UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
ENGENHARIA DA COMPUTAÇÃO

LUIGI VAZ FERREIRA

Detecting transient faults in the Configurable Reconfigurable Core architecture without false error signals

Final Report.

Prof^{a}. Dr^{a}. Fernanda Lima Kastensmidt
Advisor

Porto Alegre, December 2013

CIP – CATALOGING-IN-PUBLICATION

Ferreira, Luigi Vaz

Detecting transient faults in the Configurable Reconfigurable Core architecture without false error signals / Luigi Vaz Ferreira. – Porto Alegre: UFRGS, 2013.

100 f.: il.

Final Report (Graduation) – Universidade Federal do Rio Grande do Sul. Engenharia da Computação, Porto Alegre, BR–RS, 2013. Advisor: Fernanda Lima Kastensmidt.

1. Fault tolerance. 2. Transient faults. 3. Circuit level time-redundancy. 4. CGRA. I. Kastensmidt, Fernanda Lima. II. Título.

AGRADECIMENTOS

No âmbito acadêmico, agradeço a Professora Fernanda Lima Gusmão Kastensmidt da Universidade Federal do Rio Grande do Sul por sua orientação na realização deste trabalho. Agradeço também, ao Professor Thomas Schweizer da Eberhard Karls Universität Tübingen pela oportunidade a mim fornecida de contribuição no projeto ARES no ano de 2012.

Agradeço aos meus pais, Cledi Teresinha Barboza Vaz e Edson Luis Gomes Ferreira, e ao meu padrastro, Jesse James Vivian, pelo apoio durante toda a jornada de realização desta graduação. Bem como ao resto de minha família pela compreensão de minhas ausências nas reuniões e momentos importantes devido a atividades vinculadas ao curso de Engenharia da Computação.

A Mariana Cezimbra Franzen, agradeço pelo amor, compreenção e companheirismo a mim sedidos durante todos estes anos de nosso relacionamento. A ela, agradeço também o nosso filho, Bernardo Franzen Ferreira, o qual ainda esta em geração em seu ventre.

Agradeço aos meus amigos e colegas de curso Mauricio Kempf Bonatto, Mateus Felipin Dalepiane e Alexandre Hertzog pelas horas de descontração e momentos de alegria nos intervalos das aulas.

Por último e não menos importante, agradeço aos meus amigos Rafael Padilha Ferreira e Jean Michel Ramos Prestes pelas horas de estudo na biblioteca e pelas corridas diárias para esfriar a cabeça.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

CRC	Configurable Reconfigurable Core
PE	Processor Element
FSM	Finite State Machine
FU	Functional Unit
FEHM	Flexible Error Handling Module
TMR	Triple Modular Redundancy
DWC	Duplication With Comparison
DEDC	Data Error Detection Component
TEDC	Transition Error Detection Component
ECC	Error Correction Codes
SBU	Single-bit upset
MCU	Multiple-cell upset
MBU	Multiple-bit upset
SET	Single-event transient
SEFI	Single-event functional interrupt
SEL	Single-event latchup
ECC	Error Correction Codes
ED	Essential Detector
DSTB	Double Sampling with Time Borrowing
SDF	Standard Delay Format

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Fault tolerance is an important issue to worry about in the computing world. The detection of errors provided by transient faults, among others source of errors, in a determined region of a computer architecture is necessary to increase the reliability of the architecture. Circuit level time-redundancy (NICOLAIDIS, 1999) is a good technique to detect transient errors (those caused by transient faults) with low impact in area in a first moment. This work presents two implementations of the technique and the conditions we must respect in order to maximize the error detection without generating false errors signals. In addition, these implementations are used to construct the component that is part of a flexible error handling module (FEHM) (SCHWEIZER et al., 2012). This module is incorporated in the Configurable Reconfigurable Core (OPPOLD et al., 2007) architecture in order to detect and mask errors caused by permanent and transient faults. As a result, the simulations confirm that the conditions formulated avoids the false errors signals generation. However, the synthesis results shows that the technique can impact, not only in the area, but also in the performance of the circuit. Furthermore, fault injection simulations show that the FEHM has a vulnerability in the presence of transient faults.

Keywords: Fault tolerance, transient faults, circuit level time-redundancy, CGRA.

Detectando falhas transientes na arquitetura *Configurable Reconfigurable Core* sem sinais falsos de erro

RESUMO

Tolerância a falhas é uma questão importante a se preocupar no mundo da computação. A detecção de erros providos de falhas transientes, entre outras fontes de erros, em uma determinada região de uma arquitetura de computador é necessária para aumentar a confiança da arquitetura. Redundância no tempo a nível de circuito (NICOLAIDIS, 1999) é uma boa técnica para detectar erros transientes (erros causados por falhas transientes) com um baixo impacto na área a priori. Este trabalho apresenta duas implementações desta técnica, bem como as condições que são necessárias respeitar para maximizar a detecção sem a geração de sinais falsos de erro. Adicionalmente, as implementações são usadas para construir o componente que é parte de um módulo flexível de tratamento de erro (*flexible error handling module* - FEHM -) (SCHWEIZER et al., 2012). Este módulo é incorporado na arquitetura Configurable Reconfigurable Core (CRC) (OPPOLD et al., 2007) com o objetivo de detectar e mascarar erros causados por falhas permanentes e transientes. Como esperado, as simulações confirmam que as condições formuladas evitam a geração de sinais falsos de erros. Entretanto, os resultados de sintese mostram que a técnica pode impactar, não somente em área, mas também na performance do circuito. Além disso, simulações com injeção de falhas apresentam que o FEHM possui uma vulnerabilidade na presença de falhas transientes.

Palavras-chave: tolerância a falhas, falhas transientes, redundância no tempo a nível de circuito, CGRA.

1 INTRODUCTION

Fault tolerance is the ability of a system to continue correct performance of its tasks after the occurrence of hardware or software faults. A fault is simply any physical defect, imperfection, or flaw that occurs in hardware or software. Fault tolerance can be achieved in systems by incorporating various forms of redundancy, including hardware, information, time, and software redundancy (JOHNSON, 2000).

A fault can causes an error if not masked by the application, and consequently, a error can causes a failure which can be catastrophic. A failure is an event that occurs when the delivered system service deviates from correct service. The error is the service deviation and the cause of the error is the fault (AVIZIENIS et al., 2004). On semiconductor devices, a service is the logic value of an output. If an output has a different logic value that it is supposed to have, then, a service deviation has occurred.

Upsets in the nodes of semiconductors are the sources of faults. It happens when energetic particle passes through the pn-junction of a CMOS transistor in the off state, a short is momentarily created between the substrate and the struck drain terminal. The amount of charge that is collected produces a transient current pulse that lasts until the deposited charge disappears by recombination or is conducted away via open current paths to V_{DD} or ground, returning the logic node to its original state. The transient current pulse, called Single Event Transient (SET), is the upset event that can or not generates a fault in the circuit (KASTENSMIDT, 2007).

Fault tolerance on semiconductor devices has been meaningful since upsets were first experienced in space applications several years ago. Since then, the interest in studying fault tolerant techniques in order to keep integrated circuits (ICs) operational in such hostile environment has increased, driven by all possible applications of radiation tolerant circuits, such as space missions, satellites, high-energy physics experiments and others. Spacecraft systems include a large variety of analog and digital components that are potentially sensitive to radiation and therefore fault tolerant techniques must be used to ensure reliability. However, the continuous evolution of the fabrication technology of semiconductor components, in terms of transistor geometry shrinking, power supply, speed, and logic density, the fault tolerance starts to be a matter of concern for circuits operating at ground level as well (KASTENSMIDT, 2007).

As redundancy is the way to achieve fault tolerance, naturally, handling errors and compensating failures in system on chips (SoCs) requires additional hardware for detecting and replacing faulty parts, and thus decrease the overall profitability and will very likely make CMOS-based silicon scaling economically less feasible.

To accomplish a small hardware overhead a low-cost reliability methodology was proposed on the Adaptive Reliability for Embedded Systems (ARES) project of the Eberhard Karls University of Tübingen. The focus of the project is to devise methods to develop and implement a multi-functional, self-adaptive coarse-grained reconfigurable core as a reliability enhancer. In addition, the author of this document was part of the project in the year of 2012 and the work here presented is based in the activities that were done to the project.

The small hardware overhead was reached making instead of every component of a SoC having its own set of redundant hardware, only a multi-functional coarse grained reconfigurable architecture (CGRA) was hardened (MOTOMURA, 2002; DE SUTTER; RAGHAVAN; LAMBRECHTS, 2013; HARTENSTEIN, 2001). Then, in a second step this hardened CGRA was used to improve the overall system reliability by applying dynamic functional verification (KüHN et al., 2012) on the SoC.

Hardening of the CGRA itself is achieved by means of a low-cost triple modular redundancy (TMR) method (SCHWEIZER et al., 2012) in combination with remapping strategies (EISENHARDT et al., 2011), which move (remap) functionality of defective processing elements (PEs) to unused, non-defective PEs and thereby allowing graceful degradation. Defective PEs are identified by a flexible error handling module (FEHM). This module allows to detect and to mask errors that were generated by permanent, and transient faults.

Transient faults that occurs in sequential logics, called Single Event Upset (SEU), causes a bitflip in the output value of the logic. If the sequential logic is not being used in the current clock cycle operation, the bitflip will not influence in the current computation and will remain in the sequential logic until it be updated by a new data, or be detected by the TMR at some clock cycles after it occurrence. In the last case the error was caused by a transient fault that became pertinent in the circuit. Nevertheless, if the sequential logic is being used in the current clock cycle computation, when the bitflip has occurred, the transition that results of the bitflip can generate a wrong computation value when it is stored in the next sequential circuit. It can happens in time enough to not be detected and masked by the TMR. In addition, transient faults that occurs in combinational logics can generate an error whether the transient pulse is stored in a memory cell or a latch. It is a problem because it also can happens in time enough to be detected by the TMR. Furthermore, it can occurs in the signals that compose the TMR logic (that is a combinational logic).

By this reason, the focus of this work is to implement the component of the FEHM that will be able to detect when an error comes from a transient fault that was generated in the CGRA. To achieve this purpose, it was chosen to use the circuit level time-redundancy technique to implement the component, because it results in a sensor that can inform whether an error was generated by a transient fault. In addition, the conditions to implement the chosen technique to maximize the detection capability of the sensor and avoid false errors signals are in this work formulated.

Therefore, this document is organized as follow: Chapter 2 explain the multi-functional CGRA architecture, the FEHM components, the transient faults effects, and the techniques to detected transient errors (caused by transient faults) in digital systems; Chapter 3 has the proposed work to implement the component that will be able to detect when a error was caused by a transient fault in a specific configuration of the CRGA; Chapter 4 presents the component implementation in a hardware description language and its validation in the electrical level; Chapter 5 shows the error diagnostic capability of the FEHM with the implemented component in a fault injection in a timing simulation on the logic level; Finally, the conclusions are drawn in chapter 6.

2 CONTEXTUALIZATION

2.1 Configurable Reconfigurable Core

On the ARES project was developed a modified architecture model that is a coarse-grained reconfigurable architecture. This architecture is the Configurable Reconfigurable Core (CRC) that has the focus of use a fast reconfiguration to optimize area, performance and power. It also represents a range of processor-like architectures (OPPOLD et al., 2007).

The CRC consists of a Processor Element (PE) rectangular array connected through a reconfigurable network(Figure 2.1).

Each PE, Figure 2.2, is composed of a Functional Unit (FU), which executes arithmetic and logical operations, a registers set, a context memory, which holds all the configuration contexts, and a finite state machine (FSM) that changes the context every clock cycle. A PE context determines the FU operation, the FU operand sources (ports or registers), the FU result destination (ports or registers), and the route through input and output ports for the CRC.

The context memory and the FSM must be configured using an external source. Hence, each PE has a module that is represented in Figure 2.2 as *external reconfiguration.*

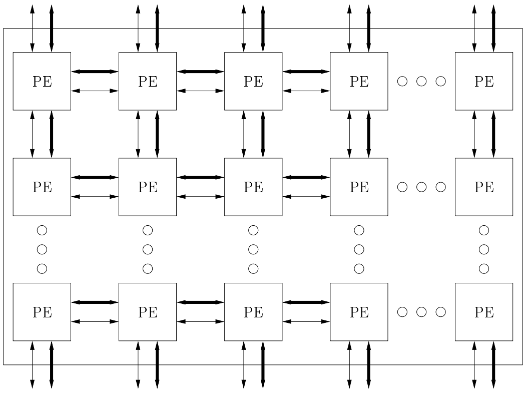


Figure 2.1: PE array.

The CRC, as it is presented in Figure 2.1, has no capacity to detect or mask errors, so, a modification in the architecture design was proposed to increase the CRC reliability. This modification is the insertion of a new module in the architecture that is able to detect and mask transient errors that occur in the PE. The Flexible Error Handling Module is presented in the next section.

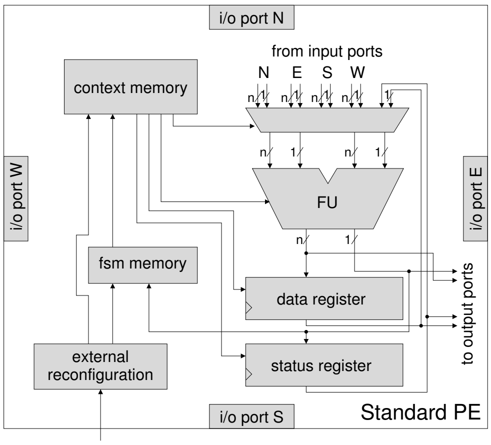


Figure 2.2: PE components.

2.2 Flexible Error Handling Module

Flexible Error Handling Module (FEHM) (SCHWEIZER et al., 2012) is based on Triple Modular Redundancy (TMR) to detect and mask errors. It monitors three input channels, and each input channel has an enable signal. Consequently, if one channel is disabled, the FEHM will work as a Duplication With Comparison (DWC). However, in the DWC operation mode, only error detection is possible.

As it is shown in Figure 2.3, the FEHM provides three outputs: 3 data error signals, 3 transition error signals, and 1 output channel, where the data is presumably right when the module is in TMR mode.

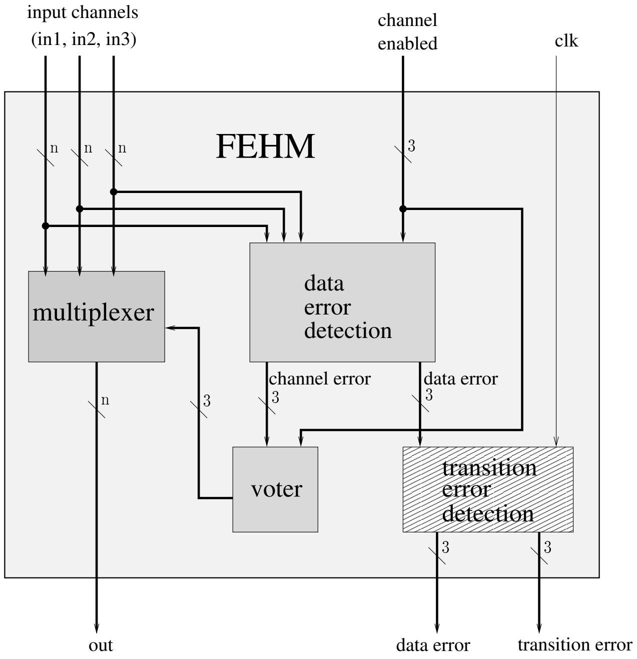


Figure 2.3: FEHM components.

The FEHM uses an integrated voting mechanism to mask the data error by forwarding a correct channel through the *out*.

The data error signals are provided by the Data Error Detection Component (DEDC). It compares each pair of input channel of the module. Inside the pairs, the input channels are compared to each other with a bitwise XOR operation. This operation is reduced to one bit per channel with a XOR tree. This one bit per channel (channel error) indicates if one channel has a different value among the others.

The Transition Error Detection Component (TEDC) monitors the data error signals (Figure 2.3) provided by DEDC, and signalizes when a error caused by a fault in the combinational logic formed by the path from the input of the FUs and the output of the DEDC occurs.

2.2.1 FEHM inclusion

The inclusion of the FEHM in the architecture is made using two possible approaches: TripleFU or FEHM-Cluster. TripleFU, Figure 2.4, implements traditional TMR, since two more FUs are inserted inside a PE and each FU output is directed to the FEHM inputs. This approach considerably increases the PE area. FEHM-Cluster, Figure 2.5, monitors the FU of three different PEs and the FEHM is embedded in one of the PEs(Center-PE). The Side-PEs have a dedicated route to carry the result of theirs FU straight to FEHM (SCHWEIZER et al., 2012).

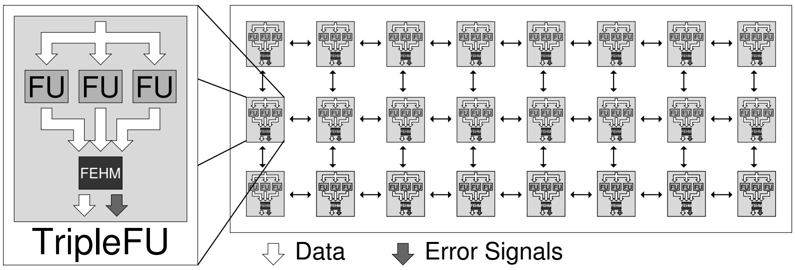


Figure 2.4: TripleFU approach.

2.3 Transient Errors

Transient errors are also known as soft-errors and are caused by transient faults. Soft-errors are errors that do not cause any permanent damage to the devices, just some perturbations in the hardware such as transient voltage pulses and bitflips. These perturbations can flip some logic value stored in a memory element in the system.

Soft-erros are caused by radiation. Neutron particles that are generated by cosmic radiation interact with the Earth atmosphere and can cause some perturbations in digital systems. Alpha particles emitted by radioactive impurities present in low concentration in chip package materials are another source of soft-errors.

In (NICOLAIDIS, 2011), when a particle strike causes a bit-flip (upset) in a memory cell or a latch, we consider that a Single-bit upset (SBU) has occurred. However, when two or more memory cells or latches suffer a bit-flip, it is a Multiple-cell upset (MCU). Whether the event causes the upset of two or more bits in the same word, we have a Multiple-bit upset (MBU). A Single-event transient (SET) occurs whether the strike of the particle in the combinational logic causes a voltage

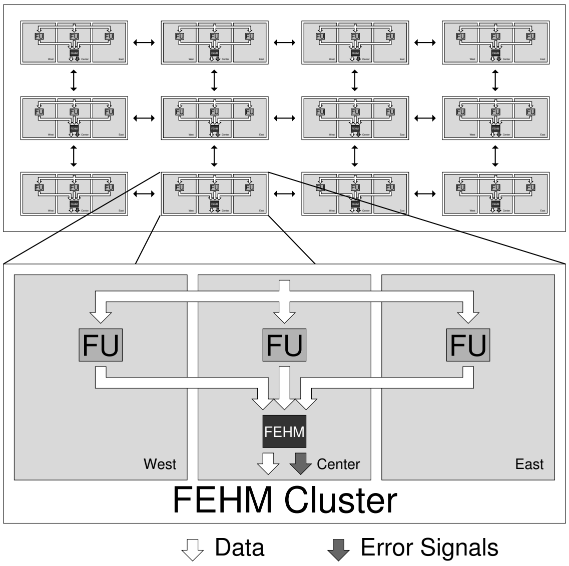


Figure 2.5: FEHM-Cluster approach.

glitch in the circuit, and it becomes a bit error when captured in a storage element. By (KASTENSMIDT, 2007), the voltage glitch might not be stored in a storage element since logic, electrical and window-latch masking occurs. Logic masking, as it is presented in Figure 2.6.(a), occurs when the transient pulse is barred in some logic gate due to the input values of the gate. For example, if the voltage glitch happens in the input of a NAND gate, and the inputs are both in high level (1), the pulse will not be propagated to the output of the NAND gate. Electrical masking occurs if the pulse is attenuated as it propagates through the logic chain and fades out before it reaches the registered output, as shown in Figure 2.6.(b). The window-latch masking takes place when the the SET effect happens out of the latch period of a storage element (depends of the setup and hold time to a memory element).

The perturbation of control registers, clock signals, and reset signals that causes loss of functionality is called Single-event functional interrupt (SEFI). But, when the event creates an abnormal high-current state by triggering a parasitic dual bipolar circuit, which requires a power reset, it is a Single-event latchup (SEL). This last one can possibly cause permanent damage to the device, in which case the result is a hard error (NICOLAIDIS, 2011).

In the literature we can find the term SEU (Single-event upset) as a references to soft-error, but this term could mean a reference to SBU and MBU together (NICOLAIDIS, 2011). On this work, SEU has the last meaning.

2.4 Transient error detection techniques

Error Correction Codes (ECC), Redundancy in time and space, Error detection with logic synthesis and Logic implications are some techniques to achieve transient errors (or soft-errors) mitigation that are applied in electrical or logic circuits (ALVES, 2011). There is also the Hardened Storage cells (SRAM cells, latches, flip-flops) that will preserve their state even if the state of one of their nodes is altered by an ionizing particle strike. Therefore, it is a good way to construct a cell to prevent SEU (NICOLAIDIS, 2010).

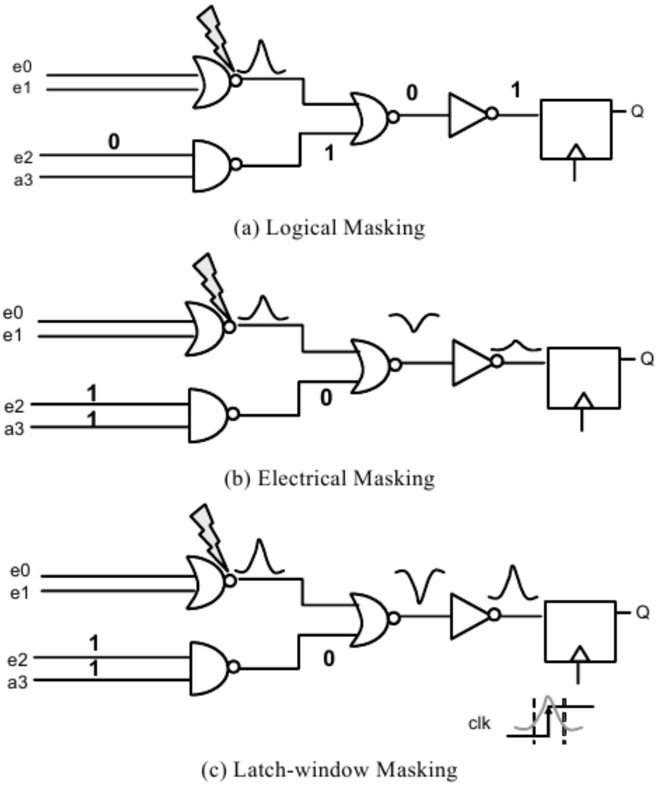


Figure 2.6: Single Event Transient (SET) in a combinational circuit (KASTENSMIDT, 2007).

Error Correction Codes are good to detect transient errors in memory elements and transmission lines. They can indicate if the data that has been read is the same data that was stored in the memory element by using some parity recover. They can also indicate if the stored data was altered by a particle strike. ECC can be extended to other logic structures with easily predictable outputs, such as multipliers, adders, and programmable logic arrays (PLAs). However, detecting errors in apparently random logic, without any discernible logic regularity, is a significantly harder problem. In random logic we must determine if the logic operations performed in some circuit inputs are correct (ALVES, 2011).

Error detection with logic synthesis is a synthesis tweaking applied in order to minimize the number of potential transient errors, via error masking. Features of logic gates are used to fix the error as the signal propagates downstream. For example, a single alpha particle striking any input line of a two-input logic OR gate will have no effect at that gate output when both inputs are logic ones. It is a effect that happens naturally in a circuit run time, and it is knowing as *logic masking.*

The Logic implications is a new method to detect transient errors. This method takes an existing design and searches all internal circuit nodes for consistent logic patterns among them. When an invariant pattern is found, it will append some simple checker hardware that reinforce the validity of the relationship (ALVES, 2011) .

The techniques such as Error detection with logic synthesis and ECC do not achieves the TEDC implementation requirements, since it should operate like a sensor that monitors the output of the DEDC component. Logic Implications could be the chosen technique, but, it will probably change the components which are already implemented.

Redundancy in space, to monitors the desired signal, implies the duplication of the DEDC component. In this case, the TEDC component would be a comparator. However, it does not fit in the FEHM design, since there must exist just one DEDC per FEHM.

Redundancy in time, as it is explained in (ALVES, 2011), is the re-execution of the same logic multiple times, while storing any intermediate data in memory, and reporting any output differences throughout the various executions. However, its multiple logic re-execution significantly decreases the throughput of the circuit.

The concept of circuit level time-redundancy (NICOLAIDIS, 2010) is interesting to the TEDC implementation. The data is checked, at least, twice at each clock cycle and the result of each checked data is compared to each other. If the value is the same in all the checking moments, it is assumed as correct. This technique differs from the redundancy in time concept, since there is no logic re-execution to detect the error, just the logic circuit output is read more than once in the same execution with a period of picoseconds between each read time. This technique is a good choice to implement the TEDC component, since it operate as a sensor and it do not have to make changes in the logic circuit that has been monitored. The circuit level time-redundancy technique is presented in the next section.

2.5 Circuit level time-redundancy

Circuit level time-redundancy is a design solution to transient errors mitigation. The main idea is that transient faults have a limited duration (just a few hundreds picoseconds - the exactly value depends of circuit features and particle energy- (NICOLAIDIS, 1999)) then apply fine time-grain redundancy(i.e. within the clock cycle). Low hardware cost can be achieved, because one does not need to construct the same logic more than once.

The implementation of this technique results in a transient error sensor. Generally, this sensor is put in place of temporal barriers (pipe-lines) as it is shown in Figure 2.7. The sensor can also be put in the end of critical paths or in parallel with some elements to monitor the data time.

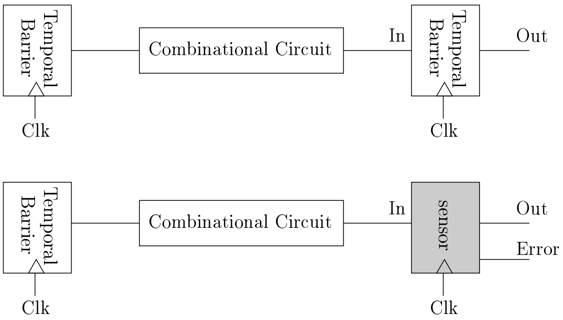


Figure 2.7: Sensor in place of a temporal barrier.

Basically, to detect the transient error, the same data is checked, at least, in two different moments. If the data value is the same in both checking moments, it is assumed that data value as correct. For example, in Figure 2.8, T_1 and T_2 are the checking moments of the data value and the situation which any error occurs (Figure 2.8.(a)), the input signal *In* has the same value in T_1 and T_2. However, in Figure 2.8.(b), the input signal *In* was modified by a voltage glitch and it switches the *Error* signal from low to high in T_2. In this last one situation, the *Error* signal still on the high level until the next T_2 moment, where the *In* signal has the same value in both moments (T_1 and T_2).

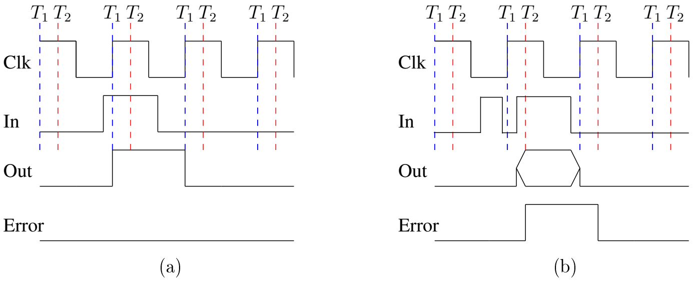


Figure 2.8: (a) conceptual diagram without transient fault; (b) conceptual diagram with transient fault.

A comprehensive way that shows the design of these sensors is presented in Figure 2.9. In these block diagrams, the *Memory Element* and the *Redundant Memory Element* could be as a flip-flop as a latch, the *Comparator block* could be a XOR gate, and the delay elements δ_{In} and δ_{Clk} could be implemented with buffers.

In Figure 2.9.a), the δ_{Clk} is inserted in the *Redundant Memory Element* clock line to check if the data arrived respecting the *Memory Element* latch conditions. As it was mentioned, the signal $\delta_{Clk} + Clk$ could be constructed using a buffer element in the *Clk* line. All the same, one can create a second clock line with a difference δ_{Clk} with the *Clk* line. δ_{Clk} was described in some works (NICOLAIDIS, 1999; ANGHEL; NICOLAIDIS, 2000) using the follows relationships:

$$\begin{cases} \delta_{Clk} = D_{tr} - D_{setup} \\ \delta_{Clk} < D_{min} \end{cases} \tag{2.1}$$

In (2.1), D_{tr} is the maximal transient pulse duration that will be detected when it generates a error. This term is determined by the designer. D_{setup} is the *Redundant Memory Element* setup time and D_{min} is the *Combinational Circuit* minimum path delay. In addition, the relation $\delta_{Clk} < D_{min}$ means that all the path times should exceed the δ_{Clk}. It must be obeyed to avoid false error signals. For example, if both Figure 2.8 waveforms diagrams are from the circuit that is shown in Figure 2.9.a), δ_{Clk} is equal $T_2 - T_1$ and T_2 must be less than the minimum delay of the data path. Therefore, the designer should choose a D_{tr} that makes δ_{Clk} obey the relation (2.1). However, there is another relation to δ_{Clk} that maximizes the transient pulse duration which will be detected without a false error signal generation. This last relation is based on D_{min} and in the *Redundant Memory Element* time to hold (D_{hold}) as it is presented in (2.2).

$$\delta_{Clk} = D_{min} - D_{hold} \tag{2.2}$$

The equation (2.2) is based on the principle that the data which takes the minimum delay will not be save in the *Redundant Memory Element*, since it will come after the time to hold of this element.

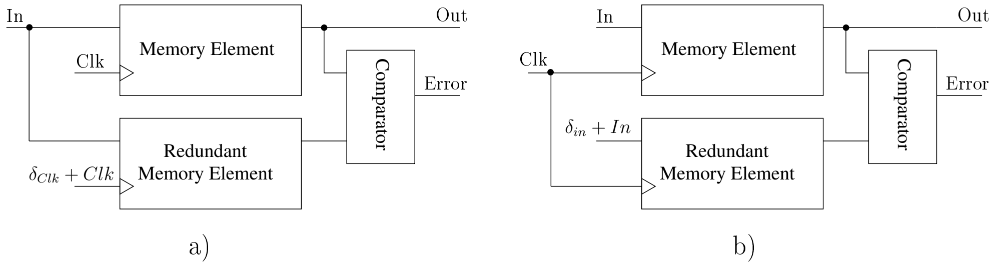


Figure 2.9: Conceptual design of the sensor

In this work, the period that a transient fault can be detected by a sensor will be referenced as T_W. In the case of the Figure 2.8, T_W is equal $T_2 - T_1$ (or equal to δ_{Clk} in the latest example).

The circuit in the Figure 2.9.b) is used to detect when a transient fault causes a violation (an error) in the projected guardband. It can be seen in Figure 2.10.

A guardband δ is defined in Figure 2.10. When the signal *In* comes between t_0 and t_0+T$-\delta$ (T is the clock period) the guardband will not be violated. On the other hand, if the signal *In* arrives between $t_0 + \mathrm{T} - \delta$ and $t_0 + \mathrm{T}$, a violation guardband error will appear.

The guardband in Figure 2.9.b) is defined with the term δ_{In}. The value of δ_{In} must consider the maximum path delay to ensure that false error signals will not occur.

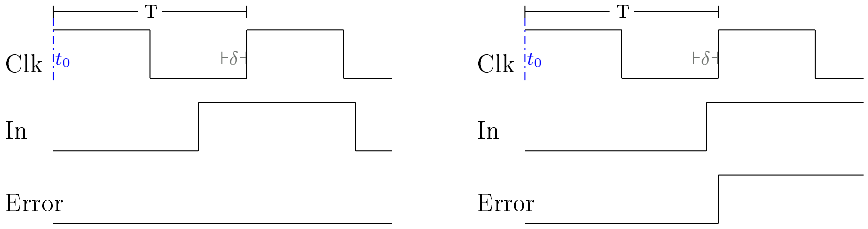


Figure 2.10: Guardband violation example.

Note that all the sensors that were presented here have a delay element δ or in the clock line or in the input line of the sensors. It δ can affect the clock period (T) of the circuit were the sensor is inserted. However, a sensor implementation that uses a latch as *Memory element* and a flip-flop as *Redundant Memory Element* is proposed in (BOWMAN et al., 2009). In a first moment, this sensor implementation does not have a delay element, then, it is a good improvement in the sensors implementation, since it will have a small impact in the performance of the circuit were it is inserted. The design of this sensor is presented in chapter 3.1, where the proposed work is presented.

3 PROPOSED WORK: TRANSIENT FAULTS SENSORS

The objective of the work is to implement the Transient Error Detector Component (TEDC) of the FEHM using the Circuit level time-redundancy technique.

To the execution of the work, two transient faults sensors design that use the concept of circuit level time-redundancy were chosen to be implemented and incorporated into the FEHM.

Therefore, the design and implementations conditions to each sensor is presented in section 3.1. Section 3.2 shows the flow synthesis that is used. Finally, section 3.3 presents the simulations tools that will be used on each simulation level.

3.1 Sensors design, conditions, and description

This section presents both sensors that will be implemented to construct the TEDC. One of them is a classical design solution to transient errors mitigation using circuit level time-redundancy (NICOLAIDIS, 1999). The other is an evolution of the first design (BOWMAN et al., 2009). Thus, the section 3.1.1 presents the classical design and section 3.1.2 presents the other design.

3.1.1 Essential Detector

Figure 3.1 presents one implementation using digital blocks of the Figure 2.9.a) diagram. One can find this implementation in (NICOLAIDIS, 1999).

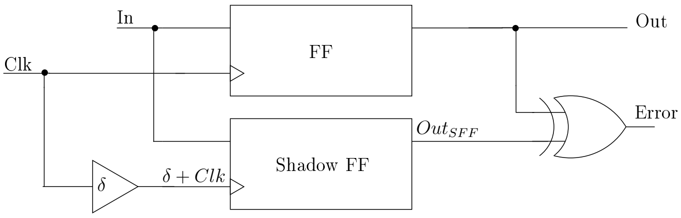


Figure 3.1: Transient fault detector implemented with two flip-flops

The implementation in Figure 3.1 uses two flip-flops and will be called in this work as Essential Detector (ED). The *FF* and *Shadow FF* are the *Memory Element* and the *Redundant Memory Element* respectively. One buffer (δ) is used to construct the element δ_{Clk}. A demonstration about the internal signals of the circuit is presented in Figure 3.1. It is supposed that both flip-flops are sensitive at the rise edge of the clock.

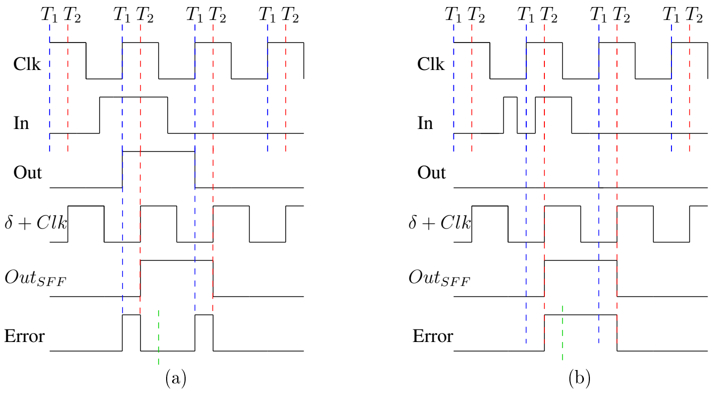


Figure 3.2: Internal signals to Figure 3.1

The delay element δ causes the occurrence of the *Shadow FF* latch edge in T_2. In the situation where any transient pulse occurs (Figure 3.2.(a)), the data is correctly saved inside the two flip-flops. It is seen in the waveforms of the signals *Out* and Out_{SFF}. However, in the another situation (Figure 3.2.(b)), the flip-flop *FF* could not save the correct value of the signal *In*, since a SET has occurred.

The delay δ that is generated by the buffer must be enough to Shadow FF not save the data when it come, in the current clock cycle, from D_{min} (the minimal path delay) until the next clock latch edge.

The data will not be save in a flip-flop if it comes after the flip-flop time to hold (T_{h_f}). Therefore, the delay δ should considerate D_{min} and T_{h_f} times. Consequently, to this implementation:

$$\delta = D_{min} - T_{h_f} \tag{3.1}$$

Therefore, this sensor will be able to indicate all the errors that are caused by transient pulses with durations less than D_{min}. Thus, the T_W to the ED sensor is equal to D_{min} ($T_W = D_{min}$).

3.1.1.1 Behavioral verilog description

The verilog description that was used for the ED is presented in the Verilog code block 3.1. The signals of the verilog description are mapped in the ED design as it is presented in Figure 3.3.This implementation uses a buffer in the clock line, but, the verilog code has an inversor in the place of the buffer. The explanation to this is that the synthesis tool optimize the circuit. So, with the optimization, the buffer was replaced by a wire connection. Therefore, the solution to this case was to put an inversor with a specific name (like “clk_buf”) and , after the synthesis process, manually replace the inversor by the required buffer.

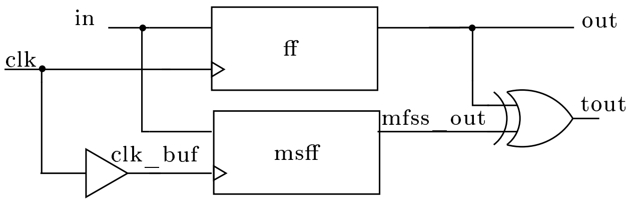


Figure 3.3: Verilog mapping in ED design.

Verilog code 3.1: ED verilog description

```
//ED
module tedm (out, tout, in, clk);

  // ports
  output out, tout;
  input in, clk;

  // internal wires
  wire ff_out, msff_out;

  // connections
  assign eout = ff_out;
  assign tout = msff_out ^ ff_out;

  //ED
  assign clk_buf= ~clk;
  ret_dff_high_edge msff (.q(msff_out),
                 .d(in), .clk(clk_buf)); // shadow ff

  ret_dff_high_edge ff (.q(ff_out), .
                 d(in), .clk(clk)); // ff
endmodule // tedm
```

3.1.2 Double Sampling with Time Borrowing

Double sampling with time borrowing (DSTB), Figure 3.4, is a solution that eliminates the metastability in data path. Now, only in the error path is possible the metastability occurrence. By (BOWMAN et al., 2009), this is a drastic simplification on the sensor metastability handle.

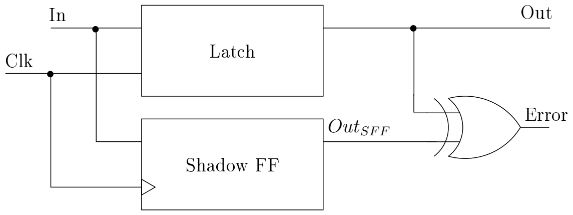


Figure 3.4: DSTB design.

The sensor uses one flip-flop (Shadow FF) to double sample the data. The *Memory Element* is a latch. It is supposed that the *Shadow FF* is sensitive to the rise edge of the clock and the latch is high level sensitive. As the latch is high level sensitive, then its T_W is all the high level of the clock signal (the duty cycle of the clock). This fact is shown in Figure 3.5.

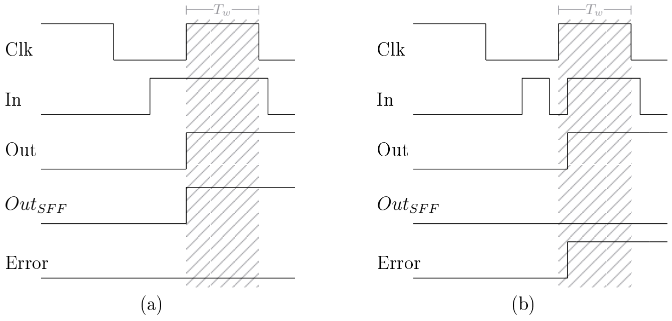


Figure 3.5: DSTB operation

To not generate false errors signals, all the that of the current clock cycle must to be out of the T_W period, because it is the evaluation time of the data that comes from the previous clock cycle. Therefore, the data transition that take the minimum path delay must exceeds the duty-cycle period of the clock signal, because it is the T_W of the DSTB. Then, we have the follow relations:

$$T_W = Clk_{duty-cycle} \tag{3.2}$$

$$T_W < D_{min} \tag{3.3}$$

In (3.3) the term D_{min} is the minimum path delay of the combinational circuit. The relation (3.3) must be respected to avoid false error signals. If the delay of the minimum path is to small that it is impossible to reduce the duty-cycle of the clock signal, buffers can be inserted in the data path to push the occurrence of the data to after the high level of the clock (*In* node in Figure 3.4) as it is shown in Figure 3.6. Therefore, the delay δ that the buffer might generate is based in D_{min}, the clock duty cycle, and the latch time to hold (T_{h_l}). Nevertheless, this will affect the clock period, since the delay of the maximum path (D_{max}) could exceed the clock period and it could cause a false error signal.

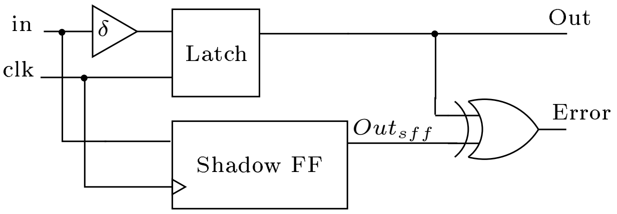


Figure 3.6: DSTB circuit with a extra delay element.

Figure 3.7 has an illustration of the situation where the data with minimum delay comes inside of DSTB T_W period. The δ value might put all the valid data out of the duty cycle period in the current clock cycle, since that data will be evaluated in the next clock cycle. The valid data are which come between D_{min} and D_{max}. All data that come between the high clock edge and D_{min} might be take as error generated by a transient fault, consequently, the T_W is theoretically the period between the positive edge of the clock and D_{min}. D_{valid} represents the valid data period in Figure 3.7. The latch must not save the valid data of the current clock cycle, since the value which was saved in the Shadow Flip-Flop is the value result from the previously clock cycle. Both elements (Latch and Flip-Flop) must save the values of the previously execution cycle. Therefore, that is the reason which the T_{h_l} is necessary to define δ.

Looking at Figure 3.7, we can see that δ, to the DSTB sensor, is defined as:

$$\delta = C_{duty} - D_{min} + T_{h_l} \tag{3.4}$$

C_{duty} in the project parameters is:

$$C_{duty} = \rho \ * \ \mathrm{T},\ 0 < \rho < 1 \tag{3.5}$$

Figure 3.8 shows all the intervals which the clock period T must cover.

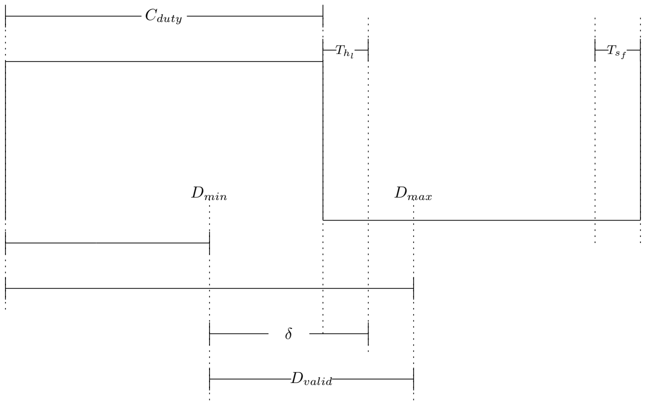


Figure 3.7: Conceptual waveform with D_{min} inside T_W

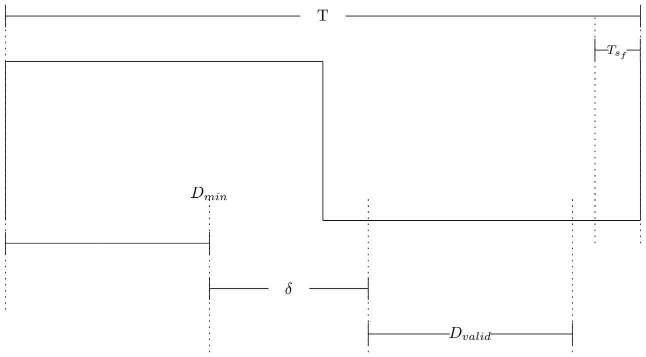


Figure 3.8: Conceptual waveform with all the intervals that clock period must cover.

The T definition needs to take the flip-flop setup time (T_{s_f}) because all valid data should be saved inside the Shadow flip-flop. Consequently, T should respect the follow condition:

$$\mathrm{T} \geq D_{min} + \delta + D_{valid} + T_{s_f} \tag{3.6}$$

D_{valid} in the project parameters is:

$$D_{valid} = D_{max} - D_{min} \tag{3.7}$$

Using (3.4) and (3.5) we can define δ with the project parameters as:

$$\delta = \rho \, * \, \mathrm{T} - D_{min} + T_{h_l} \tag{3.8}$$

Consequently, using (3.6), (3.7), and (3.8) we can define the condition to T as:

$$\mathrm{T} \geq D_{min} + \rho \, * \, \mathrm{T} - D_{min} + T_{h_l} + D_{max} - D_{min} + T_{s_f}$$

$$\mathrm{T} \geq \rho \, * \, \mathrm{T} + D_{max} - D_{min} + T_{s_f} + T_{h_l} \tag{3.9}$$

Isolating T in the left side of the inequality, we get:

$$\mathrm{T} \geq \frac{D_{max} - D_{min} + T_{s_f} + T_{h_l}}{1 - \rho} \tag{3.10}$$

Therefore, when a buffer is inserted in DSTB Latch data line the follows relations should be respected:

$$\begin{cases} \mathrm{T} \geq \frac{D_{max}-D_{min}+T_{s_f}+T_{h_l}}{1-\rho},\ 0<\rho<1 \\ \delta = \rho\ *\ \mathrm{T} - D_{min} + T_{h_l},\ 0<\rho<1 \end{cases} \tag{3.11}$$

The implementation of the DSTB using the conditions (3.11) decrease the T_W period to be equal to D_{min}. However, it will ensure that false error signals will not be generated.

3.1.2.1 Behavioral verilog description

The verilog description to the DSTB that was used is presented in the Verilog code block 3.2. The signals of the verilog description are mapped in the DSTB design as it is presented in Figure 3.9. One inversor ("in_buf") was inserted in the input line, and after the synthesis it was manually replaced by the required buffer.

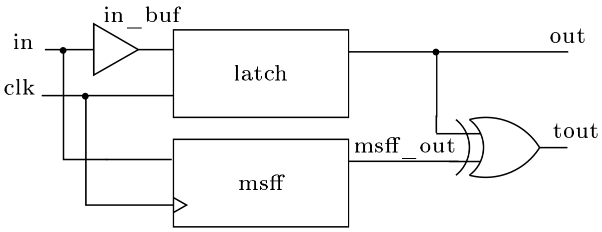


Figure 3.9: Verilog mapping in DSTB design.

Verilog code 3.2: DSTB verilog description

```
//DSTB
module tedm (out, tout, in, clk);

  // ports
  output out, tout;
  input in, clk;

  // internal wires
  wire latch_out, msff_out;

  // connections
  assign out = latch_out;
  assign tout = msff_out ^ latch_out;

  //DSTB
  assign in_buf = ~in;
  ret_dff_high_edge msff (.q(msff_out),
                    .d(in_buf), .clk(clk)); // shadow ff
  d_latch latch (.q(latch_out),
                    .d(in), .clk(clk)); // latch

```

```
endmodule // tedm
```

3.1.3 Considerations

By the conditions that were presented in sections 3.1.1 and 3.1.2 to the design of the sensors that were chosen to be implemented in this work, it is very important to obtain the D_{min} and D_{max}, because if the δ value is arbitrarily chosen, the valid data transitions could be interpreted as errors that were generated by a transient fault. Whether both sensors were projected using the relations to δ that were defined to each one, then errors generated by transient faults of duration less than D_{min} will be always indicated by the sensors.

As buffers could be inserted in the input line of the DSTB design, it could influence in the performance of the whole circuit, since it will deslocate all the valid data. Thus, the minimum period of the clock that the circuit operates without the sensor may be increase when DSTB is inserted in the circuit.

3.2 Flow synthesis

The whole architecture was described in a behavioral verilog and synthesized with the Design Compiler tool of Synopsys to a gate description verilog. This process used the Nangate FreePDK45 Generic Open Cell Library of Si2 (SI2, 2012) (Silicon Integration Initiative) to make the technological mapping. Besides of the gate description verilog, a Standard Delay Format (SDF) file is also generated by the design compiler. With these files, it is possible to perform a timing simulation in the logic level.

From the gate description verilog was generated a spice net-list using the Calibre tool (a Verification-tool of Mentor Graphics). The synthesis process is represented in a graphic way in Figure 3.2.

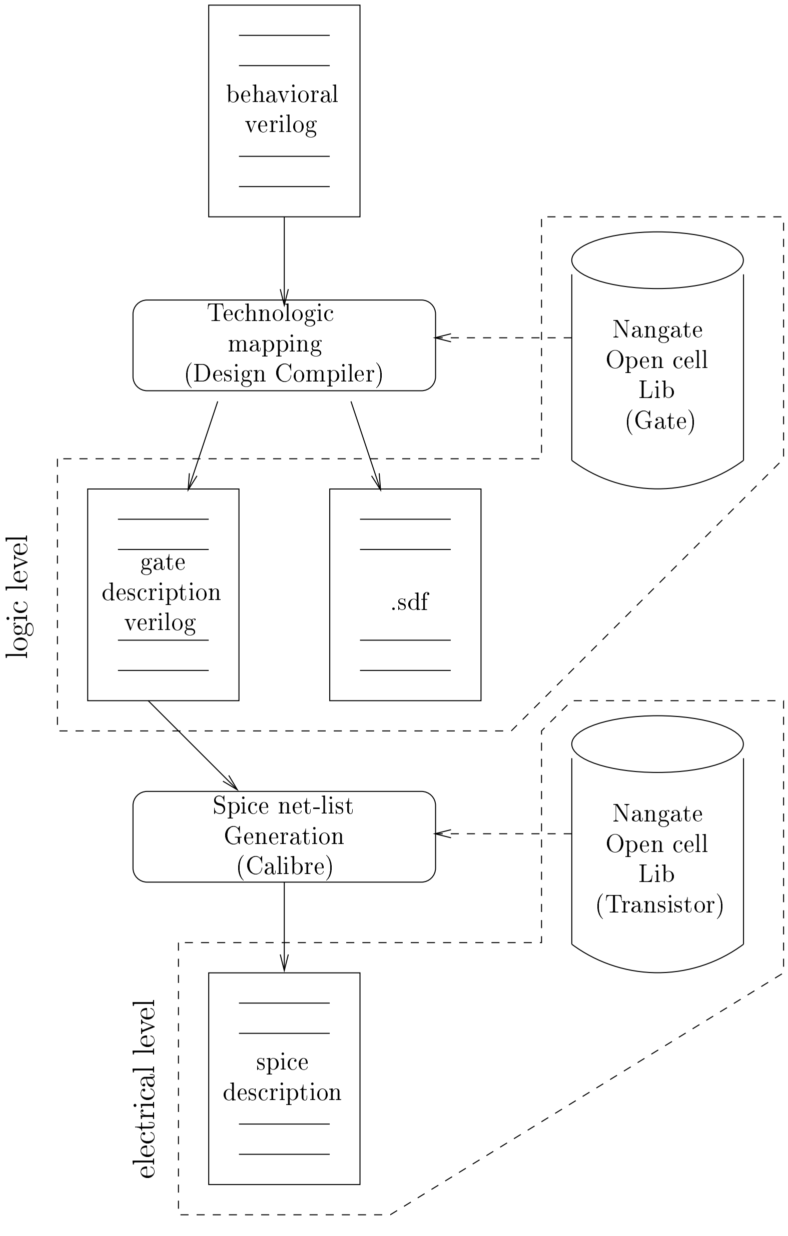

behavioral verilog
Technologic mapping (Design Compiler)
Nangate Open cell Lib (Gate)
logic level
gate description verilog
.sdf
Spice net-list Generation (Calibre)
Nangate Open cell Lib (Transistor)
electrical level
spice description

Figure 3.10: Flow synthesis

3.3 Simulation tools

The sensors implementation process were firstly validated with a spice simulation, because it is a more complete simulation process since it is in an electrical level. Capacitance and resistance are cared about on this level of simulation. So, the NanoSim tool (SYNOPSYS, 2010) was used to this purpose.

Another thing that was done on the electrical level was the delays scenarios determination. It used the NanoTime (SYNOPSYS, 2009) analyzer tool to discover some path delays.

The transient fault detection capability was simulated in the logic level using the Modelsim tool (GRAPHICS, 2002). It was done on this level, because it take a long time in the electrical level. In addition, it is more simple to make fault injection simulations using the Modelsim SE.

4 SENSORS IMPLEMENTATION PROCESS VALIDATION

On this chapter, the implementation process of the sensors with all the conditions that were drawn in chapter 3 is validated. To the sensors implementation, it is necessary to know the values of the maximum and minimum path delays where the sensors aim to monitor. The process was validate taken the parameters to the FEHM-Cluster approach.

By this reason, section 4.1 presents the path that will be monitored in the FEHM-Cluster by the sensors. Section 4.2 aim to explain the way that the scenarios to obtain D_{max} and D_{min} to the target path where determined. Section 4.3 shows the calculus to D_{max} and D_{min} using the scenarios that were discovered in section 4.2. Section 4.4 presents the sensors construction based in the conditions that were drawn in chapter 3 to an false error signal free implementation. Finally, section 4.5 has the synthesis results to a ASIC using a open cell library of 45 nm technology.

4.1 Monitored path

A set of three instances of the same sensor type is placed in the TEDC as in Figure 4.1. The verilog description of the sensors in chapter 3 shows that both sensors were described with the same signature (tedm). Therefore, the TEDC verilog description is presented in the Verilog code block 4.3.

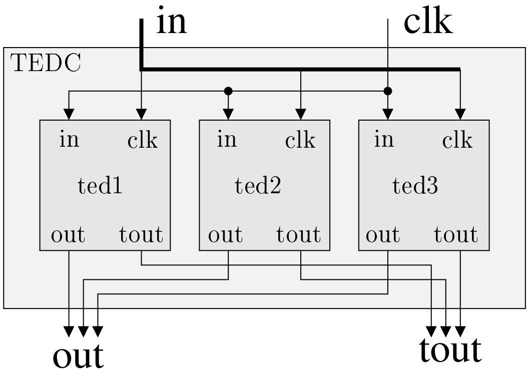


Figure 4.1: TEDC modules and signals.

Verilog code 4.3: TEDC verilog description

```
module tedc (out, tout, in, clk);
  // ports
  output [2:0] out, tout;
  input clk;
  input [2:0] in;
  tedm ted1 (.out(out[0]), .tout(tout[0]),
                                .in(in[0]), .clk(clk));
  tedm ted2 (.out(out[1]), .tout(tout[1]),
                                .in(in[1]), .clk(clk));
  tedm ted3 (.out(out[2]), .tout(tout[2]),
                                .in(in[2]), .clk(clk));
endmodule // tedc
```

The TEDC is inserted in the FEHM as it is presented in Figure 4.2. The verilog description of the FEHM is presented in the Verilog code block 4.4.

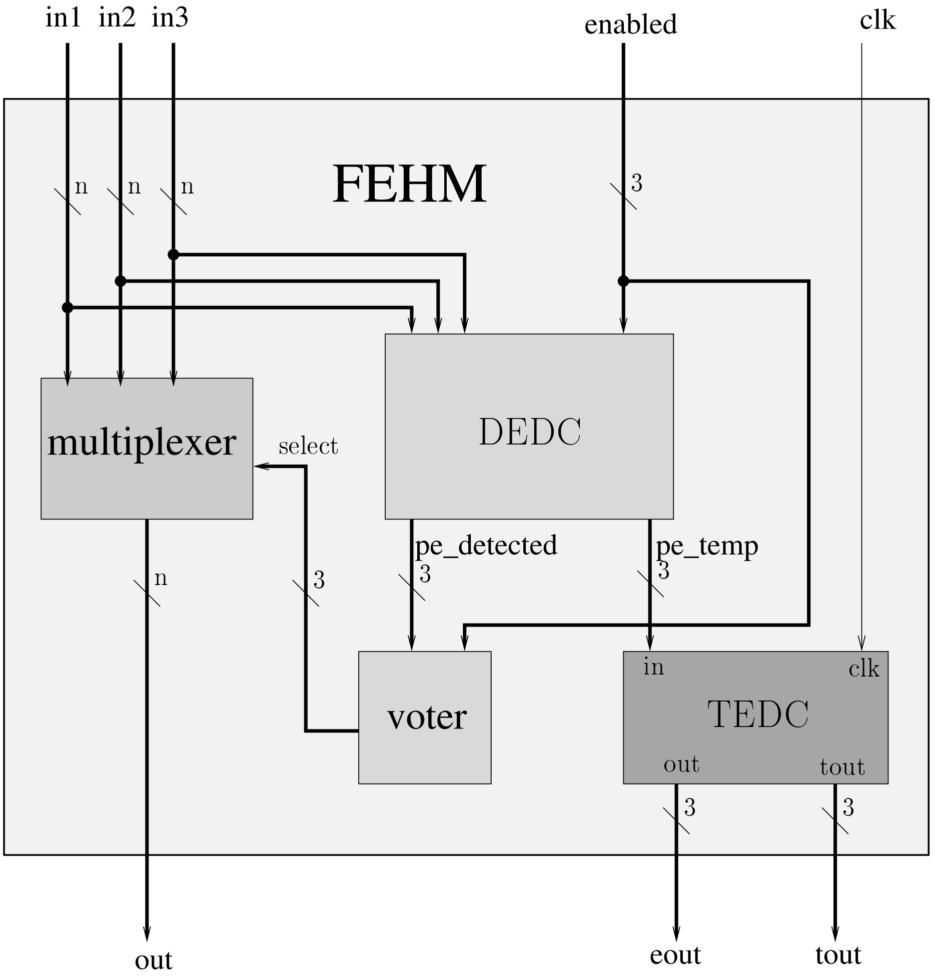


Figure 4.2: FEHM components and signals.

Verilog code 4.4: FEHM verilog description

```verilog
module fehm (out, eout, tout, in1, in2, in3, enabled, clk);
  // MUST be set on instantiation!
  parameter WIDTH = 0;
  // ports
  output [WIDTH-1:0] out;
  output [2:0] eout, tout;
  input [WIDTH-1:0] in1, in2, in3;
  input [2:0] enabled;
  input clk;

  // internal wires
  wire [2:0] pe_detected, pe_temp, select;

  //DEDC
  pedm3 #(.WIDTH(WIDTH))
  dedc (.eout(pe_detected), .in1(in1), .in2(in2),
                                       .in3(in3), .enabled(enabled));
  assign pe_temp = pe_detected & enabled;

  //TEDC
  tedc tedc (.out(eout), .tout(tout),
                         .in(pe_temp), .clk(clk));
  //voter
  assign select[0] = ~pe_detected[0] & enabled[0];

  assign select[1] = &{~pe_detected[1], enabled[1],
                         pe_detected[0] | ~enabled[0]};

  assign select[2] = &{~pe_detected[2], enabled[2],
                         pe_detected[0] | ~enabled[0],
                         pe_detected[1] | ~enabled[1]};
  //multiplexer
  mux3 #(.WIDTH(WIDTH))
  mux (.out(out), .in1(in1), .in2(in2),
                         .in3(in3), .select(select));

endmodule // fehm
```

Looking at Figure 4.2 and Verilog code block 4.4, it is possible to notice that the signal set which the TEDC should to monitors is named as *pe_temp*. Therefore, the delays of the paths that ends on this set of nodes (*pe_temp[0]*, *pe_temp[1]*, and *pe_temp[2]* nodes) should to be measured to the buffer sensors implementation.

In context of the FEHM-Cluster, the monitored paths are shown in red in Figure 4.3. It start in the input of the multiplexers which defines the FU operands source of all the PEs of the cluster and end in the *pe_temp* set of nodes.

Therefore, the maximum and minimum delay to the paths that are shown in Figure 4.3 must be discovered.

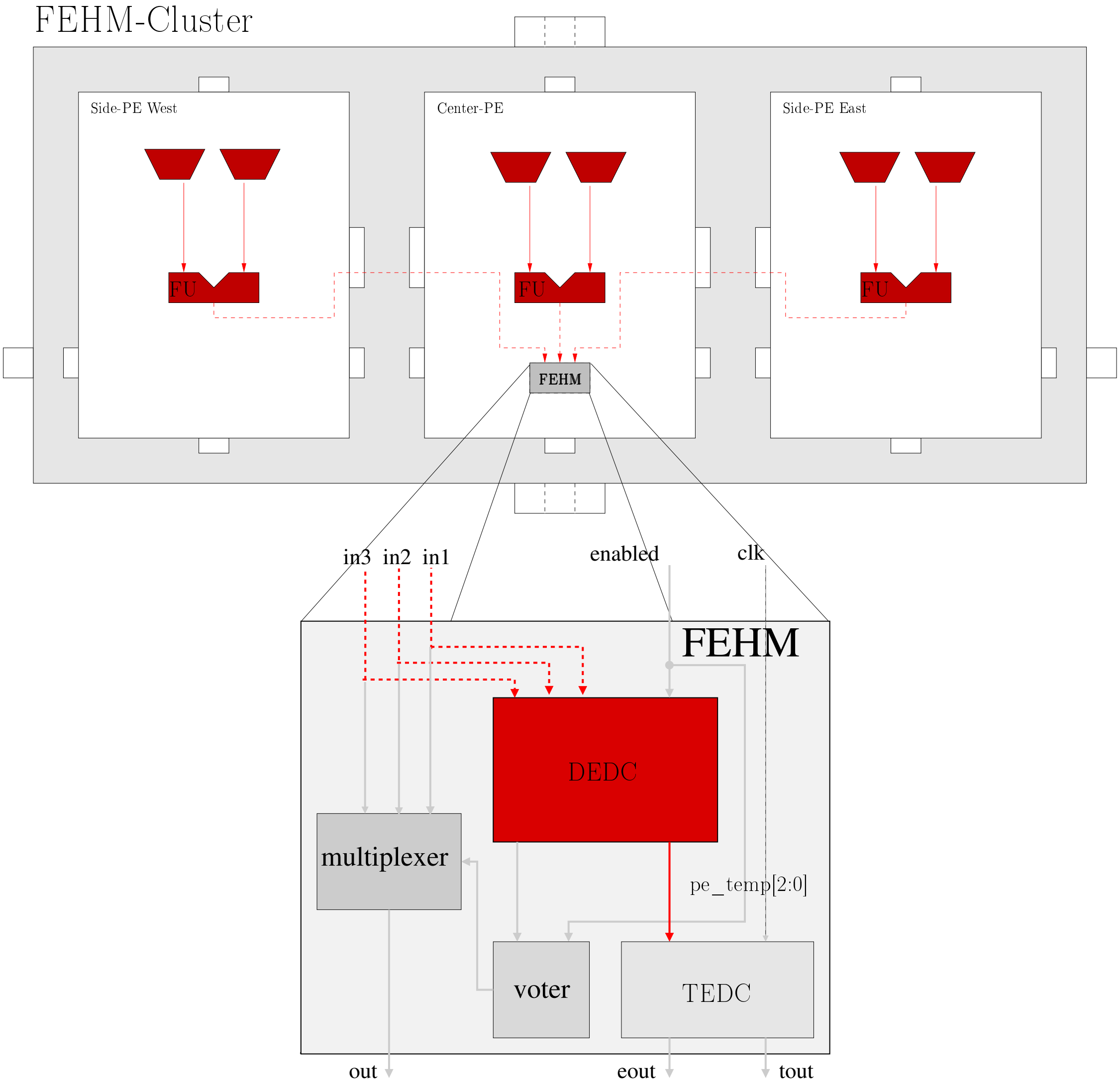


Figure 4.3: Monitored paths.

4.2 Delays scenario

Each simulation level (electrical and logical) has its own maximum and minimum delay time. Therefore, here, the scenario to take both delays values is obtained. It involves the FU operation, the FU input values and the internal route of the chosen approach. Once the scenario was obtained, it is possible to configure the architecture to get the desired data. The sections 4.2.1 and 4.2.2 describe all that was done to determine the scenarios.

The determination of the scenarios was made using the principle that each operation is executed with one clock cycle. With this in mind, the required delay, in this case, is the time between the rise edge of the clock and some change in any node of the set denominated as *pe_temp* (Figure 4.3).

4.2.1 Maximum Delay Scenario

The maximum delay scenario of the FEHM-Cluster was discovered using three steps. They are described below:

First step: Discover what is the slowest FU operation.

Second step: Discover the worst input vectors to the operation found on the early step.

Third step: Discover the configuration that does the longest routing between the FEHM-Cluster input ports and the FU input of each PE inside a cluster.

4.2.1.1 The slowest Functional Unit operation

The information about the slowest FU operation was found in a paper that describes the “Cost functions” used by the architecture compiler (OPPOLD et al., 2004) of the CRC. These functions need the time delay of such paths, digital blocks or FU operations. The values found in this paper are outdated, but the relation between the operation times are enough to get the searched information. The Table 1 of the article (OPPOLD et al., 2004) shows us that the slowest operation is the multiplication.

4.2.1.2 The worst input vectors

The second step goal was reached with the Nanotime tool help. It is a Transistor-level Static Timing Analysis. Basically, we should inform the Nanotime in what Spice file and, inside of this file, what subcircuit we want to analyze. The set of input and output ports should be also informed. With all these informations, the tool can calculate the delay of all possible paths that have the input and output pins in the set informed before. To discover the worst input vectors, only the multiplier of the FU was analyzed. The commands to analyze the timing of the multiplier in the Nanosim are shown in the Nanotime Commands block 4.1.

Nanotime Commands 4.1: Setup commands.

```
set search_path {.}
set library_path {.}
set link_path {*}
set oc_global_voltage 1.2

register_netlist -format spice {pe_cluster_h_fehm.sp \
tech.sp}
link_design fu_1_DW02_mult_0

set input_ports {A[*] B[*]}
set_port_direction -input $input_ports

set output_ports {TC PRODUCT[*]}
set_port_direction -output $output_ports
report_port
```

In the Nanotime Commands block 4.1, the *link_design fu_1_DW02_mult_0* command informs the Nanotime tool that we want analyze only the multiplier of the FU. The *fu_1_DW02_mult_0* is the name of the multiplier instance in the FU. In addition, the name of the multiplier operands are A and B, and name of the multiplier outputs are TC and $PRODUCT$.

The follow methodology was used to get the worst multiplier input vectors. First, all the circuit inputs were let free in a way that the tool could analyze all the possible paths. It is shown in Figure 4.4. The Nanotime commands to trace and report the eight slowest paths are in the Nanotime Commands block 4.2. The result of the first path trace is presented in the Nanotime Result block 4.1.

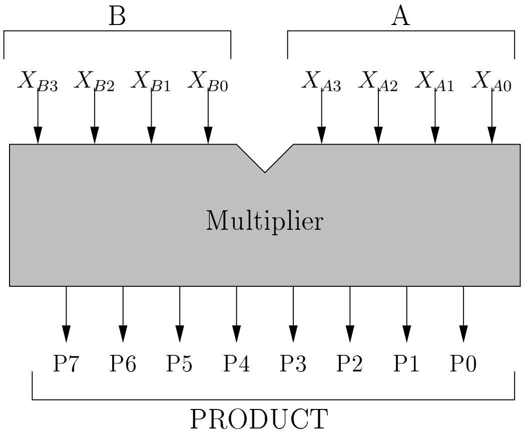


Figure 4.4: All multiplier inputs free

Nanotime Commands 4.2: Trace and report the eight slowest paths

```
trace_paths -full_path_enumeration

report_paths -max -max_paths 8
```

Nanotime Results 4.1: Maximum delay. First path trace.

```
               Path   Path
  Slack       Delay   Type      Startpoint         Endpoint
---------  ---------  ------  --  --------------  --  -----------

  8.297       0.703    D-O  r   B[2]            f   PRODUCT[7]
  8.298       0.702    D-O  r   B[1]            f   PRODUCT[7]
  8.309       0.691    D-O  f   B[1]            f   PRODUCT[7]
  8.309       0.691    D-O  r   A[0]            f   PRODUCT[7]
  8.310       0.690    D-O  r   A[1]            f   PRODUCT[7]
  8.321       0.679    D-O  f   B[2]            f   PRODUCT[7]
  8.331       0.669    D-O  f   B[1]            r   PRODUCT[7]
  8.333       0.667    D-O  f   B[0]            f   PRODUCT[7]
```

The Nanotime Result block 4.1 shows us that the slowest path begins in the pin B[2], ends in the pin PRODUCT[7], and its delay is equal to 0.703 ns. Therefore, these informations ({start point, end point, delay time}) are taken as reference values to the next path traces. The reference values will be represented, formally, like in (4.1).

$$\{\text{B[2], PRODUCT[7], 0.703 ns}\} \tag{4.1}$$

The second path trace was done with the pins B[3], B[1], and B[0] fixed in 1. It tries to define the value of the B operand. With this configuration ($B = \begin{bmatrix}1 & X_{B2} & 1 & 1\end{bmatrix}$ and $A = \begin{bmatrix}X_{A3} & X_{A2} & X_{A1} & X_{A0}\end{bmatrix}$) , whether the result of the path trace is equal to (4.1), then we can assume that the B operand value is:

$$B = \begin{bmatrix}1 & X_{B2} & 1 & 1\end{bmatrix} \tag{4.2}$$

The Nanotime Commands block 4.3 shows the commands to force the value 1 in the pins previous cited and the Nanotime Results block 4.2 has the second path trace results.

Nanotime Commands 4.3: Forcing the values in the B vector

```
set_case_analysis 1 B[0]
set_case_analysis 1 B[1]
set_case_analysis 1 B[3]
```

Nanotime Results 4.2: Second path trace result.

```
                  Path   Path
    Slack        Delay   Type      Startpoint         Endpoint
 ----------  ----------  ------  --  ------------  --  ----------

    8.297        0.703    D-O  r  B[2]          f  PRODUCT[7]
    8.309        0.691    D-O  r  A[0]          f  PRODUCT[7]
    8.310        0.690    D-O  r  A[1]          f  PRODUCT[7]
    8.321        0.679    D-O  f  B[2]          f  PRODUCT[7]
    8.337        0.663    D-O  f  B[2]          r  PRODUCT[7]
    8.338        0.662    D-O  f  A[1]          f  PRODUCT[7]
    8.343        0.657    D-O  r  B[2]          r  PRODUCT[7]
    8.350        0.650    D-O  f  A[0]          f  PRODUCT[7]
```

Nanotime Results block 4.2 confirms that we can assume the value to the B operand as in (4.2) because the slowest path has the same begin, end, and delay time as (4.1).

Now, the A operand should be discovered. The value of the A vector was got in a interactive way. First, the follow configuration, $A = \begin{bmatrix}1 & 1 & 1 & X_{A0}\end{bmatrix}$ and $B = \begin{bmatrix}1 & X_{B2} & 1 & 1\end{bmatrix}$ was used like it is shown in Figure 4.5. The goal is always the same: to verify if the configuration generates the same slowest path that is represented in (4.1). The commands to make the configuration are shown in Nanotime Commands block 4.4 and the result is in Nanotime Results block 4.3.

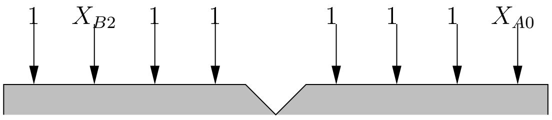


Figure 4.5: A[0] free to test.

Nanotime Commands 4.4: Forcing values in the B and A vector

```
set_case_analysis 1 B[0]
set_case_analysis 1 B[1]
set_case_analysis 1 B[3]

set_case_analysis 1 A[1]
set_case_analysis 1 A[2]
set_case_analysis 1 A[3]
```

Nanotime Results 4.3: First interactive to find the values of the A vector

```
              Path   Path
   Slack     Delay   Type       Startpoint          Endpoint
 --------  -------- ------ --  --------------- --  ------------
   8.307     0.693   D-O  r   B[2]             f   PRODUCT[7]
   8.320     0.680   D-O  r   A[0]             f   PRODUCT[7]
   8.321     0.679   D-O  f   B[2]             f   PRODUCT[7]
   8.350     0.650   D-O  f   A[0]             f   PRODUCT[7]
   8.359     0.641   D-O  f   B[2]             r   PRODUCT[7]
   8.389     0.611   D-O  f   A[0]             r   PRODUCT[7]
   8.389     0.611   D-O  r   B[2]             f   PRODUCT[6]
   8.402     0.598   D-O  r   A[0]             f   PRODUCT[6]
```

The result shows us that with $A = \begin{bmatrix}1 & 1 & 1 & X_{A0}\end{bmatrix}$ the worst path delay is not the same as in (4.1) since it is equal to 0.693 ns. However, the A vector received successive rotates left like it is shown in Figure 4.6. On each rotate, a path trace was done and the slowest path for each case was compared with (4.1). The process stopped when the A operand took the follow configuration: $A = \begin{bmatrix}1 & X_{A2} & 1 & 1\end{bmatrix}$. Nanotime Results block 4.4 shows that the slowest path to $B = \begin{bmatrix}1 & X_{B2} & 1 & 1\end{bmatrix}$ and $A = \begin{bmatrix}1 & X_{A2} & 1 & 1\end{bmatrix}$ is the same as (4.1).

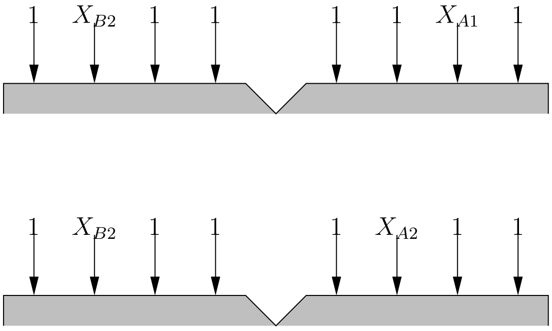


Figure 4.6: Rotates of the A vector

Nanotime Results 4.4: Delays to vector (4.2.1.2)

	Path	Path				
Slack	Delay	Type		Startpoint		Endpoint
8.297	0.703	D–O	r	B[2]	f	PRODUCT[7]
8.328	0.672	D–O	f	B[2]	f	PRODUCT[7]
8.355	0.645	D–O	f	B[2]	r	PRODUCT[7]
8.379	0.621	D–O	r	B[2]	f	PRODUCT[6]
8.392	0.608	D–O	r	B[2]	r	PRODUCT[5]
8.399	0.601	D–O	r	A[2]	f	PRODUCT[7]
8.407	0.593	D–O	r	B[2]	r	PRODUCT[6]
8.410	0.590	D–O	f	B[2]	f	PRODUCT[6]

These path traces show us that the worst input case to the multiplier is the follow:

$$\begin{cases} A = \begin{bmatrix} 1 & X_{A2} & 1 & 1 \end{bmatrix} \\ \\ B = \begin{bmatrix} 1 & R_{B2} & 1 & 1 \end{bmatrix} \end{cases} \tag{4.3}$$

In (4.3) the R_{B2} represents a transition from 0 to 1. A X_{A2} is still used to represents the transition of the pin A[2] since nothing is clear about it. So, the scenario to the maximum delay should consider the two situations to the A[2] pin: a transition from 0 to 1 and a transition from 1 to 0.

4.2.1.3 The longest FEHM-Cluster internal route

The longest internal route is worth to find since it will make the operands arrive a little bit later in the multiplier input ports. The meaning of “longest” is that the operands must pass through the maximum number of elements as possible.

An external sight of the FEHM-Cluster shows us that it is a black-box with four input/output ports. When the cluster is operating in TMR mode, all the three internal PEs should execute, in the same clock cycle, the same operation with the same operands. The Figure 4.7 illustrates the operands going from the Center-PE to the Side-PEs. The route represented in this figure is inherent. Consequently, it is not necessary a especial configuration to trace this path between the Center-PE and the Side-PEs.

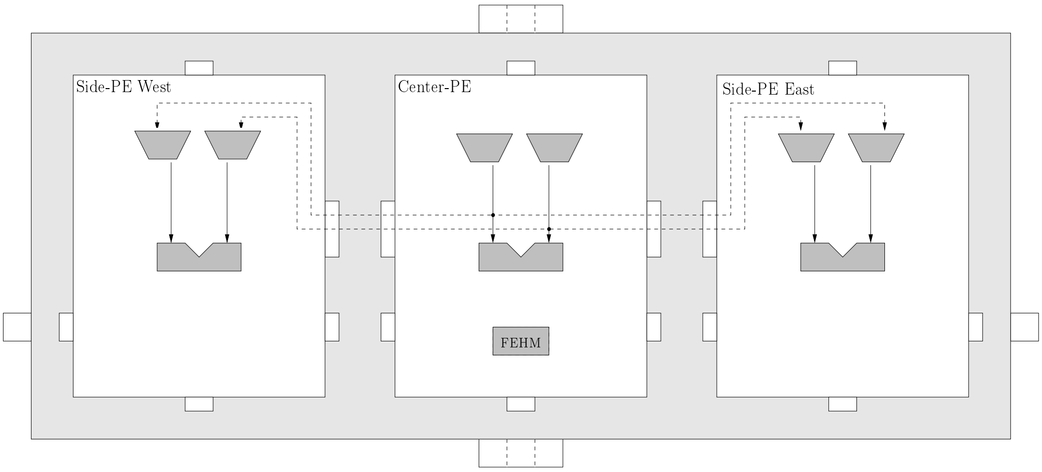


Figure 4.7: Operands from the Center-PE to the Side-PEs.

Different from the North and South FEHM-Cluster ports, the East and West ports do not have a direct net to Center-PE. As a result, the data should pass through the Side-PEs and be routed to the Center-PE when it come from those ports. The Figure 4.8 shows all the connections that were mentioned.

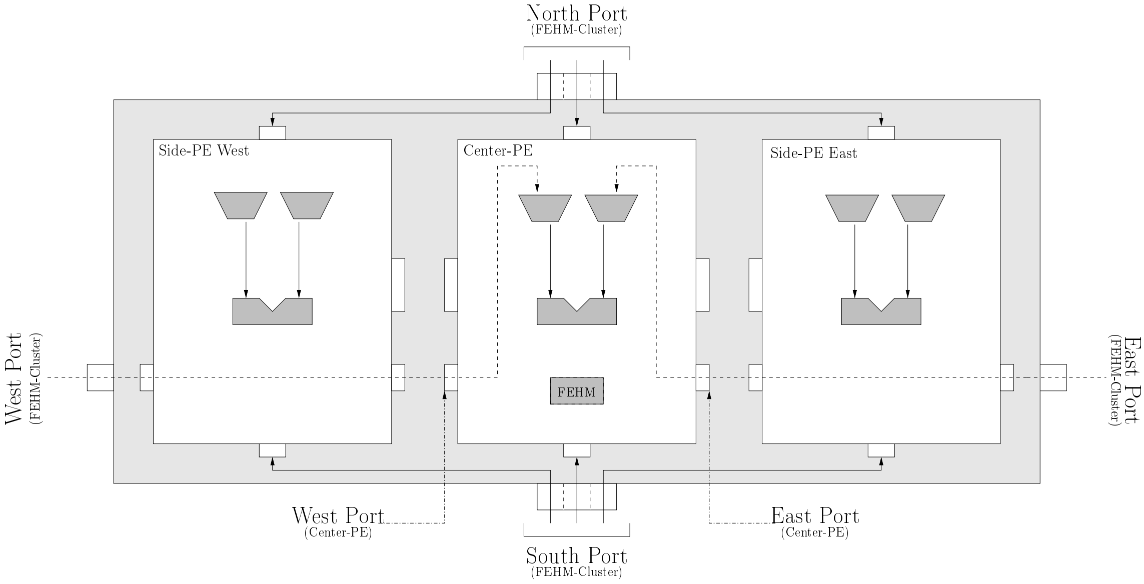


Figure 4.8: Data from the East and West FEHM-Cluster ports to the Center-PE.

Therefore, the longest internal route for the FEHM-Cluster is reached when the data come from the West and East ports. The entire path which the data will travel with that configuration is illustrated in Figure 4.9.

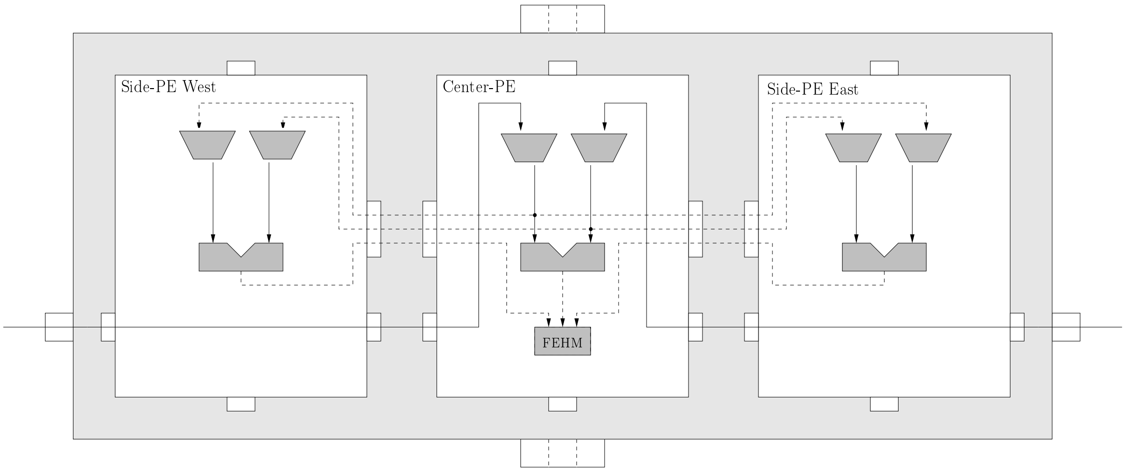


Figure 4.9: Longest internal route: The operands come from the East and West FEHM-Cluster ports.

4.2.2 Minimum Delay Scenario

The minimum delay scenario was also determined using three steps. These are practically the same steps used to discover the maximum delay scenario. The steps are:

First step: Discover the fastest FU operation.

Second step: Discover the best input vectors to the operation that was found in the first step.

Third step: Discover the shortest FEHM-Cluster internal route.

4.2.2.1 The fastest Functional Unit operation

The FU that was used in the architecture has sixteen operations (arithmetics, logicals, shifts, comparisons ...) and the follow interface:

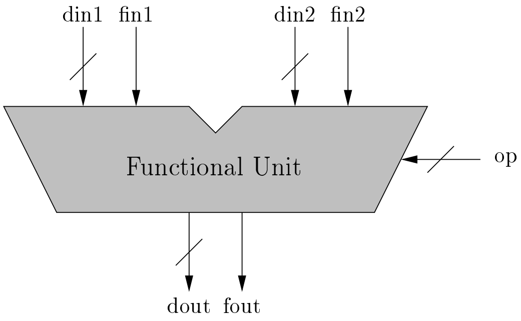


Figure 4.10: Functional Unit interface.

The signal *op* is the operation selector and it has 4 bits of width ($\log_2 16 = 4$). To discover among of all the operations what is the fastest, a different exercise was done. On this time, the Nanotime tool was used to discover the operation. Now, the circuit that was analyzed in the Nanotime is the whole Functional Unit. A method like whose in section 4.2.1.2 was done to discover the operation. First, all the input signals were let free to incorporate the values defined by the tool. The Figure 4.11 has a conceptual sight of the first step, the Nanotime Commands block 4.5 has all the commands to make this first analysis, and the Nanotime Results block 4.5 has the result.

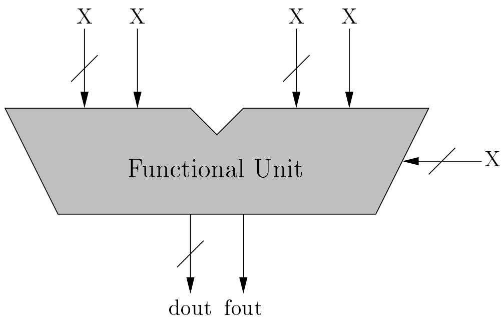


Figure 4.11: Signal values to the first Nanotime analysis.

Nanotime Commands 4.5: First analysis to found the fastest FU operation.

```
set search_path {.}
set library_path {.}
set link_path {*}

set oc_global_voltage 1.2
register_netlist -format spice {pe_cluster_h_fehm.sp \
tech.sp}
link_design fu_1
report_port

set input_ports {din1[*] din2[*] fin1 fin2 op[*]}
set_port_direction -input $input_ports
set output_ports {fout dout[*]}
set_port_direction -output $output_ports

report_port
report_design
report_net
report_cell
create_clock -name MCLK -period 10.0
report_clock
match_topology
check_topology

set_input_delay -clock MCLK -rise 1.0 $input_ports
set_input_delay -clock MCLK -fall 1.0 $input_ports
report_port -input_delay
set_output_delay -clock MCLK 0.0 $output_ports
report_port -output_delay
```

```
check_design

trace_paths -full_path_enumeration
report_paths -min -max_paths 8
```

Nanotime Results 4.5: Result to the first analysis.

```
                 Path   Path
Slack       Delay   Type      Startpoint          Endpoint
----------  ---------  ------  --  ----------------  --  -----------

     1.045       0.045     D–O  r   din1[6]           r   dout[6]
     1.045       0.045     D–O  r   din1[4]           r   dout[4]
     1.045       0.045     D–O  r   din1[7]           r   dout[7]
     1.046       0.046     D–O  f   din1[4]           f   dout[4]
     1.046       0.046     D–O  f   din1[7]           f   dout[7]
     1.046       0.046     D–O  f   din1[6]           f   dout[6]
     1.052       0.052     D–O  r   din1[3]           r   dout[3]
     1.052       0.052     D–O  r   din1[2]           r   dout[2]
```

Nanotime Results block 4.5 shows us that the fastest operation makes the data whose come from the pins din1[4], din1[6], and din1[7] take 0.045 ns long until the result be ready in the pins dout[4], dout[6], and dout[7] respectively. Table 4.1 shows the relationship between the pins extracted from Nanotime Results block 4.5.

input pin	transition	output pin	transition
din1[4]	rise	dout[4]	rise
din1[6]	rise	dout[6]	rise
din1[7]	rise	dout[7]	rise

Table 4.1: Relationship to the fastest path delay extracted from Nanotime Results block 4.5.

From this point, on an exhaustive way, all the operations were analyzed in the Nanotime. The operation code was defined using the command "set_case_analysis" for each operation. The result was compared with Nanotime Results 4.5. In the end of these analysis, three different operations got the expected result. The operations are in the Table 4.2.

Operation code(hex)	Operation
5	or
6	xor
A	==

Table 4.2: The fastest operation.

These three operations are part of the minimum delay scenario. Therefore, it is necessary to define de input values for each one of them.

4.2.2.2 The best input vectors

For each operation in Table 4.2, the operand *din2* and the flags signals (*fin1* and *fin2*) were set in zero. The Nanotime tool analyzed the circuit with this configuration and the results were like in Nanotime Results block 4.5 .Therefore, we can conclude that:

$$\begin{cases} din2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} & fin2 = \begin{bmatrix} 0 \end{bmatrix} \\ \\ fin1 = \begin{bmatrix} 0 \end{bmatrix} \end{cases} \tag{4.4}$$

Looking at Table 4.1 we can see that all the pins (din1[4], din1[6] and din1[7]) must take a rise transition. So, in the next Nanotime analysis the set of pins:

$$\{\text{din1[0]} \quad \text{din1[1]} \quad \text{din1[2]} \quad \text{din1[3]} \quad \text{din1[5]}\} \tag{4.5}$$

were tested to see if they can influence in the path delay of the other pins. For each operation in Table 4.2 the set of pins (4.5) took two different values. First, all the pins were set to low level, the circuit was analyzed and the result was compared with the result in Nanotime Result block 4.5 . In the sequence, the same procedure was done, however with all the pins in the high level. In both cases, the result was the same that in Nanotime Result block 4.5 . The Figure 4.12 shows the pins configuration to both cases.

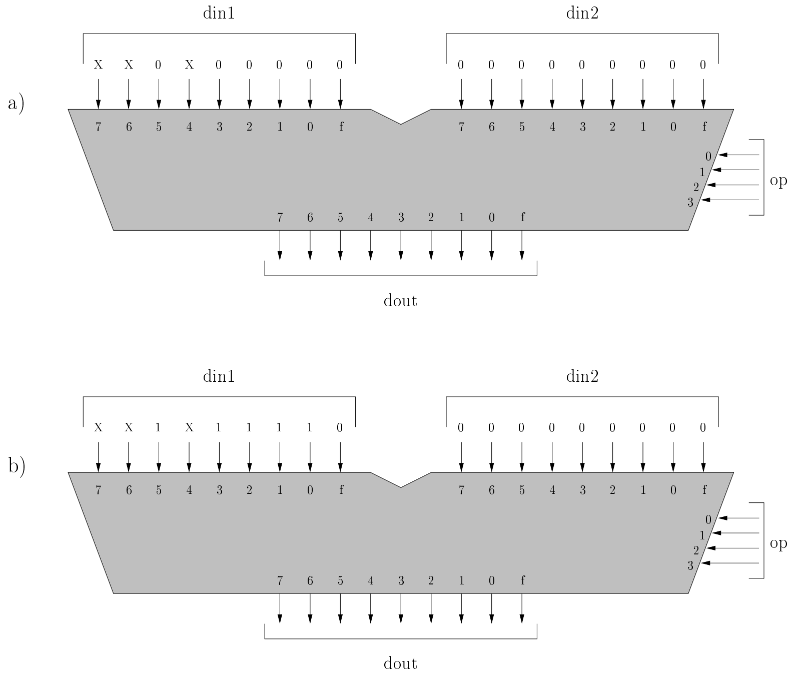


Figure 4.12: Configuration of the pins in the set (4.5).

As it was found, the pins in set 4.5 do not influence in the path delay from din1[4 − 6 − 7] to dout[4 − 6 − 7] when the FU is executing any operation of the Table 4.2. In addition, the only restriction in the pins din1[4 − 6 − 7] is a rise transition. Therefore, the follow values to signals din1, fin1, din2, and fin2 were chosen:

$$
\begin{cases}
din1 = \begin{bmatrix} R_7 & R_6 & 1 & R_4 & 1 & 1 & 1 & 1 \end{bmatrix} & fin1 = \begin{bmatrix} 0 \end{bmatrix} \\
din2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} & fin2 = \begin{bmatrix} 0 \end{bmatrix}
\end{cases}
\tag{4.6}
$$

In (4.6) the terms R_7, R_6, and R_4 represent a rise transition in the pins din1[7], din1[6], and din1[4] respectively.

4.2.2.3 The shortest FEHM-Cluster internal route

The shortest FEHM-Cluster internal route is that makes the shortest path between the operands source and the input ports of the Functional Unit. As operand source, I mean the ports. In section 4.2.1.3 was shown that the North and the South FEHM-Cluster ports have a direct connection with the North and South Center-PE ports respectively (Figure 4.8). Consequently, the data to the minimum delay scenario come from the North and the South ports like in Figure 4.13.

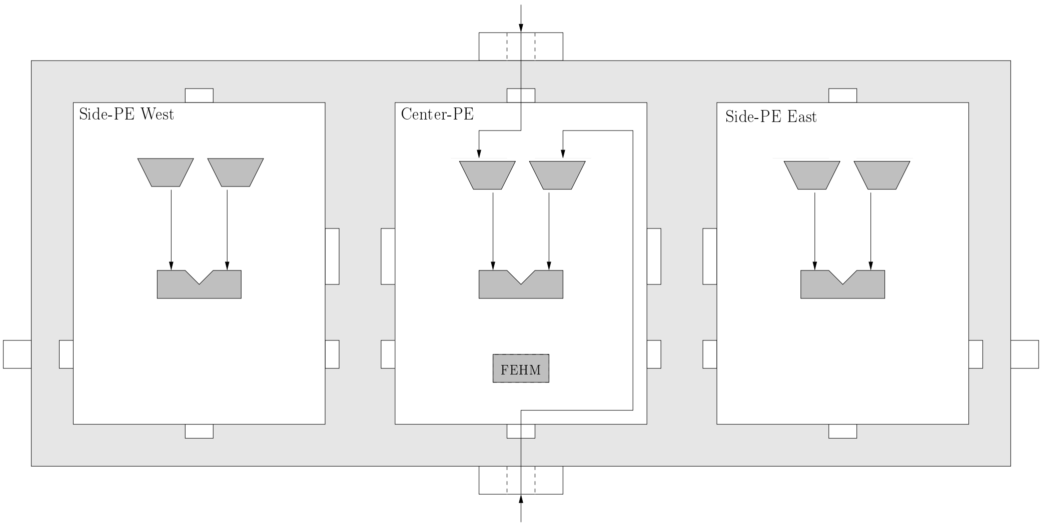


Figure 4.13: Shortest internal route.

The data come from the ports and do not come from the internal registers because was found that once the ports are selected to be the source of the FU operands, any change in this line (from the ports to the FU input) take less time than the time necessary to change a context. A register is selected by a context. Consequently, if you are only using registers, you should change the context to take another value.

4.3 D_{min} and D_{max} in the Electrical Level

To discover the delays value, the scenarios that were determined in section 4.2 were configured in the target approach.

The maximum delay of the monitored path in the electrical level is calculated in the section 4.3.1. Section 4.3.2 shows the calculus to the minimum delay with the Spice net-list simulation.

4.3.1 Maximum Delay (D_{max})

Since we want to find the delay between the high edge of the clock and the output of the DEDC, erros should be done and undone to get some transitions in *pe_temp*. To generate a error, one input channel of the FEHM should be different among the others. As this section is about the maximum delay, the channel that was chosen to be different is the channel that comes from the East Side-PE. All the same, it also could come from the West Side-PE, however, that was an arbitrary choice. The channel from the Side-PEs to the FEHM are longer than the channel from the Center-PE, because it has the FEHM inside. The others FEHM channels were took from the Center-PE North Port. The Figure 4.14 has the illustration about the sources of the FEHM input channels.

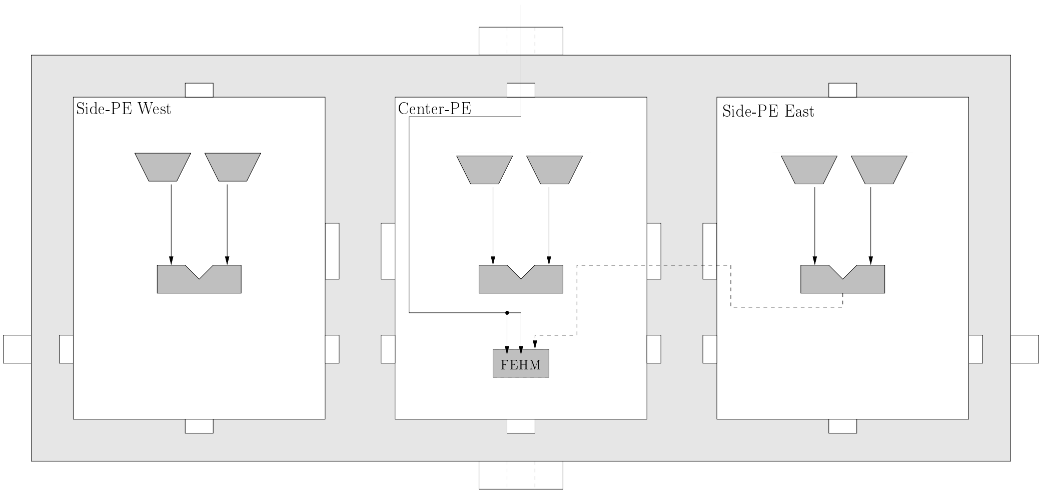


Figure 4.14: FEHM input channel sources.

Using the conception shown in Figure 4.14 and the longest route configuration discussed in section 4.2.1.3, the FEHM-Cluster configuration to the maximum delay calculus is shown in Figure 4.15.

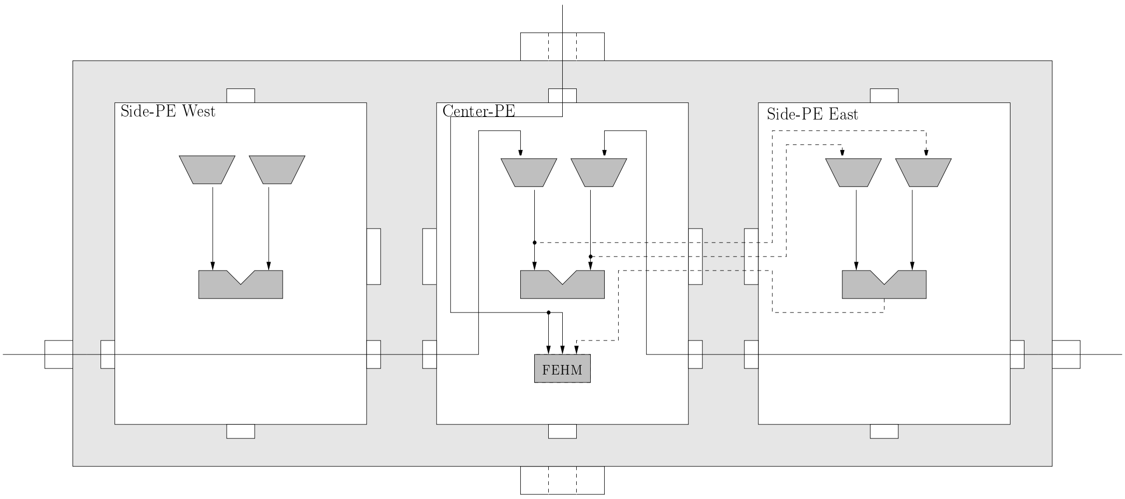


Figure 4.15: Paths configuration to the maximum delay calculus.

The behavioral verilog was changed with a direct connection between the North Port of the Center-PE and the two input channels of the FEHM (in2 and in3) to make the connections showed in Figure 4.14. The original and the changed code are shown in the Verilog code blocks 4.5 and 4.6 respectively.

Verilog code 4.5: Original code. The input channels of the FEHM came from the FU output of the East, West and Center PEs.

```
fehm #(.WIDTH(DATA_WIDTH + 1))
 inst_fehm (.out(outEDM), .eout(pe_out), .tout(te_out),
        .in1({frst_e_edm, drst_e_edm}),
        .in2({foutFU_tmp, doutFU_tmp}),
        .in3({frst_w_edm, drst_w_edm}),
        .enabled(edm_enabled), .clk(clk) );
```

Verilog code 4.6: Direct connection between the Center-PE North Port and the *in2* and *in3* input channels of the FEHM.

```
fehm #(.WIDTH(DATA_WIDTH + 1))
  inst_fehm (.out(outEDM), .eout(pe_out), .tout(te_out),
        .in1({frst_e_edm, drst_e_edm}),
        .in2({finN, dinN}),
        .in3({finN, dinN}),
        .enabled(edm_enabled), .clk(clk) );
```

Since the FEHM-Cluster was simulated in the Nanosim, the whole configuration process of each PE of the cluster was done using a .vec file. It describes all the logic values for each input node of the circuit in each instant of time. This .vec file was generated by a Python script created by the author of this work.

The .vec file has two parts: the configuration part and the execution part. In the first part, all the respective configuration streams are passed to each PE inside the FEHM-Cluster. The configuration streams have the route and operation informations. In the second part, all the values are passed to the FEHM-Cluster ports. The FEHM-Cluster was configured to take the firsts four bits of the East FEHM-Cluster port as B operand and the firsts four bits of the West FEHM-Cluster port as the A operand.

As it was said in section 4.2.1.2, nothing is clear about the transition of the *A[2]*. Consequently, two simulations were done. They are the “First case” and the “Second case”. In both cases, the B operand has a transition from B_{16} to F_{16}, because in section 4.2.1.2, it was defined that the pin *B[2]* should take a transition from 0 to 1 and all the other pins could be 1. The “First case” does the A operand transition from F_{16} to B_{16} and the “Second case” does the A operand transition from B_{16} to F_{16}. Table 4.3 has the values of the FEHM-Cluster ports to each simulation case.

The Table 4.3 also shows the values of the multiplication results and the “error” flag. Obviously, the “Error” column shows the comparison result between the North port value and the multiplication result. Each simulation case has two steps. These steps are necessary to make the transitions in the two operands and to purposeful generate the errors. The errors are generated in the first step of each case where a different value of the multiplication result is set in the North port. The second step sets the error signal in low level putting a value equal the multiplication result

Case	FEHM-Cluster Ports			Results	
	West(A)	East(B)	North	Multiplier	Error(*pe_temp[0]*)
First	0F	0B	00E100	A5	1
	0B	0F	00A500	A5	0
Second	0B	0B	00E100	79	1
	0F	0F	00E100	E1	0

Table 4.3: Simulation cases to find the maximum delay. All the values of the table are in hexadecimal.

in the North port. The delay is measured, in both cases, when the error signal (*pe_temp[0]*) does a stable transition from 1 to 0.

The reason to measure the delay in a stable transition from 1 to 0, and not in another way, is the calculation time to each bit of the multiplication result. There are bits that will be ready earlier than others, consequently, the *pe_temp[0]* will change from 0 to 1 when the earliest bit of the multiplication result comes carring a different logic value than the bit with the same index in the North port. On the other hand, the *pe_temp[0]* will change stably from 1 to 0 unless all the bits in the multiplication result have the same logic value as bits with the same index in the North port. Therefore, the *pe_temp[0]* fall transition takes longer than the rise transition, because it needs to wait until the latest bit of the FU result be ready.

Figure 4.17 and Figure 4.18 have the Nanosim simulation to the "First" and "Second" cases respectively. To both, the *pe_temp[2:0]* is the channel presented in the Figure 4.3, the *clk* is the global clock, and the other signals/vectors are localized like in the Figure 4.16.

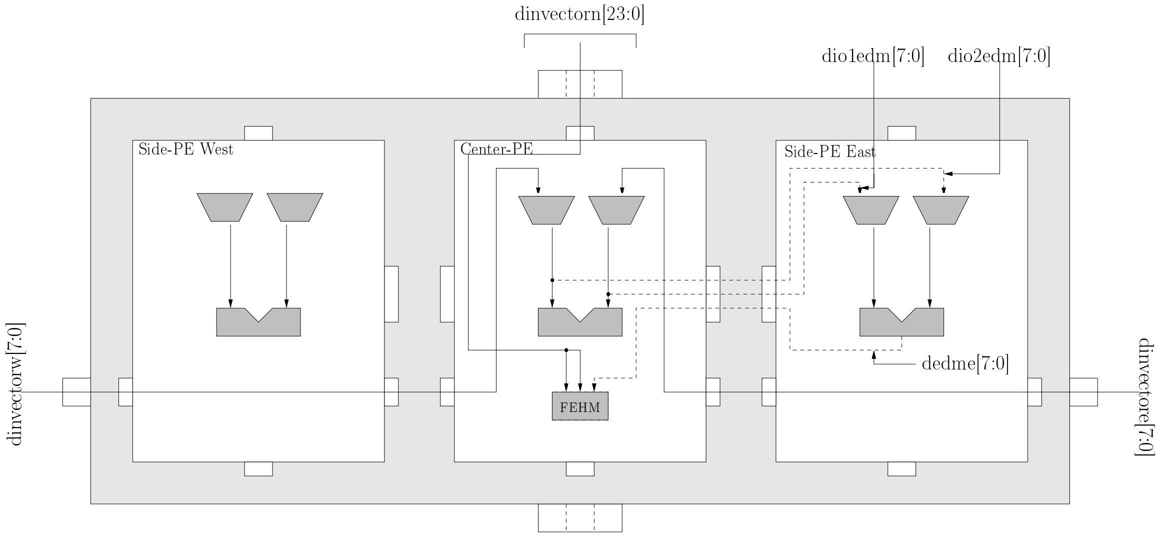


Figure 4.16: Mapping between the internal/external FEHM-Cluster signals and the signals shown in Figures 4.17 and 4.18.

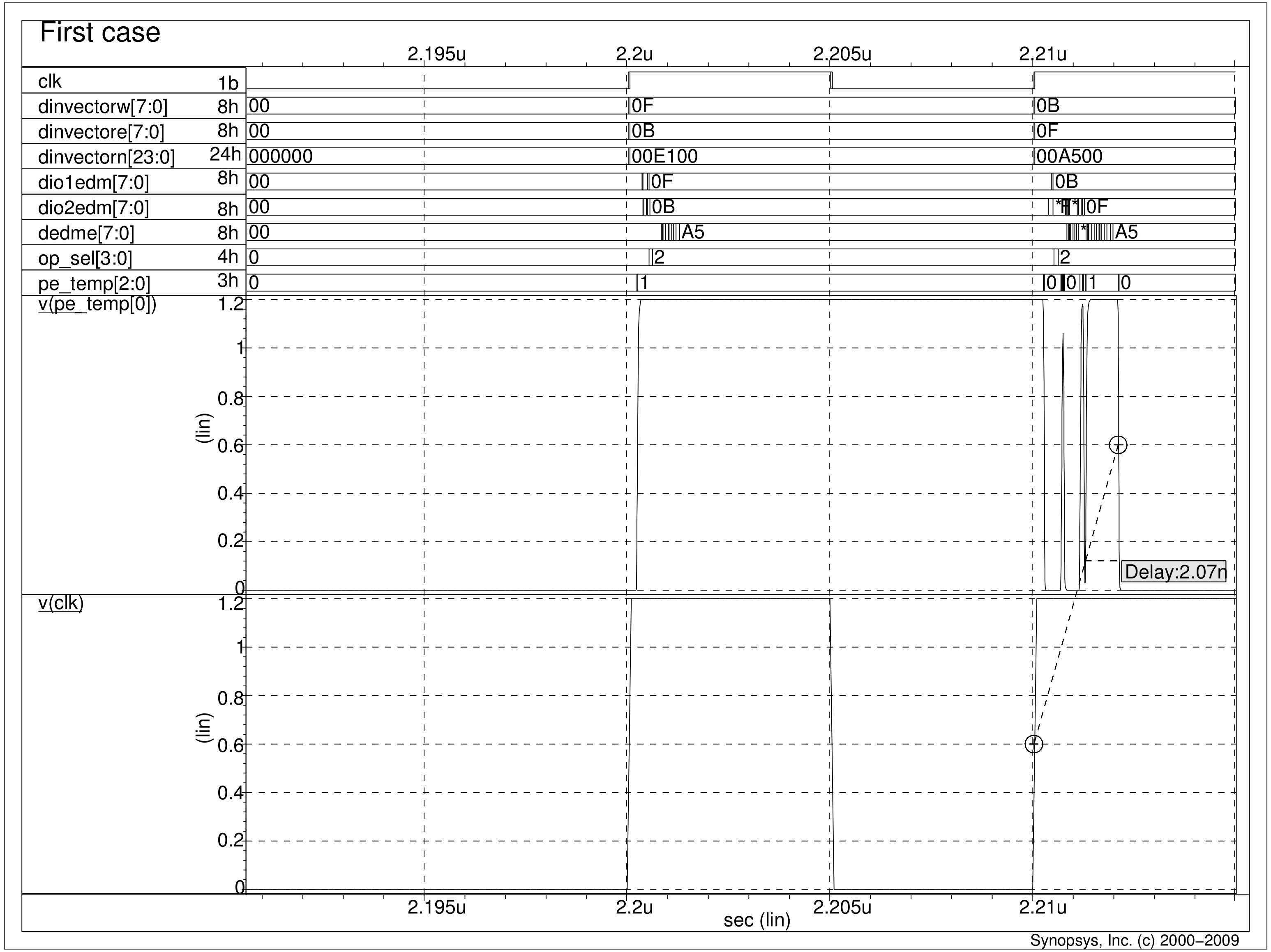


Figure 4.17: First case simulation. Delay equal to 2.07 ns.

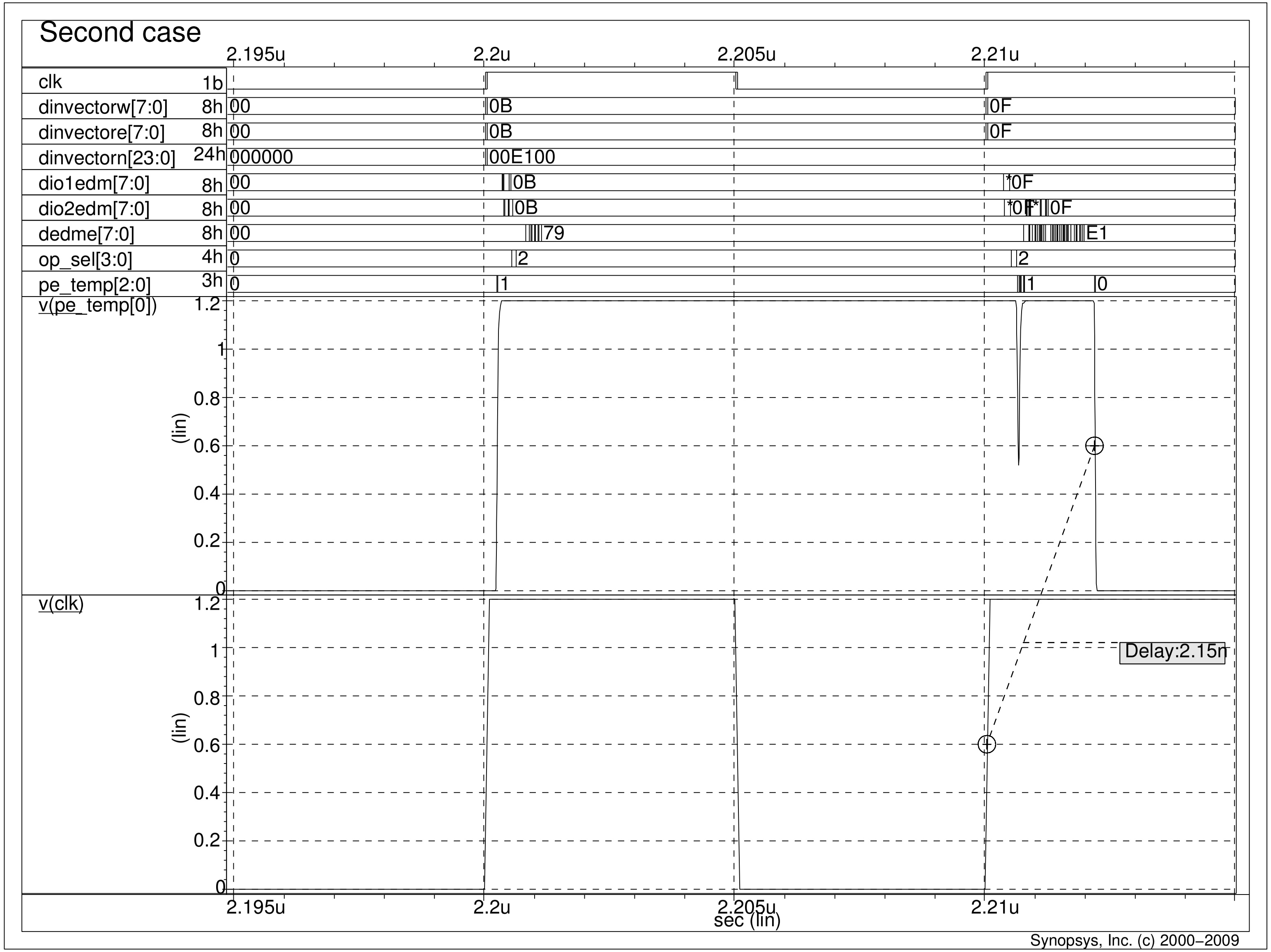


Figure 4.18: Second case simulation. Delay equal to 2.15 ns.

The delay that was measured in the second case simulation is bigger than the delay that was measured in the first case simulation. Therefore:

$$D_{max} = 2.15 \text{ ns} \tag{4.7}$$

4.3.2 Minimum Delay (D_{min})

As the FEHM is inside of Center-PE, the channel that will take a different value is the channel which comes from the Center-PE FU. Its is shown in the Figure 4.19.

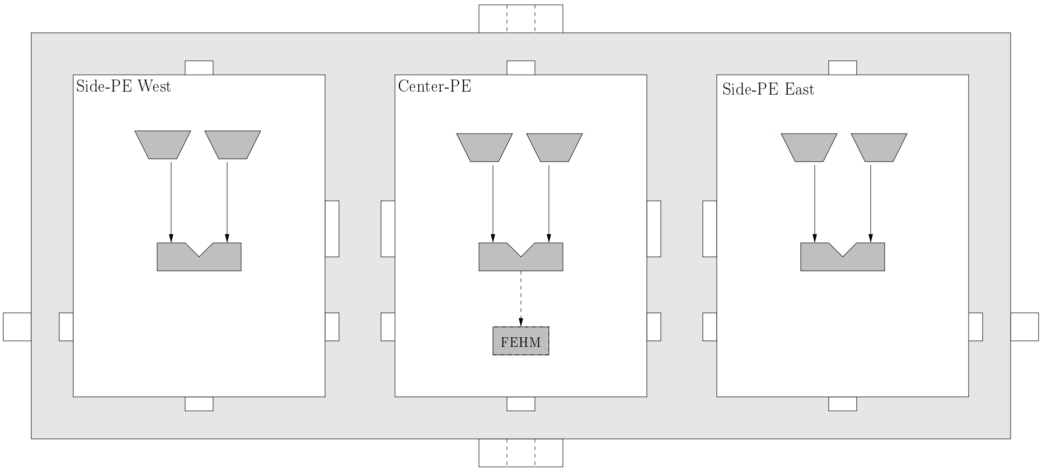


Figure 4.19: Channel that will take a different value (minimum delay).

The others channels of the FEHM were taken directly from the East FEHM-Cluster port. Like in maximum delay calculus, the behavioral verilog code of the Center-PE was hardly modified. The modification is presented in the Verilog Code block 4.7 and in Figure 4.20.

Verilog code 4.7: Modificated code. The input channels *in1* and *in3* of the FEHM come from the Center-PE East port.

```
fehm #(.WIDTH(DATA_WIDTH + 1))
 inst_fehm (.out(outEDM), .eout(pe_out), .tout(te_out),
          .in1({finE, dinE}),
          .in2({foutFU_tmp, doutFU_tmp}),
          .in3({finE, dinE}),
          .enabled(edm_enabled), .clk(clk) );
```

Like in section 4.3.1, a error must be generated with the purpose to get to the delay in the desired node (Figure 4.3). Therefore, each operation in Table 4.2 was simulated using 2 clock cycles. However, the error was generated in the second clock cycle and the delay was measured when the wanted node takes a rise transition. This was done in order to the reasons mentioned in section 4.3.1. Next table shows us the values of each signal that was used in the simulation to found the minimum delay.

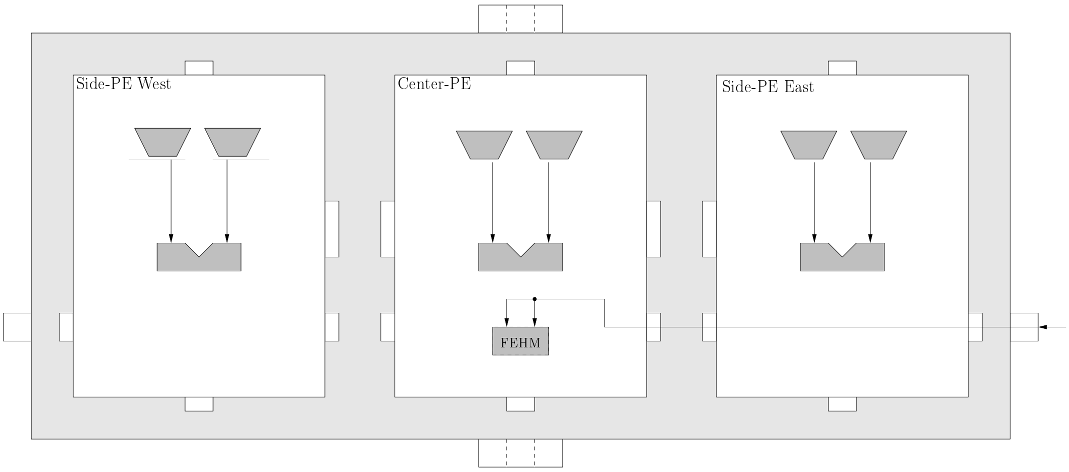


Figure 4.20: Illustration of the modified code in Verilog code block 4.7.

Operation	FEHM-Cluster Ports			Results	
	North(din1)	South(din2)	East	FU	Error(pe_temp[1])
5 (or)	000000	000000	00	00	0
	00FF00	000000	00	FF	1
6 (xor)	000000	000000	00	00	0
	00FF00	000000	00	FF	1
A (==)	000000	000100	00	00	0
	00FF00	000000	00	FF	1

Table 4.4: Signal values to find the minimum delay.

Figure 4.22 has the Nanosim simulation to the situations presented in Table 4.4. In Figure 4.22 the signal *clk* is the global clock, the signal *pe_temp[2:0]* represents the desired set of nodes which is presented in Figure 4.3, the *op_sel[3:0]* is the FU operation code, and the others signals are localized like in Figure 4.21.

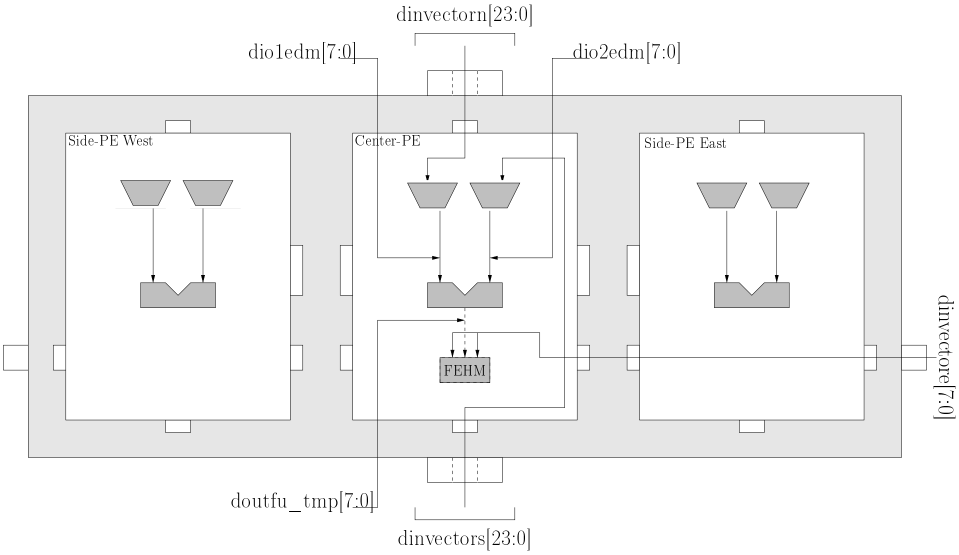


Figure 4.21: FEHM-Cluster signals localization (minimum delay).

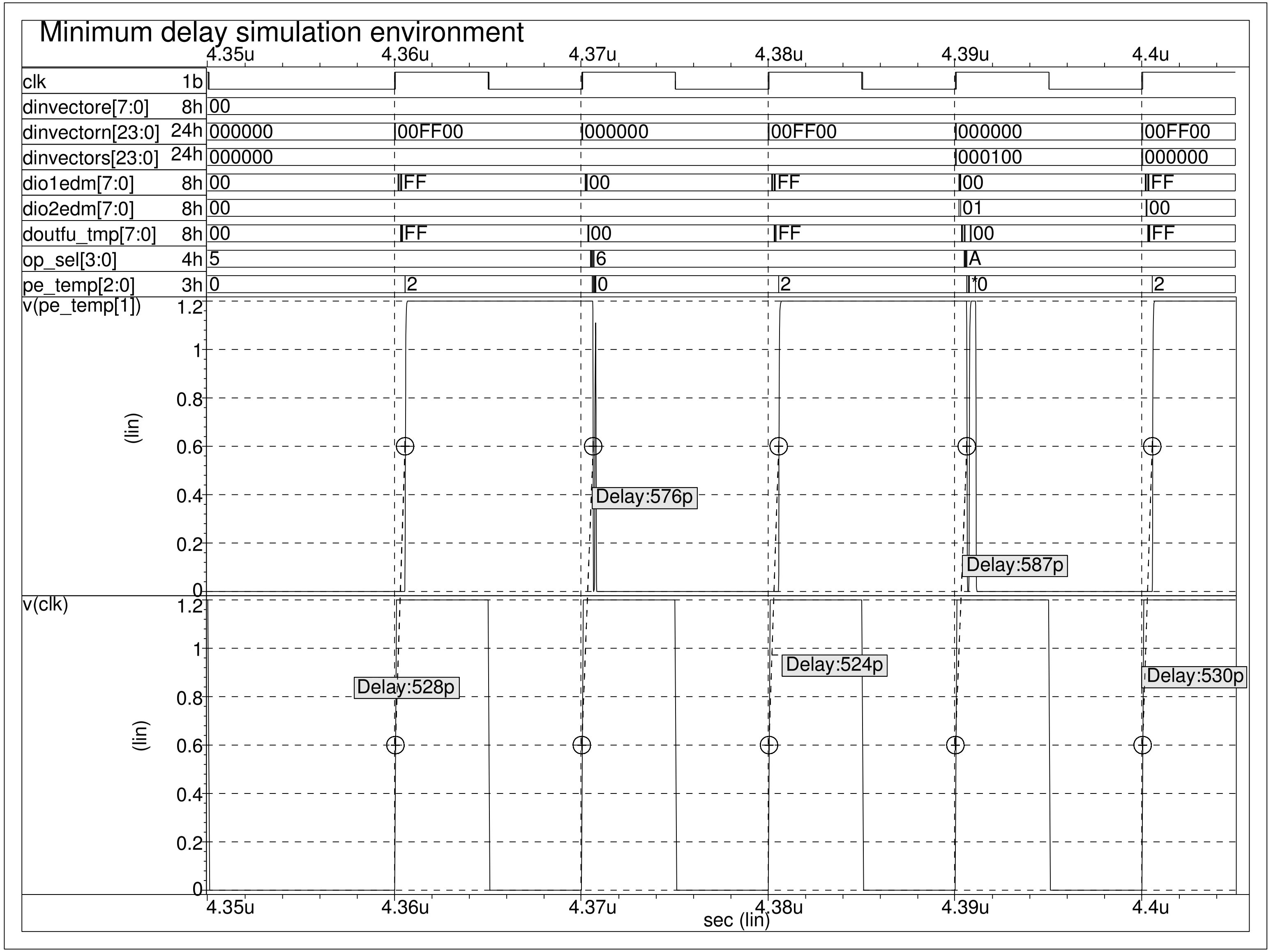


Figure 4.22: Simulation result to find the minimum delay.

Looking at the Figure 4.22 we can see that among the three operations, the XOR operation got the minimum delay. Consequently:

$$D_{min} = 0.524 \text{ ns} \tag{4.8}$$

4.4 Construction and simulation results

On this section, a unit validation of the sensors going to be performed. This validation is in the electrical level using the Spice net-list description. The sensors will be projected to the required parameters to monitor the path that ends in *pe_temp* (the output of the DEDC).

The parameters to the sensors implementations are the maximum (D_{max}) and minimum (D_{min}) delays of the monitored path, the clock period, and the clock duty cycle. To this validation, will be used a arbitrary clock period (T) of 10 ns and a clock duty cycle of 50 % ($\rho = 0.5$). As it was discovered, the D_{max} is equal to 2.15 ns and D_{min} is equal to 0.524 ns.

The objective of this section is to verify whether with the project parameters to the sensors, both implementations will work as expected in section 3.1. Once the relations that were presented in section 3.1 are here satisfied, it is possible to project the sensors to operate in the maximum performance of the target circuit (FEHM-Cluster).

4.4.1 DSTB implementation

As it was presented in section 3.1, the T_W of the DSTB, without a buffer in the input line, is all the clock duty-cycle period. In addition, the minimum path delay (D_{min}) must respect the (3.3) and exceeds the T_W period. However, to the project parameters, the condition does not is satisfied, since:

$$T_W = 0.5 * 10 \text{ ns} = 5 \text{ ns}$$
$$D_{min} = 0.524 \text{ ns}$$
$$T_W > D_{min}$$

In order to fix this situation, as it is shown in Figure 4.23, one buffer must be inserted in the data line of the latch. The goal of this buffer is to push out the fastest data from the clock duty-cycle period.

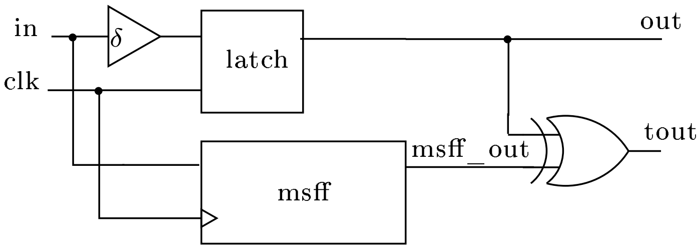


Figure 4.23: DSTB circuit with a extra delay element.

The first step to construct the circuit is to verify if the previously defined period T satisfies the condition (3.10). The latch that was used in this work has a time to hold equal to -0.033 ns and the flip-flop has a setup time of the 0.0418 ns in the transistor level. Then:

$$10 \text{ ns } \geq \frac{2.15 - 0.524 + 0.0418 - 0.033}{1 - 0.5} \text{ ns}$$

$$10 \text{ ns } \geq 3.2696 \text{ ns}$$

As one can see, the period T = 10 ns satisfies the condition (3.10).

Now, the delay δ can be discovered with (3.8). So:

$$\delta = 0.5 \ * \ 10 - 0.524 - 0.033 \text{ [ns]}$$

$$\delta = 4.443 \text{ ns} \tag{4.9}$$

The value $\delta = 4.443$ ns is theoretical. The buffer that was constructed has a approximate value. It is 4.444 ns to a positive edge and 4.455 ns to a negative edge. As a result, the DSTB circuit that was implemented has one detection period to rise transitions (T_{W_r}) and another to fall transitions (T_{W_f}) of data. Both of them start before of clock positive edge and end before of D_{min}. The reason to both start before the clock latch edge is the Shadow Flip-Flop time to hold. Some Nanosim commands were used to calculate the two detection periods. The results were:

$$\begin{cases} T_{W_r} = 0.544\text{ns} \\ T_{W_f} = 0.555\text{ns} \end{cases} \tag{4.10}$$

The T_{W_r} starts on 0.024 ns before the clock positive edge and ends 0.520 ns after that. T_{W_f} starts 0.034 ns before the positive clock edge and ends 0.521 ns after. As a result, the data that take 0.524 ns longer to be ready in the DSTB input node will not be get as a transient error.

Figure 4.24 shows a simulation in Nanosim where there are any errors. The data come always in D_{valid} period. Figure 4.25 shows a simulation where the data come in the two detection periods. To both images, the signals that are presented are placed as in Figure 4.23.

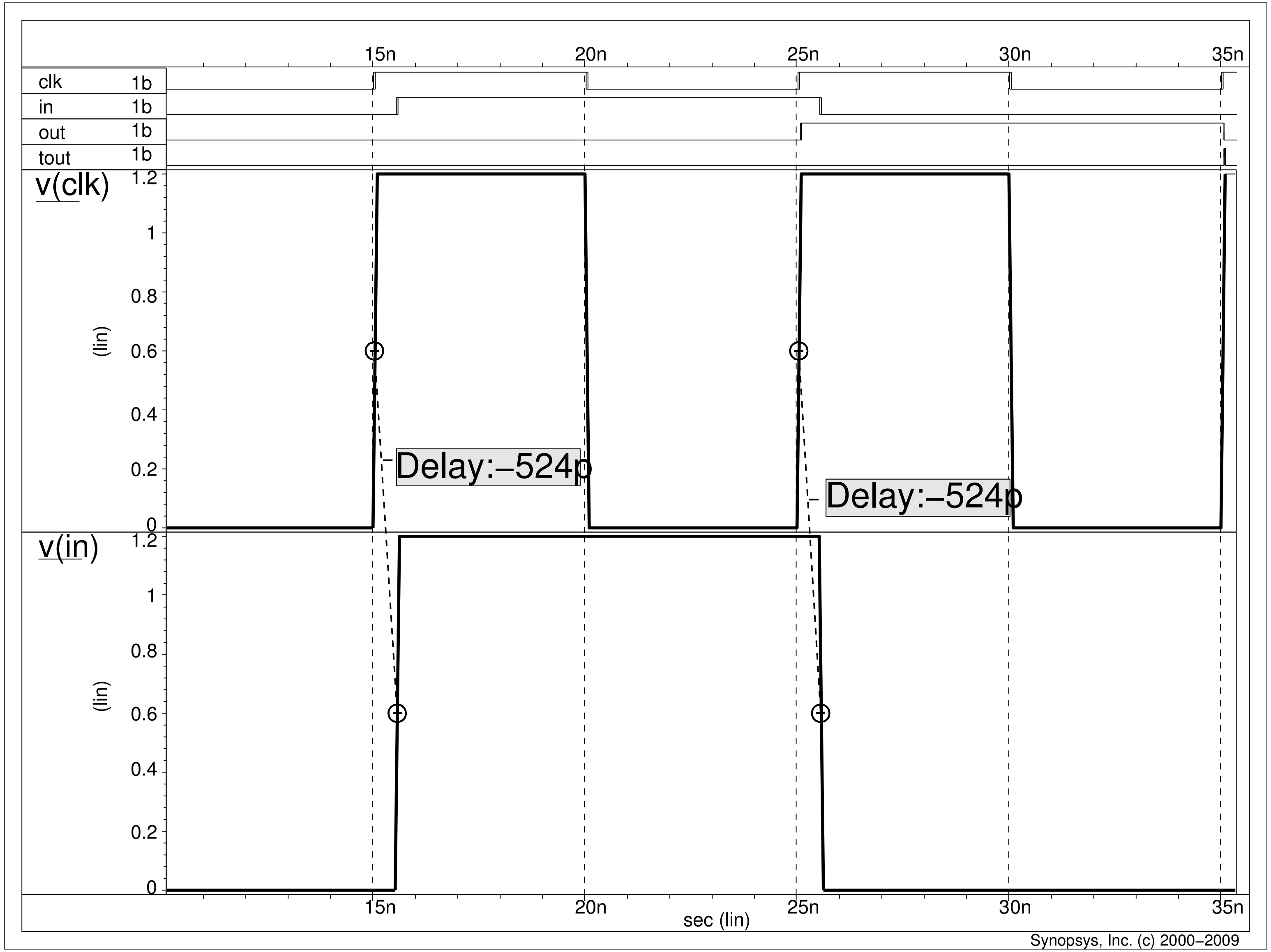


Figure 4.24: DSTB Spice simulation without errors.

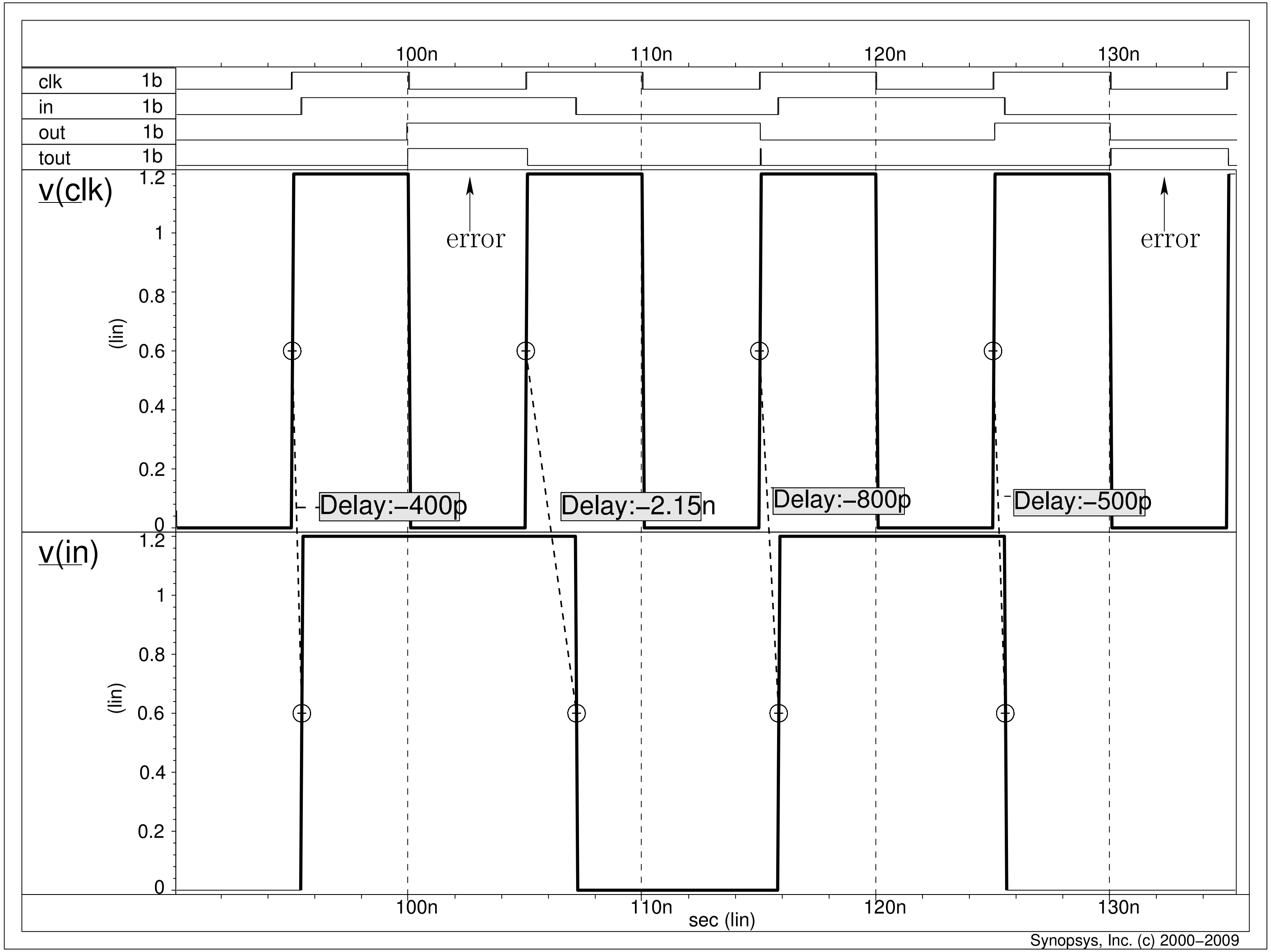


Figure 4.25: DSTB Spice simulation with errors.

4.4.2 ED implementation

In section 3.1, it is said that the δ value to the buffer which will be inserted in the clock line of the ED must to be calculated as $\delta = D_{min} - T_{h_f}$. As D_{min} was found to a rise transition, the time to hold that was used was also the flip-flop T_{h_f} to a rise transition, consequently, $T_{h_f} = -0.018$ ns. As a result, δ is theoretically equal to 0.542 ns. However, the practical buffer has a positive edge delay equal to 0.541 ns. Therefore, there is a little window of time where the errors will not be detected.

The T_{W_r} and T_{W_f} to the ED are equal to:

$$\begin{cases} T_{W_r} = 0.539 \text{ ns} \\ T_{W_f} = 0.532 \text{ ns} \end{cases} \tag{4.11}$$

T_{W_r} and T_{W_f} respectively begin 0.022 ns and 0.038 ns before the clock latch edge. Figure 4.27 presents the simulation to the ED sensor to a situation which there is any error. Figure 4.28 shows that when a transition occurs in the period between the rise edge of the clock and D_{min}, it will be taken as a error caused by transient fault. Both images presents the simulation values to the nodes presented in Figure 4.26.

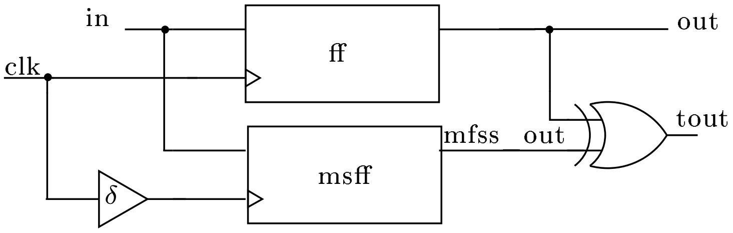


Figure 4.26: Signals location to the ED simulations of the Figure 4.27 and Figure 4.28.

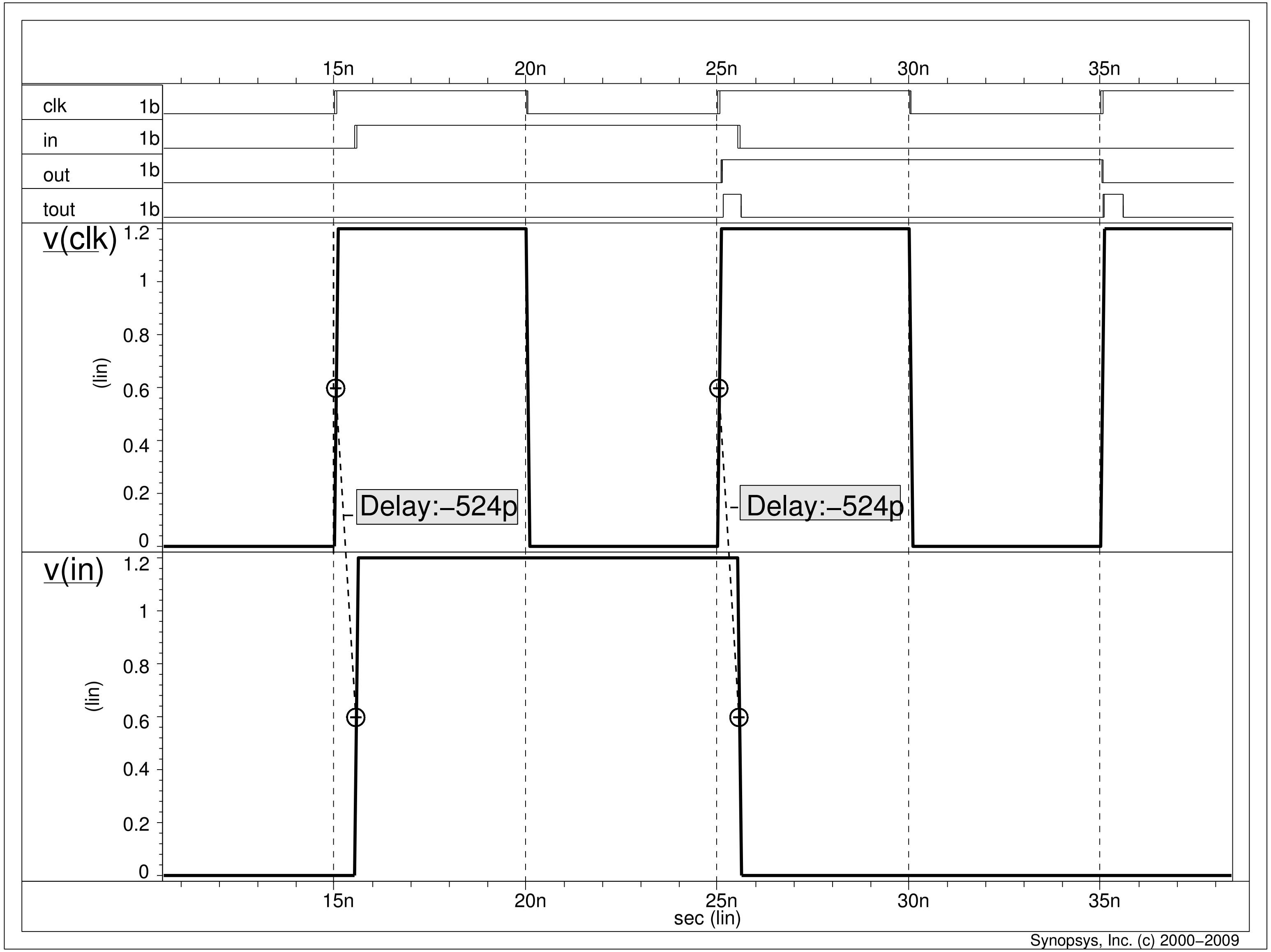


Figure 4.27: ED Spice simulation without errors.

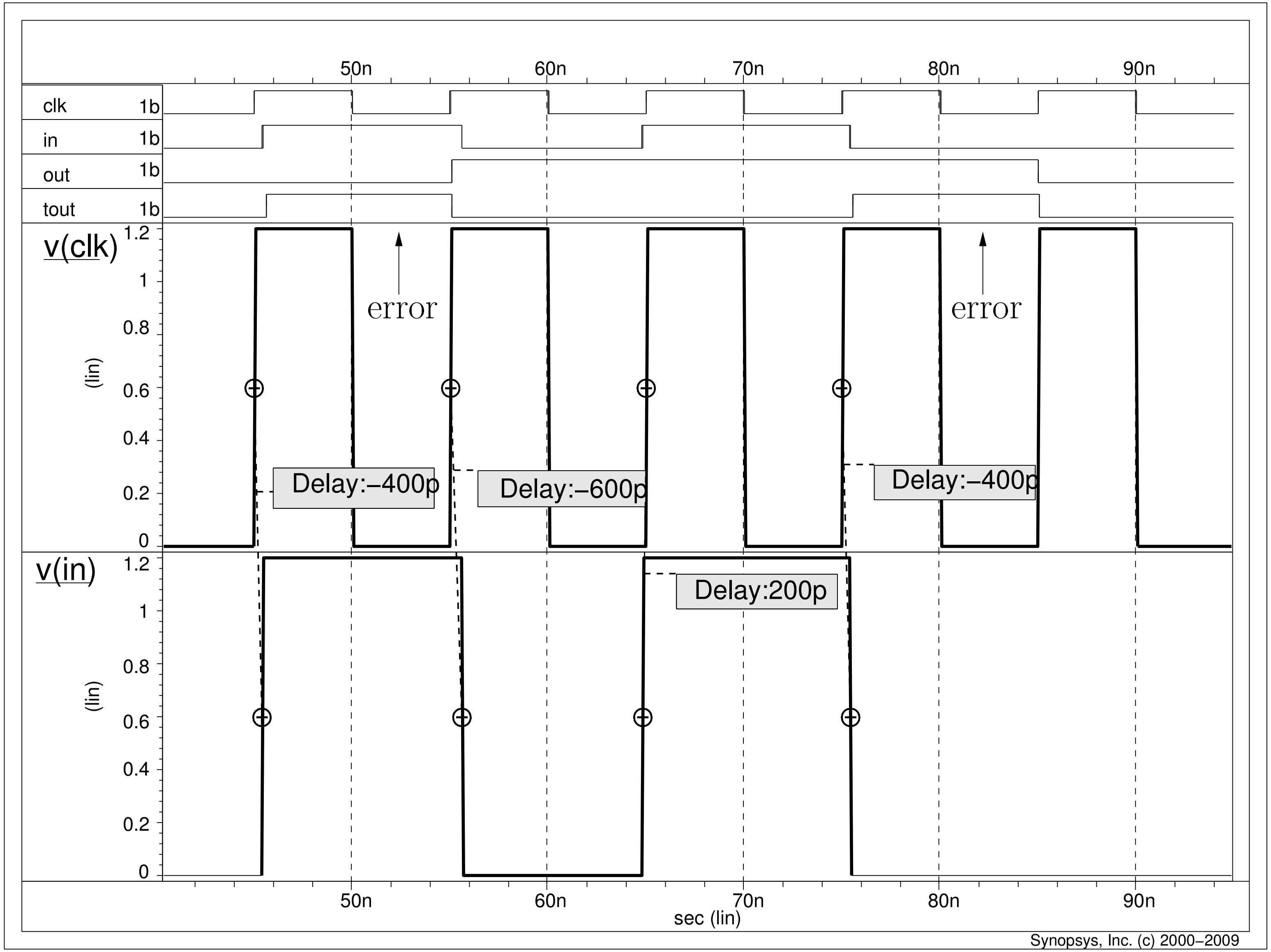


Figure 4.28: ED Spice simulation with errors.

4.5 Synthesis results

The results were obtained to the architecture be implemented in an ASIC using the Nangate FreePDK45 Generic Open Cell Library (SI2, 2012). It is a open cell library to the technology of 45 nm.

The Table 4.5 shows the area results to the FEHM without the sensors, and the sensors implementation without the buffers.

Block	number of logic gates	area(μm^2)
FEHM without sensors	156	96.8239
ED sensor without buffer	21	19.152
DSTB sensor without buffer	16	14.896

Table 4.5: Individual blocks areas to ASIC construction to 45 nm.

To the sensors project parameters that were defined at the beginning of the chapter, the buffers that were implemented in sections 4.4.2 and 4.4.1 have area and delay that are presented in Table 4.6.

Block	number of logic gates	area (μm^2)	δ (ns)
ED buffer	41	39.368	0.541
DSTB buffer	343	366.016	4.444

Table 4.6: Buffers areas and delays to a clock period of 10 ns.

In Table 4.6 it is possible to notice that the buffer to the ED sensor is smallest than the buffer to the DSTB. The explanation to this effect is the reference edge of the clock that is used to construct the buffer and the D_{min} value. On this project, the D_{min} is equal to 5.24 % of the clock period. So, the minimum path delay is nearest of the rise edge of the clock than the fall edge. The buffer construction to the ED sensor took as reference the rise edge of the clock instead of DSTB buffer that took the fall edge.

To each sensor type, the FEHM has three instances of each one. So, the total area to a FEHM implemented with ED is the sum of the FEHM area without sensors plus three times the sum of the ED sensor with its buffer. The same is true to a FEHM that is implemented with the DSTB sensor. So, Table 4.7 presents the total area of the FEHM with the sensors and buffers included.

block	number of logic gates	area(μm^2)
FEHM with ED	342	272.3839
FEHM with DSTB	1233	1239.5599

Table 4.7: Total area to the FEHM with both sensors.

As the target circuit to the sensors project is the FEHM-Cluster, D_{min} and D_{max} will never be change if the configuration of the cluster (data width and number of internal registers) is the same. However, the clock period that was used in the parameters was not the minimal clock period of the FEHM-Cluster. The maximal frequency of the cluster, without the sensors, is about to 443 MHz (2.256 ns) with a duty cycle of 50 % ($C_{duty} = 1.128$ ns). D_{min} still inside of the duty-cycle ($0.514\ ns < 1.128\ ns$), then, to implement a DSTB sensor to work with this parameters it still be necessary to insert a buffer in the clock line. Therefore, using the equation 3.4, $\delta = 0.571\ ns$ to the DSTB buffer. In addition, the relation (3.10) (pg. 29) is not satisfied to this parameters, since $2.256\ ns < 3.2696\ ns$. Then, the performance of the circuit will be affected if the DSTB is used. Table 4.8 presents the FEHM area and the FEHM-Cluster best performance with and without the DSTB inclusion in the TEDC of the FEHM.

DSTB inclusion	FEHM area(μm^2)	FEHM-Cluster performance (ns)
yes	266.1651	3.2696
no	96.8239	2.256

Table 4.8: Impact of the DSTB sensor in the performance of the FEHM-Cluster and in the area of the FEHM.

The performance with the ED inclusion is the same as without the sensor. The only thing that is altered is the area of the circuit. It is presented in Table 4.7.

5 FAULT INJECTION AND RESULTS

Once the sensors implementation process worked in the electrical level, the FEHM behavior was analyzed with both sensors. In addition, the errors diagnostic capability was measured to this module.

Using the output values of the sensors it is possible to inform if the error that was detected was caused by a persistent or a transient fault. A SEU, that is a transient fault, could be interpreted as a persistent fault if the bitflip that is caused by the SEU generates an error at some clock cycles after the cycle when it occurred. Therefore, as only transient faults were injected, it is expected that all the errors generated by the faults with duration equal or less than D_{min} be diagnosed as provided by a transient fault.

The fault injection was performed in the logic level, because it was necessary 20,000 (twenty thousand) clock cycles to each injected transient fault width. Therefore, a big number of clock cycles were simulated, then, a simulation in the electrical level would be onerous. Another reason to the simulation in the logic level is the used tool (Modelsim SE) and the HDL language to the testbenches construction (SystemVerilog).

The FEHM is a module that is formed by 156 logic gates plus the gates that are necessary to construct the sensors. The module was tested with the two sensors implementations that were chosen to be implemented in this work. To turns the circuit more sensitive to a fault occurrence, all the simulations were made with the circuit working in the maximum FEHM-Cluster clock frequency (443 MHz). However, only the FEHM was simulated not the whole cluster.

As the fault injection was performed in the logic level, it is necessary to calculate the delays of the monitored path and project the sensors to work with this parameters. Therefore, the project of the sensors in the logic level is presented in section 5.1. The scheme of the fault injection is presented in section 5.2. The test cases of analysis are presented in section 5.3 and the results of the fault injection to the test cases are shown in section 5.4.

5.1 Logic level Project

To the fault injection, the circuit should operate in the maximum frequency, because it is the minimum clock period, then, the error rate will be bigger than in other clock frequency. Therefore, the sensors going to be projected to operate with a clock frequency of 443 MHz ($T = 2.256\ ns$) and a duty cycle of 50% ($\rho = 0.5$). In addition, to construct the sensors, as it was said in the section 4.4, it is necessary the values of the maximum and minimum delay of the monitored path. For this reason, the delay values going to be obtained in the section 5.1.1 and the δ to each type of sensor will be calculated in the section 5.1.2.

5.1.1 Delays in the Logic level

The scenarios that were obtained in the section 4.2 were used together with the configurations that were defined in the section 4.3. From the cases which were presented to the maximum delay, it was possible to discover that its value (D_{max}) is 1.806 ns. The minimum delay value (D_{min}) is 0.614 ns.

5.1.2 Buffers construction in the Logic level

As it was shown in section 4.4.2, the δ_{ED} value of the ED must to be generated by the follow equation:

$$\delta_{ED} = D_{min} - T_{h_f} \tag{5.1}$$

To discover the T_{h_f} value in the logic level, a testbench was made to this purpose. The result is that the T_{h_f} to a rise transition is equal to -0.023 ns and to a fall transition is equal to -0.088 ns. All the same in section 4.4.2, the time to hold which was chosen is to a rise transition, since the minimum delay was obtained to this kind of transition. Therefore, using the equation (5.1) the δ_{ED} value is the follow:

$$\delta_{ED} = 0.614 - (-0.023) = 0.637\ ns \tag{5.2}$$

On this simulation level, it is possible to create a buffer that generates exactly the δ_{ED}. One just need to insert the buffer in the gate description verilog and define the delay of the buffer in the SDF file.

In the maximum frequency it still need to insert a buffer that generates a delay δ_{DSTB} in the input line of the DSTB, because the minimum delay is inside of the DSTB detection period. However, different of the electrical level, the minimum clock period satify the condition 3.10. So, using the equation presented in 4.4.1 it is possible to obtain the δ_{DSTB}. The equation is in the follow:

$$\delta_{DSTB} = \rho\ *\ \mathrm{T} - D_{min} + T_{h_l} \tag{5.3}$$

The same interactive method was used to determine the time to hold to the latch (T_{h_l}). As a result, the time to hold to a rise transition is -0.023 ns and to a fall transition is -0.088 ns. As the maximum delay was determined with a fall transition, then, the time to hold value that will be used is -0.088. Therefore, the δ_{DSTB} is:

$$\delta_{DSTB} = 0.5\ *\ 2.256\ ns - 0.614\ ns + (-0.088\ ns) = 0.426\ ns \tag{5.4}$$

With the two δ defined in the logic level, it is possible to make the time simulation with the fault injection in the Modelsim tool.

5.2 Fault Injection Scheme

To make the fault injection, it was necessary the development of three routines. A routine to simulate the dynamic of the input channels of FEHM. A routine to inject de faults in the module, and in the input channels. Another routine to collect the values of the desired output signals. The scheme is shown in a graphic way in Figure 5.1.

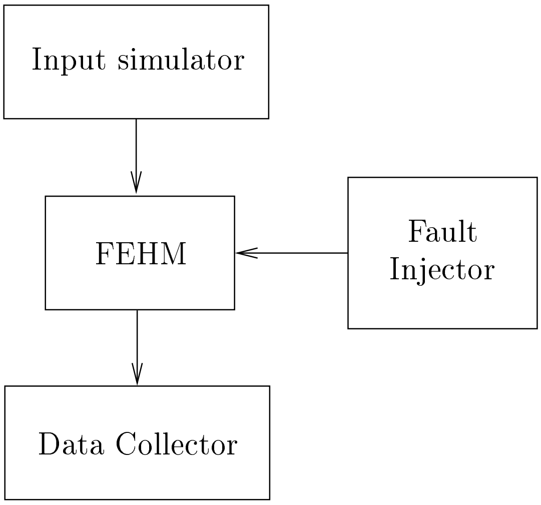


Figure 5.1: Fault injection scheme.

The section 5.2.1 explained as the input simulator was made. Section 5.2.2 describes the fault injection methodology. Finally, the section 5.2.3 describe where and when the data are collected.

5.2.1 Input simulator routine

As only the FEHM was used to the fault injection, it was necessary to make a routine that simulate the input channels behavior of the module. The routine is demonstrated in the diagram below:

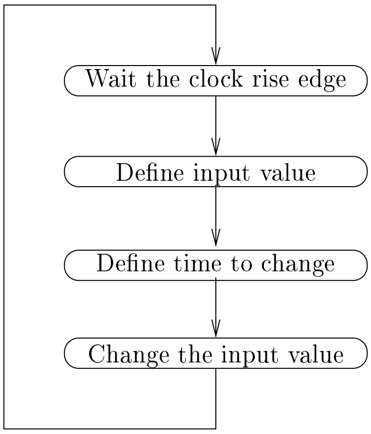


Figure 5.2: Input simulation diagram.

It is shown in the diagram that at each clock cycle the input channels change theirs value. The value and the moment when the data will change are defined randomly.

To the correct input channels simulation, the data can not change in any moment of the clock period. Thus, the transition moment is chosen randomly inside of a T_i period. It is represented in Figure 5.3.

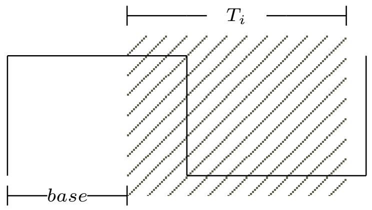
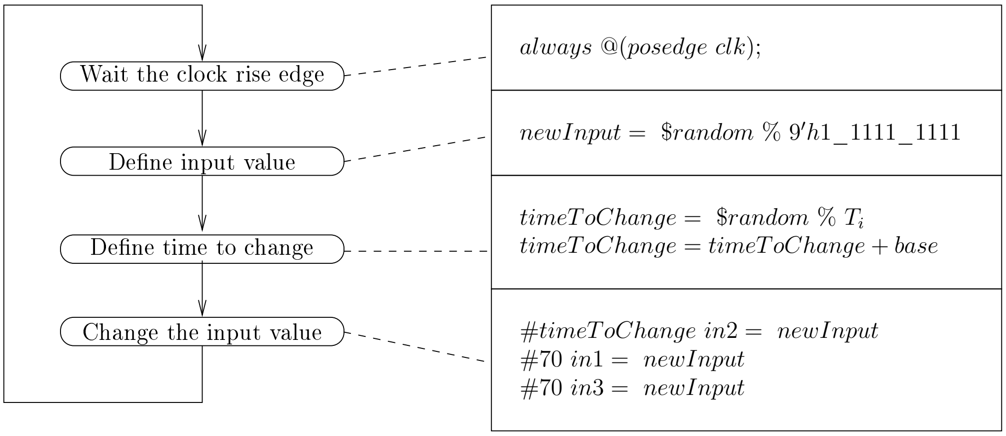


Figure 5.3: Period to change the value of the input channels.

The T_i is equal 1.192 *ns* and the *base* is equal to 0.470 *ns*. Any changes in the input channel value in this period do not generate errors.

The Verilog commands that were used to construct the routine are presented in Figure 5.4 below.



```
always @(posedge clk);

newInput = $random % 9'h1_1111_1111

timeToChange = $random % Ti
timeToChange = timeToChange + base

#timeToChange in2 = newInput
#70 in1 = newInput
#70 in3 = newInput
```

Figure 5.4: Verilog commands to the input simulations.

Looking at the Figure 5.4, it is possible to note that the value of the three channels are changed in different time moments. This difference was noted in the simulations. So, to take a more realistic simulation of the input channels it was considered in the routine.

5.2.2 Fault injector routine

The transient faults were inserted in the module at every two clock cycles. In a first moment, the injection was made at each clock cycle, but it was noticed that a fault overlap occurred in some cases. With faults at every two clock cycles, it was possible to analyze the individual effects of each injected fault. The fault injection routine is presented in Figure 5.5.

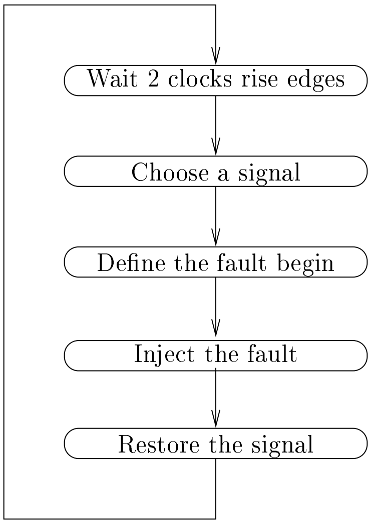


Figure 5.5: Fault injection routine.

As the signal where the fault will be inserted as the moment when the fault will begins are set randomly. The fault injection act, in this context, refers to take the signal original value and inverts it in the fault begin moment. The signal remains with its inverted value during all the fault duration period.

The fault can be injected on any moment of the T_f period. As it is illustrated in Figure 5.6, the T_f is exactly the clock period. In addition, the Figure 5.7 shows the verilog commands that were used to make the fault injection routine.

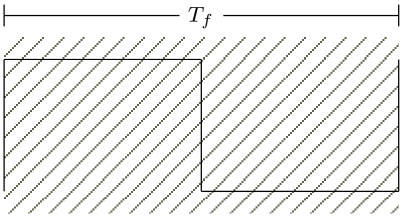
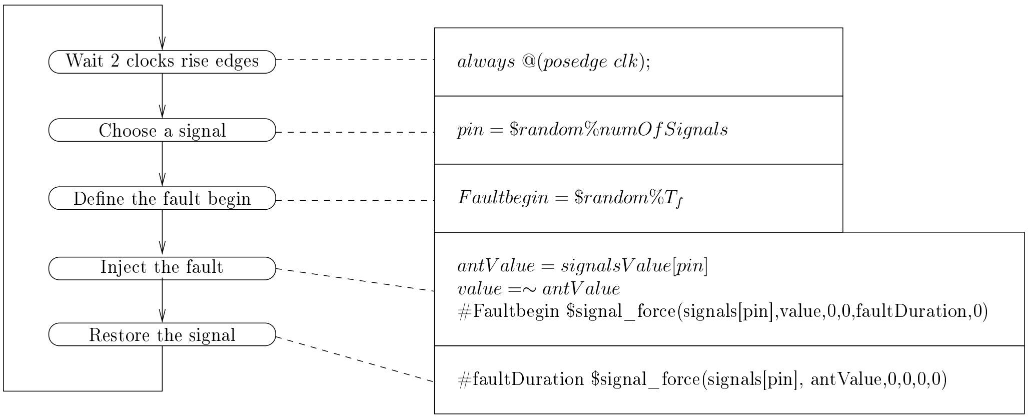


Figure 5.6: Injection period.

With the routine showed in this section, 10.000 faults were injected in the FEHM using three test cases. This cases going to be presented in the section 5.3.



```
always @(posedge clk);
pin = $random%numOfSignals
Faultbegin = $random%T_f
antValue = signalsValue[pin]
value =~ antValue
#Faultbegin $signal_force(signals[pin],value,0,0,faultDuration,0)
#faultDuration $signal_force(signals[pin], antValue,0,0,0,0)
```

Figure 5.7: Verilog commands to the fault injection.

5.2.3 Data collector

The FEHM behavior was analyzed through signals *out*, *eout*, and *tout* as it is presented in Figure 5.8. The *out* signal has the value that appears in more than one input channel of the FEHM. The *eout* signal indicates if a input channel has a different value among the others. The signal *tout* indicates whether the error that has occurred was caused by a transient fault.

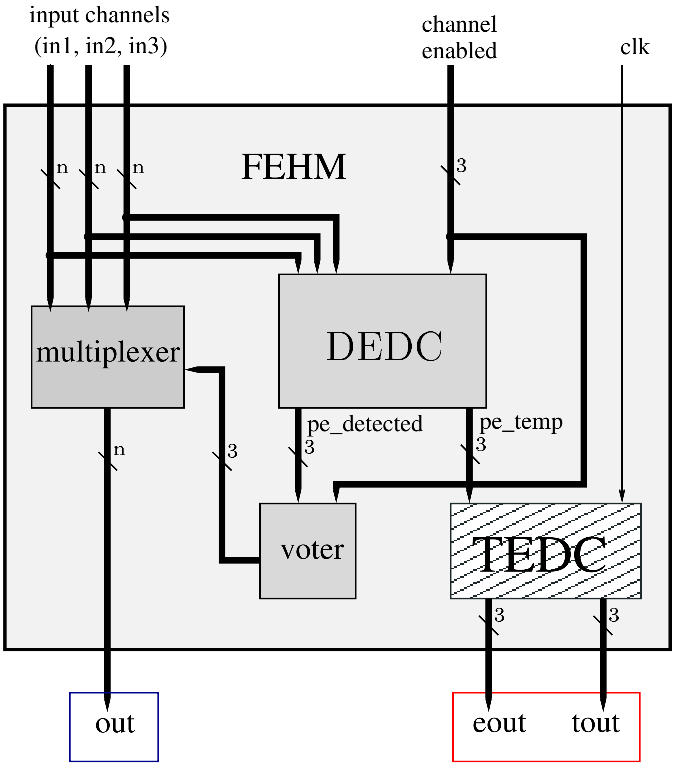


Figure 5.8: Analyzed FEHM signals.

The *out* signal is read in a different moment from the others. It is collected at every rise edge of the clock, while the signals *eout* and *tout* are collected in the fall edge of the clock. However, only the signals of the ED are read exactly in the fall edge of the clock, the signals of the DSTB are collected some picoseconds after that. The reason to this is that the *eout* signal comes from a latch which is transparent in the high level of the clock. When the data arrived very close of the clock fall edge, the latch generates small oscillations during the memorization period. Thus, just a few picoseconds after the fall edge, the data will be stable in the latch output. The Figure 5.9 presents the sensor signals mapping with the monitored FEHM signals, and Figure 5.10 illustrates the read moments of the signals.

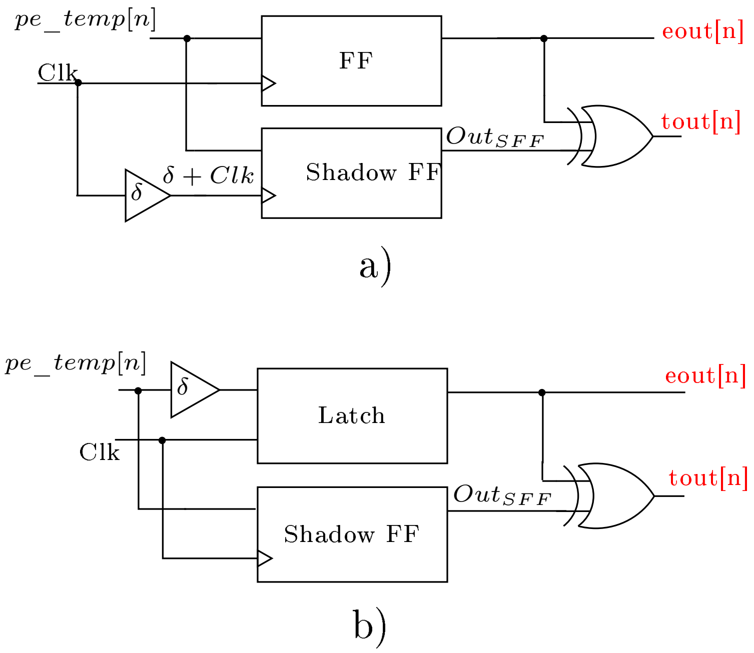


Figure 5.9: Sensors signal mapping: a) ED mapping; b) DSTB mapping.

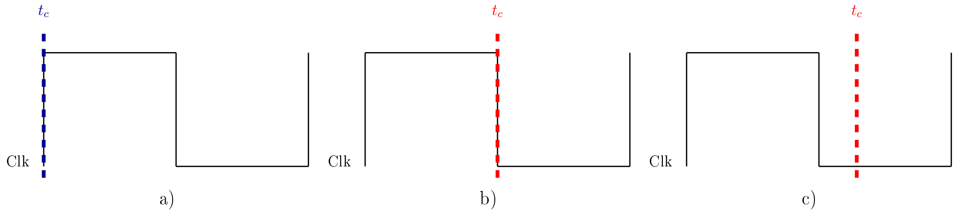


Figure 5.10: Read moments: a) *out* signal; b) *eout* and *tout* to the ED; c) *eout* and *tout* to the DSTB.

5.3 Test cases

Transient faults with duration (width) of 20%, 25%, 30%, 45%, and 50% of the clock period were injected in the FEHM to obtain the behavior of this module in this kind of fault occurrence. Three test cases were used to the fault injection. One case was to verify the behavior of the FEHM outputs when a transient fault occurs in some place where generates a error in the inputs of the module. Two cases were to investigate what happens when a transient fault occurs inside of the module. The cases with theirs configurations are listed below:

- First case: fault only in the input signals of the FEHM and all the input channels with the same value;
- Second case: fault in all the module internal signals and all the input channels with the same value;
- Third case: fault only in the voter signals and one channel always different among the others.

The third case is important, since the voter is not monitored or has redundancy. Faults that happens direct in the voter signals, when it generates some error, will not be detected. In addition, the errors generated in the voter causes a incorrect value in the module *out* signal.

The next section presents the results to each test case, and the interpretations to each output signals combination as well.

5.4 Results

The interpretation to the signals *eout* and *tout* is described in Table 5.1.

tout[n]	eout[n]	Error detected	Diagnosis
1	0	yes	Transient fault
1	1	yes	Transient fault
0	0	no	No error
0	1	yes	Undiagnosed

Table 5.1: *eout* and *tout* interpretation.

The interpretation in Table 5.1 is based on the fact that only transient faults were injected in the FEHM. In addition, inside of the module there is no memory elements that are not the elements necessary to implement the sensors. So, always that an error is detected in this simulation , means that this error was caused by a transient fault occurrence. Every time that *eout* signal indicates that a error has occurred, the signal *tout* must indicate that this error was caused by a transient fault. For this reason, the fourth line of the Table 5.1 has an *Undiagnosed* in the Diagnosis column, because was not possible to the sensors indicate whether the error was caused by a transient fault.

The first and second line of the table 5.1 show us that independent of the *eout* value, when the *tout* is equal to 1, means that an error was caused by a transient fault. The third line of the table means that there are faults which were masked logically, electrically, or in time.

Both sensors were projected to diagnose generated errors from transient faults with width less than 0.614 ns. In addition, the clock period to the simulation is equal to 2.256 ns. Therefore, the T_W to both sensors is equal to 27.21% of clock period. Consequently, errors that were generated by transient faults with width less than T_W (27.21 %) should to be diagnosed as a result of a transient fault.

The *out* signal was analyzed according to the voting algorithm that was implemented in Verilog. The analysis was made by the description in Table 5.2.

Input channels relationship			out value	Interpretation
$in1 = in2$	$in1 = in3$	$in2 = in3$	$out = in1$	ok
$in1 = in2$	$in1 = in3$	$in2 = in3$	$out \neq in1$	error
$in1 \neq in2$	$in1 \neq in3$	$in2 = in3$	$out = in2$	ok
$in1 \neq in2$	$in1 \neq in3$	$in2 = in3$	$out \neq in2$	error
$in1 \neq in2$	$in1 = in3$	$in2 \neq in3$	$out = in1$	ok
$in1 \neq in2$	$in1 = in3$	$in2 \neq in3$	$out \neq in1$	error
$in1 = in2$	$in1 \neq in3$	$in2 \neq in3$	$out = in1$	ok
$in1 = in2$	$in1 \neq in3$	$in2 \neq in3$	$out \neq in1$	error
$in1 \neq in2$	$in1 \neq in3$	$in2 \neq in3$	$out = 0$	ok
$in1 \neq in2$	$in1 \neq in3$	$in2 \neq in3$	$out \neq 0$	error

Table 5.2: *out* signal analysis.

The original values of in1, in2, and in3 were stored in auxiliary variables before the fault injection in these channels. The stored values were worth to obtain the *out* expected value and compare it with the simulation result value. When both are different, means that an error has occurred.

Theoretically, when the FEHM is working in the TMR mode, the *out* is guaranteed to be correct. However, in the practical results, it was verified that in some cases up to 17% of the injected faults, errors were generated in the *out* signal with the module working in TMR.

First case

According with the interpretations that was gave to *tout* and *eout* signals, we can distribute the analyzed data into three groups: *Transient fault*, *No error*, and *Undiagnosed*. With this in mind, the fault injection result to the different widths is presented in Table 5.3 to the FEHM implemented with ED, and Table 5.4 to the FEHM implemented with DSTB. The charts in Figure 5.11 and in Figure 5.12 also contains the results to the first case.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
No error (%)	76.91	69.66	64.55	61.42	60.47
Error detected: Transient fault (%)	23.09	30.34	34.33	26.79	24.63
Error detected: Undiagnosed (%)	0	0	1.12	11.79	14.9

Table 5.3: Faults in the inputs of the FEHM (ED): *tout* and *eout* signals.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
No error (%)	76.61	69.15	64.71	61.38	60.19
Error detected: Transient fault (%)	23.39	30.85	33.86	26.62	24.51
Error detected: Undiagnosed (%)	0	0	1.43	12	15.3

Table 5.4: Faults in the inputs of the FEHM (DSTB): *tout* and *eout* signals.

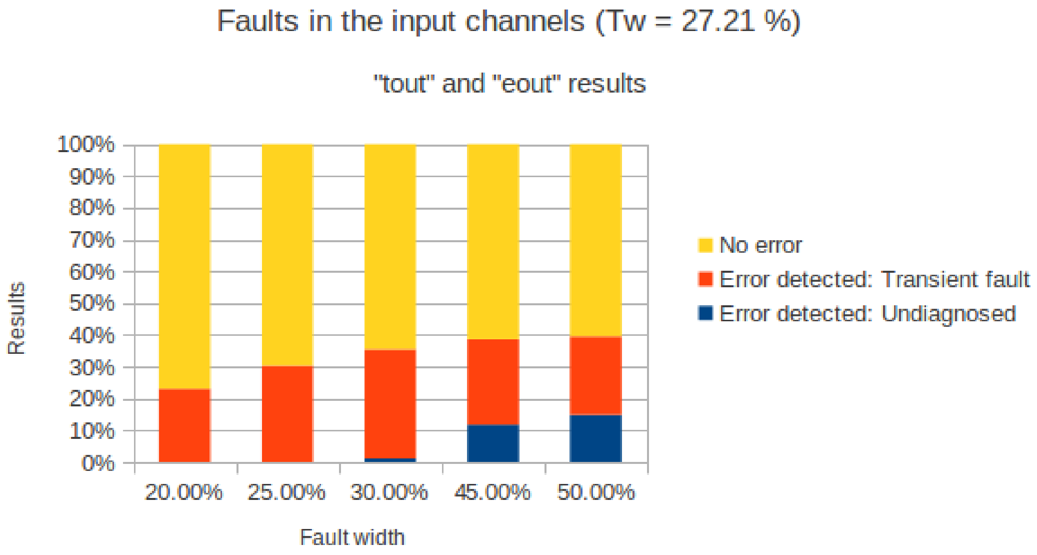


Figure 5.11: Faults in the inputs of the FEHM (ED): *tout* and *eout* signals.

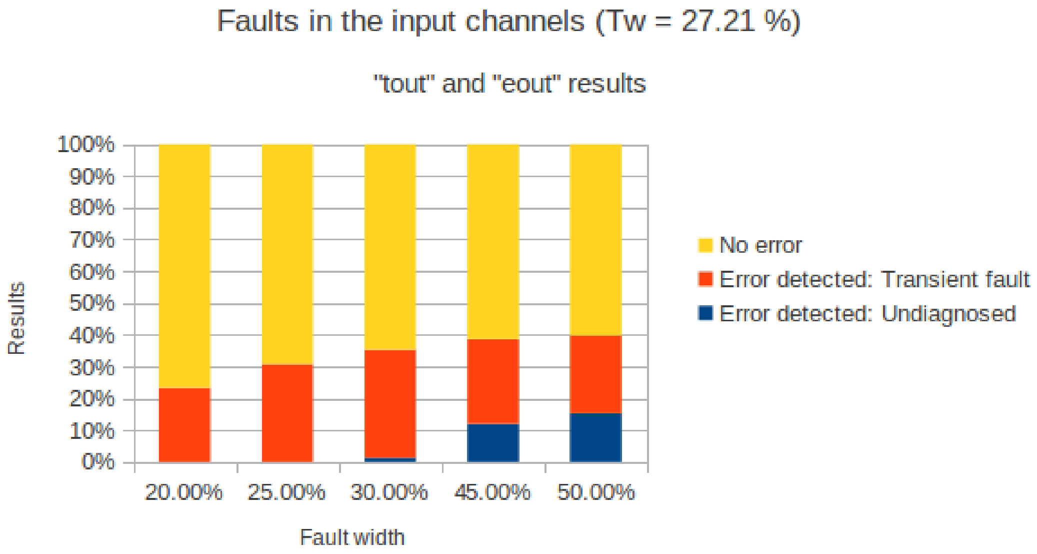


Figure 5.12: Faults in the inputs of the FEHM (DSTB): *tout* and *eout* signals.

It is possible to notice that in both charts (Figure 5.11 and Figure 5.12) all the errors which were caused by faults of width equal to 20% and 25% of the clock period, the occurrence of *Undiagnosed* errors is null. However, to 30%, 45%, and 50% width the *Undiagnosed* errors occurrence increase as the width of the fault increases. A migration from *Transient fault* group to the *Undiagnosed* is also perceptible.

The results to the *out* signal are in the Table 5.5 and Table 5.6. In addition, they are presented in the charts of the Figure 5.13 and the Figure 5.14.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
Ok (%)	96.22	96.47	96.62	96.98	97.06
Error (%)	3.78	3.53	3.38	3.02	2.94

Table 5.5: Faults in the input of the FEHM (ED): *out*.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
Ok (%)	96.22	96.6	96.72	97.09	97.11
Error (%)	3.78	3.4	3.28	2.91	2.89

Table 5.6: Faults in the input of the FEHM (DSTB): *out*.

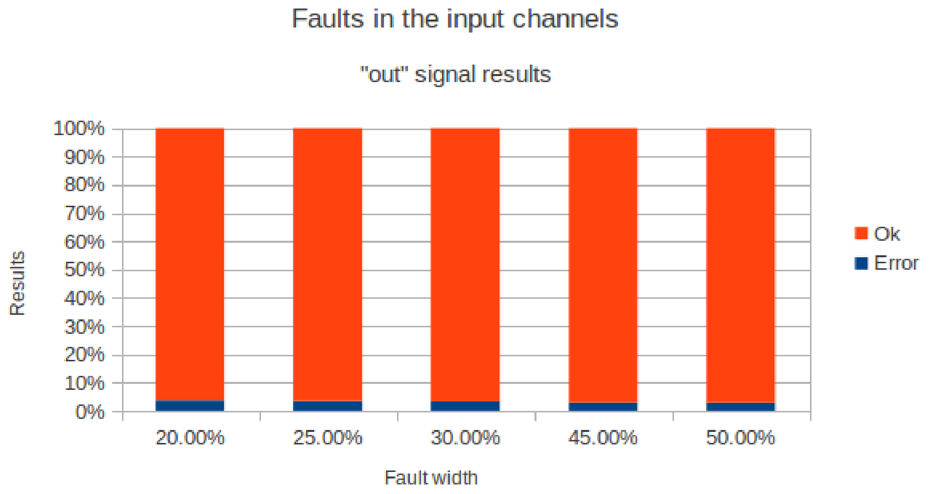


Figure 5.13: Faults in the input of the FEHM (ED): *out*.

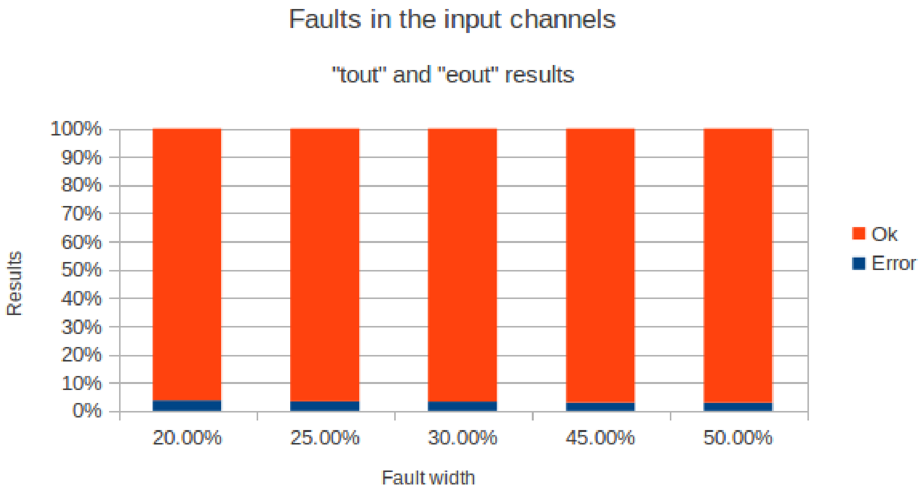


Figure 5.14: Faults in the input of the FEHM (DSTB): *out*.

The number of errors in the *out* signal decreases as the fault width increases. As only the input channels were the fault injection target in this case, a fault can generate a error as the follow: the selected channel by the multiplexer has its value changed by a transient fault at some few picoseconds before the rise edge of the clock (the check moment to the *out* signal), so, there is no enough time to the module detect the error and select a channel that has a correct value.

Second case

The second case was the fault injection in all the internal signals of the FEHM. This case is important to verify the robustness of the module in the transient fault occurrence. The results are shown in the Table 5.7 and Table 5.8, and are represented in the charts of the figures 5.15 and 5.18.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
No error (%)	97.9	97.61	97.03	95.15	94.27
Error detected: Transient fault (%)	2.1	2.39	2.9	4.29	4.88
Error detected: Undiagnosed (%)	0	0	0.07	0.54	0.85

Table 5.7: Fault in the internal signals of FEHM (ED): *eout* and *tout* signals.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
No error (%)	97.85	97.31	96.61	94.63	94.08
Error detected: Transient fault (%)	2.15	2.69	3.26	4.65	5.1
Error detected: Undiagnosed (%)	0	0	0.13	0.72	0.82

Table 5.8: Fault in the internal signals of FEHM (DSTB): *eout* and *tout* signals.

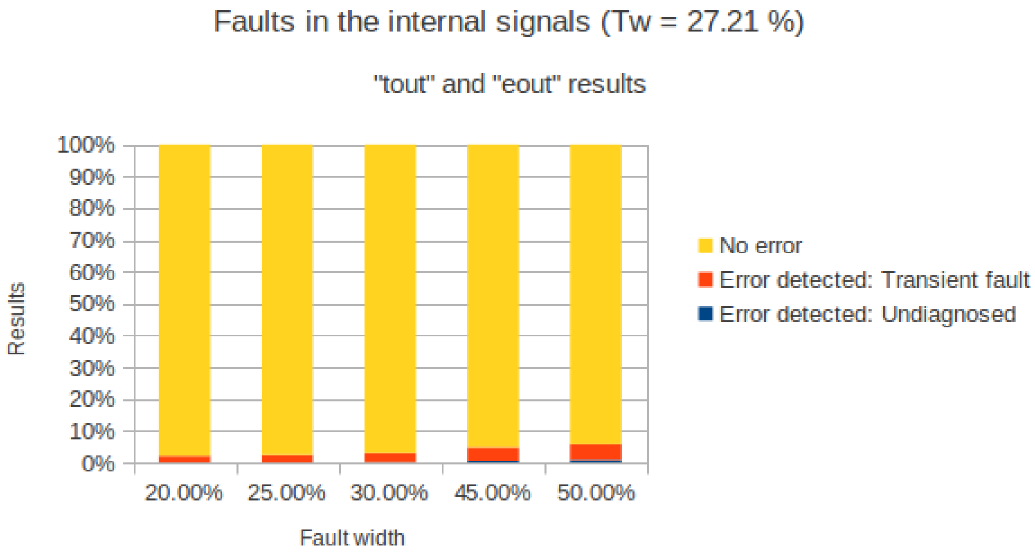


Figure 5.15: Fault in the internal signals of FEHM (ED): *eout* and *tout* signals.

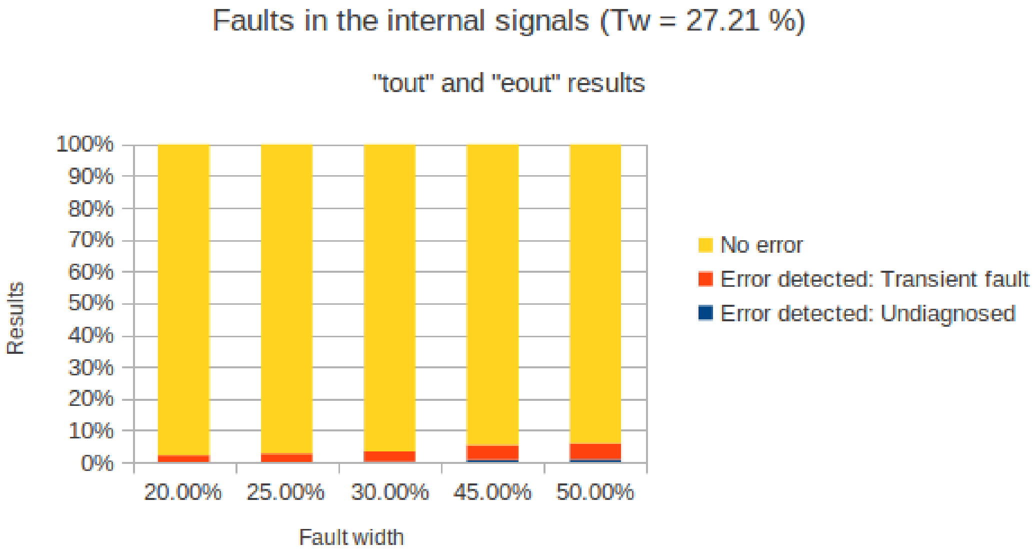


Figure 5.16: Fault in the internal signals of FEHM (DSTB): *eout* and *tout* signals.

The charts show us that as the fault width increases, the number of *Transient faults* generated errors and *Undiagnosed* errors increase. To a fault width of the 50% of clock period, the occurrence of diagnosed errors is about 5% of the faults injected. The incidence of undiagnosed errors correspond to about 0.8%. In addition, compared with the first case, the number of errors that were caused by faults inside of the FEHM are less than the number of errors that were caused by faults in the module inputs.

Errors in the *out* signal, in this case, happens when the fault occurs in the signals that are referent to the multiplexer and to the voter. The transient fault can directly change the values of the signal *out* with an upset in some internal signal of the multiplexer. Or the fault can change the voter to select the standard output ($out = 000000000$) to when all the three input channels have different values among each other. The result to the *out* signal is in tables 5.9 and 5.10, and in the chart of figures 5.17 and 5.18.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
Ok (%)	96.7	96.07	95.25	93.42	92.78
Error (%)	3.3	3.93	4.75	6.58	7.22

Table 5.9: Faults in the internal signals of FEHM (ED): *out* signal.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
Ok (%)	96.32	95.56	94.6	92.69	92.18
Error (%)	3.68	4.44	5.4	7.31	7.82

Table 5.10: Faults in the internal signals of FEHM (DSTB): *out* signal.

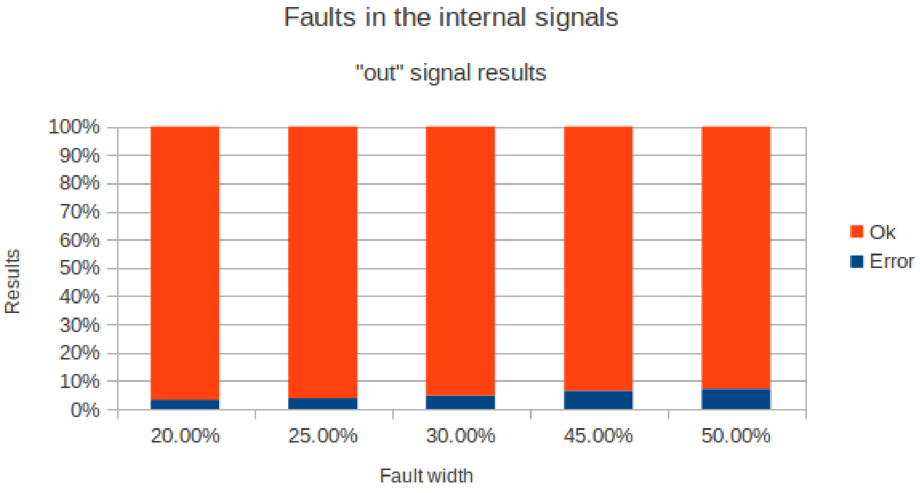


Figure 5.17: Faults in the internal signals of FEHM (ED): *out* signal.

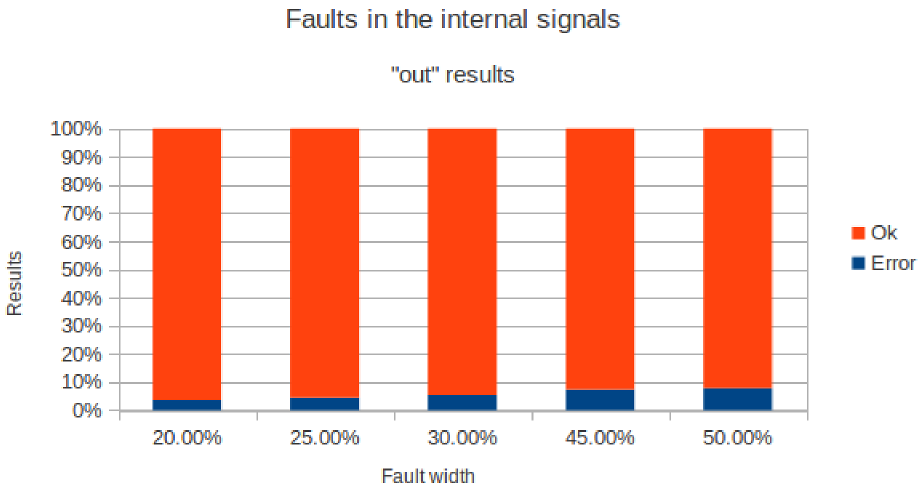


Figure 5.18: Faults in the internal signals of FEHM (DSTB): *out* signal.

Third case

Only the FEHM voter was the target of fault injection in this case. In addition, there was always a input channel with a different value among the others. As it is possible to notice, the sensors does not monitor the voter signals, thus, the injected faults will not be detected by them. This fact is observed in the chart of the figures 5.19 and 5.20, and in the tables 5.11 and 5.12.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
No error (%)	100	100	100	100	100
Error detected: Transient fault (%)	0	0	0	0	0
Error detected: Undiagnosed (%)	0	0	0	0	0

Table 5.11: Faults in the voter of FEHM (ED): *eout* and *tout* signals.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
No error (%)	100	100	100	100	100
Error detected: Transient fault (%)	0	0	0	0	0
Error detected: Undiagnosed (%)	0	0	0	0	0

Table 5.12: Faults in the voter of FEHM (DSTB): *eout* and *tout* signals.

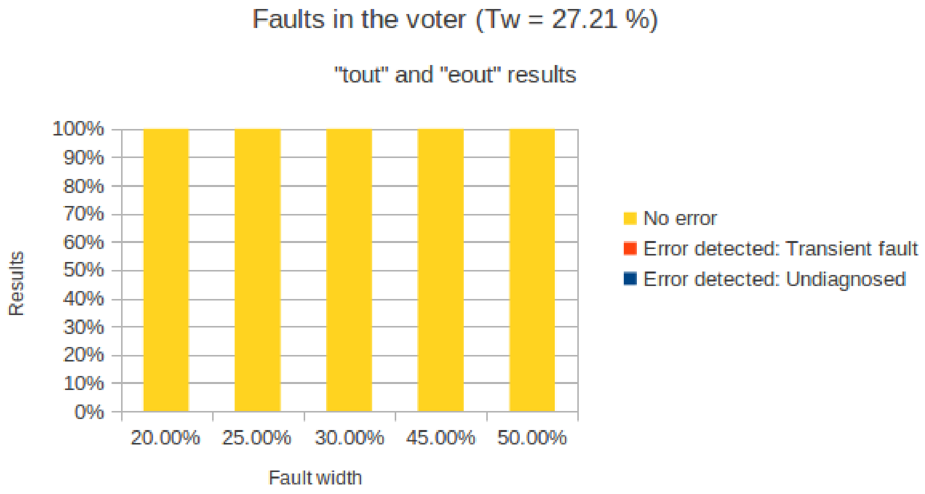


Figure 5.19: Faults in the voter of FEHM (ED): *eout* and *tout* signals.

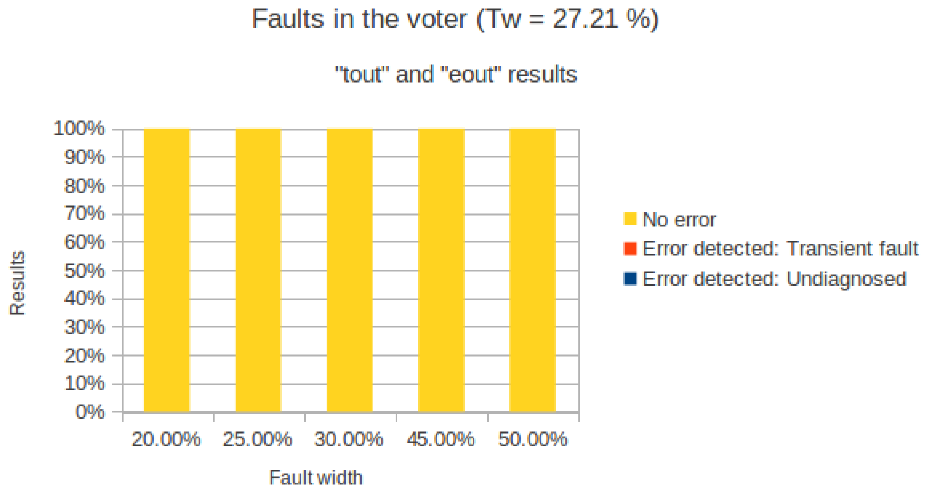


Figure 5.20: Faults in the voter of FEHM (DSTB): *eout* and *tout* signals.

The charts of figures 5.21 and 5.22, and the results in the tables 5.13 and 5.14, show us that to a fault width of the 50%, about 17% of the faults that were injected in the voter generate an error in the *out* signal. The error happens because the fault forces the voter to select the input channel that has the wrong value.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
Ok (%)	91.08	89.72	88.1	84.12	82.78
Error (%)	8.92	10.28	11.9	15.88	17.22

Table 5.13: Faults in the voter of FEHM (ED): *out* signal.

Fault Width (% of the clock period)	20 %	25 %	30 %	45 %	50 %
Ok (%)	91.51	90.03	88.54	84.39	83.12
Error (%)	8.49	9.97	11.46	15.61	16.88

Table 5.14: Faults in the voter of FEHM (DSTB): *out* signal.

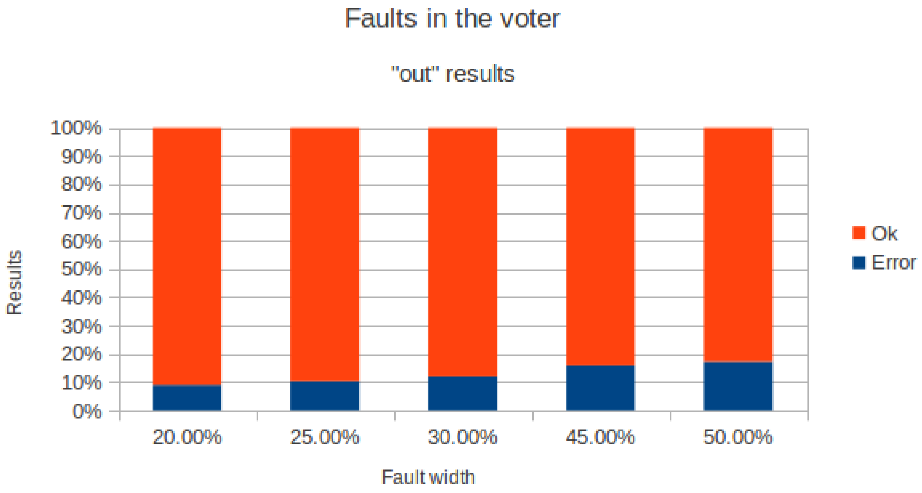


Figure 5.21: Faults in the voter of FEHM (ED): *out* signal.

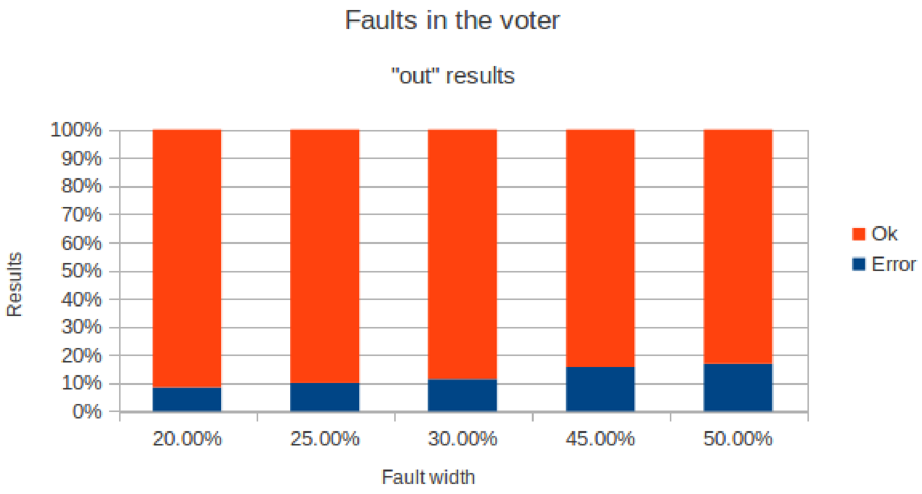


Figure 5.22: Faults in the voter of FEHM (DSTB): *out* signal.

6 CONCLUSION

The circuit level time-redundancy technique proved to be good to the design requisitions of the TEDC, since it implementation results in a sensor that can be attached in a node without impact in the design of others components.

As it was seen in chapter 3.1, the sensors design that were chosen to be implemented have different construction conditions. The conditions that were pointed out should to be respected to maximize the transient error detection window (T_W) of the sensors without false errors signal detection.

The DSTB design, in a first moment, has a little impact in a circuit area and performance. However, it is true only if the minimum path delay of the node where it is attached is out of the level period of the clock in which the latch is sensitive. The level of the duty cycle on this work.

In addition, in chapter 4, where the sensors where implemented using the values of the parameters conditions to the FEHM-Cluster, the inclusion impact of the DSTB was bigger than the impact of the ED. It is seen in the section 4.5 that the synthesis results of area has reached worst values to the DSTB than to ED. The area was impacted by the size of the buffers of the sensors that should generate the delay δ to each sensor design. Furthermore, it was also demonstrated that the best performance of the FEHM-Cluster should decrease 44.9 % when the DSTB sensor is used to construct the TEDC.

The FEHM has obtained approximate diagnose capability with both sensors. The TEDC implemented with the DSTB has detected up to 0.51 % more errors than when it was tested with the ED sensor. However, the difference is less than 1 % and faults were injected randomly. Therefore, it can be said that both options took the same error detection rate.

If we analyze the synthesis and fault injection results, it is possible to notice that between both sensors design, using the Nangate FreePDK45 Generic Open Cell Library to the technological mapping, and using the FEHM-Cluster as the target to the sensors project, the ED sensor is the best choice to the construction of the TEDC block. It sensor obtained less impact in the area of the FEHM. In addition, the FEHM-Cluster performance is minimally decreased by it, because the delay element δ to it sensor is in the clock line of the Shadow Flip-Flop, not in the input line.

As a last consideration, even the FEHM working in TMR mode, the signal *out* will be affected by transient errors that occurs in the circuit. It is also proved in the fault injection that was done in chapter 5. To all the test cases, the signal *out* presented some alteration on it value. The explanation to this fact is that as the multiplexer as the voter of the FEHM are not protected against transient errors. Therefore, to turns the module more reliable, it is necessary to harden these components (voter and multiplexer) against transient errors.

REFERENCES

ALVES, N. State-of-the-Art Techniques for Detecting Transient Errors in Electrical Circuits. **Potentials, IEEE**, [S.l.], v.30, n.3, p.30–35, 2011.

ANGHEL, L.; NICOLAIDIS, M. Cost reduction and evaluation of a temporary faults detecting technique. **Proceedings of the Design, Automation and Test in Europe Conference (DATE)**, [S.l.], March 2000.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. **Dependable and Secure Computing, IEEE Transactions on**, [S.l.], v.1, n.1, p.11–33, 2004.

BOWMAN, K. et al. Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance. **Solid-State Circuits, IEEE Journal of**, [S.l.], v.44, n.1, p.49–63, 2009.

DE SUTTER, B.; RAGHAVAN, P.; LAMBRECHTS, A. **Handbook of Signal Processing Systems**. 2.ed. [S.l.]: Springer, 2013. p.553–592. ISBN: 978-1-4614-6858-5.

EISENHARDT, S. et al. Spatial and Temporal Data Path Remapping for Fault-Tolerant Coarse-Grained Reconfigurable Architectures. In: DEFECT AND FAULT TOLERANCE IN VLSI AND NANOTECHNOLOGY SYSTEMS (DFT), 2011 IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2011. p.382–388.

GRAPHICS, M. **Modelsim SE User's Manual**. [S.l.]: Mentor Graphics, 2002.

HARTENSTEIN, R. Coarse grain reconfigurable architectures. In: DESIGN AUTOMATION CONFERENCE, 2001. PROCEEDINGS OF THE ASP-DAC 2001. ASIA AND SOUTH PACIFIC. **Anais...** [S.l.: s.n.], 2001. p.564–569.

JOHNSON, B. **The Electrical Engineering Handbook**. [S.l.]: CRC Press LLC, 2000.

KASTENSMIDT, F. L. SEE Mitigation Strategies for Digital Circuit Design Applicable to ASIC and FPGAs. In: IEEE NSREC, SHORT COURSE, 2007. **Anais...** [S.l.: s.n.], 2007.

KüHN, J. M. et al. Improving System Reliability using Dynamic Functional Verification on CGRAs. In: INTERNATIONAL WORKSHOP ON HIGHLY-EFFICIENT

ACCELERATORS AND RECONFIGURABLE TECHNOLOGIES (HEART), Okinawa, Japan. **Proceedings...** [S.l.: s.n.], 2012.

MOTOMURA, M. A dynamically reconfigurable processor architecture. **Microprocessor Forum**, [S.l.], 2002.

NICOLAIDIS, M. Time redundancy based soft-error tolerance to rescue nanometer technologies. In: VLSI TEST SYMPOSIUM, 1999. PROCEEDINGS. 17TH IEEE. **Anais...** [S.l.: s.n.], 1999. p.86–94.

NICOLAIDIS, M. Design techniques for soft-error mitigation. In: IC DESIGN AND TECHNOLOGY (ICICDT), 2010 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.208–214.

NICOLAIDIS, M. **Soft Errors in Modern Electronic Systems**. [S.l.]: Springer, 2011.

OPPOLD, T. et al. Cost Functions for the Design of Dynamically Reconfigurable Processor Architectures. In: WORKSHOP ON SYNTHESIS AND SYSTEM INTEGRATION OF MIXED INFORMATION TECHNOLOGIES (SASIMI), Kanazawa, Japan. **Anais...** [S.l.: s.n.], 2004.

OPPOLD, T. et al. CRC - Concepts and Evaluation of Processor-Like Reconfigurable Architectures (CRC - Konzepte und Bewertung prozessorartig rekonfigurierbarer Architekturen). **it - Information Technology**, [S.l.], v.49, n.3, p.157–, 2007.

SCHWEIZER, T. et al. Using Run-Time Reconfiguration to Implement Fault-Tolerant Coarse Grained Reconfigurable Architectures. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS PHD FORUM (IPDPSW), 2012 IEEE 26TH INTERNATIONAL. **Anais...** [S.l.: s.n.], 2012. p.320–327.

SI2. **NanGate FreePDK45 Generic Open Cell Library.** [S.l.]: Si2 Silicon Integration Initiative, 2011., 2012.

SYNOPSYS. **Static Timing Verification of Custom Blocks Using Synopsys' NanoTime Tool**. [S.l.]: Synopsys, 2009.

SYNOPSYS. **NanoSim User Guide**. [S.l.: s.n.], 2010. v.Synopsys.

ANNEX A GRADUATION WORK I

SET detection technique to incorporate in the Configurable Reconfigurable Core architecture

Luigi V. Ferreira[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

lvferreira@inf.ufrgs.br

Abstract. *Fault tolerance is a important issue to worry about in the computing world. The SET detection, among others transient errors, in a determined region of a computer architecture is necessary to increase the reliability of the architecture. By this reason, in this work is defined a transient error detection technique to construct a component which will be able to detect the SET occurrence in the Configurable Reconfigurable Core architecture [Oppold et al. 2007]. Classical and state-of-the-art techniques are presented and analized to find a technique which has the implementation concepts that fit in the SET detector requiriments. As a result, a classical technique is defined to construct the detector component, since its implementation implies in a simple sensor that does not entails changes in others component of the architecture. The result of the work seems to be a good design solution to increase the Configurable Reconfigurable Core reliability.*

Resumo. *Tolerância a falhas é uma questão importante para se preocupar no mundo da computação. A detecção de SET, entre outros errors transientes, em uma determinada região de uma arquitetura de computador é necessária para aumentar confiança (reliability, em inglês) da arquitetura. Por esta razão, uma técnica de detecção de erro transiente é definida neste trabalho com o objetivo de construir um componente que será capaz de detectar a ocorrência de SET na arquitetura Configurable Reconfigurable Core [Oppold et al. 2007]. Técnicas clássicas e no estado-da-arte são apresentadas e analizadas com o objetivo de encontrar uma técnica daqual seus conceitos de implementação se enquadrem nos requerimentos do detector de SET. Como resultado, uma técnica clássica é definida para construir o component de detecção, desde que sua implementação implica em um simples sensor que não acarreta mudanças nos outros componentes da arquitetura. O resultado do trabalho parece ser uma boa solução de design para aumentar a confiança do Configurable Reconfigurable Core.*

1. Introduction

The idea of creating computing systems with flexible hardware dates back to the 1960s, it was the emergence of the SRAM-based field-programmable gate array (FPGA) in the 1980s that boosted Reconfigurable Computing as a research and engineering field [Teich 2009]. This flexible hardware is known as reconfigurable architectures. These architectures are able to adapt their hardware to application demands and serve broad and relevant application domains from embedded to high-performance computing like telecommunication and networking among others.

FPGA is a fine-grained reconfigurable architecture, since it is reconfigurable at bit level [Ferreira et al. 2011]. It consists of an array of logic blocks and routing channels. Each logic block is composed by, at least, a Look-Up Table (LUT) and a flip-flop. The configuration of this device is made through a bitstream file that has the configuration signals to the logic blocks and the routing channels. Another concept is the coarse-grained reconfigurable architectures (CGRA) that are reconfigurable at word level - 16 bits, 32 bits, etc. - [Ferreira et al. 2011]. A major benefit of using word level reconfiguration is the massive reduction of configuration memory and configuration time, as well as drastic complexity reduction of the P&R (placement and routing) problem [Hartenstein 2001]. Some CGRA can be reconfigured during a single clock cycle.

In contrast to FPGAs, CGRAs are designed specific to application(s) with dedicated functional units, application-specific bit-width and interconnects [Rakossy et al. 2012]. The CGRA functional units are inside of blocks that are called as processing elements (PE). Beyond of the FUs, the PEs consist in a set of registers and a configurable memory cache at least. The registers can keep the FU result and the FU operands. The configurable memory cache keeps the configuration context of the PE, that means, the signals that indicate the FU operation, the source of the FU operands, and the FU result destination. The CGRAs are also known as processor-like reconfigurable architectures.

On the Adaptive Reliability for Embedded Systems project (ARES) of the Eberhard Karls University of Tübingen was developed a modified architecture model that is a coarse-grained reconfigurable architecture. This architecture is the Configurable Reconfigurable Core (CRC) that has the focus of use a fast reconfiguration to optimize area, performance and power. It also represents a range of processor-like architectures [Oppold et al. 2007].

The CRC consists of a Processor Element (PE) rectangular array connected through a reconfigurable network(Figure 1). Each PE, Figure 2, is composed of a Functional Unit (FU), which executes arithmetic and logical operations, a registers set, a context memory, which holds all the configuration contexts, and a finity state machine (FSM) that changes the context every clock cycle. A PE context determines the FU operation, the FU operand sources (ports or registers), the FU result destination (ports or registers), and the route through input and output ports for the CRC. The context memory and the FSM must be configured using an external source. Hence, each PE has a module that is represented in Figure 2 as “external reconfiguration”.

The CRC, as it is presented in Figure 1, has not capacity to detect or mask transient errors, so, a modification in the architecture design was proposed to increase the CRC reliability. This modification is the insertation of a new module in the architecture that is able to detect and mask transient errors that occur in the PE. Inside of this new module, there is a component which must detect the occurrence of SET in the mechanism that detect errors in the data. The focus of this work is to determine the technique to construct the SET detection component that causes less impact on the architecture. For this purpose, this article is organized as follows: section 2 presents information about what can generate a transient error and what are their consequences; section 3 presents the module that enables the architecture to detect and mask transient errors; section 4 describes some techniques to detect transient errors; section 5 presents the choosed technique; section 6,

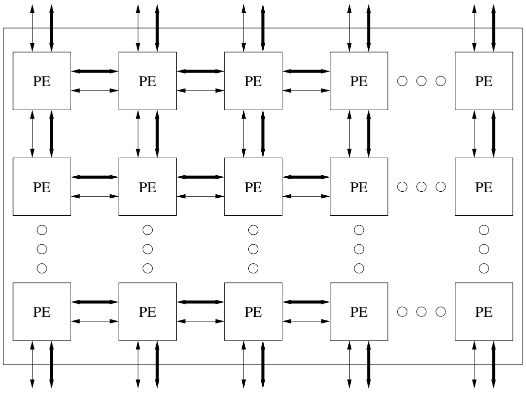


Figure 1. PE array.

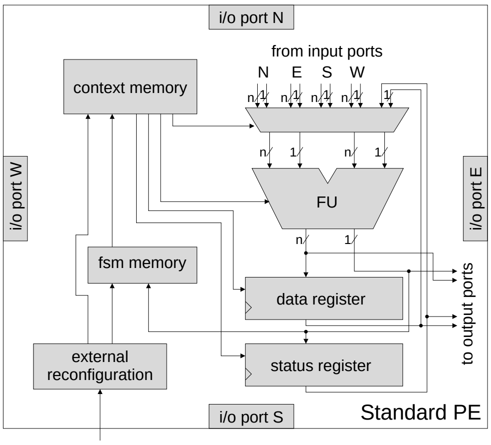


Figure 2. PE components.

some existing implemented examples of the choosed technique are presented; section 7 presents what will be done in the next work; conclusions are drawn in section 8.

2. Transient Errors

Transient errors are also known as soft-errors. Soft-errors are errors that do not cause any permanent danification to the devices, just some pertubations in the hardware such as transient voltage pulses and bitflips. These pertubations can flip some logic value stored in a memory element in the system.

Soft-erros are caused by radiation. Neutron particles that are generated by cosmic radiation interact with the Earth atmosphere and can cause some pertubations in digital systems. Alpha particles emitted by radioactive impurities present in low concentration in chip package materials are another source of soft-errors.

When a particle strike causes a bit-flip (upset) in a memory cell or a latch, we consider that a Single-bit upset (SBU) has occured. However, when two or more memory cells or latches suffer a bit-flip, it is a Multiple-cell upset (MCU). Whether the event causes the upset of two or more bits in the same word, we have a Multiple-bit upset (MBU). A Single-event transient (SET) occurs if the strike of the particle causes a voltage glitch in the circuit, and it becomes a bit error when captured in a storage element.

The pertubation of control registers, clock signals, and reset signals that causes loss of functionality is called Single-event functional interrupt (SEFI). But, when the event creates an abnormal high-current state by triggering a parasitic dual bipolar circuit, which requires a power reset, it is a Single-event latchup (SEL). This last one can possibly cause permanent damage to the device, in which case the result is a hard error [Nicolaidis 2011].

In the literature we can find the term SEU (Single-event upset) as a references to soft-error, but this term could mean a reference to SBU and MBU together [Nicolaidis 2011].

3. Flexible Error Handling Module

A new module was developed to enable the CRC architecture to detect and mask faults. This module is the Flexible Error Handling Module (FEHM) [Schweizer et al. 2012] that is based on Triple Modular Redundancy (TMR), since it monitors three input channels. Each input channel has an enable signal. Consequently, if one channel is disabled, the FEHM will work as a Duplicantion With Comparation (DWC). However, in the DWC operation mode, only error detection is possible.

As it is shown in Figure 3, the FEHM provides three outputs: 3 data error signals, 3 transition error signals, and 1 output channel, where the data is presumably right when the module is working in TMR mode.

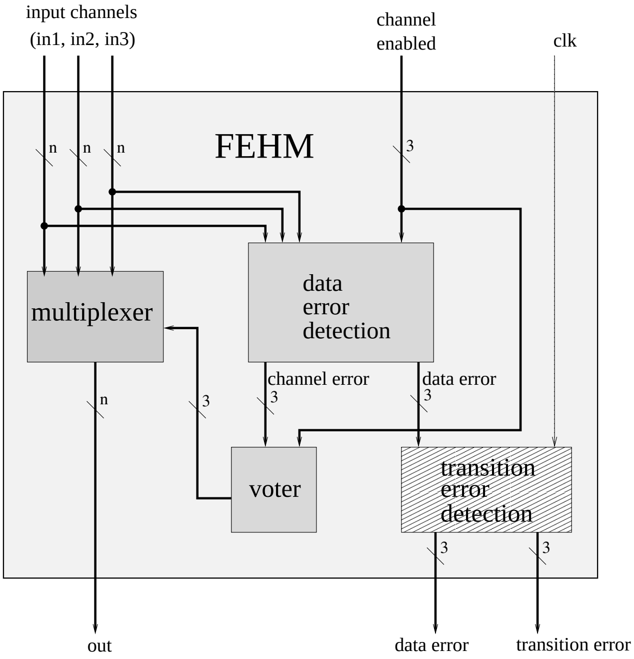


Figure 3. FEHM components

The data error signals are provided by the Data Error Detection Component (DEDC). It compares each pair of input channel of the module. Inside the pairs, the input channels are compared to each other with a bitwise XOR operation. This operation is reduced to one bit per channel with a XOR tree. This one bit per channel (channel error) indicates if one channel has a different value of the others.

The Transition Error Detection Component (TEDC) monitors the data error signals (Figure 3) provided by DEDC, and signalizes when a SET occurs in the DEDC. The implementation of the TEDC is the focus of this work.

The inclusion of the FEHM in the architecture is made using two possible approaches: TripleFU or FEHM-Cluster. TripleFU, Figure 4, implements traditional TMR, since two more FUs are inserted inside a PE and each FU output is directed to the FEHM inputs. This approach considerably increases the PE area. FEHM-Cluster, Figure 5, monitors the FU of three different PEs and the FEHM is embedded in one of the PEs(Center-PE). The Side-PEs have a dedicated route to carry the result of theirs FU straight to FEHM [Schweizer et al. 2012].

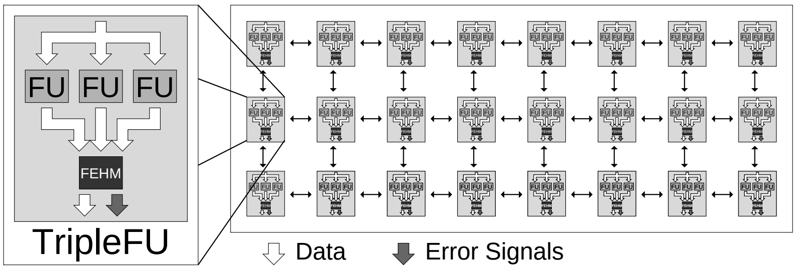


Figure 4. TripleFU approach

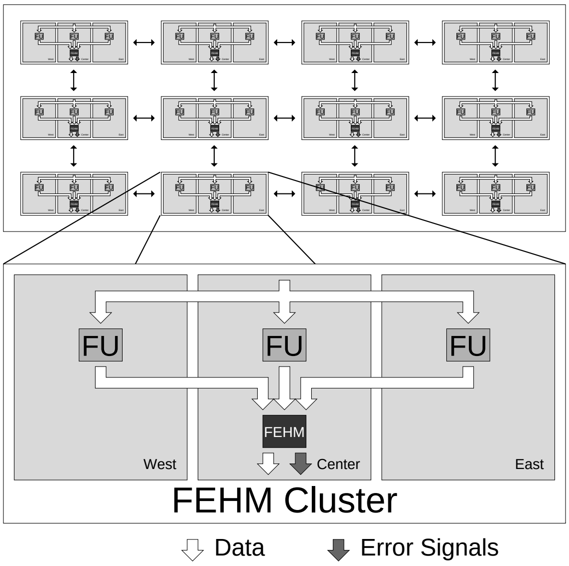


Figure 5. FEHM-Cluster approach

4. Transient error detection techniques

Error Correction Codes (ECC), Redundancy in time and space, Error detection with logic synthesis and Logic implications are some techniques to acheive transient errors (or soft-errors) mitigation that are applied in electrical or logic circuits [Alves 2011]. There is also the Hardened Storage cells (SRAM cells, latches, flip-flops) that will preserve their state even if the state of one of their nodes is altered by an ionizing particle strike. Therefore, it is a good way to construct a cell to prevent SEU [Nicolaidis 2010].

Error Correction Codes are good to detect transient errors in memory elements and transmission lines. They can indicate if the data that has been read is the same data that was stored in the memory element by using some paroty recover. They can also indicate if the stored data was altered by a particle strike. ECC can be extended to other logic structures with easily predictable outputs, such as multipliers, adders, and programmable logic arrays (PLAs). However, detecting errors in apparently random logic, without any discernible logic regularity, is a significantly harder problem. In random logic we must determine if the logic operations performed in some circuit inputs are correct [Alves 2011].

Error detection with logic synthesis is a synthesis tweaking aplied in order to minimize the number of potential transient errors, via error masking. Features of logic gates are used to fix the error as the signal propagates downstream. For example, a single alpha particle striking any input line of a two-input logic OR gate will have no effect at that gate output when both inputs are logic ones.

The Logic implications is a new method to detect transient errors. This method takes an existing design and searches all internal circuit nodes for consistent logic patterns amoung them. When an invariant pattern is found, it will append some simple checker hardware that reenforce the validity of the relationship [Alves 2011] .

The techniques such as Error detection with logic synthesis and ECC do not achieves the TEDC implementation requirements, since TED should operate like a SET sensor that monitors the output of the DEDC component. Logic Implications could be a choosed technique, but, it will probably change the components which are already implemented.

Redundancy in space, to monitors the desired signal, implies the duplication of the DEDC component. In this case, the TEDC component would be a comparator. However, it does not fit in the FEHM design, since there must exist just one DEDC per FEHM.

Redundancy in time, as it is explained in [Alves 2011], is the re-execution of the same logic multiple times, while storing any intermediate data in memory, and reportong any output differences throughout the various executions. However, its multiple logic re-execution significantly decreases the throughput of the circuit.

The concept of circuit level time-redundancy [Nicolaidis 2010] is interesting to the TEDC implementation. The data is checked, at least, twice at each clock cycle and the result of each checked data is compared to each other. If the value is the same in all the checking moments, it is assumed as correct. This technique differs from the redundancy in time concept, since there is no logic re-execution to detect the error, just the logic circuit output is read more than once in the same execution with a period of picoseconds between each read time. This technique is a good choice to implement the TEDC component, since it operate as a sensor and it do not have to make changes in the logic circuit that has been monitored.

5. Circuit level time-redundancy

Circuit level time-redundancy is a design solution to SET mitigation. The main idea is that transient faults have a limited duration (just a few hundreds picoseconds - the exactly value depends of circuit features and particle energy- [Nicolaidis 1999]) then apply fine time-grain redundancy(i.e. within the clock cycle). Low hardware cost can be achieved, because one does not need to construct the same logic more than once.

The implementation of this technique result in a SET sensor. Generally, this sensor is put in place of temporal barriers (pipe-lines) as it is shown in Figure 6. The sensor can also be put in the end of critical paths or in parallel with some elements to monitor the data time.

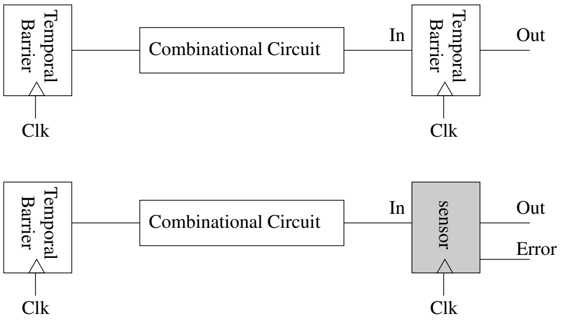


Figure 6. SET sensor in place of a temporal barrier.

Basically, to detect fault, the same data is checked, at least, in two different moments. If the data value is the same in both checking moments, it is assumed that data value as correct. For example, in Figure 7, T_1 and T_2 are the checking moments of the data value and the situation which any error occurs (Figure 7.(a)), the input signal In has the same value in T_1 and T_2. However, in Figure 7.(b), the input signal In was modified by a voltage glitch and this switchs the signal Error from low to high in T_2. In this last one situation, the Error signal still on the high level until the next T_2 moment, where the In signal has the same value in both moments (T_1 and T_2).

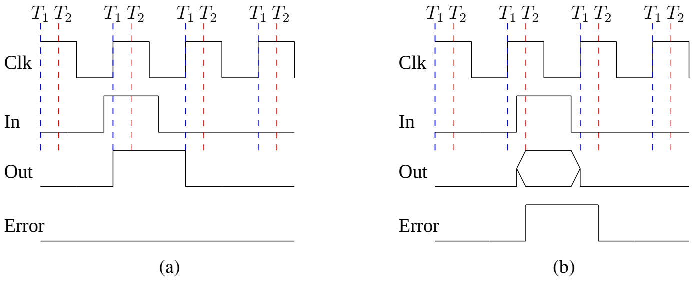


Figure 7. (a) conceptual diagram without transient fault; (b) conceptual diagram with transient fault.

A comprehensive way that shows the design of these sensors is presented in Figure 8. In these block diagrams, the Memory Element and the Redundant Memory Element could be as a flip-flop as a latch, the Comparator block could be a XOR gate, and the delay elements δ_{In} and δ_{Clk} could be implemented with buffers.

In Figure 8.a), the δ_{Clk} is inserted in the Redundant Memory Element clock line to check if the data arrived respecting the Memory Element latch conditions. As it was mentioned, the signal $\delta_{Clk} + Clk$ could be constructed using a buffer element in the “Clk” line. All the same, one can create a second clock line with a difference δ_{Clk} with the “Clk” line. δ_{Clk} was described in some works [Nicolaidis 1999, Anghel and Nicolaidis 2000] using the follows relationships:

$$\begin{cases} \delta_{Clk} = D_{tr} - D_{setup} \\ \delta_{Clk} < D_{min} \end{cases} \tag{1}$$

In (1), D_{tr} is the maximal SET that will be detected. This term is determined by the designer. D_{setup} is the Redundant Memory Element setup time and D_{min} is the Combinational Circuit minimum path delay. In addition, the relation $\delta_{Clk} < D_{min}$ means that all the path times should exceed the δ_{Clk}. It must be obeyed to avoid false error signals. For example, if both Figure 7 waveforms diagrams are from the circuit that is shown in Figure 8.a), δ_{Clk} is equal $T_2 - T_1$ and T_2 must be less than the minimum delay of the data path. Therefore, the designer should choose a D_{tr} that makes δ_{Clk} obey the relation (1). However, there is another relation to δ_{Clk} that maximizes the transient pulse duration which will be detected. This last relation is based on D_{min} and in the Redundant Memory Element time to hold (D_{hold}) as it is presented in (2).

$$\delta_{Clk} = D_{min} - D_{hold} \tag{2}$$

The equation (2) is based on the principle that the data which takes the minimum delay will not be save in the Redundant Memory Element, since it will come after the time to hold.

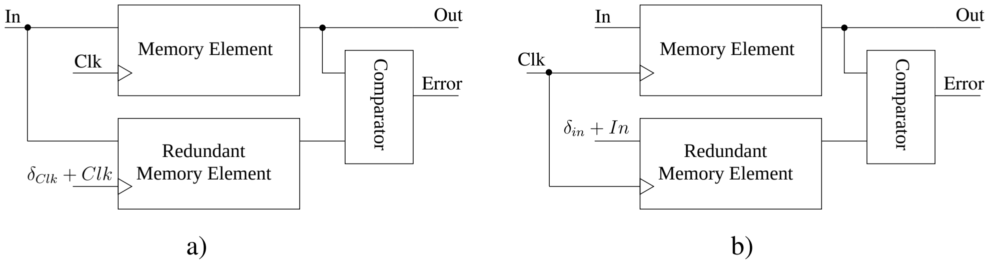


Figure 8. Conceptual design of the SET sensor

In this work, the period that a SET can be detected by a sensor will be referenced as T_W. In the case of the Figure 7, T_W is equal $T_2 - T_1$ (or equal to δ_{Clk} in the latest example).

The circuit in the Figure 8.b) is used to detect when a transient error causes a violation in the projected guardband. It can be seen in Figure 9.

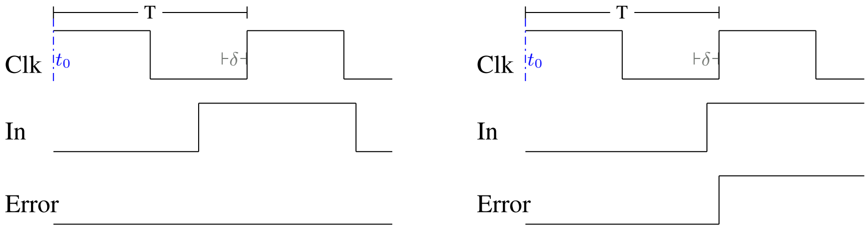


Figure 9. Guardband violation example.

A guardband δ is defined in Figure 9. When the signal In comes between t_0 and t_0+T$-\delta$ (T is the clock period) the guardband will not be violated. On the other hand, if the signal In arrives between t_0+T$-\delta$ and t_0+T, a violation guardband error will appear.

The guardband in Figure 8.b) is defined with the term δ_{In}. The value of δ_{In} must consider the maximum path delay to ensure that false error signals will not occur.

6. Examples

Figure 10 presents one implementation using digital blocks of the Figure 8.a) diagram. One can find this implementation in [Nicolaidis 1999].

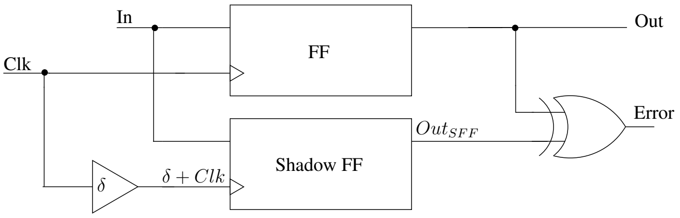


Figure 10. Transient fault detector implemented with two flip-flops

The implementation in Figure 10 uses two flip-flops and will be called in this work as Essential Detector (ED). The “FF” and “Shadow FF” are the “Memory Element” and the “Redundant Memory Element” respectively. One buffer (δ) is used to construct the element δ_{Clk}. A demonstration about the internal signals of the circuit is presented Figure 10. It is supposed that both flip-flops have theirs latch edges in the rise edge of the clock.

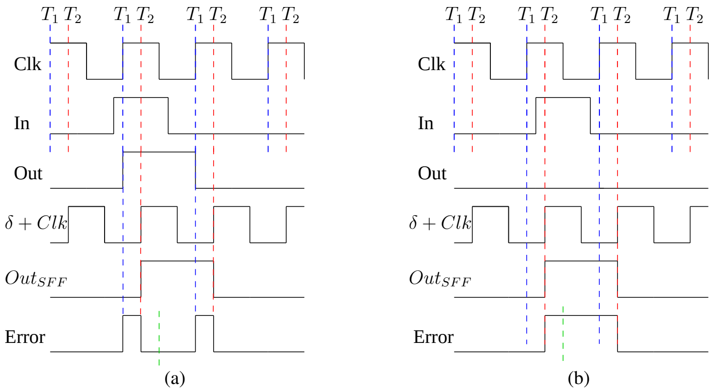


Figure 11. Internal signals to Figure 10

The delay element δ causes the occurrence of the Shadow FF latch edge in T_2. In the situation where any error occurs (Figure 11.(a)), the data is saved inside the two flip-flops. It is seen in the waveforms of the signals Out and Out_{SFF}. However, in the another situation (Figure 11.(b)), the flip-flop FF could not save the new value of the signal In, since it has arrived after the latch edge.

6.1. Razor Flip-Flop

The Razor Flip-Flop (RFF), Figure 12, was proposed to construct a new approach to Dynamic Voltage Scaling (DVS), which is based on dynamic detection and correction of speed path failures in digital designs [Ernst et al. 2004, Ernst et al. 2003].

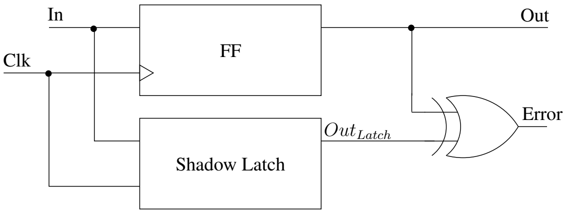


Figure 12. Razor Flip-Flop architecture

The digital design uses a latch (Shadow Latch) to double sample the signal In. If the flip-flop (FF) is sensitive to the rise edge and the lacth (Shadow Latch) is high level sensitive, the T_W of this sensor is all the high level period of the clock signal. The RFF operation is similar to the Double Sample with Time Borrowing sensor and its operation is explained in section 6.2.

6.2. Double Sampling with Time Borrowing

Double sampling with time borrowing (DSTB), Figure 13, is a solution that eliminates the metastability in data path. Now, only in the error path is possible the metastability occurence. By [Bowman et al. 2009], this is a drastic simplification on the sensor metastability handle.

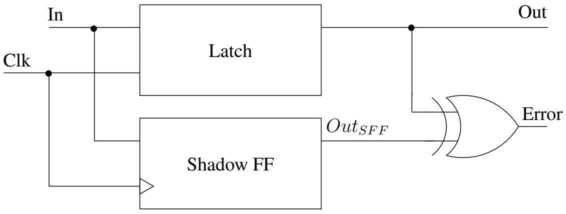


Figure 13. DSTB design

The sensor uses one flip-flop (Shadow FF) to double sample the data. The Memory Element is a latch. It is supposed that the Shadow FF is sensitive to the rise edge of the clock and the latch is high level sensitive. Its T_W is all the high level of the clock signal. This fact is shown in Figure 14.

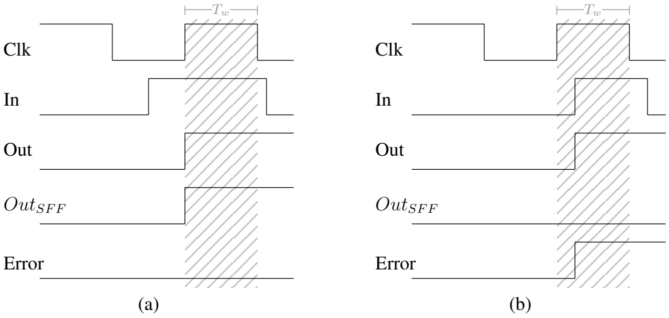


Figure 14. DSTB operation

The minimum path delay must exceeds the duty-cycle period of the clock signal, then, we have the follow relations:

$$T_W = Clk_{duty-cicle} \tag{3}$$

$$T_W < D_{min} \tag{4}$$

In (4) the term D_{min} is the minimum path delay of the combinational circuit that is shown in Figure 6. The relation (4) must be respected to eleminate false error signals. If the delay of the minimum path is to small that is impossible to reduce the duty-cycle of the clock signal, buffers can be inserted in the data path to push the occurrence of the data to after the high level of the clock (In node in Figure 13). Nevertheless, this will affect the clock period, since the delay of the maximum path could exceed the clock period and this could cause a false error signal. Therefore, to the DSTB, the minimum and maximum path delays between temporal barriers must be well known.

6.3. Transition Detector with Time Borrowing

The Transition Detector with Time Borrowing (TDTB), Figure 15, is the SET sensor which has the lowest clock energy [Bowman et al. 2009]. It is constructed using one latch and one transition detector.

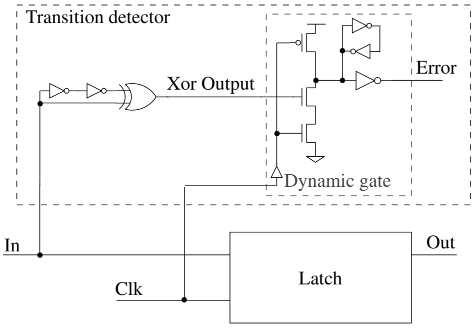


Figure 15. TDTB design

The Transition detector monitors the In node signal during the high level of the clock fase. The operation of the TDTB can be seen in Figure 16.

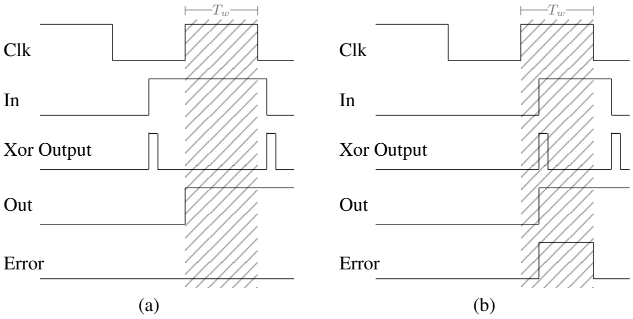


Figure 16. TDTB signals

In the conceptual waveform diagram in Figure 16, the Xor Output node generates small pulses on every In node signal transition. These pulses does not affect the

Dynamic gate output (Error) during the low clock fase (Figure 16.(a)). If the input In signal arrives after the rise clock edge, Clk' is logically high, then the pulse discharges the Dynamic gate in the Error node (Figure 16.(b)). As it is shown in Figure 16, the Error node signal remains logically high just in the high level period of the clock. It happens because Dynamic gate pre-charges when the clock signal switchs from high to low. In [Bowman et al. 2009], a set-dominant latch (SDL) is inserted in the Dynamic gate output to keep the Error signal high during all the low clock fase when a SET occurs.

7. Implementation

The TEDC will be implemented in Verilog and two differents circuit level time-redundancy designs will be constructed. The designs are the DSTB and the ED. Both implementations will be inserted into the FEHM and the designs will be tested in the FEHM-Cluster approach.

Faults will be injected in the FEHM-Cluster and the SET detection results will be observed to take a trade-off between the two TEDC implementations. Area and power cost will be also estimated for both.

8. Conclusion

A good technique to enable the CRC architecture to detect transient errors as SET is the circuit level with time-redundancy. This technique fits in the TEDC design requirements, since it is not necessary to make changes in the architecture components where is desirable to detect the SET occurrence.

As it was pointed out in the sections 5 and 6, to implement the sensors is necessary the knowledge of the maximum and minimum path delays to not take false SET detections.

References

Alves, N. (2011). State-of-the-Art techniques for detecting transient errors in electrical circuits. *Potentials, IEEE*, 30(3):30–35.

Anghel, L. and Nicolaidis, M. (2000). Cost reduction and evaluation of a temporary faults detecting technique. *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*.

Bowman, K., Tschanz, J., Kim, N. S., Lee, J., Wilkerson, C., Lu, S., Karnik, T., and De, V. (2009). Energy-Efficient and Metastability-Immune resilient circuits for dynamic variation tolerance. *Solid-State Circuits, IEEE Journal of*, 44(1):49–63.

Ernst, D., Das, S., Lee, S., Blaauw, D., Austin, T., Mudge, T., Kim, N. S., and Flautner, K. (2004). Razor: circuit-level correction of timing errors for low-power operation. *Micro, IEEE*, 24(6):10–20.

Ernst, D., Kim, N. S., Das, S., Pant, S., Rao, R., Pham, T., Ziesler, C., Blaauw, D., Austin, T., Flautner, K., and Mudge, T. (2003). Razor: a low-power pipeline based on circuit-level timing speculation. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 7–18.

Ferreira, R., Vendramini, J., Mucida, L., Pereira, M., and Carro, L. (2011). An FPGA-based heterogeneous coarse-grained dynamically reconfigurable architecture. In *Compilers, Architectures and Synthesis for Embedded Systems (CASES), 2011 Proceedings of the 14th International Conference on*, pages 195–204.

Hartenstein, R. (2001). Coarse grain reconfigurable architectures. In *Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific*, pages 564–569.

Nicolaidis, M. (1999). Time redundancy based soft-error tolerance to rescue nanometer technologies. In *VLSI Test Symposium, 1999. Proceedings. 17th IEEE*, pages 86–94.

Nicolaidis, M. (2010). Design techniques for soft-error mitigation. In *IC Design and Technology (ICICDT), 2010 IEEE International Conference on*, pages 208–214.

Nicolaidis, M. (2011). *Soft Errors in Modern Electronic Systems*. Springer.

Oppold, T., Schweizer, T., de Oliveira Filho, J. A., Eisenhardt, S., and Rosenstiel, W. (2007). CRC - concepts and evaluation of processor-like reconfigurable architectures (CRC - konzepte und bewertung prozessorartig rekonfigurierbarer architekturen). *it - Information Technology*, 49(3):157–.

Rakossy, Z., Naphade, T., and Chattopadhyay, A. (2012). Design and analysis of layered coarse-grained reconfigurable architecture. In *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, pages 1–6.

Schweizer, T., Kuster, A., Eisenhardt, S., Kuhn, T., and Rosenstiel, W. (2012). Using run-time reconfiguration to implement fault-tolerant coarse grained reconfigurable architectures. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 320–327.

Teich, J. (2009). From dynamic reconfiguration to self-reconfiguration: Invasive algorithms and architectures. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 11–12.