# Asynchronous Circuits Transient Faults Sensitivity Evaluation[1]

Y. Monnet, M. Renaudin, R. Leveugle
TIMA Laboratory
46, avenue Felix Viallet
38031 GRENOBLE cedex-FRANCE

{yannick.monnet, marc.renaudin, regis.leveugle}@imag.fr

## ABSTRACT

This paper presents a transient faults sensitivity evaluation for Quasi Delay Insensitive (QDI) asynchronous circuits. Because of their specific architecture, asynchronous circuits have a very different behavior than synchronous circuits in the presence of faults. We address the effects of transient faults in QDI circuits and describe the causes that lead the faults to be memorized into one or more soft errors. Therefore, a refined fault sensitivity criterion is defined for this class of circuits. This methodology enables us to point out the weak parts of a circuit. An analysis tool is implemented to support this evaluation. This tool provides a quantitative study of the fault sensitivity, and enables us to compare the robustness of different architectures of a circuit along the steps of its design flow. The objective of this work is to evaluate the circuits robustness against natural faults (single fault model) and intentional fault injection (multiple faults model).

**Categories and Subject Descriptors**

B.8.1 [**Performance and reliability**] Reliability, Testing, and Fault-Tolerance

**General Terms:** Design, Reliability, Security

**Keywords:** Asynchronous circuits, Quasi Delay Insensitive, transient fault, fault model, simulation.

## 1. INTRODUCTION

A transient fault is a current transient, for instance induced by the hit of a particle or a crosstalk, which can propagate in combinational logic. When the fault is propagated up to a memory point, the transient pulse can be memorized into a soft error. This may leads to an incorrect behavior of the circuit. When a wire forks to multiple gates, a single current transient on this wire may cause multiple transient faults, and potentially multiple soft errors.

Most of the integrated circuits today are synchronous. Their activities are controlled by a global clock which triggers at the same time the memorization of the complete state of the circuit. The behavior and the sensitivity of synchronous circuits exposed to fault injections have widely been studied [1, 2, 3, 4].

Asynchronous circuits represent a class of circuits which are not controlled by a global clock but by the data themselves. QDI circuits are asynchronous circuits that operate correctly regardless of gates delays in the system. Their delay-insensitive property makes them naturally robust against some categories of faults such as delay faults. Thus, QDI circuits can be attractive to design fault tolerant systems [11, 12, 13]. In [5], a general definition of asynchronous circuit's sensitivity to fault injection was proposed.

In this paper, we describe a refined fault sensitivity criterion which takes into consideration the fault effects at the behavioral level. An analysis tool is implemented to evaluate the fault sensitivity in a simulation environment. A metric gives the possibility to compare the transient fault sensitivity of different architectures.

This paper is organized as follows. Section 2 introduces the QDI asynchronous technology. In the context of QDI circuit's analysis, the fault models used in this work are given in Section 3. Section 4 introduces the fault sensitivity criterion as defined in [5] and refines it. We examine the faults effects at the behavioral level. The analysis tool is presented in Section 5 as well as the design flow that includes this tool. Section 6 presents a case study. We compare different architectures of an asynchronous DES crypto processor module to validate the evaluation and to show the impact of the architecture on the fault sensitivity. Section 7 concludes the paper.

## 2. ASYNCHRONOUS LOGIC: QUASI DELAY INSENSITIVE

### 2.1 Overview

An asynchronous circuit is composed of individual modules which communicate to each other by means of point-to-point communication channels [6]. Therefore, a given module becomes active when it senses the presence of incoming data. It then computes and sends the result to the output channels. Communications through channels are governed by a protocol which requires a bi-directional signalling between senders and receivers (request and acknowledgment). They are called Handshaking protocols (Figure 1).

[1] This work is partially supported by the RNRT Duracell project.

*DAC 2005*, June 13–17, 2005, Anaheim, California, USA.
Copyright 2005 ACM 1-59593-058-2/05/0006…$5.00.

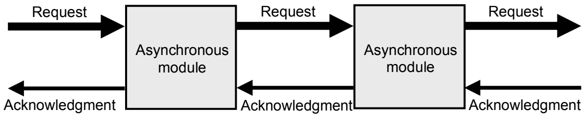

**Figure 1. Handshake based communication between modules**

The communication protocol is the basis of the sequencing rules in asynchronous circuits. There are two main classes of handshaking protocols: two-phase and four-phase. Only the four-phase protocol is considered in this work. Figure 2 describes the four-phase protocol, which requires a return to zero phase for both data requests and acknowledgments. In phase 1, a valid data is detected. This data is acknowledged in phase 2. Then the data is re-initialized in phase 3 (return to zero phase) and the acknowledgment signal is reset in phase 4.

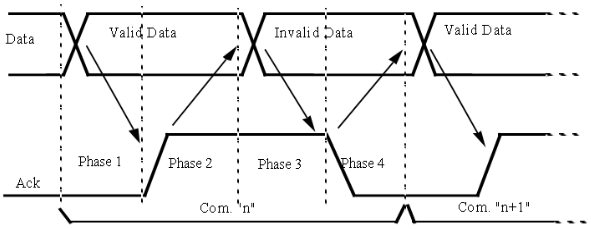

**Figure 2. Four-phase handshaking protocol**

## 2.2 Memory elements

The memory cells used in asynchronous QDI circuits are Muller gates. The Muller gate (or C-Element) generates a rising transition when rising transitions occur at all the inputs and generates a falling transition when falling transitions occur at all the inputs [6].

## 2.3 Computational blocks and memory blocks

Figure 3 shows a general structure of an asynchronous stage. Similarly as in synchronous circuits, it is composed of a computational logic block and a memory block (registers). The computational block computes data inputs. The memory block not only stands for registers but also implements the four-phase communication protocol.

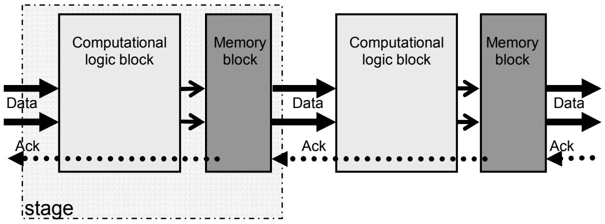

**Figure 3. Basic structure of an asynchronous stage**

Both logic and registers implementation contain standard combinational gates (AND, OR, NAND …) as well as Muller gates.

- In the memory block, Muller gates are used to implement the communication protocol between the next and the previous asynchronous stages. Concretely, some of the inputs of the Muller gates are connected to the output of the computational logic, while other inputs are connected to the acknowledge signal of the next stage (synchronization function).
- In the computational logic block, all inputs of Muller gates implemented in this block are directly or indirectly connected to data inputs. Here, Muller gates are mainly used as a logical "AND" operator to compute incoming data. However, their state-holding nature is necessary to ensure the QDI properties of the circuits, while this property would be lost by the use of standard "AND" gates.

The distinction between computational blocks and memory blocks is necessary to focus on the sensitivity evaluation. A fault that propagates to a Muller gate implemented in the computational block doesn't have the same consequences as a fault that propagates to a Muller gate implemented in the memory block. The next section explains in detail this difference.

Finally, the *global circuit state* is defined as the state of all its Muller gates implemented in memory blocks. Muller gates in computational blocks are excluded. Indeed, they do not hold data information at the behavioral level.

# 3. FAULT MODELS

Many fault models based on different abstraction levels (transistor-level, gate-level, macro-cells ...) have been proposed in the test domain. In the present paper, the fault effect is considered as a logical perturbation in the circuit. Therefore, whatever the physical effects causing faults are, we assume that the fault eventually becomes one or more logical errors. Transient fault models generally used in test are considered in the global scope of this work. However, it is important to clearly define these models when applied to asynchronous circuits. We use the following terminology:

- *Transient fault*: a transient fault is a current transient, for instance induced by the hit of a particle, which can propagate in the circuit, thereby corresponding to a signal toggle at a logical level with a short duration.

In the context of asynchronous blocks defined in Section 2.3, any node of the computational block can be affected, including nodes connected to Muller gates. If a transient fault is propagated to a Muller gate input, it can be captured. With a rigorous point of view, this captured fault no longer represents a transient fault, since it is memorized. However, this error doesn't affect the global state of the circuit as defined in the previous section. Figure 4 (a) represents a transient fault that occurs on a computational node. It causes a pulse which duration is *t1*. (b) is the same transient pulse that has been propagated and captured by a Muller cell in the computational block. Because all inputs of this gate are connected to data inputs, this gate finally resets during phase 3 of the communication protocol. Therefore, the fault can be interpreted as a transient fault which duration is *t2*. Like a transient, this fault may propagate to the output of the computational block and may be memorized in the memory block.

As a conclusion, any fault that propagates through a computational logic block is considered as a transient fault in the scope of this work.

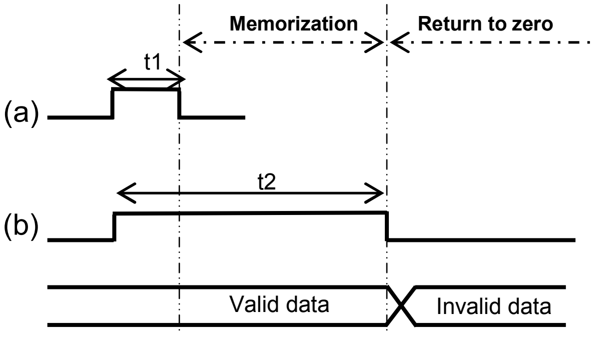

**Figure 4. A transient fault captured in a computational logic block**

A transient fault can occur in a memory block as well, because it is also composed of combinational gates in order to implement the communication protocol. However, only combinational gates of these blocks are included. The case of Muller gates implemented in a memory block is considered in the following definition:

- *Soft error*: A soft error is an abnormal modification of the global state of the circuit. It can be either the memorization of a transient fault which has propagated to a Muller gate in a memory block, or a fault injected straight upon this Muller gate. The latter can be compared to a Single Event Upset (SEU) in synchronous circuits. In any case, a soft error results in one or more memory bit-flips.

Although this paper is focused on transient fault effects, delay faults have to be defined because they can be a consequence of transient faults:

- *Delay fault*: A gate delay fault modifies the time needed for a transition to occur at the gate's output. This fault can be considered as a temporary "stuck-at" fault throughout the fault's activity.

Figure 5 shows the relationship between a transient fault and its possible consequences on the circuit's behaviour. From now, when a Muller cell is mentioned, it is implicitly a Muller cell implemented in a memory block.

Case 1: The transient fault is filtered. The pulse can be either logic-blocked, or naturally attenuated until it disappears.

Case 2: The transient fault is propagated up to an input of a Muller cell, but it is not memorized.

Case 3: The transient fault is propagated up to an input of a Muller cell, and it is memorized. However, this cell was selected to flip in the normal execution process. This case is called *premature firing*: the cell was supposed to flip, but due to the fault it flipped sooner than it should have. This case can be assimilated to a delay fault on the output of the Muller cell. Delay faults don't affect the circuit logical function, except if they occur on an isochronic branch, since it is the only timing assumption in QDI circuits. In the present case, the delay fault has no consequences on the circuit function. If a Muller gate output is connected to a fork, both branches are affected by the delay fault. Similarly, the transient fault can cause a *delayed firing* if the output of the Muller gate fires later as it should have.

Case 4: The transient fault is propagated up to a Muller cell, and it is memorized. This gate wouldn't have flipped in the normal execution process. Therefore, the transient fault is memorized into a soft error which can lead the circuit to fail.

The next section discusses the sensitivity criterion and examines the cases presented in Figure 5.

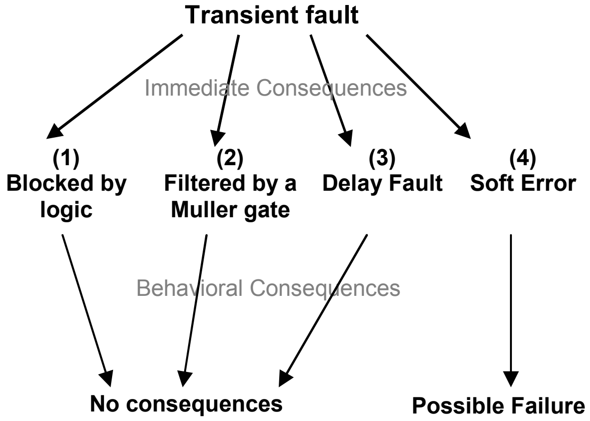

**Figure 5. Transient faults possible effects on QDI circuits**

## 4. REFINED SENSITIVITY DEFINITION

This section defines a criterion for the asynchronous circuits transient fault sensitivity. The Muller gate sensitivity defined in [5] is first introduced and refined. A set of states is associated with each Muller gate instance implemented in a memory block. Then a Muller gate fault-sensitivity is specified according to the fault models previously mentioned. Finally, we define a fault sensitivity criterion for QDI asynchronous circuits.

### 4.1 Muller gate sensitivity

Definition: An N-input Muller gate is said to be M-sensitive to 0 (respectively 1) if, and only if, exactly M of its inputs as well as its output are equal to 0 (respectively 1). In this case, if M faults are injected (or propagated) to these M inputs, the gate generates a rising (respectively falling) transition.

Figure 6-(a) shows a 2-input Muller gate which state is 1-sensitive to 1. If a faulty transition occurs on the first input (a transition which is a consequence of a propagated transient fault), the output is reset and this transition is memorized. Note that the gate is not in a 1-sensitive to 0 state because its output is already set to one. A fault occurring on its second input has no effect on the output, it is filtered.

Obviously, 2-input gates are highly sensitive to a single fault. Figure 6 (b) shows a 5-input Muller gate which state is 3-sensitive to 0. A single fault occurring on any of its input is filtered. However, three simultaneous faults occurring on the three first inputs generate a flip to 1.

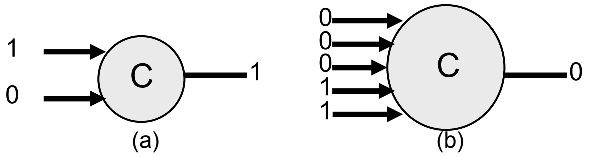

**Figure 6. A "1-sensitive to 1" Muller gate (a) and a "3-sensitive to 0" Muller gate (b)**

The set of states of an N-input Muller gate is then defined according to four cases:

1 – M-sensitive to 0 (M є [1 ... N-1])
2 – M-sensitive to 1 (M є [1 ... N-1])
3 – Set
4 – Reset

Cases 1 and 2 are sensitive ones. Each case includes N-1 states, from 0-sensitive to (N-1)-sensitive. Case 3 is the gate set state; all inputs are equal to 1. Case 4 is the gate reset state; all inputs are equal to 0. As a result, an N-input Muller gate is characterised by 2N disjoined stable states. Set and Reset are not considered as sensitive because transient faults cannot be memorized in these states.

## 4.2 Sensitivity Validation/Invalidation

Figure 7 shows the states evolution for a Muller gate in a normal execution process. The initial state of the gate is "reset". Then, the gate holds a succession of sensitive states until it comes back to the "reset" state (a) or changes to the "set" state (b). Similarly, a symmetric case could be presented with the "set" state as initial state.

### *4.2.1 State sensitivity validation*

In Figure 7 (a), the Muller gate output never changes. From the circuit's behavior point of view, it means that this gate was finally not selected to hold a data. However, the gates held a collection of "sensitive to 0" states before to come back at the "reset" state. Therefore, these states have to be validated as *fault-sensitive* states. N faults that would occur in an "N-sensitive to 0" state would provoke a rising transition on the output, thus generating a soft error.

### *4.2.2 State sensitivity invalidation*

In Figure 7 (b), the Muller gate finally changes its state to a "set" state, which means that this gate was selected to memorize a data. N fault occurring in an "N-sensitive to 0" state would generate a premature firing. They would force the gate to join the "set" state prematurely. As previously explained, this transition doesn't affect the circuit logical function. As a consequence, the sensitive states held by the gate have to be invalidated, because they were finally not fault-sensitive.

### *4.2.3 Example*

Figure 8 presents a simple circuit called a Half-Buffer. When the next stage is ready to receive a data, the Ack signal is set to 1. Muller gates M00 and M01 are able to memorise the data ($I_0$, $I_1$). A Dual rail code is used, which means that only one rail $I_0$ or $I_1$ is set to encode a data bit (0 or 1).

Let's assume that the next stage is ready to receive a data. Ack is set to 1. The sensitive time for M00 and M01 is the time needed for the data ($I_0$, $I_1$) to arrive. Assuming that $I_0$ is selected to hold a data, then M00 switches to 1. The sensitive time of M00 is invalidated because this gate was selected to memorise a data. However, M01 didn't switch; this gate finally comes back to its reset state: its sensitive time is validated.

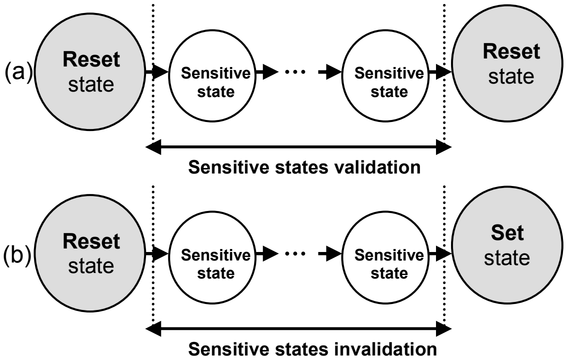

**Figure 7. State evolution scenarios for a Muller gate in a normal execution process**

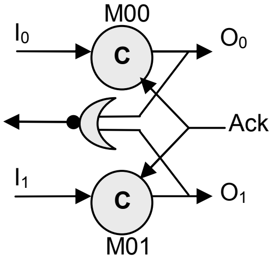

**Figure 8. Half-Buffer**

## 4.3 Asynchronous circuits sensitivity evaluation

The global circuit state was previously defined as the state of all its Muller gates implemented in the memory blocks. In a simulation environment, we are able to monitor and validate/invalidate each sensitive state for all the Muller gates. Thus, a metric is given to evaluate the sensitivity of each Muller gate. This metric is defined as the total time spent in the sensitive states by this gate with respect to the total time this gate was monitored. Finally, the N-sensitivity metric of a circuit is defined as the meantime spent in the N-sensitive states by all the monitored gates.

We can afford to dynamically draw up a circuit sensitivity map, identifying fault-sensitive blocks or gates. The most sensitive memory points are those which are frequently in a 1-sensitive state, because a single fault is able to be memorized when the gate holds this state.

If a gate is at minimum 2-sensitive (or more), it is single fault resistant. A single fault occurring on any of its inputs is never memorized.

## 4.4 Fault propagation

Once it was injected, the transient fault propagates through the circuit. The fault propagation is not studied in the scope of this work. This problem has been already largely studied [1, 7]. An evaluation of activated paths, reconvergent fan-out, and logic-blocking of faults is not relevant for the contribution of the present work.

# 5. TOOL IMPLEMENTATION

## 5.1 Design flow

Figure 9 shows the global flow for designing asynchronous VLSI circuits. It involves a high-level description language, the Communicating Hardware Processes (CHP) language. The CHP program is compiled with the synthesis environment tool TAST.

The synthesized circuit is described as a gate-level netlist in Verilog format. The sensitivity analysis tool can be used with behavioural libraries to perform a first succinct analysis. For a more accurate result, the analysis has to be performed with a back-annoted Verilog netlist after the place and route step. Timing information is provided by a SDF file.

The fault-sensitivity analysis tool directly interacts with the simulator as explained in the next subsection.

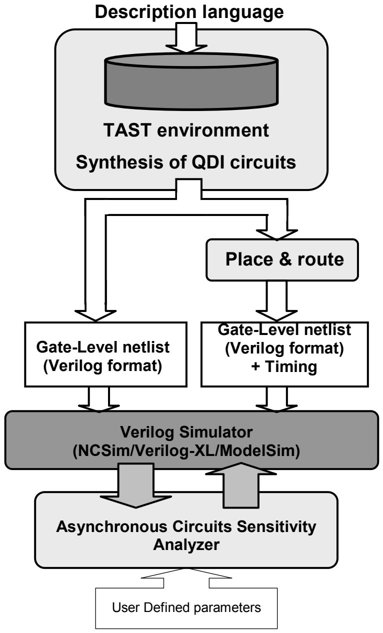
Description language
TAST environment
Synthesis of QDI circuits
Place & route
Gate-Level netlist (Verilog format)
Gate-Level netlist (Verilog format) + Timing
Verilog Simulator (NCSim/Verilog-XL/ModelSim)
Asynchronous Circuits Sensitivity Analyzer
User Defined parameters

**Figure 9. QDI circuits design flow**

## 5.2 Implementation

The algorithm was implemented using the C language. The algorithm is able to control the simulator using PLI (Programming Language Interface) which allows directly interacting with the simulation. The implementation consists of only a few hundreds of C code lines. The used PLI routines are standard, and the same implementation of the tool runs on several commercial event-driven simulators. The tool is able to analyse complex circuits without penalizing the simulation time.

User can decide to analyse all or part of the Muller gates implemented in the circuit. A particular module may for example be deeply analysed because of its critical nature, while other modules are not considered. A set of commands added in the Verilog netlist allows the user to define which modules to analyse. User can decide which gates to monitor by specifying the library cell names in a specific file. The ports that are to analyse/ignore can also be specified because it is necessary to ignore some port names that are not relevant for the analysis (RESET port for instance). Analysis start time and end time can be specified as well.

At the end of the simulation, a report gives the global N-sensitivity of the circuit under test. Each monitored gate is shown in detail and the tool points out the most N-sensitive gates.

# 6. CASE STUDY

The security of systems such as smart cards relies on the ability of the smart card to perform cryptographic operations while keeping the key secret. A particular threat is the use of fault injections to attack such devices [8, 9].

To validate the methodology, several analyses are applied to the design of an asynchronous DES crypto-processor. The DES algorithm is described in [10]. The analysis is focused on the ciphering Data-path module. Indeed, this module is known as critical for the system security. We present in Section 6.1 an analysis of the ciphering block and its basic architecture. Then, Section 6.2 proposes a hardened architecture of this module and shows that the fault sensitivity can be highly reduced by design.

## 6.1 Basic architecture analysis

Figure 10 presents the module that is analyzed. The computational block is composed of 8 SBoxes which role is to switch data, and a 32 bit XOR operator. The result of the XOR operation (between $L_{i-1}$ and the SBoxes output) is stored in the memory block.

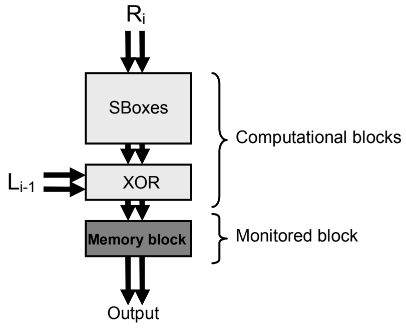

**Figure 10. Architecture of the basic monitored module**

The memory block is composed of 64 two-inputs Muller gates. 64 gates are needed to memorize a 32 bit data because the dual-Rail logic is used. All the Muller gates are monitored.

A Benchmark is applied to analyse the circuit in a behavioural environment. The chosen benchmark is a single DES encryption. We start to monitor gates sensitivity as soon as the DES starts the encryption, and stop the analysis at the encryption end. Table 1 shows the analysis results.

**Table 1. Sensitivity analysis results for the basic module**

| | |
|---|---|
| **Simulation time :** | 53 300 ps |
| **Average 1-sensitive time :** | 14 400 ps |
| **Sensitive ports :** | Input A |

The gates were monitored during 53.3 ns. On average, each gate spent 14.4 ns in a 1-sensitive state, which means they were sensitive to a single transient fault more than 27% of the time. Moreover, the sensitive port is the input A of the gate. This port is connected to the output of the computational block. During the computing time of data, some of the Muller gates are in a sensitive state.

## 6.2 Hardened architecture analysis

We propose a hardened architecture to demonstrate the influence of the architecture on the circuit sensitivity. The hardening strategy is out of the scope of this work. We chose a well-known duplication method scheme to harden the cipher module. More efficient hardening techniques were studied in [11, 12, 13]. Both SBoxes and XOR operation were duplicated.

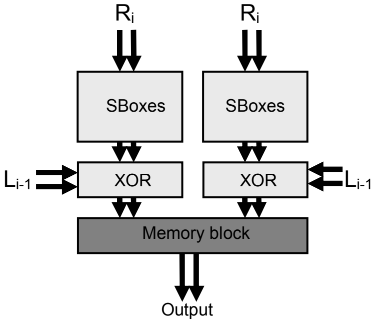

**Figure 11. Architecture of the hardened monitored module**

Registers were not duplicated but their architecture changed from two-input Muller gates to four-input Muller gates. The same bench was applied as for the basic circuit. Table 2 presents the results.

**Table 2. Sensitivity analysis results for the hardened module**

| | |
|---|---|
| **Simulation time :** | 53 300 ps |
| **Average 1-sensitive time :** | 0 ps |
| **Average 2-sensitive time :** | 14 550 ps |
| **Sensitive ports :** | Inputs A,B |

Although four-inputs Muller gates are slower than two-input Muller gates, the encryption time is constant because this part of the circuit is not the critical path. The Duplication technique makes the 1-sensitivity to shift to a 2-sensitivity. Sensitive inputs A and B are the duplicated computational block output. As expected, this hardening strategy makes the module single transient fault tolerant.

# 7. CONCLUSION

We presented a detailed transient fault sensitivity criterion for QDI asynchronous circuits. Fault models were defined in the context of asynchronous circuits. Consequences of the propagation of a transient fault were identified and examined at the behavioral level. For the memory gates used in circuits, we defined states that were identified as fault-sensitive states. Finally, a tool was implemented to evaluate the sensitivity of the circuits. This tool is included in the design flow of the circuits and gives a metric to compare asynchronous circuits architectures. Finally, we showed that the architecture of the circuit significantly influences its transient fault sensitivity. Based on this analysis, future work will be focused on hardening techniques and their efficiency.

# 8. REFERENCES

[1] R. Leveugle, K. Hadjiat, “Multi-level fault injections in VHDL descriptions: alternative approaches and experiments”, *Journal of Electronic Testing: Theory and Applications (JETTA)*, Kluwer, vol. 19, no. 5, October 2003, pp. 559-575.

[2] D. Alexandrescu, L. Anghel, M. Nicolaidis, “New methods for evaluating the impact of single event transients in VDSM ICs”, *The IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Vancouver, Canada, November 6-8, 2002, IEEE Computer Society Press, Los Alamitos, California, 2002, pp. 99-107.

[3] M. Bellato, P. Bernardi, D. Bortolato, A. Candelori, M. Ceschia, A. Paccagnella, M. Rebaudengo, M. Sonza Reorda, M. Violante, P. Zambolin , “Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA”, *Design Automation and Test in Europe Conference (DATE)*, February 16-20, 2004, pp. 584-589.

[4] M. Sonza-Reorda, M. Violante, “Accurate and efficient analysis of single event transients in VLSI circuits”, *9th IEEE International On-Line Testing symposium*, Kos, Greece, July 7-9, 2003, pp. 101-105.

[5] Y. Monnet, M. Renaudin, R. Leveugle, “Asynchronous circuits sensitivity to fault injection”, 10th IEEE International On-Line Testing Symposium, Madeira Island, Portugal, July 12-14, 2004.

[6] M. Renaudin, “Asynchronous Circuits and Systems: a promising design alternative”, *Microelectronics-Engineering Journal, Elsevier Science*, Guest Editors: P.Senn, M. Renaudin, J. Boussey, Vol54, N°1-2, 2000, pp.133-149.

[7] D. Alexandrescu, L. Anghel, M. Nicolaidis, “Simulating single event transients in DVSM ICs for ground level radiation”, *3rd IEEE Latin American Test Workshop (LATW'02)*, Montevideo, Uruguay, February 10-13, 2002.

[8] E. Biham, A. Shamir, “Differential Fault Analysis of Secret Key Cryptosystems”, *Advances in Cryptology CRYPTO 1997*, LNCS 1294, pp. 513-525, 1997.

[9] M. Renaudin, G. F. Bouesse, P. Proust, J-P Tual, L. Sourgen, F. Germain, “High-Security Smartcards”, *Design Automation and Test in Europe Conference (DATE)*, Feb 16-20, 2004.

[10] NIST, Data Encryption Standard (DES), FIPS PUB 46-2, National Institute of Standards and Technology. http://csrc.nist.gov/csrc/fedstandards.html

[11] C. LaFrieda, R.Manohar, “Fault Detection and Isolation Techniques for Quasi Delay-Insensitive Circuits”, *International Conference on Dependable Systems and Networks (DSN'04)*, Italy, June 28 - July 01, 2004, p.41

[12] Wonjin Jang, Alain J. Martin, “SEU-Tolerant QDI Circuits”,*11th IEEE International Symposium on Asynchronous Circuits and Systems*, New York City, USA, March 13-16, 2005, pp. 156-165.

[13] Y. Monnet, M. Renaudin, R. Leveugle, “Hardening Techniques against Transient Faults for Asynchronous Circuits”, *11th IEEE International On-Line Testing Symposium*, Saint Raphael, France, July 6-8, 2005.