolau, A., Rosing, T., Srivastava, M., Swanson, S., and Sylvester, D. Underdesigned and opportunistic computing in presence of hardware variability. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 32*, 1 (Jan 2013), 8–23.

[17] Hoang, G., Findler, R. B., and Joseph, R. Exploring circuit timing-aware language and compilation. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2011), ASPLOS XVI, ACM, pp. 345–356.

[18] Kwon, I., Kim, S., Fick, D., Kim, M., Chen, Y.-P., and Sylvester, D. Razor-lite: A light-weight register for error detection by observing virtual supply rails. *Solid-State Circuits, IEEE Journal of 49*, 9 (Sept 2014), 2054–2066.

[19] Nakai, M., Akui, S., Seno, K., Meguro, T., Seki, T., Kondo, T., Hashiguchi, A., Kawahara, H., Kumano, K., and Shimura, M. Dynamic voltage and frequency management for a low-power embedded microprocessor. *Solid-State Circuits, IEEE Journal of 40*, 1 (Jan 2005), 28–35.

[20] Ndai, P., Rafique, N., Thottethodi, M., Ghosh, S., Bhunia, S., and Roy, K. Trifecta: A nonspeculative scheme to exploit common, data-dependent subcritical paths. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 18*, 1 (Jan 2010), 53–65.

[21] Rahimi, A., Benini, L., and Gupta, R. K. Analysis of instruction-level vulnerability to dynamic voltage and temperature variations. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012* (March 2012), pp. 1102–1105.

[22] Rahimi, A., Benini, L., and Gupta, R. K. Application-adaptive guardbanding to mitigate static and dynamic variability. *Computers, IEEE Transactions on* (2013).

[23] Rehman, S., Shafique, M., Kriebel, F., and Henkel, J. Reliable software for unreliable hardware: Embedded code generation aiming at reliability. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011 Proceedings of the 9th International Conference on* (Oct 2011), pp. 237–246.

[24] Roy, S., and Chakraborty, K. Predicting timing violations through instruction-level path sensitization analysis. In *Proceedings of the 49th Annual Design Automation Conference* (New York, NY, USA, 2012), DAC '12, ACM, pp. 1074–1081.

[25] Sartori, J., and Kumar, R. Architecting processors to allow voltage/reliability tradeoffs. In *Compilers, Architectures and Synthesis for Embedded Systems (CASES), 2011 Proceedings of the 14th International Conference on* (Oct 2011), pp. 115–124.

[26] Sproull, R., Sutherland, I., and Molnar, C. The counterflow pipeline processor architecture. *Design Test of Computers, IEEE 11*, 3 (Autumn 1994), 48–.

[27] Stackhouse, B., Bhimji, S., Bostak, C., Bradley, D., Cherkauer, B., Desai, J., Francom, E., Gowan, M., Gronowski, P., Krueger, D., Morganti, C., and Troyer, S. A 65 nm 2-billion transistor quad-core itanium processor. *Solid-State Circuits, IEEE Journal of 44*, 1 (Jan 2009), 18–31.

[28] Tschanz, J., Kim, N. S., Dighe, S., Howard, J., Ruhl, G., Vangal, S., Narendra, S., Hoskote, Y., Wilson, H., Lam, C., Shuman, M., Tokunaga, C., Somasekhar, D., Tang, S., Finan, D., Karnik, T., Borkar, N., Kurd, N., and De, V. Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International* (Feb 2007), pp. 292–604.

[29] Wanner, L., Elmalaki, S., Lai, L., Gupta, P., and Srivastava, M. Varemu: An emulation testbed for variability-aware software. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference on* (Sept 2013), pp. 1–10.

[30] Xin, J., and Joseph, R. Identifying and predicting timing-critical instructions to boost timing speculation. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, 2011), MICRO-44, ACM, pp. 128–139.