SafeRazor: Metastability-Robust Adaptive Clocking in Resilient Circuits

SafeRazor: metastability-robust adaptive clocking in resilient circuits

Marco Cannizzaro[†], Salomon Beer[*], Jordi Cortadella[‡], Ran Ginosar[*] and Luciano Lavagno[†]

[†]Dipartimento di Elettronica
Politecnico di Torino, Italy
marco.canizzaro@polito.it
luciano.lavagno@polito.it

[*]Electrical Engineering Department
Technion, Israel
sbeer@tx.technion.ac.il
ran@ee.technion.ac.il

[‡]Department of Software
Universitat Politecnica de Catalunya, Spain
jordi.cortadella@upc.edu

***Abstract*—Razor-based circuits can run faster or at a lower voltage than those designed to work at the worst-case corner. However, all known implementations are prone to failures due to the non-deterministic timing behavior introduced by metastability, even in the case where sufficient time is left for resolution. This paper analyzes the causes why Razor-based circuits fail and proposes a new scheme combining the Razor principle with stoppable clocks in a GALS setting. This scheme avoids any timing failure due to metastability and does not require any checkpointing or pipeline restarting logic, other than the usual auxiliary latch to store valid data. The experiments show how the Razor principle can be extended to any generic logic circuit, and not just to micro-processors with sophisticated pipeline flush/recovery mechanisms. In this way, the performance/power benefits of Razor can be adopted without the complex architectural changes required by the various Razor schemes in the literature.**

***Index Terms*—digital circuits, low power design, high speed integrated circuits.**

I. Introduction

PROCESS and operating condition variability plays a key role in digital integrated circuits (ICs), forcing them to operate at speed, voltage and power conditions that are far from the optimum, due to margins. In modern process technology, this usually means operating at a frequency which is 0.7-0.5X the optimum speed of each circuit, and at 2-4X the optimum power level. This huge gap is frequently addressed by using two techniques.

- "Binning" sorts ICs after fabrication based on their performance, allowing identification of the fastest circuits and the possibility to sell them with different pricing. This is applicable mainly to processors, memories, FPGAs and graphics chips, due to their market sizes and the cost involved in the binning process.
- Adaptive voltage scaling (AVS) estimates the performance of the circuit dynamically, due to both fabrication and current operating conditions, and adapts the supply voltage to meet the performance requirements at a reduced power and energy consumption level. It is usually applied to portable electronics, such as cell phones, due to the high demand for low energy circuits in battery powered devices. However, AVS still requires a margin due to the fact that performance is estimated, by using a delay line that replicates the delay of the critical paths. This leaves about 10-20% of performance unexploited (a margin which is likely to increase with technology scaling) and 30-40% of excessive power and energy consumption [2], [8], [9].

The Razor approach presented in [4] proposed estimating the true performance of the circuit at runtime. It actively strives to run the circuit at the lowest supply voltage or at the highest performance (depending on the goal of the IC). Errors may occur as a consequence of the setup and hold time violations, but are corrected with the aid of special circuitry.

The trouble with Razor is that metastability may produce flip-flop outputs that change even after the worst-case clock-to-output delay, generating further timing errors in subsequent stages. The authors of [4], [11], [6] claim to handle metastability by detecting it and using an N-flip-flop synchronizer [7], [1] for metastability resolution. Then, the circuit is re-started from the last safely check-pointed state (before the metastability occurred).

However, *both the original Razor mechanism and all its variants that are known from the literature may miss some metastability failures*, as will be discussed in much more detail below.

The Safe Razor approach proposed in this paper relies on *locally generated clocks that can be reliably stopped every time some register enters a metastability condition*. This completely confines both metastability and logic errors to the first layer of flips-flops in which the error may occur. It uses the metastability error signal to latch a handshaking or local clock generation signal, so that no clock pulses can be propagated to the downstream flip-flops, and hence no metastability can propagate and no illegal state can be generated, until metastability has been resolved.

The key idea of the paper is that the output of a metastability detector rises (if it rises at all) *a bounded amount of time* after the clock edge that caused meta-stability to occur. Only the *falling edge* of that signal occurs an unbounded amount of time after the clock edge. This is the reason why it is impossible to confine meta-stability in a synchronous setting, once it has occurred. Only its spreading can be made very unlikely, thus increasing the Mean Time Between Failure to a sufficiently long value (centuries, or more). This property of meta-stability detectors is obvious in the Spice simulation shown in Figure 1, where the MD output rising edges are

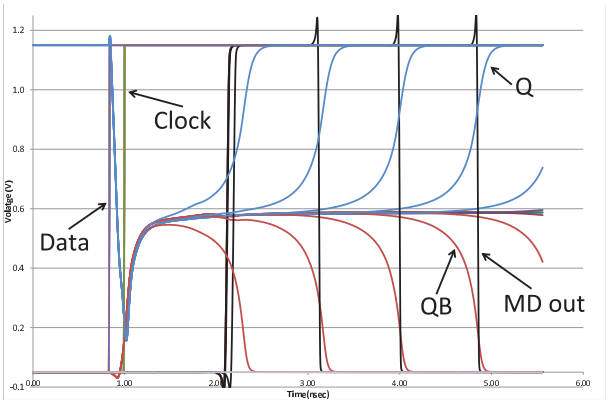


Fig. 1. Behavior of a meta-stability detector output

deterministic, while the falling edges are not.

Hence the Safe Razor approach could be applied both to asynchronous circuits (for example with a 2-phase or a 4-phase protocol), and to locally generated stoppable clocks in a Globally-Asynchronous Locally-Synchronous (GALS) architecture. In this paper we focus on the GALS approach, and we leave other asynchronous protocols to future work. This allows us to implement Safe Razor starting from any synchronous logic circuit, without the need to use expensive multi-cycle check-pointing registers or to perform combinational datapath modifications to be able to "roll back" the entire circuit state 2-3 cycles earlier. We only replace locally the flip-flops with "Safe Razor flip-flops" and use one or more locally generated stoppable clocks. *The combinational logic of the circuit and its overall architecture remain unchanged*.

The Q-modules project [10] proposed a related approach, in which metastability was detected and resolved in a specially designed cell called a Q-flop, and the next local clock pulse was generated only after its resolution. However, Q-modules were fairly large and slow, used a dedicated transistor-level design for Q-flops and used delay-insensitive inter-module communication. Our proposed approach uses standard cells for logic and storage and a standard synchronous data-path.

The paper is organized as follows. In section II we review the main metastability failures present in different Razor variants and analyze why all those approaches fail to prevent metastability. In section III we introduce the asynchronous SafeRazor architecture in a GALS system with stoppable clocks. Section IV presents the timing and performance analysis of SafeRazor. In section V we report experimental results.

II. Metastability in Razor

In this section we provide evidence and simulation results that show the vulnerability of the original Razor approach and of all its known derivatives to metastability failures. In [5] the Razor concept, usually denoted as Razor I, was presented. A modified version, usually denoted as Razor II, was later presented in [11], [4], and its metastability properties were discussed in [3]. Finally, a 2-phase latch-based architecture called Bubble Razor was proposed in [6]. Most of the other publications on Razor circuits are derivations of either Razor I or Razor II.

The authors of [3] proposed the safest approach, which confines meta-stability to the control path, and hence is immune

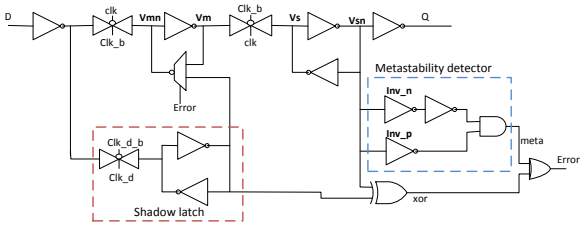


Fig. 2. Razor I circuit [5].

to the undetected timing errors described below, that plague the other Razor-like approaches. However, even they can only claim to extend the MTBF to a value that is comparable to that due to Single-Event Upsets in traditional synchronous circuits. Moreover, their approach reduces the amount of speculation that can be performed, and hence the potential gains, to about *one half of what is achievable with our approach*. This is because if the control path goes meta-stable and resolves to no error, then the maximum amount of undetected timing error of that stage of the datapath *plus* the maximum amount of timing error of the next stage must not exceeed the overalll maximum recoverable amount.

This section first analyzes the metastability properties of the Razor I implementation, and then discusses why Razor II and Bubble Razor still suffer from the same problem, which is inherent in *any synchronous approach to avoid the effects meta-stability. Its probability of occurrence can only be minimized, and only if the circuit is kept operating far away from potential meta-stability conditions*.

A. Razor I

The basic concept behind Razor I was presented in [5] (Figure 2). The incoming data is sampled twice, first by the main flip-flop at the positive clock edge and then by the shadow latch at the negative clock edge. Since the clock frequency is higher than the worst case (WC) clock frequency, differences between the sample from the main flip-flop and the shadow latch may arise and are detected by the comparator. When an error is detected, the input mux updates the data with the shadow latch data and a fail signal is issued to stop the pipeline. Since the main flip-flop may sample below marginal conditions (violating setup time) it is prone to metastability errors. For this purpose a metastability detector is used, and in the original paper it restarts the pipeline after some synchronization flip-flops.

In the implementation of Figure 2, the metastability detector is based on two inverter gates with skewed threshold voltages and designed such that the metastability voltage lies in between the thresholds. The circuit of Figure 2 presents many potential flaws, since latch feedbacks are not gated, which produces short-circuit power consumption. A better circuit implementation is provided in [4], where those contentions are eliminated, but which still has meta-stability problems and which can be analyzed in a similar way.

The Razor I approach described above presents two main potential issues in detecting metastability: (1) Metastability in the master is resolved before the negative clock edge and (2) Metastability in the slave is resolved before the next positive clock edge. We now describe both cases in detail.

1) Master metastability resolved before the negative clock edge: Assume that metastability has occurred at the master latch, and has resolved during the first phase of the clock. The metastability detector cannot detect this metastability. In Figure 2 it monitors the slave latch (rather than the master latch) and when metastability resolves in the master, it is not detectable at the slave. Metastability may resolve into either the same state as captured by the shadow latch, or the opposite state. In the latter case, the error signal is affected before the end of the cycle and triggers the restoration circuit. In the former case, however, the error signal does not trigger, but the flip-flop output may exhibit a late transition, potentially causing timing failures or metastability in subsequent logic. Such cases go undetected, as shown in Figure 3. V_m refers to the internal node in the master latch that becomes metastable (Figure 2), V_sn is the internal slave latch node, V_q is the razor flip-flop output, and clock_d is the delayed clock that is used in the shadow latch. Xor is the output of the error comparator and meta is the output of the metastability detector, while error is the error signal. Figure 3 shows the master becoming metastable until it resolves before the negative clock edge. *Neither metastability nor error are detected, but output V_q is clearly delayed.* This could cause *undetected errors* in the stages downstream, since this (non-deterministic) delay may be beyond the range of timing errors that Razor is designed to identify and correct (typically about half the clock period, and always less than a clock period).

2) Slave metastability resolved before the next positive clock edge: In this scenario, the master becomes metastable, and metastability is transferred to the slave latch at the negative clock edge [1]. Slave latch metastability is resolved during the negative clock phase, resulting in an increased output delay of V_q. But also in this case the metastability detector does not guarantee safe operation because (1) either it may not stay high long enough (2) or it may trigger a meta-stability in the synchronization flip-flops, which resolves to 0. Yet, as shown in Figure 4, output V_q may be delayed.

B. Razor II

The Razor II circuit, proposed in [11], is designed to solve the timing and design issues that appeared in Razor I. Instead of performing both error detection and correction in the flip-flop, Razor II performs only detection in the flip-flop, while correction is performed at the architectural level. This allowed reduction in the complexity and size of the Razor flip-flop, at the cost of increased latency penalty during recovery. The Razor II framework assumes that the system provides an architectural recovery mechanism such as architectural replay, a technique used in microprocessors to support speculative operations such as branch prediction.

The circuit proposed in the Razor II paper is shown in Figure 5. The flip-flop uses a positive level-sensitive latch instead of a master-slave flip-flop. Any transition on the input data within a "forbidden window" is flagged as a timing error. The error detection window lies between the rising edge of DC and the falling edge of CLK. This constrains the minimum propagation delay for a combinational logic path terminating in a Razor II flip-flop to be greater than the high clock phase duration. Delay buffers are required for those paths

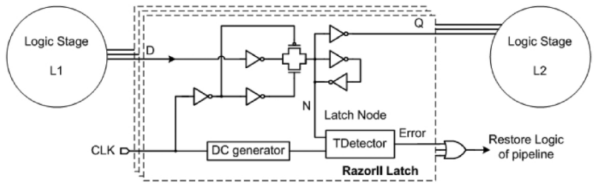


Fig. 5. Razor II flip-flop diagram.

which fail to meet this minimum path delay constraint. It is claimed in [11] that the supply voltage is never lowered to the point where the latch transitions at the falling clock edge and hence metastability of the latch node is avoided. However, a transition too close to the falling or rising edge of DC can lead to *metastability at the Error output.*

The authors of [11] claim that when the Error signal becomes metastable the actual latch node is still correct. However, if the metastable Error signal recovers from metastability by going low, it would miss to detect the increased delay output that may further propagate to flip-flops in subsequent stages and cause metastability in them. In this way, a metastable event in the error signal propagates to a data signal in subsequent flip-flops.

Figure 6 describes the scenario discussed above, when the error signal becomes metastable but resolves to low, not generating an error event. In this case, the latch output transitions after a delay that may be much longer than the maximum clock-to-output delay, and hence may produce a failure in a subsequent register.

C. Bubble Razor

Bubble Razor [6] uses a two-phase latch-based datapath instead of a flip-flop based datapath. This has two main benefits: (1) it breaks the dependency between short path constraints and speculation window, enabling large speculation windows, and (2) it allows for architecture independent local correction, which can scale to large high performance systems.

If data arrives during the speculation window, after the latch opens, then Bubble Razor flags an error. Since when the latch opens, the previous latch closes, there is no possibility of short paths being flagged as timing errors. Moreover, when a latch stalls, data is not immediately lost (unlike with flip-flop based implementations), because its neighboring latches operates out of phase. Therefore stall signals only need to be propagated to neighboring stages and the system is scalable to an arbitrary size.

Both a shadow latch plus a comparator (similar to RazorI) and a transition detector (similar to RazorII) can be used as detection and correction mechanisms. As we already demonstrated, both alternatives may have metastability, which can lead to false positive events.

D. Summary

We have shown that the non-deterministic timing of metastable signals may lead to *false positives*. The key reason is that *Razor circuits assume that any positive edge of the error signal is detected and used to trigger a recomputation*. However *synchronizers only ensure that a signal which may*

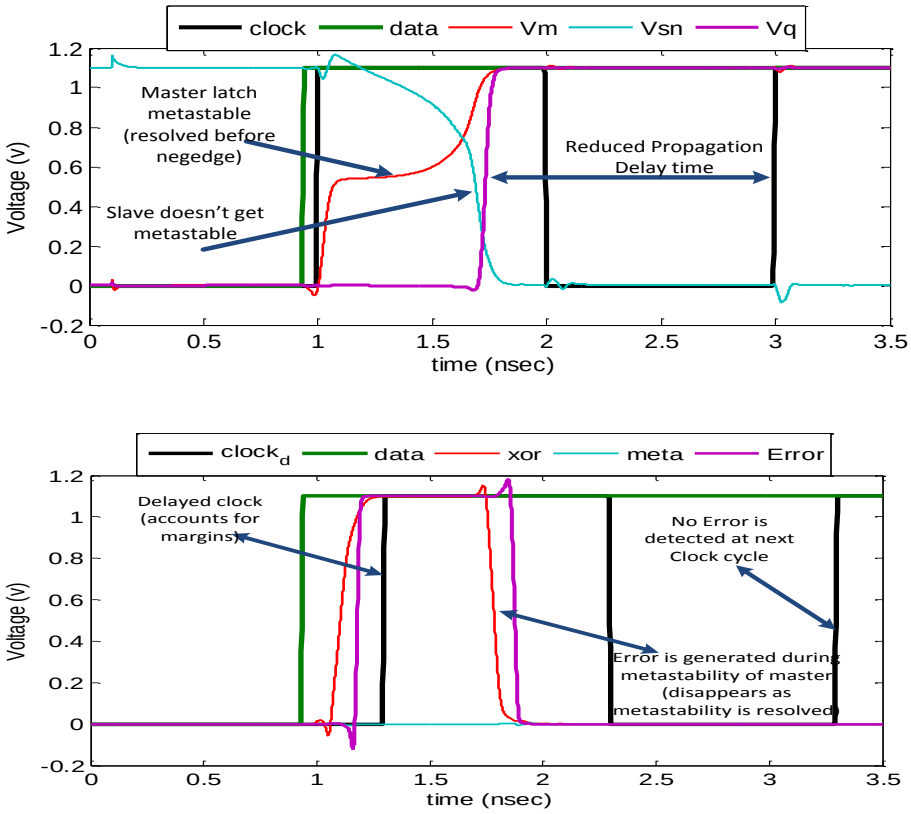


Fig. 3. Simulation of the circuit of Figure 2, when the master becomes metastable and resolves before the negative clock edge.

go meta-stable resolves with a very high probability to 0 or 1 before being used by downstream circuits. Hence *the synchronized* error *may resolve to 0, thus failing to trigger a recomputation even though the data output was delayed due to meta-stability.*

In summary, we can claim that none of the Razor variants presented in the literature completely avoids missing metastability errors, which may propagate throughout the circuit without being ever detected. In the following section we discuss how the problem, which is inherent in any synchronous circuit, can be solved by resorting to an asynchronous (GALS or handshaking-based) implementation.

III. SafeRazor architecture in a GALS system

Figure 7 shows the implementation of a *locally* synchronous SafeRazor module, which can be used in a Globally-Asynchronous Locally-Synchronous (GALS) architecture.

The architecture has three main components:

- **Combinational logic**, which is identical to the one of a conventional architecture.
- **Storage block**, containing the flip-flops (FFs) and the logic to detect metastability and speculation errors.
- **Ring oscillator**, containing the adaptable delays to generate the clock under different operating conditions: normal, metastability and/or speculation errors.

We will now introduce the different components of the architecture by incrementally explaining the modes of operation under the occurrence of different events (as shown in the timing diagram of Figure 8).

A. Normal mode of operation

When neither metastability nor errors occur in the storage block, the signals *meta* and *error* are always at zero. Thus, the two muxes in the ring oscillator and storage blocks are selecting the *"No Err"* input. In this mode, the Metastability Detection (MD) D-latch is always transparent.

This mode of operation is controlled by the ring oscillator formed by the delays d_1 and d_2. The sum of these two delays should be close to the nominal case delay of the combinational logic. The XOR gate in the ring oscillator is simply a pulse generator. The falling edge is generated after a delay d_3 from the rising edge (where $d_1 + d_2 + d_3$ must be larger than the worst-case delay of the logic, as discussed below). The outputs of the main FFs are fed back to the combinational logic through the *"No Err"* input of the mux. The output data (*data_out*) is directly sent through the *shadow* D-latch. It is important to notice that this latch is borrowing time from the next clock cycle, since the falling edge of the clock is delayed by the amount of time determined by d_3. This means that SafeRazor can improve the performance of critical paths fully enclosed within a GALS module, not of critical cycles between modules, because those must always use the non-speculative data, since metastability prevention is ensured only within a module.

B. Operation with metastability

Let us consider the case when some FF may enter a metastable state. The *Metastability Detector* MD block must

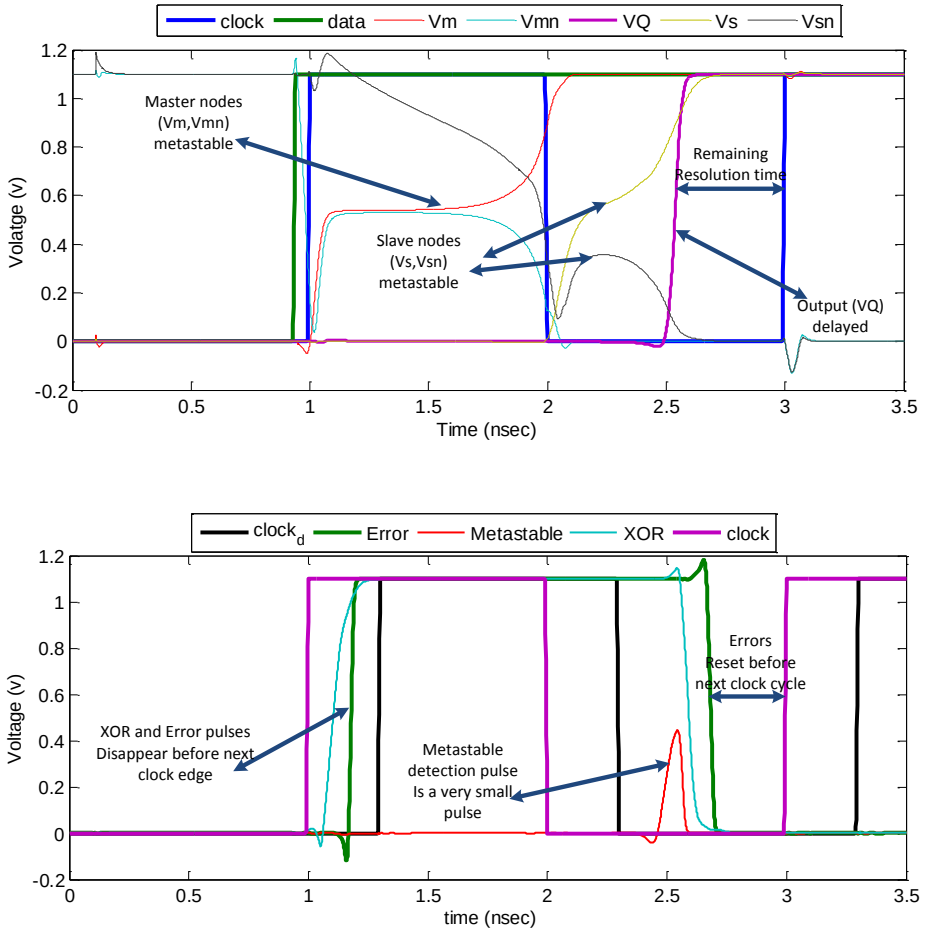


Fig. 4. Simulation of the circuit of Figure 2, when the slave becomes metastable and resolves before the positive clock edge.

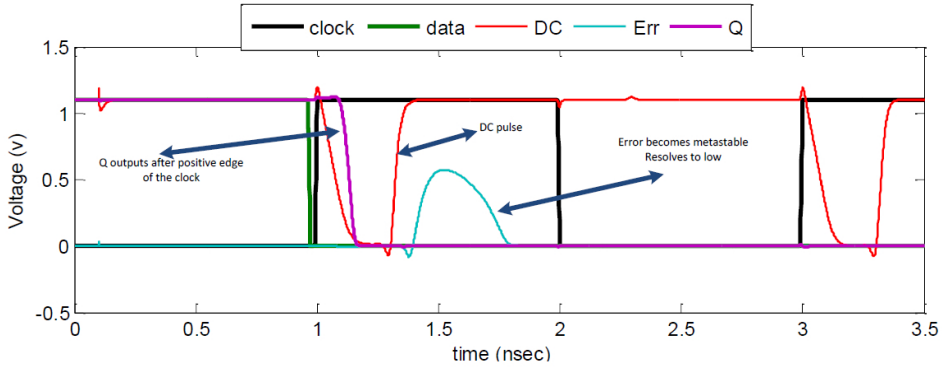


Fig. 6. Simulation of the circuit of Figure 5, when the error signal becomes metastable and resolves to low.

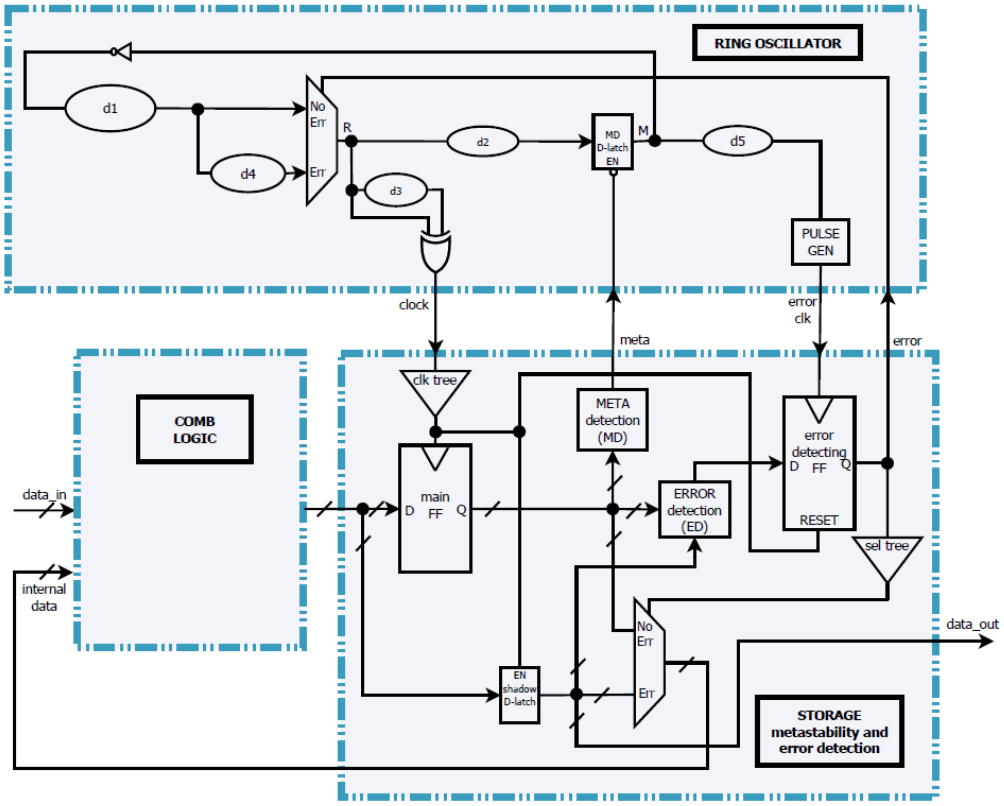


Fig. 7. A SafeRazor module in a GALS system.

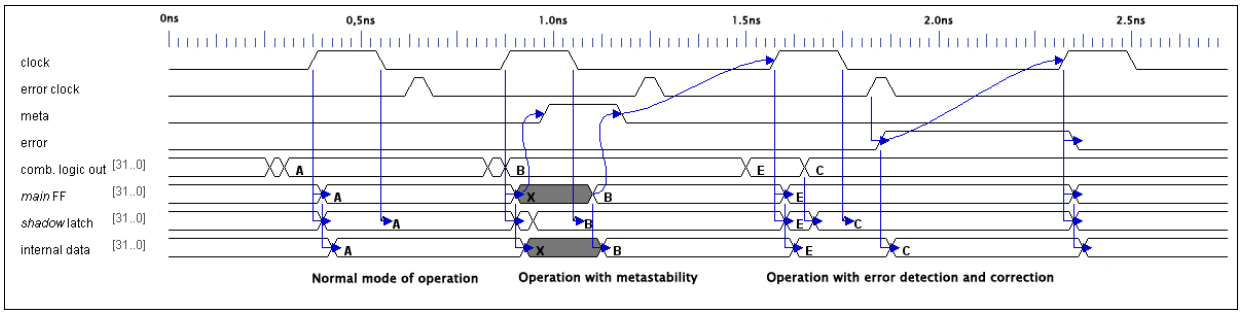


Fig. 8. Timing diagram of the Safe Razor modes of operation.

have access to the output(s) of the master latch of the FF, because only that latch can go metastable. The slave cannot do so when using our scheme, since (as discussed in Section II) any metastability in the slave is a propagation of metastability in the master, and we stop the clock (and hence prevent closing the slave) while the master is meta-stable. Any MD implementation from the literature may be used.

The module-level *meta* signal is generated by OR-ing the outputs of all MD blocks. The *meta* signal controls the MD latch, which *halts* the ring oscillator to guarantee that any decision on data errors (discussed in the next section) is only taken when any metastability condition has been resolved. In this way, *no false positives* are possible with this architecture.

C. Operation with error detection and correction

At every cycle, the error detection logic (ED) checks for a speculation error in each main FF. The shadow latch always captures the correct data value. For that, the falling edge must occur sufficiently late for the critical paths to complete under worst-case conditions. Therefore, the delay d_3 is the one that determines the amount of timing speculation.

The *Error detection* block (ED) consists of a comparator for each FF (XNOR gate) and an OR-tree to gather the error conditions from all main FFs. The output of this OR-tree is captured by the *error-detecting FF* after a delay determined by d_5. When an error is detected, the two muxes in the architecture are switched to select the *Err* inputs:

- The mux in the storage block "connects" the internal data of the combinational logic with the shadow latches.
- The mux in the ring oscillator block grants an extra delay (d_4) to the clock period that allows data to correctly propagate in the datapath.

Note that if metastability also occurs, the error detection and correction are simply delayed until metastability has been resolved. Please note that our scheme takes care of

metastability both due to better-than-worst-case design within a SafeRazor module, and due to the unrelated local clocks in the GALS architecture. We are assuming that the initial design was properly set up to work in a functionally correct manner using a GALS architecture. This means that some mechanism must be provided at the functional level in the various GALS blocks, to ensure that they correctly exchange data.

If meta-stability occurs due to unrelated clocks in different modules, then, as discussed at the end of Section III-A, the safely latched version stored in the shadow D latch is used for inter-module communication. Hence these other modules never see meta-stable or incorrect data, but always wait until meta-stability has resolved, like in Q-modules.

IV. Timing and Performance Analysis

A. Timing constraints

For simplicity, and without loss of generality, we assume that all the muxes, flip-flops, latches and single gates appearing in Fig. 7 have zero delay. The actual delays of these components can always be included in the delays of the other components.

The timing constraints will be represented as inequalities involving two paths in the circuit, e.g., $path_1 < path_2$. In a real synthesis/analysis flow, these constraints must account for the variability of the components, either using statistical information or derating (dr) factors, e.g.,

$$dr_1 \cdot path_1 < dr_2 \cdot path_2, \qquad \text{with } dr_2 \leq 1 \leq dr_1.$$

The derating factors dr_1 and dr_2 represent the effect of *on-chip variation* (OCV). They are much smaller than the deviation of the best and worst case delays with respect to the nominal case, since process, voltage and temperature variations for (closely placed) logic on a chip are much smaller than across chips and across operating conditions. Reasonable values for a modern technology such as 45nm could be 0.95 and 1.05.

For the combinational logic (COMB), we will distinguish two delays: $\text{COMB}_{\max}$ and $\text{COMB}_{\min}$. They represent the longest and shortest path in the combinational logic.

B. Normal mode of operation

In this mode of operation, we will assume that the cycle period starts with correct data in the main FFs and needs to guarantee correct data in the shadow latch by the end of the period.

Setup constraint: Two paths start at the root of the clock tree (R) and end at the EN and D inputs of the shadow latch. The launching path goes through the main FF and the combinational logic. The capturing path goes along the ring oscillator (d_2 and d_1) and the delay that determines the speculation time and the falling edge that closes the shadow latch (d_3). The constraint is as follows:

$$CT + COMB_{\max} < d_2 + d_1 + d_3 + CT. \qquad (1)$$

Hold constraint: This constraint prevents data overrun in the shadow latch by the incoming data from the next cycle. Again, the two competing paths start at the root of the clock tree (R). The launching path goes through the main FF and the shortest path of the combinational logic, whereas the capturing path goes to the EN signal of the shadow latch through d_3. In this case, the capturing path should be shorter than the launching path:

$$d_3 + CT < CT + COMB_{\min}. \qquad (2)$$

This constraint is similar to the short-path constraint in Razor-like circuits. It basically indicates that the speculation time will have to be *protected* by the min delay of the combinational logic to avoid data overruns. It can be removed with a higher flip-flop cost by using a Safe version the two-phase Bubble Razor approach, but this extension is left to future work.

Note that *CT* appears on both sides of inequalities (1) and (2). *This term cannot be canceled out since (1) it refers to different branches of the tree, and (2) it is affected by different derating factors.*

C. Operation with metastability

In case some main FF enters a metastable state, the MD logic stretches the cycle period by closing the MD latch until metastability is resolved. For a correct timing, a setup constraint on the MD latch is required to guarantee that the input of the MD latch arrives later than the EN signal when metastability occurs.

This constraint involves two paths that start at R: one path arrives at the D input of the MD latch through d_2 while the other path traverses the clock tree and the MD block:

$$d_2 > CT + MD. \qquad (3)$$

It is important to realize that the previous constraint only requires the min delay of the MD block, which is bounded.

D. Error detection

The error detection logic has to honor some timing constraints for a correct operation. The *error* signal generated at the error detecting FF is selecting one of the delays in the mux of the ring oscillator ("Err" or "No Err"). This signal must arrive before the inputs.

The two competing paths start at the output of the MD-latch (M). One path goes through d_5, the pulse generator and the error detecting FF. The other path goes through d_1:

$$d_5 < d_1. \qquad (4)$$

Another constraint involves the error detecting FF. In this case, the D input of the latch must arrive before the clock edge. The launching and capturing paths start at R. The launching path crosses d_3, the clock tree, the shadow latch, and the ED block, since the latest signal at the ED block is the one that comes from the shadow latch. The capturing path crosses d_2, d_5 and the pulse generator.

$$d_2 + d_5 > d_3 + CT + ED. \qquad (5)$$

E. Error correction

In case an error is produced, the cycle period must be stretched to allow the combinational logic to use the data

stored in the shadow latches during the next cycle. The error correction requires an extra delay (d_4).

The competing paths start at the output of the MD latch (M). The launching path goes through d_5, the error detecting FF, the buffering tree of the mux select signal (sel tree), the mux of the storage block and the combinational logic. The capturing path generates the next pulse for the main FF and goes through d_1, d_4 and the clock tree.

$$d_5 + ST + COMB_{\max} < d_1 + d_4 + CT. \quad (6)$$

F. Performance analysis: putting all together

The set of constraints (1)-(6) is a linear programming model that can be solved to maximize performance (i.e. minimize the cycle period).

The cycle period $\mathbb{C}$ in SafeRazor can be expressed as follows:

$$\mathbb{C} = d_1 + d_2 + p_{err} \cdot d_4 + d_{meta}$$

where p_{err} is the probability of error and d_{meta} is a random variable that represents the extra delay of the cycle period that is added when metastability occurs.

Unfortunately, the cost function is not linear since p_{err} depends on the timing speculation of SafeRazor (d_3), i.e., the larger the speculation, the larger the probability of error. Additionally, d_{meta} is a value with a non-trivial distribution function. However, we can make realistic assumptions on these values. First of all, p_{err} should be kept small by the overall speculation control loop (whose design is beyond the scope of this paper), hence it can be considered negligible. We will see in Section V that as soon as it becomes non-negligible, the effective clock cycle starts increasing. For the same reason, we can assume that the average value of d_{meta} is negligible, since metastability occurs very rarely and the average resolution time is short.

With these assumptions, we can rewrite $\mathbb{C}$ as

$$\mathbb{C} = d_1 + d_2 + \lambda \cdot d_4$$

where λ is a small constant that simply prevents d_4 from growing without bounds.

Finally, to guarantee that p_{err} is small, it is required that d_3 is bounded. Let us assume that we get, e.g. from the simulations reported in the next section, an upper bound on d_3, called $Spec_{\max}$ that guarantees the desired probability of error. The linear programming model can then be augmented with a new constraint:

$$d_3 < Spec_{\max} \quad (7)$$

Now, we can define a linear programming model for SafeRazor as follows:

$$\min : \mathbb{C} = d_1 + d_2 + \lambda \cdot d_4$$
$$\text{subject to } (1) - (7)$$

G. Performance: an intuitive analysis

In order to get some insight about the best usage scenarios for SafeRazor, we propose to do a *simplified analysis* with some assumptions, while the linear program above can be used to obtain the exact results.

- Let us assume that the common terms on the right and left hand side of the inequalities can be canceled out, i.e. we assume that OCV is zero.
- Let us assume that the inequalities can be satisfied with tight values ($\leq$ instead of $<$).
- Let us assume that constraints (3) and (4) are satisfied without slack, i.e., $d_2 = CT + MD$ and $d_1 = d_5$.
- Let us call $\mathbb{C} = d_1 + d_2$ the cycle period in normal operation mode.

With the previous assumptions, the timing constraints can be simplified and rewritten as follows:

$$\mathbb{C} + d_3 > COMB_{\max} \quad (C1)$$
$$COMB_{\min} > d_3 \quad (C2)$$
$$\mathbb{C} > d_3 + CT + ED \quad (C3)$$
$$d_4 + CT > ST + COMB_{\max} \quad (C4)$$

(C1) is the constraint that guarantees the correctness of the values of the shadow latch. It also shows how the cycle period $\mathbb{C}$ can be shortened by the speculation time (d_3). (C2) represents the short-path constraint of Razor-like systems and shows how the combinational logic must have a min delay longer than the speculation time. This constraint can be expensive to meet at the logic level, since it may require a lot of delay buffers in short paths. It also can be removed by using a two-phase Safe Bubble Razor approach.

(C3) is an important constraint on the cycle period. In simple words: the cycle period must be longer than the error detection period that goes from the ring oscillator to the storage block and back to the ring oscillator. This cycle includes the speculation time and the clock tree. For this reason, the SafeRazor scheme is appropriate for systems with local clocks in which the clock latency can be locally controlled.

Finally, (C4) is a constraint that guarantees the correctness of the correction cycle. Given the low probability of error, the impact of this constraint on performance is negligible.

We can easily estimate the speedup that SafeRazor can have with regard to a conventional circuit:

$$Speedup = \frac{\mathbb{C} + d_3}{\mathbb{C}} = 1 + \frac{d_3}{\mathbb{C}}$$

The previous equation is only valid for the cases in which constraint (C3) has still some slack and for those values of d_3 that produce low probability errors. Satisfying all these timing constraints proved to be very challenging in physical design, because current tools cannot satisfy relative timing constraints. Hence for each such constraint, we had to pick an absolute value, and replace it with two absolute constraints (i.e. a<b had to be replaced with a<K and K<b for some constant time K). Iterating this process to convergence was lengthy, but we believe that extending physical design tools to properly handle these constraints would be perfectly feasible.

V. Results

In this section we discuss the results of implementing a SafeRazor circuit with 3 GALS islands, each containing a pipelined multiplier with 4 stages, as shown in Figure 9.

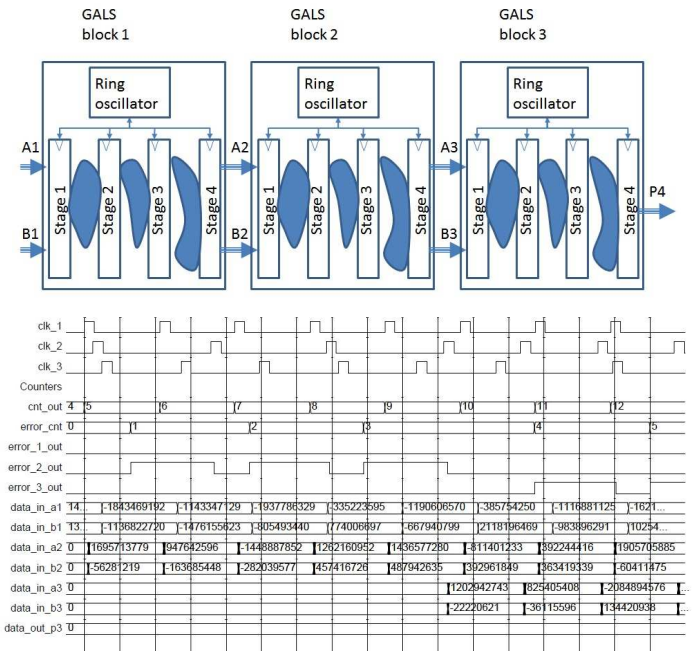


Fig. 9. GALS design including 3 pipelined clock islands

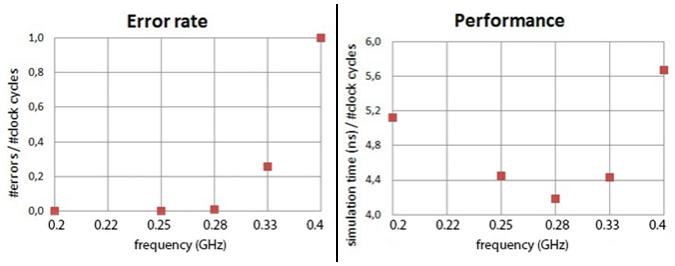


Fig. 10. Error rate and average effective clock period

Number of flip-flops	Architecture	COMB LOGIC		STORAGE		CLOCK GENERATOR		TOTAL	
		cell unit	%	cell unit	%	cell unit	%	cell unit	overhead
3,2E+01	Synchronous	8,5E+03	86%	1,4E+03	14%	-	-	9,8E+03	-
	Razor		67%	4,1E+03	33%	-	-	1,3E+04	27,7%
	Safe Razor		57%	4,1E+03	28%	2,3E+03	15%	1,5E+04	50,6%
1,6E+03	Synchronous	4,3E+05	86%	6,9E+04	14%	-	-	5,0E+05	-
	Razor		67%	2,1E+05	33%	-	-	6,4E+05	27,7%
	Safe Razor		67%	2,1E+05	32%	2,3E+03	0%	6,4E+05	28,1%
1,9E+06	Synchronous	5,1E+08	86%	8,3E+07	14%	-	-	6,0E+08	-
	Razor		67%	2,5E+08	33%	-	-	7,6E+08	27,7%
	Safe Razor		67%	2,5E+08	33%	2,3E+03	0%	7,6E+08	27,7%
2,3E+09	Synchronous	6,1E+11	86%	9,9E+10	14%	-	-	7,1E+11	-
	Razor		67%	3,0E+11	33%	-	-	9,1E+11	27,7%
	Safe Razor		67%	3,0E+11	33%	2,3E+03	0%	9,1E+11	27,7%

TABLE I
AREA COMPARISON BETWEEN TRADITIONAL, RAZOR AND SAFERAZOR

The figure also shows a simulation of several clock cycles, illustrating that the three clocks are very different, especially when errors occur. In the simulated case, stages 2 and 3 had a very high error rate because their clock frequency was about 400MHz, while that of stage 1 was lower. Note also how errors occur in stage 3 only when worst-case paths are activated due to non-zero data inputs.

The design was synthesized with Synopsys Design Compiler using a 90 nm standard cell library. Then we performed the physical layout of the netlist using Cadence Encounter, *while satisfying the timing constraints described in Section IV*. The simulation, from which we extracted error rate and performance of the system, was done with back-annotated delays extracted from the physical layout, including wiring. This is essentially the same level of sign-off accuracy that is required for timing-critical portions of industrial designs.

The synchronous clock period for the synchronous circuit was 5.19ns which corresponds to a clock frequency of about 200MHz.

Metastability was modeled digitally as follows:

- The flip-flop always becomes metastable if the interval between the data input and clock transitions is between 0 and *Tw* (this is a rather conservative approximation, since the actual probability to enter meta-stability depends on that interval).
- The probability to leave metastability, if the flip-flop enters it, depends on the interval Δt between the data input and clock transitions:

$$f(\Delta t) = Tw * e^{\frac{-\Delta t}{\tau}}$$

Tw is a parameter which depends on the rise/fall time of the gates in the selected technology. We estimated τ, as suggested in [2], as half the delay of a fanout-of-four smallest inverter.

In order to test the correct operation of SafeRazor, several simulations were carried out, each with a different clock frequency, by reducing the clock period given by the $\mathbb{C} = d_1 + d_2$ delay.

The left chart in Figure 10 shows how the error rate is 0 with a frequency from 200MHz to 280MHz. At 330MHz the error rate is 25% and at 400MHZ (double the frequency of the synchronous design) the error rate is 100%. Note that this increase in the error rate is due to the activation of paths of different lengths with different probabilities, not to the occurrence of on-chip variation, which was not modeled in our setup. Hence the only practical significance of these simulations is to show that the control loop that is in charge of tuning the d_1 and d_2 delays at runtime should keep the error rate to a low value in order to ensure the best performance.

The right chart in Figure 10 shows the effect of the speculative execution on the average effective clock period (computed as the total simulation time divided by the number of successful computations executed in the pipeline) of the SafeRazor architecture.

Performance speeds up linearly with the frequency as long as the error rate is close to 0. Then the clock period increases due to the increases of the error rate. At 330MHz, performance is still 15% better than the synchronous design while at 400MHz performance is only 7% worse than the synchronous design. This is an important result compared to the classical Razor design, which requires an extra clock cycle any time an error is detected, and hence would halve the effective clock frequency if the error rate is 100%.

Table I shows the area estimates for various block sizes, from approximately 30 to $2 * 10^9$ flip-flops and from $8 * 10^3$ to $6 * 10^{11}$ cell units respectively in the combinational logic, for: (i) a standard synchronous circuit, (ii) Razor (as described in [5]), and (iii) SafeRazor. One can see that, compared to a synchronous circuit (i), the total area overhead for Razor (ii) is always 27.7%, while in SafeRazor (iii) the overhead is increased due to the ring oscillator by 15% in the case with 30 flip-flops (which obviously is too fine-grained) and by a negligible amount in all other cases.

VI. Conclusions

The SafeRazor architecture presented in this paper allows one to implement the nearly-critical clocking approach proposed by the Razor architecture to *any logic block*, not just to microprocessors with a restartable pipeline. It does so by introducing the idea that *GALS or asynchronous local clocks can be safely stopped while metastability is resolved*. This removes the need for expensive check-pointing logic and totally eliminates metastability failures, which still affect the classical Razor approach if the metastability detection logic itself (which in that case was synchronous, while in our case it is asynchronous) goes metastable.

In this paper we focused on a GALS implementation of SafeRazor, but also handshaking-based versions could be implemented, based on the same idea. We plan to look into those in the future. Similarly we could adapt our approach to work with the Bubble Razor architecture, and reduce the impact of hold constraints and short paths, which made our methodology difficult to apply in practice. A detailed analysis of this solution is also left to future work.

References

[1] S. Beer, J. Cox, T. Chaney, and D. Zar. MTBF bounds for multistage synchronizers. *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 19:158–165, 2013.

[2] S. Beer, R. Ginosar, M. Priel, R. R. Dobkin, and A. Kolodny. The devolution of synchronizers. In *IEEE Async Symp.*, pages 94–103, 2010.

[3] K. A. Bowman, J. W. Tschanz, N. S. Kim, J. C. Lee, C. B. Wilkerson, S. Lu, T. Karnik, and V. De. Energy-efficient and metastability-immune timing-error detection and instruction-replay-based recovery circuits for dynamic-variation tolerance. In *IEEE ISSCC*, pages 402–403, 2008.

[4] S. Das, D. Roberts, Seokwoo Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. T. Mudge. A self-tuning DVS processor using delay-error detection and correction. *Solid-State Circuits, IEEE*, 41:792–804, 2006.

[5] D. Ernst, S. Nam Sung Kim, Das, S. Pant, R. Rao, Toan Pham, C. Zieslera, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *IEEE Int. Symp. Microarchitecture (MICRO-36)*, pages 7–18, 2003.

[6] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D.M. Harris., D. Blaauw, and D. Sylvester. Bubble razor: Eliminating timing margins in an ARM Cortex-M3 processor in 45 nm CMOS using architecturally independent error detection and correction. *Journal of Solid-State Circuits, IEEE*, 48:66–81, 2013.

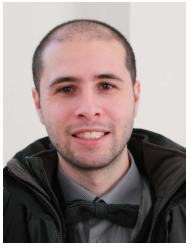
[7] R. Ginosar. Metastability and synchronizers: A tutorial. *Design Test of Computers, IEEE*, 28:23–35, 2011.

[8] J.Zhou, D.Kinniment, G.Russell, and A. Yakovlev. Adapting synchronizers to the effects of on chip variability. In *IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2008.

[9] S. Nassif, K. Bernstein, D. Frank, A. Gattiker, W. Haensch, B. Ji, E. Nowak, D. Pearson, and N.J Rohrer. High performance CMOS variability in the 65nm regime and beyond. *Electron Devices Meeting, IEEE International*, 10:569–571, 2007.

[10] F.U. Rosenberger, C.E. Molnar, T.J. Chaney, and T.-P.; Fang. Q-modules: internally clocked delay-insensitive modules. In *IEEE Trans. on Comp.*, pages 1005–1018, 1988.

[11] S.Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D.M. Bull, and D.T. Blaauw. RazorII: In situ error detection and correction for PVT and SER tolerance. In *IEEE J. Solid-State Circuits*, pages 32–48, 2009.



Marco Cannizzaro received the Bachelor's and Master's degrees in Computer Engineering from Universitá di Palermo and Politecnico di Torino, Italy, in 2006 and 2010, respectively. He received the Ph.D. in Electronics Engineering from Politecnico di Torino in 2014. He has worked as research assistant at Columbia University in the City of New York in 2009 and 2013. His main area of research interest involves asynchronous circuits for data communication and adaptive clocking in resilient circuits.

Salomon Beer received the B.Sc degree in electrical engineering and B.A in Physics from the Technion - Israel Institute of Technology, Haifa in 2004. He is currently pursuing the Ph.D. degree in computer engineering at the same institute where he is a Haso-Plattner-institut (HPI) fellow. During 2005 to 2011 he held engineering and algorithmic development positions in Freescale Semiconductor and authored several publications and patents in the field of computer architecture, VLSI systems and computer vision algorithms.

Jordi Cortadella (S'88) received the M.S. and Ph.D. degrees in Computer Science from the Universitat Politècnica de Catalunya, Barcelona, in 1985 and 1987, respectively. He is a Professor in the Department of Computer Science of the same university. In 1988, he was a Visiting Scholar at the University of California, Berkeley. His research interests include formal methods and computer-aided design of VLSI systems with special emphasis on asynchronous circuits, concurrent systems and logic synthesis. He has co-authored numerous research papers and has been invited to present tutorials at various conferences.

Dr. Cortadella has served on the technical committees of several international conferences in the field of Design Automation and Concurrent Systems. He received best paper awards at the Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (2004), the Design Automation Conference (2004) and the Int. Conf. on Application of Concurrency to System Design (2009). In 2003, he was the recipient of a Distinction for the Promotion of the University Research by the Generalitat de Catalunya. He is member of the Academia Europaea.

Ran Ginosar , PhD, Head of the VLSI Systems Research Center and Associate Professor, Electrical Engineering and Computer Science departments at the Technion-Israel Institute of Technology.

Professor Ginosar has been a visiting Associate Professor with the University of Utah and co-initiated the Asynchronous Architecture Research Project at Intel, where he worked on Intel's asynchronous test chip. He has co-founded a number of companies: i Sight, Ltd., an electronic imaging company, UltraGuide, Ltd., an electronic medical device company, Mobilian Corporation, a wireless fabless chip company that developed VLSI chip sets for wireless modems, Trig Medical, an electronic medical device company, AI Semi, a medical processor company, Plurality Ltd., a multi-processor core company, NeuroVibes, a brain-computer interface chips company, and Ramon Chips, a rad-hard integrated circuits company.

Professor Ginosar has published numerous papers and his inventions have resulted in a large number of patents. His research interests include VLSI architecture, asynchronous logic and synchronization, electronic imaging, networks-on-chip, manycore architecture and bio-chips. He is presently engaged in research of multiple aspects of synchronization, partly in collaboration with industry. He has taught a number of industrial courses on synchronization, on-chip interconnect and multiple clock domain SoC since 2003.

Luciano Lavagno (S '06) received his Ph.D. in Electrical Engineering and Computer Science from the University of California at Berkeley in 1992. He has been the architect of the POLIS project, developing a complete hardware/software co-design environment for control-dominated embedded systems, and an architect of the CtoSilicon high-level synthesis system from Cadence Design Systems. He is a co-author of two books on asynchronous circuit design, of a book on hardware/software co-design of embedded systems, and has published over 200 journal and conference papers.

He is currently a full professor with the Department of Electronics and Telecommunication Engineering of Politecnico di Torino. He has been an associate editor of IEEE TCAS and ACM TECS. His research interests include the synthesis of asynchronous and low-power circuits, the concurrent design of mixed hardware and software embedded systems, and the high-level synthesis of hardware modules from algorithmic specifications.