Identification and Confinement of Fault Sensitivity Windows in QDI Logic

Florian Huemer, Robert Najvirt, and Andreas Steininger
Institute for Computer Engineering, TU Wien, Vienna, Austria
{fhuemer,rnajvirt,steininger}@ecs.tuwien.ac.at

***Abstract*—Asynchronous, specifically QDI circuits are known to exhibit high resilience against faults affecting the timing. At the same time, their event based operation principle makes them susceptible to glitches. While synchronous circuits obtain high resilience through temporal masking that is established through the sampling of data by flip flops, asynchronous designs, by trying to be flexible about the phases of data validity, often have lower masking capabilities. Consequently, previous work has proposed to narrow down the windows in which data changes are accepted, in order to improve the temporal masking in QDI designs.**

In this paper we study the natural resilience of different QDI templates in more detail and quantitatively determine the windows of vulnerability through extensive fault injection experiments in simulation. To this end we propose a novel way of visualizing and analyzing the sensitivity windows that aids in identifying the key dependencies and vulnerabilities. In addition we introduce and evaluate two low-cost extensions for the pipeline registers which allow either to stall the operation in case a glitch creates an illegal symbol, or to prevent the creation of an illegal symbol in the first place.

I. Introduction

In contrast to synchronous circuits with their rigid time-driven operation dictated by the clock, asynchronous circuits, in particular Quasi Delay Insensitive (QDI) circuits, employ a closed loop control, established by handshake protocols, to adapt their speed of operation to the respective operating conditions. This makes their timing very robust and hence initially qualifies them for building resilient circuits. However, to leverage that potential their behavior with respect to faults in the value domain must be understood and optimized as well. In synchronous circuits it is evident how flip flops provide temporal masking and thus make the circuit immune against transient faults during certain phases. QDI circuits, in contrast, are deemed susceptible to transient faults in the value domain, due to their transition-centric operation. Still, however, these circuits also benefit from immanent masking effects, but these are more difficult to identify, as they depend on implementation, pipeline fill level, path delays etc. For any experimental assessment by physical or simulated fault injection this makes the parameter space to be covered huge. In addition, rather than just giving one or several key statistical descriptors for the masking – like the size of the sensitive window relative to the clock period in a synchronous design – it becomes important to comprehend the relevant dependencies.

One contribution of this paper is a novel method for visualizing fault sensitive windows that gives detailed information about the sensitivity of individual signals and its dependence on pipeline fill level in a compact and intuitive way. A second contribution is a thorough analysis of a Weak-Conditioned Half Buffer (WCHB) based pipeline, along with an apples-to-apples comparison of different methods proposed in literature for enhancing its resilience, first in theory and then based on our proposed experimental approach that is based on fault injection simulations. Finally, as a third contribution, we present two novel schemes that prevent the generation/propagation of illegal codes, as they often result from faults in QDI circuits, across (WCHB) pipeline stages, and thus contribute to enhancing resilience in the value domain.

The paper is structured as follows: The next section gives a brief introduction into the principles of asynchronous circuits. Then Sec. III reviews existing literature about enhancing and assessing their resilience. Sec. IV presents the design templates and mechanisms we consider in our comparison and analyzes their resilience from a theoretical point of view, followed by an experimental analysis in Sec. V. Finally Sec. VI introduces our novel enhancement schemes and assesses their effectiveness, before Sec. VII concludes the paper.

II. Background

Where synchronous circuits use the clock signal to control data transfer between storage elements (e.g., pipeline stages), asynchronous circuits use some form of closed-loop handshaking protocol. This handshake mechanism (usually) involves two signals, called request (req) and acknowledgment (ack). The data source uses the req signal to indicate the availability of new data to the sink, which then acknowledges the reception using the ack signal[1]. Depending on the number of transitions on these two wires for one complete handshaking cycle, the protocol can be classified as 2-phase or 4-phase. A 4-phase handshake is defined by the switching sequence $req{\uparrow}$, $ack{\uparrow}$, $req{\downarrow}$, $ack{\downarrow}$, where the arrow symbols denote rising and falling transitions, respectively. Hence, the handshake signals always start and end the handshaking cycle with the same logic value. In contrast to that, 2-phase protocols only use two transitions and leave the handshake signals in the opposite logic state. This means that e.g., $req{\uparrow}, ack{\uparrow}$ or $req{\downarrow}, ack{\downarrow}$ both constitute complete 2-phase handshakes.

The described request mechanism does not need to be implemented as a dedicated req wire. It is also possible to

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

[1]This explanation assumes push channels, pull channels will not be considered in this work.

implicitly encode the request into the transmitted data using some DI code [1], which leads to the class of QDI circuits [2]. The sink then has to use a completion detector (CD) to decide when the received data is complete (i.e., valid) and can thus be consumed and acknowledged. QDI circuits can also be implemented using 2-phase or 4-phase protocols. However, for circuits that actually process data (in contrast to just transmitting it, e.g., [3]), practically only the 4-phase variant is relevant. Hence, in this paper we only focus on this style. For the same reason we will only consider the dual-rail (DR) data encoding as Delay-Insensitive (DI) code. The DR code uses two rails (i.e., wires) for each bit, the true and false rail. For a DR bit d, we denote these rails by $d.t$ for the true rail and $d.f$ for the false rail. In every handshaking cycle all data rails start out in a low state (spacer) and only one rail of each DR bit can go high. After acknowledgment all rails switch back to low and the whole process start over. Hence the receiver can simply use an OR gate to check for completion on each received DR bit. The individual phases of this protocol are also referred to as *null* and *data* phase.

Fig. 1 shows an example timing diagram of the transmission of two DR bits d_0 and d_1. The bit order of the transmitted bits in the figure is given by $(d_1.t, d_1.f, d_0.t, d_0.f)$. The transmitted binary data (d_1, d_0) in the figure is given by $(0, 0)$ followed by $(1, 0)$.

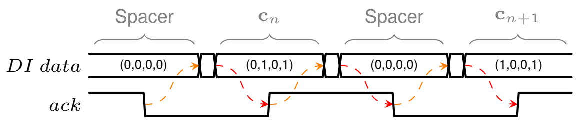


Fig. 1. 4-phase Protocol

It is important to stress that the order in which the transitions on the individual data rails arrive at the receiver does not matter. Furthermore, the individual gate and wire delays in a circuit can vary arbitrarily and the circuit is still guaranteed to work correctly. The only restriction on the delays is the so called isochronic fork constraint, which demands that for certain wire forks the delays of the individual paths after the fork must be equal. This is also the reason why this class of circuits is referred to as “only” *quasi* DI.

This fundamental property of QDI circuits, with respect to their delays, constitutes the key for obtaining tolerance against PVT variations. The disadvantage that is associated with this robustness is that a QDI circuit always entails a high area overhead when compared to corresponding synchronous or asynchronous bundled data circuits. The reason for this area overhead is the additional cost associated with the DI code, the required CDs and the more complex logic gates.

III. Related Work

Our analysis will focus on transient faults, as these are much more frequently encountered during different missions, especially in the shape of single event transients (SETs), i.e., short voltage pulses caused by radiation particle hits. For synchronous designs, there is an abundance of literature available for mitigating SETs and avoiding single event upsets (SEUs) they cause in storage elements. These concepts cannot be directly applied to asynchronous circuits for two reasons: (i) The effects of SETs are different in asynchronous logic, and (ii) most of the synchronous solutions rely on synchrony assumptions between replica, which do not (naturally) hold in asynchronous architectures [4]. In order to cope with (i), an appropriate modeling of faults (propagation and masking) in asynchronous logic is essential, which again differs from the synchronous models like [5] or [6]. In an early approach [7] built an automated verifier for speed independent circuits based on trace theory, which is a very natural means for describing the sequences of transitions relevant for the operation in asynchronous circuits. Later, [8] also used trace theory (among others) for modeling the correct (and incorrect) behavior of asynchronous circuits, more specifically interfaces. In [9] the authors perform a very thorough analysis of SET effects in a selected QDI function block. Their focus is first on the gate level and then on channels (with 4-phase/RTZ protocol), and they use handshaking expansion notation (HSE) to describe the circuit behavior. They identify deadlock, synchronization failure, token generation and token consumption as possible SET effects. In [10] a similar investigation has been conducted, again by means of HSE notation. Here, the treatment of glitch effects and their coverage is more formal, but also a validation through transistor level simulation is given. In a relatively informal case distinction [11] analyzes the consequences of SETs in 4-phase QDI circuits, and concludes that possible effects are blocking, filtering, delay fault and soft error. Based on this analysis three hardening techniques are elaborated. By means of signal transition graphs (STGs) [12] performs a very comprehensive study of SEU (rather than SET) effects in QDI circuits, while later [13] also uses STGs to investigate SETs in QDI network-on-chip links. They propose a bundle of techniques to protect QDI communication links and interfaces from glitches. Also with a focus on SEUs [14] applies symbolic simulation for an exhaustive coverage of all possible behaviors under faults. A more experimental approach is pursued in [15]. Here a simulated gate level fault injection study is performed to compare the sensitivity of synchronous versus asynchronous logic blocks to transient faults and to analyze the respective masking effects. In almost all these approaches the assessment, if any, is limited to a study of overheads. On the other hand, in those few publications where actual fault injection into asynchronous logic has actually been performed (in simulation or hardware), the focus is very narrow [10], or those experiments serve a purpose other than supporting a model [16]–[18]. This makes it difficult to perform an apples-to-apples comparison of the different resilience enhancement approaches proposed.

Fundamentally, the adverse effect of particle hits can be largely mitigated by the use of specific rad hard technologies. In their simplest form they provide just a design kit and library with larger feature sizes, which results in higher capacitances and higher critical charge, such that the charge induced by a particle hit does not cause voltage pulses with considerable amplitude [19]. Consequently, such libraries cannot leverage the benefits of technology scaling, and hence their overheads become progressively higher. A more scalable concept among the measures for obtaining SET tolerance is, like in the synchronous domain, full or partial [20] replication. There are, however, two important differences: (a) Concurrent voting is not straight¬forward, as each instance is running at its own, self-timed, pace [4]; but (b) the control loop established by the handshake allows to keep the faster instance waiting until the slower has its result available as well. This fundamental principle is outlined in [12] where a C gate performs this synchronization task. This principle has been often adopted and extended, like in [11], [18]. There are, however, two fundamental problems: (1) Without further measures (that ultimately imply undesired timing assumptions) a non-responding instance (omission fault) can deadlock the whole redundant architecture. (2) A short SET on the slower one of the redundant paths may lead to early completion and ultimately cause desynchronization of the replica. Beyond a mere replication of function blocks, a clever mutual synchronization of similar functions (like such applied to neighboring bits in a word) or the reduction of the redundant function block to a lean synchronization component are proposed (e.g., in [11]). In [21] the authors use a combination of spatial redundancy and guard gate to harden a controller against radiation. Detailed knowledge about (legal and illegal) protocol states or tokens has often been leveraged for error detection in internal and external asynchronous links, like in [4], [10], [13], [22], [23] and [24]. In [25] information redundancy is leveraged to make a whole processor low power and asynchronous at the same time. In particular, [26] relies on checking for an illegal state of a bit's DR representation before actually latching any of the two rails, while the work in [27] is based on replication of the cell plus DR conversion. A complete radiation-hard-by-design QDI processor (DD1) has been presented in [18]. To achieve SET tolerance it essentially relies on duplication and cross coupling, as well as memory protection techniques.

IV. QDI Design Styles

For our analysis we primarily look into WCHB-based design styles and some derivatives. Fig. 2 shows a single-bit WCHB pipeline with three stages. The storage elements in this buffer are C gates[2]

The operation principle of this circuit is quite simple. Assume the input data rails and the input acknowledgment (ack_{in}) are zero, i.e., the circuit is in the null phase. Due to the inverter the enable signals (en) are high, which arms the C gates for rising transitions on the data rails. After some input data arrives and one of the C gates (in the first buffer) actually switches to one, the CD, i.e., the OR gate, will eventually generate the output acknowledgment ack_{out}. At the same time this transition also travels through the whole pipeline, setting the respective C gates in each stage, until the data appears at the output of the pipeline. Since the input data has been acknowledged by the fist stage, the input rails may now enter the null phase again. This null phase will then propagate through the pipeline as well. However, it will only be able to reset the last stage in the pipeline if the output data is acknowledged by a falling edge on ack_{in} first. This means a single WCHB will keep its stored value (or spacer) until the succeeding pipeline stage acknowledges the data (or spacer), by toggling the acknowledge input of the respective stage (using its CD).

When implementing n-bit WCHBs, completion detection is performed individually on each DR bit by an OR gate. The outputs of these OR gates are then merged by an n-input C gate to generate the output acknowledgment ack_{out}

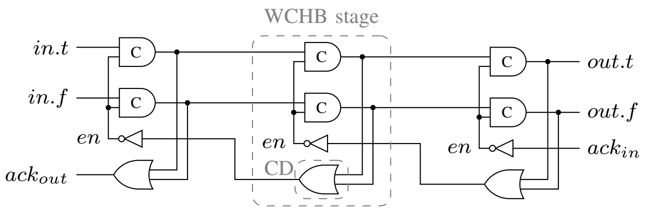


Fig. 2. 3-stage single bit WCHB pipeline

A. Fault Effects and Sensitivity Windows

Fig. 3 shows a timing diagram of a multi-bit WCHB over a single handshake cycle. The shown data signals correspond to the data outputs of the buffer, i.e., the outputs of the C gates. The DR bits d_0 and d_n symbolize the pair of data signals with the largest skew between them (windows B-C and E-F).

As soon as the input acknowledgment ack_{in} goes low (A) a WCHB waits for the data-phase and thus all its C gates are armed for rising transitions. The C gates are only disabled when the next stage acknowledges the received data (D). This leaves quite a large time window (A-D) where the buffer is susceptible to faulty (rising) input transitions. We say that the buffer is now in a (fault-)accumulating state, since all input transitions (faulty and valid ones alike) will be captured into the C gates. This is in stark contrast to e.g., synchronous designs, where there is only a single point in time (i.e., the clock edge) where data is captured by the storage elements.

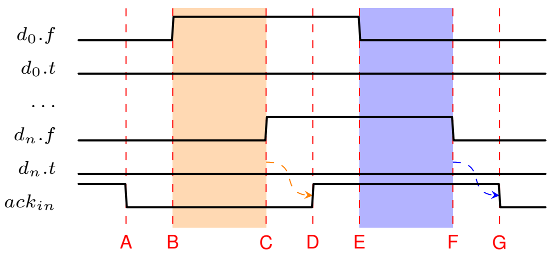


Fig. 3. WCHB time windows

Depending on when exactly a fault strikes a data wire during the data phase and erroneously sets a C gate, one can observe vastly different effects. If a fault strikes a data wire that should

[2]The Muller C-element, or short C gate, is a fundamental gate in asynchronous logic. Its function is to output the logic level seen at its inputs when these match, and to retain the last valid output state otherwise. It can hence also be viewed as an AND gate with hysteresis.

transition anyway, in the best case only the point in time when this transition happens is shifted, which is completely tolerable in the context of QDI circuits. However, given sufficient skew between the data rails, in the worst case such a fault can deadlock the whole circuit. Striking a data rail that should remain zero a fault can (besides the deadlock) cause an invalid output pattern for a DR bit (i.e., both rails set to one) or a valid but incorrect data token, depending on whether the valid transitions on the respective other DR signal arrives before the buffer is closed.

After ack_{in} has been asserted (D) to acknowledge the received data the C gates in the buffer are again armed for falling input transitions. Hence there is a time window (D-G) where the buffer is susceptible to faulty (falling) input transitions. However, since the null phase does not carry any actual data, faults don't affect the transmitted data but can again lead to deadlocks.

Since the input acknowledgment signal ack_{in} basically represents the enable signal for all the C gates in a buffer, faults affecting this signal can have severe consequences on a circuits. These range from deadlocks to lost data items or illegal output patterns. The ack_{in} (or en) signal is vulnerable during phases where the inputs to the C gates differ in their logical value (i.e., the C gate is in state-holding mode). In such a situation a single input change can flip the value of the C gate. As Fig. 3 shows, at time (C) the output of the buffer is complete, hence eventually the output acknowledgment ack_{out} would be asserted, which in turn leads to the deassertion of all input data rails. However the output of the buffer must not enter the null phase until ack_{in} is asserted, which means that there is a sensitive window for the acknowledgment signal between (C) and (D). A similar situation arises during the null phase between the points (F) and (G).

We see that although in general delays are not relevant for the correct behavior of QDI circuits, the exact fault behavior of such a circuit is highly dependent on the input skew, the involved circuit delays, the speed of operation and the actual data that is being processed. This means that the sensitivity of a buffer cannot be evaluated completely statically but must incorporate environmental conditions as well as concrete circuit details.

B. Fault Mitigation Strategies

To address these fault sensitivity windows inherent to the WCHB discussed in the previous section, several strategies have been proposed in literature. Here, we present a selection of these approaches, which will also be analyzed in more detail in the following section.

Most WCHB modifications aim at shortening the time windows in which the buffer stores input transitions. One way to achieve this, is to use two CDs for every buffer, one for the input data and another one for the output data. Consequently we refer to this buffer as the *Dual CD* WCHB. The C gates of the buffer are then only armed when there is actually data at the input (point (C) in Fig. 3). This idea is presented in more detail in [13], where it is referred to as *normally closed latch*.

Moreover, [13] further proposes another slightly different approach, which uses asymmetric C gates as storage elements for the buffer. These C gates have one additional (asymmetric) input that must be asserted in order to set the C gate, but does not need to be deasserted to reset it again, like a normal input would have to be. The asymmetric input is then fed by the inverted output of the buffer's CD, which ensures that the C gates are disarmed as soon all input transitions arrived, effectively closing the sensitivity window at point (C) in Fig. 3. In case of the DR code this can be done on a per-bit basis by using the output of the individual OR gates, which has the benefit of closing the C gate of the respective other rails as soon as one transition arrived. This approach prevents the capturing of the invalid DR state (11). However, blindly capturing the first transition creates a potential of forwarding a wrong value (50% when assuming random faults). In that sense it operates similarly to a Mutex, with the important difference that depending on the timing of the feedback path, there may still be cases where both outputs are set simultaneously. This buffer modification can be beneficial in combination with an error correcting code (ECC) on top of the DR code: It prevents the protocol from being upset by an illegal DR state, while the potential corruption of the data value it causes can be undone by the ECC. In this paper we refer to this approach as the *Locking* WCHB.

Another approach to avoid the lasting consequence of a faulty transition is to use different storage elements. For that purpose a D latch based Mousetrap-style pipeline structure as proposed in [3] can be used. Here a simple buffer control circuit, consisting of just a single XOR gate, is responsible to enable and disable the buffer's D latches. When the D latches are transparent, input glitches caused by SETs can freely propagate thorough the latch to the output. Unless the latch is closed in exactly this time instance, the latch will not store the faulty value. Similar to the previous approach the buffer is closed as soon as the CD detects the data phase. Although strictly speaking this circuit is not QDI because it introduces a small timing constraint, we still want to include it in our survey since it should show quite a different behavior in the analysis and will refer to it as MTDHB (Mousetrap-style D Latch half buffer). Note, however that this buffer has not been proposed to improve fault-tolerance.

Finally, we also want to mention a completely different approach proposed by Jang and Martin in [12]. This is not really a design or buffer style on its own, rather a technique for hardening existing QDI designs. Here the original design is duplicated and both copies are interlocked with C gates that essentially vote on every intermediate signal of the circuit. The authors formally show that this scheme is able to tolerate (single) faults on any internal signal, which means that in our analysis it should not show any erroneous behavior. However, it is easy to see that this approach entails more than double the area overhead of the original design. Moreover, the additional logic for synchronization of the two replicas also results in a slightly slower circuit. Following the terminology used in [12] we refer to WCHB using the described technique as a *doubled-up double-checking* (DD) WCHB.

V. Experimental Analysis

To experimentally analyze what fault sensitive windows the different QDI buffer styles have, we use a Questa based simulation to inject SETs. Fig. 4 shows our fault-injection simulation target: a 4 stage, 2 bit, DR QDI pipeline connected to a data generator at its input and a checker at its output. The pipeline was generated for each buffer style under evaluation and uses inertial gate delays and no wire delays. The 4 data rails between buffers 1 and 2 as well as the acknowledgment input to buffer 2 were chosen as victim wires. Whether an injected fault had an effect on the circuit was checked in both the data generator and checker, which compared timing and values to a reference run, as well as checking for protocol and coding violations. Whenever a deviation from the undisturbed *golden* run was detected, the behavior was classified as follows:

- Timing Deviation: A transition happened earlier or later than expected. The circuit being DI, this is not a fault, but rather an observation.
- Value Fault: A wrong data value was delivered to the output.
- Code Fault: An invalid DI code word was observed at the output (i.e., both rails of a DR bit high).
- Glitch: A signal changed its value twice during a protocol phase. This includes protocol violations (e.g., acknowledgment before data completion)
- Deadlock: The circuit reached a state where no further transitions were possible.

For the depiction of the results, we consider these classes to be ordered in ascending importance – whenever a fault injection triggered faults from different classes, only the one with the higher importance was plotted and counted towards the results. An example would be a DR output expected to change from the null phase (00) to a logical 1 (10) which, as a result of a fault injection changes from (00) to a logical 0 (01) at an unexpected time, generating both a value fault and a timing deviation event in the checker. If subsequently the expected transition arrived causing a code fault (11) we would consider the fault injection to yield a code fault and disregard the value fault and timing deviation.

It is important to note that the effects of a fault injection were only observed at the outputs of the pipeline with one pipeline stage between the victim wires and the checker. The motivation behind this choice was to let the additional pipeline stages perform logical and temporal masking as it would occur in a normal pipeline. Attaching the strict checker directly to the buffer that was targeted in our fault injection would

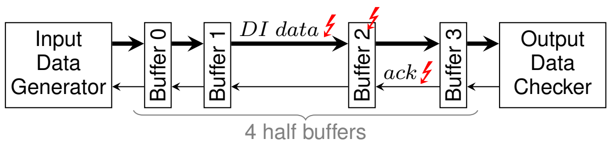


Fig. 4. Simulation Setup

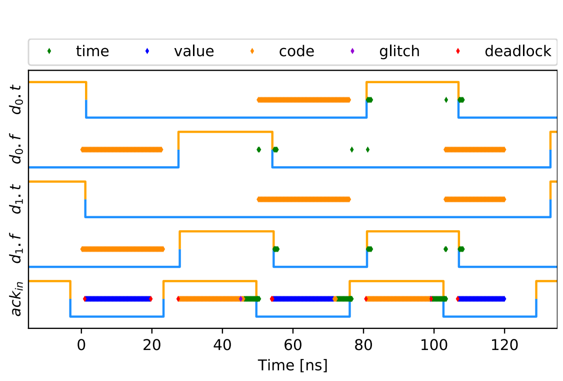
show unnecessarily high fault rates when the often less strict pipeline stage simply does not let many faults through to the checker at the output. While allowing to get better results for buffer styles that propagate fewer faults, this topology also hides some internals that would allow us to differentiate even better between the fault manifestations. When a data word is erroneously removed from the pipeline due to the injected fault, the checker would probably only see a timing deviation and a value fault for some of the following words extracted from the pipeline. The same holds true for spurious data items injected into the pipeline during the null phase.

Fig. 5a shows a simulation trace for the original WCHB. First, the simulation started with a pre-run, where it was counting several rising transitions on the ack_{in} wire to skip the initial phase, where the pipeline only starts to fill up, which is not visible in the figure. Afterwards, 400 ps pulses were injected into the victim wires in 250 ps steps between 0 ns and 120 ns from the end of the pre-run (marked by the 0 ns point on the x-axis). Markers indicate injection times of pulses that had an observable effect on the pipeline, their colors representing the classification of that effect.

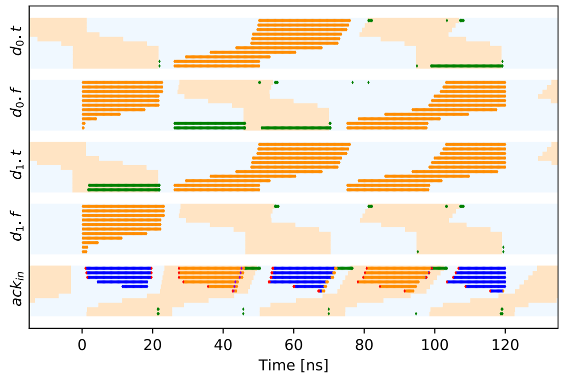
As we have seen from the analysis in the previous section, the timing of the input signals to a pipeline has a high impact on the fault sensitivity windows. By operating the circuit at a specific speed (i.e., handshake rate) the external signals determine how much time the circuit spends in the different protocol phases and hence in its sensitive windows. This is again in stark contrast to a synchronous designs, where the input signals simply don't have that much "power" over the circuit. Hence, a single simulation trace (Fig. 5a) alone only yields very little information about fault behavior of a circuit, since it only shows one specific operation point. For that reason, an essential part of the simulation setup was the possibility to choose delays of the source and sink when generating new input words and acknowledgments respectively. It allowed us running the fault injection simulation of the same pipeline with a variety of timing settings, gradually changing its operation from token-limited (where the pipeline stages mostly wait for valid data words to arrive) to bubble-limited (where the pipeline stages receive valid data at their inputs, but need to wait for the acknowledgment from the succeeding stage before being allowed to store the new data word).

Fig. 5b shows the results of such a timing variation for the same WCHB pipeline simulated in Fig. 5a: For each signal, abutted horizontal stripes represent the results of 11 different simulation runs, in which the pipeline transitioned from bubble-limited (top) to token-limited (bottom) operation. A stripe is colored blue where the signal is low and orange where it is high. Note that Fig. 5a shows the topmost WCHB simulation stripe from Fig. 5b. In the same way the other subgraphs in Fig. 5 illustrate the behavior of the pipeline using the alternative pipeline implementation styles presented in Sec. IV. This representation style, that allows to pack a large amount of information about the fault behavior of a buffer into a single figure, is one of our key contributions of this work.

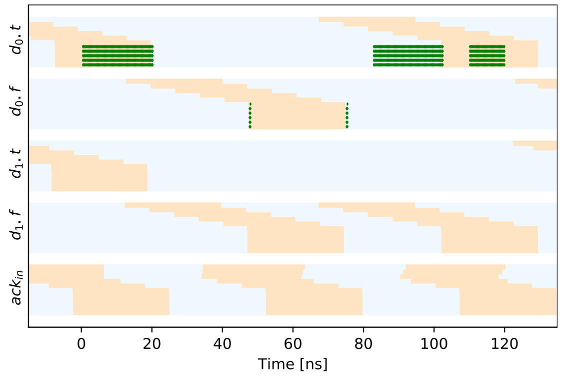
It allows us to see how the external interface timing affects



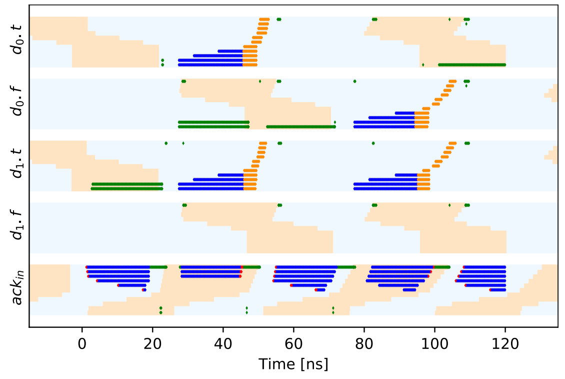
(a) Single trace of classic WCHB



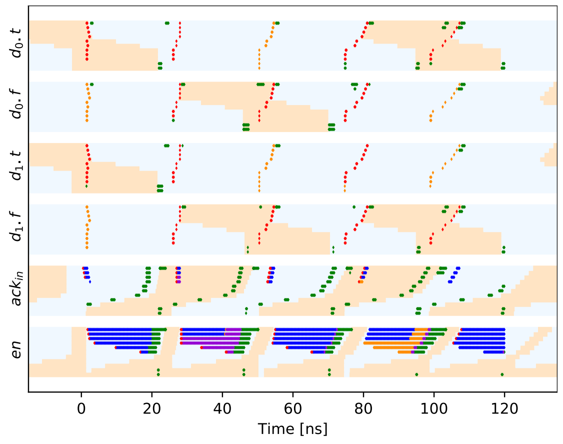
(b) Classic WCHB



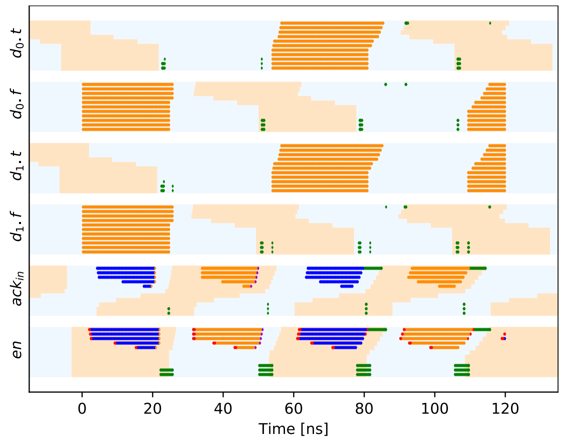
(c) DD WCHB [12]



(d) Locking WCHB [13]



(e) MTDHB [3]



(f) Dual CD WCHB [13]

Fig. 5. Simulation Results

the sensitivity to faults of the different pipeline implementations when subject to SETs. The data rails for the classic WCHB implementation show how the inactive rail is sensitive to produce a code fault the entire time the receiving C gate is accumulating, irrespective of whether the pipeline runs token- or bubble-limited. The Locking WCHB significantly reduces the sensitive windows by correctly preventing code faults in bubble-limited operation after a transition on one of the two rails was captured by a C gate. It only fails to prevent code faults for a short time corresponding to the feedback delay for locking. In token-limited operation, the injected pulse is captured and the correct and expected transition on the other data rail is prevented from turning the valid, albeit incorrect, value into a code fault.

Unsurprisingly, the DD WCHB style proves to be insensitive to SETs in all operation modes whereas the Dual CD WCHB style brings little to no improvement to the sensitivity windows. The MTDHB shows very narrow sensitivity windows on the data rails while the enable signal (the signal that activated the D latches of the buffer) is sensitive most of the time. Faults on this signal also have a wide range of possible effects.

Fig. 6 shows the ratio of injected faults that had an observable effect other than a timing deviation to all injected faults. Note that buffer styles which will be introduced in the

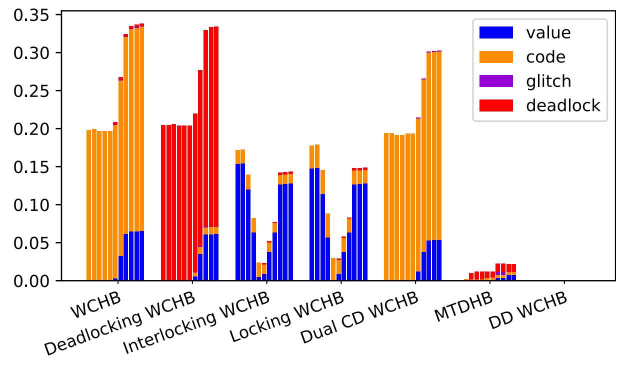


Fig. 6. The number of fault injection simulations with an observable effect on the pipeline

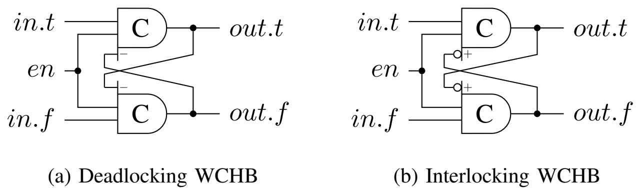


Fig. 7. Proposed WCHB modifications

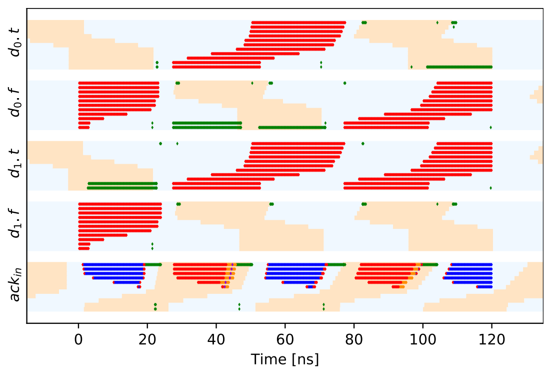
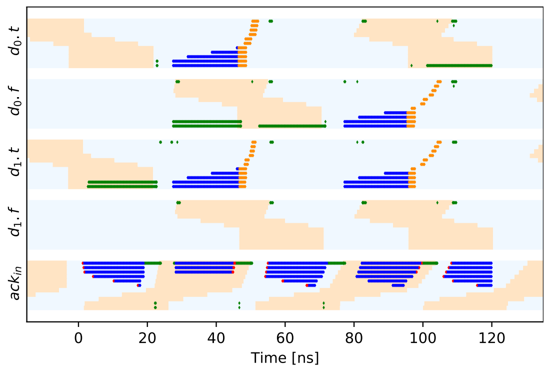



Fig. 8. Simulation results for the proposed buffer (top: Deadlocking WCHB, bottom: Interlocking WCHB)

following section are also included in this figure. To make a fair comparison and prevent speed differences from influencing the results, faults are only considered during one handshake cycle between two rising edges of the acknowledgment wire. For each considered buffer style, the 11 bars show the results for the 11 simulations depicted in Fig. 5 and Fig. 8. It is apparent that the robustness of the simulated pipelines clearly depends on the external timing. By observing a sweep of simulations with varying timing, one can qualitatively assess the effectiveness of fault mitigation techniques like the locking WCHB design instead of barely looking at numbers without the knowledge of whether the circuit was simulated with token- or bubble-limited timing. It could even happen, that a developed fault mitigation technique which is not effective in practice might appear to be advantageous in fault injection simulations, where in fact, additional delay due to added logic gates changed the circuit operation from token-limited to balanced which in itself could bring a significant improvement of fault sensitivity windows. The figure also shows that, depending on the operation mode a circuit is actually used in, it may not pay off to invest in very high overhead fault mitigation strategies, because simple and comparatively cheap approaches also yield quite good robustness.

VI. Proposed Buffer Enhancements

Fig. 7 shows two half-buffer designs (with comparatively low hardware overhead) that try to mitigate the fault-accumulating behavior of the classical WCHB discussed in Sec. IV-A. Both use cross-coupled asymmetric C gates, whose outputs are fed back to the asymmetric input of the respective other gate.

For the *deadlocking WCHB* the feedback inhibits the buffer from entering the null phase if erroneously both C gates are set, effectively causing a deadlock and preventing the circuit from processing possibly faulty data. This can be useful in applications where the correctness of the output is crucial, while deadlocking is not harmful (fail stop). The *interlocking WCHB* only allows the first transition at its input to propagate, thus prohibiting the invalid DR (11). In this sense it is similar to the locking WCHB design proposed in [13] and discussed in Sec. IV-B. However, one important difference is that their approach uses the output of the CD to deactivate the corresponding C gates (i.e., prohibiting them from switching to one) in the buffer. While this allows the use of arbitrary DI codes, it also prolongs the feedback path by the delay of the CD, which keeps the buffer open and thus susceptible to SETs on its inputs for a longer time window. Another more subtle difference to the approach in [13] is that in order to prevent an erroneous input transition from setting a C gate, after completion *all* C gates are switched to a state-holding mode (i.e., the output is driven by the internal storage loop). In our approach only the C gate connected to the rail that did not transition to high is switched to the state-holding mode (keeping its zero value).

Fig. 8 and Fig. 6 show the results of the analysis. It is clearly visible how the deadlocking WCHB, as expected, turns all code faults seen on the data rails in the classic WCHB into deadlocks, albeit without changing the sensitive window. The interlocking WCHB in turn significantly shortens the sensitive windows, since in the bubble limited case the correct transition appears early, and afterwards the interlocking closes the sensitive window. The non-zero size of the remaining window is due to the propagation delay for the locking to become active. Note that these windows are slightly shorter than those found for the approach from [13] in the previous section, due to the shorter feedback path. In the token limited case, there is a potential for faults on the non-switching rail to arrive before the correct transition on the other rail and thus lock the buffer in an incorrect (but valid) state. This is indicated by the windows with value faults that match the size of the sensitive windows in the original WCHB.

VII. Conclusion

We have analyzed the sensitivity of different variations of a WCHB to transient faults. Given that this sensitivity strongly depends on the speed of source and sink, our conclusion was that a systematic analysis requires a visualization of this dependence. Our proposed solution here is a graphical representation of the sensitive windows for each relevant signal, aggregated for different settings of source speed and sink speed, and showing a color code for the observed effect of a fault at the point corresponding to its injection. In another type of visualization we have compiled the absolute number of observed effects summed up over all relevant signals of a given buffer implementation, again plotted as a trend over various settings of sink and source speed (i.e. going from bubble limited to token limited). The required data for these plots are generated in extensive fault injection experiments into a simulation model. Using these compact representations we have identified the key vulnerability of the classical WCHB and proposed two enhancements, namely an interlocking buffer and a deadlocking buffer. For both we have sketched a target use case and given evidence for their proper operation through our graphical analysis.

On the long run, our approach aims at elaborating a generic model of the sensitivity of a QDI block to SETs which allows to correlate the types and rate of effects observed with design parameters like pipeline style, protocol, latch implementation, activity profile and others. Based on this knowledge, the assessment of a specific circuit's susceptibility to SETs should be relatively straightforward.

References

[1] T. Verhoeff, "Delay-insensitive codes — an overview," *Distributed Computing*, vol. 3, no. 1, pp. 1–8, Mar 1988.

[2] A. J. Martin, *The Limitations to Delay-Insensitivity in Asynchronous Circuits*. New York, NY: Springer New York, 1990, pp. 302–311.

[3] 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.

[4] 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.

[5] Y. S. Dhillon, A. U. Diril, and A. Chatterjee, "Soft-error tolerance analysis and optimization of nanometer circuits," in *Design, Automation and Test in Europe*, March 2005, pp. 288–293 Vol. 1.

[6] P. Liden, P. Dahlgren, R. Johansson, and J. Karlsson, "On latching probability of particle induced transients in combinational networks," in *Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing*, June 1994, pp. 340–349.

[7] D. L. Dill, "Trace theory for systematic verification of speed-independent circuits," Ph.D. dissertation, Carnegy Mellon University, 1988.

[8] A. Yakovlev, "Structural technique for fault-masking in asynchronous interfaces," *IEE Proceedings E - Computers and Digital Techniques*, vol. 140, no. 2, pp. 81–91, March 1993.

[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] 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.

[11] 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.

[12] 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.

[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] 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*, July 2007, pp. 113–120.

[15] 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.

[16] 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.

[17] B. Rahbaran and A. Steininger, "Is asynchronous logic more robust than synchronous logic?" *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 4, pp. 282–294, Oct 2009.

[18] 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.

[19] D. J. Barnhart, T. Vladimirova, M. N. Sweeting, and K. S. Stevens, "Radiation hardening by design of asynchronous logic for hostile environments," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 5, pp. 1617–1628, May 2009.

[20] 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.

[21] 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.

[22] Y. Shi and S. B. Furber, "Error checking and resetting mechanisms for asynchronous interconnect," in *Proceedings of the 18th UK Asynchronous Forum, Newcastle University*, 2006, p. 4.

[23] Y. Shi, S. Furber, J. Garside, and L. Plana, "Fault Tolerant Delay Insensitive Inter-chip Communication," in *15th IEEE Symposium on Asynchronous Circuits and Systems*, May 2009, pp. 77–84.

[24] W. Kuang, P. Zhao, J. S. Yuan, and R. F. DeMara, "Design of Asynchronous Circuits for High Soft Error Tolerance in Deep Submicrometer CMOS Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 3, pp. 410–422, March 2010.

[25] 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.

[26] 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.

[27] N. Julai, A. Yakovlev, and A. Bystrov, "Error detection and correction of single event upset (SEU) tolerant latch," in *IEEE 18th International On-Line Testing Symposium (IOLTS)*, June 2012, pp. 1–6.