Timing Speculative SRAM

Elnaz Ebrahimi, Matthew Guthaus and Jose Renau
Dept. of Computer Engineering, University of California, Santa Cruz, Santa Cruz, CA, 95064
Email:{eebrahim,mrg,renau}@ucsc.edu

***Abstract*—Static Random Access Memories (SRAMs) are considered a major bottleneck in high performance System-on-Chip (SoC) design and there is a large demand for high performance SRAMs with minimal energy consumption. Time speculation techniques such as Razor ease timing guardbands to improve performance or reduce energy consumption. The state-of-the-art approach has high area and energy overheads due to the error detection logic.**

This study proposes a timing speculative SRAM that extends the existing Replica Bitline Column to detect read timing failures. We also extend the SRAM decode logic to protect from incorrect write operations. We demonstrate our Replica-based Timing Speculative SRAM (RTS) is an energy and area efficient design alternative to prior techniques such as Razor. Our proposed design is 22% to 58% more energy efficient in reading operations and it has an error detection mechanism which is 35% to 73% more area efficient that Razor-enabled SRAM.

I. Introduction

As fast on-chip memories continue to occupy a great portion of the area in System-on-Chip (SoC) designs, their speed and power consumption significantly impacts the system performance. Up to 60% of the total active chip area is taken by memories and 20% of that area is taken by SRAMs [11]. In SRAMs read operations are the most time consuming operations that affect the access time. As technology scales down, intra-die variations such as random dopant fluctuations affect Vth and become more pronounced in small circuits such as SRAM cells causing various read and write failures. Timing guardbands prevent such failures, while inherently requiring an over-design to meet the performance goals and an increase in the overall power consumption. Hence, there is a growing demand for architectural schemes and sensing circuits which ease the conventional design methods.

Razor is one of the most common delay-error tolerant flip-flops used in voltage management techniques. It eliminates the frequency margins needed due to PVT variations [5]. Follow up designs to the original Razor proved that error recovery costly and resorted to detecting the only timing errors [4]. Read errors have been previously detected using a Razor-based technique [7] in the boundary sense amplifier. The additional logic for error detection and correction adds significant area to the sense amplifier and, in turn, increases read energy consumption. During error recovery, Razor will add a penalty of one cycle delay to every read operation to restore the correct data. The error detection scheme is applied on a per bit basis, therefore, the final error signal is a logical *OR* of all the error signals, adding additional delay. These drawbacks limit the overall efficiency as the size of the SRAM increases [4], [5]. Razor-based SRAM is assumed to be placed in a *Razor-enabled* system, however, in case of a system error, there is no mechanism to prevent incorrect writes, consequently, there is a chance of corrupting the SRAM data.

We propose a timing error detecting scheme that utilizes existing hardware called Replica-based Timing Speculative SRAM (RTS) which can

- leverage the RBL in timing speculation due to variation,
- prevent erroneous writes,
- provide an area and power efficient error detection.

II. Related Work

Concept of Timing Speculation: Ernst et al. proposed RazorI which is a delay-error tolerant flip-flop for error detection in critical paths [5]. Razor eliminates the safety margins by achieving variable tolerance through in-situ timing error detection and correction. It uses a flip-flop and shadow latch to double sample the input. The main flip-flop is triggered at the positive edge of the clock whereas the shadow latch samples input data at the negative edge, so that the data is given 1/2 cycle to stabilize before being sampled by the shadow latch. Since the setup and hold time of the main flip-flop are allowed to be violated, a metastability detector is required at the output of the flip-flop. Razor uses a comparator to detect any discrepancy between the main and the speculative data. In case of error, the error flag is raised and OR'ed with the other "error" flags, and then propagated backwards so that the correct data on the shadow latch will be restored at the next clock cycle. Due to large number of logical OR's throughout the design, the "restore" signal can become a critical path itself. Costly correction is one of the reasons that RazorII [4] was proposed, which performs error detection and leaves the data correction to the architectural replay schemes already present in processor pipelines [4]. Razor technique has been taped out in several chips [2], [6].

Timing Speculative SRAMs Karl *et al.* use the Razor technique to protect against erroneous reads by implementing a main and a shadow sense amplifier. We will compare against their design and refer to their solution as Razor. Similar to the Razor [5] technique, Razor uses a main and a shadow sense amplifier which double sample the bitline voltage swings. During the read operation, first, the main SA is enabled and shortly after the shadow SA. A comparator checks the difference between inputs and generates an error signal. The generated "error" signal will select the shadow SA value at the output MUX [7]. The Razor will have a penalty associated with using Razor system which is a 1-cycle latency, but it will protect against data dependent delays during readings of the SRAM core.

When using Razor, the read operation failures are detected and corrected, however, when there is a write operation, the lack of write failure detection leads to an erroneous write and corrupts SRAM data unless the error is fixed before the wordline is enabled. For many SRAMs, the allotted time is less than half a cycle. A simple alternative is having a "failure" aware write scheme presented in section III.

Another example of the timing speculative SRAM is the Domino Register File design by Kulkarni *et al.* [9], which double samples read output and its delayed version to detect timing errors within a clock window. And it double samples the read output and its delayed version to detect timing errors within a clock window. Similar to Razor and Memory Timing Error Correction designs, it has an area overhead and the error detection mechanism can be defective due to metastability. Khayatzadeh *et al.* propose another Razor like SRAM that has been fabricated implementing Razor at the sense amplifier boundary to detect read errors. This is a novel idea, but the issue remains where razor is applied on a per bit basis [8] and Razor'ed sense amplifier and the final logical “OR”s will add an area overhead.

Our proposed timing speculative SRAM uses the prevalent Replica Bitline (RBL) technique implemented in many SRAM designs and avoids an area overhead. Amrutur was the first to propose the RBL technique to optimize the SAE timing signal and ensure protection against erroneous reads [1]. It is a technique best suited for SAE timing generation because of bitline delay tracking in presence of variation. RBL technique has been utilized for tunable or configurable testing approaches to minimize margins needed for generating SAE and/or WL [10], [13].

III. Time Speculative SRAM

Figure 1 shows a high-level block diagram of our proposed Timing Speculative SRAM which we refer to as RTS. The logic blocks in *green* color are the enhancements to the read and write paths: read failure detection and write failure prevention.

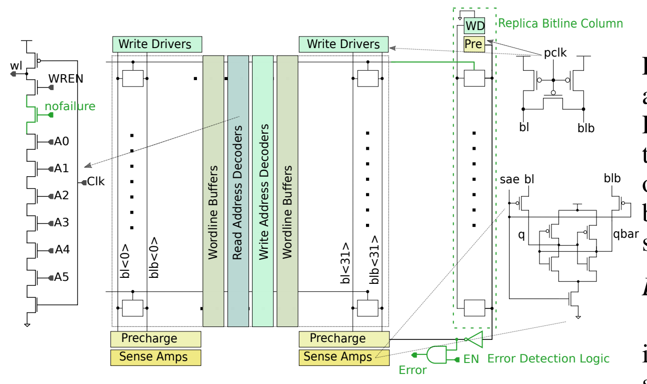

Fig. 1: Timing Speculative SRAM- 2 read and 1 write port - 1Kb

A. *Protecting from Read Time Failures*

To detect read time failures, RTS leverages the concept of RBL. The Replica Bitline Column in Figure 1 consists of a column of 6T bitcells that are identical to the bitcells used in the SRAM core, however it is controlled differently than a conventional RBL. The RBL bitcells are connected to the SRAM read and write wordlines, which activate them instead of connecting to dummy drivers and wordlines. However, we use dummy write drivers to produce a '0' bit that is written in all the RBL bitcells. Replica Bitline Column uses the same type of precharge circuit as the SRAM core. During the precharge period, when clock is '0', the RBL bitlines will be precharged to '1'. When the clock signal transitions to '1', the wordline is raised, the reading begins and one of the RBL bitcells will be discharged mimicking the SRAM bitcell discharging behavior. The time it takes to discharge the ‘1’ in Replica cells is used to predict the SAE signal as it generates sufficient delay for enabling the sense amplifier. The output of the Replica Bitline Column is connected to one large inverting buffer, shown in Figure 1, for 2 reasons. First, the RBL output signal needs to be inverted and able to drive all the 32 sense amplifiers and second, we need to compensate for the wordline enabling delay.

The error detection logic is shown in Figure 1. It includes another input signal besides RBL named EN. EN indicates the error monitoring window for RTS. If during the period that EN is *on*, the RBL signal does not discharge, an error flag will be raised. The RBL signal already predicts the worst read delay for enabling the SAE. By using the RBL to generate SAE, we allow a window for detecting errors. During the read operation, when the clock is ‘1’, SAE becomes active with some delay. If at the second half of the negative edge of the clock, SAE signal is not yet lowered, the chance to read the data properly is missed and an error signal will be generated before the rising edge of the clock. The timing diagram in Figure 2 show when the error signal is generated.

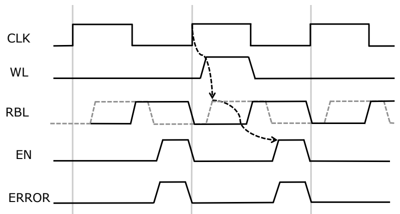

Fig. 2: RTS error signal generation

RBL is a common technique in modern SRAM designs. Leveraging this existing technology reduces the overhead of area consuming and complex error detection mechanisms. RTS error detection overhead is insignificant regardless of the SRAM size. If the SRAM is large, the area overhead of large error detection mechanisms such as Razor might be acceptable, but in small SRAMs, the area overhead is significant in comparison with other components.

B. *Protecting from Incorrect Writes*

Previous state-of-the-art RAM [7], [8] do not protect against incorrect inputs without requiring faster SRAMs. As a result, studies that use Razor-like solutions have assumptions similar to the Bubble Razor design [6], which states "Writes are clocked on the negative edge of the clock when data is guaranteed to be error free". Although writes tend to be faster than reads, requiring them to complete in half a SRAM cycle adds additional overhead. We propose a minor modification to the write decoder to make it failure-aware.

We define an external failure, corrupt data that causes the “error” or “failure” flag to be raised in the system. When SRAM is placed within a system that has an error detection mechanism, there exists an “error” or “failure” flag. To prevent an incorrect write, error has to be detected and the wordline disabled in less than half a cycle. However, when the system is equipped with error detecting latches/flops, the failure flag can be used to disable the wordline immediately. In order to protect an upcoming write in case of an external failure, RTS uses a decoder that has an active low failure input. When the write operation starts, at the rising edge of the clock, the address decoding takes place and wordline is selected. If the error flag along the input data is raised, the wordline will not be selected. Figure 1 shows the modification needed for the write address decoder used in RTS design. By adding a single NMOS transistor to the decoder with an active low failure signal input, the wordline is selected only if there is no failure present in the system. In RTS this external active low signal is called "nofail". The address bit NMOS transistors are connected in series with the "nofail" signal. When there is a failure, the WL will be deselected and the write operation will be discontinued.

IV. Experiment Setup

Table II lists 4 types of SRAM designs where each one is designed in 3 different sizes (Table I). Size *small* models a typical 1Kb register file for an in-order processor. Size *medium* represents a 2Kb register file fit for a 3-way superscalar out-of-order processor, and size *large* models a large 8Kb single cache bank. Area of a size *large* SRAM is 15% more than a size *medium*.

TABLE I: SRAM sizes to model 3 typical processor SRAMs.

Size	Ports	Words	Word Width
Small	2r1w	32	32
Medium	6r3w	64	32
Large	1r1w	128	64

The SRAM netlists and layouts are generated by FabMem, a subset of FabScalar toolset and uses FreePDK 45nm technology [3]. Table II describes components for each SRAM type which is a combination of what FabScalar offers for netlist and layouts and what we have custom built and drawn using HSpice and Cadence Virtuoso IDE. We have estimated the sense amplifier layout area for Razor based on the used delay chains and control logic shown in Razor-enabled sense amplifier [7]. Its netlist excludes the extra logic for generating the enable signals, en1 and en2, and are input signals to the HSpice simulations.

TABLE II: SRAM types

Type	Description
Trad	FabScalar default design.
RBL	FabScalar + RBL support + latch-type SA.
RTS	FabScalar + wr decoder + RBL + latch-type SA.
Razor	FabScalar RAM + Razor-ed SA.

V. Evaluation

To evaluate RTS, we run HSpice for SRAM types in table II and compare read and write energy consumption, maximum frequency, and SRAM area against Razor.

A. Energy Efficiency

To compare the read and write energy, we run size *small* with a frequency of 1GHz and sizes *medium* and *large* with a frequency of 700MHz. We run HSpice for a period of standby, followed by 10 consecutive writes and 10 consecutive reads. For multiported structures, all the read ports are active during read period and all the write ports are active during write period. Figures 3, 4, and 5 show the average energy breakdown for each size.

RTS and RBL have the RBL in common and, therefore, they consume similar energy levels during read or write operations. Considering Razor is reading and using the large sense amplifiers, it consumes the largest amount of power. In order to have Razor operate faster, either the precharge devices have to be largely sized to elongate the precharge period or the buffer sizes that drive the bitlines to the comparator in the sense amplifier [7].

For small size SRAMs, Razor read energy consumption is 11.96x and 11.89x RBL and RTS respectively. For medium size SRAMs, Razor consumes 8x energy than both RBL and RTS. For size large, the read energy consumption for Razor is 3.37x and 3.30x RBL and RTS. As we increase the size of the SRAM the energy overhead becomes smaller, however, for typical/small sizes, the overhead is a considerable part of the design budget.

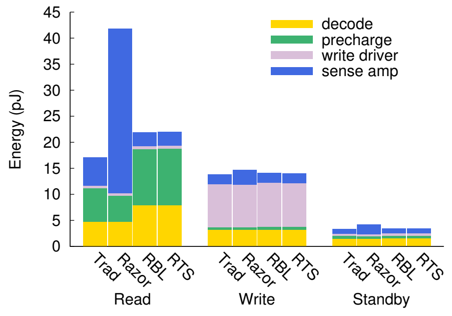

Fig. 3: Energy breakdown in *small* SRAMs.

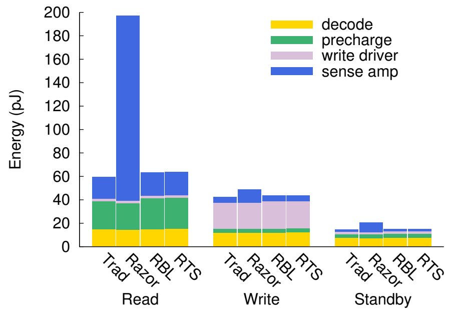

Fig. 4: Energy breakdown in *medium* SRAMs.

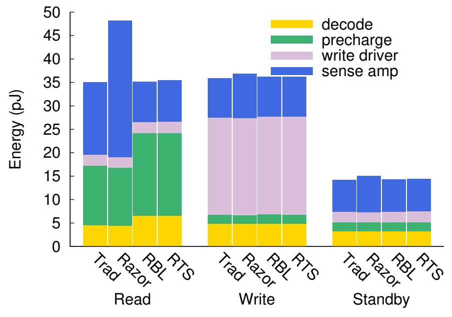

Fig. 5: Energy breakdown in *large* SRAMs.

B. Area

Table III lists the area of the components used, added, or modified in all the SRAMs. In RBL the area overhead is mainly due to the Replica Bitline Column and its dummy write driver and precharge circuits. Comparing RBL and Razor, for the *small* SRAM, RBL overhead is $111.7\mu m^2$ and Razor overhead is $332.3\mu m^2$. Razor has 3 times more area overhead than RBL. The area difference between RBL and RTS is the error detection (*AND gates*) and the slightly larger write address decoders 3.37 vs. 3.18μm^2. The precharge circuit used for the RBL is identical to the SRAM core precharge circuit and to the rest of the SRAMs. The dummy write drivers are also the same as write drivers in all the other SRAMs.

Overall, Razor has the highest area overhead among all the SRAM types and among the error detecting SRAMs, RTS will have a small overhead. If the SRAM has the RBL implemented, the overhead of RTS would be negligible compared to the overall size of the SRAM.

TABLE III: RTS overhead is due to Replica Bitline Column. The Razor sense amplifier is 5.5x RBL bitcell area and 7.4x the traditional sense amplifier. Units are in μm^2.

Type	SA	RBL	Wr Dec	Xtra Buff	Xtra PreCharge	Xtra Wr Dri	Err logic
Trad	0.98	-	3.18	-	-	-	-
RBL	1.36	1.82	3.18	3.03	0.87	1.51	-
RTS	1.36	1.82	3.37	3.03	0.87	1.51	0.98
Razor	10	-	3.18	-	-	-	-

Table IV compares the percentage difference in total area overhead of RTS and Razor with RBL for 3 different sizes. As the size increases, the overhead of RTS becomes negligible.

TABLE IV: The percentage area overhead difference of RTS and Razor with RBL.

Type	RTS	Razor
Small	0.43	5.46
Medium	0.16	2.45
Large	0.02	2.33

C. Process Variation Effects

We performed Monte Carlo simulations to analyze the frequency of the RBL and estimate the maximum clock period for performing reads. VARIUS [12] tool generates variation maps with both systematic and random variation given the SRAM floorplan. We generate a floorplan for RBL and run it with 50 variation maps. For each map, 10 different data points are read and the maximum output delay or the read speed is measured. Out of the 10 data points, the point with the maximum delay is chosen for that particular map.

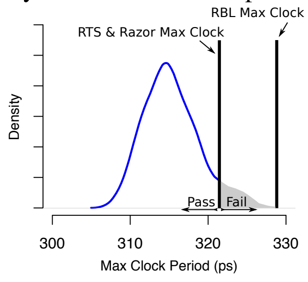

Fig. 6: RBL and Razor achieve similar results, both allow protection from infrequent failures

Figure 6 shows the max clock period density function for the Small RBL which corresponds to a typical Register File in simple in-order cores. To have a high yield, the RBL should operate around 330ps. Since both RTS and Razor can protect against infrequent errors, it is possible to have a higher operating frequency. In the figure, it shows around 322ps. Both Razor and the proposed RTS can protect against errors.

Razor has a slow dynamic *OR* gate to detect failures. The figure uses a small SRAM, and the dynamic *OR* logic is not in the critical path, but SRAMs with larger number of read bits will affect the overall maximum frequency. This would not happen in RTS, because it uses the Replica Bitline to detect the errors. For simplicity, the Razor additional overhead is not included in Figure 6. Overall, RTS achieves the same operating frequency as Razor with area and energy efficiency.

VI. Conclusion

We propose a new way to efficiently implement time speculation in SRAMs. RTS detects read and write failures by offering a simple, yet efficient solution that avoids costly shadow latches in Razor SRAMs. RTS leverages RBL to detect timing failures, and modifies the decoder logic to avoid requiring half cycle writes. This simple yet effective design shows to be 22% to 58% more energy efficient in reading operations and has an error detection mechanism which is 35% to 73% more area efficient than Razor-enabled SRAM.

Acknowledgments

This study is supported in part by the National Science Foundation under grants CNS-1059442-003, CNS-1318943-001, CCF-1337278, and CCF-1514284. Any opinions, findings, and conclusions expressed herein are those of the authors and do not necessarily reflect the NSF views.

References

[1] B. S. Amrutur, M. A. Horowitz, and S. Member, “A Replica Technique for Wordline and Sense Control in Low-Power SRAM’s,” *IEEE Journal of Solid-State Circuits*, 1998.
[2] D. Bull, S. Das, K. Shivashankar, G. Dasika, K. Flautner, and D. Blaauw, “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.*
[3] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, “FabScalar: Composing Synthesizable RTL Designs of Arbitrary Cores within a Canonical Superscalar Template,” in *Proc. of Int’l Symp. on Computer Architecture*.
[4] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, “RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance,” *Solid-State Circuits, 2009 IEEE Journal of.*
[5] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, “Razor: a Low-power Pipeline Based on Circuit-level Timing Speculation,” in *MICRO36.*
[6] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. Harris, D. Blaauw, and D. Sylvester, “Bubble Razor: An Architecture-Independent Approach to Timing-Error Detection and Correction,” in *Solid-State Circuits Conference (ISSCC), 2012 IEEE International.*
[7] E. Karl, D. Sylvester, and D. Blaauw, “Timing Error Correction Techniques for Voltage-Scalable On-Chip Memories,” in *Circuits and Systems (ISCAS), 2005 IEEE International Symposium on.*
[8] M. Khayatzadeh, M. Saligane, J. Wang, M. Alioto, D. Blaauw, and D. Sylvester, “17.3 a reconfigurable dual-port memory with error detection and correction in 28nm fdsoi,” in *2016 IEEE International Solid-State Circuits Conference (ISSCC).*
[9] J. P. Kulkarni, C. Tokunaga, P. A. Aseron, T. Nguyen, C. Augustine, J. W. Tschanz, and V. De, “A 409 GOPS/W Adaptive and Resilient Domino Register File in 22 nm Tri-Gate CMOS Featuring In-Situ Timing Margin and Error Detection for Tolerance to Within-Die Variation, Voltage Droop, Temperature and Aging,” *IEEE Journal of Solid-State Circuits*, 2016.
[10] A. Neale and M. Sachdev, “Digitally Programmable SRAM Timing for Nano-scale Technologies,” in *ISQED*, 2011.
[11] H. Pilo, I. Arsovski, K. Batson, G. Braceras, J. Gabric, R. Houle, S. Lamphier, F. Pavlik, A. Seferagic, L.-Y. Chen, S.-B. Ko, and C. Radens, “A 64Mb SRAM in 32nm High-k Metal-gate SOI Technology with 0.7V Operation Enabled by Stability, Write-ability and Read-ability Enhancements,” in *ISSCC*, 2011.
[12] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, “VARIUS: A model of process variation and resulting timing errors for microarchitects,” *Semiconductor Manufacturing, IEEE Transactions on*, 2008.
[13] K. Viveka and B. Amrutur, “Digitally Controlled Variation Tolerant Timing Generation Technique for SRAM Sense Amplifiers,” in *Quality Electronic Design (ASQED), 2013 5th Asia Symposium on.*