Towards Resilient QDI Pipeline Implementations

Zaheer Tabassam , Andreas Steininger
Institute for Computer Engineering, TU Wien, Vienna, Austria
zaheer.tabassam@tuwien.ac.at, steininger@ecs.tuwien.ac.at

***Abstract*—QDI circuits are robust towards timing issues, but this elasticity makes them vulnerable in value-domain fault scenarios because data-accepting windows are flexibly defined by the handshakes, and during these windows any data transition gets latched, even those originating from single event transients.**

As a solution, locking the data-accepting windows after the first transition contributes to robustness, but still needs consideration. We examine WCHB variants called Interlocking-WCHB and Input/Output-Interlocking-WCHB in this respect. To highlight the relevant error triggering conditions, we chose two target circuits to investigate the behavior in detail: FIFO and pipelined multiplier.

Based on the experimental results we investigate the observed errors to understand the main cause of their generation and propagation. We highlight the problematic scenarios and propose modifications in buffer styles that resolve most of these while minimizing the area overhead to 50%.

I. Introduction

Delay variations caused by process tolerances as well as fluctuations in voltage and temperature make attaining timing closure in synchronous designs increasingly harder. With their flexibly adaptive timing, asynchronous design methods, specifically the quasi delay-insensitive (QDI) one, promise to be robust alternatives. In a nutshell, QDI circuits employ local handshakes instead of a global clock to control the progress of computation and data exchange. These handshakes essentially form closed control loops that precisely adapt the timing to the actual delays, hence QDI circuits can easily accommodate delay variations, and they are robust against delay-related types of faults. However, their operation is event-based (rather than level-based as in synchronous designs), which makes them vulnerable to single event transients (SETs) and other types of transient faults. Proven fault-tolerance techniques known from the synchronous domain usually cannot directly be transferred into the asynchronous domain, but in the literature several special techniques have already been proposed to make asynchronous designs more resilient against SETs. In this paper we closely analyze residual problems with some of these techniques and, based on the insights thus gained, propose an improved buffer design for a QDI pipeline which we then validate by extensive fault-injection experiments in simulation. We also show that our approach is competitive with respect to performance penalty and area overhead.

II. Related Work

The focus of this work is on the mitigation of transient faults in asynchronous circuits, specific to the value domain, so here we survey models, effects and hardening techniques for dealing with them. In synchronous systems transient faults are efficiently mitigated through masking capabilities that are partly inherent, and, where required, additionally established with fault-tolerance techniques. The latter, however, tend to have large overhead and architecture constraints when ported to asynchronous systems.

Over the years, researchers have regarded redundancy as key requirement for error resolution and taken inspiration from synchronous hardening methods for protecting asynchronous circuits, like [1], [2]. This strategy is backed by promising results for duplication-based approaches [3], [4]. By leveraging this topology [5], [6] and [7] proved the resilience of real-word asynchronous circuits like processors and controllers. In addition, [8], [9] and [10] highlight the main contributors that must be considered for single event transient (SET)-tolerant asynchronous circuits. Investigations and results of [11], [12], and [13] suggest special ways to apply this redundancy to asynchronous circuits.

Inspired from [4], [13], and [14], the authors of [15] proposed two enhanced QDI buffer styles with high resilience for their specific domains while maintaining low area overhead. As a continuation to this, with a real-word circuit “multiplier”, [16] presents a more elaborated view of QDI templates with respect to transient faults. They not only compare templates, but also illustrate how their behavior varies with circuit dynamics. Based on [15], [17] presents more robust QDI buffer templates that shorten the windows where SETs are harmful.

In conclusion, mere replication of a circuit can enhance SET tolerance, but for a high area cost. Consequently, it seems promising to rather analyze the key contributors to faults in order to selectively target smaller modifications.

III. Asynchronous QDI Circuits

The term “asynchronous” refers to the global temporal coordination within the circuit, where each module maintains local synchrony with its neighbors via handshakes. The handshake cycle of circuits realized with a delay-insensitive (DI) protocol is more flexible in terms of timing assumptions, because the validity of data is defined by the data itself using multi-rail encoding schemes. An explicit $acknowledgement$ signal from the receiver completes the handshake cycle [18].

Throughout the article we focus on QDI circuits, where we stick with a *4-phase return-to-zero* handshake protocol with dual-rail (DR) encoding. In the DR scheme a single bit x is represented by two rails $(x.t, x.f)$ where t and f are $true$ and $false$ rails, respectively [18]. A logical “1” is represented by

This research was partially supported by the project ENROL (grant I 3485-N31) of the Austrian Science Fund (FWF).

setting these rails to $(1,0)$ and "0" by $(0,1)$. These are code words and called *(data) tokens*. The code $(0,0)$ is used as a *spacer*, as demanded in the *4-phase protocol* to separate data tokens. Note that for the considered scheme $(1,1)$ is an illegal pattern. In this protocol without global clock a module interacts with an other by simply placing a data token on its data rails, and after receiving a logical high acknowledgement signal (when considering logic low as reset) from the receiver it changes the data token for a spacer. The handshake is completed with its 4^{th} phase when the receiver responds by resetting the acknowledgment [19].

A. The Muller C-element

In QDI circuits the Muller C-element (MCE) is a fundamental building block. Its functionality is simple: only the logic level of matching inputs is passed to the output and retained until a matching pattern with the opposite logic level arrives Figure 1 shows a symbolic representation of a MCE and its variants.

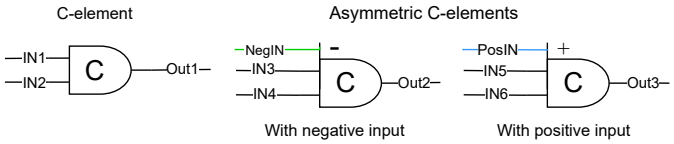

Fig. 1. MCE and its derivatives

In the *Regular MCE* the output changes for symmetric inputs after the inertial delay of the gate. This also holds for the *MCE with negative input (C-element-)*; however, here for up-transitions (only) the input $NegIN$ has no impact. Likewise, the *MCE with positive input (C-element+)* ignores $PosIN$ for the down-transition.

B. QDI Buffer Templates

As QDI circuits operate with handshakes rather than a global clock, they require special storage elements that also obey the handshake protocol with the respective encoding scheme. In this paper we will restrict ourselves to the very popular 4-phase QDI buffer template named Weak-Conditioned Half Buffer (WCHB). We will start with the basic template and then continue with some of its variants that are designed to mitigate the effects of transient faults like SETs. Figure 2 shows a 1-bit WCHB: according to the DR scheme, only one input rail may go high at a time, and if the $enable$ (en) signal is also high, the respective MCE fires, which, in turn, activates the $acknowledgment$ Ack_out and arms the same MCE for capturing the spacer.

If due to a fault both input rails ($In.T$, $In.F$) go high while the en is high, the illegal (1,1) pattern propagates to the output. To mitigate this error the authors of [15] propose their Interlocking WCHB as shown in Fig. 2. Here the MCE whose output transitioned to "1" first, blocks the other MCE from making the same transition, which effectively prevents a (1,1) output. Due to the inertial delays of the MCEs, however, this interlocking fails for error scenarios where an up-transition on both is enabled at nearly the same time.

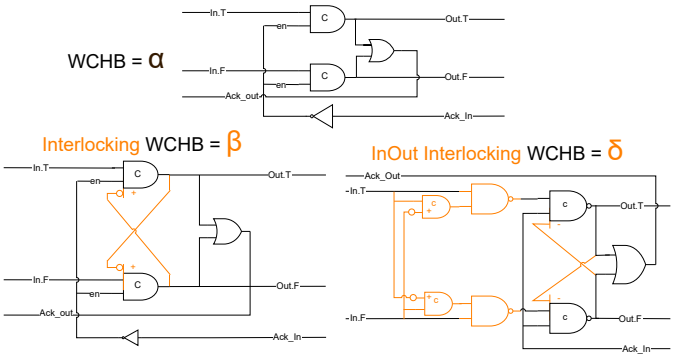

Fig. 2. Considered Buffer Styles

To address this issue [17] proposed the Input-Output Interlocking WCHB shown in Fig. 2. This approach provides a first stage of interlocking (actually input filtering) that just compares the inputs without considering the output and hence does not suffer from the delay problem as the Interlocking WCHB does. This stage is followed by a second one that actually resembles an Interlocking WCHB, albeit with inverted inputs (this is done to optimize area). This latter stage protects against faults at the output that might make the state flip to (1,1). In addition, the AND gates with their delayed and non-delayed version of the same signal provide some degree of glitch filtering. For simplicity we will in the following indicate the plain WCHB with letter α, the Interlocking WCHB with β, and the Input-Output Interlocking WCHB with δ .

C. Pipeline Load Factor

QDI circuits flexibly adapt their operation to the speed in which tokens are provided by the source and consumed by the sink. Consider the case where a fast source keeps providing tokens to the circuit, while, after their propagation, a slow sink consumes them only some time after they become available. Due to the backpressure by the handshake, new tokens start waiting at the source to be issued. This behavior is called *Bubble_limited* because most of time the pipeline is waiting for acknowledgment (termed *bubble*, as a counterpart to *token*) to complete the handshake cycle. In contrast to that, if the source is slow, then a fast sink will start waiting for a new token or spacer. This is called *Token_limited* mode of operation.

To express how much the speed of the pipeline is dominated by a lack of tokens or bubbles, the Pipeline load factor (PLF) has been discussed in detail in [16]. For the remaining discussion a PLF less than 1 indicates a token limited mode of operation, and a PLF greater than 1 a bubble limited mode. In context with our asynchronous QDI design style, the question arises why one should be concerned about these types of delay, because, while affecting throughput, they have no impact on data integrity. In context with transient faults, however, the PLF makes a difference, as the circuit's sensitivity to faults,

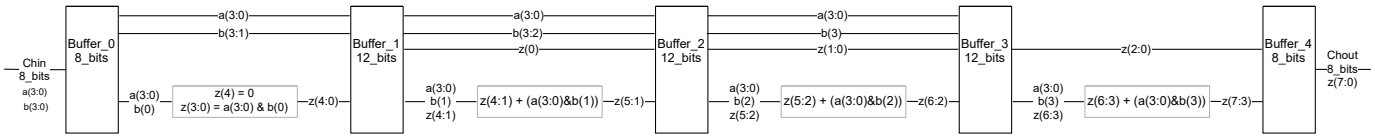

Fig. 3. Pipelined Multiplier

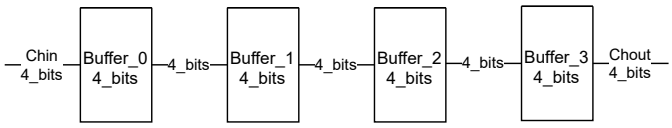

Fig. 4. Empty Pipeline Circuit

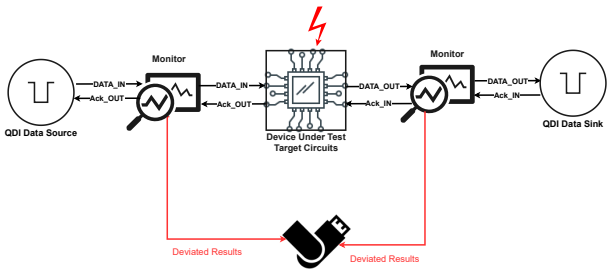

Fig. 5. Fault Injection Simulation Environment

as well as the relative distribution of different types of fault effect (see later) have been shown to depend on the PLF [16].

IV. Experiment Setup

Their event-driven nature and their timing flexibility make QDI circuits specifically sensitive to SETs, as, in the lack of rigid restrictions, faulty transitions can easily be mistaken for correct ones. So to properly mitigate effects caused by SETs without compromising the circuit's timing elasticity, the circuit operation as well as the effects SETs cause on it must be carefully analyzed and well understood. This is exactly the purpose of our experiments.

A. Circuit Designs

Different circuits have their inherent properties and behave differently under different conditions. For this first investigation we chose fundamental but useful real-world circuits. Our list comprises:

1) Empty pipeline or FIFO
2) Multiplier

The data bitwidth remains same for all, namely 4-bit. Lacking any computational function the empty pipeline, as shown in Fig. 4, might be considered an overly simplified target. However, we included it because (a) it can be found in real-world applications like up/down counters or communication interfaces, and (b) the reduction of the pipeline to buffer stages only can allow interesting insights like the identification of possible effects that are specific to the buffers, or, vice versa, such that never occur in buffers.

Due to its simplicity backtracking of effects to the root cause promises to be simpler, and potential masking of effects by combinational function blocks can be ruled out in the empty pipeline. Another advantage is that changing from one template to another already changes the whole circuit.

To make the comparison more realistic, the multiplier target is designed in a pipelined manner (other variants are under investigation for future work). This pipelined multiplier, see Fig. 3, is supposed to give insights about how pipelines with relatively complex computational units in between react to SETs, because these units also contribute to error generation. Multiplication is performed using a simple binary multiplication method.

B. Fault Injection simulation environment

Fig. 5 shows our simulation setup [20]. It consists of a QDI source and sink generating and acknowledging DR data with programmable delays to mimic real-world DI scenarios. With these programmable source/sink delays we control the PLF. Monitors are placed at the interfaces of the target circuit that check each activity and compare it to a golden run. Please note that only value deviations are considered as error, while timing issues are only considered an observation because circuits are QDI. As our concern is SETs, which occur rarely, we only inject one fault per simulation and observe the behavior. We are excluding input and output signals from the injection list because these are directly observable to the monitors with no chance for mitigation or masking, so these will simply result in higher fault rates. Injection time and location are randomly chosen, while the injection pulse length is fixed to a value higher than the longest inertial delay among all gate delays in our considered buffer templates, and hence electrical masking cannot occur (we are only interested in logical and temporal masking effects here).

C. Gate Delays

For the gate delays we utilized the timings from NanGate_15nm library file with typical conditions [21]. As an MCE is not part of this library, we considered the MCE model from [22] in which they propose a combination of simple NAND gates. Gate delays are computed from a timing matrix of the respective gate using an interpolation method with fixed index_1 (input_net_transition), as in our simulations we are not varying this parameter, while we vary index_2 (total_output_net_capacitance) depending on the fanout of the respective gate.

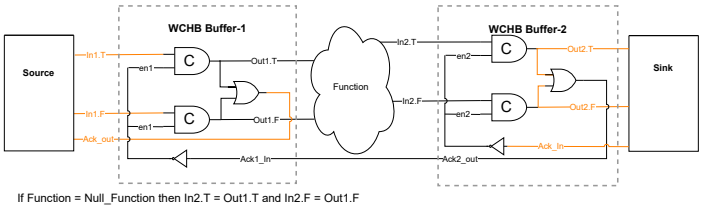

Fig. 6. 2-stage 1-bit QDI Pipeline with WCHB

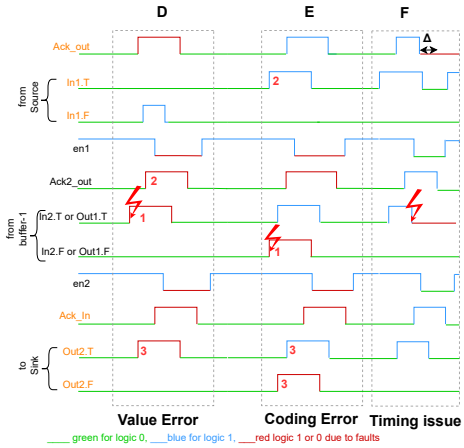

Fig. 7. Explanation of different types of errors

D. Fault effects

Deviations from the golden run are classified into two main categories, namely data errors and timing issues.

For timing issues, i.e., data arriving earlier or later than expected, data integrity remains safe, because of the circuit's delay insensitivity.

Data errors are further classified into four categories

(a) Value error: data is received correctly, but the value is not as expected.
(b) Coding error: both rails of the DR bit go high; this is illegal in our considered DR encoding.
(c) Glitch: during any handshake phase a signal makes more than one transition, or causality of signals is otherwise violated by a wrong sequence, like *acknowledgment* activated before data completion
(d) Deadlock: the circuit stops in a state where no further transition is possible.

Fig. 7 illustrates some of these effects, as they would be triggered by faults propagating through the pipeline shown in Fig. 6. Recall that we had decided to exempt the input and output rails (indicated by orange color) from fault injection.

Value error, box D: if a fault hits Out1.T at a time instant (1) when buffer-2 is armed for token reception (i.e. for an up-transition), the faulty transition is simply stored and generates the $Ack2\_out$ (2), which resets the enable signal $en1$, thus blocking the correct token from entering that just arrived at In1.F. Consequently, a formally correct token with an erroneous value is generated at Out2.T (3). The monitor compares it with the reference behavior obtained in the golden run and flags a value error because it expected (Out2.T = 0, Out2.F = 1).

Coding error, box E: The fault hits Out1.F (1), like above while buffer-2 is armed for a token, this time, however, very close to the instant when In1.T (correctly) goes to "1" as well (2). Due to the propagation delay of the OR completion detector, $Ack2\_out$ cannot block the correct transition anymore by arming buffer-1 for the spacer. So at buffer-2 both, the faulty transition on Out1.F (1) and the correct transition Out1.T arrive and get latched, thus forming an illegal code word. Now there is no way to reset any of the two rails, so the illegal code word propagates through the pipeline and reaches the output rails (3), where the monitor flags a coding error.

Timing Issues, box F: If the fault hits any of the rails at a moment when the token was already conveyed to the successor stage and acknowledged, it only changes the timing of Ack_out, as apparently the spacer arrived earlier. Please note that the scenarios of D and F are almost the same, but the different timing and fault polarity have significant impact, which changes the fault effect.

These examples are only meant to illustrate the error types and to provide some insight into possible fault effects. However, this list is by no means complete, and when injecting at randomly chosen instants we can expect further scenarios.

V. Result Analysis

Fig. 8, 10 and 12 show the error rates we experimentally observed for our target circuits with varying PLF. It is not easy at this stage to identify which factors contribute where. Let us start with glitches from fig. 8 because the magnitude of this error type is very low. Buffer δ performs very well in bubble limited mode, while other styles show some errors. So, the first question arises: What are the main causes of all these?

A. Main causes of glitches in QDI pipelines

We define a glitch as a short pulse, i.e., a sequence of two opposing transitions. We have already seen that the leading transition may or may not arrive at a point in time when actually a regular transition is expected (MCE is armed). If no transition is expected, the faulty one remains ineffective or at most causes a timing deviation. If a transition is expected, the glitch's leading transition will move the handshake process forward one step in one or the other way. However, in a QDI circuit, a next transition on the same signal will then only be expected once the reaction to the previous one has sufficiently propagated. Therefore, due to the short duration of the glitch (by definition), its trailing transition cannot arrive at an armed MCE and must hence remain ineffective. In conclusion we can state that in a QDI circuit glitches cannot propagate.

Backtracking the glitches we had observed for bubble limited mode in our experimental results gave an amazing insight: faults on only one signal contribute to all glitches we observed. Further inspection shows, that for PLF 0.25, and 0.5 all variants show 0% glitches, so we can eliminate these from further analysis. We will come back to the case of PLF = 0.1

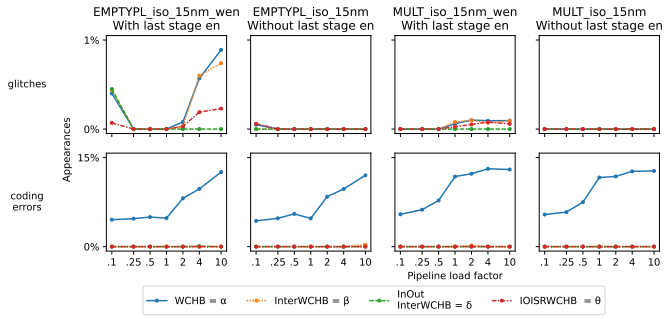

Fig. 8. Analyzing glitches and coding errors in QDI circuits

later on. For PLF ≥ 1 there is, as visible in Fig. 8, no glitch recorded as soon as we remove the last stage's $enable$ signal from injection list that always contributes to glitch generation.

Having identified the culprit through filtering of the results, we can now use Fig. 6 to explain how an SET on $en2$ ($enable$ signal of the last stage) can generate a glitch at the output. With our choice of PLF ≥ 1 the circuit is running in bubble limited mode as illustrated in Fig. 9 box J. In this mode the last stage spends a lot of time waiting for Ack_In.

1) At D1, a data token arrives at the circuit output, and as a result, $Ack2\_out$ is activated at time D2 as a confirmation to the predecessor stage.
2) This acknowledgement ripples upstream to the source, and as a result the latter issues a spacer at J1 that is passed to the last buffer stage's input rails at J2.
3) Now the last stage is waiting for Ack_In from the sink to complete the four-phase handshake at J5.
4) Accidently during this waiting window (highlighted with Sink delay) if a fault hits the $en2$ signal as shown at J3. It just produces a spacer at the output (at J4 in our example) before the sink responded to the data token,.

Recall our definition of fault effects, according to which the case where any rail changes more than once during same handshake phase it considered a glitch. So the main cause of glitches in all target circuits is when a new transition arrives during an ongoing handshake phase.

A close look at Fig. 6 reveals that $en2$ is simply the inversion of Ack_In. While the latter, being an input signal, is excluded from fault injection, the former is not. The δ design, however, works without $acknowledgment$ inversion, and therefore its Ack_In is never hit by a fault, which explains the absence of glitches in our results for that template during bubble limited mode. This is an important observation. A remedy to this that provides more fairness in the comparison of the different buffer templates is to consider the $enable$ signal of the last buffer stage as input signal as well, together with Ack_In, because these two signals are just inverted versions of each other. To verify this approach, we ran the injection experiments again, this time, however, excluding Ack_In as well as $en2$ from the injection. The results in Fig. 8 with label $''Without\_last\_stage\_en''$ indeed confirm 0% glitches for the bubble limited mode.

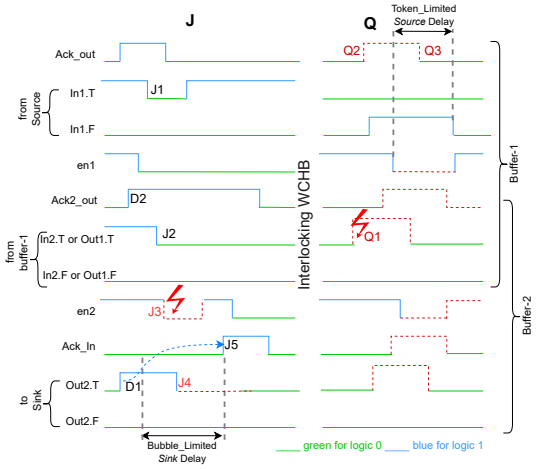

Fig. 9. Explaining the main cause of glitches

In settings with PLF = 0.1 another important problem arises: To visualize the cause with a different reference, we replace the buffer style of our Fig. 6 circuit by the Interlocking WCHB. Fig. 9 box Q illustrates the situation:

1) Q1, a fault hits Out1.T of the C-element+. Due to the interlocking buffer style, the faulty transition just prevents the other C-element+ from making a high transition.
2) This value propagates to the sink and generates a value error there – but this is not our focus here.
3) The important thing is that the Ack_out at Q2 is generated before the valid data from the source arrived – this triggers a glitch.
4) We know that In1.T remains low, while the logical 1 at In1.F is simply not passed to the next stage due to Q1. Consequently, a spacer is generated at the output of buffer-1 when en1 goes low, and it generates Ack_out low at Q3.
5) These $acknowledgements$ of data and spacer at Q2 and Q3 respectively, that were generated before the source generated a valid token and a spacer, are problematic and make our monitor flag a glitch.

It becomes clear from the explanation that this discussed scenario for PLF = 0.1 already generates some other types of errors in the pipeline. Therefore, in Fig. 8 with label $''Without\_last\_stage\_en''$ this is not counted as a glitch.

Having understood in all detail how SETs can cause glitches in our QDI circuits, we will now move on to Coding errors. As Fig. 8 shows, only the original WCHB (α) is a contributor to that type of erroneous behavior, while the improved buffer types do not show it. So the second question arises: Why does only the WCHB type α suffer from coding errors, and not the other buffer types?

B. Main Causes of Coding Error

Section IV-D already explained how coding error are generated:

Latching of a second data transition (one on each rails) is only allowed in WCHB α, in the absence of any interlocking.

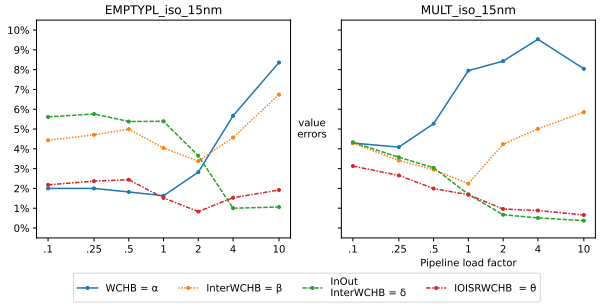

Fig. 10. Analyzing vlaue errors in QDI circuits

In other buffer styles the first transition just locks the companion MCE for further transitions. That's why other buffer styles only generate value errors for this type of fault scenario. As a consequence their value error rates are higher, as visible in Fig. 10. This is because the interlocking mechanism prioritizes the first transition that occurs, even if that is the faulty one, and blocks the one that arrives later. In this way, a valid code is enforced, but the encoded value may be wrong.

More generally, Fig. 10 shows that mitigation techniques in some cases fail to handle value errors, so in the next section we will investigate these in detail.

C. Main causes of Value Errors

Based on the circuit diagram from Fig. 6 and the timing diagram from Fig. 7, box D, we can explain the generation and propagation of value errors. A closer inspection reveals the storage loop within the MCE as important source of value errors. Recall that the storage loop retains the last output level – by feeding it back to the input – as long as the inputs are asymmetric. When a fault hits that output signal, the feedback may make the storage loop flip. Fig. 11 illustrates some situation for the buffer type β.

(i) In Token_Limited mode In.F = 1 is expected (dim dotted line shows that values have not arrived yet). But before valid data arrives, a fault at Out.T flips the MCE that is supposed to remain 0 during this phase. As a result, the valid transition is halted and the fault propagates further to the output.

(ii) In Bubble_Limited mode valid data is waiting at the buffer input for the en signal (dim dotted lines shows that after some time en goes high) but before that arrives, a fault at Out.T flips the MCE output as well as disables the other MCE that was supposed to fire valid data.

For the δ approach the idea of interlocking inputs and outputs is reasonable, but the use of extra MCEs increases the area ovehead, while failing to shows promising result during token limited mode: as visible from Fig. 10 WCHB β is performing better than its successor δ. For bubble limited mode the error rate of δ drops, as its design considered this very region.

The δ variant has another very prominent drawback. Consider the scenario presented in Fig. 11. If during token limited mode a fault hits the input of the MCE that is assumed to not change during this phase, it generates a value error without performing any input interlocking. Fig. 10 shows that indeed during token limited mode it experiences more value errors.

Finally, as illustrated in Fig. 11, logic units also contribute to the value errors.

Like the Delay-Insensitive Minterm Synthesis (DIMS) implementation of an AND gate shown in the example, these also may contain MCEs and hence suffer from the same state flips as illustrated: Out.T goes high where inputs of that MCE are not yet completed.

D. Main causes of deadlocks

For our analysis we now consider the pipelined multiplier with WCHB δ. The discussed scenario is not an abstract discussion but the visualization of a real simulation instance, however, with a simplified view. As it is not possible to show all wires and details here, we present the affected signal lines with buffers in Fig. 13 and use these for explanation.

(a) Fig. 13 shows that $b(3)$ is directly connected from buffer-2 to buffer-3. So, if a high fault hits the $b3.T$ line that is supposed to remain low, and at the same time $b3.F$ also makes a transition to high, C3 and C4 mutually interlock each other and both transitions remain at the input of buffer-3.

(b) Ack_Out is only generated when this interlocking is resolved and any output rail of buffer-3 makes a transition. That is, however, not happening.

(c) Here the important thing is that the fault, by hitting the input signal of C3, also affected the output of C1, and got latched by flipping its storage loop (as explained earlier). So the erroneous state at $b3.T$ is memorized in C1.

(d) C2 will not change $b3.F$ before receiving the Ack_out – which is, however, not generated due to the interlocking of C3 and C4.

No $acknowledgment$ means no further transitions.

VI. Novel Buffer with Improved Resilience

Basically, the concept of input output interlocking from WCHB δ performs well, and if realized in an efficient fashion, can address errors with reasonable area overhead. Our experimental investigation has shown the following residual deficiencies:

(a) SETs on the $enable$ of the last buffer stage can lead to glitches. This problem is confined to the last stage only and can be solved by appropriate design of the read interface, or specific hardening of the en signal. Consequently we do not address it in the following discussion about improvements for the buffer design in general.

(b) The interlocking between the MCEs storing true- and false-rail, respectively, can eliminate coding errors due to SETs, but at the cost of an increased rate of value errors.

(c) SETs at the output signal of a MCE can make its state flip, which can result in value errors and deadlocks.

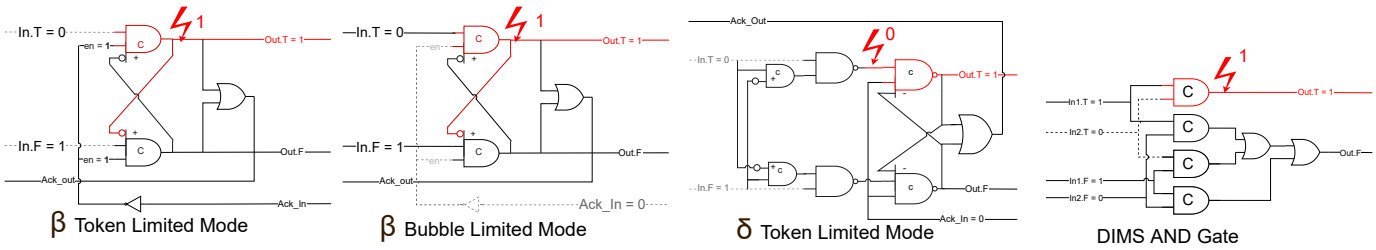

Fig. 11. Main Causes of Value Errors

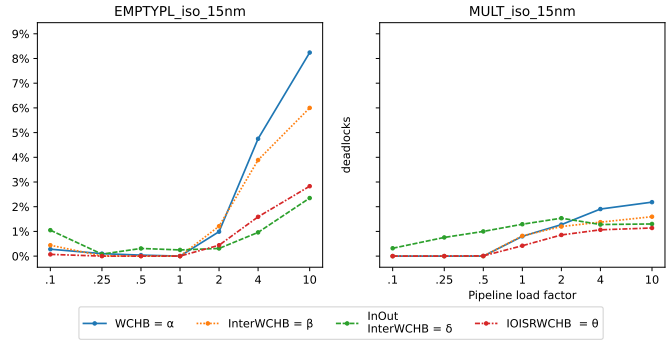

Fig. 12. Analyzing deadlocks in QDI circuits

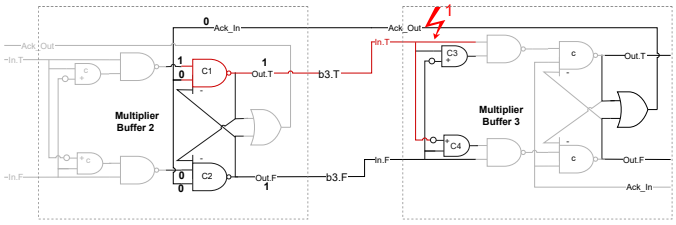

Fig. 13. Explaining the deadlocks in QDI Circuits

(d) The δ WCHB as discussed in V-C has some design issues due to which it does not perform as well as expected.

From these insights we can now derive some further enhancements, as shown in Fig. 14.

We prefer using SR latches instead of MCEs for input interlocking. This not only saves area, it also, most importantly, mitigates the problem of state flipping through faults at the output of the MCE used for input interlocking, as discussed

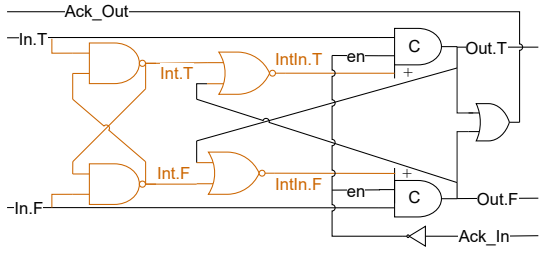

Fig. 14. IOISRWCHB = θ

in Sec V-C. In fact, this SR latch resembles the classical mutex implementation from [23] – albeit without metastability filter – much closer than the interlocked MCEs did.

Importantly, the decision to arm the C-element+ is now combining input and output state using a NOR gate that, together with its companion from the other rail, forms another SR latch. We named this approach Input Output Interlocking with SR latch WCHB (IOISRWCHB) represented by the letter θ. Fig. 8, 10 and 12 show the fault-injection results that were generated using the same settings as for all other buffer styles to make the comparison fair.

A. Mitigating Value Errors

For value errors we observe in Fig 10 that in token limited mode the rate is lower than with WCHB δ. Considering Fig. 14 we can explain this improvement as follows:

1) If a fault hits $Int.T$, $IntIn.T$ or $Int.F$, $IntIn.F$, whatever the scenario, this has no effect: Even if the respective signal value matches with the direct inputs $In.T$ or $In.F$, it just validates the output prematurely raising only the timing issue flag. Otherwise it will be masked anyway at the MCE. Recall from Fig. 11 that for WCHB δ the same scenario of hitting an internal signal generated a value error.
2) If the fault hits $In.T$ or $In.F$ its effectiveness depends on a few factors.
   (a) It first passes through the input interlock filter (NAND gate) and the output filter (NOR gate). It after that the en signal is also matched to the context of the fault then it becomes visible to the MCE.
   (b) If the fault length is less than the propagation delay of NAND plus NOR gate, the $In.T$ or $In.F$ already changed back to the regular level when the fault appears at the positive input of the MCE. In this situation the fault has no impact on the circuit. So, we safely filter most input faults.

In summary, through the strict separation of original and delayed path, our internal filter signals are immune to faults. Another important insight about the θ approach is that transitions must first pass its input interlock before being able to halt the opposite rail. In contrast, WCHB δ blindly interlocks the other rail for further transition, as explained in Fig. 13.

An important benefit of the proposed WCHB variant θ is the obtained reduction of area and throughput penalty as compared

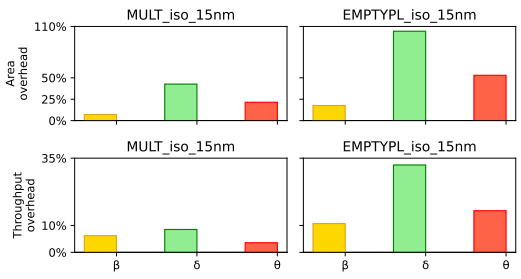

Fig. 15. Area Overhead and Throughput Reduction Comparison

to δ. The results are shown in Fig. 15, with the WCHB as baseline. Our θ approach shows a significant reduction (approx 50%) in area overhead and achieves higher throughput as compared to its base-buffer style δ. This is mainly due to the replacement of the MCEs in the input stage by simple combinational gates and the way we interlock output MCEs.

VII. Conclusion

In this work we have thoroughly analyzed the effects of SETs in QDI circuits and highlighted vulnerabilities of existing buffer designs, specifically the WCHB and some hardened variants of it. To this end we have combined statistics obtained from extensive fault-injection experiments in simulation with a very detailed investigation of fault effects on the level of individual signal traces. To make our analysis as comprehensive as possible we have performed these experiments for different target circuits that we operated with different settings for the speed of source and sink.

One of our findings was that, even though, due to their operational principle, QDI circuits do not propagate glitches, these can still occur when an SET hits the enable signal of the last buffer stage. Furthermore we could identify the premature handshake completion caused by an SET as another source of glitch-like circuit behavior.

Based on our insights we have proposed a further improved buffer design, for which our experimental validation confirmed an improved resilience to SETs, with notable resilence in token limited operation while showing comparable behaviour during extreme bubble limited mode of operation. At the same time 50% reduction in area and performance penalty of our approach are quite competitive.

For future direction we have also identified state flips of the MCE (while in state holding mode) through SETs on its output as a main contributor to error effects. So, our deep analysis suggest if somehow we are able to maintain the symmetry of input lines of the MCE during the waiting time of the pipeline (called token or bubble limited modes), faults at the output of the MCE lose their effectiveness. As a result the remaining 1 to 3% faults are easily addressable.

References

[1] T. Verdel and Y. Makris, "Duplication-based concurrent error detection in asynchronous circuits: shortcomings and remedies," in *Proceedings of the 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2002, pp. 345–353.

[2] F. A. Kuentzer and M. Krstic, "Soft Error Detection and Correction Architecture for Asynchronous Bundled Data Designs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–12, 2020.

[3] Y. Monnet, M. Renaudin, and R. Leveugle, "Hardening techniques against transient faults for asynchronous circuits," in *11th IEEE International On-Line Testing Symposium*, July 2005, pp. 129–134.

[4] W. Jang and A. J. Martin, "SEU-tolerant QDI circuits [quasi delay-insensitive asynchronous circuits]," in *11th IEEE International Symposium on Asynchronous Circuits and Systems*, March 2005, pp. 156–165.

[5] M. Marshall and G. Russell, "A Low Power Information Redundant Concurrent Error Detecting Asynchronous Processor," in *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*, Aug 2007, pp. 649–656.

[6] S. Keller, A. J. Martin, and C. Moore, "DD1: A QDI, Radiation-Hard-by-Design, Near-Threshold 18uW/MIPS Microcontroller in 40nm Bulk CMOS," in *21st IEEE International Symposium on Asynchronous Circuits and Systems*, May 2015, pp. 37–44.

[7] F. A. Kuentzer, M. Herrera, O. Schrape, P. A. Beerel, and M. Krstic, "Radiation Hardened Click Controllers for Soft Error Resilient Asynchronous Architectures," in *26th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, May 2020, pp. 78–85.

[8] Y. Monnet, M. Renaudin, and R. Leveugle, "Asynchronous circuits sensitivity to fault injection," in *10th IEEE International On-Line Testing Symposium*, July 2004, pp. 121–126.

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

[10] R. P. Bastos, Y. Monnet, G. Sicard, F. Kastensmidt, M. Renaudin, and R. Reis, "Comparing transient-fault effects on synchronous and on asynchronous circuits," in *15th IEEE International On-Line Testing Symposium*, June 2009, pp. 29–34.

[11] 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)*, 2005, pp. 484–493.

[12] K. T. Gardiner, A. Yakovlev, and A. Bystrov, "A C-element Latch Scheme with Increased Transient Fault Tolerance for Asynchronous Circuits," in *13th IEEE International On-Line Testing Symposium (IOLTS 2007)*, July 2007, pp. 223–230.

[13] W. J. Bainbridge and S. J. Salisbury, "Glitch Sensitivity and Defense of Quasi Delay-Insensitive Network-on-Chip Links," in *15th IEEE Symposium on Asynchronous Circuits and Systems*, May 2009, pp. 35–44.

[14] P. McGee, M. Agyekum, M. Mohamed, and S. Nowick, "A Level-Encoded Transition Signaling Protocol for High-Throughput Asynchronous Global Communication," in *14th IEEE International Symposium on Asynchronous Circuits and Systems*, 2008, pp. 116–127.

[15] F. Huemer, R. Najvirt, and A. Steininger, "Identification and confinement of fault sensitivity windows in qdi logic," in *2020 Austrochip Workshop on Microelectronics (Austrochip)*, Oct 2020, pp. 29–36.

[16] 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)*, 2021, pp. 25–33.

[17] Z. Tabassam, P. Behal, R. Najvirt, and A. Steininger, "Input/output-interlocking for fault mitigation in qdi pipelines," in *2021 Austrochip Workshop on Microelectronics (Austrochip)*, 2021, pp. 17–20.

[18] J. Sparsø, *Introduction to Asynchronous Circuit Design.* DTU Compute, Technical University of Denmark, 2020.

[19] Z. Tabassam, S. R. Naqvi, T. Akram, M. Alhussein, K. Aurangzeb, and S. A. Haider, "Towards designing asynchronous microprocessors: From specification to tape-out," *IEEE Access*, vol. 7, pp. 33 978–34 003, 2019.

[20] 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)*, 2021, pp. 541–548.

[21] si2.org. (2014) Silvaco open-cell 15nm library v0.1_2014_06 from si2. [Online]. Available: https://si2.org/open-cell-library/

[22] K. S. Stevens, D. Gebhardt, J. You, Y. Xu, V. Vij, S. Das, and K. Desai, "The future of formal methods and gals design," *Electronic Notes in Theoretical Computer Science*, vol. 245, pp. 115–134, 2009.

[23] C. L. Seitz, "Ideas about arbiters," *Lambda*, vol. 1, no. 1, pp. 10–14, 1980.