Towards Comparing the Robustness of Synchronous and Asynchronous Circuits by Fault Injection

Marcus Jeitler and Jakob Lechner

Institute of Computer Engineering,
Vienna University of Technology,
Vienna, Austia
{jeitler, lechner}@ecs.tuwien.ac.at
http://ti.tuwien.ac.at

Abstract. As transient error rates are growing due to smaller feature sizes, designing reliable synchronous circuits becomes increasingly challenging. Asynchronous logic design constitutes a promising alternative with respect to robustness and stability. In particular, delay-insensitive asynchronous circuits provide interesting properties, like an inherent resilience to delay-faults.
This paper presents a new approach for comparing the robustness of synchronous and asynchronous logic. In order to ensure comparability we have developed a tool to automatically transform synchronous designs into their asynchronous counterparts while preserving structural and functional equivalence. Using a saboteur-based fault injection technique, the robustness assessment of both synchronous and asynchronous circuits can then be performed.
At the example of a small-sized test design, this paper demonstrates the capabilities of the proposed approach and, based on these first results, briefly investigates the different behavior of synchronous and asynchronous circuits in the presence of faults.

1 Introduction

The common principle of all asynchronous circuits is a request/acknowledge handshaking protocol. This mechanism regulates the data flow based on the actual speed of the circuit rather than on pessimistic timing assumptions needed in synchronous circuits. Asynchronous circuits can be distinguished by the delay model they employ. In terms of robustness, delay-insensitive circuits are of particular interest since they do not rely on any timing assumptions at all.

Delay-insensitivity in asynchronous circuits is typically implemented using a dual-rail handshake protocol in combination with a certain encoding of data words which allows a completion detection at the recipient. This mechanism

This work is partially funded by the FFG Bridge program: Project "RADIAL" Project Nr: 815458

not only masks delay-faults, the dual-rail encoding with its implicit redundancy also helps in mitigating other fault types in the value domain. While the fault-tolerance properties of asynchronous circuits have been investigated [1], little attention has been paid to the direct comparison of delay-insensitive designs and fault-tolerant synchronous logic so far. Therefore, the aim of the RADIAL project is to compare a synchronous fault-tolerant processor (TMR) with its non-fault tolerant asynchronous counterpart. In addition to a theoretical analysis of the circuit's structure, fault injection experiments will be conducted in the course of the project. The experimental results are expected to provide a detailed insight into the weaknesses and strengths of asynchronous logic wrt. robustness.

The first part of this paper introduces our framework for conducting fault injection experiments in synchronous and asynchronous circuits. The tool-chain consists of two key components: A software tool for automated transformation of synchronous circuits into asynchronous counterparts as well as a powerful fault injection framework that supports the insertion of saboteur units and executes experiments by simulation or hardware emulation. The conversion process for asynchronous circuits will be briefly presented in Section 2. Subsequently, Section 3 gives an overview of the fault injection methodology.

In the second part of the paper a first robustness assessment using our fault injection environment is conducted. In order to keep complexity low, the program counter of a simple processor was used for performing fault injection experiments. Section 4.4 presents the results of these experiments. The paper concludes with Section 5 and provides an outlook on future work.

2 Tranformation of Synchronous Circuits

Since the aim of the RADIAL project is to assess the robustness of synchronous and asynchronous circuits, a common design flow for both circuit types is necessary. Therefore, we have recently developed a software tool for transforming synchronous circuits into *Four State Logic* (FSL) counterparts [2]. FSL is a delay-insensitive, dual-rail encoded implementation style for asynchronous circuits with 2-phase handshaking. The employed data encoding is based on the *Level-Encoded two-phase Dual-Rail* (LEDR) scheme [3]. LEDR represents binary data words in two alternating phases. Phase changes allow to safely separate successive data words and enable the recipient to perform the necessary completion detection. Table 1 shows the codewords for both phases, φ_0 and φ_1.

	φ_0	φ_1
0	00	01
1	11	10

Table 1: LEDR encoding scheme.

An important property of our FSL tool-chain is that the conversion process not only preserves the logical function but also retains the structure of the original circuit at gate-level. This behavior is essential for obtaining comparable results from fault injection experiments. The starting point of the FSL design flow is a conventional synchronous synthesis tool, e.g., *Synopsys*, which translates the clocked circuit description into a gate-level netlist. The latter can subsequently be processed by the FSL conversion tool: Flip-flop components are identified and grouped into registers. A directed graph which represents the data dependencies among the registers is derived from the netlist. Tokens, placed on the graph's edges, can be used to illustrate the data flow. These tokens denote a phase difference between a source and a sink register, which means that a new data word is available for the sink register. Thus, a register may switch if its input edges carry a token and the register's own output tokens have been consumed by all successor stages [4].

The initial token assignment plays a crucial role for the function of an FSL circuit. If the behavior of the original synchronous circuit should be preserved, this assignment is straightforward: In a synchronous system all registers pass their inital value to the successor stages. Thus, all registers of the resulting FSL circuit have to produce initial output tokens. However, in the presence of feedback paths this token configuration causes deadlocks because all registers hold tokens, which need to be consumed first. In order to resolve these deadlocks empty buffer registers have to be inserted into the feedback path (see Figure 1). This deadlock resolution is handled automatically by our FSL tool.

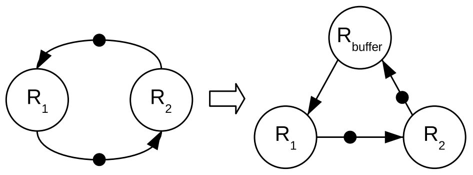


Fig. 1: Initial token assignment, deadlock removal.

Finally, an FSL netlist of the circuit is generated by replacing synchronous flip-flop components with FSL latches and by adding the required acknowledge signals between successive registers. Furthermore, all single-rail signals are converted to dual-rail signals. The asynchronous netlist can then be processed by conventional place & route tools (e.g., Altera QuartusII) for mapping the asynchronous circuit onto FPGA platforms.

3 Fault Injection Framework

In this section we will introduce FuSE, a hardware accelerated HDL-based fault injection environment which supports arbitrary synchronous or asynchronous (with respect to FSL) VHDL circuits.

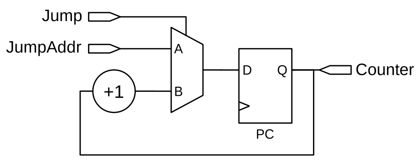
In order to overcome the drawbacks of current simulation- and emulation-based fault injection approaches [5], the FuSE concept integrates both methods in a single tool and allows the user to switch between these modes as required: At an early design stage the user can benefit from executing fault injection experiments within the preferred simulation environment for maximum observation capability. However, when some modules are completed, they can be synthesized and moved to the FPGA, which considerably improves the simulation speed while preserving the visibility of internal signals. This co-simulation support has been achieved by integrating FuSE into the SEmulator® engine – a hardware accelerator for HDL simulations. The resulting environment consists of three core components: The user front-end called Hpe_desk, an HDL simulator (ModelSim in our case) and a rather low cost hardware environment consisting of a prototyping FPGA board equipped with a proprietary PCI-express interface. A more detailed description concerning the SEmulator® and the FuSE integration can be found in [6].

To facilitate fault injection, FuSE uses a source modification approach based on saboteur devices, which currently supports stuck-at 0/1 and bit-flip faults. A saboteur can be activated or deactivated at runtime so that permanent as well as transient faults can be emulated. During the set-up phase the design under test (DUT) is enhanced with the saboteurs and the corresponding control ports. For observation and evaluation of the experiments, additional observation ports can be added. The required modification can either be specified via stylized comments, which are simply added to the source code, or via the Hpe_desk scripting interface. After the configuration the transformation of the source code is automatically performed by the Hpe_desk software.

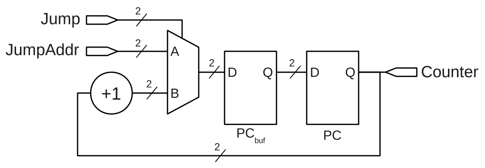
4 Case Study

In order to demonstrate the unified fault injection platform for both synchronous and asynchronous circuits, this section presents some basic experiments with a rather simple test design. The program counter of our future design under test, the SPEAR processor, was chosen as an example, because it is more transparent with respect to the transformation process and the analysis of the conducted experiments than a complex fault-tolerant synchronous processor.

A schematic representation of the program counter (PC) is presented in Figure 2. The circuit consists of the register storing the current address of the PC, an adder and a multiplexer. The multiplexer selects the incremented counter address provided by the adder or an external jump address input. Unless the jump input is active, the program counter is incremented with every clock cycle in the synchronous implementation (Figure 2a). The asynchronous PC, which has been derived from the synchronous version, produces a new counter output with every phase transition of the circuit. Figure 2b shows the corresponding FSL implementation. In order to retain a correct and equivalent behavior the circuit requires an additional buffer register (PC_{buf}) in its feedback path as explained in Section 2.



(a) Synchronous Implementation



(b) Asynchronous Implementation

Fig. 2: Program Counter.

4.1 Experiment Setup

For the fault injection experiments saboteur devices have been inserted in both versions of the program counter. The locations were determined in the synchronous VDHL description and therefore automatically transferred by the FSL transformation process[1]. The respective locations are marked in Figure 3.

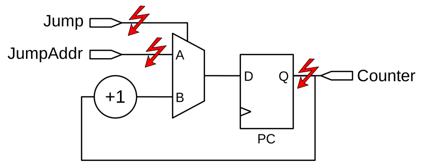


Fig. 3: Injection Location.

Due to the characteristics of the VHDL code, no saboteurs could directly be placed at the output of the adder and the multiplexer. However, faults at these locations can be simulated with the available injection points. As the counter only has one output that could transport an error to the rest of the circuit, the evaluation of the experiments is based on a comparison between this output and a reference output obtained from a fault free execution. The workload, a simple testbench, was configured to increment the counter for 10 cycles, then jump to address 4 and increment from there on.

4.2 Synchronous Fault Injection

In synchronous circuits one or multiple faults can be injected with every active clock edge. Using the inserted saboteur devices each injection point can be configured as a "stuck-at-0", "stuck-at-1" or "bit-flip" fault during the execution of the experiment. Due to the simple structure of our DUT, the outcome of the experiments was not unexpected: Every injected fault has the potential to

[1] Note, that the FSL implementation has a saboteur on each rail of a disturbed signal, doubling the number of saboteur devices.

manifest itself. While "stuck-at" faults can be masked by a matching current state of the respective signal, "bit-flips" always lead to an error. Furthermore, the duration of a fault usually has a different impact on the result. A special case is the bit-flip in the feedback loop which can neutralize itself if the faulty value is affected again in the subsequent clock cycle.

4.3 Asynchronous Fault Injection

As explained in Section 2 FSL uses a dual-rail coding scheme which encodes every binary data value in two phases: φ_0 and φ_1. Due to this encoding, the introduced fault types have different effects in an FSL circuit than in a synchronous circuit. Table 2 lists all possible single-rail and dual-rail faults marked with their observed characteristic.

FSL State		$\uparrow$		$\downarrow$		$\updownarrow$		$\uparrow\uparrow$	$\downarrow\downarrow$	$\updownarrow\updownarrow$	$\uparrow\downarrow$	$\downarrow\uparrow$
φ_0	00	01^P	10^P	00^M	00^M	01^P	10^P	11^V	00^M	11^V	10^P	01^P
	11	11^M	11^M	10^P	01^P	10^P	01^P	11^M	00^V	00^V	10^P	01^P
φ_1	01	01^M	11^P	00^P	01^M	00^P	11^P	11^P	00^P	10^V	10^V	01^M
	10	11^P	10^M	10^M	00^P	11^P	00^P	11^P	00^P	01^V	10^M	01^V
		single-rail faults						dual-rail faults				

Fault Type: $\uparrow$... stuck-at-1, $\downarrow$... stuck-at-0, $\updownarrow$... bit-flip
Characteristic: M... masked, P... phase change, V... value change

Table 2: FSL Fault Characteristics.

The FSL coding scheme describes a valid state transition as the change of a single-rail. As a result, all single-rail faults are either masked or change the phase of the signal. In theory, both outcomes should not lead to an error.

In contrast to single-rail faults, dual-rail faults can also change the value of a signal, which can become an error in our circuit if it is stored in the register. As shown in Table 2, not only fault constellations that are rather unlikely, e.g. both rails of a signal are affected in different ways, but also $\uparrow\uparrow$ and $\downarrow\downarrow$ faults can lead to a value change. However, this behavior only occurs in specific cases with values encoded in φ_0 (00, 11), otherwise the fault has no effect.

During our exhaustive experiments every proposed fault constellation was injected in order to confirm the theoretical assumptions. The corresponding results will be presented in the following section.

4.4 Experimental Results

Single-Rail Faults Although the experiments with single-rail faults did not cause an error, the observed behavior complements the theoretical characteristics presented in the previous section. A fault which is masked in one phase can keep a rail from switching, thereby delaying the execution of the subsequent phase. As a result a masked fault is transformed into a delay fault. This behavior also offers an easy way to generate and investigate delay faults in asynchronous circuits.

If a single-rail fault causes a phase shift the execution of the circuit is not necessarily stopped at once. Depending on a specific fault activation window the inconsistent phase either affects the current phase or the subsequent phase. In the first case, the circuit is halted at once. In the latter case the phase shift can either halt the circuit in the subsequent phase or become masked. However, neither case did lead to an error.

Dual-Rail Faults The considered dual-rail faults either affect both rails the same way or in different ways. As our experiments showed, if a dual-rail fault is masked then it will definitely block the execution of the next phase until the fault is removed again.

If a dual-rail fault results in a phase change, the execution is either blocked or continues with the next phase where the original fault becomes masked or introduces a value change and therefore an error. Nevertheless, if the fault is still active, the further execution will be blocked until the fault is removed again.

The third fault characteristic is an instant value change. If it hits the fault activation window it will cause an error first before the execution is halted. The only exception to this rule is the dual-rail bit-flip fault: Execution will continue and the introduced error will be removed every other cycle if the faulty value is affected again.

4.5 Beyond Experimental Results

Due to the limited complexity of our test circuit certain temporal effects, as e.g. the skew between the signal rails, were not distinctive enough to influence the execution of our experiments. This section will therefore address possible results which depend on a more intricate timing.

Skew-affected Single-Rail Faults As explained in Section 4.4 the investigated program counter is resilient to single-rail faults, which is a direct result of the implemented coding scheme. However, the assumption that single-rail faults can be tolerated mainly depends on the the skew between the inputs of an FSL register. If the skew is small, i.e., all signal rails almost switch simultaneously, then a single-rail fault will either be masked or causes an inconsistent input vector (refer to Table 2). If the skew is sufficiently large, then even a single-rail fault may introduce a value error due to a premature phase change. This behavior is illustrated in Figure 4.

The presented example shows an FSL register with two inputs *A*,*B* and an output *C*. Due to some delay at input *A*, *B* is always assigned a new value (represented by a phase change) prior to *A* in the dataflow diagram. As the inputs are inconsistent during this period, the FSL register has to retain its last

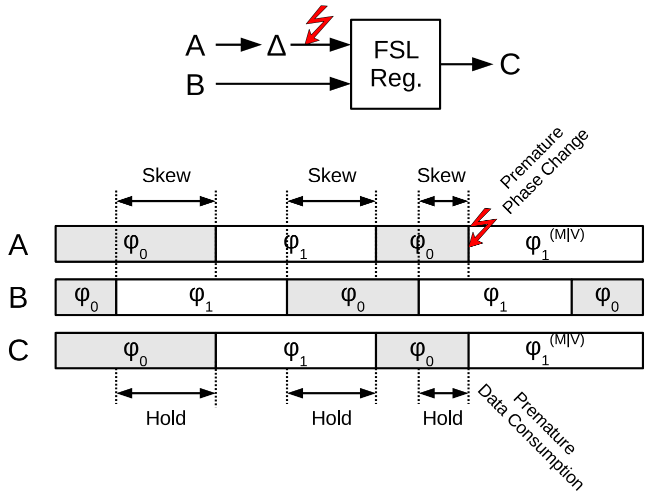


Fig. 4: Skew-affected Single-Rail Fault.

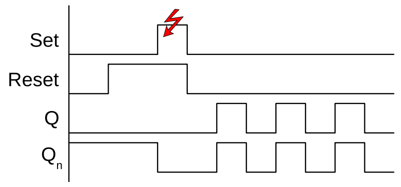
valid output. This “hold” window makes the circuit susceptible for faults, as the register only waits for the inputs to become consistent again. If a single-rail fault causes a premature phase change at A within the “hold” window, possibly wrong data is consumed and a value error might propagate.

By extending this consideration to an arbitrary number of inputs, we can derive the following properties for a single-rail error:

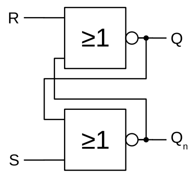
– The fault activation window is determined by the skew of the two slowest inputs.
– A single-rail fault has to hit the slowest rail pair in order to cause a premature data consumption.

Considering the current theoretical analysis we conclude, that the skew, respectively the fault activation window within the test circuit is too small so that all injected single-rail faults become masked according to the coding scheme. Future experiments should therefore be conducted on more complex circuits in order to qualitatively assess the inherent robustness of asynchronous logic.

Metastability Effects While single-rail faults only produce premature phase changes, dual-rail faults can additionally cause invalid transitions in the FSL coding scheme. These may lead to timing issues within the storage elements which are extensively used by FSL designs. Basically all FSL components, sequential registers as well as basic combinational gates, contain RS-latches for holding a data value or preserving a stable state during ongoing input transitions. Although this latch type is very convenient for building FSL gates, first fault injection experiments have uncovered an unpleasant vulnerability: If a fault causes the Set and the Reset line to be active at the same time and subsequently both signals are released almost simultaneously, the output of the latch starts to oscillate. The RS-latch basically behaves like a JK-latch in toggle mode. Figure 5a shows a waveform of this behavior.



(a) Upset on Set signal



(b) RS-latch

Fig. 5: Fault disrupting the function of an RS latch.

RS-latches can be built from two cross-coupled NOR-gates (see Figure 5b). If the *Set* and the *Reset* input are active at the same time, Q and Q_n are both forced low, which violates the equation $Q = not\ Q_n$. When the *Set* and the *Reset* inputs are released again, the NOR-gates form a loop which may start to oscillate.

5 Conclusion and Outlook

The fault injection experiments presented in this paper outline the inherent fault tolerance capabilities of an FSL circuit with respect to arbitrary rail faults. While simple, well-balanced designs are resilient to single-rail faults, more complex architectures might yet be error-prone. In this context, the duration of the "hold" window has been identified as a source for the propagation of single-rail faults. A dual-rail fault, however, does not depend on a specific activation window. Although it can cause a value error at any instant, it only affects the circuit once during its occurrence because the execution will be blocked afterwards. Therefore, the duration of such a fault has no additional effect on the amount of possible errors which is a major difference compared to a synchronous circuit where a fault can become activated periodically.

In future experiments we plan to investigate the propagation of single-rail faults as well as the necessary requirements for the fault activation. In this context, the temporal resolution for the injection will be improved in order to variate the occurrence of a fault within this critical period. The gathered results should then let us develop appropriate mitigation strategies. The improved architecture will be used for the comparison of the synchronous fault-tolerant processor and its non-fault tolerant asynchronous counterpart which has been recently developed [7].

References

1. LaFrieda, C., Manohar, R.: Fault detection and isolation techniques for quasi delay-insensitive circuits. In: DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks, Washington, DC, USA, IEEE Computer Society (2004) 41
2. Lechner, J.: Implementation of a Design Tool for Generation of FSL Circuits. Master's thesis, Vienna University of Technology, Austria (2008)
3. McAuley, A.J.: Four State Asynchronous Architectures. IEEE Transactions on Computers **41**(2) (1992) 129–142
4. Sparsø, J., Furber, S., eds.: Principles of Asynchronous Circuit Design: A Systems Perspective. Kluwer Academic Publishers (2001)
5. Benso, A., Prinetto, P.: Fault Injection Techniques and Tools for Embedded Systems. Kluwer Academic Publishers, Norwell, MA, USA (2003)
6. Jeitler, M., Delvai, M., Reichor, S.: FuSE - A Hardware Accelerated HDL Fault Injection Tool. In: 5th Southern Conference on Programmable Logic, 2009. SPL. (2009) 89–94
7. Jeitler, M., Lechner, J.: Speeding up Fault Injection for Asynchronous Logic by FPGA-based Emulation. to be published at: International Conference on ReConFigurable Computing and FPGAs, 2009. ReConFig. (2009)