Fault Classification of the Error Detection Logic in the Blade Resilient Template

Felipe A. Kuentzer and Alexandre M. Amory
Faculty of Informatics, PUCRS University
Av. Ipiranga, 6681, Porto Alegre, Brazil
felipe.kuentzer@acad.pucrs.br, alexandre.amory@pucrs.br

***Abstract*—Resilient architectures emerged as a promising solution to remove worst-case timing margins added due to process, voltage and temperature variation, improving system performance while reducing energy consumption. Asynchronous circuits can also improve energy efficiency and performance due to the absence of a global clock. A recently proposed circuit template, called Blade, leverages the advantages of both asynchronous and resilient techniques. However, Blade still presents challenges in terms of testing, which hinder its practical application. This paper evaluates the fault behavior of the Error Detection Logic (EDL) block of Blade with single stuck-at or propagation delay fault models. We propose a fault classification based on the effects observed in the overall circuit operation while in the presence of a fault. This classification shows the obtained fault coverage assuming three different testability scenarios and it also shows that a single fault can entirely disable an EDL, disabling its resilience. The proposed classification can be used in the future to improve the design for testability of resilient architectures.**

I. Introduction

Resilient designs emerged as a solution to to reduce timing margins that must be added to traditional synchronous designs to ensure correct operation under worst-case delays caused by process, voltage, and temperature (PVT) variations. Resilient architectures [1] [2] [3] rely on extra logic to detect and recover from timing violations that may appear due to the the timing margin reduction, while improving performance and energy efficiency.

Asynchronous templates can also be a solution for this problem, while improving energy efficiency due to the absence of a global clock and performance with the design for the average-case. Quasi-delay-insensitive (QDI) embed a completion signal in the data representation, which makes the design robust to delay variations, but it uses more silicon area if compared to traditional implementations. Bundled-data templates (e.g. micropipeline [4]) present lower area than QDI, but its delay lines must also be implemented with sufficiently large margins to account for PVT variations. Some solutions for these margins in bundled-data templates were proposed at [5] and [6].

The Blade template [7] is a recently proposed alternative that combines the advantages of the asynchronous bundled-data designs and resilient techniques to alleviate these timing margins. Blade uses two reconfigurable delay lines and error detection logic to detect timing violations coupled to a novel speculative handshake protocol that improves average-case performance. The authors demonstrated that Blade circuit can achieve as much as 30% power reduction at the same performance, when compared to a similar synchronous circuit, for an area overhead of about 10%.

Despite these promising results, Blade's practical usage (as all others resilient circuits) is still hindered by the challenges in terms of testability. For instance, as demonstrated in this paper, a single fault at the block called Error Detection Logic (EDL), responsible for detecting the timing violations (TV) and for triggering the recovery process, is sufficient to disable the EDL. This, in turn, disables the circuit resilience, missing timing violations. Thus, without a proper testing approach it is not possible to assure that the EDLs are going to perform according to their specification.

This paper presents a fault analysis and proposes a fault classification for the Blade's EDL block with single stuck-at faults or propagation delay faults. We evaluate the fault coverage of EDLs with three scenarios: assuming only functional test; adding scan cells to improve the observability of EDL outputs; and considering a method for injecting timing violations at the EDL input. The results show that, without DfT logic, the fault coverage of the EDL is low, jeopardizing Blade's correct functionality, as it is not possible to assure that the EDL is going to perform its violation detection role.

The remainder of this paper is organized as follows. Section II introduces the Blade template and explores the most relevant aspects associated to this paper. Section III describes the fault simulation environment used to produce the results for the analysis. Section IV presents an extensive analysis and discussion on the effects observed for each fault and how they can be detected. In Section V we propose the fault classification and discuss how it could guide future works. Finally, Section VI provides conclusions.

II. Blade Template

The Blade template [7], shown in Figure 1, consists of an asynchronous Blade controller, two reconfigurable delay lines, and an EDL. The Blade controller communicates with other stages using a typical bundled data channel L/R. The δ delay controls the moment the data at the output of the combinational logic can be sampled and propagated through the EDL. The Δ delay defines the amount of time that the latch is transparent, accepting data corrections. A TV is flagged if data changes during this time window, which allows a

978-1-4673-9007-1/16 $31.00 © 2016 IEEE
DOI 10.1109/ASYNC.2016.9

subsequent correction. This time window is defined as the timing resiliency window (TRW).

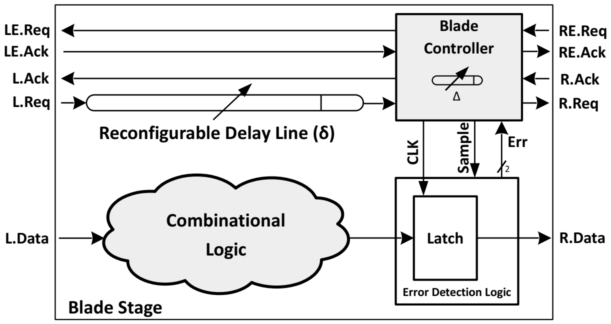

Fig. 1: Blade template [7]

The error detection logic flags a timing violation by asserting its *Err* signal. The Blade controller then communicates with its right neighbour using an additional error channel RE/LE. To recover from the timing violation the next stage delays its latch opening until the correct data is propagated through the combinational logic.

A. Error Detection Logic

The error detection logic used in Blade is shown in Figure 2. As detailed in [7], the design consists of error detecting latches, generalized C-elements and Q-Flops [9]. The latches are based on the Transition Detecting Time Borrowing (TDTB) latches proposed in [8]. The generalized C-element acts as a memory cell that stores any violation detected during the high phase of the *CLK*. This C-element switches to 0 when *CLK* is at 0 and to 1 only when both the *CLK* and the XOR output are at 1. The output of the C-element is sampled at the end of the TRW by the Q-Flop. The Q-Flop ensures safe operation against metastability at the *X* signal and the C-element by applying a filter that prevents its outputs from becoming metastable. The dual-rail signal *Err*, composed by wires *Err0* and *Err1*, stalls the controller until the outputs are stable and it can safely evaluate if an error occurred. The delay element t_{TD} defines the transition detector pulse width, while t_{comp} is the compensation delay added to ensure that a transition before the rising edge of *CLK* is not flagged as a violation. Figure 2 also presents the wire labels that are referenced throughout the Section IV.

B. Controller

The Blade controller implements a new form of asynchronous handshaking protocol, called *speculative handshaking* [7]. The implementation is divided into three interacting Burst-Mode state machines [10]. These state machines are illustrated in Figure 3. Although the controller testability is not the focus of this analysis, understanding its behavior once the EDL has a fault is important for the fault classification. In particular, there are three signals related to the EDL that can cause problem in the controller, *delay*, *Err1* and *Err0*. The highlighted doted states at the center state machine in Figure 3 are the ones that depend on transitions of the *delay* signal. This signal is not directly connected to the EDL, but the CLK signal is connected to EDL, and controls the rise and fall of the *delay*. The highlighted doted states at the right state machine of Figure 3 are the ones that react to changes of *Err1* or *Err0*. As previously described, the controller stalls waiting for these signals to resolve before continuing the protocol process, and the same will happen if the delay signal does not change its value. In summary, a faulty EDL may cause the failure of the controller. This situation is explored in Section IV.

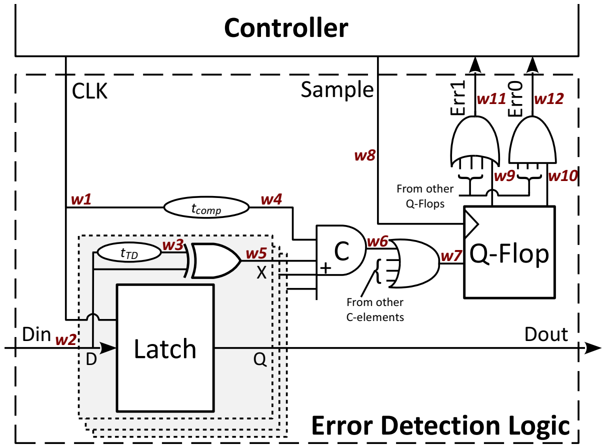

Fig. 2: EDL diagram with labeled wires, adapted from [7]

C. Timing Constraints

The timing overheads associated with the EDL for a single Blade stage are shown in the timing diagram of Figure 4. The delays are divided in five components: (i) propagation delay from *Din* to *X*, $t_{X,pd}$; (ii) the pulse width of *X* defined by the delay element t_{TD} shown in Figure 2, $t_{X,pw}$; (iii) C-element propagation delay, $t_{CE,pd}$; (iv) Q-Flop setup time, $t_{QF,setup}$; and (v) propagation delay of the OR gate before the Q-Flop, $t_{OR,pd}$. The sum of components $t_{X,pd}$ and $t_{X,pw}$ corresponds to the compensation delay t_{comp} described previously.

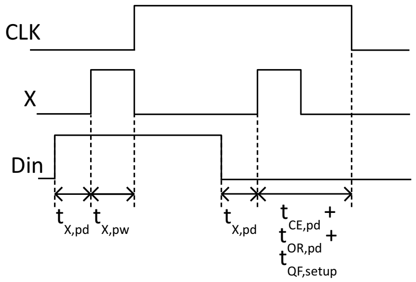

Fig. 4: Timing constraints in Blade [7]

The *Timing Resiliency Window* constraint is the most relevant for our study, since its size is directly affected by the delay components of the error detection logic. The TRW, according to [7], is defined as:

$$TRW = \Delta + t_{X,pw} - (t_{CE,pd} + t_{OR,pd} + t_{QF,setup}) \quad (1)$$

The complete timing analysis of the Blade template is available at [7].

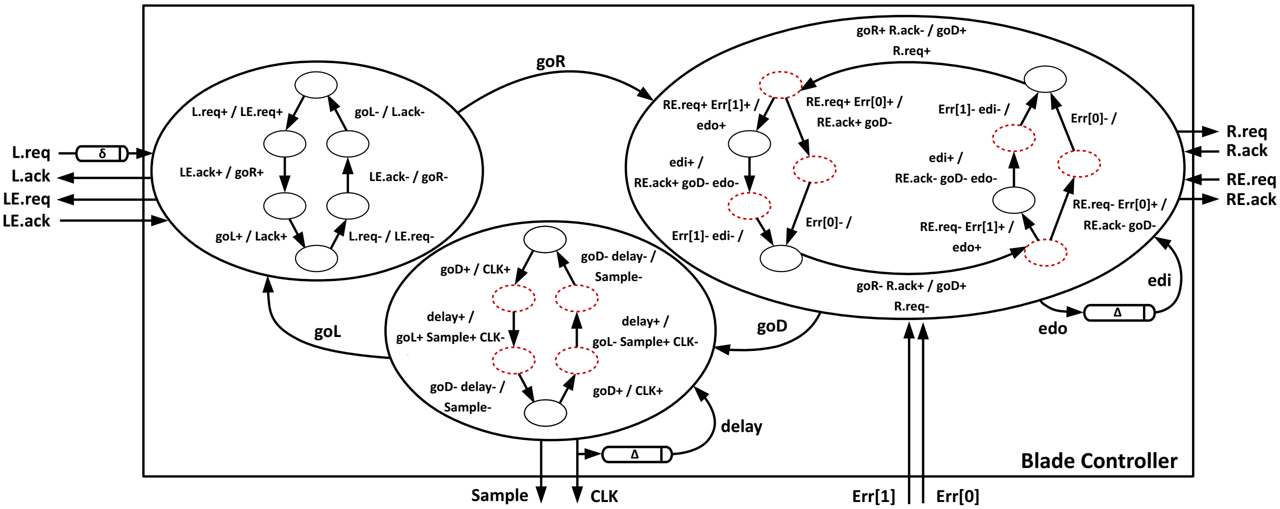

Fig. 3: Burst-mode state machine for Blade controller highlighting states that may be affected when the EDL has a fault, adapted from [7]

III. Simulation Environment

In order to evaluate the testability of the EDL, we created a behavioral simulation environment that allows controlled fault insertion at specific locations of the circuit, such as forcing wires to fixed values to produce a stuck-at or to modify the propagation delay of logic gates or other elements of the circuit to simulate a delay fault. It is also possible to simulate the circuit with a timing violation (TV) at the EDL's input. This means that the combinational takes longer than the δ delay to propagate the correct data. As discussed in the following sections, the ability to generate a timing violation is essential to detect some specific faults in the EDL.

The test scenario consists of a three stage Blade pipeline, where all three stages are identical regarding the timing constraints. Between each stage there is a string of inverters acting as the combinational logic. The simulation generates a log file informing the timing violations found on each stage. Another log file describes the injected data pattern at the first stage and the output data received at the last stage. From these logs the environment extracts the results presented in the next section.

The environment only simulates a single fault at a time, and these faults are always inserted at the middle stage (the second one). The other two stages are fault-free, and they are used to observe how the overall circuit behaves when one of the stages is faulty. The accepted faults are *stuck-at-0* (SA0), *stuck-at-1* (SA1), *shorter propagation delay* (SPD) and *longer propagation delay* (LPD). Stuck-at faults are inserted at all wires inside the error detection logic and the propagation delay fault is injected in all logic gates. We assume that data is not masked by the combinational logic, and a data transition is always propagated from one stage to another. In the following discussions the use of *error* indicates that a timing violation (not the stuck-at or propagation delay fault) was detected by the error detection logic. An TV within the resilience window must cause an error in the EDL, unless the EDL is faulty.

IV. Fault Analysis

In following sections the faults are individually analysed. As we move forward with the analysis we discuss the effects that can be observed and how they can be applied to the fault detection. For the detection of some particular faults, a timing violation (TV) must be generated to stimulate parts of EDL that are only activated in such situations. By default we assume that a TV is not required to detect the fault. However, when the TV is necessary, it is explicitly referred and discussed in the text.

A. Stuck-at Fault Analysis

The first fault to be analysed is an SA0 at $w1$. In this case, the pipeline halts waiting for the *delay* signal to go up. The same will happen with an SA1 in the this wire, but instead the pipeline halts waiting *delay* signal to go down. Both faults are easily detectable with a functional test due to a pipeline halt. An SA0 and an SA1 at $w2$ makes the latch to always capture a constant data due to the stuck value in its input. These faults are *detected at the output of the next stage*.

The next fault point is $w3$. An SA0 or an SA1 at this wire can cause a false error generation that depends on the input data. For instance, if $w3$ is SA0, the fault is detected only if the input data is at logic level 1. It means that the input data must be controllable and the signals *Err1* or $w7$ must be observable. For the rest of this analysis *Err1* is considered the observability point.

An SA fault at $w4$ differs from a fault at $w1$ because the former does not affect the latch. An SA1 at wire $w4$ always produces an error at the faulty stage, while an SA0 causes the stage to never detect a timing violation, since C-element clock never goes up. As observed in the part two of the experiment, to detect this SA0 fault a TV must be generated at the stage under test and the *Err1* signal of the next stage observed. This is possible due to a particular feature of Blade's template, where the timing violation missed at a stage, in this case because of a stuck-at fault, can still be detected in the following stages. Since the faulty stage does not capture the error, the delay is not extended as it would be expected in such situation, and an invalid data is propagated to the next stage. The following stage eventually detects the TV and delays its latch opening, such that its following stage receives the correct data. This and some other faults are defined as *detectable in the next stage*.

Another fault point is $w5$, but this fault analysis can also be extended to $w6$ and $w7$. An SA0 at these nets cause the EDL to not detect timing violations. As in the previous analysis, a TV must be inserted so that the fault is detectable at the next stage by observing its *Err1* signal. The SA1 is detected at the faulty stage, since an error is always signaled by *Err1*. The next signal, $w8$, is the Q-Flop *Sample* signal. The Blade controller requires that the signals *Err0* or *Err1* must go up and later go down before a new protocol cycle is initiated. So, if $w8$ is SA0, both error signals are also at logic level 0, causing the pipeline to halt. The SA1 has the same behavior but in the opposite direction, while the SA0 halts the pipeline because *Err0* or *Err1* never goes up, the SA1 halts the pipe because the signals never goes down. Both faults are *detectable with a functional test* due to a pipeline halt.

As in the previous analysis, the next fault is also detected by a halt in the pipeline. To detect an SA0 at $w9$ a TV must be generated in the faulty stage. In this case the *Err1* should go up but, because of the fault, it never goes up. On the other hand, the SA1 at $w9$ can be detected with a functional test when the pipeline halts waiting for *Err1* to go down, the same for an SA1 at $w11$. The detections of an SA0 at $w11$ is similar to $w9$, except that a TV is not required in this case. An SA0 at $w10$ can be detected with a functional test once the pipeline halts waiting for *Err0* signaling. On the other hand, to detect an SA1 fault, a TV must be injected at that the data input. The last fault point is $w12$ where, for both SA0 and SA1, the pipeline halts waiting for *Err0* to go up or down, respectively, and they are both testable with functional test.

B. Delay Fault Analysis

As mentioned earlier, the propagation delay of gates, latch and delay lines is considered for the delay fault analysis. The faulty gate has either an SPD fault or an LPD fault, compared to the design timing constraints. An SPD fault, except for the delay lines, does not represent a threat to the overall circuit operation. For example, looking at the C-element or the following OR gate, an SPD at any one of them will increase the TRW presented in Subsection II-C. With a bigger TRW, the faulty stage catches timing violations that otherwise would not be captured by the EDL. These group of faults that don't affect the circuit functionality can be defined as *don't care* faults. The Q-Flop and the following OR gate and AND gate in the presence of an LPD fault also don't produce any catastrophic failure to the circuit, but these faults can be detected by looking at the performance degradation of the pipeline, since the late propagation of signals *Err0* or *Err1* would delay the completeness of the protocol cycle.

Even though this analysis does not consider the datapath testability, an LPD fault in the latch can be seen as a datapath delay fault. In this case a timing violation is detected at the next stage, and a high error rate at a pipeline stage would suggest that there is a delay fault in the datapath. Unlike SPD, an LPD at the C-element and the following OR gate reduces the TRW, and it affects directly the circuit resiliency to timing violations. In particular, this fault causes the circuit to behaves differently depending on the moment that the violation occurs. For example, looking at the timing diagram of Figure 4, the $t_{CE,pd}$ can be so long that the last *X* pulse is not propagated until the falling edge of the *CLK*, thus missing the TV. In this case the violation is propagated to the next stage. With the same propagation delay fault in $t_{CE,pd}$, if this last *X* pulse appears right after the rising edge of *CLK*, then the violation would still be captured at the faulty stage. This demonstrates that the TV must be inserted at specific lines and at specific moments inside the TRW.

The next fault is an LPD at the XOR gate. Using the timing diagram of Figure 4 as reference, this fault dislocates the *X* pulse inside the TRW by increasing the $t_{X,pd}$, which produces a false error that is dependent on the input data, like the analysis of a stuck-at fault at $w3$. Now looking at the t_{TD} delay line, if an SPD fault occurs at this element, the *X* pulse width ($t_{X,pw}$) becomes smaller. This fault harms the circuit operation if the pulse gets smaller than the setup time of the C-element, otherwise the fault is a *dont't care* fault. In the case where the pulse width is smaller than the setup time of the C-element an error is missed at the faulty stage, but the next stage detects the error. A method for injecting a TV is necessary to detect this last fault. Similarly to the XOR gate LPD fault, an LPD at the t_{TD} delay line generates a false error that depends on the input data, with the difference that instead of shifting the *X* pulse into the TRW, $t_{X,pw}$ increases the pulse width that grows into the TRW.

The last delay fault analysis is at *tcomp*. An SPD at this delay element reduces the compensation delay added to ensure that *X* is not captured before the rising edge of *CLK*. In this case the C-element captures *X* pulse before the correct time, which generates a false error. This fault can be dependent on the input data, although it is unlikely that all inputs are always capturing the same logic value. If at least one data input changes every clock cycle, an error is always observed at *Err1*. An LPD at *tcomp* causes the C-element to late capture the *X* pulses. Comparing with the timing diagram of Figure 4, this fault dislocates the *CLK* to the right, which can be seen as a decrease in the TRW, and some timing violations may not be captured by the EDL. Like the LPD fault at the C-element, to detect this fault, a TV must be inserted at the beginning of the TRW. If this fault is present, the error is not captured at the faulty stage, only in the next stage. If the TV is inserted at the end of the TRW, an error is flagged at the faulty stage, which is the expected behavior of a fault-free stage, thus the fault is not detected.

C. Discussion about the fault effects on the EDL

As noted in the previous fault analysis, a faulty EDL triggers different effects in the overall circuit that can lead to a fault

TABLE I: Relevant effects for the fault classification.

Acronym	Description
UN	undetectable
PST	pipeline output stuck at a value
PH	pipeline halted
ERR_ST	errors in the faulty stage
ERR_NST	errors in the next stage

detection. Some of these faults only affect the performance of the circuit. This behavior can either be caused by a high rate of false errors or due to an LPD at a gate that does not prevent the EDL from detecting timing violations, such as the *AND Gate*. In both cases, these slower circuits can still be commercialized at a lower cost, since they are fully functional.

Some fault effects impact directly the circuit resiliency by either completely disabling the EDL or increasing/decreasing the TRW. Similar to the performance problem, an EDL that has its TRW increased by a fault is still functional and it can detect longer timing violations compared to the expected EDL behaviour. When the TRW is reduced, the EDL may not capture all the timing violations or, in the worst case, it may miss all the timing violations when the EDL is disabled by a fault, such as an SA0 at the Q-Flop sample signal. The detection of these faults is critical, once the faulty EDL propagates invalid data to be processed by the following combinational logic, which can lead to a major system failure.

The faults called *detectable in the next stage* can only be detected right in the next stage if the TRW of the next stage is equal or longer than the TRW of the stage under test. Otherwise, when the TRW of the next stage is shorter, the timing violation can be propagated beyond the next stage.

V. Fault Classification

So far the faults were discussed individually, looking at the side effects observed in the overall operation of the circuit and how each one can be detected. Now we present a fault classification that generalizes the relationship between cause and effect of the analysed faults. Table I shows the fault effects observed during the circuit simulation that are relevant to the fault classification.

Based on the individual analysis of the faults, we correlate the cause, which is the fault simulated, and the effects listed in Table I. The result of this is a fault classification that groups faults with similar effects. Tables II and III show the classification for the stuck-at faults and the propagation delay faults, respectively. Both tables present the classification assuming that a method to inject timing violations is available (*w/ TV*) and without (*wo/ TV*) this method. It is possible to see that without the TV some faults are undetectable (UN). *This demonstrates that the ability to control the injection of timing violations is important for the testability of the EDL.*

The missing items in Table III represent the faults described in the analysis as *don't care*. Especially for this classification, the LPD faults in the OR Gate (*Err1*) and the AND Gate (*Err0*) are classified as *don't care*, since the circuit operates as expected, but with a lower performance.

TABLE II: Classification for stuck-at fault model. Timing Violation (TV).

Faulty	SA0		SA1	
Line	wo/ TV	w/ TV	wo/ TV	w/ TV
$w1$	PH	PH	PH	PH
$w2$	PST	PST	PST	PST
$w3$	ERR_ST	ERR_ST	ERR_ST	ERR_ST
$w4$	UN	ERR_NST	ERR_ST	ERR_ST
$w5$	UN	ERR_NST	ERR_ST	ERR_ST
$w6$	UN	ERR_NST	ERR_ST	ERR_ST
$w7$	UN	ERR_NST	ERR_ST	ERR_ST
$w8$	PH	PH	PH	PH
$w9$	UN	PH	PH	PH
$w10$	PH	PH	UN	PH
$w11$	UN	PH	PH	PH
$w12$	PH	PH	PH	PH

This fault classification guides the next steps towards the design for testability of the EDL. As it is presented in Table IV, the three approaches were evaluated in terms of fault coverage of the 32 possible faults (*don't care* faults are not accounted). Functional test would be the first alternative. The fault coverage would be of 34%, the smallest coverage among the three. The next approach assumes a scan cell at the *Err* signals of each stage to enhance its observability. Another alternative would be to make the Q-Flops of each stage scannable. On the other hand, the first approach is preferable because using scan cells at the border of EDL and the controller could also help to improve the controlability of the controller. The observed fault coverage for this second approach is 66%. To obtain a fault coverage of 100% it is necessary to include a timing violation generator to fully exercise the error detection logic. As previously demonstrated, some faults are only observed when the circuit has a timing violation and, in some particular cases, the timing violation must occur at specific moments (at the end of the TRW) of the circuit operation. A glitch generator could be used to exercise the EDL's logic, but this is still an open research topic that must be further investigated.

VI. Conclusion

This paper addressed the testability of the error detection logic of the Blade template. A fault classification based on the relationship of cause and effect of faults in the overall circuit operation was proposed. The fault analysis and classification was used to propose three preliminary test methods and their respective fault coverage for stuck-at and delay fault models. It is shown that some faults can completely disable the error detection capability of EDL, allowing the propagation of timing violations. Without a testing approach it is not possible to assure that the EDLs are going to perform as they were designed and whether the actual resilience windows match the specification. The analyses show that, to achieve a fault coverage of 100%, design for testability circuitry must be added into the design flow, such as including scan cells to increase the observability and controlability, and a method to

TABLE III: Classification for propagation delay fault model. Timing Violation (TV).

Faulty Element	**SPD**		**LPD**	
	wo/ TV	**w/ TV**	**wo/ TV**	**w/ TV**
Latch	-	-	ERR_NST	ERR_ST / ERR_NST
t_{comp}	ERR_ST	ERR_ST	UN	ERR_ST / ERR_NST
t_{TD}	UN	ERR_NST	ERR_ST	ERR_ST
XOR Gate	-	-	ERR_ST	ERR_ST
C-element	-	-	UN	ERR_NST
OR Gate	-	-	UN	ERR_NST
Q-Flop	-	-	-	-
OR Gate (Err1)	-	-	-	-
AND Gate (Err0)	-	-	-	-

TABLE IV: Fault coverage obtained per test approach. (*) Fault Detected.

Fault		**Functional**	**Scan Chain**	**TV Gen.**
Type	**Wire/ Element**			
SA0	$w1$	*	*	*
	$w2$	*	*	*
	$w3$		*	*
	$w4$			*
	$w5$			*
	$w6$			*
	$w7$			*
	$w8$	*	*	*
	$w9$			*
	$w10$	*	*	*
	$w11$			*
	$w12$	*	*	*
SA1	$w1$	*	*	*
	$w2$	*	*	*
	$w3$		*	*
	$w4$		*	*
	$w5$		*	*
	$w6$		*	*
	$w7$		*	*
	$w8$	*	*	*
	$w9$	*	*	*
	$w10$			*
	$w11$	*	*	*
	$w12$	*	*	*
SPD	t_{comp}		*	*
	t_{TD}			*
LPD	*Latch*		*	*
	t_{comp}			*
	t_{TD}		*	*
	XOR Gate		*	*
	C-element			*
	OR Gate			*
Coverage		34%	66%	100%

inject timing violation at the EDL input. We expect that these results would motivate further research on the test of other resilient circuits such as, for instance, bubble razor.

Acknowledgment

The authors would like to thank Peter A. Beerel and Matheus T. Moreira for the discussions about testability issues on Blade. Alexandre would like to thank CNPq for the financial support (process 460205/2014-5).

References

[1] M. Choudhury, V. Chandra, K. Mohanram, and R. Aitken, ”Timber: Time borrowing and error relaying for online timing error resilience,” in DATE, pp. 1554-1559, Mar. 2010.

[2] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, ”Razor II: In situ error detection and correction for PVT and SER tolerance,” IEEE JSCC, vol. 44, no. 1, pp. 32-48, Jan. 2009.

[3] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. 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,” IEEE JSCC, vol. 48, no. 1, pp. 66-81, Jan. 2013.

[4] I. E. Sutherland, ”Micropipelines,” Commun. ACM, vol. 32, no. 6, pp. 720-738, Jun. 1989.

[5] I. J. Chang, S. P. Park, and K. Roy, ”Exploring asynchronous design techniques for process-tolerant and energy-efficient subthreshold operation,” IEEE JSSC, vol. 45, no. 2, pp. 401-410, Feb. 2010.

[6] N. Jayakuma, R. Garg, B. Gamache, and S. Khatri, ”A PLA based asynchronous micropipelining approach for subthreshold circuit design,” in DAC, 2006, pp. 419–424.

[7] D. Hand, M. T. Moreira, H.-H. Huang, D. Chen, F. Butzke, Zhichao Li, M. Gibiluka, M. Breuer, N. L. V. Calazans, and P. A. Beerel, ”Blade – A Timing Violation Resilient Asynchronous Template,” in ASYNC, pp.21-28, May 2015.

[8] K. Bowman, J. Tschanz, N. S. Kim, J. Lee, C. Wilkerson, S. Lu, T. Karnik, and V. De, ”Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance,” IEEE JSCC, vol. 44, no. 1, pp. 49-63, Jan. 2009.

[9] F. Rosenberger, C. Molnar, T. Chaney, and T.-P. Fang, ”Q-modules: internally clocked delay-insensitive modules,” IEEE Trans. on Computers, vol. 37, no. 9, pp. 1005-1018, Sep. 1988.

[10] R. Fuhrer, B. Lin, and S. Nowick, ”Symbolic hazard-free minimization and encoding of asynchronous finite state machines,” in ICCAD, pp. 604-611, Nov. 1995.