UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MARCOS TOMAZZOLI LEIPNITZ

Resilient Regular Expression Matching on FPGAs with Fast Error Repair

Dissertation presented in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Dr. Gabriel Luca Nazar

Porto Alegre
2017

CIP – CATALOGING-IN-PUBLICATION

Leipnitz, Marcos Tomazzoli

Resilient Regular Expression Matching on FPGAs with Fast Error Repair / Marcos Tomazzoli Leipnitz. – 2017.

75 f.

Advisor: Gabriel Luca Nazar.

Dissertation (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2017.

1. Field-Programmable Gate Array. 2. Network Function Virtualization. 3. Regular Expression Matching. 4. Fault-Tolerance. 5. Repair Time. I. Nazar, Gabriel Luca. II. Title.

ABSTRACT

The Network Function Virtualization (NFV) paradigm promises to make computer networks more scalable and flexible by decoupling the network functions (NFs) from dedicated and vendor-specific hardware. However, network and compute intensive NFs may be difficult to virtualize without performance degradation. In this context, Field-Programmable Gate Arrays (FPGAs) have been shown to be a good option for hardware acceleration of virtual NFs that require high throughput, without deviating from the concept of an NFV infrastructure which aims at high flexibility. Regular expression matching is an important and compute intensive mechanism used to perform Deep Packet Inspection, which can be FPGA-accelerated to meet performance constraints. This solution, however, introduces new challenges regarding dependability requirements. Particularly for SRAM-based FPGAs, soft errors on the configuration memory are a significant dependability threat. In this work we present a comprehensive fault tolerance mechanism to deal with configuration faults on the functionality of FPGA-based regular expression matching engines. Moreover, a placement-aware scrubbing mechanism is introduced to reduce the system repair time, improving the system reliability and availability. Experimental results show that the overall failure rate and the system mean time to repair can be reduced in 95% and 90%, respectively, with manageable area and performance costs.

Keywords: Field-Programmable Gate Array. Network Function Virtualization. Regular Expression Matching. Fault-Tolerance. Repair Time.

Avaliação Resiliente de Expressões Regulares em FPGAs com Rápida Correção de Erros

RESUMO

O paradigma *Network Function Virtualization* (NFV) promete tornar as redes de computadores mais escaláveis e flexíveis, através do desacoplamento das funções de rede de hardware dedicado e fornecedor específico. No entanto, funções de rede computacionalmente intensivas podem ser difíceis de virtualizar sem degradação de desempenho. Neste contexto, *Field-Programmable Gate Arrays* (FPGAs) têm se mostrado uma boa opção para aceleração por hardware de funções de rede virtuais que requerem alta vazão, sem se desviar do conceito de uma infraestrutura NFV que visa alta flexibilidade. A avaliação de expressões regulares é um mecanismo importante e computacionalmente intensivo, usado para realizar *Deep Packet Inpection*, que pode ser acelerado por FPGA para atender aos requisitos de desempenho. Esta solução, no entanto, apresenta novos desafios em relação aos requisitos de confiabilidade. Particularmente para FPGAs baseados em SRAM, *soft errors* na memória de configuração são uma ameaça de confiabilidade significativa. Neste trabalho, apresentamos um mecanismo de tolerância a falhas abrangente para lidar com falhas de configuração na funcionalidade de módulos de avaliação de expressões regulares baseados em FPGA. Além disso, é introduzido um mecanismo de correção de erros que considera o posicionamento desses módulos no FPGA para reduzir o tempo de reparo do sistema, melhorando a confiabilidade e a disponibilidade. Os resultados experimentais mostram que a taxa de falha geral e o tempo de reparo do sistema podem ser reduzidos em 95% e 90%, respectivamente, com custos de área e performance admissíveis.

Palavras-chave: *Field-Programmable Gate Array*. *Network Function Virtualization*. Avaliação de Expressões Regulares. Tolerância a Falhas. Tempo de Reparo.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ASIC	Application Specific Integrated Circuit
AUT	Area Under Test
BRAM	Block Random Access Memory
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
COTS	Commercial Of-The-Shelf
CRC	Cyclic Redundancy Check
CUT	Circuit Under Test
DFA	Deterministic Finite Automata
DMR	Dual Modular Redundancy
DWC	Duplication With Comparison
DPI	Deep Packet Inspection
DPR	Dynamic Partial Reconfiguration
DSP	Digital Signal Processor
ECC	Error Correcting Code
EDAC	Error Detection and Correction
FF	Flip-Flop
FIT	Failures In Time
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
ICAP	Internal Configuration Access Port
IOB	Input/Output Block
JTAG	Joint Test Action Group
LET	Linear Energy Transfer
LUT	Look-Up Table
MBU	Multi-Bit Upset
MTTR	Mean Time To Repair
NAT	Network Address Translation
NF	Network Function
NFA	Nondeterministic Finite Automata
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure

NIDS	Network Intrusion Detection System
NIPS	Network Intrusion Prevention System
REM	Regular Expression Matching
REME	Regular Expression Matching Engine
SBU	Single-Bit Upset
SEE	Single-Event Upset
SEL	Single-Event Latch-up
SER	Soft Error Rate
SET	Single-Event Transient
SEU	Single-Event Upset
SRAM	Static Random Access Memory
ST	Signature Translator
SUM	State Update Module
TMR	Triple Modular Redundancy

SUMMARY

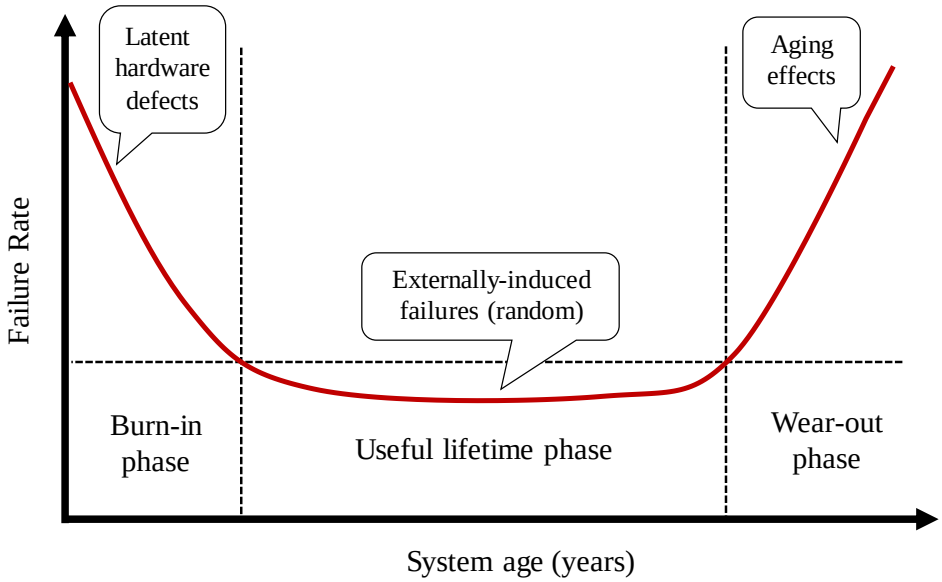
1 INTRODUCTION

The technology scaling of Complementary Metal Oxide Semiconductor (CMOS) devices have enabled an exponential growth in the number of transistors per chip (popularly known as Moore's law (MOORE, 1965)), leveraging the progress in the performance and functionality of modern computing systems. However, the decreasing of transistor's feature sizes to nanoscale and the aggressive voltage operation reduction also increases the device sensitivity to transient faults, i.e., to hardware defects that cause a component to malfunction in a limited period of time, triggered by environment disturbances such as electromagnetic interference or radiation. As a result, the soft error rates (SER) observed in modern systems have increased as the manufacturing technology evolves (DIXIT; WOOD, 2011; HUBERT; ARTOLA; REGIS, 2015).

A soft error is characterized by a manifestation of underlying transient faults, such as the inversion of the logic state in a memory cell (bit-flip) or an incorrect result in a combinational logic operation. Although soft errors do not damage the device, and therefore can be detected and corrected, the propagation of errors through the system circuitry can lead to functional failures, i.e., to a deviation of the system intended function. In other words, a failure is an error showing up at a boundary where it becomes visible to the user of the system. In some scenarios, these deviations are unacceptable, such as a change in a bank account or in other systems with high dependability constraints (MUKHERJEE, 2008).

The failure rate is an important parameter used to evaluate the reliability of a given hardware system, and depends not just on its sensitiveness to external disturbances, as mentioned, but also on other factors such as manufacturing process quality, temperature and voltage variations, technology, and age. The dependence with age is usually characterized by the bathtub curve, as illustrated in Figure 1.1 (KOREN; KRISHNA, 2007). In the early life of a system, the failure rate is typically high due to weak or defective components slipped from the manufacturing quality control, but decreases over time as the faulty components are removed (burn-in phase). After this initial period, the system reaches its useful lifetime phase, during which externally-induced failures occur randomly and a fairly constant failure rate can be observed. In the end of the system useful life, the failure rate starts to rise again because of aging effects (wear-out phase). The incidence of ionizing particles in radiation-harsh environments is a major concern during the system useful lifetime, even for terrestrial

Figure 1.1 – Bathtub curve characterizing the failure rate of a hardware system during its lifetime.

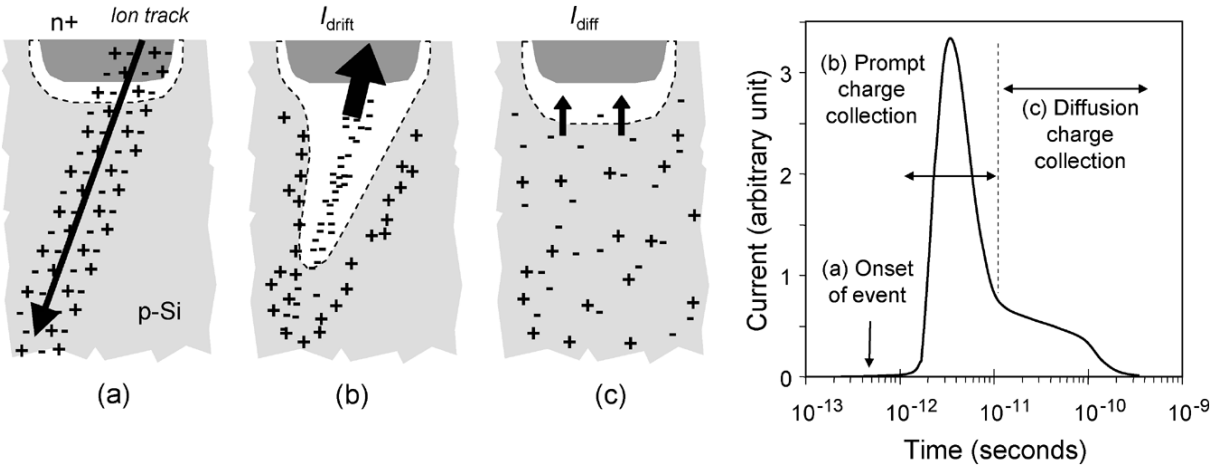


Source: the author.

applications (DODD et al., 2010). Therefore, when designing highly dependable systems, designers should be aware of the selected device technology and the environment conditions.

In aerospace applications, such as communication satellites and avionics, failures occur frequently due to the increased flow of charged particles from the cosmic radiation and solar winds, such as protons, neutrons, alpha particles and heavy ions. Sea level applications also can be affected, mainly by alpha particles and high-energy neutrons. Alpha particles can arise from radioactive impurities in the semiconductor material itself and from the packaging material, while high-energy neutrons are generated by the interaction of the cosmic radiation with the upper atmosphere. The interaction of charged particles with sensitive regions of a semiconductor device can cause enough of a charge disturbance to affect its operation transiently or permanently. When a high-energy particle strikes the silicon substrate, it liberates its energy via the production of free electron-hole pairs, resulting in an ionized track that leads to a current pulse disturbance, as illustrated in Figure 1.2. The amount of charge collected in the transistor source and diffusion nodes and the consequent impact on the device operation depends on a complex combination of factors, such as the type of particle and the energy transferred, defined as Linear Energy Transfer (LET), which depends on the mass and energy of the particle and the material in which it is traveling. Moreover, the charge accumulation needs to cross a certain threshold (critical charge) to cause a circuit malfunction (BAUMANN, 2005).

Figure 1.2 – Effect of an ionizing particle striking a reverse-biased p-n junction.



Source: (BAUMANN, 2005).

Any measurable or observable effect on a semiconductor device due to a single charged particle strike is defined as a Single-Event Effect (SEE) (JEDEC, 2006). The most relevant SEEs are the Single-Event Latch-up (SEL), the Single-Event Transient (SET), and the Single-Event Upset (SEU). SEL is a potentially destructive condition caused by the creation of a parasite structure in CMOS devices, resulting in a low-impedance path between power and ground. If the device is not destructed by the overcurrent generated, the malfunction can be solved by a power cycle. SET is a transient voltage pulse that may propagate through the system's circuitry and lead to a SEU, which is characterized by state inversion in a latch, flip-flop, or memory cell. Thereby, a SET generated by sufficient amount of collected charge may change the system's internal states during its operation, which may result in functional failures and loss of information (MUNTEANU; AUTRAN, 2008). Note that unlike SELs, SETs and SEUs do not cause any permanent damage in the device.

In digital systems, both the logic circuit and memory storage are susceptible to SEEs. In logic circuits, sequential logic elements, such as flip-flops and latches, are used to hold logic states (data or control signals) between combinational logic blocks. As explained, a SET is generated when a charged particle strikes the sensitive area of a device and enough charge is collected. This transient may propagate through a combinational circuit and eventually be captured by a sequential element depending on the sampling clock and the set-up and hold times, flipping its logic state (SEU). Due to masking effects inherently present in logic circuits, however, not all transients will result in an error. Those effects can be divided as follows.

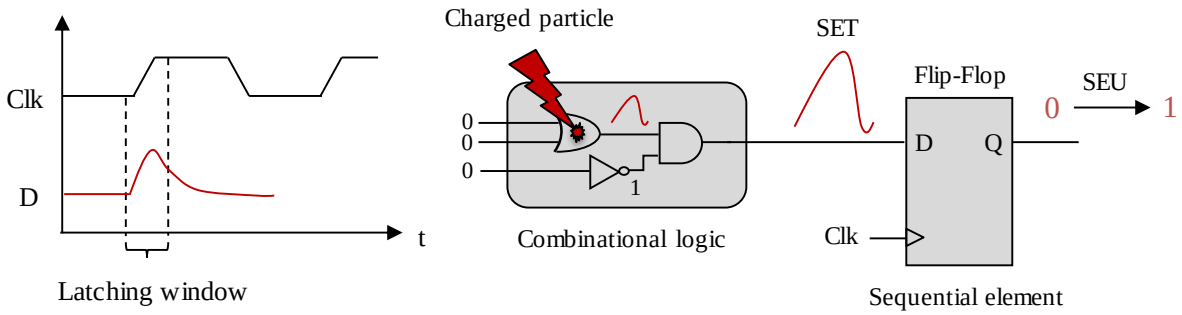
- *Logical Masking*: the logical masking effect occurs when the output of a logic gate is not affected by the node where the transient disturbance was generated or arrived. When

a transient pulse strikes the input of an *AND* gate, for example, the output will not be changed if one of the other inputs is in logical state ‘0’. Therefore, a transient pulse in a combinational logic block will propagate to a sequential element only if there are sensitive paths from the affected node to the circuit primary outputs.

- *Electrical Masking*: the electrical masking effect occurs when the transient pulse is sufficiently attenuated or even disappears as it passes through the logic gates of a sensitive path in a combinational circuit. If the transient pulse propagates towards the input of a sequential element without sufficient amplitude and width, it will be ignored and hence will not cause any effect. Note that the electrical masking phenomenon depends on the number of logic gates through the propagation path and their electrical properties.
- *Temporal Masking*: the temporal masking effect occurs when a transient pulse propagates to the input node of a sequential element, such as a flip-flop, and occurs outside of the clock latching window. In such cases, the transient will not result in an error because it will not be latched into the flip-flop.

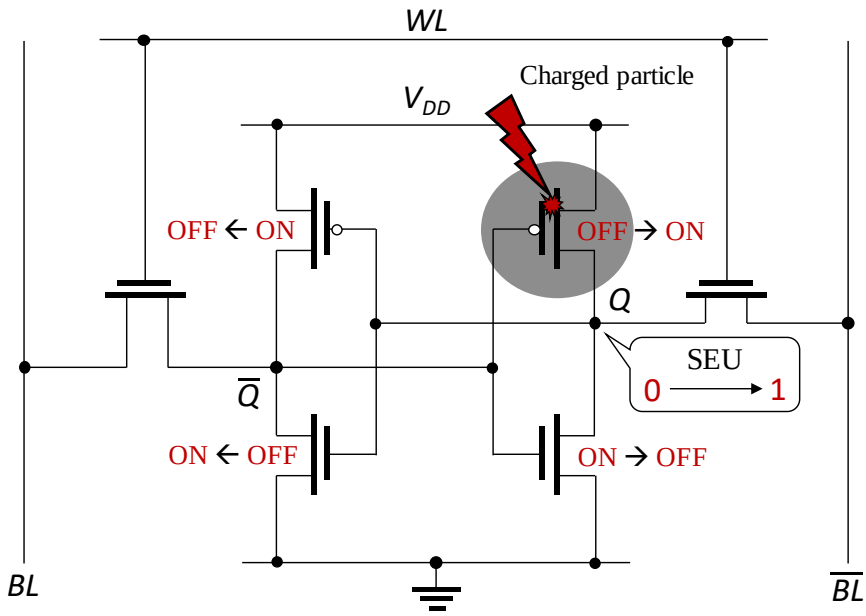
Although the amount of charge collected in newer CMOS technologies decreased due to smaller junctions as transistors shrink, the trend is higher SERs. Since one of the consequences of the technology scaling is higher transistor density per chip, the probability of an energized particle strikes a sensitive node increased. Moreover, the critical charge decreased because the supply voltage and capacitances go down across process generations. The decreasing operation voltage and propagation delay, as well as the escalating operation frequency, also decreased the probability of fault masking due to electrical or timing characteristics. Therefore, the technology scaling increased the probability of a SET to

Figure 1.3 – A SET generated in a combinational circuit can lead to a SEU.



Source: the author.

Figure 1.4 – A SEU within a typical SRAM memory cell due to a charged particle strike.



Source: the author.

propagate through a combinational circuitry and result in a SEU, as exemplified in Figure 1.3. Note that, depending on the logic fan-out, multiple upsets may occur due to a SET propagating across multiple paths.

The direct corruption of data stored in Static Random Access Memory (SRAM) cells is also a major concern when implementing high reliable systems (CHATTERJEE et al., 2014). Typical SRAM cells have six transistors connected in such a way that two complementary nodes (Q and $\overline{Q}$) can hold one of two logic states (‘0’ and ‘1’). In each state, two transistors are turned ON and two are turned OFF. When a charged particle strikes a sensitive node, such as the drain of an OFF state transistor, a bit-flip will occur if the charge collected reaches the critical charge and generates a transient current pulse that can reverse the transistor state to ON. Figure 1.4 illustrates a SEU within a SRAM memory cell due to a charged particle strike. It is important to note that a single particle strike may deposit sufficient energy to affect adjacent memory cells as well, especially when using high density arrays manufactured with state-of-the-art technologies (BHUVA et al., 2015).

The use of radiation hardened solutions, either by technology or by design, can solve the radiation effects problem for high reliable applications or critical systems operating in radiation-harsh environments. Nevertheless, Commercial Of-The-Shelf (COTS) devices become widely used, due to much lower costs, higher performance, and the use of state-of-art technologies. This trend, however, introduced the challenge of building reliable systems with non-reliable devices, leading to the need of improving the implemented system reliability with

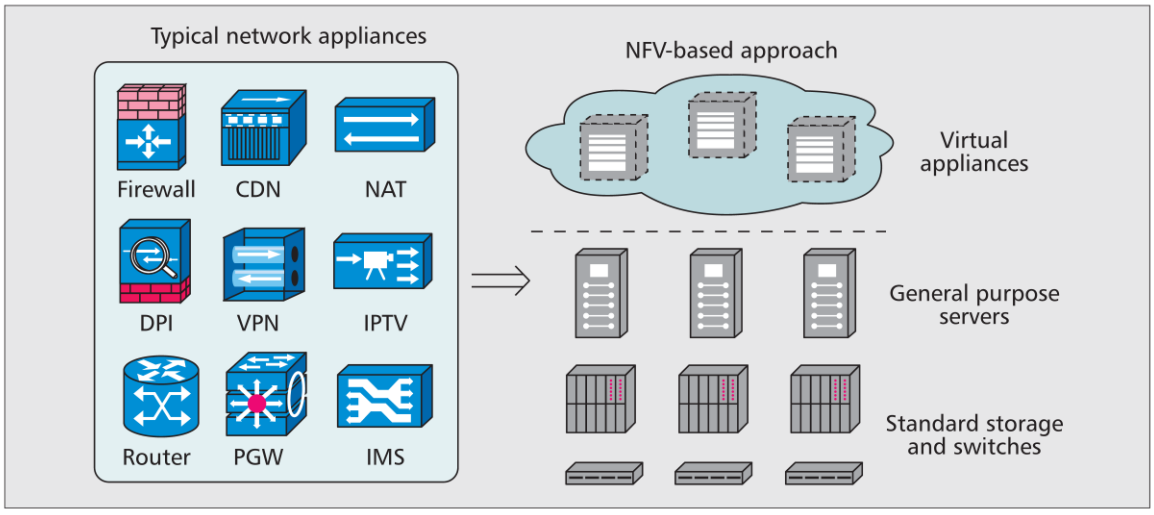
the inclusion of system level fault-tolerance mechanisms, such as hardware, information, or time redundancy. Depending on the implementation and the level of protection required, these mechanisms may impose high area, power and delay overheads. Therefore, cost-effective solutions depend on the device technology and the system's design, which demands a carefully evaluation.

1.1 Motivation

The Network Function Virtualization (NFV) paradigm has emerged as an enabling technology to improve the networking scalability and flexibility, leading to significant cost savings for deploying and managing new network services (HAN et al., 2015). By exploring the virtualization technology, the NFV infrastructure (NFVI) enables the replacement of traditional hardware-based appliances for network services, such as routers, firewalls, and gateways, by software-based appliances running on general-purpose servers. Hence, new network services can be deployed quickly and on-demand by just uploading the network functions (NFs) to commodity hardware resources instead of purchasing dedicated and vendor-specific hardware, allowing flexible deployment and dynamic provisioning of virtualized NFs. Moreover, decoupling the network functions from specialized hardware enables software and hardware to evolve independently, facilitating the network innovation and enabling new business opportunities (MIJUMBI et al., 2016a). Figure 1.5 illustrates the NFV infrastructure.

However, the shift from the traditional network infrastructure to virtualized NFs running on commodity shared hardware introduces several challenges regarding resource management,

Figure 1.5 – NFV infrastructure.

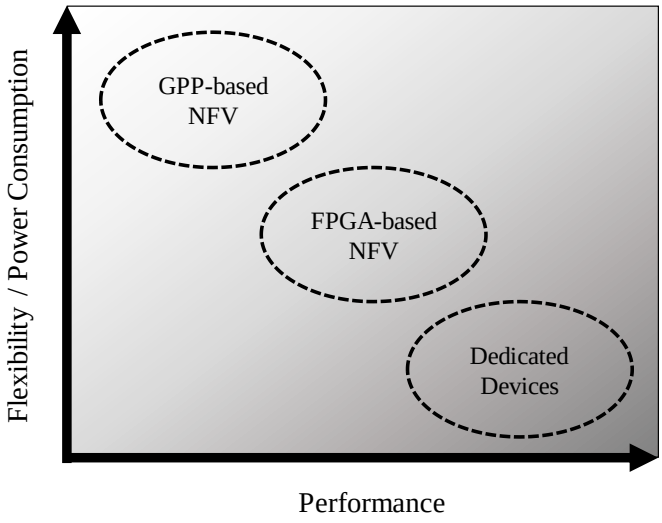


Source: (HAN et al., 2015).

function placement, portability, dependability, and performance guarantees (CHATRAS; OZOG, 2016; MIJUMBI et al., 2016b; WANG et al., 2016). Particularly, network and compute intensive functions may experience unacceptable performance degradation when deployed on an NFV-based network. To address this problem, hardware acceleration has been proposed to make a trade-off between performance and flexibility. In this scenario, Field-Programmable Gate Arrays (FPGAs) emerge as a good option for hardware acceleration of NFs that require high throughput, without deviating from the concept of an NFVI which aims at high flexibility (BRONSTEIN et al., 2015; NOBACH; HAUSHEER, 2015; UNNIKRISHNAN et al., 2011). NFs that may benefit from FPGAs include media processing, network address translation (NAT), network deduplication (Dedup), and deep packet inspection (DPI), which typically makes heavy use of regular expression matching (REM). Figure 1.6 illustrates the trade-off achieved with FPGA-based NFV.

The powerful and flexible expressiveness of regular expressions has made REM a common method to perform DPI in many content-aware network services (XU et al., 2016). For instance, Network Intrusion Detection and Intrusion Prevention Systems (NIDS/NIPS) perform DPI to scan arriving packets (payload and possibly also the header) according to thousands of signatures based on regular expressions or exact strings. Such applications require real-time packet processing to make quick decisions and to take actions to protect or to alert the network about potential security threats. However, wire-speed DPI becomes a big challenge as the bandwidth of network traffic and the number of patterns to scan increase, imposing the need of efficient REM engines (REMEs) to build high-performance DPI systems. Therefore, the

Figure 1.6 – Trade-off achieved with FPGA-accelerated NFV.



Source: the author.

virtualization of the REM task in a traditional NFV-based network could result in a major bottleneck for the DPI process, making hardware acceleration mandatory to deal with performance constraints. Since flexibility is one of the keys benefits offered by the NFV paradigm, FPGA-based REM solutions become a promising option.

With the ever increasing use of networked systems, REM-based DPI becomes heavily used in a plethora of security and mission critical applications to identify and prevent malicious attacks obfuscated by embedded content, special encoding schemes or nonstandard syntax (XU et al., 2016). For example, such attacks often target security critical systems dealing with sensitive data, such as confidential information from governments. Another example is the interruption of financial institution operations through Distributed Denial of Service (DDoS), which may result in huge financial loss. Recently (may 2017), the WannaCry ransomware attack paralyzed thousands of information systems around the world, especially networked systems of enterprises and government institutions. Therefore, to deal with the increasing number of threats that comes with the network expansion, the core of systems protecting many critical applications usually relies on REM-based DPI.

However, as stated in (HAN et al., 2015; MIJUMBI et al., 2016a), substantial research effort will be required to leverage the reliability and availability of commodity hardware to the levels expected from carrier-grade equipment. For commercial-of-the-shelf FPGAs, soft errors in the configuration memory of SRAM-based devices are a major concern and present a distinct set of challenges that demands specialized evaluation. The technology scaling and the growing number of configuration cells in SRAM-based FPGAs increased the configuration memory sensitivity to radiation induced SEUs (LESEA et al., 2005; QUINN et al., 2015). As this memory configures the logic and routing of the device, such faults may have complex and persistent effects on the functionality of the mapped design. Hence, configuration faults in FPGA-based REMEs may lead DPI-based services to failure or to behave in an unexpected way. More specifically, such faults may lead to false positive or to false negative matching alerts. Depending on the policy adopted by the network service that uses the REMEs, false positives may result in performance degradation due to unnecessary packet dropping. On the other hand, false negatives could represent a major security threat. In NIDS, for example, potentially harmful packets may go unnoticed due to the absence of matching alerts. Furthermore, due to the persistent effect of configuration faults, the service misbehavior will persist until the device is reconfigured with the original bitstream.

The configuration memory in SRAM-based FPGAs represents the majority of the total memory bits and is considerably bigger than other elements in the device, i.e., there is

substantial probability that faults affecting the FPGA will hit this memory. Therefore, the deployment of FPGA-based REM engines should be carefully evaluated due to the impact on the NFVI dependability.

1.2 Goals

This work aims to offer a resilient state-of-the-art FPGA-based REME. For that purpose, the architecture presented in (YANG; PRASANNA, 2012) is used as case study. By studying the baseline REM architecture and fault tolerance techniques for SRAM-based FPGAs, this work intends to present a cost-effective fault tolerant solution to mitigate the effects of configuration faults. In the architecture at hand, the REMEs are designed aiming at both high performance and efficient resource usage. Depending on the available resources, hundreds of REMEs can be implemented in a single FPGA. Therefore, an effective solution should be able to decreases the system's failure rate with manageable costs, i.e., minimizing area, delay, and power consumption overheads to allow the inclusion of more regexes or the adoption of smaller and lower cost FPGAs. With the low-cost requirement, some traditional fault-tolerance mechanisms, such as Triple Modular Redundancy (TMR), may not be a good option to improve the system's resilience due to the amount of additional resources needed, as well as the increased power consumption. Therefore, the fault-tolerance mechanisms to be applied must be carefully evaluated, considering the particularities of the design at hand to offer an appropriate trade-off between the mitigation solution overhead and fault masking capacity.

Additionally, the FPGA dynamic reconfigurability capability can be used to correct soft errors before they accumulate on the configuration memory, potentially defeating error mitigation techniques implemented at system level. The most common repair procedure for SRAM-based FPGAs is configuration scrubbing (CARMICHAEL, 2000), which consists in overwriting the configuration memory with its correct contents. As newer FPGAs get larger, however, their configuration memories also grow, requiring hundreds of milliseconds or even over a second to be scrubbed, depending on the available configuration bandwidth. Therefore, reducing the repair latency and hence the timeframe in which the system remains in error state is another challenge addressed by this work.

The optimal fault tolerance solution depends on the chosen FPGA device and the system design, but typically combines system level redundancy and scrubbing techniques (KASTENSMIDT; REIS, 2006). The solution proposed in this work aims to maintain the system safely operational during the timeframe between error detection and correction without

the costs of TMR. Maintaining the system safely operational during repair is important since, in high bandwidth systems, many gigabytes may be lost in a fraction of second. In other words, the failure rate and the repair time should be as small as possible to improve the system's reliability and availability.

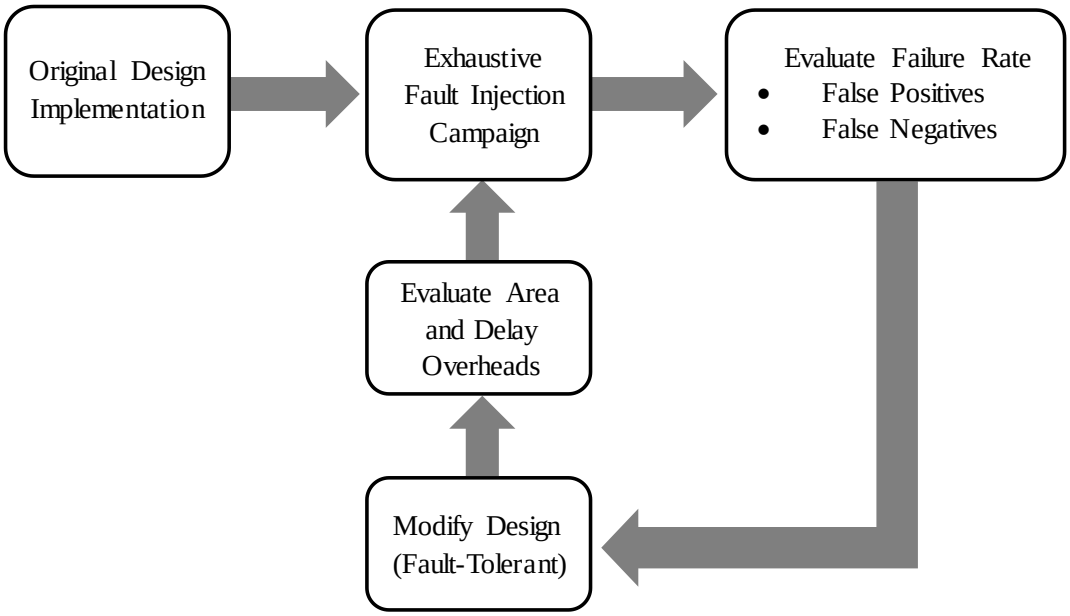
The methodology used to achieve the intended results is based on exhaustive fault injection campaigns. By means of fault injection, we evaluate the effects of configuration faults on the functionality of the baseline FPGA-based REM system. Based on the characterized fault sensitivity and on the frequency of each failure mode (false positives and false negatives), we propose cost-effective fault tolerance mechanisms to improve the system's resilience against such faults. These mechanisms are especially suitable for systems wherein one failure mode is more critical than others and they have different probabilities of occurrence. In such cases, evaluating the criticality and frequency of each failure mode is crucial to define cost-effective fault-tolerance solutions. For that purpose, we also introduce a theoretical model for early estimation of the fault-tolerance solution efficiency on reducing the system failure rate based on its capacity to deal with each failure mode. In this sense, the baseline REMEs are used as a proof of concept. As explained, the goal is improving the system reliability by masking the majority of errors triggered by configuration faults with manageable area and performance penalties.

Finally, a placement-aware fast error detection and correction mechanism is proposed to minimize the timeframe during which the system is faulty and to avoid the accumulation of faults that may defeat the error masking mechanism. Figure 1.7 exemplifies methodology used to achieve the intended goals. In a nutshell, the main contributions of this work are:

- A resilient state-of-the-art FPGA-based REM system with cost-effective fault-tolerance mechanisms designed to improve the reliability of systems where the failure modes have different criticalities and probabilities of occurrence.
- A placement-aware scrubbing technique for fast error detection and correction to improve the system availability that relies on the dynamic reconfigurability capacity of modern FPGAs.

The remainder of this dissertation is structured as follows. Chapter 2 provides a background of FPGA-based REM and fault-tolerance mechanisms for FPGAs, especially regarding error detection and correction methods. Chapter 3 presents the baseline REM architecture used as a case study, the reliability model, and the fault-tolerance mechanisms

Figure 1.7 – Fault-tolerant system design methodology.



Source: the author.

proposed. In Chapter 4, the error detection and correction method is presented and discussed in details. Chapter 5 presents the design flow and the experimental setup used to implement and evaluate the proposed techniques, respectively. Chapter 6 presents a detailed discussion of experimental results. Finally, Chapter 7 presents the conclusions and possible future works.

2 BACKGROUND AND RELATED WORK

This chapter introduces the implementation of regular expression matching in hardware and how radiation effects may affect FPGA-based designs, more specifically configuration memory faults. Also, typical fault-tolerance techniques used to mitigate such faults are presented and discussed in details.

2.1 Regular Expression Matching

According to formal language theory, regular expressions are a notation that describes the same languages accepted by finite automata (HOPCROFT; MOTWANI; ULLMAN, 2007). Due to the expressive power of metacharacters, regular expressions are a compact and flexible way to define search patterns in a wide range of DPI applications. For instance, most of the search signatures in NIDS, such as Snort (SNORT, 2017) and Bro (BRO, 2017), are described as regular expression patterns. Since regular expressions are equivalent to finite state automata in regular language description, the implementation of REMEs mainly falls into deterministic finite automata (DFA) or non-deterministic finite automata (NFA) (BECCHI; CROWLEY, 2008; FLOYD; ULLMAN, 1982). Table 2.1 shows typical REM operators.

Table 2.1 - Operators supported by finite automata for REM

Operator	***Name***	***Example***	***Description***
-	Concatenation	$\alpha\beta$	β right after α
|	Union	$\alpha \mid \beta$	Either α or β
*	Kleene Closure	α*	α zero or more times
+	Repetition	α+	α one or more times
?	Optionality	α?	α zero or one times
{*m*, *n*}	Constrained Repetitions	α{*m*, *n*}	α in *m* to *n* times
^	Start of String	^α	α at start of input
$	End of String	$\alpha$$	α at end of input
[…]	Character Class	[*A*-*C*]	Either *A*, *B* or *C*
[^…]	Inverted Character Class	[^ *A*-*C*]	Neither *A*, *B* nor *C*

Source: adapted from (YANG; PRASANNA, 2012).

To deal with the networking ever-increasing link speed and pattern scale, a DPI system should provide high-performance pattern matching, scalability of patterns, and dynamic update capacity. The dynamic update requirement imposes the implementation of finite state machines in memory (state transition tables) or reconfigurable logic. Although NFA-based REMEs make larger state transitions and therefore require much more memory bandwidth, they do not suffer from exponential state explosion, a common result of implementing complex regexes with DFAs. On the other hand, DFA approaches lead to limited memory bandwidth requirements, since each input symbol involves one state transition. Therefore, DFA-based REMEs are commonly deployed in memory-based platforms by using algorithms to minimize state explosion and hence the storage requirement, while NFA approaches are typically deployed in FPGAs, due to the fine-grained circuit parallelism and limited resources offered by such devices (ANTONELLO et al., 2012).

2.1.1 FPGA-based REM

The proposal of efficient architectures to perform REM on FPGAs has driven several works, as these devices offer at both high throughput processing of packets and flexibility to update matching patterns. The main challenges addressed are related to matching several regex patterns in parallel with high concurrent throughput and efficient resource usage. The work presented in (SIDHU; PRASANNA, 2001) introduces an algorithm to translate an arbitrary regex into a REM circuit for FPGAs. The authors use the classic McNaughton-Yamada algorithm (MCNAUGHTON; YAMADA, 1960) to construct an NFA architecture to process one text character at every clock cycle. The NFA circuits are designed combining basic blocks that implement Concatenation, Kleene closures, and Union operators.

Based on (SIDHU; PRASANNA, 2001), the solution proposed in (YAMAGAKI; SIDHU; KAMIYA, 2008) includes a technique to construct an NFA that processes multiple characters per clock cycle, improving the throughput achieved. Another main contribution is presented in (SOURDIS et al., 2008), wherein new blocks are designed to perform constrained repetitions. Through the use of Look-Up Tables (LUTs) configured as shift-registers instead of unrolling repetition blocks, the amount of resources needed is minimized. Also, prefix sharing and character classes sharing are proposed to reduce even more the design area cost. Furthermore, in (WANG et al., 2013) the authors introduce a counter based algorithm to handle the repetition of complex character classes in block memories (BRAMs), saving 74% of memory bits when compared to conventional NFA-based designs.

The work presented in (YANG; PRASANNA, 2012) aggregates many contributions of prior works with significant improvements to offer a high-performance and compact architecture. It constructs a modular NFA structure to easily map regular expressions into hardware circuits with low area requirement, including shared character classification based on BRAMs to match each input character against complex character classes in one memory access. A time and space-efficient technique is used to construct multicharacter matching engines, while parallel shift-registers allows matching constrained character repetitions against multiple input character per clock cycle. Moreover, a two-dimensional multi-pipeline architecture enables the matching of a large number of REMEs individually on a single FPGA, by means of priority grouping. A similar solution is proposed in (HIEU; THINH; TOMIYAMA, 2013), but a new method to handle infix and suffix sharing is introduced to reduce even more the resource usage and hence to allow the implementation of more REMEs in the same FPGA.

None of those works, however, took into account dependability threats. Depending on the available resources, hundreds of REMEs can be deployed in a single FPGA. Also, multi-character REMEs can be implemented to improve throughput at the cost of more resource usage. Hence, some traditional fault-tolerance mechanisms, such as TMR, may be infeasible to improve system resilience due to the high area overhead imposed (more than 200%). Therefore, implementing reliability-aware REMEs for FPGAs can be very challenging and should be carefully evaluated.

2.2 Radiation Effects on FPGAs

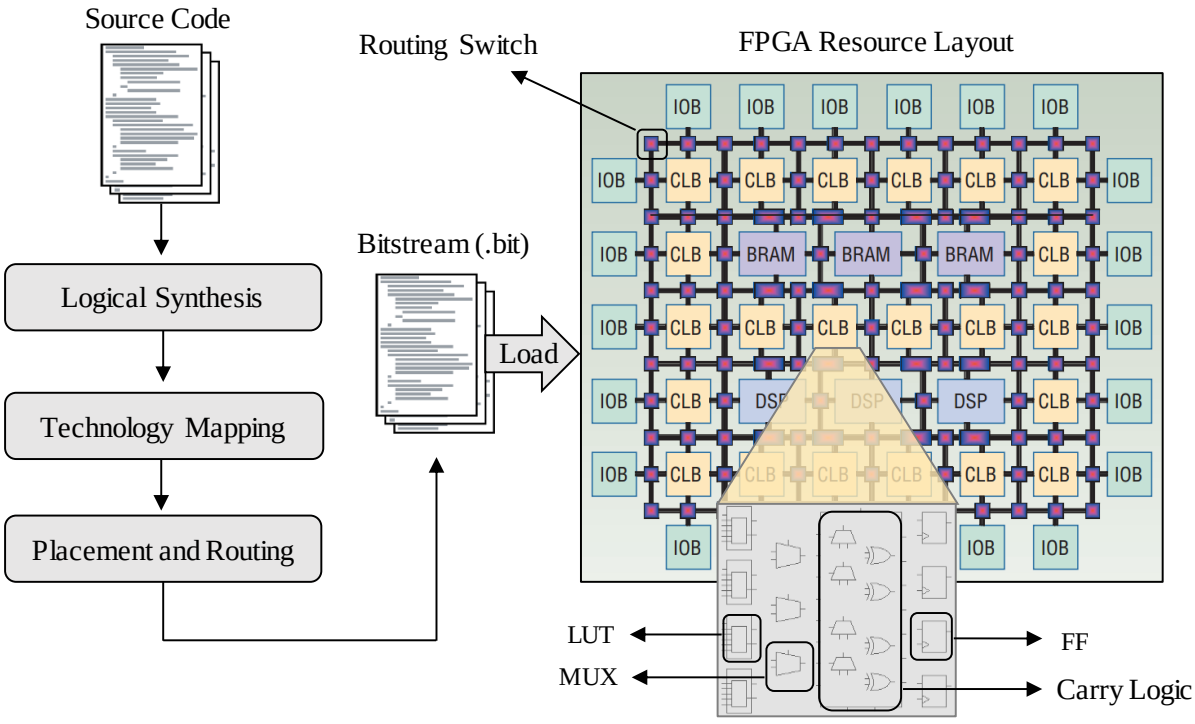
FPGAs are reconfigurable devices widely used in the implementation of computing systems. These devices offer many advantages over the use of Application Specific Integrated Circuits (ASICs), such as reconfigurability coupled with a high throughput processing of data streams and lower development time and costs. Those characteristics make FPGAs a great option for data communication applications, due to the possibility of in-field reprogramming to correct faulty behaviors or to add new features, without the performance and power penalties introduced by the use of general-purpose processors (HAUCK; DEHON, 2010). Such devices comprise a wide variety of heterogeneous resources. The typical layout of modern FPGAs is an array of interconnected blocks, including interconnect resources, clock-management resources, configurable logic blocks (CLBs), input/output blocks (IOBs), and embedded blocks such as digital signal processors (DSPs), general-purpose processors, high-speed IOBs, and memories (BRAMs). CLBs are used to perform simple combinational and sequential logic. These blocks

are typically made of LUTs, multiplexers, flip-flops, and carry logic. Programmable interconnect resources, such as routing switches, allow interconnecting CLBs, IOBs and embedded blocks to implement complex systems.

The logic and routing resources in an FPGA are controlled by the bits of a configuration memory, which may be based on antifuse, flash, or SRAM technology. The design flow of FPGA-based systems involves the creation of a bitstream to load into the device, as illustrated in Figure 2.1. Typically, the process starts with the system design written in a hardware description language (HDL), such as VHDL or Verilog. Next, the design is optimized and mapped into the FPGA's available resources through logical synthesis, technology mapping, placement, and routing. Finally, the bitstream is generated and the device can be programmed. FPGA vendors such as Xilinx, Altera, Lattice Semiconductor, and Atmel provide implementation tools to perform the design flow (GOKHALE; GRAHAM, 2005).

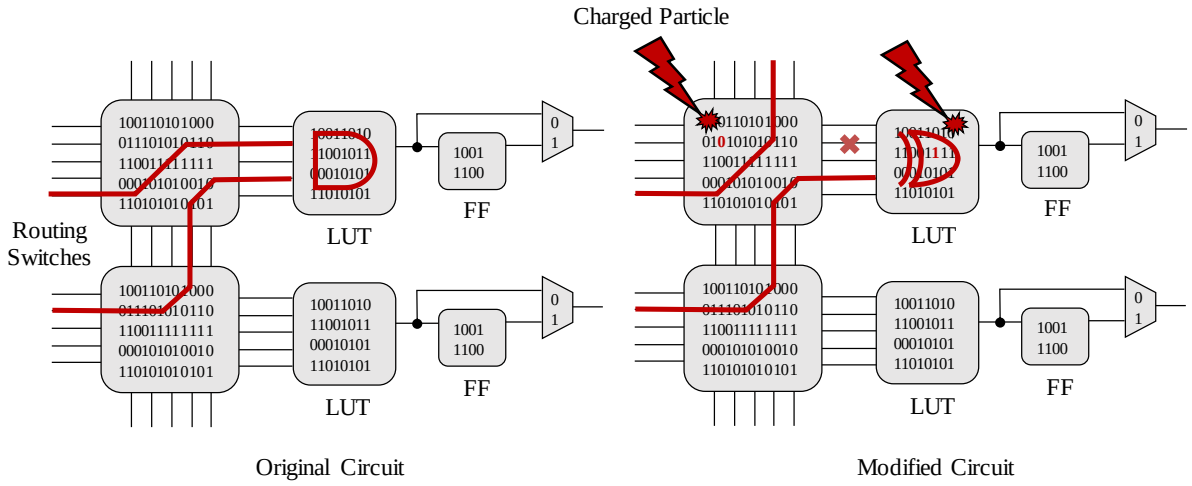
Like any other semiconductor device, FPGAs are susceptible to radiation effects. Mostly, these effects depend on the technology used to store the configuration data. Regarding the impact of SEEs on reliability and functionality, FPGAs based on SRAM technology are a special class of devices. The major concern for SRAM-based FPGAs is SEUs within the configuration memory. In such devices, this memory may represent more than 80% of the total

Figure 2.1 – Typical FPGA structure and design flow.



Source: adapted from (HAUCK; DEHON, 2010) .

Figure 2.2 - Circuit mutation due to configuration faults.



Source: adapted from (WIRTHLIN, 2015).

memory bits, increasing the probability of configuration faults. Upset configuration bits may change the logic and routing of the implemented system, as illustrated in Figure 2.2, leading to functional failures in an unpredictable way. In contrast, the primary concern for antifuse and flash-based FPGAs is SETs and SEUs within user flip-flops and block memories. Although the configuration memory cells of antifuse and flash-based FPGAs present a relative immunity to SEEs, these devices has lower logic capacity and cannot be reprogrammed an unlimited number of times, making SRAM-based FPGAs more suitable for complex systems requiring frequent reconfiguration and adaptation (VIOLANTE et al., 2004; WIRTHLIN, 2015).

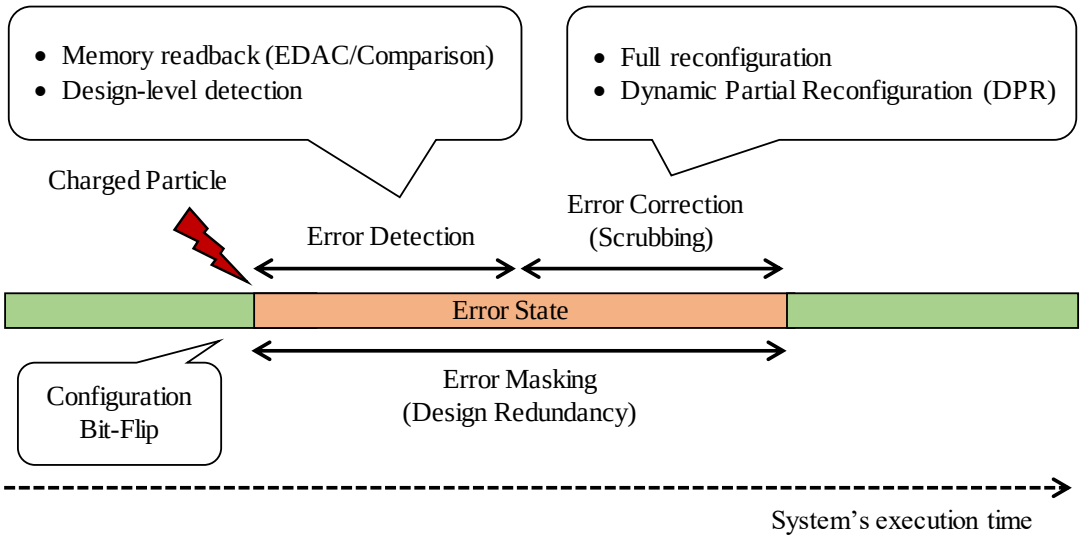
2.3 Fault Tolerance for FPGAs

The vulnerability of SRAM-based FPGAs to configuration data SEUs demands the use of techniques to mitigate their impact on the system functionality. The common way to implement reliability-aware systems in SRAM-based FPGAs is combining hardware redundancy and memory scrubbing techniques. Design level redundancy allows to detect and to mask errors at the design level, while memory scrubbing consists in overwriting the configuration memory with its correct contents. Correcting erroneous memory frames is mandatory to recover the system from failures or to prevent the accumulation of faults that can defeat redundancy techniques applied at the design level. Therefore, providing efficient fault-tolerance mechanisms is of decisive importance to offer FPGA-based systems with high reliability and availability.

To preserve the system functionality and data integrity in the event of soft errors, design redundancy and Error Detecting and Correcting (EDAC) techniques, such as TMR and Error Correcting Codes (ECCs), are typically implemented in user logic and memories. These techniques cover not just configuration faults, but also SETs and SEUs in sequential logic and embedded blocks. TMR is a hardware redundancy technique that consists in implementing the circuit three times, and their outputs compared by a voter to decide the correct one. Despite the amount of extra logic resources needed to implement this technique, it is very effective to prevent errors until more than one module is affected. This method has limitations, as demonstrated in (QUINN et al., 2007), when the design is affected by Multi-Bit Upsets (MBUs), which requires a smart floorplan usage to isolate the TMR modules. The EDAC technique is based on information redundancy, i.e., additional bits are introduced in the data word through complex data encoding techniques. Depending on the encoding scheme, one or more erroneous bits can be detected and corrected, impacting the circuitry complexity and the amount of resources needed. The best SEEs mitigation solution depends on the chosen FPGA device and the system design, but typically combines the techniques aforementioned. As mentioned, the correction of configuration faults can be made through memory scrubbing, which consists in rewriting the configuration memory with the correct content. The circuit that performs the memory scrubbing (scrubber) can be implemented externally or internally to the FPGA. An external scrubber uses external configuration ports, such as the JTAG (Joint Test Action Group) or the SelectMap (Xilinx), while an internal scrubber uses an internal configuration port, such as the Internal Configuration Access Port (ICAP) in Xilinx FPGAs. More details can be found in (KASTENSMIDT; REIS, 2006).

Regarding the scrubbing methodology, the simplest scrubbing method consists in periodically performing a full reconfiguration at a higher rate than the predicted soft error rate, without performing any error detection. A more effective approach consists in reading back the configuration memory and triggering the scrubbing only in case an error is detected. The error can be detected by comparing the readback data with the correct content (golden memory) or by built-in EDAC and Cyclic Redundancy Check (CRC) codes. This method, however, requires more complex circuitry to calculate and compare the CRC values, as well as additional memory to store them. The periodic scrubbing method, in contrast, only requires a controller to timing the reconfiguration process. With system level redundancy, such as DMR or TMR, the scrubbing procedure also can be triggered by design-level error detection. If the detection method provides the error location, then Dynamic Partial Reconfiguration (DPR) can be used to scrub only the faulty portion of the configuration memory, reducing the repair time.

Figure 2.3 – Basic design space exploration for fault-tolerant FPGA-based systems.



Source: the author.

Therefore, choosing the right configuration scrubbing methodology for an application typically implies trading-off reliability, performance, power consumption, flexibility, and complexity (HERRERA-ALZU; LÓPEZ-VALLEJO, 2013). Figure 2.3 illustrates the basic design space exploration for fault-tolerant FPGA-based systems.

2.3.1 Related Work

Partial reconfiguration through the ICAP is widely used in scrubbing mechanisms for Xilinx FPGAs. Since Virtex-4, all devices come with a self-correction mechanism that uses ECC in each configuration frame and performs CRC calculations for the whole configuration memory. The Xilinx SEU Controller is a soft-core that manages a self-correction mechanism for Virtex-5 FPGAs. This controller reads the configuration frames through the ICAP and can correct one single bit upset (SBU) per frame by using embedded ECC bits. However, the performance of this scrubber is limited, since it is implemented with an 8-bit PicoBlaze microcontroller. For 7-series family of FPGAs, Xilinx offers the Xilinx Soft Error Mitigation (SEM) Controller, where the correction method can be based on built-in ECC bits only, a combination of ECC and CRC or external golden memory. The ECC/CRC scheme allows to correct up to two adjacent bit upsets in one frame.

The proposal of efficient and on-demand scrubbing techniques to implement reliability-aware systems on SRAM-based FPGAs has driven several works (BOLCHINI; MIELE; SANDIONIGI, 2011; FOUAD et al., 2014; NAZAR, 2015; NAZAR; SANTOS; CARRO,

2015; RAO et al., 2014; REORDA; STERPONE; ULLAH, 2017; SANTOS; NAZAR; CARRO, 2016; SARI; PSARAKIS, 2011; SCHMIDT; ZIENER; TEICH, 2014; TONFAT et al., 2015). Typically, these techniques rely on frame-level or design-level error detection in combination with DPR. The work presented in (RAO et al., 2014) uses a scheme based on erasure codes to protect the configuration memory frames against multi-bit upsets (MBUs). The upset detection is performed through parity bits with an interleaved 2-D scheme stored in BRAMs. The configuration frames are grouped in clusters, wherein a redundant frame with parity information is needed. Therefore, if a faulty frame is detected in a given cluster, the scrubber needs to read only the frames of that cluster to recover the original data, reducing the repair time.

In (TONFAT et al., 2015), the authors introduce a frame-level redundancy technique based on coarse grain TMR design implemented with Hard Macro Blocks (HMBs), where each domain has the same frame data. Therefore, the design is able to mask errors at the circuit level and also correct configuration faults through bit-level majority voting and partial reconfiguration. This technique can detect and correct multiple errors in different TMR domains. However, due to the coarse granularity, the TMR scheme proposed is not able to provide a clue to the error location, imposing the need for analyzing the configuration frames of the whole circuit.

The work presented in (NAZAR; SANTOS; CARRO, 2015) introduces a combination of fine-grained error detection method and partial reconfiguration to provide a fast scrubbing mechanism. Using a statistical approach, fault injection campaigns are performed to identify a relation between error signatures, provided by a fine-grained DMR scheme, and faulty configuration memory frames. With a signature translation table, the design uses the error information to dynamically choose the optimal frame to start the scrubbing process, reducing the repair time. Despite the introduction of 10.4% of area overhead over standard DMR, this method is able to reduce the average mean time to repair (MTTR) in 80.85%. A similar approach is presented in (NAZAR, 2015) (fine-grained DMR) and (SANTOS; NAZAR; CARRO, 2016) (coarse-grained DMR), aiming to avoid missing deadlines in real-time systems.

Some works also propose specific placement strategies to reduce even more the system repair time through a more precise error localization. In (BOLCHINI; MIELE; SANDIONIGI, 2011), a reliability-aware design methodology is presented, wherein the circuit is partitioned into groups accordingly to specific reliability requirements with the aim of exploring different fault-tolerance techniques, applied at different levels of granularity. Next, each group is placed

in specific regions of the FPGA resource layout to achieve a reduction of more than 80% in scrubbing time.

In (SARI; PSARAKIS, 2011) the soft error sensitivity of a Leon3-based System-on-Programmable-Chip (SoPC) is evaluated in a Virtex-5 device. Based on the high dispersion of the sensitive bits in the configuration frames, the authors propose a constraint-driven placement strategy to reduce the number of sensitive frames used by the circuit. As a result, the scrubbing time is reduced by 41% compared with the standard placement generated by the Xilinx tool. Similarly, the work presented in (SCHMIDT; ZIENER; TEICH, 2014) proposes a netlist analysis using the Xilinx bitgen tool to distinguish critical and essential bits without the need of fault injection experiments. Then, placement and routing constraints are used to confine the critical bits into a specific region of the resource layout. Experimental results showed that the mechanism proposed may reduce the MTTR by up to 48.5%, depending on the design being implemented. The netlist analysis, however, does not consider the exploitation of redundancy mechanisms. Another work exploiting the design placement is presented in (FOUAD et al., 2014), wherein a context-aware placement algorithm is developed to reduce the time for fault detection, hardware checkpoint and hardware recovery of a Backward Error Recovery (BER) mechanism used to protect systems where both configuration bits and context must be corrected to avoid fault propagation.

A fine-grained fault detection and correction method is presented in (REORDA; STERPONE; ULLAH, 2017), based on the Duplication With Comparison (DWC) technique applied at two different levels of granularity. A coarse-grained DWC region is defined for slices that use the carry chain circuit for computations such as fast multiplications. In this case, the DWC is performed at the module level, where an LUT element is configured as a *XOR* function to generate an error detection flag for that region. On the other hand, a fine-grained DWC scheme is applied at the place and route level, by duplicating each LUT function. Then, the outputs of the two LUTs are compared with built-in hardwired resources present in the carry chain circuit of each slice. A single error detection flag is generated by connecting the comparison results in a chain of *OR* logic functions through the carry chain circuits of each column of CLBs. The scrubber circuit is implemented in a static region and manages the ICAP to perform partial reconfiguration according to the error flags.

Note that the works proposing design-level error detection usually rely on application-oblivious schemes (DMR or TMR) to trigger localized scrubbing procedures. Depending on the system being implemented, however, fine-grained solutions (LUT-level error detection) may impose unacceptable performance degradation due to stringent placement constraints affecting

the system critical path. On the other hand, coarse-grained solutions (system-level error detection) makes the error localization difficult and therefore the use of partial reconfiguration to reduce the system repair time becomes useless, especially in large designs. Therefore, in this work we propose a submodule-level error detection mechanism based on a DMR scheme with a design-aware granularity (REME-level error detection), aiming to offer a system that is able to take advantage of DPR to reduce the repair time with manageable overheads. Moreover, in the solution herein proposed the DMR scheme is used not just to detect errors, but also to mask the majority of them through a voter circuit, designed accordingly to the criticality associated with each failure mode and their probability of occurrence. In this way, we avoid the use of costly TMR solutions to improve the system reliability.

3 RESILIENT FPGA-BASED REM

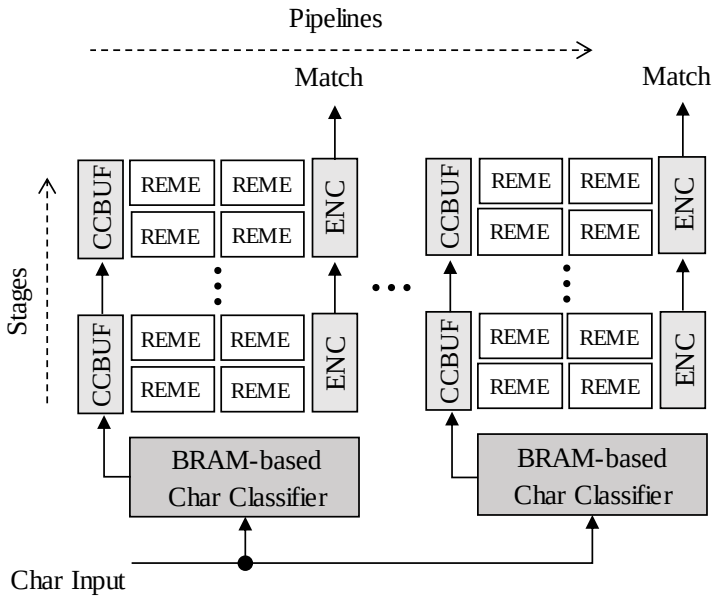
The resilient FPGA-based REM proposed in this work is based on the architecture presented in (YANG; PRASANNA, 2012), which is a state-of-the-art solution for high-performance REM on FPGAs. In this chapter, we briefly present its main modules and how they work to perform regex pattern matching. Next, the reliability model and fault-tolerance solutions proposed are presented and discussed in details.

3.1 Baseline REM Architecture

The baseline REM architecture introduced in (YANG; PRASANNA, 2012) uses a modular 2-dimensional staging and pipelining structure that allows large-scale and multi-character REM design in a single FPGA. The architecture is composed of four main elements: BRAM-based character classifiers, REMEs, priority encoders, and forwarding buffers for the character classification. The incoming 8-bit characters are classified and matched by a BRAM module of 256 bits of depth and n bits of width, where n is the number of distinct character classes to match among all the REMEs inside of the pipeline. Within a pipeline, the REMEs are grouped in stages where a single match alert is forwarded to the next stage at every clock cycle. Since outputting hundreds of match alerts may not be feasible due to the limited number of IO pins available in the FPGA, the match alerts generated are priority encoded. Inside each stage, the REMEs with lower identification number have priority to forward a match alert, while in the pipeline scope the priority is of the lower stage. Also, each stage forwards the character classification signals through buffers, but only the ones needed to feed the set of REMEs in the next stages. This approach allows matching complex character classes with just one memory access, as well as sharing character classifiers among multiple REMEs. Thus, a significant amount of resources can be saved, especially when implementing REMEs with several common states and complex character classes. For example, the regexes */a([bc]z|d)*/* and */a+[bc]z[xyz]/* will share 3 character classifiers (BRAM columns for *a*, *z* and *[bc]*). If this is not the case, the character classifiers can be implemented with logic to avoid waste of memory resources. Figure 3.1 shows an overview of the baseline REM architecture considering four REMEs per stage.

The REM engines are regular structures constructed from the NFA representation of the regular expressions. They comprise a set of interconnected State Update Modules (SUMs), which are basic structures to match a single character class. Within a SUM, the input transitions are aggregated through an *OR* gate, while the matching of the character class associated with

Figure 3.1 – REM multi-pipeline structure with BRAM-based character classifiers.



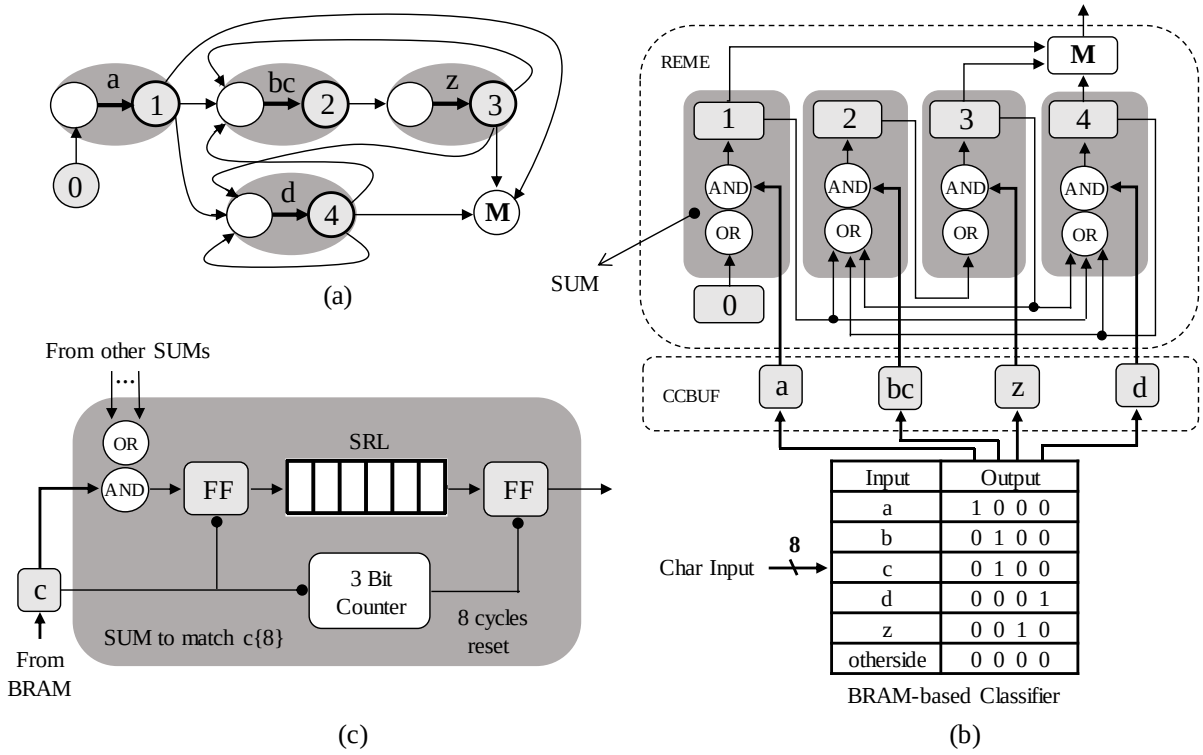
Source: the author.

the state is received as a 1-bit signal from the character classifier buffer to an *AND* gate. Each SUM is composed of $\lceil (n+1)/k \rceil$ *k*-input LUTs to perform the logic operation, where *n* is the number of input transitions, and a single flip-flop to forward the result to the next modules at every clock cycle. Constrained repetitions are implemented through LUT-based shift-registers (SRLs) instead of replicating SUMs, which reduce the resources needed. Moreover, it is possible to use *m*-character REMEs that are constructed by a spatial stacking technique that replicates the SUMs to match *m* characters at each clock cycle. Figure 3.2 (a) illustrates the NFA generated for the regular expression */a([bc]z|d)*/*, Figure 3.2 (b) shows the mapped circuit, and Figure 3.2 (c) exemplifies how constrained repetitions are implemented using the regex */c{8}/* as an example. More details can be found in (YANG; PRASANNA, 2012).

3.2 Resilient REM Architecture

Faults affecting REMEs in DPI-based network services may result in critical failures depending on the service being implemented. In NIDS, the arriving packets are scanned accordingly to a set of signatures to alert possible security threats. Therefore, the absence of matching alerts (false negatives) is critical for the network security. Although undesirable, as they can lead to network performance degradation, false matching alerts (false positives) are not as critical, since the network capacity to lead with security threats is not affected.

Figure 3.2 – Implementations of the regex pattern */a([bc]z|d)∗/* and the constrained repetition */c{8}/*. (a) NFA implementation, (b) the derived hardware on the FPGA, and (c) constrained repetition.



Source: the author.

3.2.1 Reliability Model

Reliability is an attribute commonly used to characterize the system behavior when experiencing faults. It is defined as the system ability to perform a required function, under given environmental and operational conditions and for a stated period of time. In other words, reliability is the conditional probability that the system does not experience a failure in the time interval (0, t], and can be expressed as

$$R(t) = e^{-\lambda t}, \tag{3.1}$$

where λ is a constant defining the system failure rate (KOREN; KRISHNA, 2007). Eq. (3.1) represents the exponential relationship between reliability and time and is typically used in soft error analysis because the SER is assumed to be roughly constant in the system useful lifetime, as represented in the bathtub curve in Figure 1.1. This assumption is reasonable, since in this

phase radiation-induced faults are expected to occur randomly, i.e., each component of the system has the same probability of be affected.

Observing Eq. (3.1), the key to improve the system's reliability over time is reducing the failure rate, which is typically defined in Failures in Time (FIT), the amount of failures expected per 10^9 device-hours. The failure rate for FPGA designs can be estimated as

$$\lambda = \sigma_{static} \,.\, n_{flux} \,.\, b \,.\, 10^9, \tag{3.2}$$

where σ_{static} is the sensitive area to upsets (cross-section) per bit, n_{flux} is the average neutron flux where the device is expected to operate (e.g., at sea level) and b is the amount of sensitive bits for a given design, i.e., the number of configuration bits that, if flipped, result in a failure (error propagated to the system's primary output). Neutrons are considered as these are the most common source of radiation-induced upsets for sea-level applications.

In Eq. (3.2), the static cross-section and the neutron flux depend, respectively, on the FPGA device at hand and the location where it will operate, while the number of sensitive bits depends on the design implemented. Therefore, the failure rate of a given system can be reduced by reducing the number of sensitive bits, which can be made through fault-tolerance mechanisms applied at system level, as discussed in Section 2.3. Considering the baseline REM architecture design presented in Section 3.1 and the possible failure modes (false positives and false negatives), the number of sensitive bits can be modeled as

$$b = \alpha \,.\, N_{fn} + \beta \,.\, N_{fp} + \gamma \,.\, N_{fnp}, \tag{3.3}$$

where N_{fn}, N_{fp}, and N_{fnp} are the number of sensitive bits triggering only false negatives, only false positives, and both failure modes on the baseline design (unhardened), weighted by the parameters α and β and γ, respectively. These parameters can be set according to the fault tolerance mechanism chosen, through an architectural analysis of the error masking capabilities that can be obtained for each failure mode. For instance, a given solution may reduce the number of false positives at the cost of increasing the number of false negatives ($\alpha > 1$ and $0 \leq \beta < 1$). Note that the number of sensitive bits for the baseline design is a special case wherein $\alpha = \beta = \gamma = 1$.

Eq. (3.3) can be used for an early evaluation of fault tolerance mechanisms by estimating the system failure rate obtained. It may be especially important to evaluate cost-effective

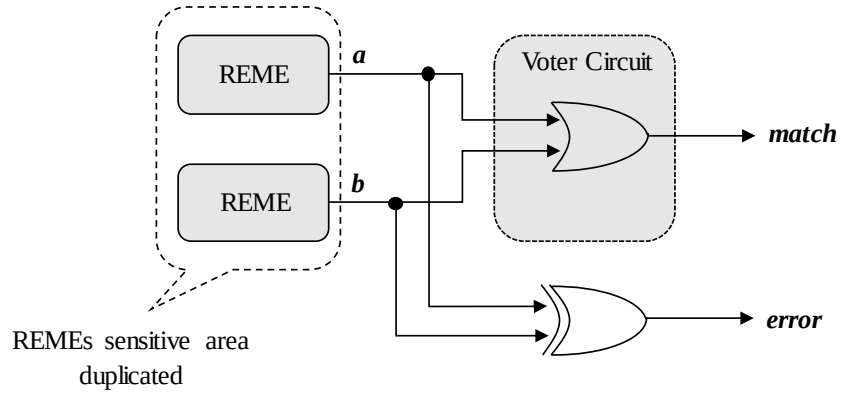
solutions in which not all failure modes can be covered properly. Thereby, evaluating the contribution of each failure mode for the overall failure rate can be very beneficial, since they often have different criticalities for the system and hence should be dealt accordingly. To exemplify, a given fault-tolerance solution designed to reduce the rate of false positives ($0 \leq \beta < 1$) at the cost of increasing the rate of false negatives ($\alpha > 1$) may impose a high overhead on the overall failure rate if the baseline design has $N_{fn} \gg N_{fp}$, i.e., if false negatives are much more likely to occur than false positives. In this case, however, the overall failure rate may not be so important, since the rate of failures that really matter are reduced. Therefore, to meet the cost-effective requirement, the fault-tolerance mechanism can be managed to primarily act over a specific failure mode according to its criticality or probability of occurrence, which depends on the system being implemented and the resources available.

3.2.2 Masking False Negative Failures

The fault-tolerance mechanisms proposed aims to offer a cost-effective solution to avoid, primarily, critical failure modes in REM-based network services, more specifically in NIDS. The REMEs consume the majority of the resources needed by the presented architecture, especially when using multi-character REMEs to improve throughput. Thus, DMR may be considered due to less aggressive area and power costs, when compared to TMR. By duplicating the REMEs, error conditions can be detected if one of them is affected, but the correct output cannot be determined. In real-world network services, such as DPI-based NIDS, the big majority of the arriving packets are forwarded without match alerts. Therefore, a non-match condition can be assumed when an error is detected with great probability of hit, as it is more likely. In other words, one solution to mask errors with a DMR scheme would be implementing a voter that choose the most likely correct output when the duplicated REMEs diverge. This approach, however, makes the network susceptible to security threats at the time window between the error detection and its correction. During this time window, harmful packets that should be dropped or reported to administrators will be forwarded silently due to absence of alert.

To avoid reliability degradation regarding critical failures, we propose a modified version of the traditional DMR solution to address this specific problem. The solution is to generate a match alert for all arriving packets when an error is detected, since false match alerts are not critical in the security point of view. In addition, it is reasonable to assume that false negatives are much more likely to occur than false positives, because breaking the sequence of

Figure 3.3 – DMR scheme with *OR*-based voter to mask critical failures for NIDS (false negatives).



a	*b*	*match*	*error*	
0	0	0	0	Error-free operation
1	1	1	0	
0	1	1	1	False negative errors detected and masked
1	0	1	1	
0	1	1	1	False positive errors detected but forwarded
1	0	1	1	

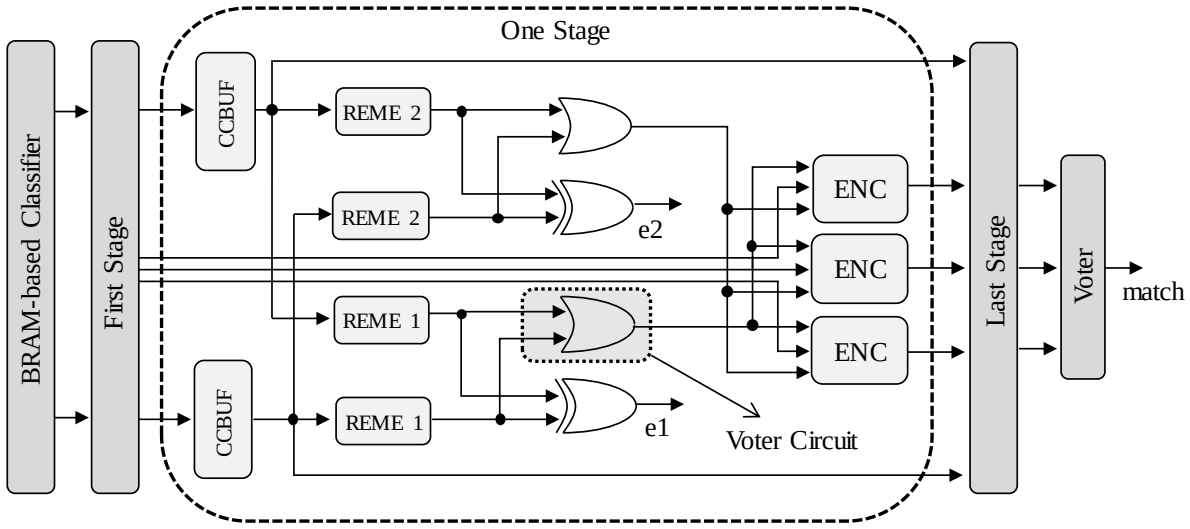
Source: the author.

logic states needed to forward a match in a REME is much more simple than generating a non-existent matching chain. Therefore, it is expected that a fault-tolerance solution designed to avoid false negatives will mask the majority of errors. Nevertheless, this assumption is supported by the fault injection results presented in Section 6.3.

This solution can be implemented by *OR*ing the outputs of the duplicated modules. In other words, if the duplicated modules diverge, a match will be assumed by the *OR* function. Thus, if a critical error occurs (causing a false negative), it will be masked and the correct output will be forwarded (positive match). Note that, depending on how the match alerts are managed by the network service using the REMEs, this approach may affect the system performance, since false matching alerts are always forwarded. Thus, the time window it remains in error state should be as small as possible, by applying adequate fault correction mechanisms. For that purpose, an *XOR* gate provides an error detection signal for each DMR domain, with the aim of triggering the error correction mechanism that will be discussed in Section 4. Figure 3.3 illustrates how the *OR*-based voter works.

The character classifier buffers are also duplicated to avoid error propagation to two domains of the same duplicated REME. A TMR technique is applied on priority encoders to not only detect errors but also to determine the correct output through a majority voter, since

Figure 3.4 – Resilient REM architecture solution to mask false negative failures.



Source: the author.

they are critical elements that aggregate all match alerts through the stages. The voting of the TMR scheme is performed only on the outputs of the last stage, aiming to reduce single point of failures. It is important to note that each stage may have dozens of REMEs, but has just one priority encoder, which represents a small fraction of the resources needed. Therefore, the TMR scheme can be applied without relevant drawback on resource usage and power consumption. Although the BRAM-based character classifiers represent single points of failures, as they feed multiple REMEs in the architecture at hand, faults in such elements are not considered for evaluation. The reason is because the built-in ECC is very efficient and can be easily applied to offer protection for their content without the need of hardware redundancy. Figure 3.4 illustrates the solution adopted to mask false negatives (*OR*-based voter) with two REMEs per stage. Comparing this solution with the baseline design, the area overhead is expected to be a little more than 100%, mainly due to the duplicated REMEs since, as explained, these elements represent the big majority of the resources needed.

The reliability model presented in Section 3.2.1 allows to estimate the system failure rate by setting three parameters (α, β, and γ) used to estimate the number of sensitive bits related with false negative or false positive failures (or both failure modes). More precisely, these parameters can be deduced from an architectural analysis of the expected fault-tolerant mechanism effect on each failure mode. With the scheme herein proposed, whenever a REME produces a match alert, including false ones generated by faults, it will be forwarded to the system's primary outputs. Therefore, the duplication of the design sensitive area also doubles the probability of false positive failures ($\beta = 2$), as showed in Figure 3.3. On the other hand,

false negatives should be completely masked ($\alpha = 0$). Moreover, sensitive bits that trigger both failure modes in the baseline design, in this solution will trigger only false positives ($\gamma = 0$). Therefore, the number of sensitive bits for the fault-tolerance solution showed in Figure 3.4 can be estimated as

$$b = 2(N_{fp} + N_{fnp}), \tag{3.4}$$

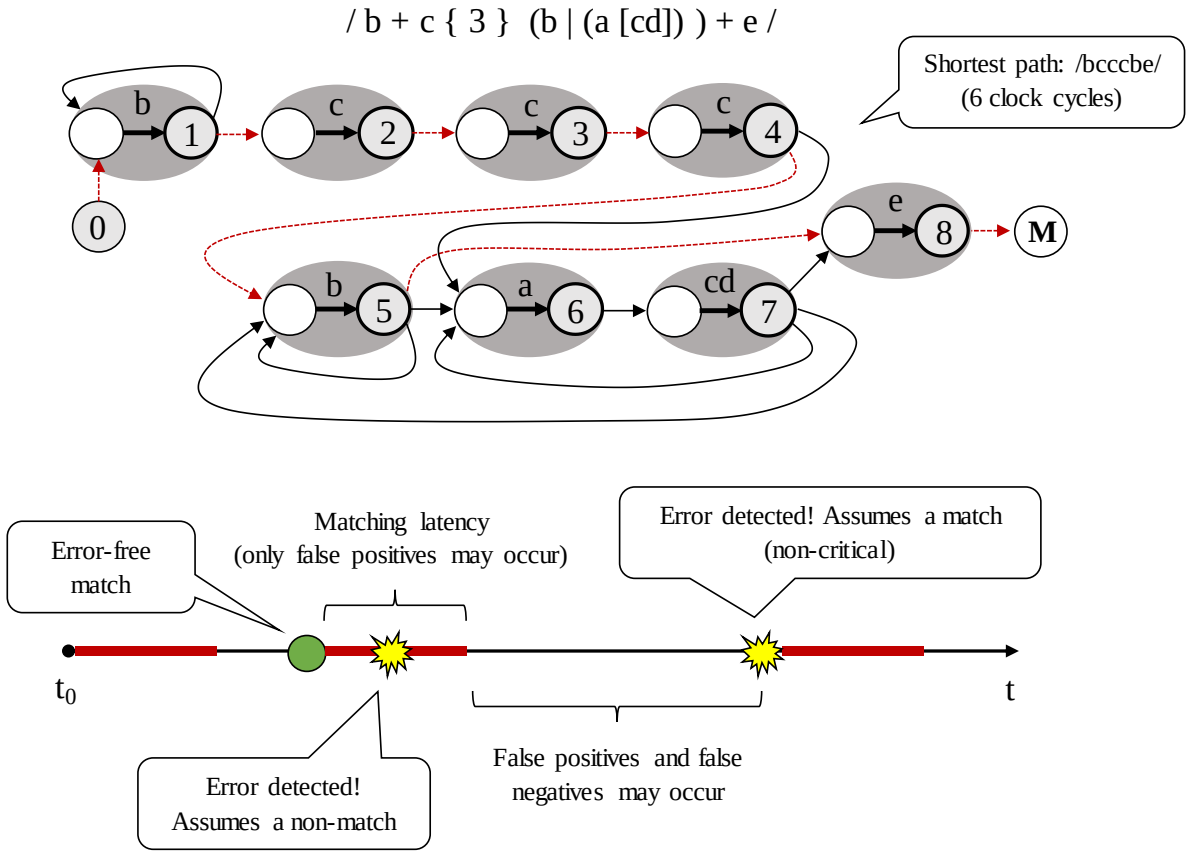
where N_{fp} and N_{fnp} are the number of sensitive bits triggering only false positives and both failure modes in the unhardened design, respectively, as explained in Section 3.2.1. Note that Eq. (3.4) is an estimation that assumes a perfect fault coverage, i.e., disregarding faults that may lead REMEs of the same DMR domain to fail in the same way (common mode failures). Moreover, faults that may affect the voters/comparators (very small area) or defeat the TMR scheme applied on priority encoders are not considered as well, since they are very unlikely to happen. Nevertheless, Eq. (3.4) will be recalled in Section 6.3 to compare the failure rate obtained with the proposed model and the failure rate obtained with fault injection experiments.

3.2.3 Masking False Positive Failures

Although the fault-tolerance scheme presented in Section 3.2.2 is able to mask critical failures for NIDS (false negatives), reducing the rate of false positives as well is important to reduce the overall failure rate and may be beneficial for other applications, especially because the criticality associated with each failure mode depends on the service being implemented. For instance, false positives may be critical in network forensics applications related with capturing information of crimes for law enforcement (RUAN et al., 2011). In such cases, instead of the *OR* function, used as a voter to mask false negatives when an error is detected by the DMR scheme, we can use an *AND* function to mask false positives in a similar way. Therefore, the values attributed to the α and β parameters of the failure rate model proposed in Section 3.2.1 are inverted ($\alpha = 2$ and $\beta = 0$), while the parameter δ remains the same ($\gamma = 0$), since false positives are not forwarded. However, decreasing the rate of critical failures at the cost of increasing the rate of non-critical ones could be problematic in some cases. While effective in dealing with critical failures, the solutions proposed so far may have a major impact on the system overall failure rate if, for example, the baseline design is much more sensible to faults triggering non-critical failures.

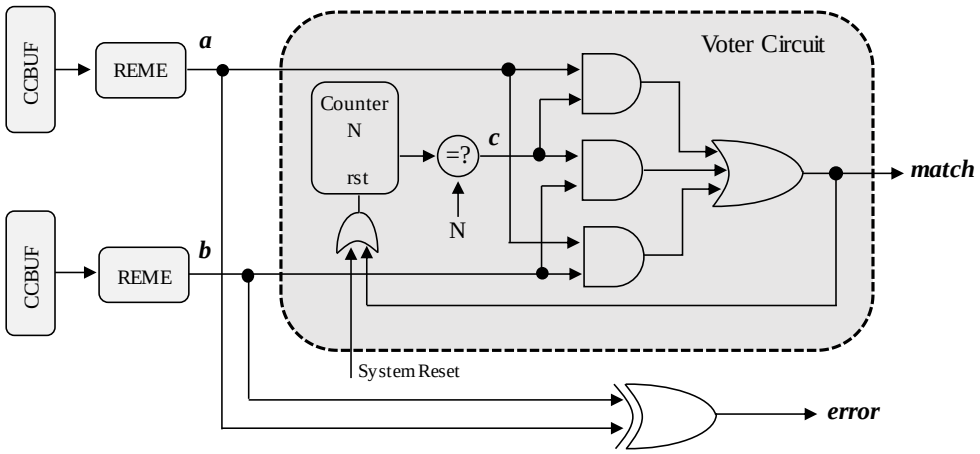
Since in this work the underlying service considered for using the REMEs are NIDS, the *OR*-based voter circuit can be improved to reduce not just the rate of false negatives (primarily), but also the rate of false positives. In this way, the DMR scheme can reduce the overall failure rate even more by increasing the fault coverage. To deal with false positives, we propose to replace the *OR* function by a voter circuit that considers the fact that each regular expression represents a set of exact string patterns. Traversing the finite state machine (NFA) shortest path from the initial state to the final state will match the shortest exact string represented by the regex pattern. As the REMEs process one or more characters per clock cycle, there is a minimum number of clock cycles before a match can occur. If a match occurs before that, it is a false positive for sure and should not be forwarded. Figure 3.5 illustrates the finite state machine for the regex pattern */b+c{3}(b|(a[cd]))+e/* and its shortest path (in red, dashed), which corresponds to the shortest string that can be matched. Observe that the number of clock cycles needed to match this string represents the matching latency of the related REME. If an error is detected during this time window, the voter should forward a non-match result instead of always assuming that the correct result is a match.

Figure 3.5 – Voter circuit behavior when an error is detected during the matching process.



Source: the author.

Figure 3.6 – Counter-based voter circuit to choose the most likely correct output.



a	*b*	*c*	*match*	*error*
0	0	x	0	0
0	1	0	0	1
1	0	1	1	1
1	1	x	1	0

Source: the author.

Therefore, we propose a counter-based voter circuit wherein a match can be forwarded only after the minimum number of state transitions is achieved. When the outputs of duplicated REMEs diverge, the decision circuit will forward the most likely correct output according to a signal generated by comparing the counter output with the minimum number of state transitions needed to generate a match. Whenever a match is signaled by the voter circuit or the analysis of a new packet starts, the counter is reinitialized and the matching latency period starts again. In other words, the most likely correct output is a positive match only after the minimum matching latency of the related REME, preserving the voter capacity in masking false negatives. Figure 3.6 shows the proposed voter. Note that the counters can be implemented by previously calculating the shortest paths of the NFAs at design time. Also, the comparator output is considered only if the duplicated REMEs diverge.

The reset scheme proposed for the counters, which restarts the matching latency period when the system is reinitialized or a match is assumed, works properly if we assume that the REMEs do not perform overlapping matches. However, if overlapping matches are possible, which depends not just on the REMEs implemented but also on dynamic factors such as the sequence of input symbols being processed, the matching latency may change after the occurrence of a true positive. In such cases, the counter reset should be controlled in a different way to prevent true positives from being ignored after previous ones having reinitialized the

counter, which may result in false negatives. The assumption that overlapping matches will not occur, or at least are very uncommon, is based on the REMEs being implemented (NIDS) and supported by the experimental results presented in Section 6.3, wherein the number of false negatives observed with the counter-based voters is almost the same as the observed with the *OR*-based voters. Therefore, the counter reset scheme herein proposed is cost-effective for NIDS, since it uses a single LUT (*OR* function) without diminishing the voter's ability to mask false negatives. As mentioned, however, this reset scheme requires a carefully evaluation, since it may be inadequate for other systems.

The additional resources needed to implement this solution are relatively low. The counter can be implemented using the carry chain circuits available in each CLB and requires at most $\lceil \log_2 n \rceil$ LUT-FF pairs, where n is the number of state transitions in the REME shortest path. On the other hand, the comparator and the combinational circuit can be implemented with $\lceil (\log_2 n)/k \rceil$ and $\lceil 6/k \rceil$ k-input LUTs, respectively. Therefore, the additional resources needed to implement are a small fraction of the total resources needed to implement the associated DMR domain.

It is important to observe that during the system operation, assuming that faults occur randomly, the matching latency of each implemented REMEs will define the likelihood of faults triggering false positives being masked. If a false positive occurs outside of this matching latency window, as explained, it will be forwarded to the system's primary output. Moreover, the network service that uses the REMEs is responsible for resetting the counters (system reset) when a match does not occur for a long time, either because of harmless packets being processed or the absence of detected errors during the period in which matches are assumed. Typically, the REM system is reinitialized after processing either a single packet or an entire flow, which depends on how the network service manages the REMEs. Therefore, the parameter β of the failure rate estimation model presented in Section 3.2.1 can be defined only after knowing these details, which may be somewhat difficult due to dynamic factors. Nevertheless, the failure model for the resilient design herein proposed will be experimentally defined in Section 6.3 considering both voters (*OR*-based and counter-based).

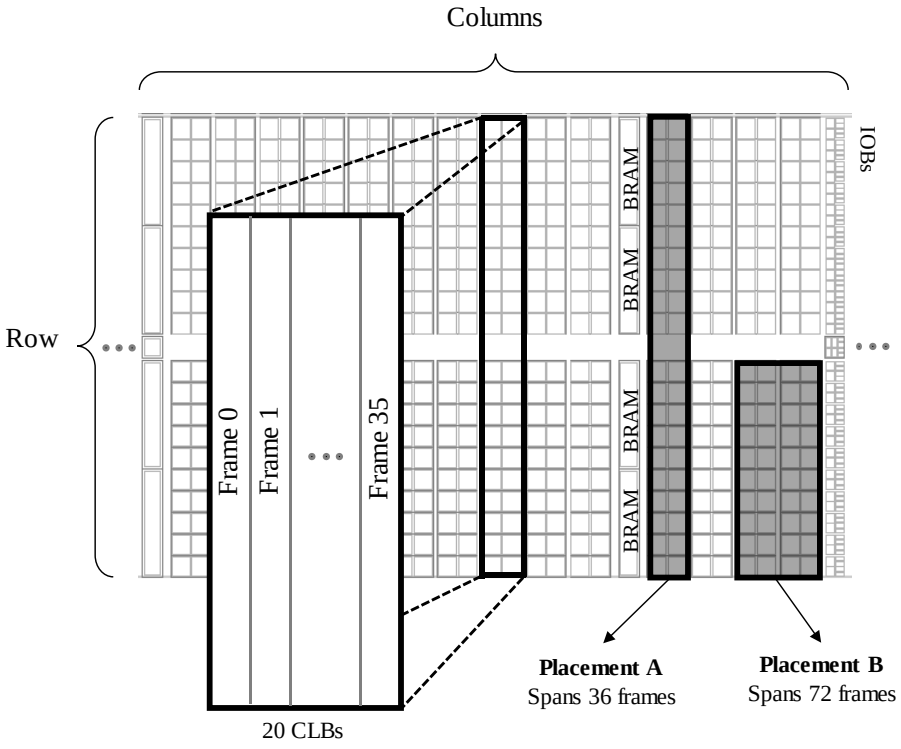
4 ERROR DETECTION AND CORRECTION

The mechanism we propose for scrubbing the configuration memory and correcting flipped bits relies on error detection at design level and DPR. The DMR scheme provides an error flag through an *XOR* gate that compares the outputs of the duplicated REMEs. Therefore, an error signature for the whole system is generated. To reduce the system repair time, we propose to establish a relation between error signatures and the design placement, in such a way that a given signature triggers the scrubbing procedure to selectively rewrite only the configuration frames related with the DMR domain affected. Thereby, when the design detects an error, it is not needed to reconfigure the whole FPGA, but rather only the faulty part through partial reconfiguration. In this way, the time during which output failures might persist is minimized, increasing the system availability and minimizing the time exposed to potential security threats. In (NAZAR, 2015; NAZAR; SANTOS; CARRO, 2015; REORDA; STERPONE; ULLAH, 2016; SANTOS; NAZAR; CARRO, 2016b), error detection signals were used to minimize repair time. Those techniques made use of user-circuit-level error detection, but were either placement-oblivious (NAZAR, 2015; NAZAR; SANTOS; CARRO, 2015; SANTOS; NAZAR; CARRO, 2016b), or imposed very strict placement constraints that may severely affect post-routing frequency (REORDA; STERPONE; ULLAH, 2016). In this work we propose a set of looser placement constraints to greatly enhance the precision of the error location mechanism with reduced impact on operating frequency.

To define an efficient placement strategy, first we need understand the configuration memory organization and its relation with the FPGA resource layout. In this work we use Xilinx FPGAs as a proof of concept, although the technique is applicable to any FPGA that supports partial reconfiguration. The structure of modern SRAM-based FPGAs from Xilinx is divided into rows (clock regions), where each row is divided into columns that correspond to a column of basic blocks in the array (CLB, DSP, block RAM, IOB, etc.). Each column is configured by a certain number of frames, which depends on the block type. A frame can be thought as a vertical stack that spans the whole height of a row, as illustrated in Figure 4.1, and can be accessed by a specific addressing scheme that depends on the FPGA family. For instance, the memory frames of a CLB column configure elements such as LUTs, flip-flops, multiplexers, and the interconnection between them.

In SRAM-based FPGAs, DPR can be performed through the SelectMAP interface or the ICAP. Those interfaces allow, respectively, external devices or user logic to access the configuration memory to read/write a set of contiguous frames starting from a given address. A

Figure 4.1 – Relation between resource layout and configuration frames in Xilinx Virtex-5 FPGAs. Different placements may result in different repair times due to the number of frames used.



Source: the author.

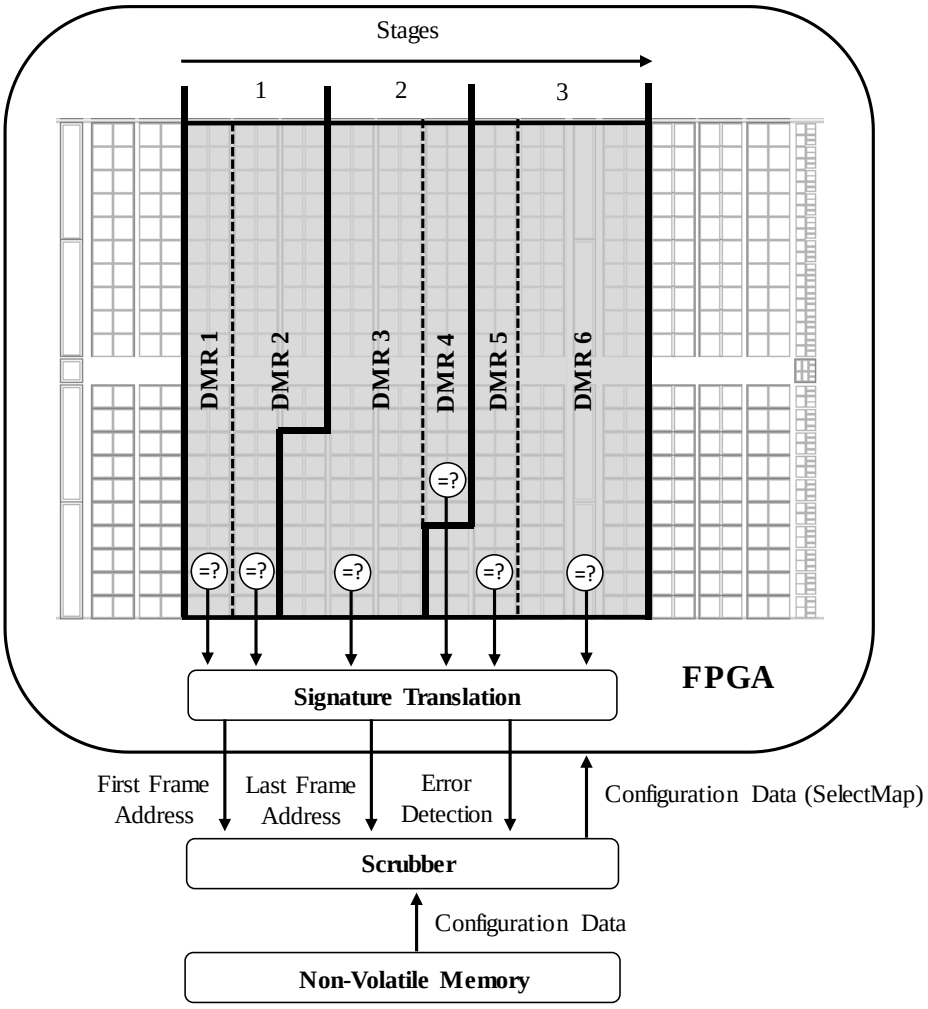
frame is the smallest amount of configuration information that can be accessed, so the repair time will depend not just on the SelectMAP/ICAP bandwidth, but also on the number of frames between the frame defined to start the scrubbing and the faulty one. If the placement of a given circuit is constrained to a specific region, then the number of frames to scrub will depend on the number of columns covered by that region, since the mapping between individual frames and specific resources is not available from vendors. For instance, a circuit placed in the bottom-half of two columns uses twice as many configuration frames as the same circuit placed in one entire column, as illustrated by placements A and B in Figure 4.1.

Therefore, we propose a placement strategy wherein each REME and its copy are vertically placed and grouped accordingly to the stage that they belong, aiming to use the minimum number of columns as possible. In this way, the number of configuration frames that each DMR domain spans will be limited by the resources needed. Based on a given error signature, which identifies the faulty DMR domain, the scrubber can perform a partial reconfiguration comprising only the configuration frames of the region where that domain was placed, reducing the repair time.

An important observation is that design elements shared by multiple DMR domains, such as priority encoders and character classifier buffers, are not included in the placement constraints. In each stage, the priority encoder aggregates the outputs of all DMR domains and the match result of the previous stage, while a character classifier buffer forwards its content to the next stage at every clock cycle and may feed several REMEs. Therefore, their placement is essential to define the system critical path. Moreover, those elements represent less than 5% of the total area usage, which means that the probability of a configuration fault affecting them is very low. Even if an error occurs, its localization is difficult due to error propagation. For instance, depending on the system state and the input symbols, a fault that changes the configuration of a character classifier buffer may not affect the nearby REMEs (logical masking) but may affect the REMEs in later stages. Regarding errors in priority encoders, they are masked in the design herein proposed, due to the TMR scheme and do not trigger local scrubbing. To avoid accumulation of faults, a global preventive scrubbing can be performed periodically. For all those reasons, constraining the placement of character classifier buffers and priority encoders may result in high performance degradation without a significant impact in the system repair time.

Although the error signature may offer a reliable localization of errors originated in the REMEs, some of them may not be detected. If two domains of the same REME are affected at the same time (common mode failures), the mitigation scheme fails and the design is not able to detect the error. Such cases may happen when design elements of different domains share resources such as routing matrices. For instance, a single fault affecting a Programmable Interconnect Point (PIP) may generate multiple effects, such as the deletion of a given routing and the addition of another one. To minimize this problem, the proposed placement reduces the sharing of routing matrices between domains of the same REM engine by placing them in different sets of CLBs. More specifically, after defining the resources needed to implement each DMR domain, the set of slices of the region where a given domain will be placed is divided between the duplicated REMEs in such a way that slices of the same CLB are not designed to different instances. The reason is because the input and output signals of each slice in a given CLB passes through the same routing matrix. Therefore, if REMEs of the same DMR domain are placed in the same CLB, faults in PIPs of this routing matrix may result in common mode failures. In other words, the REMEs placement in the FPGA resource layout is made at CLB-level granularity. Note also that the error detection does not guarantee the correct localization of the faulty frame. Therefore, the design error detection scheme combined with partial

Figure 4.2 – Placement solution proposed.



Source: the author.

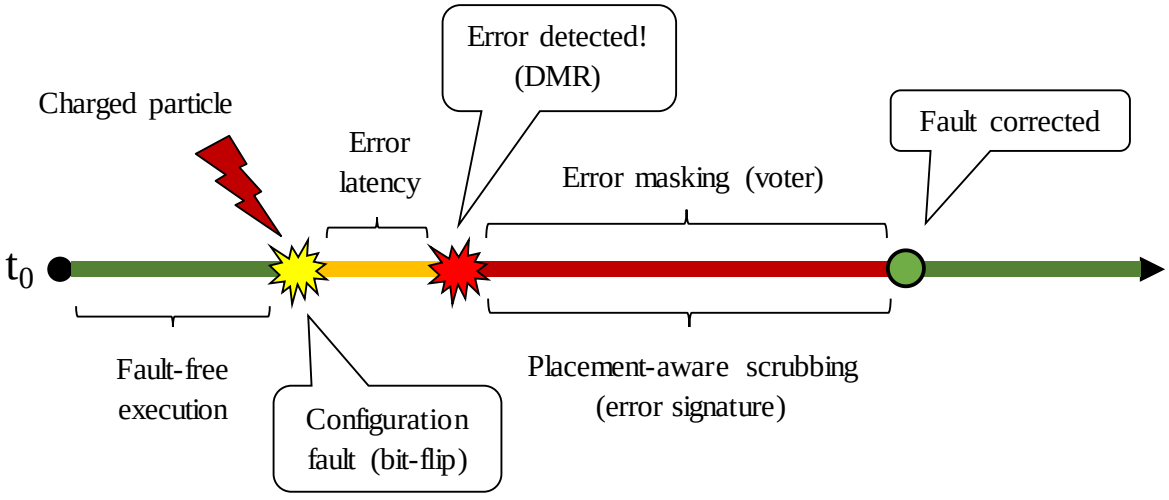
reconfiguration may not be sufficient to repair the system in some cases, which reinforces the need of periodic global scrubbing as a countermeasure.

Figure 4.2 illustrates the placement-aware reconfiguration strategy. The placement constraints define reconfigurable regions for each DMR domain, aiming to cover the minimum amount of resource columns as possible, similarly to (SARI; PSARAKIS, 2011; SCHMIDT; ZIENER; TEICH, 2014). In each domain, a single error flag is provided by a comparator. Also, DMR domains that process the same symbol at the same clock cycle, i.e., that belongs to the same stage and share character classifier buffers, are placed contiguously to minimize the delay penalty. Note that the restrictions imposed to the placement tool can be loosened by generating reconfigurable regions comprising the stages instead of individual DMR domains, which may reduce the delay overhead at the cost of increasing the repair time. As mentioned, fine-grained solutions with stringent placement constraints often impose severe performance degradation,

while coarse-grained solutions limit the system capacity to localize errors, which may increase the system repair time due to bigger reconfigurable regions (NAZAR; SANTOS; CARRO, 2015; SANTOS; NAZAR; CARRO, 2016). Therefore, the hybrid redundancy scheme (DMR and TMR) and the placement strategy herein proposed consider the particularities of the design at hand to manage area and delay costs.

Regarding the scrubbing methodology, when an error is detected, i.e., there is at least one error flag active, the scrubber can be triggered to perform partial reconfiguration. In that case, the scrubbing can be performed on the region defined by the Signature Translation, which translates the error signature in three parameters: the first and the last frame address to scrub and an error indication signal. Details about how this module is implemented are presented in Chapter 5. As mentioned previously, the error detection does not guarantee the correct localization of the faulty frame. Thus, if the error remains after the scrubbing, a global reconfiguration is needed. Moreover, periodic global reconfiguration can be used to correct errors that affect the signature translation module or that are not detected, as mentioned. As will be showed in Chapter 6, however, the majority of faults are detected and can be corrected with partial reconfiguration. Therefore, the use of global reconfiguration will have a negligible impact on the system repair time. Note that faults on the signature translator are not a concern because it does not affect the rest of the system, i.e., the REM system will continue to deliver the results correctly. Errors in this module may be a problem only if multiple faults accumulate in the timeframe between two error correction procedures, affecting the REM system and the error correction mechanism at same time. This situation, however, is very unlikely to happen if the period of the periodic global scrubbing is set properly, i.e., it is sufficiently shorter than the

Figure 4.3 – System behavior when experiencing configuration faults.



Source: the author.

Table 4.1 – Assumptions and restrictions considered for the proposed techniques.

	Error Masking Mechanism	***Scrubbing Mechanism***
Restrictions	Common mode failures and errors in voters/comparators are not covered.	Priority encoders and character classifier buffers are not included in the placement constraints (very small area).
Assumptions	Critical failures (false negatives) are the most common and overlapping matches will not occur (NIDS).	Errors not detected or affecting the signature translator can be corrected with periodic global scrubbing (not implemented).

Source: the author.

raw fault rate. Figure 4.3 illustrates the fault-tolerance mechanism and the scrubbing methodology used to improve the system's reliability and availability, while Table 4.1 summarizes the restrictions and assumptions considered for the techniques herein proposed.

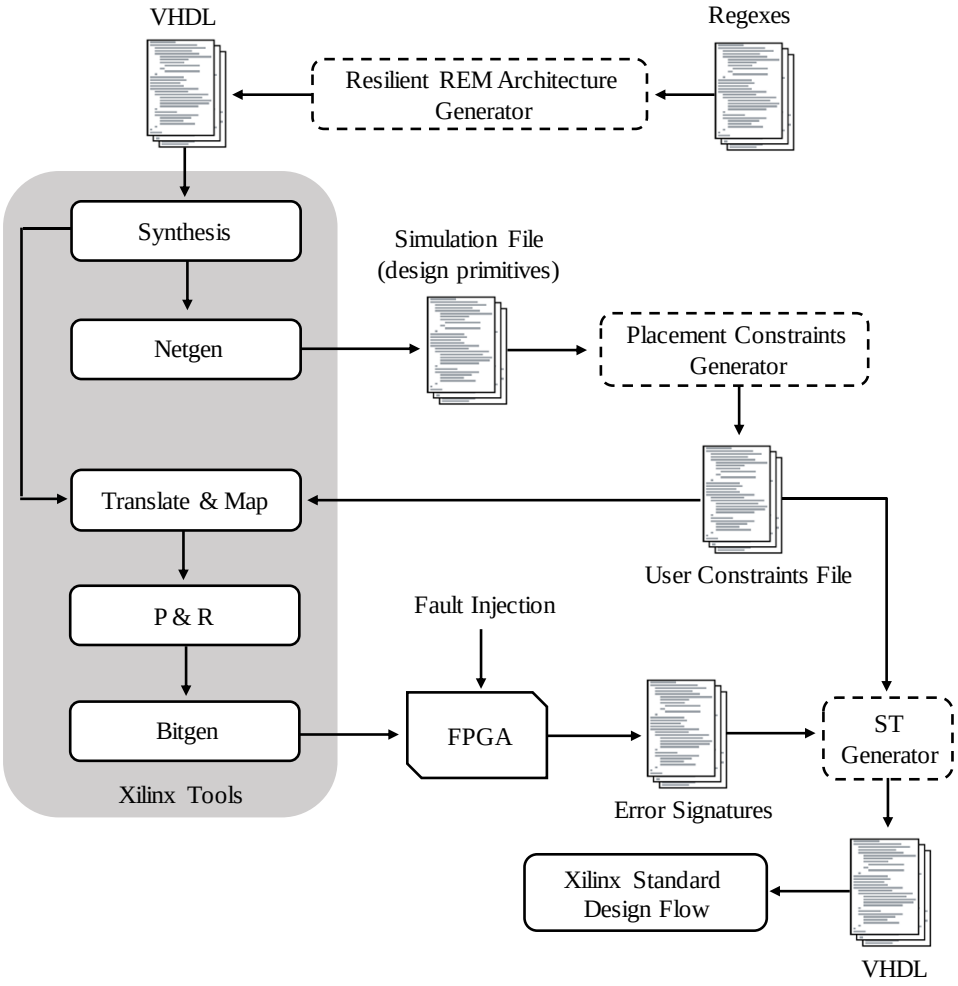
5 DESIGN FLOW

In this chapter, we describe the design flow developed to obtain the resilient REM architecture described in Chapters 3 and 4. The design flow is divided in three main steps, described in the next sections, and uses the Xilinx standard tools and fault injection experiments. The design flow is illustrated in Figure 5.1. The shaded rectangles represent specific tools implemented in this work, aiming to generate he REM architecture with the fault-tolerance and scrubbing techniques herein proposed.

5.1 Resilient REM Architecture Generation

The hardware description (VHDL) of the baseline REM architecture can be generated through the implementation tool presented in (YANG; PRASANNA, 2012). First, the regular expressions to be implemented (REMEs) should be described in a text file and parsed by a C++

Figure 5.1 – Design flow.



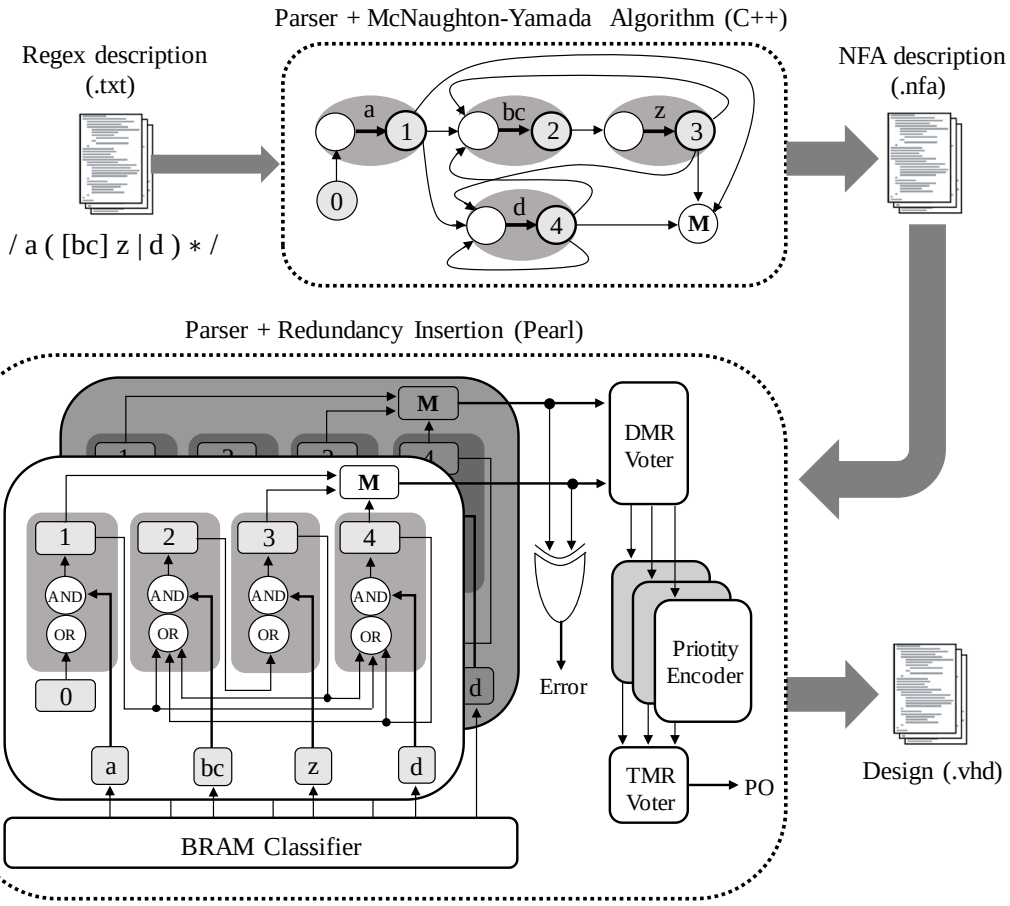
Source: the author.

program. This program translates the regular expressions in NFAs through the McNaughton-Yamada algorithm (MCNAUGHTON; YAMADA, 1960), and outputs a second file (.nfa) describing them in an specific manner. Next, this second file is parsed by a Perl program that generates the VHDL files. To obtain the resilient REM architecture herein proposed, only the Perl program is modified, aiming to duplicate the REMEs (DMR) and to triplicate the priority encoders (TMR), as well as to insert the voter circuits for each redundant domain and the comparators for error detection. Figure 5.2 shows the steps needed to generate the resilient REM architecture described in Section 3. Note that the module responsible for triggering the scrubbing mechanism is inserted in the next steps, since it depends on the design's placement.

5.2 Placement Constraints Generation

The VHDL design files generated by the implementation tool described in Section 5.1 are ready to be synthesized. After the synthesis step, a specific placement of the synthesized circuit is necessary to use the placement-aware scrubbing technique proposed. Therefore, the

Figure 5.2 – Resilient REM design generator.



Source: the author.

design flow has a deviation from the Xilinx standard design flow, since the placement constraints can be defined only after the amount of resources needed for each module is known. Once the design net-list is generated by the standard synthesis tool, the Xilinx *netgen* tool is used to generate a post-synthesis simulation file wherein the circuit is described with design primitives (LUTs, FFs, etc.) of the target FPGA. A Python program was written to parse this file to obtain the type and count of primitives used by each module of the design. Given the resources needed to implement each DMR domain, a user constraint file is generated to guide the mapping tool to place the REMEs as described in Chapter 4.

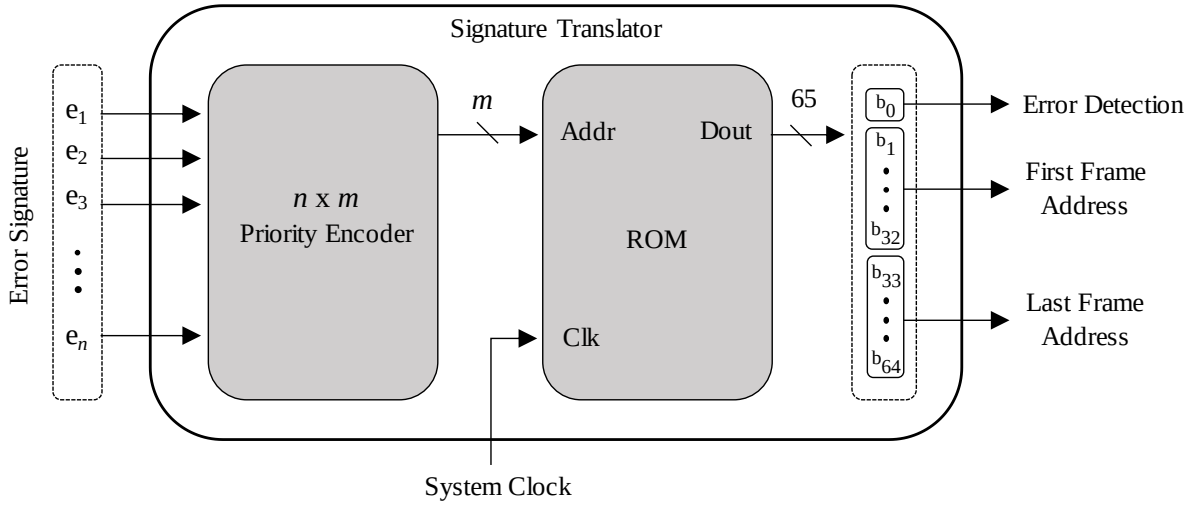
More specifically, the *AREA_GROUP* constraint is used to bound groups of slices in the FPGA resource layout where each REME and its copy should be placed, comprising the number of resources (LUTs and FFs) needed to implement each DMR domain. As explained in Chapter 4, only the duplicated REMEs are included in the placement constraints to avoid unnecessary delay penalties, since these elements uses the majority of the resources needed, as will be shown in Section 6.2. In this way, the mapping tool is able to optimize the placement of other elements. Additionally, the placement scheme avoids CLB sharing between REMEs of the same DMR domain, as the input and output signals of each slice in a given CLB passes through the same routing matrix. The goal is minimizing the probability of common mode failures resulting from errors in PIPs of such routing matrixes.

Note that the standard design flow does not allow constraining the routing. If an error is detected in a given DMR domain due to faults affecting routes leaving the constrained area, the scrubber will be triggered to reconfigure the wrong region and therefore a global scrubbing will be needed, increasing the system repair time. This problem can be solved with the Xilinx Isolation Design Flow (IDF) (XILINX, 2017a), for example, by setting the property *CONTAIN_ROUTING* in the *AREA_GROUP* constraint. However, the DMR domains must be configured as isolated partitions, which may impose high area and delay overheads. Since the number of failures resulting from faults affecting routes leaving the constrained area is expected to be very low, we assume that the MTTR improvement does not justify the overheads associated with the IDF (at least for the design at hand). This assumption is supported by the results presented in Section 6.4, wherein a very small fraction of the detected errors cannot be successfully corrected with DPR, meaning that the impact of this limitation on the system MTTR is negligible.

5.3 Signature Translator Generation

The hardware description (VHDL) of the signature translator (ST) module is inserted only after an exhaustive fault injection campaign is performed. For designs with dozens of DMR domains, the implementation of this module may be infeasible due to the amount of different error signatures. However, given the system architecture and the placement strategy proposed, it is expected that just a small fraction of the possible error signatures will be generated. Therefore, we used the fault injection results to compress the error signature, in a similar way that it is done in (NAZAR; SANTOS; CARRO, 2015). We find that more than 96% of the error signatures observed have just one error flag active, so the error flags are priority encoded in the same way that it is done with the match signals of each DMR domain. The priority encoder will deliver an error signature of $m = log_2 n$ bits, where n is the number of DMR domains implemented. This encoded error signature is then used for addressing a memory (ROM) that contains the frame addresses required to perform the partial reconfiguration and the error detection signal used to trigger the scrubber. This memory delivers a 65-bit signal in which the least significant bit represents the error detection signal, the next 32 bits represent the first frame address, and the 32 most significant bits represent the last frame address to scrub. Those frame addresses are obtained from the constraint file previously generated, i.e., given that the region where each REME was placed is known, we can define the first and the last frame address of the region accordingly to the addressing scheme of the target FPGA. Figure 5.3 shows the Signature Translator module.

Figure 5.3 – Signature translation module implementation.



Source: the author.

6 EXPERIMENTAL RESULTS

In this chapter, we present results obtained with the fault-tolerance mechanisms herein proposed. First, the experimental setup used to evaluate the effectiveness of the proposed techniques is presented and described in details. Next, we evaluate the area and delay overheads with regard to the baseline REM architecture. Finally, we evaluate the failure rate achieved with each voter design presented in Chapter 3, as well as the repair time obtained with the scrubbing mechanism proposed in Chapter 4.

6.1 Experimental Setup

The fault model considered for evaluating the reliability and the repair time of the design herein proposed is soft errors (bit-flips) on the configuration memory of SRAM-based FPGAs. This fault model was chosen as these are the most common faults for SRAM-based FPGAs, mainly due to the large size of the configuration memory. Those bit-flips may generate complex and persistent failures by changing the configuration of routing interconnections and design elements. Therefore, the design susceptibility must be closely investigated, since configuration faults are a major reliability concern for SRAM-based FPGAs. To do so, we insert single bit upsets within the configuration memory using the fault injection platform presented in (LEIPNITZ; HESS; NAZAR, 2016), which is adequate for evaluating high-throughput systems. This platform is composed of a circuit embedded in the FPGA that manages the ICAP to perform sequential or random fault injection by flipping the configuration memory bits of a specific area, defined as the Area Under Test (AUT). A program running on a host computer communicates with the FPGA to send commands and stimuli vectors and to receive the results for evaluation and the system status for monitoring.

6.1.1 REM Circuit

A set of regex patterns from the Snort rule sets was chosen to generate the REM circuit. The criteria used to select the regexes was based on heterogeneity, i.e., we avoid the inclusion of distinct regexes with common prefixes, and choose regexes of different lengths (number of states) that include all regex operators supported by finite automata: concatenation (-), Kleene closures (*), unions (|), repetitions (+), constrained repetitions {m, n}, optionality (?), start of string (^), end of string ($), complex character classes ([…]) and inverted character classes

([^…]). The goal is to construct an REM circuit that uses all the features and structures available on the baseline architecture. The result is a set of 64 regex patterns from the *web-activex*, *oracle*, *spyware-put*, *backdoor*, *exploit* and *smtp* Snort rule sets (see Appendix). In this case, the placement strategy will divide the design layout into 64 regions, one for each DMR domain. Moreover, to evaluate a system with high throughput, we implemented an REM architecture that processes two characters per clock cycle.

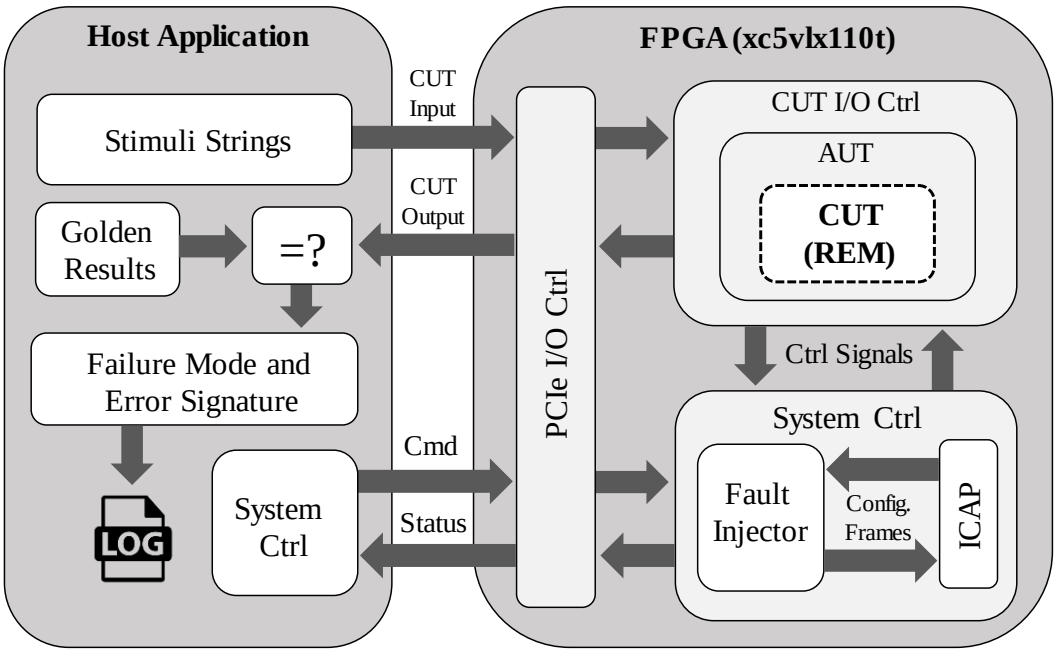
6.1.2 Stimuli Strings

We developed a software application (C++) to generate the strings to stimulate the REM circuit. The application allows the generation of random strings and also strings formed from the NFA representation of the regexes. Since the state update modules receive the character matching and classification as a 1-bit signal from the BRAM-based classifiers, they are unaware about which character is matched. Therefore, the most important condition to stimulate all parts of the REMEs is ensuring that all states of their NFA representation structures were visited when generating a set of valid strings, independently from the character generated. Taking that into account, we generate 100 valid strings for each regex pattern by randomly traversing the NFA structures. Note that the REM architecture used performs exhaustive matching, i.e., the match for the incoming characters is evaluated at every clock cycle. Thus, complete valid strings also stimulate the circuit behavior for partially valid strings. In total, 423,982 characters were applied to the system, two per cycle, for each injected fault.

6.1.3 Fault Injection

The REM hardware module was synthesized for a Xilinx Virtex-5 FPGA, with the development platform XUPV5-LX110T (XILINX, 2017b), and defined as the circuit under test (CUT) on the fault injection platform. The first step before performing the fault injection campaign was the generation of golden outputs by executing the REM circuit without any configuration fault. With the golden outputs, we send the stimuli characters from a host computer to the FPGA through the PCIe interface. In parallel, the match outputs are received and compared with the golden outputs. This process is repeated for each configuration fault injected. At the end of the fault injection campaign, we have a log file with the number of false positives and false negatives received for each configuration bit flipped, as well as the information retrieved by the signature translator, i.e., the first and last frame address of the

Figure 6.1 – Experimental setup.



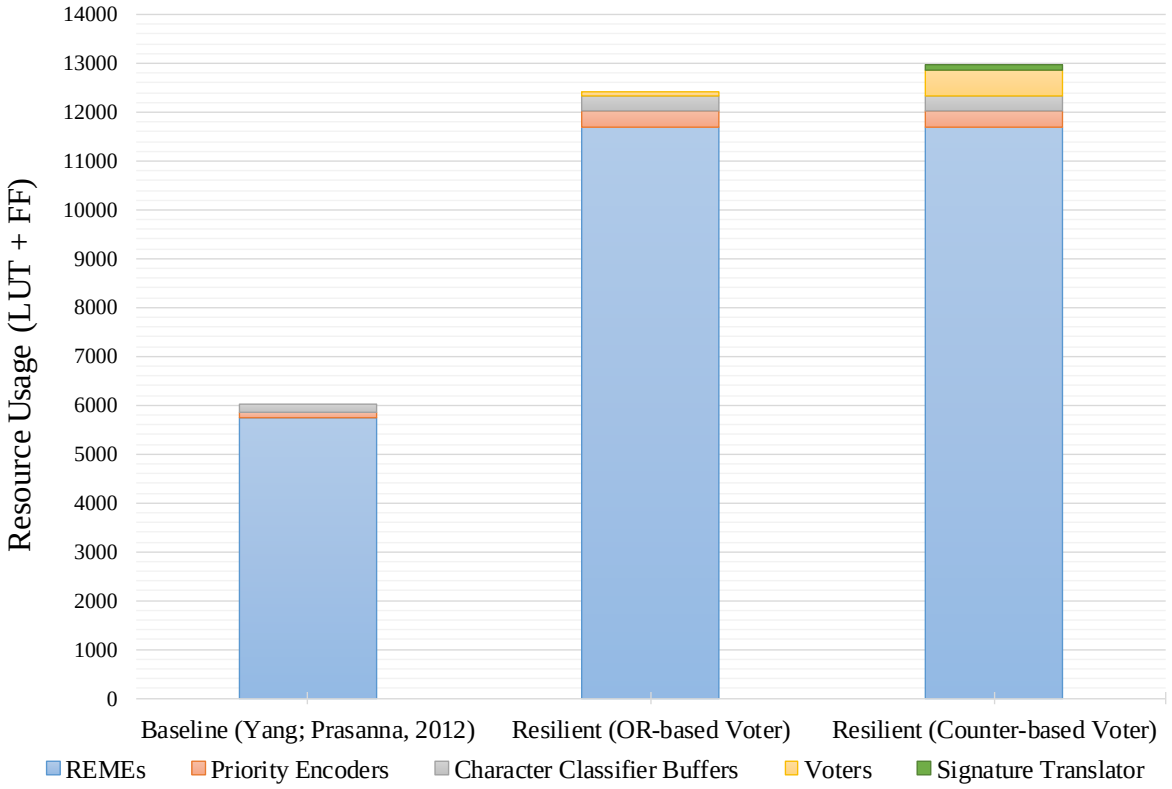
Source: the author.

region to scrub and the error flag. We make an exhaustive fault injection campaign, i.e., all configuration bits of the AUT were flipped. Figure 6.1 illustrates the experimental setup used to evaluate the design reliability and to validate the placement strategy by verifying the relation between the error signatures received and the region where each fault was injected. Note that the fault injection in each configuration bit (spatial domain) is performed only before the CUT execution, i.e., the effects of faults at different clock cycles (time domain) are not evaluated.

6.2 Area and Delay Overheads

Figure 6.2 shows the amount of resources needed (LUTs and FFs) to implement the baseline REM architecture and the two resilient versions proposed in this work (*OR*-based voter without placement constraints and counter-based voter with placement constraints), as well as the design occupation of the main modules. The synthesis tool (XST) was configured to make optimizations to reduce area usage. As expected, the additional resources used to implement the proposed solutions are mostly due to the DMR solution applied on the REMEs (93.4% of additional LUTs and 94.6% additional flip-flops). Observe that the TMR solution for priority encoders and the duplication of character classifier buffers had a minor impact, as they consume much less resources than the REMEs. All implementations use three BRAMs for character classification, but the solution with counter-based voters includes an additional one for the

Figure 6.2 – Resource Usage and Design Occupation.



Source: the author.

signature translation, as explained in Section 5.3, which is necessary to implement the scrubbing mechanism proposed. Comparing it with the baseline design, an increase of 115.8% in the number of resources needed was observed. On the other hand, the first implementation imposed an area overhead of 105.3%. Therefore, the replacement of *OR*-based voters by counter-based voters and the addition of the priority encoder needed to compress the error signature increased the resources needed in only 5.1%.

Regarding performance, the baseline version limits the clock frequency to 170MHz, while the *OR*-based voter implementation limits to 161MHz and the counter-based voter implementation limits to 141MHz. For an architecture processing two characters per clock cycle, this means a max throughput of approximately 2.73Gb/s, 2.58Gb/s and 2.26Gb/s, respectively. Therefore, the maximum clock frequency allowed in the full implementation (counter-based voter plus scrubbing) had a decrease of 17.2%. Most of this penalty stems from the restrictions imposed to the placement tool, as the version without placement constraints had a decrease of only 5.5%. In case the attained performance is insufficient, it can be improved by accepting more characters per cycle, at the cost of additional area, or relieving the placement constraints on the slowest paths of the circuit, at the cost of increased repair time.

6.3 Reliability Evaluation

Table 6.1 shows the sensitiveness of the baseline REM architecture and the resilient versions proposed in Section 3 regarding configuration faults. False negatives, which are the most critical ones from a security perspective, were by far the most frequent mode in the unhardened circuit, triggered by 92.2% of the sensitive bits. As can be seen, the hardware with the *OR*-based voter not just reduced the circuit sensitiveness to configuration faults (from 11.6% to 1.05%), but also inverted the most frequent failure mode (from false negatives to false positives). The major reduction of sensitive bits in the resilient circuit can be explained by observing the behavior of the baseline circuit: if the majority of the sensitive bits triggers false negatives, then the application of a DMR solution wherein a positive is assumed when an error is detected will mask the majority of the faults. Therefore, when the DMR solution proposed assumes a positive, it is not just avoiding the critical failure mode (false negatives), but also assuming the most likely correct output. Although almost doubling the number of configuration bits, the proposed fault-tolerance solution experienced a decrease of 99.4 % in critical sensitive bits triggering false negatives failures. This solution, however, increased the number of sensitive bits triggering false positives in 112.4%, mainly due to the area overhead, as discussed in Section 3.2.1. Whenever a false positive occurs in one of the duplicated REMEs, it will be forwarded to the system's primary output.

On the other hand, the implementation with the counter-based voter further reduced the circuit sensitiveness to configuration faults (from 1.05% to 0.28%) by masking not just false negative but also false positive failures. More precisely, false positives experienced a decrease of 54.7% with regard to the unhardened circuit. Comparing it with the *OR*-based voter design,

Table 6.1 - Resilient REM Architecture Evaluation

FPGA-based REM Version	***Number of Configuration Bits***	***Insensitive Bits (Correct Outputs) Amount and % of Total***	***Sensitive Bits (Failures)***			
			Amount and % of Total	*Amount and % of Failures*		
				False Positives Only	*False Negatives Only*	*False Positives and False Negatives*
Baseline (Yang; Prasanna, 2012)	1,162,997	1,028,058 (88.4 %)	134,939 (11.6 %)	10,533 (7.81 %)	123,934 (91.84 %)	472 (0.35 %)
Resilient (OR-based Voter)	2,255,747	2,231,976 (98.95 %)	23,771 (1.05 %)	22,764 (95.76 %)	851 (3.58 %)	156 (0.66 %)
Resilient (Counter-based Voter)	2,350,211	2,343,630 (99.72 %)	6,581 (0.28%)	5,765 (87.60%)	771 (11.72%)	45 (0.68%)

Source: the author.

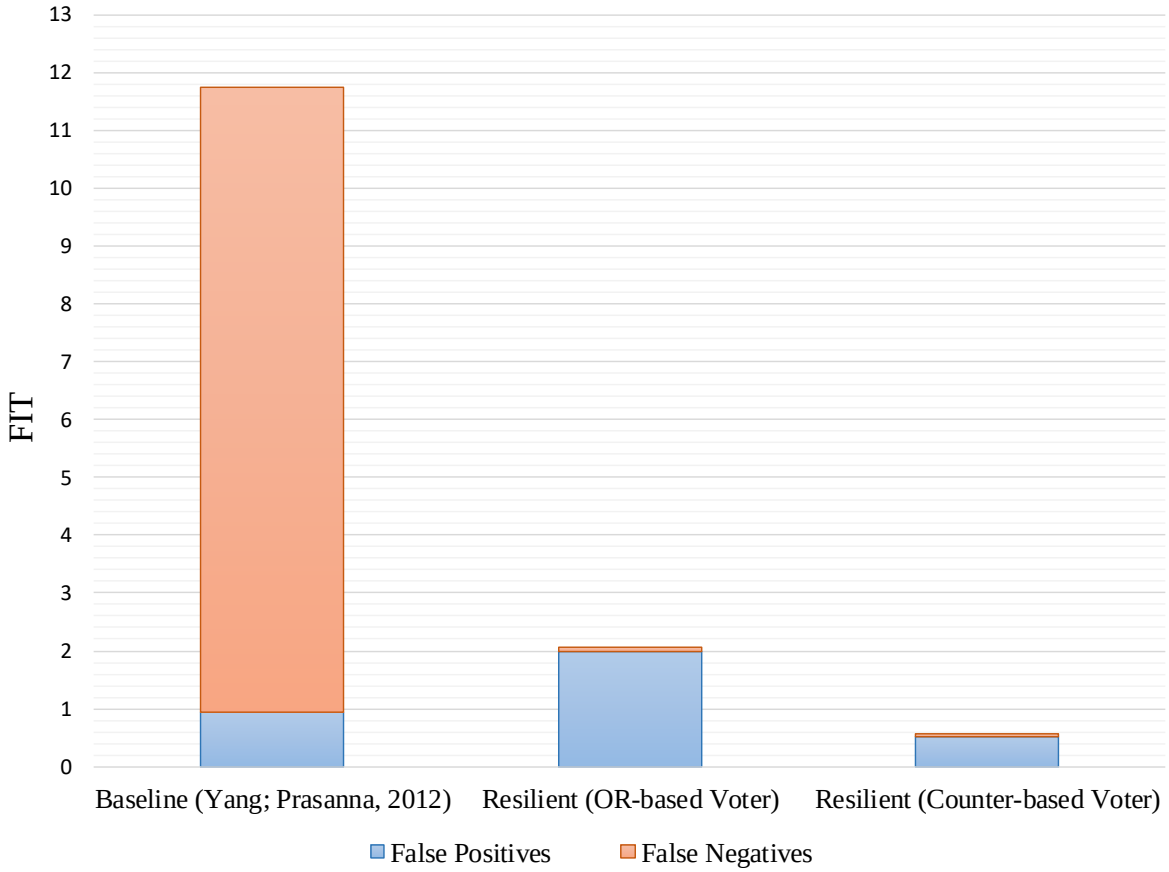
the benefit is even more evident, since a decrease of 74.7% was observed in false positives with only 5.1% of area overhead. As expected, the amount of sensitive bits triggering false negatives remained almost the same, since both voters have the same behavior in the time window in which false negatives may occur. This is an important result, as the enhanced voter does not impact the design capacity to mask the majority of critical failures.

Once we have the number of failures due to the injected faults, it is possible to calculate the failure rates of the REM implementations evaluated. Recall from section 3.2.1 that the failure rate is presented as the amount of failures expected per 10^9 device-hours (FIT) and can estimated by multiplying the neutron flux (n_{flux}), the measured static cross-section (σ_{static}), and the number of sensitive bits (b). The considered neutron flux is 13 n/(cm^2.h) (JEDEC, 2006), and the measured static cross-section for Virtex-5 devices is 6.70x10^{-15} cm^2 (XILINX, 2017c). The number of sensitive bits is given by the fault injection results presented in Table 6.1.

Figure 6.3 shows the FIT achieved by each design, as determined by Eq. (3.2). The failure rate for the unhardened implementation is 11.75 FIT, while the failure rate for the resilient version with the *OR*-based voters is about 2.07 FIT. Considering only critical failures (false negatives), the failure rate is about 10.79 FIT and 0.074 FIT, respectively. Therefore, this fault-tolerance solution experienced a reduction of about 82.4 % in overall failure rate and of about 99.31 % regarding critical failures. However, the failure rate of false positives more than doubled, from 0.96 FIT to 1.988 FIT, as expected from the analysis in Section 3.2.2.

Once we have the failure rates obtained through fault injection experiments, more specifically the number of sensitive bits for each failure mode in the baseline design (N_{fn}, N_{fp}, and N_{fnp}), Eq. (3.4) can be used to determine the theoretical number of sensitive bits in the hardened solution with *OR*-based voters. The theoretical overall failure rate calculated is about 2.01 FIT, which represents a difference of only 3 % with regard to the experimental one (2.07 FIT). This difference is due to common mode failures on the redundant modules (DMR and TMR) and faults affecting the voter circuits, which are not considered in the theoretical model. The experimental α and β parameters from Eq. (3.3) can be calculated as well, and compared with the theoretical ones deducted from the architectural analysis presented in Section 3.2.2 ($\alpha = 0$ and $\beta = 2$). Dividing the failure rates of the resilient version by the rates of the baseline version we obtain that $\alpha \cong 0.007$ and $\beta \cong 2.07$. This result shows that the reliability model proposed in Section 3.2.1 can offer a good approximation for the system failure rate, through

Figure 6.3 – FIT evaluation.



Source: the author

an early architectural analysis of the fault-tolerance mechanism applied. Note that the number of sensitive bits triggering both failure modes is a very small fraction (less than 1%) of the total failures observed. Therefore, for the system evaluated, the γ parameter cannot be properly inferred, since the faults not covered by the theoretical model are representative for the total number of failures observed in this category. However, the impact of this situation on the overall failure rate estimation is negligible, since such faults are very unlikely to happen. With those considerations, this analysis can be used to provide early reliability estimations for other applications aiming to benefit from the techniques herein proposed.

Regarding the resilient implementation with the counter-based voters, the failure rate achieved is about 0.57 FIT. Considering only critical failures (false negatives), the failure rate is about 0.067 FIT. Therefore, this solution experienced a reduction of about 95% in overall failure rate and of about 99.4% regarding the rate of critical failures if compared with the baseline design. Observe that the reduction in the rate of critical failures is similar to the reduction obtained with the *OR*-based voter implementation (difference of 0.07%). On the other hand, the new voter design is able to reduce even more the overall failure rate (from 2.07 FIT

to 0.57 FIT) by decreasing the failure rate of false positives as well, from 0.96 FIT to 0.50 FIT, without jeopardizing the false negatives reduction obtained with the previous design. This result represents an improvement of 72.3% in overall failure rate and a reduction of about 52% regarding the rate of false positives. Note that the experimental α parameter remained almost the same, while the β parameter decreased from 2.07 to 0.52. Therefore, the fault-tolerance solution herein proposed is able to keep the system operational during repair with costs much lower than those of traditional TMR. Moreover, reducing the rate of false positives is especially important for applications where such failures may be critical, as explained in Section 3.2.3.

It is important to observe that, although representative in terms of regex features, the set of REMEs implemented for experimental evaluation is very small (only 64) compared to real-life use cases. Therefore, the design sensitive area considered is very small, representing only a small fraction of the total area available in the FPGA at hand. In this way, the failure rates observed in Figure 6.3 were expected to be low and are provided mostly to allow the comparison between implementation variations and to show a possible approach to reach failure rates for other, probably larger, use cases. We must consider that the architecture proposed in (YANG; PRASANNA, 2012) aims to allow the implementation of hundreds of REMEs in a single FPGA. For example, with the ever increasing number and complexity of patterns to scan, dozens of FPGAs are needed to implement a single instance of the Snort rule set, which is composed of thousands of signatures (SNORT, 2017). In other words, millions of REMEs are expected to run in parallel around the world. In this context, the failure rate reduction obtained with the techniques proposed is much more significant. Moreover, the FPGA used as case study uses a relatively old manufacture technology. As transistors shrink and FPGAs integration capabilities increase, more susceptible circuits may be used, and the techniques herein presented also grow in relevance.

6.4 Repair Time Evaluation

In this section, we evaluate the error detection coverage attained with the design proposed and estimate the system repair time. Analyzing the fault injection results, there are 698 sensitive bits triggering undetected errors, which represents only 0.51% of the total errors observed in the unhardened version. The undetected errors happen mainly due to common mode failures or faults affecting the interconnections between the DMR domains and the system primary output. Also, errors in voter circuits that are not logically masked cannot be detected

as well. However, as explained in Section 4, those errors are very unlikely to happen and can be corrected with periodic global reconfiguration (not implemented for this work).

The majority of configuration frames are related to the REMEs, as those elements represent more than 90% of the design area. Therefore, it is expected that most of the errors induced by configuration faults will be detected by the DMR scheme. Regarding the error signatures retrieved by the system, 96.5% of the detected errors can be corrected with partial reconfiguration, i.e., by scrubbing the related area, while the remaining errors will need global reconfiguration to be corrected.

Once we have the total amount of configuration frames used by the design, F_T, the amount of configuration frames used by each reconfigurable region (where each DMR domain is placed) defined in the placement constraints, F_{DMR_k}, the number of reconfigurable regions, *N*, the time required to scrub a single configuration frame, t_F, and the probability of an error be corrected with partial reconfiguration, *P*, the system MTTR can be estimated as

$$MTTR = t_F \cdot \left[\left(\frac{1}{F_T} \cdot \sum_{k=1}^{N} {F_{DMR_k}}^2 \right) + ((1 - P). F_T) \right] \quad (6.1)$$

Eq. (6.1) is divided in two main terms (sum in the square brackets): the first one represents the repair time when the detected error can be corrected with DPR. In such case, the mean number of frames to scrub is obtained through a weighted mean, since both the probability of being subject to faults and the repair time of each DMR domain is proportional to its size, in configuration frames, meaning that evenly dimensioned DMR domains are a better choice, whenever possible. On the other hand, the second term represents the repair time when the error cannot be corrected with DPR. Therefore, the whole design should be reconfigured. Observe that the repair time associated with DPR is always paid, since before identifying the need for a full module scrub, a local scrub is first attempted, while the full scrub only takes place when local scrub fails (probability 1-*P*).

Note that every time a scrubbing procedure is triggered, it is necessary to issue a write command and an extra frame (dummy frame) to the SelectMAP interface. Since this overhead is negligible, it is omitted here for the sake of clarity, although it is taken into account in the reported MTTR figures. Given that each configuration frame in the Virtex-5 FPGA has 1312 bits, and the configuration bandwidth of the SelectMAP and ICAP interfaces is 3.2Gbps (32-bit wide port operating at 100MHz), the design evaluated in this work attained a MTTR of about

172µs, which represents a reduction of about 90% with regard to standard global reconfiguration (1.89ms). This result shows that the placement strategy herein proposed can be very beneficial to improve the system availability with manageable area and delay overheads.

7 CONCLUSION

In this work we have presented a fault-tolerance solution for REM on FPGAs, which can be used in NFV environments to improve the performance of compute intensive network services, such as DPI. The main goal was to improve the system reliability and availability by introducing a fault-tolerance scheme that is able not just to maintain the system safely operational, but also to detect and quickly correct errors with acceptable area and delay costs. To do so, we evaluated the failure modes of a state-of-the-art REM architecture for FPGAs by means of exhaustive fault injection campaigns on the configuration memory. The results show that a critical failure mode for NIDS (false negatives) is more frequent, making the network susceptible to security threats. Thus, we proposed a hybrid fault-tolerance solution (DMR and TMR) with a counter-based voter circuit that can mask the majority of failures by choosing the most likely correct output when an error is detected. This solution is adequate for systems in which the failure modes have different criticalities and probabilities of occurrence, especially when one failure mode is much more critical than the other and also is the most likely to occur. In this way, the techniques proposed are able to reduce the rate of critical failures while reducing the overall failure rate as well with manageable costs.

The DMR scheme with counter-based voters considers the matching latency of the implemented REMEs to avoid forward false positive failures that can be easily detected, since that correct matches are not possible during this timeframe. Outside of the matching latency window, the voter forwards the most likely correct output when the duplicated REMEs diverge, i.e., a positive match. Therefore, an expressive decrease in the system failure rate was observed, especially regarding false negatives. Moreover, a placement strategy that considers the particularities of the design at hand allows the system to detect and localize most of the errors. Therefore, partial reconfiguration can be performed to expressively reduce the system repair time and improve its availability. As a result, we obtained a cost-effective resilient REM system for FPGAs.

Future works include the evaluation of the applicability of the fault-tolerance and scrubbing techniques herein proposed for other FPGA-based applications and network services. For that purpose, the reliability model herein defined (parameters α, β, and γ) can be used for an early estimation of the efficiency of the fault-tolerance mechanisms proposed in this work or for implementing other techniques. Integrating the error detection mechanisms with network-level fault tolerance mechanisms is also a promising possibility to further minimize the impact on performance metrics. Moreover, other systems may present properties similar to the ones

that made the proposed technique beneficial for NIDS-applied REM: a certain type of failure that is much more likely and critical than other types of failures. Systems that trigger security or safety-critical alarms and warnings in general are likely to present these properties. Thus, applying the techniques herein proposed for other systems with these properties is also a relevant future work, as fine tuning for each scenario may provide cost-effective solutions for many critical applications.

The fault-tolerance and scrubbing techniques proposed in this work resulted in two papers. The first one is already published (LEIPNITZ; DE SOUZA; NAZAR, 2016), while the second one is under review for publication on the ACM Transactions on Reconfigurable Technology and Systems (TRETS) journal.

REFERENCES

ANTONELLO, R. et al. Deep packet inspection tools and techniques in commodity platforms: Challenges and trends. **Journal of Network and Computer Applications**, v. 35, n. 6, p. 1863–1878, nov. 2012.

BAUMANN, R. C. Radiation-induced soft errors in advanced semiconductor technologies. **IEEE Transactions on Device and Materials Reliability**, v. 5, n. 3, p. 305–315, sep. 2005.

BECCHI, M.; CROWLEY, P. Efficient regular expression evaluation: theory to practice. In: PROCEEDINGS OF THE 4TH ACM/IEEE SYMPOSIUM ON ARCHITECTURES FOR NETWORKING AND COMMUNICATIONS SYSTEMS. **Proceedings...** New York, NY, USA: ACM Press, 2008. p. 50-59.

BHUVA, B. L. et al. Multi-Cell Soft Errors at Advanced Technology Nodes. **IEEE Transactions on Nuclear Science**, v. 62, n. 6, p. 2585-2591, dec. 2015.

BOLCHINI, C.; MIELE, A.; SANDIONIGI, C. A Novel Design Methodology for Implementing Reliability-Aware Systems on SRAM-Based FPGAs. **IEEE Transactions on Computers**, v. 60, n. 12, p. 1744–1758, dec. 2011.

BRO. **The Bro Network Security Monitor**. 2017. Available from: https://www.bro.org. Accessed: 26 apr. 2017.

BRONSTEIN, Z. et al. Uniform handling and abstraction of NFV hardware accelerators. **IEEE Network**, v. 29, n. 3, p. 22–29, may 2015.

CARMICHAEL, C. et. al. **Correcting Single-Event Upsets Through Virtex Partial Configuration (XAPP216)**. 2000. Available from: https://www.xilinx.com/support/documentation/application_notes/xapp216.pdf. Accessed: 12 mar. 2017.

CHATRAS, B.; OZOG, F. F. Network functions virtualization: The portability challenge. **IEEE Network**, v. 30, n. 4, p. 4–8, jul. 2016.

CHATTERJEE, I. et al. Impact of technology scaling on SRAM soft error rates. **IEEE Transactions on Nuclear Science**, v. 61, n. 6, p. 3512–3518, dec. 2014.

DIXIT, A.; WOOD, A. The impact of new technology on soft error rates. In: 2011 INTERNATIONAL RELIABILITY PHYSICS SYMPOSIUM. **Proceedings...** [S.l.]: IEEE, 2011. p. 5B.4.1-5B.4.7.

DODD, P. E. et al. Current and future challenges in radiation effects on CMOS electronics. **IEEE Transactions on Nuclear Science**, v. 57, n. 4, p. 1747-1763, aug. 2010.

FLOYD, R. W.; ULLMAN, J. D. The Compilation of Regular Expressions into Integrated Circuits. **Journal of the ACM**, v. 29, n. 3, p. 603–622, jul. 1982.

FOUAD, S. et al. Context-aware resources placement for SRAM-based FPGA to minimize

checkpoint/recovery overhead. In: 2014 INTERNATIONAL CONFERENCE ON RECONFIGURABLE COMPUTING AND FPGAS. **Proceedings...** [S.l.]: IEEE, 2014. p. 1-6.
GOKHALE, M.; GRAHAM, P. S. **Reconfigurable computing : accelerating computation with field-programmable gate arrays**. 1st ed. [S.l.]: Springer, 2005.

HAN, B. et al. Network function virtualization: Challenges and opportunities for innovations. **IEEE Communications Magazine**, v. 53, n. 2, p. 90–97, feb. 2015.

HAUCK, S.; DEHON, A. **Reconfigurable computing: the theory and practice of FPGA-based computation**. 1st ed. [S.l]: Elsevier/Morgan Kaufmann, 2008.

HERRERA-ALZU, I.; LÓPEZ-VALLEJO, M. Design techniques for Xilinx Virtex FPGA configuration memory scrubbers. **IEEE Transactions on Nuclear Science**, v. 60, n. 1, p. 376–385, feb. 2013.

HIEU, T. T.; THINH, T. N.; TOMIYAMA, S. ENREM: An efficient NFA-based regular expression matching engine on reconfigurable hardware for NIDS. **Journal of Systems Architecture**, v. 59, n. 4–5, p. 202–212, april-may 2013.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. **Introduction to automata theory, languages, and computation**. 3rd ed. [S.l.]: Pearson/Addison Wesley, 2007.

HUBERT, G.; ARTOLA, L.; REGIS, D. Impact of scaling on the soft error sensitivity of bulk, FDSOI and FinFET technologies due to atmospheric radiation. **The VLSI Journal Integration**, v. 50, p. 39–47, jun. 2015.

JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. **Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Error in Semiconductor Devices (JESD89A).** 2006**.** Available from: https://www.jedec.org/sites/default/files/docs/jesd89a.pdf/. Accessed: 26 apr. 2017.

KASTENSMIDT, F. L.; REIS, R. **Fault-tolerance techniques for SRAM-based FPGAs**. Boston, MA: Springer US, 2006. 184 p. (Frontiers in Electronic Testing, v. 32).

KOREN, I.; KRISHNA, C. M. **Fault-tolerant Systems**. 1st ed. San Francisco, CA, USA: Elsevier/Morgan Kaufmann, 2007.

LEIPNITZ, M. T.; DE SOUZA, E. N.; NAZAR, G. L. Low cost resilient regular expression matching on FPGAs. In: 2016 IEEE INTERNATIONAL SYMPOSIUM ON DEFECT AND FAULT TOLERANCE IN VLSI AND NANOTECHNOLOGY SYSTEMS. **Proceedings...** [S.l.]: IEEE, 2016. p. 75-80.

LEIPNITZ, M. T.; HESS, G. L.; NAZAR, G. L. A fault injection platform for FPGA-based communication systems. In: 2016 IEEE 7TH LATIN AMERICAN SYMPOSIUM ON CIRCUITS AND SYSTEMS. **Proceedings...** [S.l.]: IEEE, 2016. p. 59-62.

LESEA, A. et al. The rosetta experiment: Atmospheric soft error rate testing in differing technology FPGAs. **IEEE Transactions on Device and Materials Reliability**, v. 5, n. 3, p. 317–328, sep. 2005.

MCNAUGHTON, R.; YAMADA, H. Regular Expressions and State Graphs for Automata. **IEEE Transactions on Electronic Computers**, v. EC-9, n. 1, p. 39–47, mar. 1960.

MIJUMBI, R. et al. Network function virtualization: State-of-the-art and research challenges. **IEEE Communications Surveys and Tutorials**, v. 18, n. 1, p. 236–262, firstquarter 2016a.

MIJUMBI, R. et al. Management and orchestration challenges in network functions virtualization. **IEEE Communications Magazine**, v. 54, n. 1, p. 98–105, jan. 2016b.

MOORE, G. E. Cramming more components onto integrated circuits. **Electronics**, v. 38, n. 8, apr. 1965.

MUKHERJEE, S. **Architecture Design for Soft Errors**. 1st ed. San Francisco, CA, USA: Elsevier/Morgan Kaufmann, 2008.

MUNTEANU, D.; AUTRAN, J. L. Modeling and simulation of single-event effects in digital devices and ICs. **IEEE Transactions on Nuclear Science**, v. 55, n. 4, p. 1854–1878, aug. 2008.

NAZAR, G. L. Improving FPGA repair under real-time constraints. **Microelectronics Reliability**, v. 55, n. 7, p. 1109–1119, Jun. 2015.

NAZAR, G. L.; SANTOS, L. P.; CARRO, L. Fine-grained fast field-programmable gate array scrubbing. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 23, n. 5, p. 893–904, may 2015.

NOBACH, L.; HAUSHEER, D. Open, elastic provisioning of hardware acceleration in NFV environments. In: INTERNATIONAL CONFERENCE ON NETWORKED SYSTEMS. **Proceedings...** [S.l.]: IEEE, 2015. p. 1-5.

QUINN, H. et al. Domain crossing errors: Limitations on single device triple-modular redundancy circuits in xilinx FPGAs. **IEEE Transactions on Nuclear Science**, v. 54, n. 6, p. 2037–2043, dec. 2007.

QUINN, H. et al. The Cibola Flight Experiment. **ACM Transactions on Reconfigurable Technology and Systems**, v. 8, n. 1, p. 1–22, mar. 2015.

RAO, P. M. B. et al. Protecting SRAM-based FPGAs Against Multiple Bit Upsets Using Erasure Codes. In: 2014 51ST ACM/EDAC/IEEE DESIGN AUTOMATION CONFERENCE. **Proceedings...** [S.l.]: IEEE, 2014. p. 1-6.

REORDA, M. S.; STERPONE, L.; ULLAH, A. An Error-Detection and Self-Repairing Method for Dynamically and Partially Reconfigurable Systems. **IEEE Transactions on Computers**, v. 66, n. 6, p. 1022–1033, jun. 2017.

RUAN, K. et al. Cloud forensics**.** In: PETERSON G.; SHENOI S. **Advances in Digital Forensics VII**. [S.l.]: Springer-Verlag Berlin Heidelberg, 2011. p. 35-46. (IFIP Advances in Information and Communication Technology, v. 361).

SANTOS, L. P.; NAZAR, G. L.; CARRO, L. Low Cost Dynamic Scrubbing for Real-Time Systems. In: PROCEEDINGS OF THE 12TH INTERNATIONAL SYMPOSIUM ON

APPLIED RECONFIGURABLE COMPUTING. **Proceedings...** [S.l.]: Springer-Verlag New York, 2016. p. 144- 156. (Lecture Notes in Computer Science, v. 9625).

SARI, A.; PSARAKIS, M. Scrubbing-based SEU mitigation approach for systems-on-programmable-chips. In: 2011 INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE TECHNOLOGY. **Proceedings...** [S.l.]: IEEE, 2011. p. 1-8.

SCHMIDT, B.; ZIENER, D.; TEICH, J. Minimizing scrubbing effort through automatic netlist partitioning and floorplanning. In: PROCEEDINGS OF THE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM. **Proceedings...** [S.l.]: IEEE, 2014. p. 299-304.

SIDHU, R.; PRASANNA, V. Fast regular expression matching using FPGAs. In: THE 9TH ANNUAL IEEE SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES. **Proceedings...** [S.l.]: IEEE, 2001. p. 227–238.

SNORT. **Snort - Network Intrusion Detection & Prevention System**. 2017. Available from: https://www.snort.org. Accessed: 26 apr. 2017.

SOURDIS, I. et al. Regular expression matching in reconfigurable hardware. **Journal of Signal Processing Systems**, v. 51, n. 1, p. 99–121, apr. 2008.

TONFAT, J. et al. Analyzing the Effectiveness of a Frame-Level Redundancy Scrubbing Technique for SRAM-based FPGAs. **IEEE Transactions on Nuclear Science**, v. 62, n. 6, p. 3080–3087, dec. 2015.

UNNIKRISHNAN, D. et al. ReClick - A modular dataplane design framework for FPGA-based network virtualization. In: 2011 7TH ACM/IEEE SYMPOSIUM ON ARCHITECTURES FOR NETWORKING AND COMMUNICATIONS SYSTEMS. **Proceedings...** [S.l.]: IEEE, 2011. p. 145-155.

VIOLANTE, M. et al. Simulation-based analysis of SEU effects in SRAM-based FPGAs. **IEEE Transactions on Nuclear Science**, v. 51, n. 6, p. 3354–3359, dec. 2004.

WANG, C. et al. Toward High-Performance and Scalable Network Functions Virtualization. **IEEE Internet Computing**, v. 20, n. 6, p. 10–20, nov. 2016.

WANG, H. et al. MIN-MAX: A counter-based algorithm for regular expression matching. **IEEE Transactions on Parallel and Distributed Systems**, v. 24, n. 1, p. 92–103, jan. 2013.

WIRTHLIN, M. High-reliability FPGA-based systems: Space, high-energy physics, and beyond. **Proceedings of the IEEE**, v. 103, n. 3, p. 379–389, mar. 2015.

XILINX. **Isolation Design Flow**. 2017a. Available from: https://www.xilinx.com/applications/isolation-design-flow.html. Accessed: 25 jul. 2017.

XILINX. **Xilinx University Program XUPV5-LX110T Development System**. 2017b. Available from: https://www.xilinx.com/univ/xupv5-lx110t.htm. Accessed: 26 apr. 2017.

XILINX. **Device Reliability Report, First Half 2016 (UG116)**. 2017c. Available from:

https://www.xilinx.com/support/documentation/user_guides/ug116.pdf. Accessed: 26 apr. 2017.

XU, C. et al. A Survey on Regular Expression Matching for Deep Packet Inspection: Applications, Algorithms, and Hardware Platforms**. IEEE Communications Surveys and Tutorials**, v. 18, n. 4, p. 2991–3029, fourthquarter 2016.

YAMAGAKI, N.; SIDHU, R.; KAMIYA, S. High-speed regular expression matching engine using multi-character NFA. In: 2008 INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS. **Proceedings...** [S.l.]: IEEE, 2008. p. 131-136.

YANG, Y. H.; PRASANNA, V. High-performance and compact architecture for regular expression matching on FPGA. **IEEE Transactions on Computers**, v. 61, n. 7, p. 1013–1025, jul. 2012.

APPENDIX - REGULAR EXPRESSIONS IMPLEMENTED

Snort Rule Set (v2.8)	*ID*	*Regex*
Web-activex	4893	<OBJECT\s+[^>]*classid\s*=\s*[\x22\x27]?\s*clsid\s*\x3a\s*\x7B?\s*3050F4F5-98B5-11CF-BB82-00AA00BDCE0B
Web-activex	7011	new[\r\n\s]*ActiveXObject[\r\n\s]*\([\r\n\s]*(\x22HtmlDlgSafeHelper.HtmlDlgSafeHelper\x22|\x27HtmlDlgSafeHelper.HtmlDlgSafeHelper\x27)[\r\n\s]*\)|(\w+)[\r\n\s]*=[\r\n\s]*(\x22HtmlDlgSafeHelper.HtmlDlgSafeHelper\x22|\x27HtmlDlgSafeHelper.HtmlDlgSafeHelper\x27)[\r\n\s]*\x3b.*new[\r\n\s]*ActiveXObject[\r\n\s]*\([\r\n\s]*\1[\r\n\s]*\)
Web-activex	7941	<\x00O\x00B\x00J\x00E\x00C\x00T\x00(\s\x00)+([^>]\x00)*c\x00l\x00a\x00s\x00s\x00i\x00d\x00(\s\x00)*=\x00(\s\x00)*([\x22\x27]\x00)?(\s\x00)*c\x00l\x00s\x00i\x00d\x00(\s\x00)*\x3a\x00(\s\x00)*(\x7B\x00)?(\s\x00)*6\x002\x003\x00E\x002\x008\x008\x002\x00-\x00F\x00C\x000\x00E\x00-\x001\x001\x00D\x001\x00-\x009\x00A\x007\x007\x00-\x000\x000\x000\x000\x00F\x008\x007\x005\x006\x00A\x001\x000\x00
Web-activex	7957	<\x00O\x00B\x00J\x00E\x00C\x00T\x00(\s\x00)+([^>]\x00)*c\x00l\x00a\x00s\x00s\x00i\x00d\x00(\s\x00)*=\x00(\s\x00)*([\x22\x27]\x00)?(\s\x00)*c\x00l\x00s\x00i\x00d\x00(\s\x00)*\x3a\x00(\s\x00)*(\x7B\x00)?(\s\x00)*8\x00B\x00D\x002\x001\x00D\x002\x000\x00-\x00E\x00C\x004\x002\x00-\x001\x001\x00C\x00E\x00-\x009\x00E\x000\x00D\x00-\x000\x000\x00A\x00A\x000\x000\x006\x000\x000\x002\x00F\x003\x00
Web-activex	8419	new[\r\n\s]*ActiveXObject[\r\n\s]*\([\r\n\s]*(\x22WebViewFolderIcon.WebViewFolderIcon.1\x22|\x27WebViewFolderIcon.WebViewFolderIcon.1\x27)[\r\n\s]*\)|(\w+)[\r\n\s]*=[\r\n\s]*(\x22WebViewFolderIcon.WebViewFolderIcon.1\x22|\x27WebViewFolderIcon.WebViewFolderIcon.1\x27)[\r\n\s]*\x3b.*new[\r\n\s]*ActiveXObject[\r\n\s]*\([\r\n\s]*\1[\r\n\s]*\)
Web-activex	8422	<OBJECT\s*[^>]*\s*classid\s*=\s*(\x22|\x27|)\s*clsid\s*\x3a\s*{?\s*0006F063-0000-0000-C000-000000000046\s*}?\s*\1
Web-activex	8776	new[\r\n\s]*ActiveXObject[\r\n\s]*\([\r\n\s]*(\x22DirectAnimation.DAUserData.1\x22|\x27DirectAnimation.DAUserData.1\x27)[\r\n\s]*\)|(\w+)[\r\n\s]*=[\r\n\s]*(\x22DirectAnimation.DAUserData.1\x22|\x27DirectAnimation.DAUserData.1\x27)[\r\n\s]*\x3b.*new[\r\n\s]*ActiveXObject[\r\n\s]*\([\r\n\s]*\1[\r\n\s]*\)

Web-activex	8824	new[\r\n\s]*ActiveXObject[\r\n\s]*\([\r\n\s]*(\x22DirectAnimation.DAGeometry.1\x22|\x27DirectAnimation.DAGeometry.1\x27)[\r\n\s]*\)|(\w+)[\r\n\s]*=[\r\n\s]*(\x22DirectAnimation.DAGeometry.1\x22|\x27DirectAnimation.DAGeometry.1\x27)[\r\n\s]*\x3b.*new[\r\n\s]*ActiveXObject[\r\n\s]*\([\r\n\s]*\1[\r\n\s]*\)
Web-activex	8851	1([^>]\x00)*1(?P<q2>\x22\x00|\x27\x00|)1({\x00)?1(}\x00)?(?P=q2)(?=\s\x00|>\x00)
Web-activex	9131	new[\r\n\s]*ActiveXObject[\r\n\s]*\([\r\n\s]*(\x22WZFILEVIEW.FileViewCtrl.61\x22|\x27WZFILEVIEW.FileViewCtrl.61\x27)[\r\n\s]*\)|(\w+)[\r\n\s]*=[\r\n\s]*(\x22WZFILEVIEW.FileViewCtrl.61\x22|\x27WZFILEVIEW.FileViewCtrl.61\x27)[\r\n\s]*\x3b.*new[\r\n\s]*ActiveXObject[\r\n\s]*\([\r\n\s]*\1[\r\n\s]*\)
Web-activex	9826	(\w+)\s*=\s*(\x22BOLDOWNLOADER.BolDownloaderCtrl.1\x22|\x27BOLDOWNLOADER.BolDownloaderCtrl.1\x27)\s*\x3b.*\w+\s*=\s*new\s*ActiveXObject\s*\(\s*\1\s*\)|\w+\s*=\s*new\s*ActiveXObject\s*\(\s*(\x22BOLDOWNLOADER.BolDownloaderCtrl.1\x22|\x27BOLDOWNLOADER.BolDownloaderCtrl.1\x27)\s*\)
Web-activex	10190	<\x00o\x00b\x00j\x00e\x00c\x00t\x00(\s\x00)*([^>]\x00)*c\x00l\x00a\x00s\x00s\x00i\x00d\x00(\s\x00)*=\x00(\s\x00)*(\x22\x00|\x27\x00|)c\x00l\x00s\x00i\x00d\x00(\s\x00)*\x3a\x00(\s\x00)*({\x00)?(\s\x00)*6\x007\x00D\x00A\x00B\x00F\x00B\x00F\x00-\x00D\x000\x00A\x00B\x00-\x004\x001\x00f\x00a\x00-\x009\x00C\x004\x006\x00-\x00C\x00C\x000\x00F\x002\x001\x007\x002\x001\x006\x001\x006\x00(}\x00)?\5
Web-activex	10414	(\w+)\s*=\s*(\x22JNILOADER\.JNILoaderCtrl\x22|\x27JNILOADER\.JNILoaderCtrl\x27)\s*\x3b.*(\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*\1\s*\)(\s*\.\s*(LoadLibrary)\s*\(|.*\3\s*\.\s*(LoadLibrary)\s*\()|(\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(\x22JNILOADER\.JNILoaderCtrl\x22|\x27JNILOADER\.JNILoaderCtrl\x27)\s*\)(\s*\.\s*(LoadLibrary)\s*\(|.*\7\s*\.\s*(LoadLibrary)\s*\()

Web-activex	10427	(<object\s*[^>]*\s*id\s*=\s*(?P<m1>\x22|\x27|)(?P<id1>.+?)(?P=m1)(\s|>)[^>]*\s*classid\s*=\s*(?P<q1>\x22|\x27|)\s*clsid\s*\x3a\s*{?\s*BA61606B-258C-4021-AD27-E07A3F3B91DB\s*}?\s*(?P=q1)(\s|>).*(?P=id1)\s*\.\s*(DeleteFile|StartBatchUploading|StartStrBatchUploading|StartUploading)|<object\s*[^>]*\s*classid\s*=\s*(?P<q2>\x22|\x27|)\s*clsid\s*\x3a\s*{?\s*BA61606B-258C-4021-AD27-E07A3F3B91DB\s*}?\s*(?P=q2)(\s|>)[^>]*\s*id\s*=\s*(?P<m2>\x22|\x27|)(?P<id2>.+?)(?P=m2)(\s|>).*(?P=id2)\.(DeleteFile|StartBatchUploading|StartStrBatchUploading|StartUploading))\s*\(
Web-activex	11201	(?P<c>\w+)\s*=\s*(\x22OA\.OActrl(\.\d)?\x22|\x27OA\.OActrl(\.\d)?\x27)\s*\x3b.*(?P<v>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(?P=c)\s*\)(\s*\.\s*(DoOleCommand|FTPDownloadFile|FTPUploadFile|HttpUploadFile|Save|SaveWebFile|OpenWebFile)\s*|.*(?P=v)\s*\.\s*(DoOleCommand|FTPDownloadFile|FTPUploadFile|HttpUploadFile|Save|SaveWebFile|OpenWebFile)\s*)|(?P<n>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(\x22OA\.OActrl(\.\d)?\x22|\x27OA\.OActrl(\.\d)?\x27)\s*\)(\s*\.\s*(DoOleCommand|FTPDownloadFile|FTPUploadFile|HttpUploadFile|Save|SaveWebFile|OpenWebFile)\s*|.*(?P=n)\s*\.\s*(DoOleCommand|FTPDownloadFile|FTPUploadFile|HttpUploadFile|Save|SaveWebFile|OpenWebFile)\s*)\s*\(
Web-activex	11651	1([^>]\x00)*1(?P<q38>\x22\x00|\x27\x00|)1({\x00)?1(}\x00)?(?P=q38)(?=\s\x00|>\x00)
Web-activex	11654	(<object\s*[^>]*\s*id\s*=\s*(?P<m17>\x22|\x27|)(?P<id1>.+?)(?P=m17)(\s|>)[^>]*\s*classid\s*=\s*(?P<q41>\x22|\x27|)\s*clsid\s*\x3a\s*{?\s*00140200-B1BA-11CE-ABC6-F5B2E79D9E3F\s*}?\s*(?P=q41)(\s|>).*(?P=id1)\s*\.\s*(BrowseDir)|<object\s*[^>]*\s*classid\s*=\s*(?P<q42>\x22|\x27|)\s*clsid\s*\x3a\s*{?\s*00140200-B1BA-11CE-ABC6-F5B2E79D9E3F\s*}?\s*(?P=q42)(\s|>)[^>]*\s*id\s*=\s*(?P<m18>\x22|\x27|)(?P<id2>.+?)(?P=m18)(\s|>).*(?P=id2)\.(BrowseDir))\s*\(
Web-activex	12021	(?P<c>\w+)\s*=\s*(\x22NCTWMAFile2\.WMAFile2\x22|\x27NCTWMAFile2\.WMAFile2\x27)\s*\x3b.*(?P<v>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(?P=c)\s*\)(\s*\.\s*CreateFile\s*|.*(?P=v)\s*\.\s*CreateFile\s*)|(?P<n>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(\x22NCTWMAFile2\.WMAFile2\x22|\x27NCTWMAFile2\.WMAFile2\x27)\s*\)(\s*\.\s*CreateFile\s*|.*(?P=n)\s*\.\s*CreateFile\s*)\s*\(

Web-activex	12663	(?P<c>\w+)(\s\x00)*=(\s\x00)*(?P<q4>\x22|\x27|)I\x00E\x00R\x00P\x00C\x00t\x00l\x00.\x00I\x00E\x00R\x00P\x00C\x00t\x00l\x00(?P=q4)(\s|>)(\s\x00)*\x3b\x00.*(?P<v>(\w\x00)+)(\s\x00)*1\(\x00(\s\x00)*(?P=c)(\s\x00)*\)\x00|(?P<n>\w+)(\s\x00)*1\(\x00(\s\x00)*(?P<q5>\x22|\x27|)I\x00E\x00R\x00P\x00C\x00t\x00l\x00.\x00I\x00E\x00R\x00P\x00C\x00t\x00l\x00(?P=q5)(\s|>)(\s\x00)*\)\x00
Web-activex	12961	<object\s*[^>]*\s*classid\s*=\s*(?P<q1>\x22|\x27|)\s*clsid\s*\x3a\s*{?\s*AD5FBDB8-C518-47F7-B4F1-F1F58D21A716\s*}?\s*(?P=q1)(\s|>)
Web-activex	13227	(?P<c>\w+)(\s\x00)*=(\s\x00)*(?P<q4>\x22|\x27|)Y\x00S\x00h\x00o\x00r\x00t\x00c\x00u\x00t\x00_\x00D\x00L\x00L\x00.\x00S\x00h\x00o\x00r\x00t\x00c\x00u\x00t\x00(?P=q4)(\s|>)(\s\x00)*\x3b\x00.*(?P<v>(\w\x00)+)(\s\x00)*1\(\x00(\s\x00)*(?P=c)(\s\x00)*\)\x00|(?P<n>\w+)(\s\x00)*1\(\x00(\s\x00)*(?P<q5>\x22|\x27|)Y\x00S\x00h\x00o\x00r\x00t\x00c\x00u\x00t\x00_\x00D\x00L\x00L\x00.\x00S\x00h\x00o\x00r\x00t\x00c\x00u\x00t\x00(?P=q5)(\s|>)(\s\x00)*\)\x00
Web-activex	13603	(?P<c>\w+)\s*=\s*(\x22rmocx\.RealPlayer\s*Download\s*Handler(\.\d)?\x22|\x27rmocx\.RealPlayer\s*Download\s*Handler(\.\d)?\x27)\s*\x3b.*(?P<v>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(?P=c)\s*\)(\s*\.\s*(Console|Controls)\s*|.*(?P=v)\s*\.\s*(Console|Controls)\s*)|(?P<n>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(\x22rmocx\.RealPlayer\s*Download\s*Handler(\.\d)?\x22|\x27rmocx\.RealPlayer\s*Download\s*Handler(\.\d)?\x27)\s*\)(\s*\.\s*(Console|Controls)\s*|.*(?P=n)\s*\.\s*(Console|Controls))\s*=
Web-activex	13787	(?P<c>\w+)\s*=\s*(\x22HanGamePluginCn18\.HanGamePluginCn18\x22|\x27HanGamePluginCn18\.HanGamePluginCn18\x27)\s*\x3b.*(?P<v>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(?P=c)\s*\)(\s*\.\s*(hgs_startGame|hgs_startNotify)\s*|.*(?P=v)\s*\.\s*(hgs_startGame|hgs_startNotify)\s*)|(?P<n>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(\x22HanGamePluginCn18\.HanGamePluginCn18\x22|\x27HanGamePluginCn18\.HanGamePluginCn18\x27)\s*\)(\s*\.\s*(hgs_startGame|hgs_startNotify)\s*|.*(?P=n)\s*\.\s*(hgs_startGame|hgs_startNotify)\s*)\s*\(
Web-activex	13885	(?P<c>\w+)\s*=\s*(\x22UUUPGRADE\.UUUpgradeCtrl\.1\x22|\x27UUUPGRADE\.UUUpgradeCtrl\.1\x27)\s*\x3b.*(?P<v>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(?P=c)\s*\)(\s*\.\s*Update\s*|.*(?P=v)\s*\.\s*Update\s*)|(?P<n>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(\x22UUUPGRADE\.UUUpgradeCtrl\.1\x22|\x27UUUPGRADE\.UUUpgradeCtrl\.1\x27)\s*\)(\s*\.\s*Update\s*|.*(?P=n)\s*\.\s*Update\s*)\s*\(

Web-activex	14013	(<object\s*[^>]*\s*id\s*=\s*(?P<m1>\x22|\x27)(?P<id1>.+?)(?P=m1)(\s|>)[^>]*\s*classid\s*=\s*(?P<q1>\x22|\x27)\s*clsid\s*\x3a\s*{?\s*32E26FD9-F435-4A20-A561-35D4B987CFDC\s*}?\s*(?P=q1)(\s|>).*(?P=id1)\s*\.\s*(NewObject)|<object\s*[^>]*\s*classid\s*=\s*(?P<q2>\x22|\x27)\s*clsid\s*\x3a\s*{?\s*32E26FD9-F435-4A20-A561-35D4B987CFDC\s*}?\s*(?P=q2)(\s|>)[^>]*\s*id\s*=\s*(?P<m2>\x22|\x27)(?P<id2>.+?)(?P=m2)(\s|>).*(?P=id2)\.(NewObject))\s*\(
Web-activex	14308	(?P<c>\w+)\s*=\s*(\x22Vmc2vmx\.CoVPCConfiguration\x22|\x27Vmc2vmx\.CoVPCConfiguration\x27)\s*\x3b.*(?P<v>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(?P=c)\s*\)|(?P<n>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(\x22Vmc2vmx\.CoVPCConfiguration\x22|\x27Vmc2vmx\.CoVPCConfiguration\x27)\s*\)
Web-activex	14422	(?P<c>\w+)\s*=\s*(\x22vmdbCOM\.VmdbDatabase\x22|\x27vmdbCOM\.VmdbDatabase\x27)\s*\x3b.*(?P<v>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(?P=c)\s*\)|(?P<n>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(\x22vmdbCOM\.VmdbDatabase\x22|\x27vmdbCOM\.VmdbDatabase\x27)\s*\)
Web-activex	14440	<object\s*[^>]*\s*classid\s*=\s*(?P<q1>\x22|\x27)\s*clsid\s*\x3a\s*{?\s*96a05576-987f-4f6d-9102-8799e3ded07b\s*}?\s*(?P=q1)(\s|>)
Web-activex	14524	<object\s*[^>]*\s*classid\s*=\s*(?P<q1>\x22|\x27)\s*clsid\s*\x3a\s*{?\s*dfd8b167-5652-4962-a162-9a227825afaa\s*}?\s*(?P=q1)(\s|>)
Web-activex	14528	(?P<c>\w+)\s*=\s*(\x22vmappcfg\.HotfixWz\x22|\x27vmappcfg\.HotfixWz\x27)\s*\x3b.*(?P<v>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(?P=c)\s*\)|(?P<n>\w+)\s*=\s*new\s*ActiveXObject\s*\(\s*(\x22vmappcfg\.HotfixWz\x22|\x27vmappcfg\.HotfixWz\x27)\s*\)
Web-activex	14898	(?P<c>\w+)(\s\x00)*=(\s\x00)*(?P<q4>\x22|\x27)H\x00P\x00R\x00u\x00l\x00e\x00s\x00E\x00n\x00g\x00i\x00n\x00e\x00.\x00C\x00o\x00n\x00t\x00e\x00n\x00t\x00C\x00o\x00l\x00l\x00e\x00c\x00t\x00i\x00o\x00n\x00(?P=q4)(\s|>)(\s\x00)*\x3b\x00.*(?P<v>(\w\x00)+)(\s\x00)*=\x00(\s\x00)*n\x00e\x00w\x00(\s\x00)*A\x00c\x00t\x00i\x00v\x00e\x00X\x00O\x00b\x00j\x00e\x00c\x00t\x00(\s\x00)*\(\x00(\s\x00)*(?P=c)(\s\x00)*\)\x00|(?P<n>\w+)(\s\x00)*=\x00(\s\x00)*n\x00e\x00w\x00(\s\x00)*A\x00c\x00t\x00i\x00v\x00e\x00X\x00O\x00b\x00j\x00e\x00c\x00t\x00(\s\x00)*\(\x00(\s\x00)*(?P<q5>\x22|\x27)H\x00P\x00R\x00u\x00l\x00e\x00s\x00E\x00n\x00g\x00i\x00n\x00e\x00.\x00C\x00o\x00n\x00t\x00e\x00n\x00t\x00C\x00o\x00l\x00l\x00e\x00c\x00t\x00i\x00o\x00n\x00(?P=q5)(\s|>)(\s\x00)*\)\x00

Web-activex	15273	(?P<c>\w+)(\s\x00)*=(\s\x00)*(?P<q9>\x22|\x27|)M\x00W\x006\x00P\x00D\x00F\x004\x001\x007\x00.\x00P\x00D\x00F\x004\x001\x007\x00(\.\x00\d\x00)?(?P=q9)(\s|>)(\s\x00)*\x3b\x00.*(?P<v>(\w\x00)+)(\s\x00)*=\x00(\s\x00)*n\x00e\x00w\x00(\s\x00)*A\x00c\x00t\x00i\x00v\x00e\x00X\x00O\x00b\x00j\x00e\x00c\x00t\x00(\s\x00)*\(\x00(\s\x00)*(?P=c)(\s\x00)*\)\x00|(?P<n>\w+)(\s\x00)*=\x00(\s\x00)*n\x00e\x00w\x00(\s\x00)*A\x00c\x00t\x00i\x00v\x00e\x00X\x00O\x00b\x00j\x00e\x00c\x00t\x00(\s\x00)*\(\x00(\s\x00)*(?P<q10>\x22|\x27|)M\x00W\x006\x00P\x00D\x00F\x004\x001\x007\x00.\x00P\x00D\x00F\x004\x001\x007\x00(\.\x00\d\x00)?(?P=q10)(\s|>)(\s\x00)*\)\x00
Exploit	1382	^PRIVMSG\s+nickserv\s+IDENTIFY\s[^\n]{100}
Smtp	3824	^AUTH\s+\S+\s+[^\n]{128}
Exploit	2320	^USER\s[^\n]{49}
Smtp	2260	^VRFY[^\n]{255,}
Smtp	2183	^\s*Content-Transfer-Encoding\s*\x3A\s*[^\n]{100}
Exploit	2584	^PRIVMSG\s+[^\s]+\s+\x3a\s*\x01SENDLINK\x7c[^\x7c]{69}
Exploit	3085	goaway\?message=[^\s]{500}
Exploit	3517	PUTOLF\s+((\S+\s+){4}[^\s]{256}|(\S+\s+){6}[^\x3c]{512})
Exploit	3538	^\x01.{23}(\x25|\x26)
Exploit	3540	^\x01.{23}(\x25|\x26).{15}(\x0A|\x34)
Exploit	3651	^Entry \/file\/[0-9.]{71,}\/\/.*\x0aannotate\x0a
Exploit	3677	^CSeq\x3A\s*[^\nA-Za-z]*[A-Za-z][^\n]{16,}
Exploit	4127	\x2fnds\x2f[^&\r\n\x3b]{500}
Exploit	4638	^.{10}[\x14\x15]\x01.{1}[\x00-\x03]
Spyware-put	5756	\x2Fezsb\d{4}\x2Fbar_pl\x2Ffav\.fcgi
Spyware-put	5864	tid\x3D\x7B([0-9A-z]+\x2D){4}[0-9A-z]+\x7D
Backdoor	6022	\x2B\x2D{3}\x7C[^\r\n]*\x7C\x2D{3}\x2B
Backdoor	6026	\s{23}DIMBUS\s+Server\s+v\d+\x2E\d+

Backdoor	6114	^\x21{3}Optix\s+Pro\s+v\d+\x2E\d+\s+Server\s+Online\x21{3}
Spyware-put	6258	^User-Agent\x3A[^\r\n]*Agent[0-9]{7}
Backdoor	7643	^answer\x00{6}NetControl\x2EServer\s+\d+\x2E\d+\s+\x22The\s+UNSEEN\x22\s+Project
Backdoor	7732	^SINFO\x3B[^\r\n]{1,20}\x3BPONG\x3B
Spyware-put	8545	^\x23\d+\x7c([0-9A-E]{2}\x2d){5}[0-9A-E]{2}\x7croogoo\x7c
Spyware-put	9827	^Subject\x3a[^\r\n]*\d{5,6}\x5f[123]?\d\x2El
Spyware-put	5816	^(TC|FT)P\s+Redirections?\s+destroyed\x21
Spyware-put	12672	\x2Ftoolbar\x2F(img\x2F){0,1}\x2Ephp\?
Smtp	3682	Content-Type\x3A\s+audio\/(x-wav|mpeg|x-midi).*filename=[\x22\x27]?.{1,221}\.(vbs|exe|scr|pif|bat)
Backdoor	15165	^\x2F40e800[0-9A-F]{30,}$
Oracle	2698	CREATE\s.*?FILE\s+((AS|MEMBER|TO)\s+)?(\x27[^\x27]{512}|\x22[^\x22]{512})
Oracle	15255	^.{12}\x00{6}\x09\x01.{16}.{300}
Oracle	15261	ora_osb_bgcookie\x3d[^\x20\x26\x3b]{1}
Oracle	11001	((\w+)[\r\n\s]*\x3a=[\r\n\s]*(\x27[^\x27]{32,}\x27|\x22[^\x22]{32,}\x22)[\r\n\s]*\x3b.*snap_owner[\r\n\s]*=>[\r\n\s]*\2|snap_owner\s*=>\s*(\x27[^\x27]{32,}|\x22[^\x22]{32,})|(\s*(\x27[^\x27]{32,}|\x22[^\x22]{32,}))