Study and Comparison of QDI Pipeline Components' Sensitivity to Permanent Faults

Raghda El Shehaby and Andreas Steininger
TU Wien, Institute of Computer Engineering, Vienna, Austria
relshehaby@ecs.tuwien.ac.at, steininger@ecs.tuwien.ac.at

***Abstract*—In the presence of permanent faults, QDI circuits exhibit the beneficial property of halting their operation until a repair procedure has been conducted. The state in which the circuit resides, however, does not always remain clean, i.e., a recovery process might be needed. This depends on how the circuit reacts in these situations. In this study, we investigate the effect a permanent fault has on the different components of the pipeline, the logic function unit and the buffer. Our aim is to identify the weaknesses of each component and try to enhance each one accordingly. We perform extensive fault-injection simulations on different circuits following the famous 4-phase communication protocol, while varying the logic function and buffer style for comparison. Our results show that the logic function does not affect the resilience of a specific buffer type, and hence we can deduce which buffer should perform better for a specific application based on parameters we extract from our experiments. On a parallel note, the implementation style of the logic also has an impact on the block's ability to hold out against faults. We investigate two of these styles.**

***Index Terms*—permanent faults, QDI circuits, WCHB, fault injection**

I. Introduction

Asynchronous design is known for its potential to save dynamic power and to obtain high speed as compared to the widely used synchronous design. Having a flexible handshake (instead of a rigid clock) govern the communication between individual units also provides good robustness against process/temperature/voltage (PVT) variations – a very desirable feature in modern process technologies. Much less known is the fact that asynchronous circuits (specifically the so-called delay insensitive ones) also exhibit a self-checking behavior in the sense that any stuck-at fault in their logic units will make them deadlock. This is not only relevant for testing, it also facilitates recovery: while a synchronous circuit keeps on computing after a permanent fault occurred, thus polluting its state, an asynchronous one simply stops operating, ideally in the last correct state.

When it comes to self-repair, asynchronous circuits can again leverage their intrinsic elegance: re-routing a signal to bypass a defective gate is easily (and automatically) tolerated by the flexible timing obtained through the closed-loop handshaking, where in synchronous designs most often timing issues will be encountered. So in the ideal case mitigating a permanent fault in an asynchronous circuit is as simple as bypassing the defective gate – the circuit will continue processing from the last correct state, with the timing automatically adapting to the new delays. With this vision in mind, we are investigating, how reliably the desired scenario of a *fail-stop* will indeed be encountered after a permanent defect in an asynchronous circuit and how undesired effects can be mitigated. This paper specifically will be dedicated to comparing different pipeline buffer styles and logic implementation styles with respect to this property.

This research was partially supported by the project ENROL (grant I 3485-N31) of the Austrian Science Fund (FWF) as well as the Doctoral College on Resilient Embedded Systems (DC-RES)

II. Background on Asynchronous Logic

With the use of local handshakes between two communicating parties, asynchronous circuits transfer data without the need for a rigid time grid. This handshake provides a closed-loop control, instead of capturing data at specific periodic clock ticks, which specifies when the sender (or *source)* has provided a new data item (also referred to as *token*), and when the receiver (or *sink*) has processed it. This makes computation in asynchronous circuits data-driven. The *Delay-insensitive (DI)* class resides on top of the stack of asynchronous circuits, being the most flexible in timing because they automatically adapt to gate and wire delays. The more realistic, and very robust, member of asynchronous circuits is *quasi delay-insensitive (QDI)*, imposing only the *isochronic fork* constraint[1].

A handshake cycle depends on the communication protocol being used. The simplest, and most commonly used, method is the *4-phase*, or *return-to-zero (RTZ)*, protocol with the dual-rail data encoding scheme. A *completion detection (CD)* unit is needed to indicate validity and completeness of a token. This is a simple OR gate in the case of a single bit, and requires additional logic for higher bit-widths. An explicit *ACK* signal is used to switch between the two phases of the protocol, a *spacer* or *NULL* phase and a *DATA* phase. Each dual-rail bit is represented using two wires, namely the *true* and *false* rails. Each bit can have only one rail set to logic '1' at a time. After every DATA phase, both rails become logic '0', which is the SPACER carrying absolutely no information. It is forbidden to have both rails set to '1', this constitutes an *illegal* codeword.

A sequence of components where data travels between them form an asynchronous pipeline. It is divided into stages where buffers separate logic function units. A fundamental building block in such a pipeline is the *Muller C-element (MCE)*. It can be viewed as the simplest form of a storage element, as well as it is used in the CDs and is sometimes used in controlling the buffers. It is an essential component in asynchronous circuits

[1]The delays of the individual paths after a fork must be equal, i.e., a signal must arrive at all ends of the fork at the same time [1].

which operates like an AND gate with hysteresis: it sets the output to the corresponding matching inputs value, and retains its previous logic level when inputs don't match. A very common buffer template adhering to the 4-phase protocol is the *weak-conditioned half-buffer (WCHB)* which uses 1 MCE for each bit as its storage element. When the buffer is in its NULL phase awaiting new data, all the MCEs are armed for rising transitions an the data rails. When the token is received and acknowledged by the downstream stage, the MCEs are disabled.

III. Related Work

Our work focuses on investigating the effects of permanent faults in asynchronous circuits which very few studies explore. We know from literature that DI circuits are always self-checking under the stuck-at fault (SAF) model, causing the circuit to halt [2]. This work, as well as [3] and [4], investigate the circumstances under which the circuits stops. With a focus on QDI circuits, the authors in [5] perform a thorough analysis of SAF effects, among other types of faults. The fault's impact is eventually translated to a deadlock. [6] proposes a formal method to exhaustively analyze all possible behaviors of a QDI circuit under single-event upsets (SEUs) using symbolic simulation. Even though the study focuses on transient faults, much can be conveyed to permanent faults. In addition to the deadlock effect, [7] shows that a SAF can also result in isochronic fork violation, token generation or token consuming. In case of permanent faults, reaching some kind of fail-safe deadlocking behavior is considered favorable to a continued, erroneous operation of the circuit, as is identified in [8], [5], and [9]. From a more experimental perspective with quantitative statements on the behavior of QDI circuits under faults, the authors in [10] use a special visualization scheme to compare the fault tolerance of the classical WCHB against other variants of the template. The authors also propose two enhancements to try to mitigate the faulty behavior of the half-buffer, namely deadlocking and interlocking WCHBs, which use cross-coupled asymmetric MCEs. Their analysis involves only transient faults. In [11], the authors provide varying probabilities for the fail-stop behavior with a set of experiments, and they take a close look at the analysis of the 11-illegal state of the dual-rail 4-phase communication protocol. [12] leverage a fully automated setup for performing fault injection simulations in QDI circuits. Their results show the robustness of a set of buffer styles under transient faults. These works give us a valuable foundation to understanding more about the impact of faults on the behavior of asynchronous circuits. This can help us make quantitative predictions for a given parameter set which is representative enough instead of exploring the huge design space, and hence we can move forward with more complex circuits. It is worth mentioning that a recent survey [13] about the challenges of QDI circuits' tolerance to soft errors. The author discusses mitigation and hardening schemes for QDI circuits, providing a comprehensive overview of the existing techniques and a comparative analysis of their value and limitations.

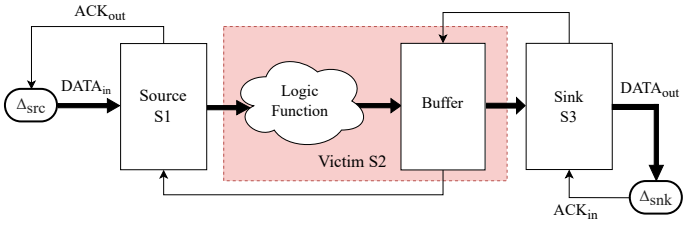


Fig. 1: Target Circuit

IV. Experimental Setup

Our aim is to provide a basis for comparison between different QDI pipelines styles in order to evaluate their robustness. As mentioned in literature, the results from recent works focus on transient faults where the parameter space is not exactly the same. We explore the QDI design paradigm space by implementing different styles and run experiments where parts of these circuits are subjected to faults, while taking into consideration the same operating conditions. We investigate the types of erroneous behavior and we provide a comparison on the resilience of the circuits to permanent faults, breaking down the circuit to its two main components, namely the logic function unit and the buffer.

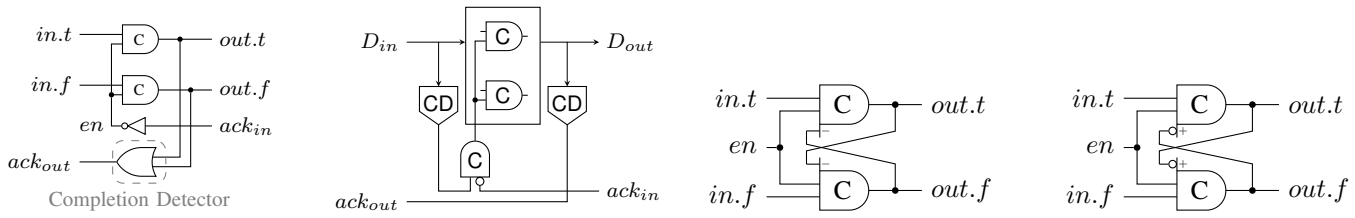
A. Target Circuit

When trying to understand how a circuit reacts under faulty operation, testing large-scale, complex circuits which require excessive computational performance to allow adequate fault coverage, comes with a huge time and resource overhead. Meanwhile, there doesn't seem to be consensus over a representative function or application for all kinds of asynchronous circuits. Our circuit in figure 1, therefore, consists of an elastic pipeline which resembles the delay-insensitive behavior of an asynchronous pipeline very well, while varying the following set of parameters:

- *Buffer Style*: Focusing on the 4-phase communication protocol, we examine the widely used classic WCHB with one completion detection, a WCHB with two CDs [14], in addition to a couple of variants introduced in [10] using asymmetrical C gates, as shown in figure 2.
- *Logic Function*: We select the functions of an adder and a multiplier, since these structures have been considered representative in many other works as well [5], [9], [12], while also being part of larger practical applications.
- *Logic Implementation Style*: We implement our functions using delay-insensitive minterm synthesis (DIMS) [15] and NCLX [16], both following the QDI design style.
- *Data Width*: 1- and 2-bit datapaths for a variable number of inputs.

B. Tool

In order to be able to conduct our experiments, the work is divided into two main stages, (i) we need to synthesize our circuit into the QDI design paradigm and (ii) we need to perform the fault-injection simulations while managing the fault space. The tool set introduced in [12] and [17] conveniently covers both tasks as shown in figure 3.



(a) WCHB (b) DualCDWCHB (c) Deadlocking WCHB (d) Interlocking WCHB

Fig. 2: WCHB-based buffers

The target circuits are described using *production rule sets (PRSs)* [18] which is a low abstraction description method for asynchronous circuits, equivalent to a transistor netlist. The Python-based custom tool takes in a simple language specification of the circuit and constructs the PRS-based representation with 10% randomly-varied inertial delays to model process, voltage and temperature (PVT) variations in the circuit. Our test targets are created by modifying parameters that can be quickly manipulated, to switch between the data-width and the implementation style of the combinational logic part of the circuit, and the buffer styles of the storage elements. The tool also generates a VHDL model of the circuit, employing the same delays, and includes a testbench generator. The circuit and the testbench, both described in VHDL, serve as input to the next stage.

For the simulation part of the process, *simulation tasks* are added to a central SQL database. Each task entails simulating exactly one fault, in one location, at a specific time in the circuit. These fault-injection experiments are executed using the ModelSim HDL simulator. Each processing core used for the simulations regularly looks for the added tasks in the database waiting to be processed.

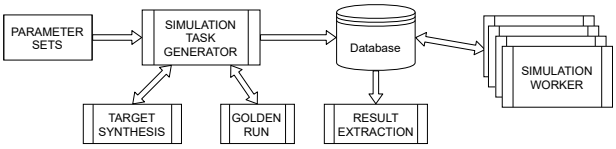


Fig. 3: Overview of the toolset. Source [17]

C. Extending The Tool

While the fault-injection tool covers many simulation parameters, it is targeting transient faults. Our focus is on permanent faults and, hence we use the stuck-at fault (SAF) model, which has established itself as the de-facto standard in permanent fault modeling. A SAF permanently fixes a signal at either logic *high* (stuck-at-1 or SA1) or logic *low* (stuck-at-0 or SA0). We, therefore, do not need to consider all the parameters, such as the injection duration. We also need to devise a method to judge whether the circuit resides in a benign state, i.e., the fault has caused only a halt of the circuit and in case of repair, normal operation will resume with all tokens reaching the output side correctly.

Our fault injection parameter space includes:

- Fault Polarity: SAFs of both polarities, i.e., SA0 and SA1.
- Fault Location: All inputs and outputs of the gates of a chosen victim stage.
- Speed of Source and Sink: The reaction times T_{source} and T_{sink} are varied to cover both a fast and slow environment.
- Time of Injection: All potential time intervals where a fault might have an effect are covered.

We adapt the tool to inject in all inputs and outputs of our chosen victim components of the circuits, as this maps to the transistor at input or output getting defective and not the wire.

As for the manipulation of the delays of source and sink, it results in two modes of operation, namely token-limited and bubble-limited. In the former mode, the ACK signal arrives faster than there is a new token. In the latter, an available token is ready to be pushed into the pipeline which is still waiting the old data to be captured. We use absolute values for covering this parameter because we have learnt from our experiments that not only the relative speed T_{source} vs T_{sink} has an impact, but also the absolute speed. This further enables us to choose the granularity of presenting the results.

We also do not inject randomly, but exhaustively, in all points considered relevant to our scope. The injection time between experiments is incremented with a step smaller than the smallest gate delay. This yields a very dense coverage for this continuous parameter while still having manageable space.

In every simulation, we let the circuit run for a number of handshake cycles on the output side such that it gets into operation mode. Then we inject our fault and wait for the system to exhaust all of its scheduled transitions until it reaches deadlock. We then remove the fault and allow the system to run for completion. This is an objective way of deciding on the correctness of the state after the fault-induced deadlock. Using an identical, fault-free pipeline for the evaluation of correct behavior, we compare the results of each experiment to this *golden reference* and we record the effects accordingly.

V. Results & Analysis

In literature, we find different terminology for fault effect classification on different abstraction levels. We start by explaining what an incorrect behavior of our circuits entails and how we further distinguish these classes.

When a circuit is hit by a fault, and this fault is not masked, normal operation varies from that of the golden run. Any occurring delay is tolerated by definition of a QDI circuit, and there is no need to consider timing information. We did, however, collect them as a means of validation of our results since a permanent fault will always cause a timing deviation from the golden run. We classify the types of failure as follows:

- *Coding Error*: While one of the rails of a signal is correctly set to '1', a fault can change the other rail of the same

TABLE I: Incorrect Behavior over Injection Count(%) after Unfreezing the Pipeline

Circuit \ Buffer	WCHB		Interlocking		Deadlocking		Dual-CD	
	Token Limited	***Bubble Limited***	***Token Limited***	***Bubble Limited***	***Token Limited***	***Bubble Limited***	***Token Limited***	***Bubble Limited***
Empty Pipeline 1bit	15	25	13.5	11.5	12	21	6	21
Empty Pipeline 2bit	12	24	10.5	12.5	10	20	6	19
Adder DIMS 1bit	16.5	24.5	16	13.5	15.5	22.5	11	20.5
Adder DIMS 2bit	15	22	12.4	9.5	14.5	21.9	15.5	19.8
Adder NCLX 2bit	11.5	17.9	10.4	11.4	11	17.1	14	17
Multiplier DIMS 2bit	11.5	17.5	9.8	10.8	10.8	16.5	13.5	17.2
Multiplier NCLX 2bit	11	18	9.5	12.5	10.5	16.9	13	17

signal to '1' as well, violating the 4-phase protocol, and thus producing an illegal codeword.

- *Value Error*: A fault can cause a token to change its value by switching the wrong rail of a signal, leading to an undetected different value.
- *Clean Deadlock*: A fault might cause neither an illegal nor a wrong-value token, yet the pipeline remains in deadlock even after the fault is removed, that's why we refer to this deadlock as *clean*.

According to our data collection, a *coding* error and a *value* error can appear in the same simulation, where a specific fault was injected, in different points in time along the operation of the pipeline. In our setup, we consider that the appearance of an illegal codeword as having the highest severity; so a value error will be categorized as such only when no coding error has been recorded for a given simulation while a legal data value, different from the one propagating through the golden run, was delivered to the output side.

Note that another dimension of the effect classes is the number of tokens going in and out of the pipeline. We measure the count of *token generation* and *token consumption*. *Coding* and *value* errors fall under the category of *token modification*, which results have shown to be the most dominating among these three effects.

A. Fault Effect Distribution

We have conducted millions of simulations on different logic functions and buffer variations. Figure 4 shows the results of an empty pipeline, an adder and a multiplier, both implemented in DIMS and NCLX styles, and all of a bit width of 2. For each circuit, we can see the performance of each of our four buffers, with the Interlocking WCHB steadily showing the lowest percentage of incorrect behavior for all circuits. This is further broken down in figure 5 where the causes of this incorrect behavior are presented. The most dominating failure type is the *coding* error reaching over 80% in most of the circuits. Thanks to the mechanism of the Interlocking buffer of allowing only the first rail of a signal to transition to '1', regardless of whether it is the right one, figures 5a and 5b show how the Interlocking buffer eliminates all the illegal codewords, converting part of them to value errors.

It is not possible to show all parameters in each plot, so figures need to be compressed in order to visualize the effects. Table I shows the effect of varying the reactions times of source

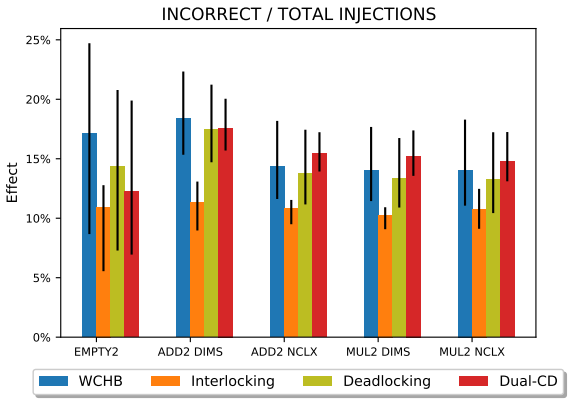
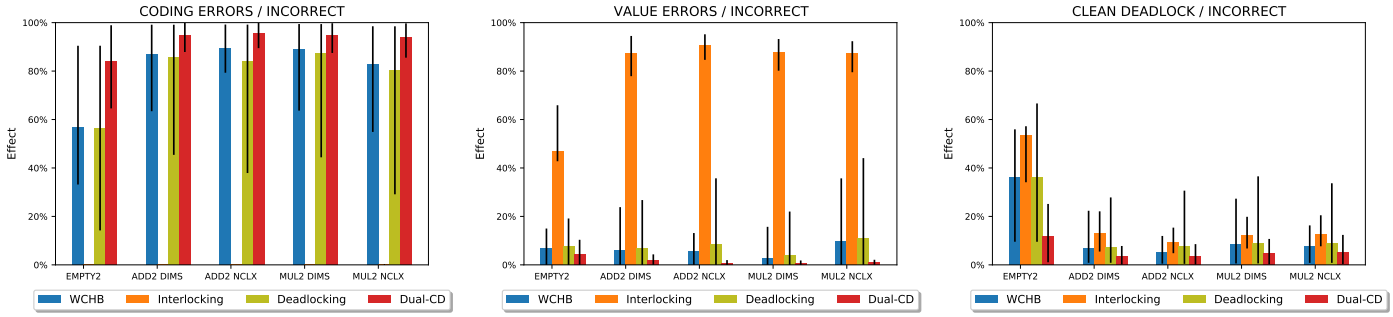


Fig. 4: Total Incorrect

and sink. The numbers present the corner case percentages of the erroneous behavior of the circuits, when allowed to run with a maximum difference between speeds of source and sink.

In another experiment, we compare between the empty pipeline and the adder with DIMS implementation style, each constructed with 1- and 2-bit datapaths. We investigate the details of these small-scale circuits to build a strong foundation for understanding larger circuits. Figure 6 shows the percentage of incorrect behavior with its sub-effects distribution. The percentages shown are calculated for all injections per buffer, sweeping over source and sink delay combinations. Once more, the interlocking WCHB has the best performance, while the WCHB reaches 18% of malicious behavior. Having lighter shades of green, the 2-bit adder seems to show better performance. This might not necessarily be the case: a larger circuit yields a larger area with more signals as victims to the fault injection, and while the absolute number of *hits* might have gotten higher, the set of signals with a higher sensitivity to faults might not have increased. We will take a closer look at which signals were more prone to error in the different circuits later on.

In addition, the circuit generated from the synthesis phase can differ slightly even when using the same logic style implementation. In dual-rail logic there is not always one unique way to implement a given function. This is something we are able to control in our own synthesis tool, but might not be able to do with a different synthesizer. Choosing between the designs is a question of optimization, which is out of the scope of this paper, however, this has helped us better understand the reaction



(a) (b) (c)

Fig. 5: Distribution of Incorrect Behavior

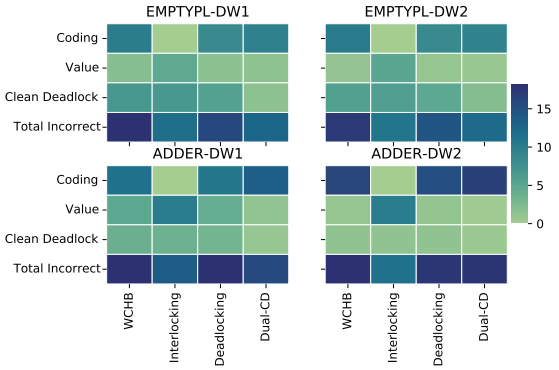


Fig. 6: Effects (average percentage)

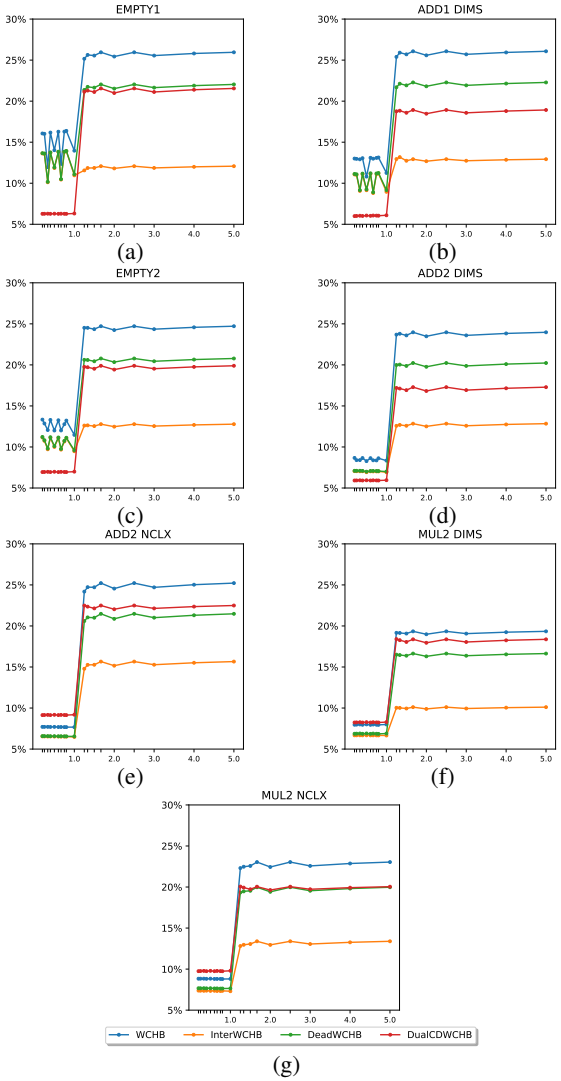
of certain circuits. It is worth noting that our experiments on data width comparison do not extend to larger than a 2-bit datapath mainly because of the state explosion problem of the fault injection space. Taking the example of the 2-bit adder yielding around 200 million injections resulting in a overhead in both time and memory for such a small circuit.

B. Pipeline Components Comparison

The QDI design space includes several parameters which can be investigated separately and tested together in a plug-and-play manner. Our hypothesis is that each one of these parameters does not substantially affect the other. To that end, and with a focus on the buffer style and the logic implementation style, we extracted the results concerning each one, respectively.

Among the challenges encountered when dealing with fault injection simulations is which logic function is representative enough to use and can the results from this circuit be genrealized to other combinations. With a focus on buffer robustness we experiment across combinations of different parameter settings, namely the logic circuit's function, implementation style and input data width. We take into account the system failures caused by the faults hitting the buffers only. For example, we investigate the WCHB resilience in an empty pipeline with an exhaustive set of data inputs, while comparing it to the results of an adder and a multiplier both implemented in DIMS and NCLX. Figure 7 presents the results of buffer-only-injected pipelines for all our circuit variations and our assumption is confirmed but with a margin of error.

Our final experiment aims at better understanding the weak points of a circuit by investigating which signals are the most



(a) (b) (c) (d) (e) (f) (g)

Fig. 7: Incorrect percentages caused by buffer injections

responsible for the malicious behavior. Figure 8 shows the example of multiplier circuit with both logic implementation styles. The top 5 signals of both *buffer (B)* and *logic(L)* are each presented in a subplot. The percentage for each signal also shows which error type is dominant. For DIMS and NCLX styles, the most sensitive buffer signals are the *enable*, the *ack* and the inputs and outputs of the buffer. This demonstrates that the completion detection signals are less prone to cause a

system failure when injected by permanent faults. This also explains why varying the data width results in an apparent higher performance since we are using ratios. As for the logic function in DIMS, the whole block would require protection. This is intuitive since DIMS style uses a large number of C elements which would store the error when hit by a fault. The NCLX logic blocks show a better performance than DIMS, but still no specific signal stands out as the culprit. The percentages shown in the top-right corners of the subplots reflect the proportion of incorrect state caused by the buffer and the logic, respectively, from the total incorrect of the circuit. These numbers show that we cannot judge the performance of the buffer only by changing the logic implementation style; our expectations are again validated.

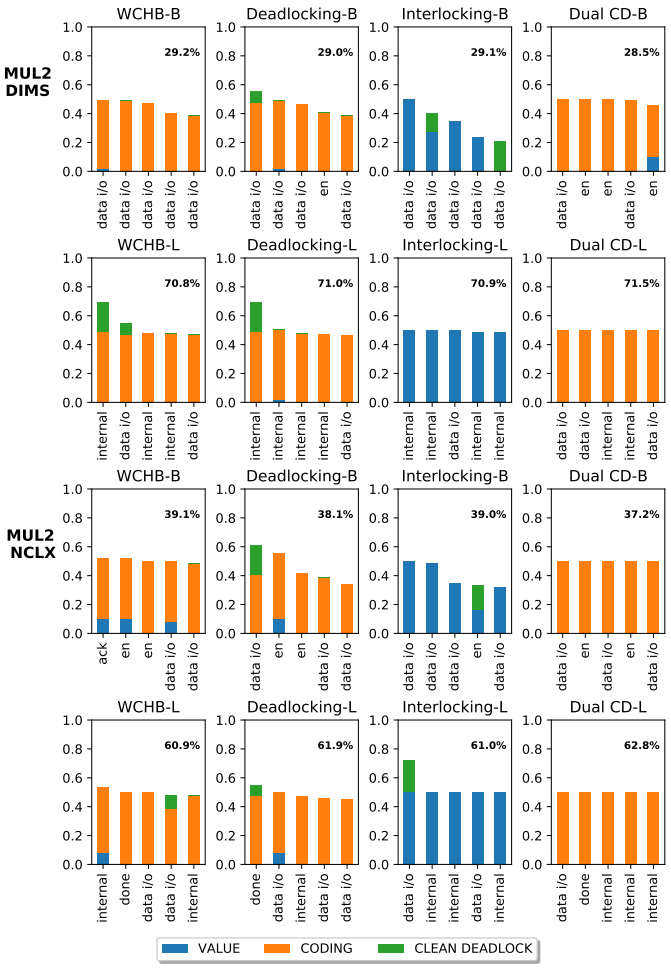


Fig. 8: Signals causing the most errors

VI. Conclusion & Future Work

Even though asynchronous circuits have a lot of potential to be used as an infrastructure in fault-tolerant applications, more research is needed in order to understand the behavior of the circuit under permanent faults. Our work examines the components of a QDI pipeline in depth. Among the buffer styles we used in our experiments, we have identified the interlocking WCHB as very robust, but the value errors it suffers from still need to be addressed. Having observed the completion detection to be relatively robust forms another argument to put focus on the data path. For the logic implementation style we have, as expected found some benefits of NCLX over DIMS, but our experience also was that implementation details may matter more here than just the choice of the style. In general, decomposing optimization of buffers and logic seems to be viable, as our results confirm.

Our next steps will be dedicated to devising mitigation techniques for those signals that turned out sensitive, and to establish efficient diagnosis for guiding a self-repair action.

References

[1] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *Beauty is our business*. Springer, 1990, pp. 302–311.

[2] H. Hulgaard, S. M. Burns, and G. Borriello, "Testing asynchronous circuits: A survey," *Integration, the VLSI journal*, vol. 19, no. 3, pp. 111–131, 1995.

[3] A. J. Martin and P. J. Hazewindus, "Testing delay-insensitive circuits," California Institute of Technology Pasadena Department of Computer Science, Tech. Rep., 1990.

[4] P. Liden and P. Dahlgren, "Coverage of transistor-level and gate-level stuck-at-faults in cmos checkers," in *Proceedings of ISCAS'95-International Symposium on Circuits and Systems*, vol. 3. IEEE, 1995, pp. 2124–2127.

[5] C. LaFrieda and R. Manohar, "Fault detection and isolation techniques for quasi delay-insensitive circuits," in *International Conference on Dependable Systems and Networks, 2004*. IEEE, 2004, pp. 41–50.

[6] Y. Monnet, M. Renaudin, and R. Leveugle, "Formal analysis of quasi delay insensitive circuits behavior in the presence of seus," in *13th IEEE International On-Line Testing Symposium (IOLTS 2007)*. IEEE, 2007, pp. 113–120.

[7] M. Zamani, H. Pedram, and F. Lombardi, "Templated-based asynchronous design for testable and fail-safe operation," in *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2011, pp. 146–152.

[8] S. J. Piestrak and T. Nanya, "Towards totally self-checking delay-insensitive systems," in *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*. IEEE, 1995, pp. 228–237.

[9] S. Peng and R. Manohar, "Efficient failure detection in pipelined asynchronous circuits," in *20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)*. IEEE, 2005, pp. 484–493.

[10] F. Huemer, R. Najvirt, and A. Steininger, "Identification and Confinement of Fault Sensitivity Windows in QDI Logic," in *Proceedings. Austrochip Workshop on Microelectronics 2020*. IEEE, 2020.

[11] R. El Shehaby and A. Steininger, "Analysis of state corruption caused by permanent faults in wchb-based quasi delay-insensitive pipelines," in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2021, pp. 63–68.

[12] P. Behal, F. Huemer, R. Najvirt, A. Steininger, and Z. Tabassam, "Towards explaining the fault sensitivity of different qdi pipeline styles," in *2021 27th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE, 2021, pp. 25–33.

[13] A. A. Sakib, "Soft error tolerant quasi-delay insensitive asynchronous circuits: Advancements and challenges," in *2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE, 2021, pp. 1–6.

[14] W. J. Bainbridge and S. J. Salisbury, "Glitch sensitivity and defense of quasi delay-insensitive network-on-chip links," in *2009 15th IEEE Symposium on Asynchronous Circuits and Systems*. IEEE, 2009, pp. 35–44.

[15] J. Sparsø and J. Staunstrup, "Delay-insensitive multi-ring structures," *Integration*, vol. 15, no. 3, pp. 313–340, 1993.

[16] A. Kondratyev and K. Lwin, "Design of asynchronous circuits using synchronous cad tools," *IEEE Design & Test of Computers*, vol. 19, no. 4, pp. 107–117, 2002.

[17] P. Behal, F. Huemer, R. Najvirt, and A. Steininger, "An automated setup for large-scale simulation-based fault-injection experiments on asynchronous digital circuits," in *2021 24th Euromicro Conference on Digital System Design (DSD)*. IEEE, 2021, pp. 541–548.

[18] A. J. Martin, "Synthesis of asynchronous vlsi circuits," 1991.