Politecnico Di Torino

Dipartimento di Elettronica e Telecomunicazioni

Corso di Laurea Magistrale in Ingegneria Elettronica

Master Degree's Thesis

Hybrid On-Line Self-Test Architecture for Computational Units on Embedded Processor Cores

Relatore: Edgar Ernesto Sanchez Sanchez

Correlatori: Andrea Floridia, Davide Piumatti

Autore: Gianmarco Mongano

Anno Accademico 2018-2019

Quello che per un uomo è "magia",

per un altro è ingegneria

Robert A. Heinlein

Abstract

Safety-critical applications require reaching high fault coverage figures for on-line testing in order to be compliant with functional safety standards currently in use. In the present days, in order to meet such strict requirements, different solutions are adopted by semiconductor manufactures. The range of applied approaches may vary from pure hardware-based mechanisms to software-based ones. Each of these possible solutions presents both advantages and drawbacks. Typically: software approaches are less intrusive and have the advantage of reduced test application time compared to hardware ones. Conversely, although hardware approaches are normally invasive and have longer test application time, they also yield high defect coverage. This thesis aims to suggest an innovative Design for Test infrastructure, accessible via software, for enabling a high fault coverage on-line test of arithmetic units within embedded processor cores. The end-goal of this alternative design is to overcome limitations of both hardware and software-based test approaches, while striving for a low invasive on-line test. Such architecture was implemented on an open source processor, the OpenRISC 1200 and its effectiveness evaluated by means of exhaustive fault injection campaigns.

Ringraziamenti

Ed eccomi qui a scrivere i ringraziamenti per la mia tesi, avvenimento che, fino a qualche settimana fa, sembrava lontanissimo.

In primis è doveroso, ma soprattutto giusto, ringraziare mio Padre e mia Madre. Senza di loro infatti non potrei essere a questo punto della mia vita per diversi motivi: quelli puramente biologici (del tipo lape ed il fiore), quelli materiali (vitto, alloggio, pagamento delle tasse universitarie) e quelli affettivi. Loro hanno sempre creduto in me. Infatti, nonostante io abbia scelto un percorso impegnativo e pieno di ostacoli, il loro supporto e la loro fiducia non mi sono mai mancati, anzi, spesso, sul buon risultato di alcuni esami loro sembravano addirittura pi sicuri di me. Vi voglio bene. Questo ringraziamento va esteso a tutto il resto dei parenti e degli amici di famiglia, dai quali cui ho sempre ricevuto appoggio e complimenti, cose che non solo fanno sempre piacere, ma spronano anche.

Voglio poi anche ringraziare i vari colleghi e, soprattutto, amici che ho incontrato in questi anni di studio (sia per quanto riguarda il percorso triennale che quello magistrale). In particolar modo, per quanto riguarda questi ultimi due anni ho il piacere di ringraziare il già Dottore Magistrale Andrea ed il Dottore (sicuramente Magistrale nella prossima sessione) Paolo, con i quali ho passato, in questo periodo, tantissimo tempo e che hanno sopportato me e le mie battute di cui secondo loro la maggior parte non faceva ridere (anche se lo so che mentono).

Altre persone da ringraziare sono il professor Sanchez, che mi ha permesso

di svolgere questa tesi e Davide ed Andrea che, oltre a svolgere i loro lavori, hanno sempre avuto molta attenzione nei miei confronti e sono stati sempre molto esaustivi nell'aiutarmi con i miei dubbi e le mie domande sullo svolgimento della tesi. Un grazie va anche a tutto il Laboratorio 3 del DAUIN, dove forse non ho socializzato con tutti allo stesso modo, ma è stato sicuramente un bell'ambiente dove lavorare. Infatti, anche quando ne uscivo, la sera, frustrato perchè il risultato del mio lavoro non aveva dato i suoi frutti, la mattina seguente mi alzavo comunque sereno e felice di ritornarvi.

In queste ultime righe, più distaccate dalle altre, e non perchè riguardino soggetti meno importanti ma, anzi, per dare loro maggior risalto, voglio ringraziare quelle persone che non ci sono più ma che ci sono state. La prima persona è mia nonna Biagia., che finché è stata in vita mi ha riempito di affetto, mi ha visto iniziare luniversit` ma, purtroppo, non è riuscita nemmeno a festeggiare con me la laurea triennale. La seconda persona è Anna. Lei è l'amica di famiglia di una vita, mi ha visto crescere e mi definiva il figlio maschio che non aveva avuto e che, per pochi mesi, non può essere qui a celebrare questo momento. Il terzo e ultimo ringraziamento che mi sento di fare è un po atipico, e va a mio cugino Renzo, che non ho mai conosciuto, ma a cui mi son sempre sentito legato in qualche modo e sono sicuro che se quell'incidente non fosse mai avvenuto oggi saremmo, oltre che cugini, buoni amici.

Grazie

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays the usage of electronics devices in everyday life has been growing considerably. In particular they are used in safety-critical applications at high level of criticality, as they could cause damage to the human beings and/or properties, hence the correct behaviour of these elements becomes a priority. Moreover the manufacturing cost of these electronic elements has reduced over the years, while the costs of the testing process has actually increased, reaching the point of being comparable with the manufacturing ones. For these reasons the manufacturing companies are interested in techniques that allow achieving the best trade-off between costs and results.

In the next sections are addressed: i) the impact of these electronic devices in safety-critical application, ii) what is intended with "testing" and iii) the principal test procedures.

1.1 Electronic Devices and Safety Critical Applications

As mentioned, there is a massive presence of electronics devices that have the task of regulating very critical applications. For example in the automotive field, as shown in Figure 1.1, electronic devices are used for a number of different applications. Some of them are not particularly relevant from the safety point of view, while others, like the airbag system and the anti-lock blocking system (ABS) are elements expected to work properly and without errors during the entire operational life of the vehicle.

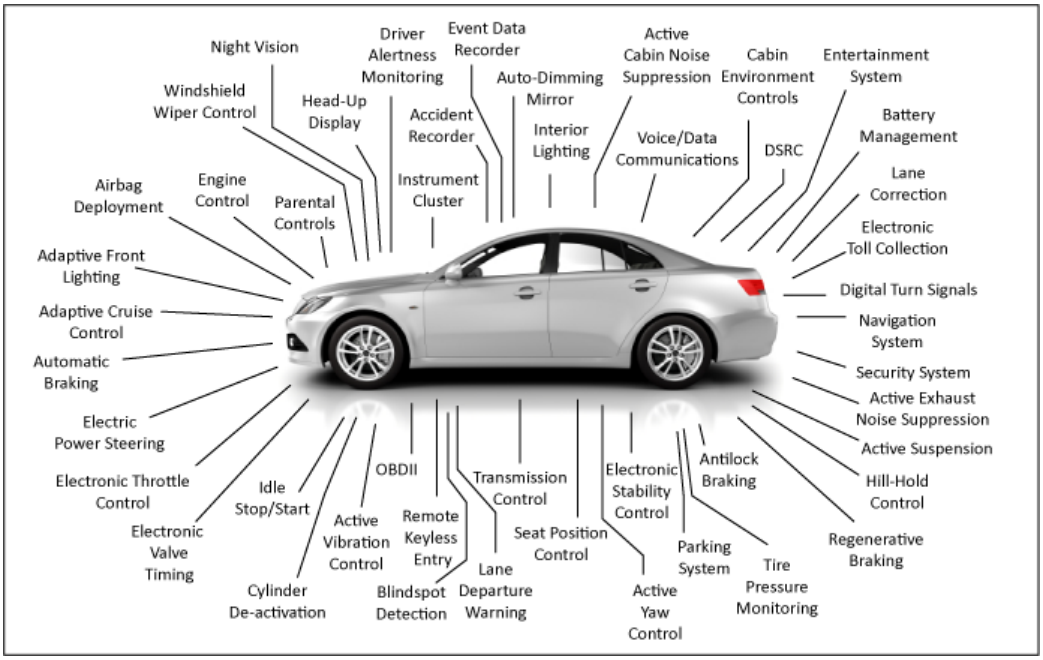


Figure 1.1: Electronic devices usage within a modern car

The part of the overall safety of a system depends on the system operating correctly in response to its inputs (including the safe management of operator error, hardware and software failures and environmental changes)

is defined as *Functional Safety*. The main objective of functional safety is to remove the risk of physical injury or damages to the health of people, either directly or indirectly (through damage to properties or to the environment), by the proper implementation of one or more automatic protection functions. In order to regulate the functional safety of a system, international functional *Safety Standards* for different application fields have been introduced. For example the ISO 26262 is an international safety standard for functional safety management of electronic systems for automotive applications, which defines the various requirements that the automotive companies must meet about their products.

In order to meet the various requirements imposed by those standards, high quality test procedures must be applied to the devices. Such mechanisms could be very expensive, for this reason companies are interested in solutions that allow achieving the best results, but at reasonable cost.

1.2 Introduction to Testing

Each product must perform a *Mission*, which is characterized by a *Function* and a *Duration* (operating life time). If the product is used in applications with processing criticalities (in which a processing error may cause loss of human lives or significant economic damages) it becomes extremely important that the product is able to complete its mission. Consequently, the quality of an electronic system has a growing importance with respect to the device success. A key parameter when assessing the quality of a product is the *De-*

pendability. This criterion is defined as the trustworthiness of a computing system that allows reliance to be justifiably placed on the service it delivers. The dependability of a product may be evaluated in a scientific way trough some attributes such as:

- *Reliability*: it is the probability that a product will perform its intended function adequately for a specific period of time.

- *Availability*: it is the probability that a system behaves correctly and is able to perform its task at a generic time (it differs from reliability since it relates to a single time instead of a period).

- *Safety*: it is the probability that the system either behaves correctly or it is able to interrupt its activity without causing serious damages.

In order to satisfy the reliability constraints a product has to be tested. *Test* is defined as the process aiming at identifying faulty products, hence the ones that do not function according to the specifications. As mentioned before, over the years the testing cost has become comparable with the production cost (cost of the silicon) (Figure 1.2). It is therefore important to analyze the various test techniques, as each one of them has advantages and disadvantages, and chose the optimal one in order to minimize the cost and maximize the results.

The incorrect behaviour of the devices is caused by a *Fault* or, in other words, by a defect in the system. It is relevant to point out that a fault

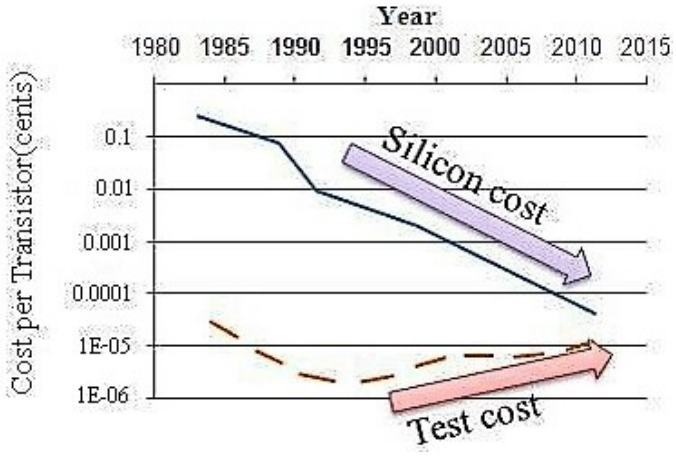


Figure 1.2: Comparison between test and production cost

not necessarily causes a failure in the system, in facts the fault needs to be activated at first; this creates an *Error* inside the system (i.e. a discrepancy between the expected behaviour and the actual one), which then needs to be propagated to the output in order to generate a failure (Figure 1.3).

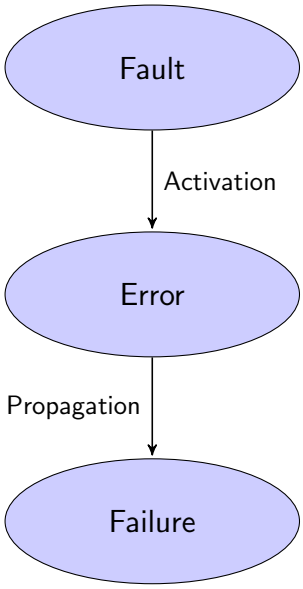


Figure 1.3: Fault, Error and Failure evolution

From this definition it is possible to notice two important aspects about the fault detection process: 1) the *Controllability* and, 2) the *Observability*. Basically a device could be seen as a black-box that receives inputs and produces outputs. Controllability is intended as the possibility to act on the inputs of the system, in order to excite the faults. Observability is meant as the possibility to see the propagation of the error to the output. In case a fault has been activated and leads to an error, but the misbehaviour of the system is not visible through its outputs we have what is called *Error-Masking*.

A fault can have different sources, but in general they can be divided in two categories:

- *Design faults*
- *Physical faults*

Design faults are related to rules violation and/or incomplete specifications during the designing phase. Physical faults are related to how the component has been mounted and fabricated.
Physical faults can also be divided in:

- *Manufacturing faults* (errors in the interconnections, component improperly assembled).
- *Manufacturing defects* (not directly related to human errors but to the manufacturing process).

- *Physical malfunction* (due to phenomena like electromigration or corrosion).

These physical defects have also a classification based on their duration and they could be divided in:

- *Permanent*
- *Temporary*

In particular a temporary fault again could be defined as *Intermittent* or *Transient*. With intermittent is intended a malfunction that occurs at intervals, usually irregular, in a device that behaves normally at other times. With transient is meant a malfunction that remains active for a short period of time.

Generally the physical defects are not approached in a direct manner, as this could be difficult to deal with, hence an abstract model called *Logical Faults* is used. The way a logical fault shapes a physical defect is called *Fault Model*. Different fault models exist and the identification of the optimal one depends on several parameters, such as the characteristics and the type of circuit, but the most commonly used is the *Stuck-At Fault Model*. This model is based on the assumption that a specific signal of the circuit has a fixed value 0 (stuck-at-0) or 1 (stack-at-1). For example in the Figure 1.4 the value on the input of the gate is assumed to be 0, no matter the real value applied to the input B.

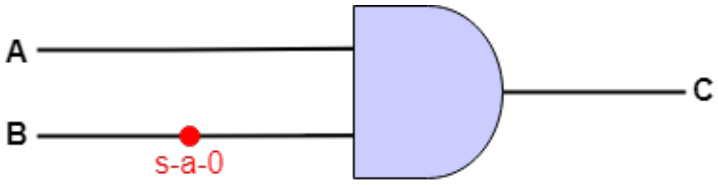


Figure 1.4: Stuck-at-fault example

As already mentioned, a fault in a device is detected by observing the misbehaviour produced by the fault itself on the device's outputs. The device to be tested during the test phase is called *Unit Under Test* (UUT). Normally, in order to detect a fault, a properly identified set of stimuli, also defined as *Test Vectors*, are applied to the UUT. The response of the UUT is then captured and compared with the outputs of the circuit with no errors (*Faulty-Free Circuit*), if there is a difference the circuit is faulty, otherwise it is considered as functioning (Figure 1.5).

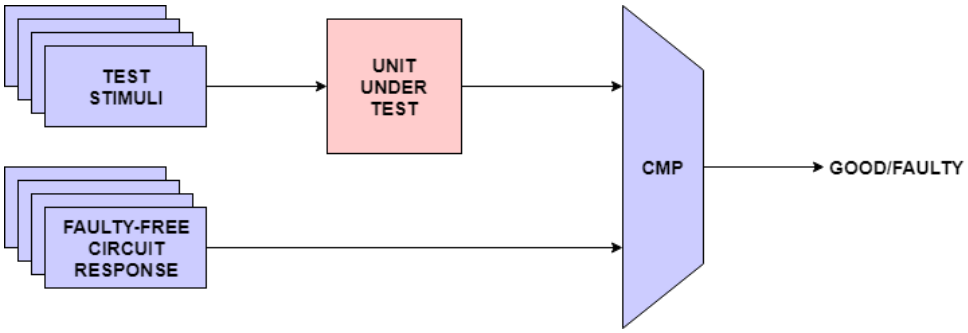


Figure 1.5: Test application

The higher the quality of the applied test, the higher will be the quality of the final product. The metric usually used to have a measure of this parameter is the *Defect Level*. This parameter is defined as the percentage of faulty devices that pass the test and is measured in *part per million* (ppm) or *parts per billion* (ppb).

The choice of the test vectors is an important phase as it can produces costs that might not be negligible. In facts a higher number of test vectors can imply a considerably long test time. Also, considering that in certain types of test strategies these vectors have to be stored in memories, they might require considerable amount of memory space. Usually, in order to avoid these contraindications, the test vectors are generated and evaluated using software tools.

The effectiveness of the chosen test vector given a specific fault model, is called *Fault Coverage*. This parameter is defined as the percentage of possible faults detected by the test over the total number of faults.

The application of the set of test vectors to the UUT is the test application phase. Generally the high number of test vectors and the number of devices to test (hundreds or even thousands) can make the test application phase very expensive in terms of time. Hence, special machines called *Automatic Test Equipment* (ATE) are used in order to maximize the quantity of devices tested. This tool is used to apply test stimuli, collect the responses produced by the UUT and compare them with the expected ones. Due to its complexity, accuracy and speed the cost of an ATE is rather high.

Integrated Circuit test can be classified in several different types:

- *Verification Testing*: it aims at executing a final verification of design correctness and compliance with specification defining also the exact

operating limits of the circuit in terms of temperature, voltages and other parameters.

- *Production Testing*: its objective is to guarantee the correct behaviour of the produced devices.

- *Reliability Testing*: this type of test measures the functionality of the product during an interval of time in a real-life environment. The major fields in reliability testing are the environmental test (product that has to work in extreme condition of the environment such as high/low temperature or levels of humidity) and the electrical test (product that has to survive to high/low voltages or currents). An example of this type of test is the *Burn-In*, a manufacturing test phase used for safety-critical modules and designed to check for the early life faults that that can affect the products. In facts in the early life of a product the *Failure Rate* (frequency at which a component fails) is high but rapidly decreasing as defective products are identified and discarded, then in the mid-life of a device the failure rate is low and constant while in the late life of the product the failure rate increases as age and wear take their toll on the product. This trend of the failure rate with respect to the product life is referred as the *Bathtub Curve* (Figure 1.6).

- *Incoming Inspection*: it is performed by the buyer (generally a system company) of a circuit, to make sure that it works correctly before it is used.

- *In-field Testing*: this type of test could be divide in *On-Line Testing*

and *Power On Self-Test*. The first one is used for the continuous testing of the device even during its operational life, without stopping its normal operations, while the second one is based on testing the device when it is powered-on.

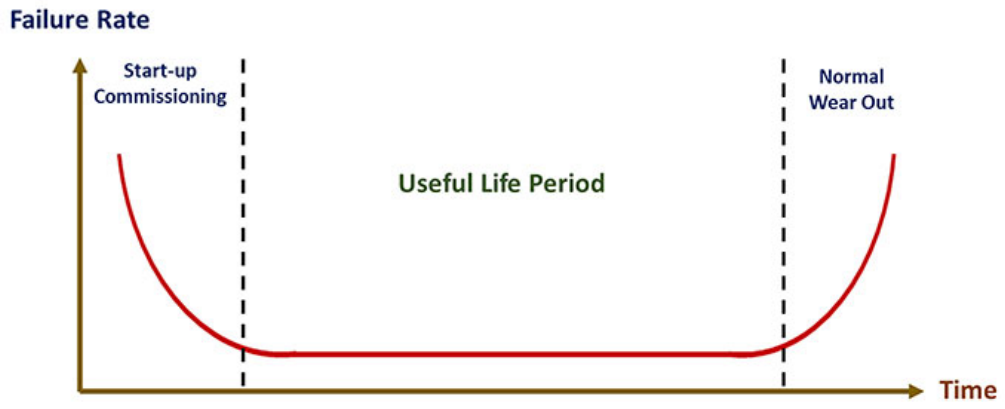


Figure 1.6: Bathtub curve

Usually when dealing with safety-critical applications, *Functional Safety Analysis* is used to evaluate the safety level achieved by the product. It comprises of quantitative evaluations such as the *Failure Mode Effect and Diagnostic Analysis* (FMEDA). FMEDA is a structured approach to define ways the safety device can fail. Hence it is useful to understand how much reliable the product might be and if there are good mechanisms able to detect the failures and to bring the system to a safe state. For example in [1] is presented a FMEDA analysis for automotive applications.

1.3 Safety Mechanism

In general a device is tested just before being delivered to the final user, this is called *End-Of-Manufacturing Testing*. The main goal of this process is to identify all the samples that are not working inside the specification window of the product. In safety critical fields such as the automotive one, this is achieved by reducing the *Defective Parts Per Million* (DPPM). This is a measurement used today by many customers to measure quality performance. One DPPM means one defect in a million.

The most critical systems in terms of safety need not only to be tested at the end of their production cycle, but should also be tested when they are already deployed in their operational field (*In-Field* testing).

In order to better manage these critical applications, various solutions, also referred to as *Safety Mechanisms*, are adopted. These safety mechanisms provide the possibility to identify the presence of errors (and maybe correct them), or permit the in-filed testing. These solutions are very important in any system with a goal of high reliability that can have the undesirable presence of *Single Points Of Failure* (SPOF) and *Latent Faults*. A SPOF is a part of a system that, if it fails, will stop the entire system from working. Latent faults are instead faults that are present in the system, although but hidden from regular means of detection, they become more dangerous in conjunction with a second fault.
Safety mechanisms can be grouped in two main categories:

- *Hardware-based* safety mechanisms
- *Software-based* safety mechanisms

The hardware-based safety mechanisms are solutions that require adding to the original circuit more hardware in respect to the original one. If the main goal is to detect/correct a fault, a product can be designed to be a *Fault Tolerant System* ([2], [3]). This type of system is able to stops the propagation of the fault and to send to the user the expected output value (*Triple-Modular-Redundancy* or TMR, Figure 1.7), or an alarm signal advertising that an error has occurred (*Duplication With Comparison* or DWC, Figure 1.8).

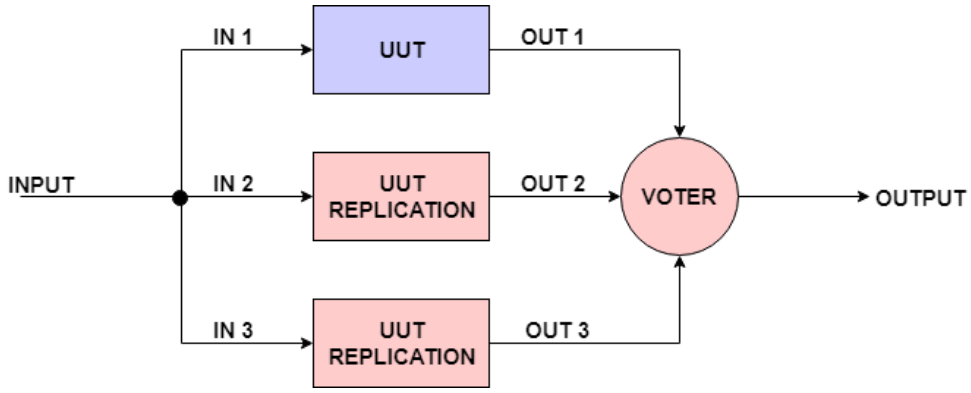


Figure 1.7: TMR, the UUT is replicated 2 times and a majority voter decides the output

Otherwise if the objective is to achieve in-filed testing there are solutions that permit to achieve high fault coverage, still using more hardware than in the original product. However, the need of extra hardware and the long test application time make these in-field safety mechanisms rather costly. Moreover, the utilization of such techniques requires a temporary disconnection of the UUT from its normal operation in order to apply the test, as in the

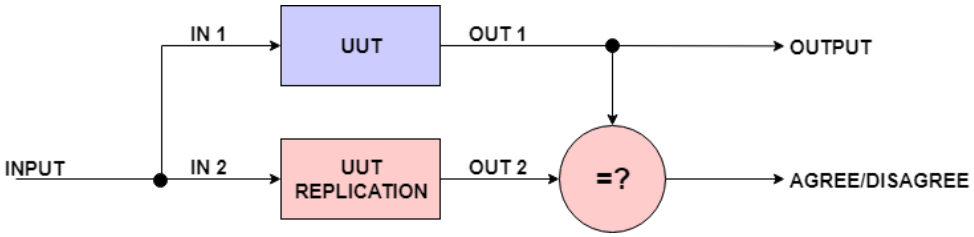


Figure 1.8: DWC, the UUT is replicated 1 times and a comparator module verifies if the two outputs are in agreements or not

case of this test, applied when the devices is switched on (POST). These limitations make a pure hardware-based mechanism not suitable to satisfy the safety standard requirements, as a given fault coverage must be guaranteed periodically, even when the system is already fully in function (on-line testing).

For this reason software-based mechanisms are applied. This family consists in a set of software procedures, or test programs, that form what is called *Software Test Library* (STL). It consists of a set of software procedures executed by the processor core, which main target is to test the processor coreitself and the peripherals surrounding it. One of the main advantages of this type of approach is that no extra hardware is needed, although the generation of a high-quality test program is still a time consuming process. Moreover, the fault coverage achievable by a pure software approach is normally lower in respect to the one obtained by a hardware approach.

As discussed, both hardware and software approaches have advantages and disadvantages. For this reason it could result advantageous merging these two approaches, in order to create a hybrid one. This solution per-

mits to exploit the advantages and reduce the limitations of the two distinct strategies.

In the next chapter are discussed the meaning of a combinational and sequential circuit, the fault simulation process and the software and hardware-based mechanisms in order to give to the reader the necessary background. Moreover, the proposed approach of combining hardware and software mechanisms to create a hybrid architecture is described in details in *Chapter 3*. *Chapter 4* presents experimental results and further analyses on the proposed approach. Finally *Chapter 5* reports the conclusions of this work, along with forecasts for future directions.

Chapter 2

Background

A description of a combinational and sequential circuit and of a fault simulator tool is presented in this chapter. Follows a more detailed discussion on the two main categories of in-field testing approaches, hardware and software-based, focused on the existing strategies and their advantages and disadvantages.

2.1 Combinational and Sequential Circuit

A *Combinational Circuit* is defined as a time independent circuit which does not depends upon previous inputs to generate any output but only on their present values (Figure 2.1).

Figure 2.1: Combinational circuit

Instead, as shown in Figure 2.2 a *Sequential Circuit* is a circuit which is dependent on present as well as past inputs to generate any output. It requires memory elements that are dependent to clock signal (in order to memorize a value) and reset signal (in order to initialise the memorised value to a default one).

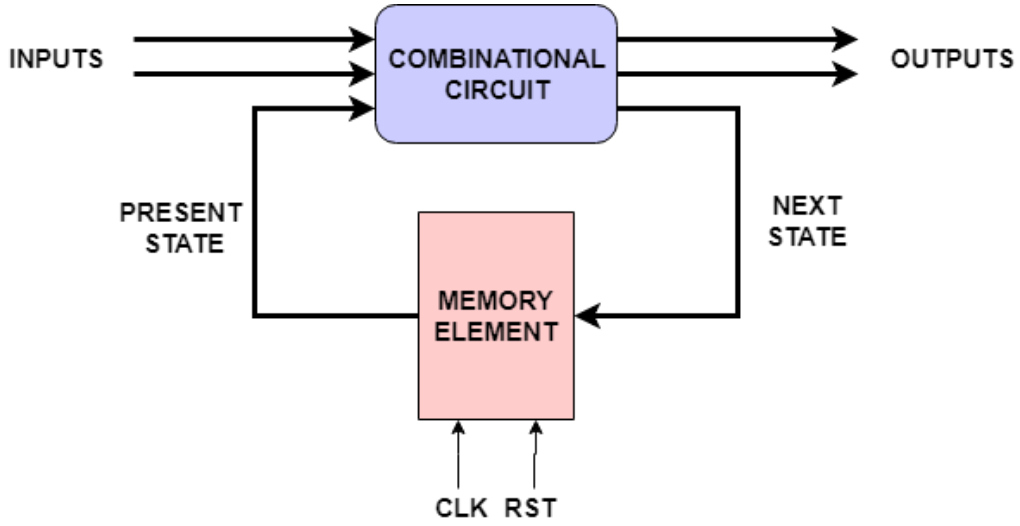


Figure 2.2: Sequential circuit

2.2 Fault Simulation

A *Fault Simulator* (Figure 2.3) is a software tool that receives as inputs:

- *Circuit Description*: files describing the circuit. In general they are VHDL or Verilog file.

- *Fault List*: text file containing the faults that are considered in the circuit, they could be all the possible faults or only a part of them.

- *Test Set*: set of patterns that are applied to the circuit.

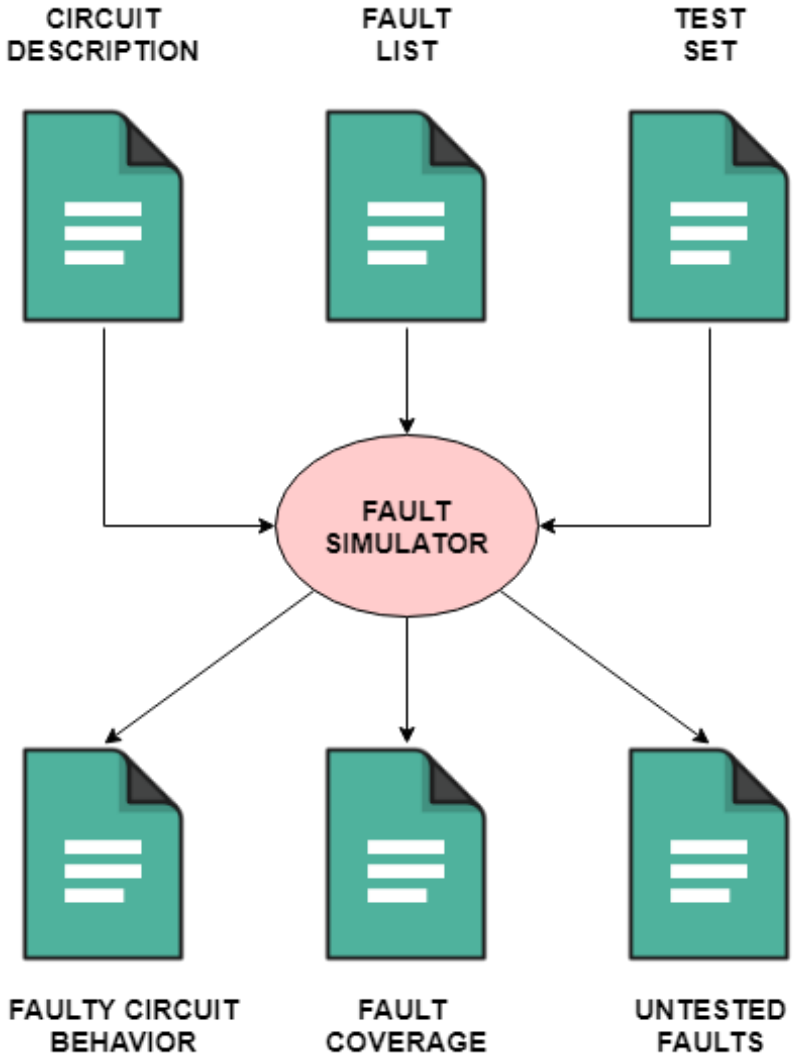


Figure 2.3: Fault simulation environment

The fault simulator computes the behavior of the circuit in the presence of each fault when the given test set is applied. The choice of the patterns to apply to the circuit is an important phase, as it can produce costs that will not be negligible. In facts a higher number of test vectors can imply

a long test time and, with the facts that in certain types of tests strategies these test vectors have to be stored in memories, in terms of memory space. Usually in order to avoid these contraindications the test vectors are generated using a software tool called *Automatic Test Pattern Generator* (ATPG).

The main purposes of a fault simulator are the analysis of a faulty circuit behaviour and the Fault Coverage computation, which permits also to find the *Untested Faults* (faults undetectable with the applied test patterns).

2.3 Built-In Self-Test

Built-In Self-Test (BIST) is an approach very effective in terms of achieving a high fault-coverage. It is based on inserting additional hardware in the system whose purpose is to test the device itself when being in the field. The basic scheme of a BIST approach is shown in Figure 2.4. The principal components of the BIST are:

- *Unit Under Test* (UUT): it is the circuit to be tested, it is delimited by its Primary Inputs (PIs) and Primary Outputs(POs).

- *Test Pattern Generator* (TPG): it generates the test patterns for the UUT, they can be generated in a pseudo-random manner or deterministically.

- *Multiplexer*: it disconnects the UUT from the PIs when the *Test Mode* is activated in order to fed the UUT with the pattern of the TPG.

- *Output Data Evaluator* (ODE): it analyzes the sequence of values on the POs and compares it with the expected one.

- *BIST Controller*: it controls the test execution, managing the TPG and ODE modules, reconfiguring the UUT and driving the multiplexer. It is activated by the Normal/Test signal and generates the Go/Nogo signal.

- *Normal/Test*: when it has the "Normal" value the UUT is fed with its PIs, no BIST test is performed. Instead if its value is "Test" it activates a BIST session, then the UUT is fed with the TPG patterns and the ODE captures the outputs.

- *Reconfigure*: in some BIST architecture the UUT internal logic is reconfigured in order to improve the controllability and observability.

If the UUT to be tested using BIST is a block of logic we are talking about *Logic Built-In Self-Test* (LBIST), instead if it is a memory, it is defined as *Memory BIST* (MBIST). A memory is a device that is able to read/store a data to/from a determined location defined by an address, for this reason the BIST architecture must be extended also for what concerns the addresses generation (Figure 2.5).

A TPG and an ODE module can be implemented in different ways. For example, a very popular solution is to use *Linear Feedback Shift Registers* (LFSRs). From the test pattern generation point of view, the LFSR architecture is presented in Figure 2.6, where a circle can correspond to a open or

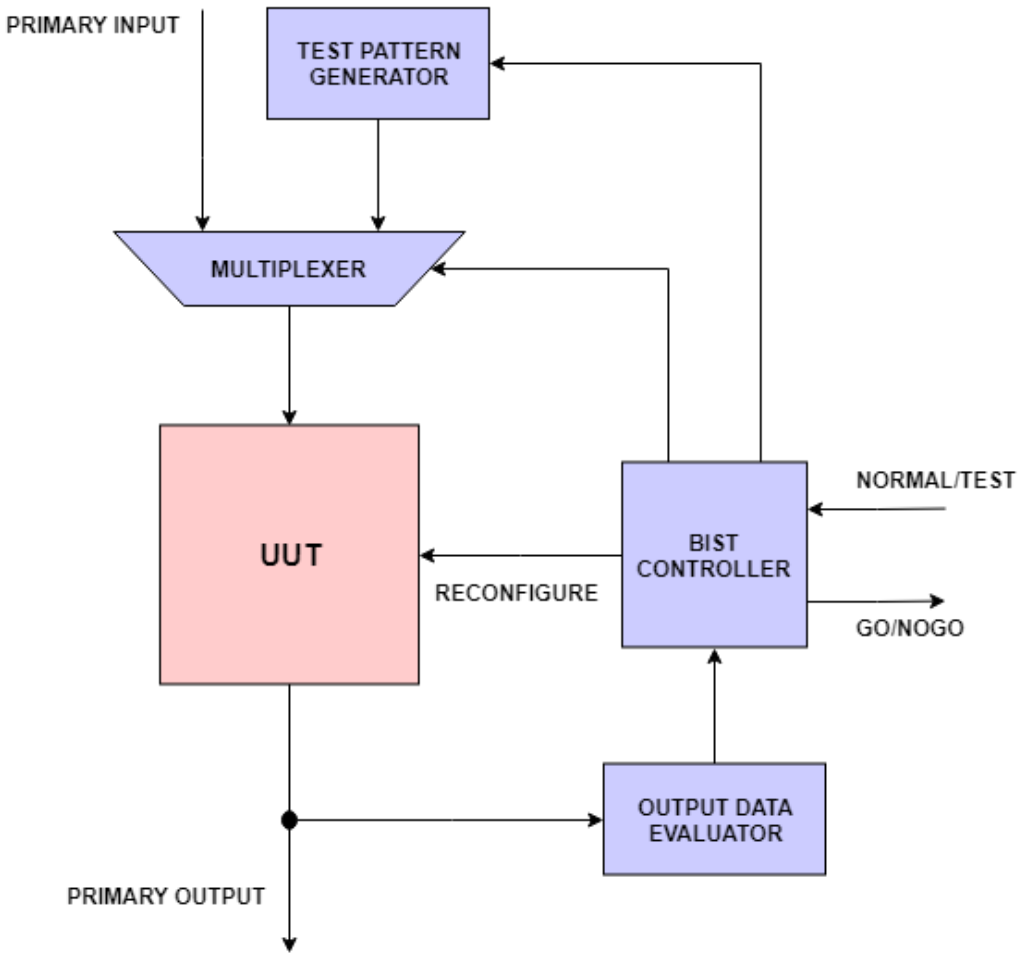


Figure 2.4: BIST architecture

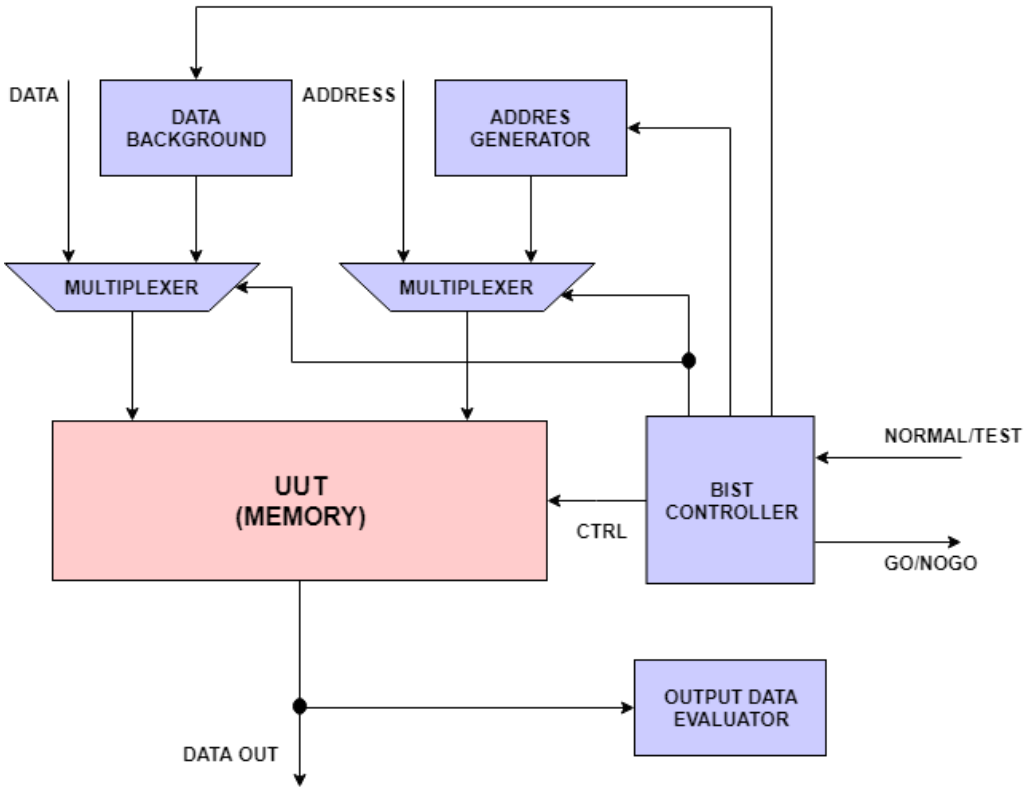


Figure 2.5: MBIST architecture

close connection, depending on the chosen pseudo-random pattern generation function to be implemented. They are pseudo-random pattern because after some test cycles the pattern are repeated and each new generated pattern has a dependency from the previous one. Usually, to avoid this drawback, a *Phase Shifter* is used. This is composed of some xor gates fed by couples or triples of LFSRs outputs, in order to generate uncorrelated values.

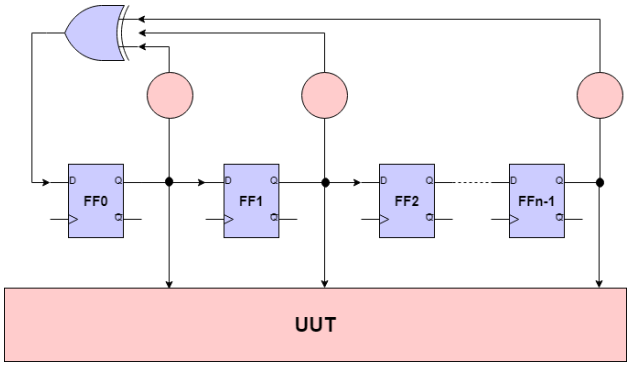


Figure 2.6: LFSRs as pseudo-random pattern generator architecture

Then the LFSR could also be used to implement a compactor for the ODE. In this case its task is to compact in a single vector of n bits, named *Signature*, the sequence of output values of the UUT. To do this is used what is called a *Multiple Input LFSRs* (MISRs). The architecture of a MISR is shown in Figure 2.7 and, again, a circle could indicate an open or close connection depending on the wanted compaction function.

Exist several examples of both MBIST and LBIST. For example in [4] a strategy that combine the existing best practices and present a concept of full-scale functional safety solution for testing memories in automotive System-on-Chips based on the designed versatile BIST engine and in-field

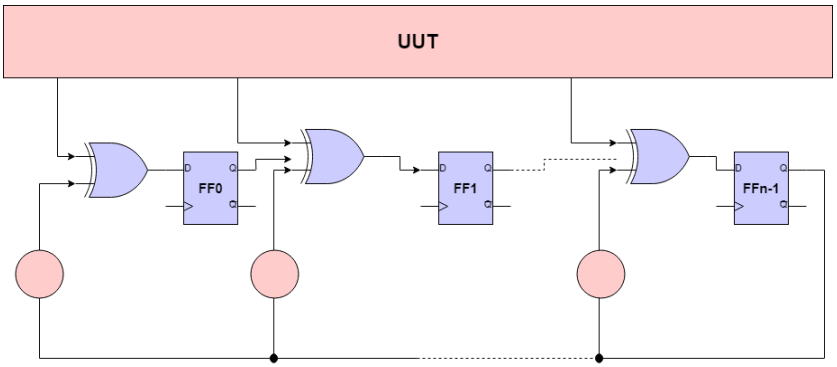


Figure 2.7: MISR architecture

error correction capability is discussed; while in [5] is presented a methodology to include LBIST in a multicore processor for periodic online testing and presents also the results obtained of the presented work done on actual Arm IP.

From the point of view of the area overhead, exist solutions that partially reuse some *Design for Testability* (DfT) techniques intended for end-of-manufacturing test. For instance the *Self-Testing Using MISR and Parallel SRSG* (STUMPS) architecture reuses the *Scan Chains*. Scan chain is a DfT method that during the test mode reconfigures all the flip-flops of the UUT in order to form a shift-register and shift-in and out patterns in the UUT. The STUMPS architecture (Figure 2.8), as described in [6], is used to first shifting-in the pseudo-random patterns generated by the LFSRs and then shifting-out the system responses that will be compacted in the MISR.

In conclusion BIST approaches permits to reach good level of fault coverage with the cost of a non-negligible area overhead. Moreover BIST is

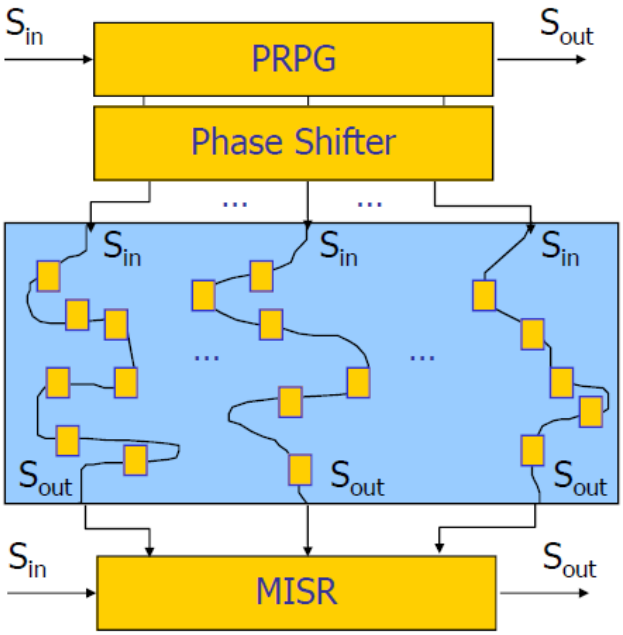


Figure 2.8: STUMPS architecture

effective only when huge amount of patterns are generated, which requires a significant amount of test time. Besides, since random values are forced into flip-flop and PIs, the state of the system is completely destroyed. Therefore, it is very difficult to adopt such approach for periodic in-field testing. Indeed, solutions based on BIST find applicability during the POST.

2.4 Software-Based Self-Test

Software-based mechanisms are popular solutions for the on-line testing. These mechanisms are developed according to the *Software-Based Self-Test* (SBST) approach. A SBST is a special kind of test for processors, it is used both for in-field test and for end-of-manufacturing test as a complement to other solutions. Such strategy, initially proposed by [7], has been studied by

different research groups (for example [8] and [9]) as described in [10]). Then this type of test has also been extended in safety applications, for example in [11] is described an approach for the development of test programs for microprocessors used in safety-critical automotive embedded systems.

In general a SBST is based on a carefully devised test program that is executed by the processor in order to exciting possible faults. In general a SBST procedure is divided in three phases (Figure 2.9):

1. *Test program upload*: the test code and the test data are downloaded into processor memory.

2. *Test program execution*: the processor executes the test program fetching the instructions from the memory.

3. *Test result download*: along the execution of the test program, results have to be stored in a suitable location readable from the outside.

Typically the memory where the test program is stored could be of two types: i) *RAM* memory and ii) *Flash* memory. The RAM memory is a volatile memory, for this reason the test programs are uploaded every time they have to be executed. The Flash memory instead is a non-volatile memory, hence the test programs are uploaded just once and they are run any time it is required.

When used for in-field test, the SBST test program could be executed at the moment of system reset, or during the normal operations. In the first

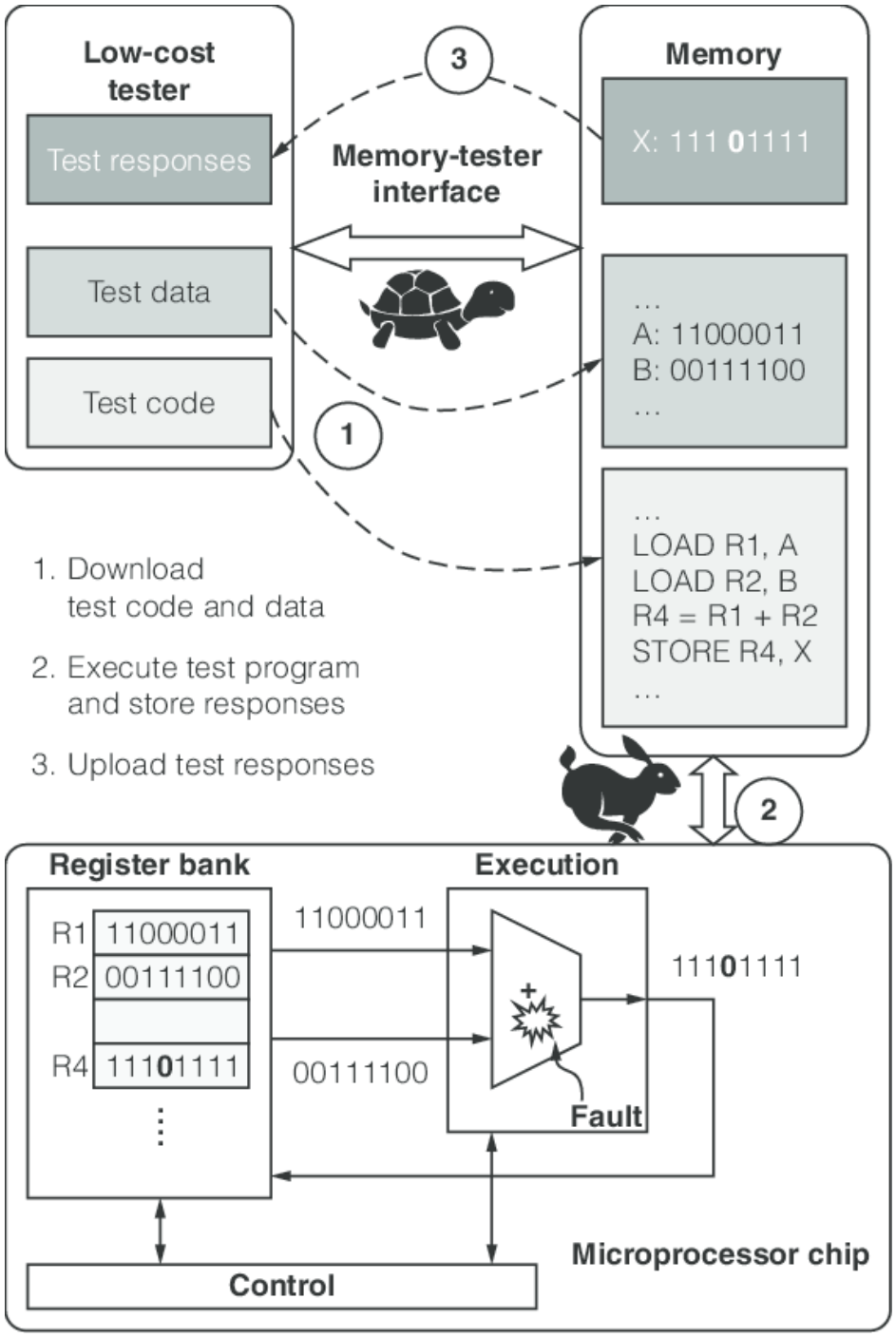


Figure 2.9: SBST flow

case there are not particular constraints that the test program has to respect because the state of the processor has not to be preserved and then restored. Instead when the execution of the test program is activated during the normal operation, the devices is temporarily disconnected from its normal flow of operation. For this reason the test duration has to be enough long to detect the possible fault but also enough short to guarantee the correct behavior of the device. For example a safety critical application as the airbag activation of a vehicle that is tested in-filed during the normal operation, must not be so long in order to guarantee that the airbag is correctly activated in case an accident occurs just in that moment.

The main advantages are: i) the reduced test application time and ii) it is not required to add extra hardware to the processor for executing the self-test, since the already existing programmable resources have been already exploited. Normally, the only hardware required is a Flash memory that should be reserved for storing the test code and occasionally a portion of RAM memory for storing the data needed by the test. In order to generate a good test program different techniques can be adopted. In [12] a combination of SBST methodologies with verification-based self-test programs supplemented by directed random test-program generation is discussed. In [13] is presented a methodology to generate automatically a test program from only a simulatable RTL description of the processor and the instruction set architecture (ISA specification). Nevertheless, as discussed in [14], the most common SBST strategies are in general of three types:

- *ATPG-Based*: this methodology guarantees the highest possible cov-

erage, given the fact that test patterns for a specific module are automatically generated by means of an *Automatic Test Pattern Generator* (ATPG). A tool that, given a circuit description, generates tests vectors. Further work is needed to transform the obtained patterns into a valid sequence of instructions that the processor will execute.

- *Deterministic*: it consists into the implementation of a documented algorithm or methodology. The expected fault coverage level goes from medium to high. In some cases, it requires several adjustments and simulations, until an appropriate level of fault coverage is reached.

- *Evolutionary-based*: this methodology uses an evolutionary engine to continuously generate functional test programs. The fault coverage can be very high, but it requires the generation of many programs increasing the generation time.

One of the disadvantages of a software-based self-test, is that the achievable fault coverage of a set of test programs is usually limited by the fact that a given test program can exclusively reproduce pure functional stimuli. Hence, a certain amount of faults that could potentially lead to a failure cannot be excited, as they can be detected only with values that the "legal" instructions of the processor are not able to reproduce. Another issue is the responses of each test program, which are essential for determining whether a given fault can be labelled as detected or not. In facts normally results are accumulated via software in order to form a signature and compare it with the expected one. However it may happen that not all the faults effects are

reflected in the final signature, hence some of them can be missed or their effect could be masked.

Chapter 3

Proposed Approach

As mentioned before, both SBST and BIST approaches have both undeniable advantages as well as limitations. Briefly, concerning the SBST, it is evident that the intrinsic flexibility of a software approach does not bring any penalty in terms of silicon area in the final device, nevertheless it suffers from:

1. *limited controllability*: exclusively pure functional pattern can be produced;
2. *limited observability*: faults effects might be masked and not correctly stored in the final signature;

Contrarily, BIST has full access to the underlying hardware but:

3. it is nearly impossible to maintain the system state unaltered during the self-test procedure;
4. the test application time is long.

Hence the idea is to create a Hybrid architecture to overcome the limitations of the current solutions adopted for both hardware and software approaches. In doing this it becomes possible to complement the execution of a SBST, which yields to an acceptable fault coverage at run-time, with the direct application of specific test patterns to the targeted units in order to have a more completed fault coverage (which can be hardly achieved with the SBST solely).

Hybrid approaches to the in-field testing are not new ([15], [16], [17] and [18]). In respect to other existing works like [19], [20] and [21], the proposed DfT does not require detailed knowledge of the processor it is applied to. Moreover, the extra DfT hardware is not a completely separated module but is inserted within the processor core exploiting the existing functionalities of the processor.

The proposed approach is intended for the on-line testing of arithmetic modules, both combinational and sequential. The resulting architecture is called *OPTIMUS*, which stands for *O*-nline *P*rogrammable *T*est *I*nfrastructure for com*M*p*U*tational module*S*.

In the next sections OPTIMUS (general implementation, internal registers usage, main components and available operational modes) and the possible test procedure are presented.

3.1 OPTIMUS

The architecture of OPTIMUS is visible in Figure 3.1. As mentioned above, it has been implemented in order to handle combinational and sequential arithmetic blocks. The working principle of OPTIMUS is, having the full access of the primary input of the two module, to compact in a signature the output of the modules in order to increase the observability and also to feed the module with loaded test pattern maintaining the processor in a safe state in case a special instruction has occurred.

In the next sections are described the implementation, the principal components and functionalities of OPTIMUS.

3.1.1 Implementation

Here is addressed the insertion of OPTIMUS in the processor. Figure 3.2 shows the Inputs and the Outputs of OPTIMUS. OPTIMUS is a sequential circuit, so it needs to use a CLOCK and RESET signal, these signal are the same used from the entire processor. The other signal can be divided in three groups:

1. the signals related to the Combinational unit;
2. the signals related to the Sequential unit;
3. the signals related to implement the *Special Purpose Register* (SPR) interface in order to correctly write and read the internal registers.

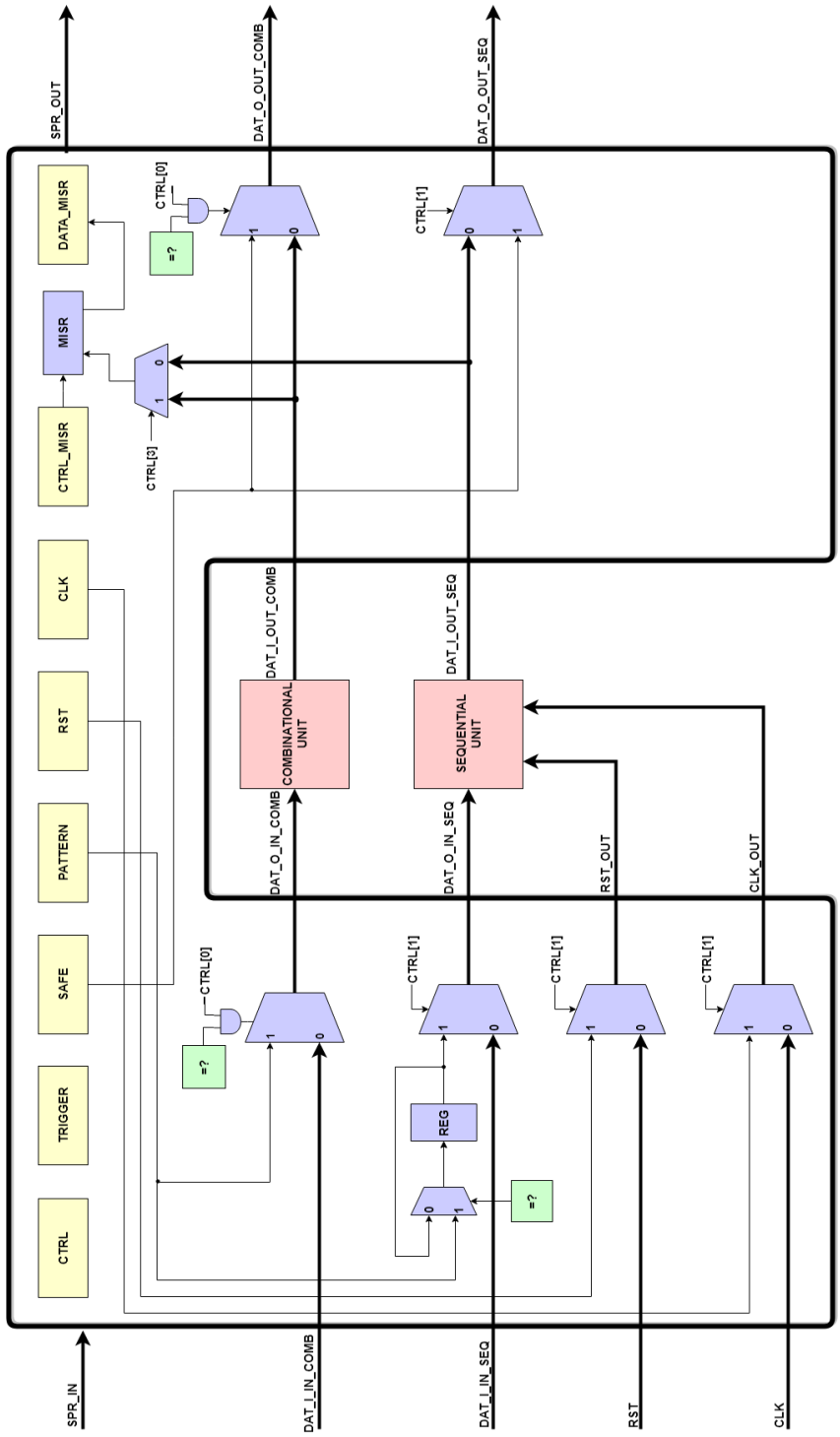


Figure 3.1: OPTIMUS schematic assuming it is applied to a combinational and a sequential unit

```
//CLOCK and RESET
input                       clk;
input                       rst;

//COMBINATIONAL UNIT
input   [comb_in-1:0]   dat_i_in_comb;
output  [comb_in-1:0]   dat_o_in_comb;
input   [comb_out-1:0]  dat_i_out_comb;
output  [comb_out-1:0]  dat_o_out_comb;

//SEQUENTIAL UNIT
input   [seq_in-1:0]    dat_i_in_seq;
output  [seq_in-1:0]    dat_o_in_seq;
output                  rst_out;
output                  clk_out;
input   [seq_out-1:0]   dat_i_out_seq;
output  [seq_out-1:0]   dat_o_out_seq;

//SPR INTERFACE
input   [spr_in-1:0]    spr_in;
output  [spr_out-1:0]   spr_out;
```

Figure 3.2: Inputs and outputs of OPTIMUS

3.1.1.1 Combinational Unit Signals

OPTIMUS is inserted around the combinational module, so it needs the declaration of four signals:

- *dat_i_in_comb*: it composes the original inputs of the combinational circuit before the insertion of OPTIMUS, it comes from other modules of the processor. This signal is an input of OPTIMUS;
- *dat_o_in_comb*: it is the effective input of the combinational circuit, it could be the dat_i_in_comb signal or, in case the necessary conditions are verified, the test pattern. This signal is an output of OPTIMUS and the input of the combinational module;
- *dat_i_out_comb*: it is the combinational circuit output, if the internal

MISR is active, it is used to capture the response of the module and compute the signature. This signal is an input of OPTIMUS and the output of the combinational module;

- *dat_o_out_comb*: it is the OPTIMUS output that is sent to other modules of the processor for what concerns the combinational unit. It could be the dat_i_out_comb signal or, in case non functional pattern are applied to the combinational circuit, it could be a safe value in order to maintain the processor in a safe state. This signal is an output of OPTIMUS.

3.1.1.2 Sequential Unit Signals

Again, as discussed for the combinational unit, OPTIMUS is inserted around the module. Hence, the input and output signals of OPTIMUS concerning the sequential circuit are:

- *dat_i_in_seq*: it composes the original sequential circuit inputs before the insertion of OPTIMUS, it comes from other modules of the processor. This signal is an input of OPTIMUS;

- *dat_o_in_seq*: it is the effective input of the sequential module, it could be the dat_i_in_seq signal, or in case the necessary conditions are verified, the test pattern. This signal is an output of OPTIMUS and an input of the sequential unit;

- *rst_out*: it is the reset signal for the sequential circuit, it could be the reset signal used by the entire processor or, in case OPTIMUS is

activated to handle the sequential circuit, a reset controlled externally using the SPR functionalities. This signal is an output of OPTIMUS and an input of the sequential unit;

- *clk_out*: it is the clock signal for the sequenial module, it could be the clock signal used by the entire processor or, in case OPTIMUS is activated to handle the sequential circuit, a clock controlled externally using the SPR functionalities. This signal is an output of OPTIMUS and an input of the sequential module;

- *dat_i_out_seq*: it is the sequential circuit output, if the internal MISR is active, it is used to capture the response of the module and compute the signature. This signal is an input of OPTIMUS and the output of the sequential module;

- *dat_o_out_seq*: it is the OPTIMUS output that is sent to other modules of the processor for what concerns the sequential circuit. It could be the dat_i_out_seq signal or, in case non functional pattern are applied to the sequential module, it could be a safe value in order to maintain the processor in a safe state. This signal is an output of OPTIMUS.

3.1.1.3 SPR Interface

OPTIMUS has been implemented as a module inside a processor having all the internal registers addressed as Special Purpose Registers. This implementation permits to easily load and store value in the module. The operation is done respectively using 2 instructions: i) *Move To Special Purpose Register*

(MTSPR) and ii) *Move From Special Purpose Register* (MFSPR).

An example of these two instructions is presented in Figure 3.3.

```
1 l.mtspr r0, r3, 0xcafe
2 l.mfspr r4, r0, 0xcafe
```

Figure 3.3: Example of MTSPR and MFSPR instructions

The first instruction is a write operation. It is saying to move the content of register r3 into the special register defined by content of register r0 logically ORed, with the hexadecimal value 0xcafe. In particular the value of r0 is always 0 hence the instruction is saying to move the value of r3 in the SPR with address 0xcafe. The second instruction is instead a read operation. It indicates that the content of the SPR, which address is defined by the content of register r0 logically ORed with the hexadecimal value 0xcafe, is moved in register r4. Again as before if it used register r0 this means that the content of the SPR with address 0xcafe is moved into register r4.

Hence, as part of the SPRs in the processor, OPTIMUS has to conform to the internal SPR interface for writing and reading data. The number of needed signal and their width may vary depending on what type of processor OPTIMUS is implemented, but in general there are:

- *spr_in*: input signals of the SPR interface, used to correctly select the unit and the SPR registers of the unit that has to be read or written. This signal is an input of OPTIMUS
- *spr_out*: output signal of the SPR interface, this signal presents the value of the SPR register that is read. This signal is an output of OP-

TIMUS.

The SPRs of OPTIMUS can be grouped in six categories:

- *Control*
- *Trigger*
- *Safe*
- *Pattern*
- *Reset & Clock*
- *MISR*

3.1.1.4 Control SPR

This category contains the register that is used to activate and configure OPTIMUS for different operational modes (they will we described in next sections). Basically it is used to disconnect the unit under test from its normal inputs in order to feed it with the test patterns (and the clock and the reset signal for sequential circuit). This register permits also to send as output a safe pattern, in order to maintain the processor in a safe state. It also selects the data that reach the MISR to compute the final signature. It is composed of 3 bits, as follows: the bit[0] is used to manage the input and the safe pattern that are relative to the combinational circuit, the bit[1] serves to manage the sequential circuit and the bit[2] drives the input of the MISR that, if active, receive the responses that come from the combinational or from the sequential circuit (Figure 3.4).

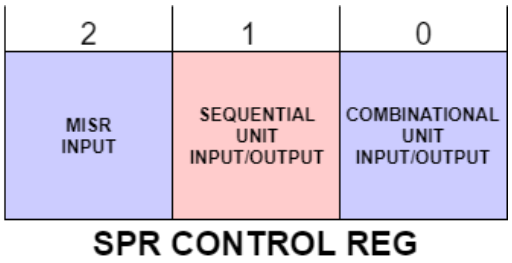


Figure 3.4: SPR Control reg bits

3.1.1.5 Trigger SPR

In this type of register is stored the value of the primary input that the sequential or the combinational circuits have when a specific instruction is invoked in the test program. When this programmable instruction appears with OPTIMUS correctly configured, the inputs of the unit under testing are switched from the "real" ones to the test ones. In particular this register is used both for the combinational and the sequential units, also if the two modules have different numbers of input bits. For this reason it has been implemented in order to have the size of the module with higher number of inputs. For the module with fewer bits, the extra ones are discarded.

3.1.1.6 Safe SPR

In this category of SPR is stored the safe value of the combinational or of the sequential circuit, that OPTIMUS sends as output for the other modules of the processor, when one of the two modules is fed with test pattern in order to guarantee that the processor maintains a safe state. This is necessary as the test patterns that feed a module are not functional patterns, so they could result in a non-supported instruction by the processor instruction set

that can drive the processor in a not stable state. Again as described before this type of register has been designed in order to fit the size of the module with the higher number of output bit. For the module with less bits, the extra ones are discarded.

3.1.1.7 Pattern SPR

In this category of register is stored the test pattern to be applied to the tested unit when OPTIMUS is active and a Trigger Instruction has occurred. Again the total size of this group of registers follows the same rule discussed for the Trigger SPRs.

3.1.1.8 Reset & Clock SPRs

These two registers of one single bit are used as clock and reset input to the sequential module under test when the test patterns are applied. In facts, when dealing with these type of circuits, the test pattern has not to be simply applied. It actually also requires to be captured by the registers of the module by means of a clock pulse and sometimes also of a reset pulse. For these reasons these registers have been designed in order to recreate a pulse. In Figure 3.5 is shown how they work: initially their value is 0, then with a MTSPR instruction the value 1 is written and after a clock pulse they are automatically turned to 0 again.

3.1.1.9 MISR SPR

From this category belong the Control and the Data SPRs for the MISR. The MISR Control register is a single bit register used to activate the internal

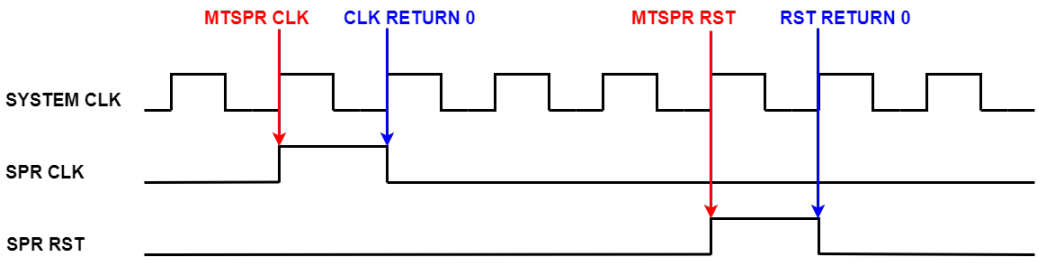


Figure 3.5: Reset & Clock SPRs pulse

MISR. Once the MISR is active the computation of the signature using the data that are coming starts and is stored in the Data MISR register that can only be written by OPTIMUS itself. If the MISR is not active no signature is computed. Again the Data MISR register is designed in order to have a total amount of bits equal to the output bits of the largest unit that can be tested. The value of the Data MISR register will be the spr_dat_o output of OPTIMUS using MFSPR instructions.

3.1.2 OPTIMUS Main Components

Here are addressed the components that permit the signature computation (*MISR*) and the detection of the trigger instruction (*Trigger Comparator*).

3.1.2.1 MISR

It has been implemented as shown in Figure 3.6. The *input_misr* signal comes from the output of the combinational or of the sequential circuit, depending which of the two modules is tested. Consequently, until the MISR is enabled, the signature is stored in the data MISR SPRs.

As discussed, in order to make a fault detectable, it is necessary that the

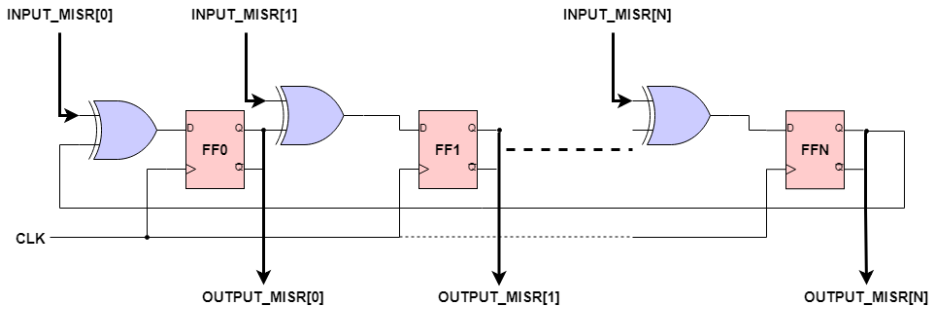


Figure 3.6: OPTIMUS internal MISR

responses of a circuit are propagated to the outputs. In facts the two tested circuits are internal modules of the processor, then the signature computed by the MISR needs to be moved. To do this the contents of the data MISR SPR is first of all moved to some general-purpose register through MFSPR instruction, then using Store operation the data contained in this register is visible to the output of the processor (Figure 3.7).

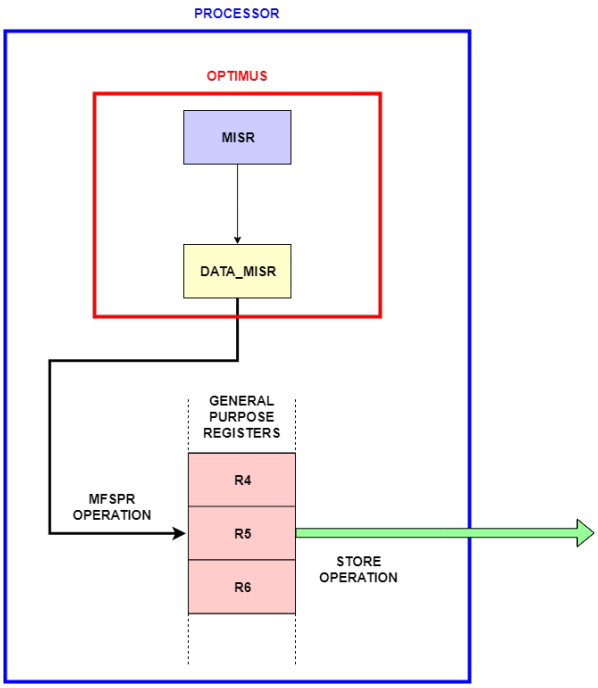


Figure 3.7: Propagation of the signature to the processor output

3.1.2.2 Trigger Comparator

This component is in charge of detecting whether a trigger instruction is present or not. In Figure 3.1 it was drawn for simplicity as a green square with a "=?" inside, but in figure 3.8 is possible to see its real implementation.

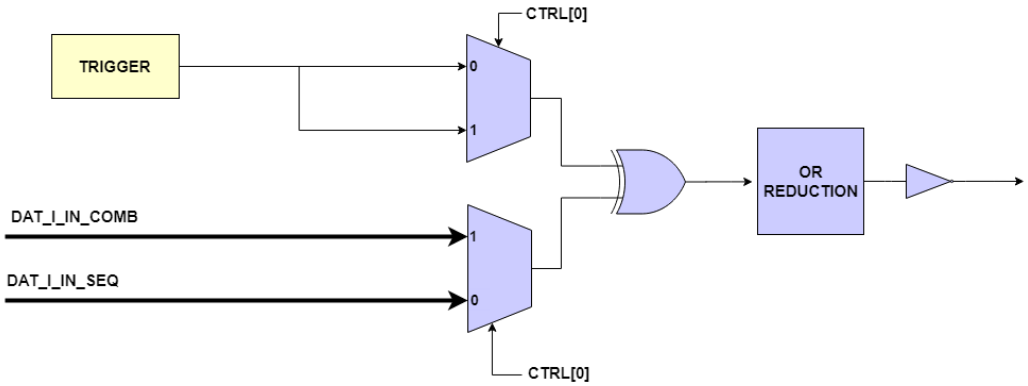


Figure 3.8: Trigger comparator

Basically this block compare the value contained in the Trigger SPR with the one that the tested unit is receiving. Instead of using a trigger comparator for each unit, in order to reduce the needed hardware, multiplexers driven from the Control SPR are used to select if the unit to be tested is the combinational one or rather the sequential one. Again the implementation respects the size of the module with the largest number of bits. The extra bits of the module with less width are filled with "00...00". The two signals to be compared reach a XOR gate, the output of this gate will be all 0s only if the two signals were equal. Then the XOR gate output is reduced to a single bit using an OR reduction operation and finally its value is complemented. At the end the final value will be "1" if the input of the unit and the value contained in the Trigger SPR are equal, otherwise the final value will be "0".

3.1.3 OPTIMUS Operational Mode

OPTMUS is able to perform three different operational modes:

- *Normal Mode*
- *Combinational Test Mode*
- *Sequential Test Mode*

In particular the last two modes require using non-functional pattern as input for the combinational or the sequential units.

3.1.3.1 Normal Mode

In this mode OPTIMUS is not active. This means that it works as if it is transparent. Hence the inputs and the outputs of both the combinational circuit and the sequential one pass through OPTIMUS with no modifications (Figure 3.9). To configure this mode the ctrl SPR needs to have value "X00". The X could be either 1 or 0, as it drives the data that goes in the internal MISR. If it is not active, is not relevant which data arrive. Thanks to the fact that the MISR can be independently activated writing "1" in the ctrl_misr SPR, in Normal Mode is also possible to use a normal SBST procedure with augmented observability to capture the response of the combinational (ctrl SPR = "100") or of the sequential (ctrl SPR = "000") circuit.

3.1.3.2 Combinational Test Mode

In this mode OPTIMUS is configured to test a combinational unit. The ctrl SPR is set to "101" and in order to compute the signature the ctrl_misr is

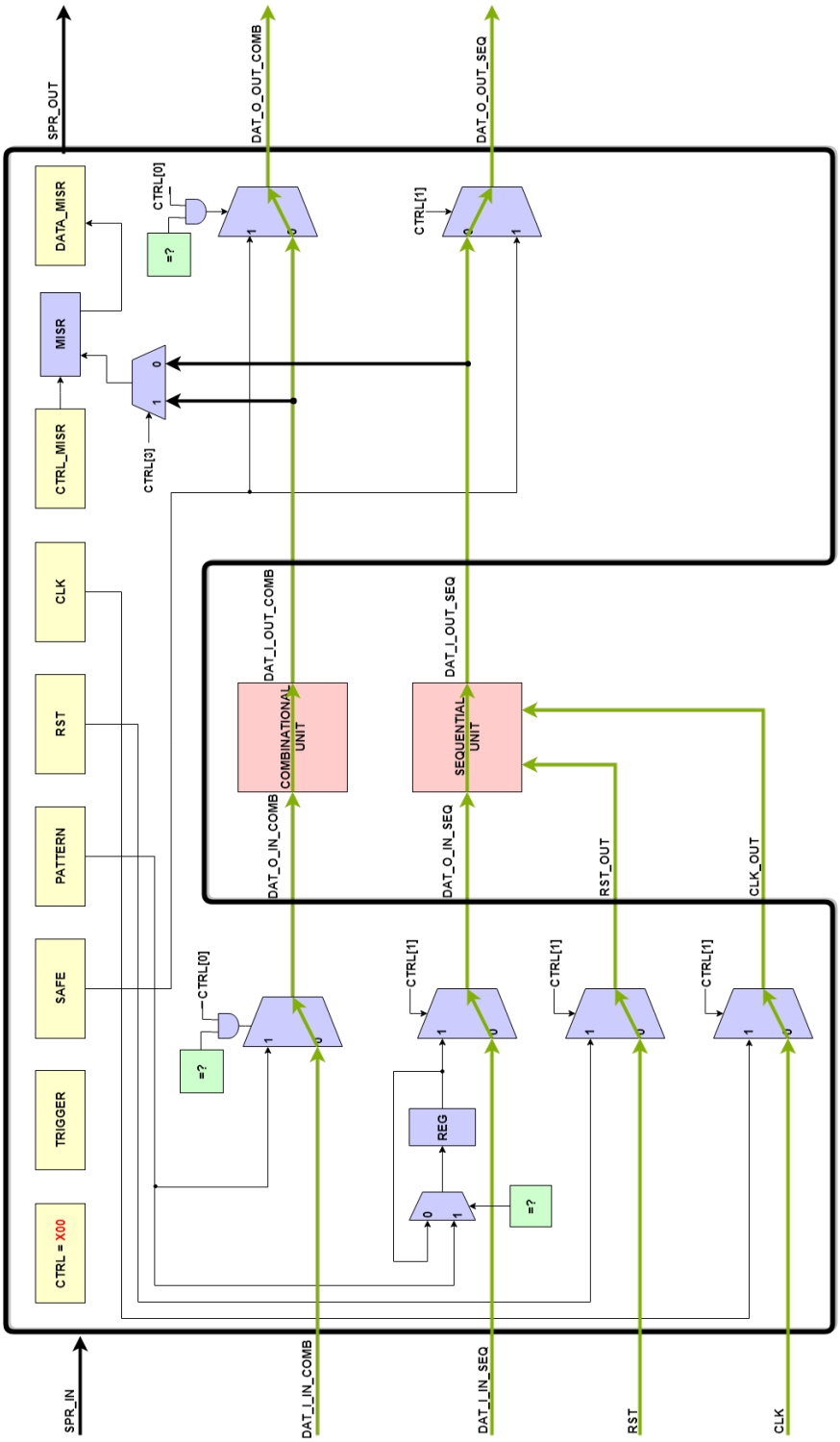


Figure 3.9: OPTIMUS during normal operations

equal to "1". Then each time the inputs of the combinational circuit that arrive to OPTIMUS are equal to the value stored in the Trigger SPR the effective input of the combinational module and its output sent to the other modules are swapped respectively with the content of the Pattern and Safe SPRs. This also means that, if OPTIMUS is active to test the combinational module, the input/output signals have a modification only when the trigger instruction appears. For this reason between Trigger Instructions is possible to set new values for the various SPRs. While the combinational unit is tested, the sequential one operates normally (Figure 3.10).

3.1.3.3 Sequential Test Mode

When configured in this operational mode, OPTIMUS is ready to test a sequential circuit. The ctrl SPR has value "010" and the ctrl_misr "1". When configured in this way the real clock and reset signals of the tested circuit are disconnected and substituted with the value in the rst and clk SPRs and the safe patterns for the sequential circuit are sent to the other modules of the processor. In particular dealing with sequential circuits requires that, from a trigger instruction and the other, the inputs of the circuit must be stable. For this reason the configuration in Figure 3.11 is used. This permits to fed the sequential module with a stable value and the next test pattern is applied only after that the Pattern SPR is loaded with a new content and a Trigger Instruction is invoked. Again when the sequential circuit is under test, the combinational one operates normally (Figure 3.12).

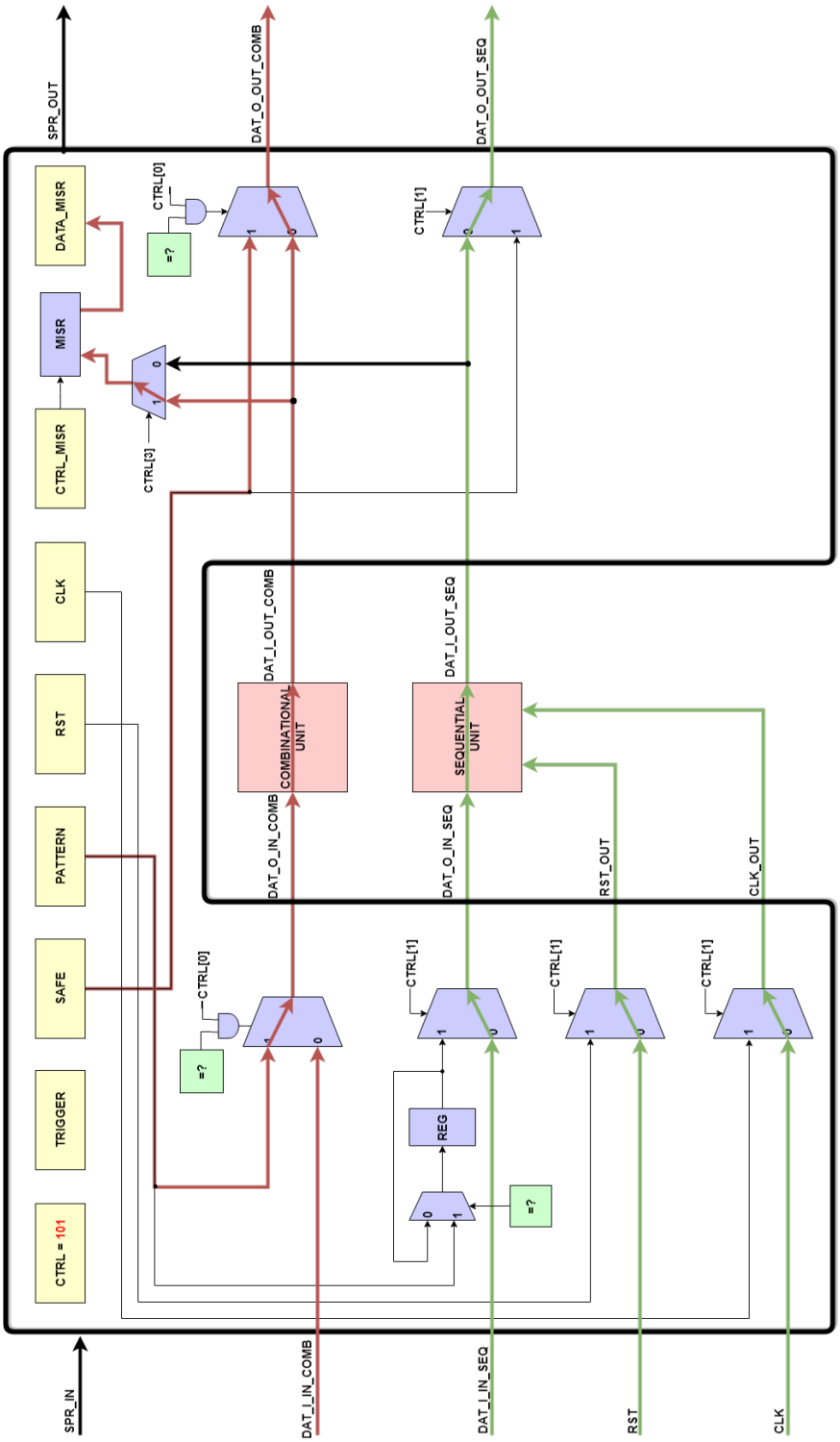


Figure 3.10: OPTIMUS with test pattern applied to the combinational unit

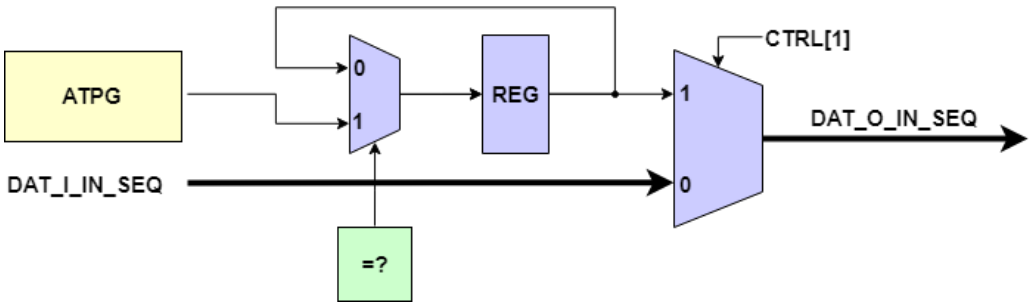


Figure 3.11: Stable input for the sequential unit between test pattern applications

3.2 Test Generation Flow

As previously discussed, OPTIMUS supports the direct application of test pattern to both combinational and sequential circuits without affecting the other modules of the processor. However, it is important to support the insertion of OPTIMUS within the processor with a suitable test generation flow. A trivial approach could be to generate the test patterns trough an ATPG process considering all the possible faults that affect the tested circuit. This approach could lead to a a high number of test patterns being generated, negatively affecting the test application time and the memory space required for saving the test program. A better approach consists in leveraging the hardware offered by OPTIMUS. As mentioned, the SBST techniques are limited to error-masking problems when computing the test program signature. This drawback can be mitigated by exploiting the internal MISR of OPTIMUS (which can be independently activated). This means that at the beginning of a SBST test program, the MISR can be activated and then, at the end of the self-test procedure, disabled. Then, the final signature computed by the MISR can be retrieved and compared with the expected one. By adopting

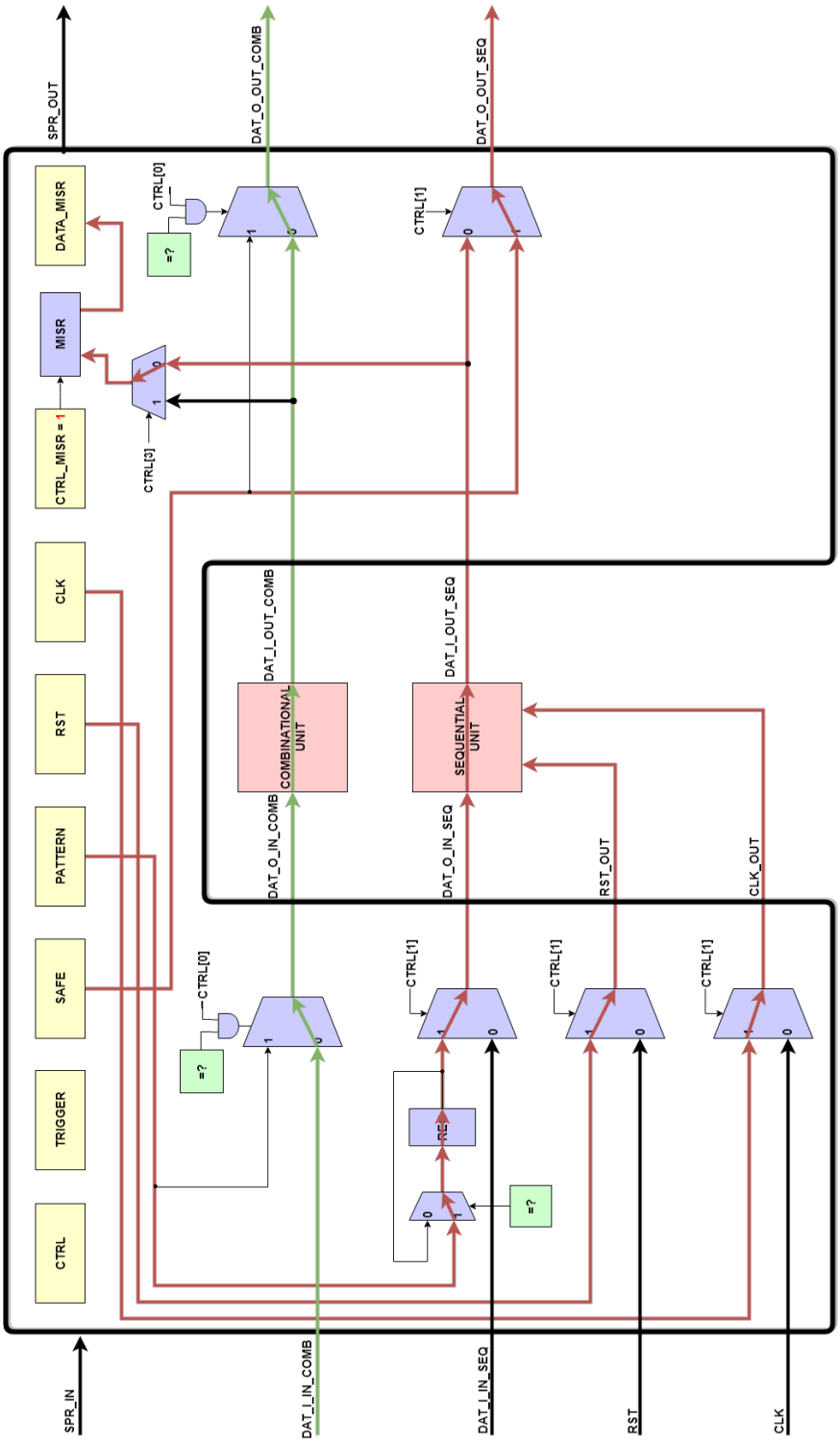


Figure 3.12: OPTIMUS with test pattern applied to the sequential unit

this approach, a higher number of faults can be detected by the test programs. At this point, starting from the faults which are not detected by the test programs, it is possible to ask to an ATPG tool to generate test patterns for the remaining faults. Thus, by exploiting the execution of test programs is possible to reduce the number of required test patterns to fully test the MUT.

A further strength of this approach lies when generating test patterns for a sequential circuit. Sequential ATPG hardly reaches high fault coverage figures, and it becomes harder and harder as the complexity of the circuit grows. Instead, by first applying the software self-test is possible to detect faults that require complex sequence of data, that an ATPG tool cannot generate. Hence, the sequential ATPG is used for generating sequential test patterns for the remaining faults, and it is extremely likely that the fault coverage would be higher than with a basic sequential ATPG.

In the next chapter the implemetation of OPTIMUS in a open-source processor and its consequences are discussed. Then results obtained with different test generation flow are presented.

Chapter 4

Case of Study and Experimental Results

The experiments were conducted on the *OpenRISC 1200* (OR1200) soft-core processor (Figure 4.1). The *OpenRISC 1200* (OR1200) is a 32-bit open-source soft-core processor [22], implementing a five stages integer pipeline. Its basic instruction pipeline fetches instructions from the memory subsystem, dispatches them to available execution units and maintains a state history to ensure a precise exception model and that operations finish in order. The OR1200 implements 32 *General Purpose Registers* (GPRs) of 32 bit width used for logical and arithmetical operation. It implements also various *Special Purpose Registers* (SPRs) used for special tasks or to permits the communication between the core and its peripherals. The architecture of the OR1200 is intended for embedded, portable and networking applications such as the automotive and portable computer environments.

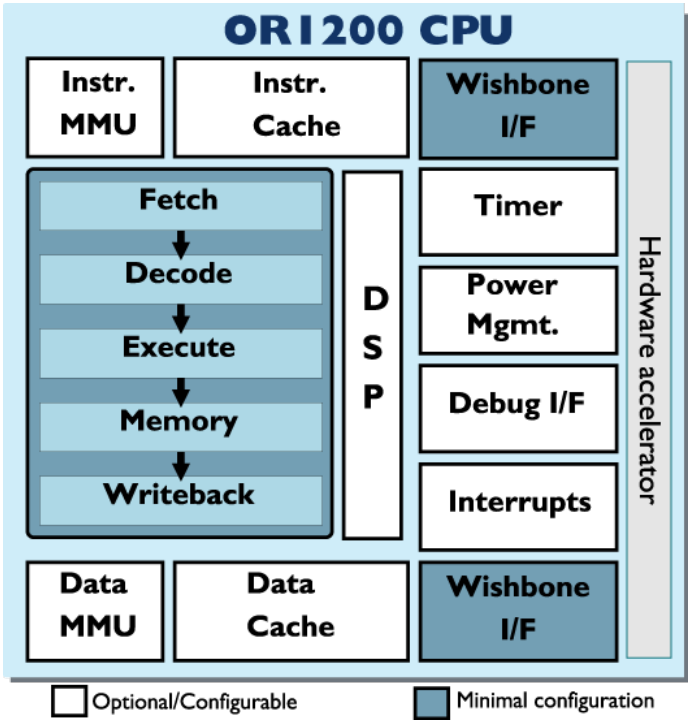


Figure 4.1: OR1200 CPU

All the logic simulation were performed using *Mentor Graphics ModelSim®(Questasim)*, whereas *Synopsys Design Compiler®* used as logic synthesis tool. Then *Synopsys TetraMAX®* was used as ATPG tool and as a fault simulation environment. Concerning the fault model, stuck-at faults were exclusively considered.

The only computational modules included within the OR1200 are the *Arithmetic Logic Unit* (ALU) and the *Multiply and Accumulate* (MAC). The former is a pure combinational module, in charge of executing basic logic and arithmetic operations. The latter is instead a complex sequential module, which is mainly used for multiplication, division and multiplication with accumulation of the result operations.

In the next section the specific implementation of OPTIMUS in the OR1200 is discussed. Then the results and the consequences obtained with the insertion of OPTIMUS are discussed.

4.1 OPTIMUS in the OR1200

Figure 4.2 clarifies how OPTIMUS was designed for the ALU and the MAC of the OR1200, also underlining the number of bits of the various signals and the 20 (numbered from 0 to 19) SPRs required. The ALU receives an amount of input signals of 120 bits and generates as output 36 bits, while the MAC receives 141 inputs bits, where two of them are the clock and the reset signal.

In Figure 4.3 and 4.4 the various inputs and outputs of the two modules are visible, while in Figure 4.5 and 4.6 is shown how these signals were grouped in single signals in order to handles the two modules with OPTIMUS.

For what concerns the SPRs, in the OR1200 processor the special purpose registers of all units are grouped into thirty-two groups. Each group can have different register address decoding depending on the theoretical number of registers in that particular group. In the OR1200 is present a unit called *or1200_sprs*, which is in charge to handle MTSPR and MFSPR operations. Starting from the 32 bits of the spr addr it decodes the group number and generates the spr_cs signal. In this modules this signal is on 32 bit, but the various module with SPR interfaces use only a single bit of this signal: the

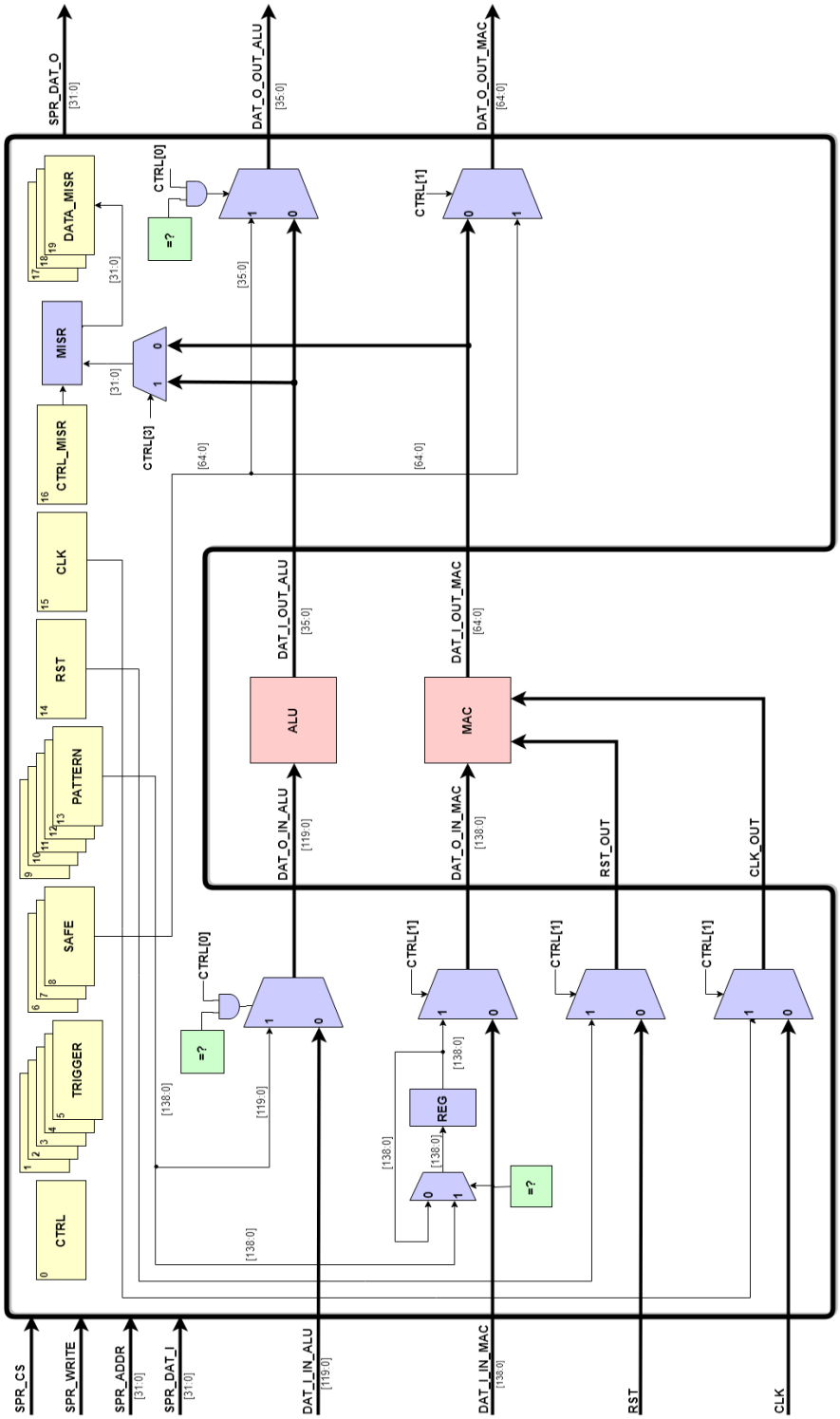


Figure 4.2: OPTIMUS in the OR1200 processor

```
input             clk;
input             rst;
input             ex_freeze;
input             id_macrc_op;
input             macrc_op;
input  [32-1:0]   a;
input  [32-1:0]   b;
input  [2-1:0]    mac_op;
input  [4-1:0]    alu_op;
output [32-1:0]   result;
output            mac_stall_r;
```

Figure 4.3: Inputs and outputs of OPTIMUS

```
input  [32-1:0]   a;
input  [32-1:0]   b;
input  [32-1:0]   mult_mac_result;
input             macrc_op;
input  [4-1:0]    alu_op;
input  [2-1:0]    shrot_op;
input  [4-1:0]    comp_op;
input  [5-1:0]    cust5_op;
input  [6-1:0]    cust5_limm;
output [32-1:0]   result;
output            flagforw;
output            flag_we;
output            cyforw;
output            cy_we;
input             carry;
input             flag;
```

Figure 4.4: Inputs and outputs of OPTIMUS

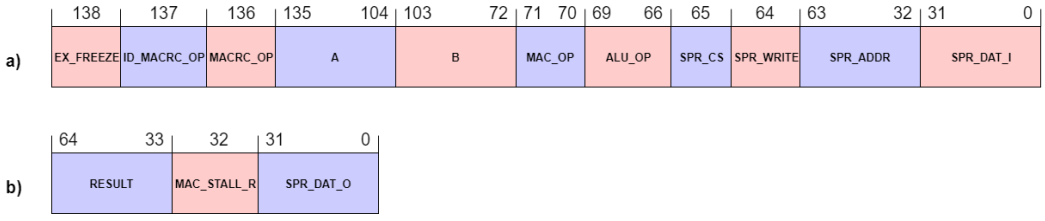


Figure 4.5: Grouping of the MAC signals to form one input of 139 bits and one output of 65 bits

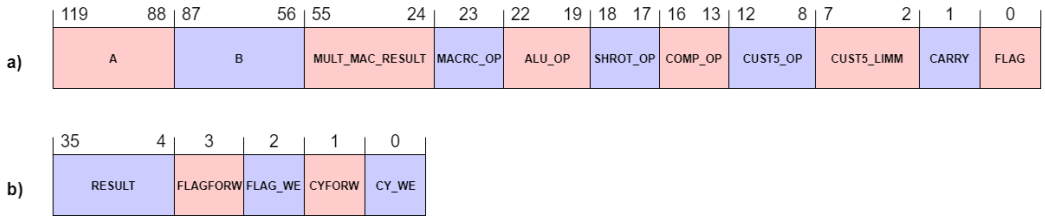


Figure 4.6: Grouping of the ALU signals to form one input of 120 bits and one output of 36 bits

unit of group 0 the spr_cs[0], the group 1 spr_cs[1] and so on. The 5-bit group index (bits [15:11]) of the spr_addr signal defines the number of the group, while the 11-bit register index (bits [10:0]) defines the address of the SPR register of a group. For this reason theoretically the maximum number of SPRs that a group can have are $2^{11} = 2{,}048$.

For the MFSPR operations, the *or1200_sprs* unit, is also in charge of select the correct value from all the SPRs of the various units. For this reason the unit has been modified in order to consider also the data coming from OPTIMUS (Figure 4.7).

Moreover the OR1200 processor has predefined number of groups for the various units and the groups 24-31 are left for custom units. For this reason to OPTIMUS has been assigned the group 24 and, with the facts that it has 20 SPR (numbered from 0 to 19) registers, it use 5 bits of address. Hence the spr_addr used by OPTIMUS expressed as hexadecimal number is in the range 0xc000-0xc013 (Figure 4.8).

Table 4.1 shows the 20 SPRs, also giving information about their number, bit size, if they are readable (R), writable (W) or both (R/W) and the type of the category they belong to, among the six described in the previous chapter.

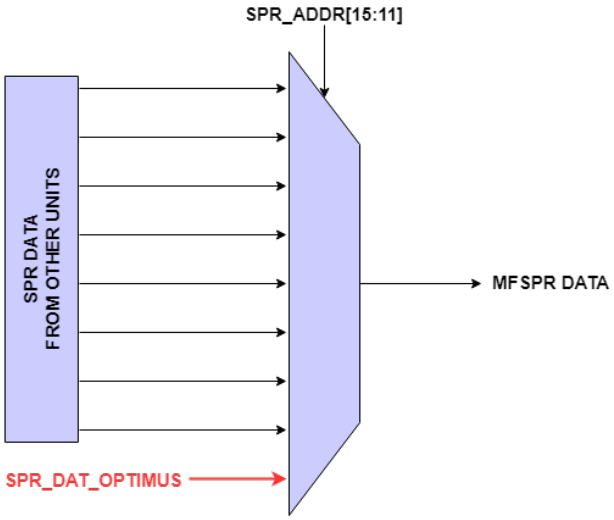


Figure 4.7: Addition of SPR data coming from OPTIMUS in the OR1200 SPRS unit

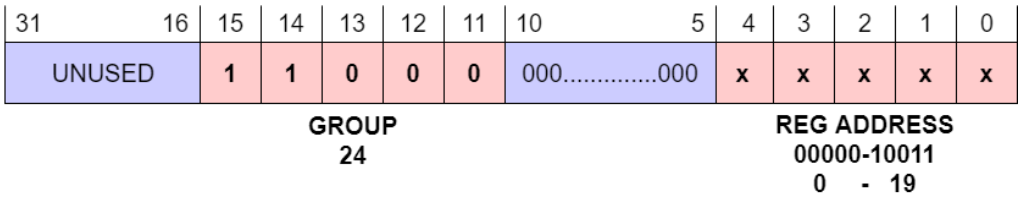


Figure 4.8: SPR_ADDR for OPTIMUS

NUMBER	NAME	SIZE	ACCESS	CATEGORY
0	reg0_ctrl_contr	3	R/W	CONTROL
1	reg1_trigger	32	R/W	TRIGGER
2	reg2_trigger	32	R/W	TRIGGER
3	reg3_trigger	32	R/W	TRIGGER
4	reg4_trigger	32	R/W	TRIGGER
5	reg5_trigger	11	R/W	TRIGGER
6	reg6_safe	32	R/W	SAFE
7	reg7_safe	32	R/W	SAFE
8	reg8_safe	1	R/W	SAFE
9	reg9_pattern	32	R/W	PATTERN
10	reg10_pattern	32	R/W	PATTERN
11	reg11_pattern	32	R/W	PATTERN
12	reg12_pattern	32	R/W	PATTERN
13	reg13_pattern	11	R/W	PATTERN
14	reg14_rst	1	R/W	RESET & CLOCK
15	reg15_clk	1	R/W	RESET & CLOCK
16	reg16_ctrl_misr	1	R/W	MISR
17	reg17_data_misr	32	R	MISR
18	reg18_data_misr	32	R	MISR
19	reg19_data_misr	1	R	MISR

Table 4.1: SPRs of OPTIMUS

In order to support the SPR interface of the OR1200, OPTIMUS needs these 5 signals:

- *spr_cs*: it is a chip select signal, if active it means that the unit has been select and write operations could be possible. This signal is an input of OPTIMUS;
- *spr_write*: this signal enable the write operation, if both the chip select and this signal are active, supposing that a valid spr_addr is given, a register is written. This signal is an input of OPTIMUS;
- *spr_addr*: it indicates the address of the register to be written or read. This signal is an input of OPTIMUS;
- *spr_dat_i*: it is the value that has to be written in the SPR register. This signal is an input of OPTIMUS;
- *spr_dat_o*: in case of read operation, this signal presents the value of the SPR register. This signal is an output of OPTIMUS.

4.2 Post-Synthesis Results

The entire system (namely the CPU including OPTIMUS) was synthesized using the *NandGate 45nm* technology library. Table 4.2 reports the post-synthesis results of the two CPUs: the original OR1200 and the one with OPTIMUS implemented. It can be noticed that, since OPTIMUS architecture is shared between the ALU and the MAC, it is possible to achieve a

reasonable 20% of area overhead. As an example, if OPTIMUS was implemented only for the MAC unit, the area overhead would have been around 15%. Concerning the timing, OPTIMUS introduces a performance degradation of 5%, with a slightly reduction of the achievable operating frequency (the clock period increases of about 0.14 ns).

DESIGN TYPE	AREA [µm^2]	Cycle Time [ns]
OR1200 CPU with OPTIMUS	3382.54	2.62
OR1200	27057.00	2.48

Table 4.2: Designs comparison

Table 4.3 shows more in detail the area occupied by OPTIMUS. Nearly half of the total area is devoted to the registers (SPRs and MISR), while the other half stems from the Combinational Logic (CL). A considerable portion of the CL is devoted for implementing the various multiplexers (needed for applying the test patterns to the Primary Inputs, as well as the safe output values as Primary Outputs) and the comparator used to check the presence of the Trigger Instruction. The remaining area implements the processor SPR Interface.

MODULE		AREA[µm^2]	TOTAL AREA [%]
Registers		3287.76	9.7
CL	SPR Interface	1051.49	3.1
	Muxes & Comparator	2436.29	7.2

Table 4.3: Area Breakdown

4.3 Self-Test Programs Effectiveness and Characteristics

In order to verify the effectiveness of the proposed test generation flow, it was performed first a fault simulation using an in-house STL, developed targeting the on-line test of the CPU. Basically each test program computes internally a signature, which is then stored in memory at the end of the test program execution. The generation of ATPG patterns started from the fault list obtained from the execution of the STL on the ALU and on the MAC. In order to do this, a fault simulation tool (TetraMAX®) is used. It receives the netlist of the OR1200, generates the ATPG patterns for the considered unit and writes them in a *.stil* file. Then this file is read by a program written in Python in order to convert the patterns in instructions that the processor has to execute (Figure 4.9).

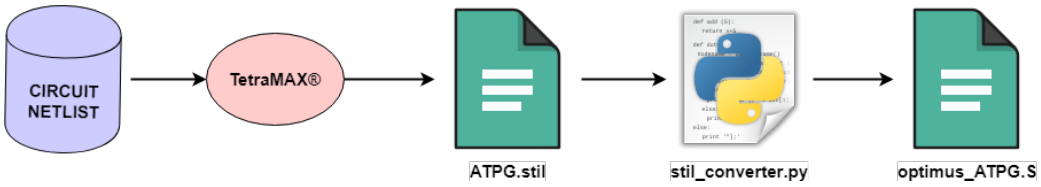


Figure 4.9: Generation of a test program for OPTIMUS

In figure 4.10 and 4.11 is shown a typical structure of a .stil file and of a test program of OPTIMUS for a combinational circuit and for a sequential one respectively. In both cases, first of all the ctrl and the ctrl_misr are configured, then the Trigger and Safe SPRs are uploaded with the wanted values. Then there is a sequence of loads in the Pattern SPR and Trigger Instructions

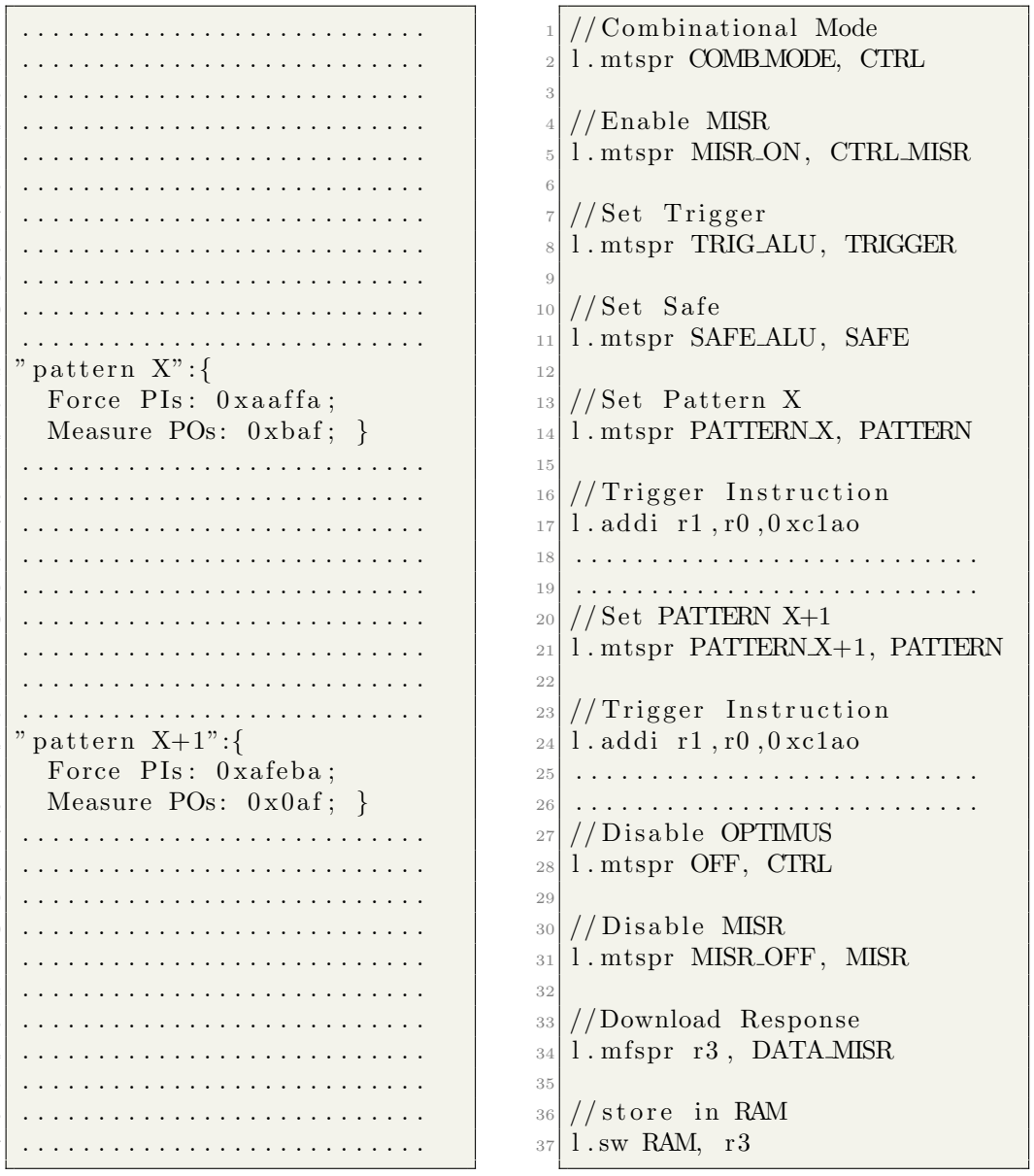
that invoke the application of the test patterns. Here, in case a sequential unit is tested, could appear also sequences of clock and reset pulses. At the end OPTIMUS is turned off and the internal MISR is disabled, the value of the data_misr SPRs is saved into some general purpose register and finally, using a store in memory instructions it is propagated to the processor output.

It is important to notice that, the fault coverage of the STL, with the internal MISR of OPTIMUS activated, was computed before the actual generation of test patterns. Table 4.4 shows the fault coverage obtained for the two modules using the following approaches:

- *Pure STL*: the modules are tested using the original STL that does not consider the implementation of OPTIMUS (internal MISR not enabled).
- *STL+MISR*: the two modules are tested using the same STL but here OPTIMUS is considered and the MISR is enabled.
- *OPTIMUS ATPG Test*: starting from the result acquired with the MISR activated, the modules are tested using the test program obtained from the generation of the ATPG patterns for the remaining faults.

UUT	PURE STL [%]	STL+MISR [%]	OPTIMUS ATPG TEST [%]
ALU	94.98	95.49	99.51
MAC	94.16	95.3	98.08

Table 4.4: Fault Coverage with different strategies



```
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
"pattern X":{
  Force PIs: 0xaaffa;
  Measure POs: 0xbaf; }
.....
.....
.....
.....
.....
.....
.....
.....
"pattern X+1":{
  Force PIs: 0xafeba;
  Measure POs: 0x0af; }
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
```

```
//Combinational Mode
l.mtspr COMB_MODE, CTRL

//Enable MISR
l.mtspr MISR_ON, CTRL_MISR

//Set Trigger
l.mtspr TRIG_ALU, TRIGGER

//Set Safe
l.mtspr SAFE_ALU, SAFE

//Set Pattern X
l.mtspr PATTERN_X, PATTERN

//Trigger Instruction
l.addi r1,r0,0xc1ao
.....
.....
//Set PATTERN X+1
l.mtspr PATTERN_X+1, PATTERN

//Trigger Instruction
l.addi r1,r0,0xc1ao
.....
.....
//Disable OPTIMUS
l.mtspr OFF, CTRL

//Disable MISR
l.mtspr MISR_OFF, MISR

//Download Response
l.mfspr r3, DATA_MISR

//store in RAM
l.sw RAM, r3
```

Figure 4.10: Structure of a .stil file and its transformation in test program for a combinational circuit

```
1  ..........................
2  ..........................
3  ..........................
4  ..........................
5  ..........................
6  ..........................
7  ..........................
8  ..........................
9  ..........................
10 ..........................
11 ..........................
12 "pattern X":{
13   Force PIs: 0xdead;
14   Measure POs: 0xbeef;
15   "clk"=Pulse;
16   Measure POs: 0xcafe;}
17 ..........................
18 ..........................
19 ..........................
20 ..........................
21 ..........................
22 ..........................
23 ..........................
24 ..........................
25 ..........................
26 ..........................
27 ..........................
28 "pattern X+1":{
29   Force PIs: 0xb01a;
30   Measure POs: 0xcafe;
31   "rst"=Pulse;
32   Measure POs: 0xc1a0;}
33 ..........................
34 ..........................
35 ..........................
36 ..........................
37 ..........................
38 ..........................
39 ..........................
40 ..........................
41 ..........................
42 ..........................
43 ..........................
```

```
1  //Sequential Mode
2  l.mtspr SEQ_MODE, CTRL
3
4  //Enable MISR
5  l.mtspr MISR_ON, CTRL_MISR
6
7  //Set Trigger
8  l.mtspr TRIG_MAC, TRIGGER
9
10 //Set Safe
11 l.mtspr SAFE_MAC, SAFE
12
13 //Set ATPG X
14 l.mtspr ATPG_PATTERN_X, ATPG
15
16 //Trigger Instruction
17 l.muli r1,r0,0xeffe
18
19 //CLK Pulse
20 l.mtspr PULSE, CLK
21 ..........................
22 ..........................
23 //Set ATPG X+1
24 l.mtspr ATPG_PATTERN, ATPG
25
26 //Trigger Instruction
27 l.muli r1,r0,0xeffe
28
29 //RST Pulse
30 l.mtspr PULSE, RST
31 ..........................
32 ..........................
33 //Disable OPTIMUS
34 l.mtspr OFF, CTRL
35
36 //Disable MISR
37 l.mtspr MISR_OFF, MISR
38
39 //Download Response
40 l.mfspr r3, DATA_MISR
41
42 //store in RAM
43 l.sw RAM, r3
```

Figure 4.11: Structure of a .stil file and its transformation in test program for a sequential circuit

Table 4.5 shows instead the principal characteristics (execution time in clock cycle and size in bytes) of the test programs used: the basic STL and the two test programs generated for the applications of the ATPG patterns for the ALU and the MAC.

TEST PROGRAM	DURATION[CC]	SIZE[BYTES]
STL	49,590	30,296
ALU Self-Test	10,268	20,556
MAC Self-Test	11,996	24,012

Table 4.5: Self-test programs characteristics

A total of 124 test patterns were generated for the ALU, while 43 full sequential patterns for the MAC. For sake of completeness, it was computed also the size and the execution time of the self-test procedure, which applies patterns generated from scratch, that is without first exploiting the fault coverage achieved with an STL. For the ALU 282 patterns were generated while 146 full sequential patterns for the MAC. Although the ALU still reached more than 99% of fault coverage, while for the MAC the ATPG was not able to go beyond the 97% of fault coverage. The ALU and MAC patterns were translated into self-test procedures. For the ALU the test program execution time was of 23,224 clock cycle and the size in bytes of almost 46,468 bytes. For the MAC the self-test program results in an execution time of 41,022 clock cycles and a size of 82,028 bytes. How is possible to notice with these data, not starting from a previous result of fault simulation and then generate the ATPG pattern, comports a higher number of required pattern and a longer and greater test program.

4.4 OPTIMUS FMEDA

When dealing with safety-critical applications, it is important to verify the safeness of the additional hardware introduced within the CPU. The introduction of OPTIMUS comports a reduction of the reliability of the entire system. Consequently, a *Failure Mode Effect and Diagnostic Analysis* (FMEDA) was performed. The focus of this analysis was oriented to the identification of possible critical failure modes and identifications of faults whose occurrence lead to the such failure.

Faults inside OPTIMUS become relevant exclusively when the module is not supposed to be active, which is during the normal operation of the user code. Then, two possible *Failure Modes* (FM) were identified:

- *FM1*: also if OPTIMUS is not supposed to be active, the Primary Inputs of the module under test are replaced;
- *FM2*: again, OPTIMUS is not active but the Primary Outputs of a tested circuit are replaced with the safe patterns;

Trough faults injection campaigns, the faults leading to these two FM were first identified. A set of application programs were fault simulated with faults injected within OPTIMUS only (for a total of 27,468 stuck-at faults). After the fault simulation process, concerning FM1, 10% of critical faults were identified. As countermeasures, two options are possible: the first one concerns the usage of the self-test routines developed for ALU and MAC that exploit OPTIMUS. By adopting this strategy, the 99.67% of such critical

faults were detected. The second choice concerns selective hardening of the gates causing the FM1, using a DWC configuration for each gate. This option yields higher reaction time to the failures (the DWC allows for immediate detection) while maintaining the overhead low. By using the latter solution, the OPTIMUS area overhead moves from 20% to 24.23%. It is worth noting that if a classical DWC configuration was used (i.e., all the logic duplicated), the overhead would have been beyond the 40%. For FM2, 2% of faults lead to this specific failure mode. Since these faults cause MUT POs to be replaced with the safe output, they can be hardly identified with the already existing self-test routines. Also in this case, the best option would be the selective hardening of the gate causing this failure (area overhead around 21.35%). When adopting selective DWC both failure modes (clearly, gates causing both FM1 and FM2 are replicated only once), the area overhead reaches 25.38%.

4.5 Further Analyses

As discussed, not starting the generation of the ATPG patterns from the results obtained with a STL, could lead to high number of test pastern required and big test program (for the point of view of the execution time and the size). But it also true that often the creation of a good STL requires a lot of time, time that could stretch the *Time To Market* (TTM) of a product. For this reason in this chapter is discussed if OPTIMUS could be used in order to substitute the STL and reduce the TTM. The analysis was conducted on the ALU and on the MAC using as test vector random numbers, hence values

that did not require effort to be found but that are able to detect only a low number of faults.

4.5.1 Analyses on the ALU

The experiments were conducted using OPTIMUS on the ALU starting from zero (then full ATPG), five, ten, twenty and thirty random test patterns. In Figure 4.12 is possible to notice how, also if the number of test pattern was augmented, the fault coverage obtained with these random test pattern without activating OPTIMUS, was saturating around a value of 87%. Hence no other random patterns were used.

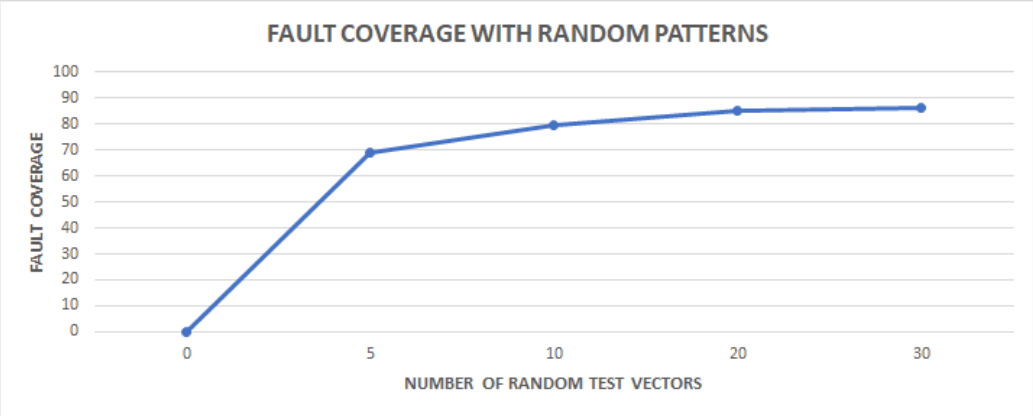


Figure 4.12: Fault coverage obtained with random test vectors for the ALU

Then, knowing that using all these test programs with the activation of the internal MISR of OPTIMUS first and the usage of the ATPG patterns then, was possible to achieve almost the same Fault Coverage (around 99.4%), analyses on the required test pattern and the characteristics of the test programs were conducted.

Analysing Figures 4.13, 4.14 and 4.15 is possible to notice that higher are the number of random test vectors used, lower is the number of ATPG pattern, specially using the STL. Then also the duration and the size of the test program of OPTIMUS requires less effort. But if combined with the characteristics of the random vector programs there is the risk to obtain value that are worse than the one obtained with a STL.

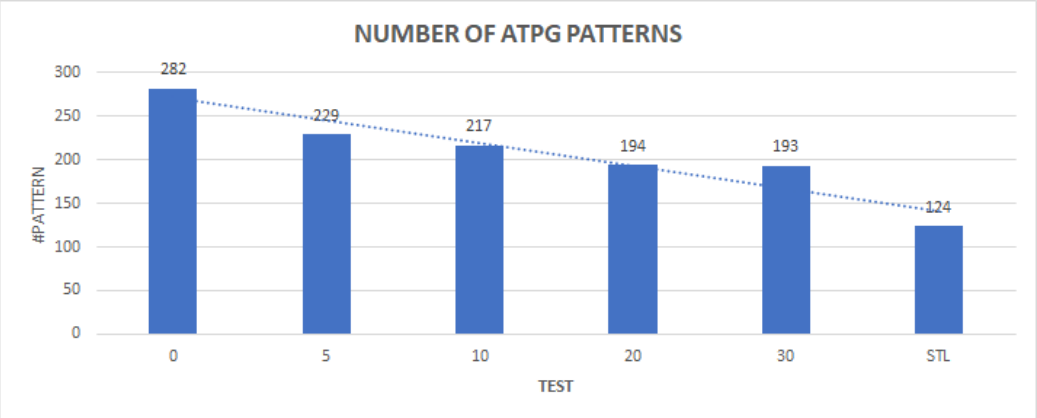


Figure 4.13: Number of ATPG patterns generated using a full ATPG approach, using different numbers of random test vectors or using the STL for the ALU

These results tell that, from what concerns a combinational circuit, OPTIMUS through a full ATPG test process could be used as a substitution of the STL. This because, also if it requires a higher number of ATPG patterns, the execution time and the size of the test programs are lower than the ones required using a STL+OPTIMUS approach.

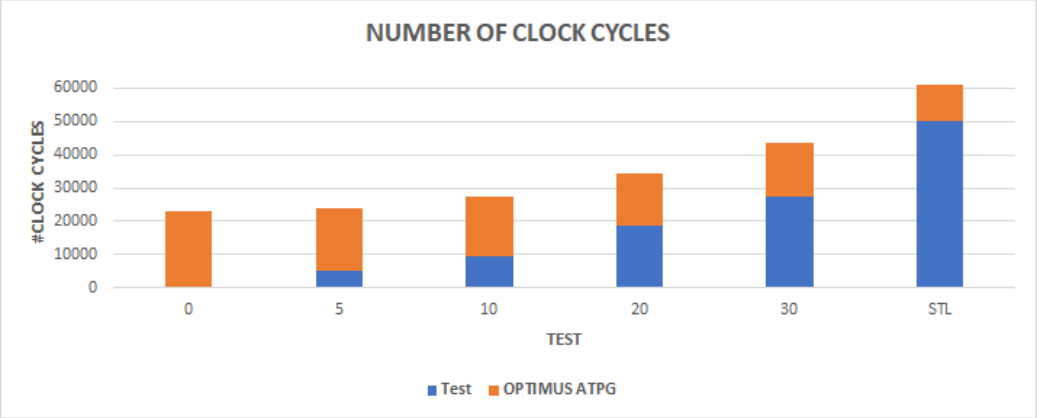


Figure 4.14: Number of clock cycles required from the different test programs to test the ALU. The orange color is used for the test program of OPTIMUS, while the blue color for the normal test program. The height is the sum of the two methods

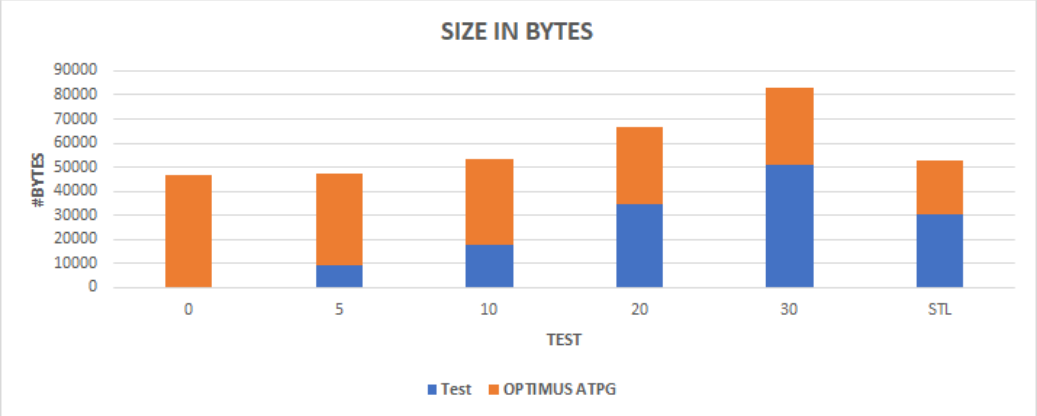


Figure 4.15: Size in bytes required from the different test programs to test the ALU. The orange color is used for the test program of OPTIMUS, while the blue color for the normal test program. The height is the sum of the two methods

4.5.2 Analyses on the MAC

Again, as done for the ALU, is possible to notice in Figure 4.16 that with thirty random test vectors the fault coverage of the MAC is saturated at a value around 75%.

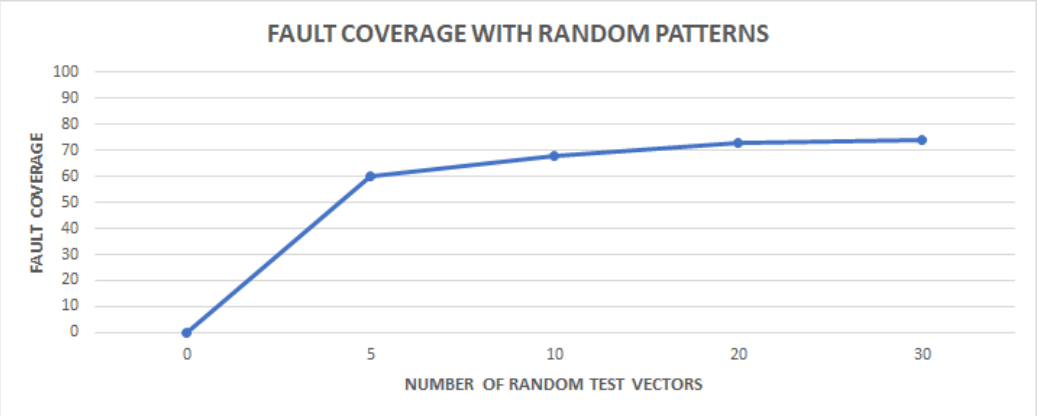


Figure 4.16: Fault coverage obtained with random test vectors for the MAC

Again with all the different test strategies almost the same fault coverage of 97% has been achieved. In Figures 4.17, 4.18 and 4.19 are shown the obtained results. As expected a full ATPG approach requires the highest number of ATPG patterns, while with the STL the lowest. As the ALU the clock cycles required from the full ATPG method and the random test programs+OPTIMUS is still lower than the one needed by the STL. Instead the size of the test program is greater that the one of the STL. For these reasons OPTIMUS could replace the STL only if there is no problem of memory space in order to save the test program.

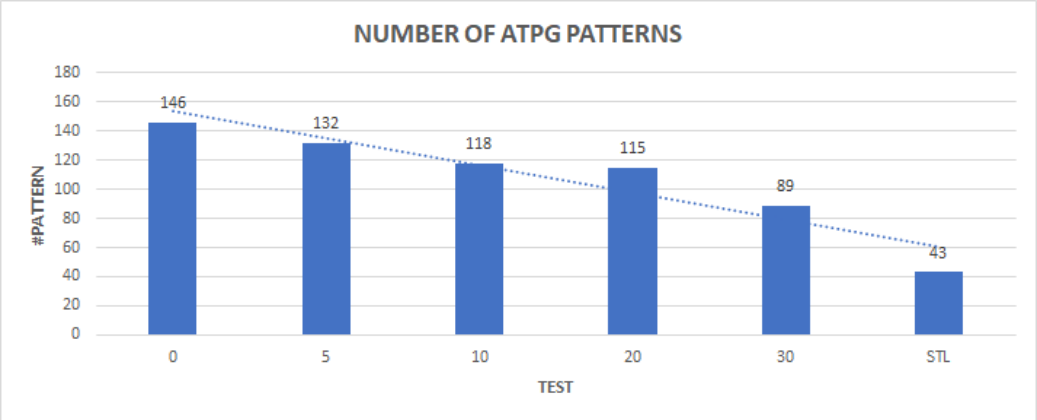


Figure 4.17: Number of ATPG patterns generated using a full ATPG approach, using different numbers of random test vectors or using the STL on the MAC

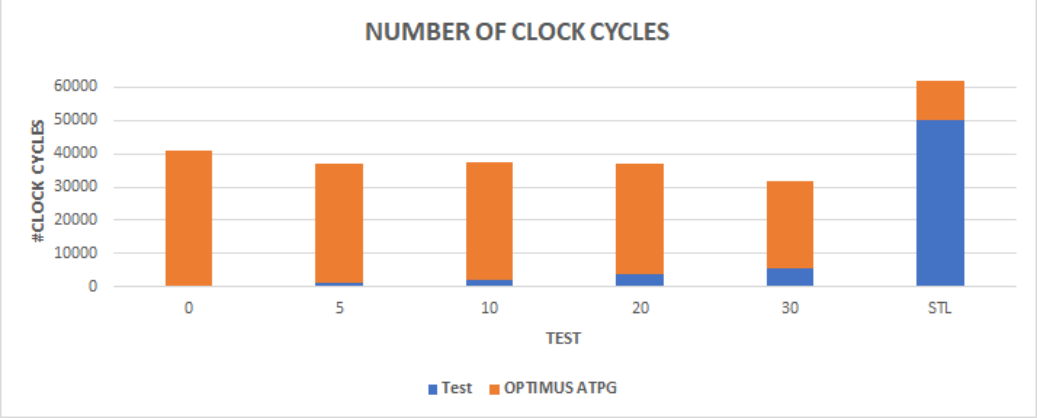


Figure 4.18: Number of clock cycles required from the different test programs to test the MAC. The orange color is used for the test program of OPTIMUS, while the blue color for the normal test program. The height is the sum of the two methods

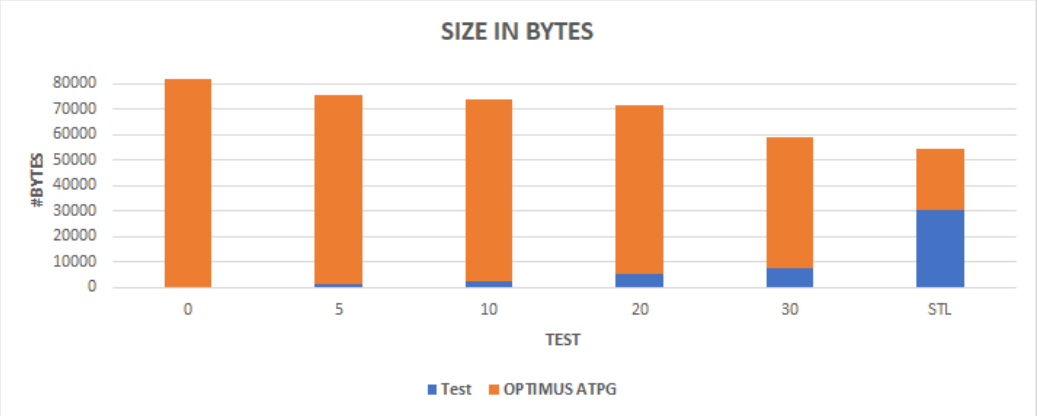


Figure 4.19: Size in bytes required from the different test programs to test the MAC. The orange color is used for the test program of OPTIMUS, while the blue color for the normal test program. The height is the sum of the two methods

Chapter 5

Conclusions and Future Works

In this work OPTIMUS has been presented, a hybrid approach for the on-line test of computational modules composed of a programmable DfT module, accessible via software, which works in conjunction with a software-based test program. The main advantages of this proposed architecture are:

1. the minimum interference with the other modules embedded within the processor core during the self-test;

2. the capability of achieving high fault coverage of computational modules during the on-line test.

The most relevant improvements of this work could be:

- to modify OPTIMUS in order to be used with more complex processors (i.e. superscalar processor);

- consider different fault models (i.e delay fault);

- evaluate the usage of OPTIMUS with other modules of the processor (i.e. the decoder or the control unit)

Bibliography

[1] A. Nardi and A. Armato, "Functional safety methodologies for automotive applications," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2017, pp. 970–975.

[2] E. Fujiwara, *Code Design for Dependable Systems: Theory and Practical Applicationn*. New York, NY, USA: Wiley-Interscience, 2006.

[3] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 124–134, March 1984.

[4] G. Tshagharyan, G. Harutyunyan, and Y. Zorian, "An effective functional safety solution for automotive systems-on-chip," in *2017 IEEE International Test Conference (ITC)*, Oct 2017, pp. 1–10.

[5] T. McLaurin, "Periodic online lbist considerations for a multicore processor," in *2018 IEEE International Test Conference in Asia (ITC-Asia)*, Aug 2018, pp. 37–42.

[6] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Publishing Company, Incorporated, 2013.

[7] S. M. Thatte and J. A. Abraham, "Test generation for microprocessors," *IEEE Transactions on Computers*, vol. C-29, no. 6, pp. 429–441, June 1980.

[8] A. Paschalis, D. Gizopoulos, N. Kranitis, M. Psarakis, and Y. Zorian, "Deterministic software-based self-testing of embedded processor cores," in *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, March 2001, pp. 92–96.

[9] A. Jasnetski, R. Ubar, and A. Tsertov, "On automatic software-based self-test program generation based on high-level decision diagrams," in *2016 17th Latin-American Test Symposium (LATS)*, April 2016, pp. 177–177.

[10] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. Sonza Reorda, "Microprocessor software-based self-testing," *IEEE Design Test of Computers*, vol. 27, no. 3, pp. 4–19, May 2010.

[11] P. Bernardi, R. Cantoro, S. De Luca, E. Snchez, and A. Sansonetti, "Development flow for on-line core self-test of automotive microcontrollers," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 744–754, March 2016.

[12] N. Kranitis, A. Merentitis, G. Theodorou, A. Paschalis, and D. Gizopoulos, "Hybrid-sbst methodology for efficient testing of processor cores," *IEEE Design Test of Computers*, vol. 25, no. 1, pp. 64–75, Jan 2008.

[13] L. Chen, S. Ravi, A. Raghunathan, and S. Dey, "A scalable software-based self-test methodology for programmable processors," in *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, June 2003, pp. 548–553.

[14] E. Sanchez, "Increasing reliability of safety critical applications through functional based solutions," in *2018 13th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS)*, April 2018, pp. 1–1.

[15] A. Floridia and E. Sanchez, "Hybrid on-line self-test strategy for dual-core lockstep processors," in *2018 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct 2018, pp. 1–6.

[16] P. Bernardi, L. M. Ciganda, E. Sanchez, and M. S. Reorda, "Mihst: A hardware technique for embedded microprocessor functional on-line self-test," *IEEE Transactions on Computers*, vol. 63, no. 11, pp. 2760–2771, Nov 2014.

[17] P. Bernardi, R. Cantoro, L. Gianotto, M. Restifo, E. Sanchez, F. Venini, and D. Appello, "A dma and cache-based stress schema for burn-in of automotive microcontroller," in *2017 18th IEEE Latin American Test Symposium (LATS)*, March 2017, pp. 1–6.

[18] T. F. Hsieh, J. F. Li, K. T. Wu, J. S. Lai, C. Y. Lo, D. M. Kwai, and Y. F. Chou, "Software-hardware-cooperated built-in self-test scheme for channel-based drams," in *2017 International Test Conference in Asia (ITC-Asia)*, Sep. 2017, pp. 107–111.

[19] W. C. Lai and K. T. Cheng, "Instruction-level dft for testing processor and ip cores in system-on-a-chip," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, June 2001, pp. 59–64.

[20] M. Nakazato, S. Ohtake, M. Inoue, and H. Fujiwara, "Design for testability of software-based self-test for processors," in *2006 15th Asian Test Symposium*, Nov 2006, pp. 375–380.

[21] K. Batcher and C. Papachristou, "Instruction randomization self test for processor cores," in *Proceedings 17th IEEE VLSI Test Symposium (Cat. No.PR00146)*, April 1999, pp. 34–40.

[22] [Online]. Available: https://openrisc.io/.