BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

PHD THESIS

Brno, 2017 Ing. Lukáš Mičulka

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

METHODOLOGY FOR FAULT TOLERANT SYSTEMS DESIGN INTO LIMITED IMPLEMENTATION AREA IN FPGA

METODIKA NÁVRHU SYSTÉMŮ ODOLNÝCH PROTI PORUCHÁM DO OMEZENÉHO IMPLEMENTAČNÍHO PROSTORU NA BÁZI FPGA

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR AUTOR PRÁCE	Ing. LUKÁŠ MIČULKA
SUPERVISOR ŠKOLITEL	Doc. Ing. ZDENĚK KOTÁSEK, CSc.

BRNO 2017

Abstract

The work presents a methodology of fault tolerant system design into an FPGA with the ability of the transient fault and the permanent fault mitigation. The transient fault mitigation is done by the partial dynamic reconfiguration. The mitigation of a certain number of permanent faults is based on using a specific fault tolerant architecture occupying less resources than the previosly used one and excluding the faulty part of the FPGA from further use. This inovative technique is based on the precompiled configurations stored in an external memory. To reduce the required space for a partial bitstream the relocation technique is used.

Abstrakt

Tato práce popisuje navrženou metodologii pro návrh systémů odolných proti poruchám v FPGA schopnou ochránit systém před projevy přechodných a trvalých poruch. Oprava přechodné poruchy je prováděna částečnou dynamickou rekonfigurací. Oprava omezeného počtu trvalých poruch je založena na použití odolných architektur využívajících menší množství zdrojů než předchozí použitá architektura. Vadná část FPGA tak není dále využívána. Tato technika je založena na použití předkompilovaných konfigurací uložených v externí paměti. Pro snížení paměťových nároků pro uložení konfiguračních bitových posloupností je použita technika relokace.

Keywords

fault tolerant system design, partial reconfiguration, design methodology, FPGA.

Klíčová slova

návrh systémů odolných proti poruchám, částečná rekonfigurace, metodika návrhu, spolehlivost, FPGA.

Reference

MIČULKA, Lukáš. *Methodology for Fault Tolerant Systems Design into Limited Implementation Area in FPGA*. Brno, 2017. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Doc. Ing. Zdeněk Kotásek, CSc.

Methodology for Fault Tolerant Systems Design into Limited Implementation Area in FPGA

Declaration

Hereby I declare that this PhD thesis was prepared as an original author's work under the supervision of associate professor Zdeněk Kotásek. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

....
Lukáš Mičulka
August 29, 2017

Acknowledgements

I would like to thank my supervisor for his professional guidance, his help and expert advices.

Contents

1 Introduction **4**
1.1 Preface . . . 4
1.2 Structure of thesis . . . 6

2 Common knowledge **7**
2.1 Digital systems . . . 7
2.1.1 Fixed logic devices . . . 8
2.1.2 Programmable logic devices . . . 9
2.2 FPGA . . . 10
2.2.1 FPGA structure . . . 10
2.2.2 Storing bitstream configuration . . . 11
2.2.3 Xilinx Virtex family . . . 12
2.2.4 FPGA reconfiguration . . . 13
2.2.5 FPGA design synthesis and implementation . . . 15
2.2.6 Design workflow with partial dynamic reconfiguration . . . 17
2.2.7 Configuration bitstream structure . . . 18
2.3 Faults in digital systems . . . 19
2.3.1 Terms . . . 19
2.3.2 Fault classification . . . 20
2.3.3 Faults in FPGA . . . 21
2.4 Dependability of systems . . . 22
2.4.1 Reliability . . . 24
2.4.2 Availability . . . 26
2.4.3 Safety . . . 26
2.4.4 Maintainability . . . 27
2.4.5 System dependability analysis . . . 27
2.4.6 Dependable system design . . . 29

3 Related areas of topic **31**
3.1 Fault tolerant systems . . . 31
3.1.1 Static fault tolerance . . . 32
3.1.2 Dynamic fault tolerance . . . 33
3.2 Fault detection and localization techniques . . . 33
3.2.1 Replication and concurrent error detection . . . 35
3.2.2 Off-line testing . . . 36
3.2.3 Bitstream readback . . . 37
3.2.4 Roving STAR . . . 37
3.2.5 Summary . . . 39

3.3 Transient fault mitigation 40
3.3.1 Configuration bitstream scrubbing 40
3.3.2 Recovery by partial dynamic reconfiguration 41
3.4 Techniques for system recovery after permanent fault occurence 42
3.4.1 Incremental mapping and design routing 43
3.4.2 Approach based on precompiled alternate configurations 44
3.4.3 Evolutionary algorithms 46
3.4.4 Summary 46
3.5 Fault injection techniques 47

4 Motivation and goals of the research **49**
4.1 Motivation 49
4.2 Goals of the research 50

5 Methodology for fault tolerant system design into limited implementation area in FPGA **53**
5.1 Methodology basic principles 53
5.1.1 System design based on the methodology 54
5.1.2 Structure of fault tolerant system under design - basic principles . . 55
5.1.3 Generations of alternative FT architecture configurations 56
5.1.4 Reducing the number of configuration bitstreams via bitstream relocation technique 57
5.1.5 Synchronization issues 58
5.2 Generic partial dynamic reconfiguration controller 59
5.2.1 The design goals of GPDRC 60
5.2.2 GPDRC unit design 60
5.2.3 Implementing GPDRC unit as fault tolerant 62
5.3 Fault mitigation procedure 62
5.4 Summary 65

6 Design of fault tolerant architecture by means of developed methodology principles **66**
6.1 Prerequisities 66
6.1.1 PRM isolation 66
6.1.2 Bitstream relocation technique 67
6.2 Fault tolerant architectures design 71
6.2.1 System design partitioning 71
6.2.2 Selection of the degradation strategy for recovering from permanent faults 72
6.2.3 The allocation of implementation area for FT architectures 73
6.2.4 The instantiation of GPDRC 74
6.3 Design tool for automatic generation of fault tolerant architectures 74
6.3.1 Inputs 75
6.3.2 Outputs 75
6.3.3 The process of FT architecture generation 75
6.4 The implementation of generated FT architectures 78
6.5 Summary 79

7 Implementation and experimental results **81**

7.1 Implementation of GPDRC 81
7.1.1 Generic implementation of GPDRC and its scaling 81
7.1.2 The reconfiguration time of PRM 82
7.2 Evaluation of hardware overhead of FT architectures developed to secure a given part of system 83
7.2.1 FT Architecture of Generation 0 84
7.2.2 FT Architecture of Generation 1 84
7.2.3 Non-FT Architecture of Generation 2 85
7.2.4 Evaluation of resource overhead 86
7.2.5 Modelling reliability of proposed FT architectures 86
7.3 Implementation results of different approaches to the partitioning of original system 91
7.4 SEU testing platform for the evaluation of FT system design by means of methodology principles 93
7.4.1 The implemented test and evaluation equipment 93
7.4.2 Process of SEU faults injection 95
7.4.3 Experimental results of GPDRC transient fault mitigation process . 96
7.4.4 Testing and evaluating recovery from permanent fault occurence . . 97
7.5 Summary 98

8 Conclusions **99**
8.1 Benefits of this research 99
8.2 Possible enhancements of methodology 100
8.2.1 Bitstream compression 100
8.2.2 Adoption of isolation design flow 101
8.2.3 Other possible enhancements 102

A Author's publications **117**

B Publications cited by other authors **119**

Chapter 1

Introduction

This chapter brings a brief introduction to the topic of this thesis. It is focused on fault tolerant aspects of electronic devices and it is describing the methodology for fault tolerant system design developed specially for Field Programmable Gate Arrays (FPGA). Before the methodology is proposed in the following chapters, contemporary state of the knowledge in this topic and known approaches for solving issues connected with it are discussed.

1.1 Preface

The progress in manufacturing electronic devices mainly stands on shrinking its parts such as chips and transistors. The Moore's law says that every two years the power of new computer chips will be doubled. Although this idea was formed in 1965 it withstanded as the truth for 50 years. In these days when the shrinking of integrated circuits achieves the 14-nm resolution (half-node shrink) [42], it is clear that the law will be void very early.

The scaling of transistors to such small sizes provides high performance, low power and also lower costs per unit but has also very strong drawbacks. From the system dependability viewpoint, the rapid downsizing of circuitry brings increased defect rates because the wires and devices made of few atoms and bonds are more susceptible to the occurrence of defective parts. These small devices are also very fragile on overstress and other environmental influences during operational lifetime. Additionally, small changes inside fabric caused by these factors can lead to large impact on device performance. It also brings bigger susceptibility to transient upsets. Small nodes use less charge to hold state or data and can be easily altered and upset by noise from outside environment such as radiation.

This implies the motivation to make electronic devices dependable even in very harsh environmental conditions. This is very recent topic. Besides others, it is deeply connected with space exploration. In 2016, the spacecraft from NASA New Frontiers mission came to the orbit of one of Jupiter's months, Juno. This mission is specific also because of its high demands to electronic devices of spacecraft. The radiation in radiation belts of this planet is much stronger then on the Earth's orbit [8]. The electronic devices in spacecraft has to be hardened against such a dose of radiation but it has to be also made fault tolerant to be able to survive the time of mission even if some fault occurs. As an example of its increased dependability, Juno spacecraft's data handling system is based on RAD750 processor from BAE Systems company which is designed to accommodate as much single event effects as possible to survive them for at least 15 years without intervention from Earth [51]. The loose of money in case of unsuccessful mission will be very big and repeating of mission will be time-demanding because the flight to Juno's orbit takes almost 5 years.

It is clear now, that one of the key system indicators is its dependability. It is an integrated measure consisting of several attributes such as the availability and relibility of system, its maintainability and durability and also its safety and security. All together it expresses the ability of system to produce outputs that can justifiably be trusted. To increase the dependability of system, several mechanism can be adopted. One of the most popular approaches is Fault Tolerant (FT) system design which enables a system to continue its intended operation when some part of the system fails. The operation after the fault occurrence can be at a reduced level, but it should not fail completely. Many FT techniques use hardware redundancy in order to reduce the probability of failure. By replicating the desired circuitry and comparing the results, fault in one or more replicated system units can be detected and reported or the fault mitigation process of faulty unit can be triggered.

Nowadays, developers implementing a particular system can select from various electronic devices. There are many fabrics starting from simple integrated circuits, universal purpose microprocessors, custom chips or programmable microcontrollers and more complex logic devices available on the market. For rapid prototyping and implementing systems consisting of small number of units, the FPGA technology became very popular and frequently used. They provide high logic density and possibility to easily upgrade the implemented designs in order to comply with the latest standards or to modify the function or the structure of implemented system. Another benefit of FPGA design in comparison with custom chips is their relatively short design cycle supported by the possibility of using existing low cost design tools. These benefits together result in low non-recurring engineering costs (NRE) for FPGA design. On the other side, their drawback is their vulnerability to radiation effects [61]. This mainly concerns SRAM-based FPGAs which are becoming increasingly popular for many applications due to their high-throughput capabilities and relatively low cost. The use of fault tolerant system design can be the solution to overcome their higher rate of fault occurrence.

The ability of FPGA to be configured many times also brings new possibilities from the perspective of system fault tolerance. When the system in FPGA is affected by fault, the reconfiguration can be used to overcome its effects. Partial dynamic reconfiguration capable to reconfigure only some parts of implemented system while the others can run without interruption and also to change their layout and connections in FPGA can be used to implement the new advanced fault localization and mitigation methods. This flexibility allows the use of same FPGA for multiple missions without the need of replacement. When some resources of FPGA are permanently damaged, the custom circuit designs can be created to avoid these resources and the implemented application can continue to run further in the same piece of FPGA. With this approach, we can achieve very good dependability and extend the operational time of the system in harsh environmental conditions.

The aim of this work is to propose alternative methodology for fault tolerant design into FPGA. This methodology can be used in systems with limited redundant area where no spare resources can be activated during the system lifetime, only the resources dedicated at the system design time can be utilized. To mitigate the faults which will appear during the system operation the partial dynamic reconfiguration of FPGA will be used. The methodology will focus on recovering from errors caused by transient and also several independently appearing permanent faults. The SEU faults will be simulated by injecting faults into configuration memory of FPGA. In the end of this work the hardware overhead of this solution is evaluated and the quality of the design of secured system is tested.

1.2 Structure of thesis

The introduction to the topic of fault tolerant system design in FPGAs and brief motivation is described in the first chapter. The second chapter will focus on the presention of current technologies which can be used for electronic device implementation together with their benefits and known drawbacks. This chapter also presents the problem of system depedendability and its impact on system operational lifetime and introduces the concept of fault tolerance of system. The introduction to typical problems in the field of fault tolerant system design and known approaches for coping with them are described in the third chapter. This chapter includes many known techniques, starting with the fault tolerant techniques which can be used universally for all typical fabrics of which electronic devices can be made through to special techniques for SRAM-based FPGAs. The motivation for the research in this topic together with the goals of research are described in the fourth chapter. The fifth chapter is focused on the description of the proposed methodology. At the beginning, the key parts of methodology are presented. The way, how the original system has to be modified to extend its operational lifetime is described. The following sections include the description of utilizing the FPGA features such as partial dynamic reconfiguration to achieve this. The methodology involves specific FT architecture design. The design of these architectures based on the user requirements on dependability indicators and occupied resources on chip can be automated. The FT architecture design together with developed tool for its automated design and implementation is the content of chapter six. In the seventh chapter, the experimental results and their evaulation are presented. In the last chapter of thesis, the obtained results are summarized and the benefits of research are stated. It includes also the possible ways for the subsequent research orientation in this area.

Chapter 2

Common knowledge

In this chapter, the digital systems technologies and concept of fault tolerance will be presented.

2.1 Digital systems

The circuit implementing certain function can be designed using two basic approaches - as an analog or a digital system. While the analog systems use continuous set of input and output values, the digital system is based on the use of finite number of discrete values. Usually, two values are used - logical one and logical zero. These two approaches are often mixed in real systems. While the analog part of the system can be used for processing the signal on system input the digital part is performing the computation. Although the first mass-produced electronic devices were analog, during the last decades the digital system became more popular. The main drawback of analog design is its susceptibility to noise where small change in the signal can cause a significant change in the information present in the signal and can cause the information to be lost. Since digital signals take on one of only two different values, a disturbance would have to be about one-half the magnitude of the digital signal to cause an error. This property of digital circuits can be exploited to make signal processing noise-resistant. With proper techniques such as securing and detection codes the corrupted digital signal can be easily reconstructed [43].

Digital systems mostly use signals with 2-level logic. It means that voltage value on signal should be assignable one of two logical values according to their tolerance intervals stated by the manufacturer of digital circuit. Two possible implementations of logical values then exist:

- In *positive logic* the logical one is represented by higher voltage and logical zero by lower voltage.
- In *negative logic* the logical one is represented by lower voltage and logical zero by higher voltage.

In 2-level logic the voltage values between lower voltage tolerance interval and higher voltage tolerance interval represent undefined or forbidden values. This undefined values can cause problems in digital circuit since the implemented functions are defined only for two possible logical values (zero and one) on input signals. For other values on inputs the output will be undefined. Signals with two levels can be used in Boolean logic for digital circuit design or analysis. The basic building blocks for digital systems are gates implementing basic Boolean functions such as negation (NOT), conjunction (AND), disjunction (OR) and exclusive disjunction (XOR). These blocks transform the input vector of binary values to output vector of binary values. The complex digital systems can be built by connecting these simple gates.

Other types of digital circuits using 3-level logic gates with the third logical value of high impedance [21] or more levelled logic exist (e.g. [1]). New gates can be built also with polymorphic electronics where the gates can implement different functions according to the state of environment (temperature, power supply voltage, light) [54].

The digital circuit can be split into two classes:

- *Combinational logic* refers to the circuits the output of which is a function of the present values of the inputs only. Combinational logic circuits do not contain any memory elements and thus when the inputs are changed, the information about the previous inputs is lost. The behaviour of combinational circuits is described by the set of output functions.

- *Sequential logic* refers to the circuits the outputs of which are also dependent upon past inputs and outputs to them. Thus they implement some form of memory. They consist of two parts. The function is implemented by combinational logic and its outputs are stored in registers. The values in these memory elements are called state variables and can be used in the subsequent cycles of computation. The behaviour of sequential circuit is described by the set of next-state functions and the set of output functions.

  Sequential circuits can be divided according to the way in which the circuit changes its output.

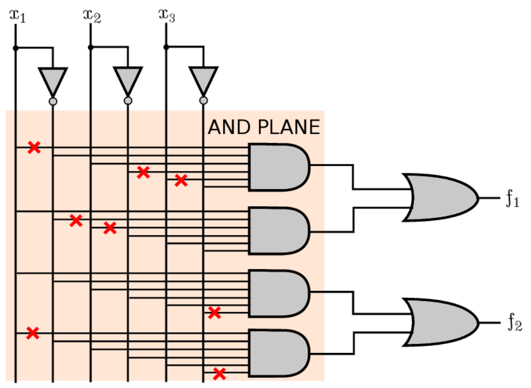
  - *Asynchronous sequential circuits* change their state (and outputs) immediately when input vector changes. The state time of this circuit depends only on the internal logic circuit delays. As an example, asynchronous counter can serve.
  - *Synchronous sequential circuits* use the synchronisation signal usually called clock signal. The input vector is sampled just with the change of clock signal. The concept of the global clock signal for all units in the system can be used or several independent clock signals can exist.

In complex digital systems, both combinational and sequential subsystems can be identified. Last decades brought still the growing effort of creating an integrated circuit (IC) by combining millions of gates and billions of transistors into a single chip. This process is known as Very Large Scale Integration (VLSI). It is a structured design flow that enables a great number of transistors to sit together and work on a single microchip by saving microchip area.

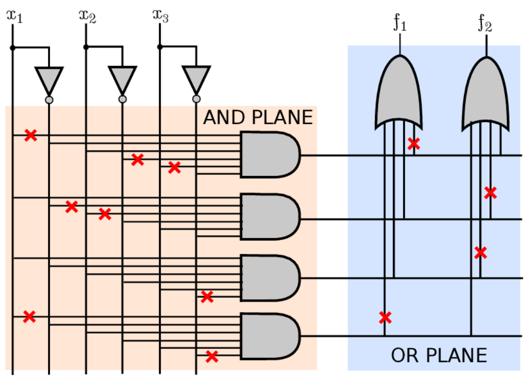
According to requirements on system reconfiguration we can distinguish between two approaches: fixed logic devices for specific applications and programmable logic devices.

2.1.1 Fixed logic devices

For a specific usage in some applications, the Application Specific Integrated Circuit (ASIC) can be developed. These types of circuits stand on the opposite side to the usage of circuit for general purpose such as microcontrollers or other programmable logic devices. The ASICs have fixed configuration of gates and interconnections and they perform one function



(a) PAL scheme



(b) PLA scheme

Figure 2.1: Simple programmable logic devices

or a set of functions forever. Once they are manufactured, they cannot be changed. The time required to go from design to prototypes and to a final manufacturing run can be several months depending on the complexity of the device. Every error in design phase or change of requirements can cause the necessity of developing a new design. Specific design is often not reusable in other circuit designs. On the other hand, ASIC can be very well optimized to satisfy requirements of specific application (for timing, space, cost, etc.).

2.1.2 Programmable logic devices

A Programmable Logic Device (PLD) is an integrated circuit with internal logic gates and interconnects. These gates can be connected to obtain the required logic configuration. The circuit can be configured by the end user to realize various designs. Programming of such a device often involves placing the chip into a special programming unit, but modern chips can be often configured in-system which means that its configuration can be modified directly in application where this chip is used. Another (more modern) term for PLD used in literature is Field-Programmable Device (FPD).

The term PLD includes several specific concepts of reconfigurable devices:

- *Simple PLD (SPLD)* is the simplest, smallest and least-expensive form of programmable logic devices.
  - *Programmable Logic Array (PLA)* is a small PLD which can realize a sum-of-product functions by implementing them using a set of input inverters, AND-gates and OR-gates. Both planes, AND-plane and OR-plane can be programmed to realize a function (see Figure 2.1b).
  - *Programmable Array Logic (PAL)* is a small PLD that has the same components as PLA with the difference in fixed OR-plane (see Figure 2.1a).
- *Complex PLD (CPLD)* consists of an arrangement of multiple SPLD-like blocks on a single chip (see Figure 2.2).
- *Field-Programmable Gate Array (FPGA)* is a PLD featuring a general structure that allows very high logic capacity. While CPLDs offer logic resources with a wide number of inputs (AND planes), FPGAs have more narrow logic resources with higher ratio of flip-flops to logic resources.

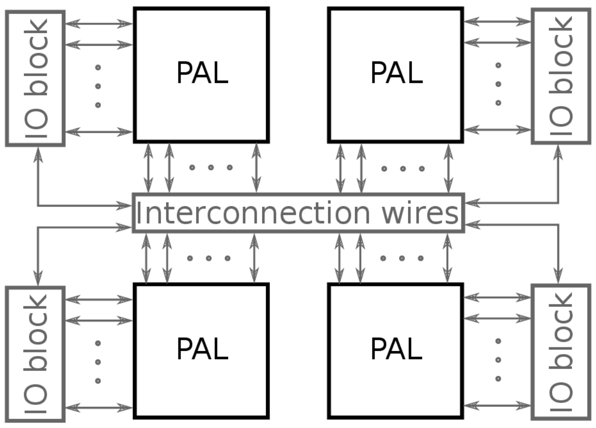


Figure 2.2: CPLD scheme

2.2 FPGA

The semiconductor industry makes a huge progress in last decades. From the first expansion when the transistor was introduced over the spread of integrated circuits to the era of ASICs the evolution of electronics was driven forward in order to get the fastest, smallest and cheapest system implementations. In parallel with fixed logic devices, the programmable logic devices have been developed with the beginning of 1970's. This approach started to play bigger role with the introduction of field-programmable gate arrays in late 1980's. The first FPGA (XC2064) was constructed by Ross Freeman in 1985. The FPGA chip spread across all industries is now driven by the fact that FPGAs combine the best parts of ASICs and processor-based systems. FPGAs can reach up to hardware-timed speed and reliability, but they do not require so big effort and expenses to create custom ASIC design.

The application in FPGA is defined by the configuration memory which determines the function of its blocks and also how these blocks are connected together. This configuration can be programmed after manufacturing, there are some one-time programmable FPGAs available, but the dominant types are SRAM-based which can be reprogrammed as the design evolves. The most of FPGA market is divided between Xilinx and Altera company. In 2015, they together occupied more than 85% of it [44].

2.2.1 FPGA structure

FPGAs are programmable logic devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects (see Figure 2.3). The application can be implemented by designing configuration for the matrix of CLBs (configure their implemented functions) and interconnection between them.

The configurable logic block is the core of the logic structure of FPGAs. The logic blocks within a CLB reside slices that consist of look-up tables (LUTs), carry chains, and registers. These slices can be configured to perform logical functions, arithmetic functions, memory functions, and shift register functions. Their stucture differs in construction between diffent families of FPGAs. The structure of CLB from Xilinx Virtex 7 FPGA family is shown in Figure 2.4. The function of CLB is determined by its configuration which is a small part of overall configuration bitstream. The first FPGA XC2064 contained only 64 configurable logic blocks, each of them consisting of two three-input lookup tables. Nowadays, in high

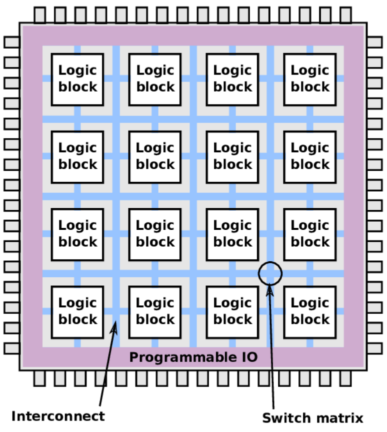


Figure 2.3: FPGA architecture

end FPGAs there are hundreds of thousands of CLBs on single chip with four six-input LUTs.

FPGAs are reffered to as the coarse-grained architectures because besides the configurable matrix of CLBs they also include different types of resources implementing frequently used system functions. These resources are reffered to as hard blocks and they are added to FPGA to avoid the need to implement these units in CLBs and also due to fact that they can be optimized to give better performance and spare some logic on the chip. Typical examples of these hard blocks in FPGAs are Block RAM memories (BRAMs), Digital Signalling Processor (DSPs) units, high speed I/O blocks (IOBs), clocking manager modules such as Digital Clock Managers (DCMs) in Xilinx FPGAs, communication modules (Ethernet, RS232), etc.

The matrix of CLBs is connected with outer environment by programmable I/O blocks. They provide the interface between package pins and the internal configurable logic. They can be used for enhanced source-synchronous interfacing. With this, the optimizations such as clock dividers, data serializers/deserializers, per-bit deskew for input and output signals, dedicated I/O and local clocking resources become available [67]. Signals from the input of FPGA travel through the global routing network and are processed in the CLBs or other hard blocks. Processed signals are then routed back to the IOB as an output or routed to another destination for further processing. The Programmable Interconnect Points (PIPs) provide the routing paths used to connect the inputs and outputs of IOBs and CLBs into logic networks. A PIP is a CMOS transistor switch that can be configured to be turned on or off.

2.2.2 Storing bitstream configuration

Configuration bitstream can be stored in FPGA using various technologies:

- *SRAM-based FPGAs* use the SRAM memory cells based on array of latches to store configuration data for CLBs and other resources settings. They are popular due to their high throughput and ability to be reconfigured many times. They are also usually cheaper option to other types of memory. Their drawback is the fact that this memory is volatile and thus the configuration has to be loaded again to FPGA when it is powered on. Modern SRAM-based FPGAs have highest densities, but consume a lot of power and need an external non-volatile memory to store configuration bit-

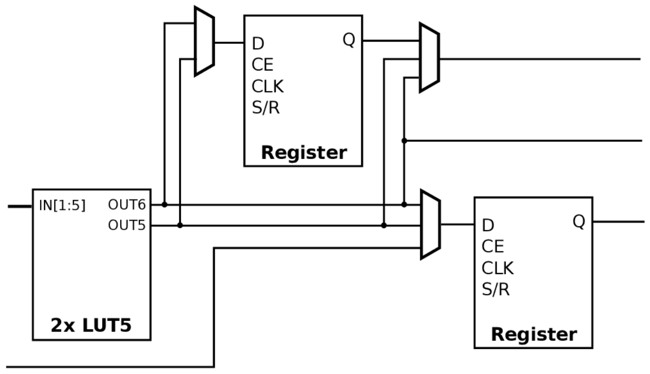


Figure 2.4: LUT and flip-flops connections inside CLB (Xilinx Virtex 7)

stream. This can be done by two different approaches. When master mode is used, the FPGA itself reads configuration data from an external source (ie. flash memory). In the slave mode some external controller is used to load the configuration from external bitstream memory to SRAM memory in FPGA. Frequently some dedicated configuration interface or boundary-scan (JTAG) interface is used to load the configuration into FPGA's configuration memory. SRAM-based FPGAs include most chips of Virtex and Spartan families from Xilinx and Stratix and Cyclone families from Altera.

Some SRAM-based FPGAs can be equipped with an internal flash memory. This non-volatile memory can be used to store the configuration when the FPGA is not powered and thus this FPGA does not need any external configuration memory. In addition, there can be more stored configurations from which the current one is chosen during the FPGA startup. This approach can be used to prevent unauthorized bitstream copying. This can be found for example in Spartan-3AN FPGAs from Xilinx [64].

- *Flash-based FPGAs* use flash memory as a primary resource for configuration storage. There is no SRAM memory to hold configuration data in FPGA. This technology has an advantage of being less power consumptive. Flash-based FPGAs are also more tolerant to radiation effects and due to their volatility they also can be a solution to prevent unauthorized bitstream copying. To flash-based FPGAs belongs families such as Igloo and ProASIC3 manufactured by Actel [4] [5].

- *Antifuse-based FPGAs* are different from the previous types since they can be programmed only once. After manufacturing the antifuse FPGA is not set by any configuration. The function of CLBs and other blocks and the interconnection is set permamently when the configuration is loaded to FPGA and the antifuse is burned. After this moment, the antifuse-based FPGA cannot be reconfigured. As an example of these FPGAs the Axcelerator family manufactured by Actel can be mentioned [62].

2.2.3 Xilinx Virtex family

Virtex is high-performance family of FPGA from Xilinx company. They are SRAM-based FPGAs and they can offer a great number of logic blocks together with the variety of built-in hard macros and can be used for high-performance applications. During the last two decades this family evolved with the shrinking node size enabling to add more logic to single chip. The comparison of number of resources available in different Virtex families can be seen in able 2.1. Virtex families are further divided into subfamilies according to their desired application. According to it, they have different row and column count in their matrix of CLBs and different variety of hard macro resources. The LXT subfamily is designed for advanced logic applications, SXT for signal processing, TXT systems with double density for advanced serial connectivity and FXT for embedded systems.

FPGA family	Technology [nm]	Slices [#]	Distibuted RAM [Kb]	Block RAM [Kb]	DSPs [#]
Virtex 4	90	5472 - 63168	86 - 1392	648 - 9936	32 - 512
Virtex 5	64	3120 - 51840	210 - 3420	936 - 18576	24 - 1056
Virtex 6	40	11640 - 118560	1045 - 8280	5616 - 38304	288 - 2016
Virtex 7	28	51000 - 178000	4388 - 17700	28620 - 50760	1260 - 2520

Table 2.1: The comparison of Virtex FPGA families

In this research, FPGA XC5VSX50T from Virtex 5 SXT subfamily was mainly used for design implementation. This subfamily contains CLBs based on 6-input LUT, 36 Kbit dualport BRAM modules which can be also configured as two independent 18 Kbit dual-port RAM blocks and in addition these BRAMs can be connected in cascade to form a larger memory block. There is also a possibility to utilize cascadable embedded DSP48E slices with two's complement multipliers and 48-bit adder/subtracter/accumulator for parallel computing or to use each of DSP slices to bitwise logical functions. The CLB is divided into two different slices, the first type is referred to as SLICEL and has only capability to implement logic function and the second type SLICEM can be used as a memory and implement 32-bit shift register or 64-bit distributed RAM. The maximal operational frequency of FPGAs from this family is 550 MHz.

2.2.4 FPGA reconfiguration

A considerable part of the FPGA (in terms of its area) is used for configuration memory which defines the implemented hardware circuitry (application) in FPGA. It contains the configuration of CLBs, IOBs, DSPs, BRAMs and other resources and also the routing between these blocks. The process of loading the configuration bitstream through reconfiguration interface to the configuration memory is called reconfiguration of the FPGA.

Depending on the structure of reconfigurable device, we can divide reconfiguration according to its granularity. The granularity of reconfiguration is defined as the size of the smallest block of reconfigurable device which can be addresed by mapping tools. Fine-grained architecture offers greater flexibility for the implementation of design. The drawback of using these architectures can be the increase of power, area and delay because of greater quantity of routing. Corse-gained architecture uses bigger configurable blocks which can be optimised for its intended use and are typically oriented on word-width datapaths. This can reduce the area, time and routing requirements and the reconfiguration time. The drawback can be seen in possible inefficient utilisation of resources such as in case when the operands width of implemented function unit does not meet with the building block of reconfigurable architecture. Also the implementation of design into coarse-grained architecture is usually more difficult due to the need of sufficient number of specific building blocks. Typical FPGA combines these approaches together by using the array of CLBs with interconnection matrix and specific hard blocks (DSPs, BRAMs, etc.).

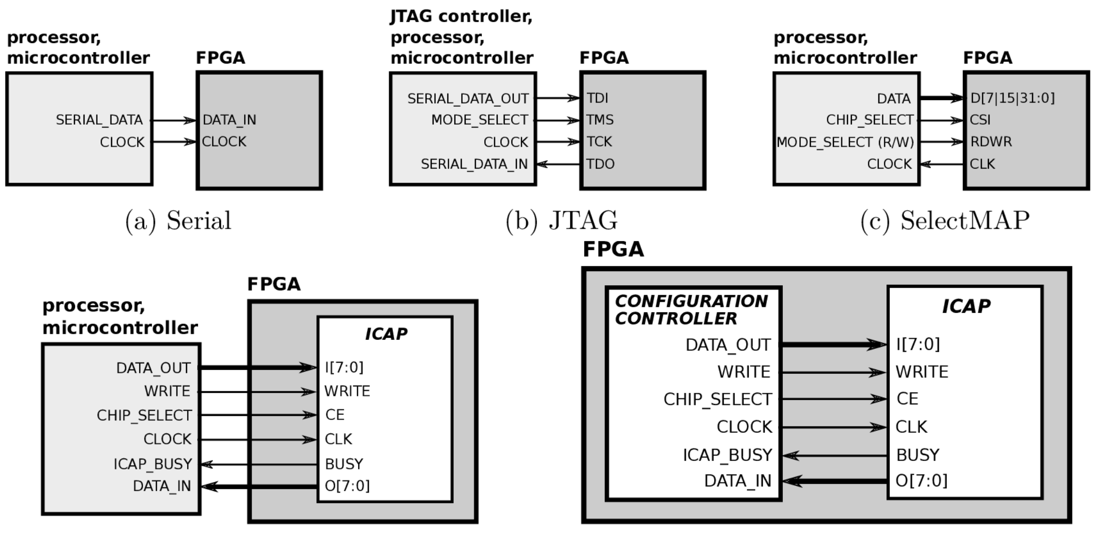
The important measure of reconfiguration is the deployment time. This means the time needed to finish this process and bring the FPGA application back into running mode. The deployment time depends at most on the size of configuration bitstream and the width of datapath between the bitstream storage and reconfiguration interface. Typical time for reconfiguration of entire medium sized FPGAs such as Virtex 5 SXT 50 FPGA is several milliseconds. Several ways how to burst the process of reconfiguration exists. These include the partial reconfiguration of just a small portion of FPGA which has to be changed or the use of compressed configuration bitstream.

In modern FPGAs, several different types of reconfiguration can be identified:

- *Full reconfiguration* is the process of changing configuration of all resources in device. While the full reconfiguration process takes place, the device is running only in idle mode. The implemented application begins to run immediately after this process is finished. Typically, the full reconfiguration is done after power-up of device.
- *Partial reconfiguration* is the process of changing a portion of reconfigurable hardware circuitry while the rest of design is not changed. Just the partial configuration bitstream is written to configuration memory. Partial reconfiguration can be divided according to its influence on the run of application in FPGA:
  - *Partial static reconfiguration* process stops the running application (circuitry) even if only a part of it is changed. The application in device is brought up after this process is completed.
  - *Partial dynamic reconfiguration* (PDR) process changes a portion of implemented circuitry without any intervention to the rest of it. This requires a special FPGA design flow where each part of FPGA allowing PDR has to be encapsulated by adding special macros on buses going inside or outside from this part to isolate it from the unchanged (static) part.

FPGA can be configured via different configuration interfaces. Xilinx FPGAs from Virtex family offer the following interfaces:

- *Serial Peripheral Interface* (SPI) is an external configuration interface of FPGA containing only two single bit signals (clock and data). The FPGA is configured by loading single bit of data per clock cycle. This interface is typically used for devices in a serial daisy chain or in case when single device is configured by an external microprocessor or CPLD. The connection of configuration controller and FPGA is shown in Figure 2.5a.
- *Byte-wide Peripheral Interface* (BPI) is similar to SPI with one difference, data signal is 8-bit or its multiplies wide.
- *JTAG* (Joint Test Action Group) is external interface based on serial data transfer. It is primarily used for testing and debugging purposes defined by IEEE standard 1149.1. It contains a Test Access Port (TAP) and boundary-scan architecture. In programmable devices such as FPGA, it can be used for in-system programming (configuration). Several devices can be connected in daisy chain and then they can be configured at once. This interface has bigger priority than others. When the JTAG controller is loading data serially on Test ClocK (TCLK) signal edge the run of other interfaces is stopped. The connection between JTAG controller (or other device implementing its function) and FPGA is shown in Figure 2.5b.



(a) Serial (b) JTAG (c) SelectMAP

(d) ICAP with external controller (e) ICAP with internal controller

Figure 2.5: The configuration interfaces of FPGA

- *SelectMap* is external configuration interface which supports bidirectional communication with FPGA device by an 8-bit, 16-bit, or 32-bit data bus. It can be used either for configuration of device or for bitstream readback from device. Data bus width is automatically detected. There are three possible modes of configuration: single-device, multiple devices connected as daisy-chain where each device can be configured by different bitstream and multiple devices connected in parallel where every devices are configured by the same bitstream. FPGA and configuration controller connected via SelectMAP is shown in Figure 2.5c.

- *ICAP* (Internal Configuration Access Port) is a fast internal parallel interface. The ICAP interface is a subset of the SelectMAP interface and thus they cannot be used simultaneously. The process of configuration of device and its bitstream readback is essentially the same as with the use of SelectMAP. Since ICAP provides ability of partial self-reconfiguration of FPGA the care must be taken during reconfiguration to avoid the change of reconfiguration controller and all circuitry performing the reconfiguration process. These parts of FPGA should be made static (not dynamically reconfigurable). Thus, the ICAP cannot be used for full reconfiguration of FPGA and it is designated to perform partial dynamic reconfigurations.

  When implementing the FPGA design with PDR ability, the ICAP primitive can be instantiated in FPGA and it can be driven by reconfiguration controller which will read the partial bitstream from non-volatile memory and transfer it to ICAP interface during PDR process. This controller can be located in different fabric as shown in Figure 2.5d or it can be instantiated in the same FPGA as shown in Figure 2.5e.

2.2.5 FPGA design synthesis and implementation

To implement the digital system into FPGA the series of steps are needed. These steps can be grouped to design entry and its synthesis and implementation. These steps can be done separately but modern advanced design tools such as Xilinx ISE offer the possibility to do all these steps in one development environment.

At the beginning, the system description has to be specified in some programming language or some visual editor. To simplify this process, special Hardware Definiton Languages (HDL) were introduced. They accent the parallelism in digital circuits and they offer the possibility to specify the system by its behaviour and structure. VHDL and Verilog belong to most known HDLs and they are used as an open standards of Institute of Electrical and Electronics Engineers (IEEE) to describe the digital circuits on different levels and for simulation purposes. They can be used for digital system design not only for FPGAs but also for CPLDs. The common level of abstraction in HDL is Register Transfer Level (RTL). It is based on the premise that synchronous systems can be described as a set of registers which are connected between themselves and with inputs and outputs by combinational logic.

As the optional step after the design entry in HDL, the simulation of design can be run in some simulation tool such as ModelSim or ISim and it can uncover the design faults or timing limitations. A set of testbenches is needed to perform the simulation.

After succesful simulation, the design synthesis is performed. This process converts the HDL input to netlist files which describe the system as a set of logical gates and their connections. The synthesis is consisted of several steps. After parsing of HDL file, the synthesis tool tries to infer specific design building blocks (ie. MUXes, RAMs, adders, etc.) for which it is able to create efficient technology implementations. The next step is Finite State Machine (FSM) recognition. When some FSM is identified in the design, the most efficient encoding algorithm for its implementation can be chosen according to the specified optimization goal (i.e. area, speed). The synthesis tool also tries to reduce the amount of inferred macros and share some resources which can lead to a reduction of the area as well as the increase in the clock frequency. Finally, the low level optimization is made including implementation of macros, timing optimization, technology mapping and register replication. The output of synthesis are files with netlists described most often by Electronic Design Interchange Format (EDIF) or some vendor specific file format such as NGC from Xilinx. This file can also contain the contraints specified by designer which will be used in further steps of design implementation. In constraints file, the designer can specify timing, placement, and other design requirements.

Some design tools such as Xilinx ISE also offer the possibility to add to system design pre-synthetised macros which are optimized for its purpose (memories, bus controllers, etc.) and can be also customized (bus widths, memory sizes, etc.). These macros can be added as an additional input to the synthesis tool and synthetised together with the HDL sources.

When the design is synthetised it has to be implemented to specific FPGA. This means that the logical design is converted into a physical file format that can be downloaded to the selected target FPGA. The implementation can be customized by setting custom goals and strategies in constraints file which are taken into account by implementation tool. Adding constraints can cause the situation that the implementation tool will not be able to implement design due to insufficient number of resources in specified FPGA and timing or routing issues.

The implementation of design consists of several consecutive steps. These steps are often done as a batch by the design tool.

- The *translate* process merges all netlist files from synthesis together with design constraints and create Native Generic Database (NGD) file. This file contains the logical design reduced to FPGA primitives.

- The *map* process is mapping the logic primitives from previous step into available resources on the target FPGA such as CLBs and IOBs. The output design is a Native Circuit Description (NCD) file that physically represents the design mapped to the components of FPGA.
- The *Place And Route* (PAR) process takes as an input a mapped NCD file and it places and routes the design. An NCD file with routing prepared for bitstream generation is created as an output of this step.
- The *bitstream generation* process encodes the NCD file with routing into configuration bitstream for the target FPGA. The output bitstream can be directly used for target FPGA reconfiguration.

The FPGA design workflow used by Xilinx ISE tool is show in Figure 2.6.

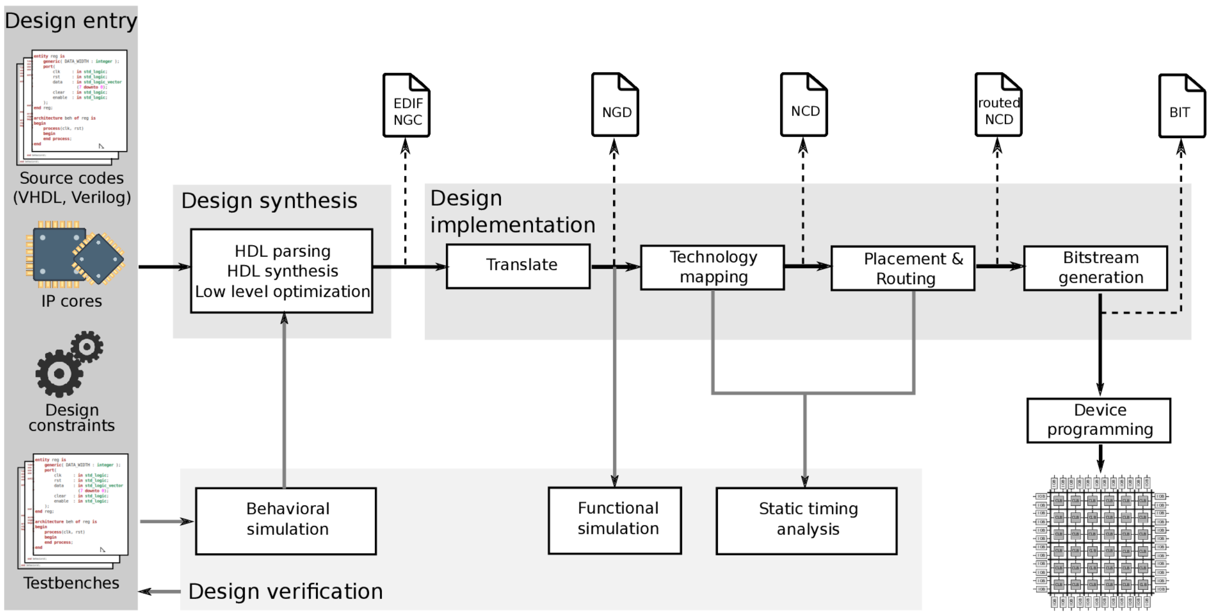


Figure 2.6: Design workflow by Xilinx ISE tool

2.2.6 Design workflow with partial dynamic reconfiguration

This paragraph describes the changes to standard design workflow when PDR is used. It is shown in Figure 2.7.

In Xilinx design workflow, the hierarchical design must be strictly used. This means the design has to be partitioned into modules which create hierarchical structure where on the top of hierarchy is one top-level module. In this module, the static and dynamic components can be identified. While design floorplanning, the Partial Reconfiguration Regions (PRRs) for each Partial Reconfiguration Module (PRM) have to be determined. There can be more PRMs assigned to one PRR enabling the dynamic change of implemented circuitry in FPGA. The size of PRR and its shape is limited according to FPGA architecture structure. Typically, the FPGA is divided into several tiles containing the same number and type of resources and the same relative position of resources in tile. PRR can consist of one or more of these tiles.

The implementation of design using PDR begins with top-level module which consists of static modules and PRMs. While implementing top-level design, the PRMs are instantiated as black-boxes. On the boundary between PRMs and static part of design, the logic for isolating these modules during PDR is added. Previously, Xilinx FPGAs were using the bus macros for this purpose. Bus macro is interface consisting of two LUTs applied to each single signal. The first LUT is placed in PRM next to its boundary and the second is place next to it but outside (in the area neigbouring with this PRM). Nowadays, the proxy logic is used [66]. Proxy logic is using single LUT for each signal and it is placed automatically by implementation tool. When top-level module is implemented with PRMs instantiated as black-boxes, constraints for static part and PRMs are determined and full design with chosen startup PRMs is implemented. In the next phase, each PRM is implemented in turn. The MAP and PAR process is constrained by one or several constraint files.The last phase is merging of each PRM created with static design. Their interfacing correctness is verified and partial reconfiguration bitstreams are created. Finally, the full configuration bitstream which will be used after power-up of FPGA is created.

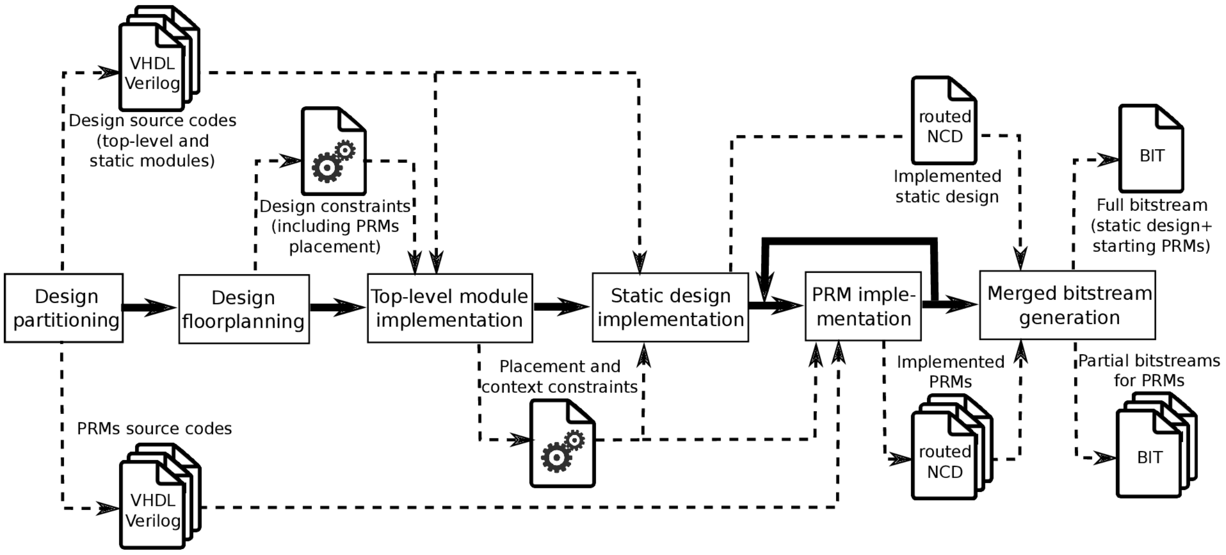

VHDL
Verilog
Design source codes (top-level and static modules)
Design constraints (including PRMs placement)
routed NCD
Implemented static design
BIT
Full bitstream (static design+ starting PRMs)
Design partitioning
Design floorplanning
Top-level module implementation
Static design implementation
PRM imple-mentation
Merged bitstream generation
Placement and context constraints
Implemented PRMs
routed NCD
Partial bitstreams for PRMs
BIT
PRMs source codes
VHDL
Verilog

Figure 2.7: Design workflow with PDR by Xilinx

2.2.7 Configuration bitstream structure

The configuration bitstream contains the necessary data for configuration of FPGA resources, i.e. definition of routing between components via setting the programmable interconnect points, enabling of flip/flops or LUTs inside CLBs, definition of LUT functions, etc. The configuration data of Xilinx FPGAs is arranged into configuration frames. One frame is the smallest unit of the configuration which can be addressed and handled by the reconfiguration interface of FPGA and its internal reconfiguration logic. The size of one configuration frame varies according to FPGA family, in Virtex 5 FPGAs family the size is 1312 bits and it is formed by 41 32-bit words [67].

As the single frame configures (partially) 20 CLBs located in one FPGA column at once, the smallest possible PRM which can be created must contains multiples of these 20 CLB columns. To configure these 20 CLB completely, the set of 36 frames is needed (see Figure 2.8).

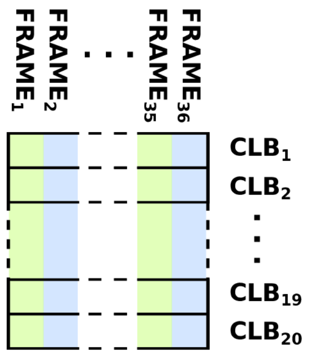


Figure 2.8: The configuration of the smallest possible PRM by configuration frames

The complete configuration bitstream can be divided into three parts.

- The *header* is composed of the mixture of synchronizing words, dummy words and 32-bit commands used to initialize the reconfiguration process by setting internal registers. The main goal is to prepare the FPGA to receive the subsequent configuration data organized in frames. These commands include the initialization of Cyclic Redundancy Check (CRC), setting the configuration and control options, setting the Frame Address Register (FAR) with the address of the first configuration frame. At the end of header there is the write configuration command starting the process of loading configuration frames.
- The *configuration data* is consisted of the bulk of configuration frames. Due to the fact that the frames are typically organized to configure consecutive blocks of resources, the frame address is automatically incremented when the next frame is recognized.
- The *footer* is used to issue the commands to finish the reconfiguration (i.e. CRC) and prepare the FPGA for the start (i.e. the reset of flip-flops).

The mapping between the bits in the configuration bitstream and the specific FPGA resource is typically not documented by the manufacturer. This relation is not needed for the standard system design flow in FPGA but it makes some task difficult such as precise fault injection via bitstream manipulation [50].

2.3 Faults in digital systems

Very fast scaling of technology in last decades has an adverse impact on the reliability of components for digital systems. An increasing error susceptibility for disturbances from the operational environment can be identified but the components also become more vulnerable to permanent faults. Faults occur in a digital system in all phases of its existence - from the design and fabrication phase through the whole lifetime.

2.3.1 Terms

In this paragraph the difference between the terms fault, failure and error is presented.

- A *fault* is a difference in hardware configuration between correctly configured system and its current state. If there is no difference, fault is not present.

- An *error* in a system is a deviation from the required operation of system or subsystem which causes the difference between actual processed data and expected correct data. The reason of error occurrence is fault presence in the system.
- A *failure* of the system means that it is not producing correct outputs and thus the system is working with the behaviour which differs from the required one. A failure is caused by an error.

Not every fault has to manifest itself necessarily by an error which can be detected. Errors that are present in a system but not detected are *latent errors*.

2.3.2 Fault classification

Faults in digital systems can be classified according to their time of occurence:

- *Design faults* are present in the system from the beginning of its lifetime. They can be inherited from the existing system on which the new one is built, created by human designer or by design tool. These faults are present in every piece of final system.
- *Fabrication faults* can be present in the final solution due to an imperfect manufacturing process (i.e. short-circuits, opens, incorrect transistor threshold voltage, improper doping profiles, mask alignment errors, poor encapsulation in VLSI circuits etc.). Accurate identification of fabrication defects is important in improving the manufacturing yield.
- *Operational faults* are caused by external disturbance during the digital system lifetime. The sources of these faults can be operator mistakes, environmental extremes, electromagnetic interference and wear-out failures when the product exceeds its design lifetime.

Faults in digital systems are usually classified according to their duration:

- *Transient faults* appear in the system for a short time, they are mostly caused by random environmental disturbance such as radiation, pollution, humidity, temperature, pressure, vibration, power supply fluctuations, electromagnetic interference, static electrical discharges, ground loops etc. The physical resource is not damaged. These faults are not correlated with each other and it is nearly not possible to detect them due to their hardly predictable influence on the system behavior.
- *Permanent faults* affect the functional behavior of a system permanently. They refer to a physical damage of system resource. This resource cannot be used any more without physical repair or replacement. The error caused by permanent fault is also called hard error.

  The permanent faults can arise as a result of various physical phenomenons, such as:

  - *Time Dependent Dielectric Breakdown (TDDB)* is the result of long-time application of relatively low electric field. It is an opposite to immediate breakdown which is caused by strong electric field. TDDB comes with reduction in gate oxide thickness. The charges are trapped in oxide that creates electric field. The charge flow through the oxide results in a breakdown after some time.

  - *Electro-mitigation* is causing permanent failure of interconnect. It develops voids in metal line due to heavy current densities over time period.
  - *Hot Carrier Effects* are parasitic effects at the drain side of channel and they are caused by hot carriers traveling with saturation velocity. It is attributed by slow creation of traps at the oxide surface. Their impact is the change of the transistors threshold voltages which consequently affects the power and performance of the device.

  The permanent fault can manifest as permanent logical value 1 or 0 on signal which is called stuck-at fault. Another manifestation can be short-circuit between two or more signals, delayed propagation of signal through the circuit compared to specification or delayed level transition from logical 0 to 1 or inversely.

- *Intermittent faults* appear, disappear, and reappear repeatedly. They are difficult to predict, but their effects are highly correlated. Most intermittent faults are caused by the optimistic design or manufacturing not counting with all circumstances which can affect its run. The system works well most of the time, but fails under atypical environmental conditions.

2.3.3 Faults in FPGA

All common faults of digital systems can also occur in FPGAs. There are also other specific faults which can appear in specific FPGA resources such as configuration memory. Most of the modern FPGAs are based on SRAM which is vulnerable to radiation. Alpha-particles as the part of the radiation is the main reason of transient faults presence in FPGA. Errors caused by transient faults are frequently called soft errors.

Very often we are faced with two types of faults:

- *Single-Event Upset (SEU)* causes a random change in the state of a digital memory element (or other sequential element) by an ionizing particle that collided with this element. SEU can be very critical in case of SRAM-based FPGAs and can lead to undesirable change of function implementation. The system can be turned to correct operation by correcting the attacked element to its proper state. The probability of the SEU for one memory cell is extremely small for typical conditions on earth. But with increasing memory size or exposure in harsh environment the probability can considerably increase. This can become a real problem for example in the space or aircraft designs.

- *Single-Event Transition (SET)* causes one or more voltage pulses (i.e. glitches) on signals which can be then propagated through the circuit. There is no need to correct this erroneous state due to its short and temporary existence.

In ASICs, SEU faults are considered as transient. If they hit combinatorial logic, the error on the output lasts only until the next value is processed by the logic. In case they hit sequential logic, the errors exists in storage cell until next value is written into it.

The variety of effects in FPGAs after SEU occurence is bigger as shown in Figure 2.9. In FPGAs, the combinational and sequential logic of implemented circuit is set by the values of configuration memory cells. The combinational logic of final circuit is implemented by LUTs and routing settings. The SEU occurence in LUT changes its implemented function. In routing settings, the impact of SEU can be the change in the connection of logic gate output or input to interconnection matrix. Both faults can be restored by reconfiguration of FPGA by the original configuration bitstream. The sequential logic of final circuit can be implemented by flip-flops in CLBs or by hard blocks such as Block RAMs. The effect of SEU on flip-flops is transient. With the next write operation to memory cell implemented by flip-flop the error disappears. When the SEU fault occurs in Block RAM cell, the reconfiguration of FPGA is needed to restore its state [28].

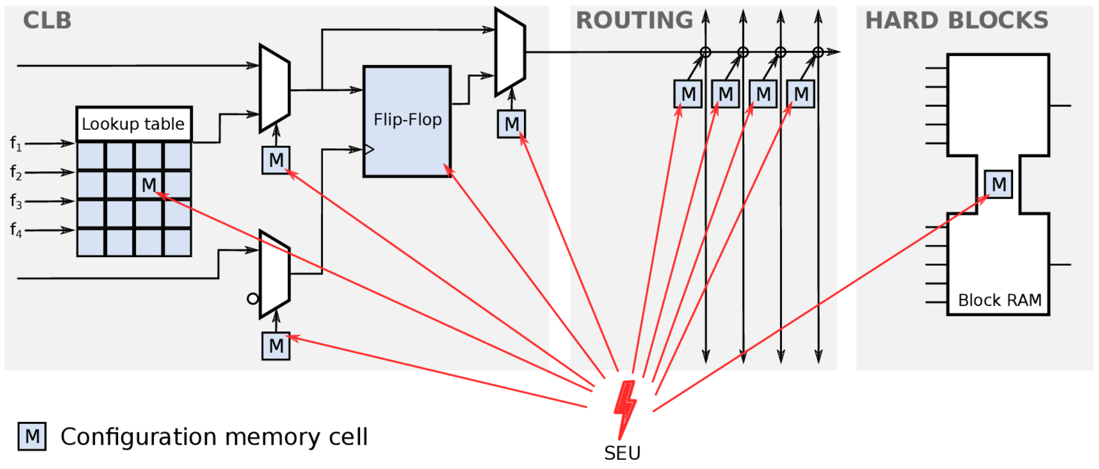


Figure 2.9: Bits in configuration memory and resources of FPGA sensitive to SEUs (Xilinx Virtex FPGA family) [28]

In this work, the fault in configuration memory is considered as transient in case it can be corrected by reconfiguration of FPGA with using the original configuration bitstream and restoring its state by synchronization process. If the reconfiguration of FPGA does not correct the fault, it is considered as permanent fault.

2.4 Dependability of systems

The dependability is the ability of a system to deliver its intended level of service to its users [6]. In other words, the dependability of system (service) can be expressed as the system ability to work without failures at least for the given period and without more severe failures than it is acceptable. With the fact that the computing becomes common in all parts of human life, its dependability plays an important role in all its applications.

The system dependability is an integrated measure consisting of several attributes and it cannot be counted as a single number. It can be evaluated from the availability and reliability parameters of the system, its maintainability, safety and security and other adjacent attributes such as repairability or durability. To summarise, the dependability expresses the ability of system to produce outputs that can justifiably be trusted.

To be able to evaluate dependability attributes, some system fault model has to be adopted. The most frequently used model is using only two states. At the given time the system (component) is either working properly or it is faulty and producing incorrect outputs. The system state can change as the time evolves and this change is done instantly. The change from a functioning to a failed state is referred to as failure and the change from failed state to the functioning state is referred to as repair [25].

The systems can be divided according to their survivability of occured faults to repairable and non-repairable ones. Survivability in this context means that the system can be used again to produce the correct outputs after fault occurence (not necessarily immediately, time for fault mitigation may be needed). The non-repairable system will stay after the first fault occurence in failed state forever. The repairable system can be brought from failed state to functioning state when repair process is completed (if it is possible). It is assumed that the repair process will bring the failed system back to the state where it will produce correct outputs and it will be able to recover from the same variety of possible faults as in the beginning of its lifetime.

The changing of system states according to fault occurences and their repairs is shown in Figure 2.10. The lifetime of system (T_{life}) describes the time from the first run of system to the moment when unrepairable fault (or faults) occurs. System operational time (T_{op}) is reffered to as the time for which the system is working properly. For the non-repairable system we can state that the operational time before the first and only one fault occurence T_{op1} is equal to system lifetime T_{life}. When the system is affected by fault the time needed to its detection and localization by system test equipment should be assumed before fault mitigation is started. If Figure 2.10, the time needed to detect fault is denoted as T_{det}, time needed for fault localization as T_{loc} and fault mitigation time as as T_{mit}. The time needed for system repair consists of these 3 parts.

To simplify the evaluation of system dependability, several dependability statistic indicators have been introduced.

Mean Time To Failure (MTTF) is the expected (mean) time for a system to fail. It is a statistical value and the length of the observation interval for the calculation of MTTF must be infinite. This parameter can be evaluated as the average of all operational time periods (2.1).

$$MTTF = \frac{1}{n}\sum_{i=1}^{n} T_{opi} \tag{2.1}$$

Failure intensity (λ) shows the mean frequency of system failures. It can be derived from MTTF indicator (2.2).

$$\lambda = \frac{1}{MTTF} \tag{2.2}$$

Mean Time To Repair ($MTTR$) is the expected (mean) time for a system to be repaired. This can be evaluated as the average of all repair time periods (2.3).

$$MTTR = \frac{1}{n}\sum_{i=1}^{n} T_{deti} + T_{loci} + T_{miti} \tag{2.3}$$

Repair intensity (μ) shows the mean frequency of system repairs. It can be derived from MTTR indicator (2.4).

$$\lambda = \frac{1}{MTTR} \tag{2.4}$$

Mean Time Between Failures ($MTBF$) is the mean elapsed time between inherent failures of a system (2.5).

$$MTBF = MTTF + MTTR \tag{2.5}$$

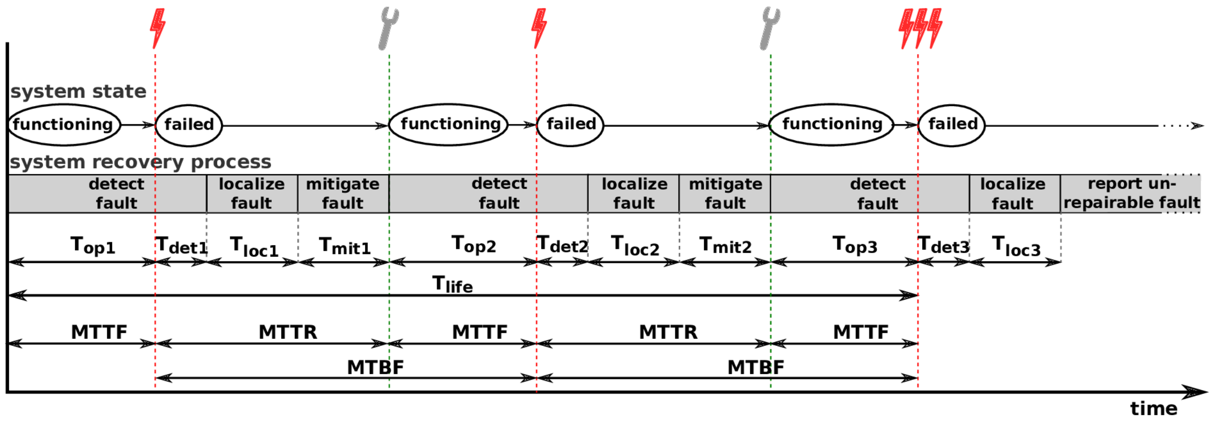


Figure 2.10: The chain of system states during its lifetime

2.4.1 Reliability

The reliability $R(t)$ of a system at time t is the probability that the system operates without a failure in the interval $[0, t]$ under given operational conditions. The correct operation of system at time 0 is a premise. In other words, the reliability can be understood as an ability to continuously deliver correct service and meet requirements of implemented function during the given time period. This correct service ends in the moment of failure occurence. High reliability means that long time interval elapses before the first system failure occurs.

Since the reliability can be expressed as the probability of system run without failures before or at time t, we can define a random variable T as the time to failure and then express it by formula 2.6.

$$R(t) = P(T > t) \tag{2.6}$$

As the opposite to the reliability which expresses the probability of success, the unreliability $Q(t)$ of a system at time t can be defined. It expresses the probability that the system will fail in the interval $[0, t]$. Again, the correct operation of system at time 0 is a premise. The reliability and the unreliability are related as shown in 2.7.

$$Q(t) = 1 - R(t) = P(T \leq t) \tag{2.7}$$

Reliability is a function of time but the specification of time period can vary according to the system under consideration. The time of correct operation can be specified in time units such as hours, days or years or it can be stated for example as the number of correctly processed outputs until a fault can appear.

To model the reliability of (non-repairable) system, an exponential distribution is frequently used (2.8). There is also the possibility of using other distributions to model system reliability but this typically requires more detailed information on the system and a more detailed analysis. For most situations the exponential distribution is adequate.

$$R(t) = e^{-\mu t} \tag{2.8}$$

When the reliability function of system $R(t)$ is known, the failure probability $Q(t)$ (the unreliability of system) can be evaluated (2.7) and used to derive the failure density function $f(t)$ (2.9). The failure density function $f(t)$ is defined as the probability per unit of time that the first failure of system will occur at time t. The correct operation of system at time 0 is a premise. Besides others, this density function can be used to determine the probability of failure occurence within time interval bounded by time t_0 and t_1 as shown at 2.10.

$$f(t) = \frac{dQ}{dt} = \mu e^{-\mu t} \tag{2.9}$$

$$Q(t_0 \rightarrow t_1) = \int_{t_0}^{t_1} f(t)dt = \mu \int_{t_0}^{t_1} e^{-\mu t}dt = e^{-\mu t_0} - e^{-\mu t_1} \tag{2.10}$$

In practice, the failure rate function is used to describe the reliability of system. The failure rate $\lambda(t)$ is defined as the probability per unit time that the failure of system will occur at time t, given that the system was correctly operating at time 0 and has survived to time t. To define failure rate for the repairable system, this premise can be extended as the systems that do not need to survive in original state, but then they have to be repaired to fully operating state at time t.

$$\lambda(t) = \frac{f(t)}{R(t)} = \frac{f(t)}{1 - Q(t)} \tag{2.11}$$

To evaluate the failure rate function $\lambda(t)$, the failure density function $f(t)$ can be used (2.11). In practice, the failure rate is often measured by observing the correct operation of a system. Due to fact, that the failures do not occur frequently, it is often measured using many identical copies of a component or system. As the failure rate of a system depends on time, it can vary a lot over the life cycle of the system. It is often reported, that the failure rate has the shape of bathtub curve as shown in Figure 2.11.

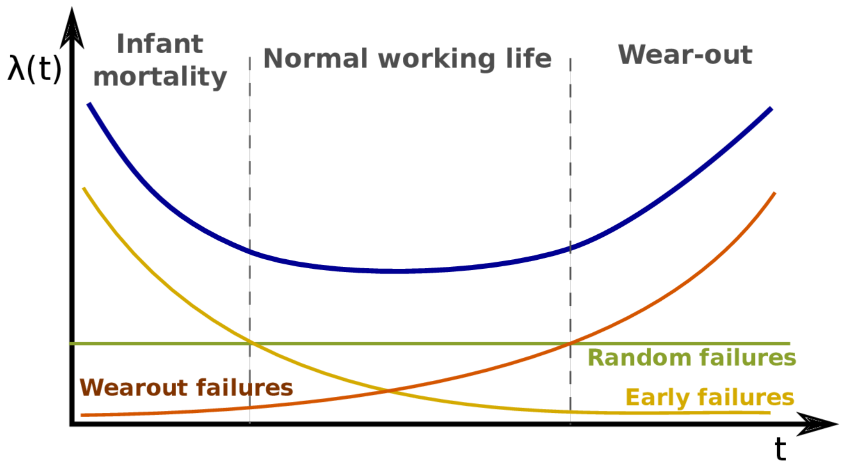


Figure 2.11: The failure rate of a system during its lifetime

At the beginning of system lifetime, there is a high failure rate which is decreasing in time. This period is often called infant mortality or wear-in mode. Failures occuring during this period are typically caused by a variety of factors such as the occurence of defective parts, defects in materials, damages in handling, out of manufacturing tolerance, etc. To avoid this situation, manufacturers frequently perform burn-in process of the product in their factory to avoid such situations when failures from wear-in mode will happen in customer premises. The mitigation of these failures includes design improvement, care in materials selection and tightened production quality control.

When early failures pass away the failure rate typically becomes nearly constant and its amplitude is the lowest during this time. This part of system lifetime is considered as normal life period. The failures occuring in this period are typically considered as random and externally induced. It is difficult to predict which failure mode will manifest, but the failure rate is predictable. This part of system lifetime should be the longest one.

When system is coming to the end of its lifetime, failures typically occur at increasing rates. This period is reffered to as wear-out mode. Wear-out mode failures are mostly caused by material fatigue or by strength deterioration due to cyclic loading. When these failures begin to predominate it is considered that the system has aged beyond its useful life.

For the reliability considerations, the random failure rate (the middle part of bathtub curve) is widely used. The wear-in mode failures are often considered as an issue of quality control, the wear-out mode failures are assumed as the result of poor maintenance.

With the knowledge of system reliability function, MTTF for the first system failure can be derived (2.12).

$$MTTF = \int_0^\infty R(t)dt \tag{2.12}$$

2.4.2 Availability

The availability $A(t)$ of a system at time t is the probability that a system is in operational state (not in failure) at a given time. Its unavailability can be caused by the fault occurence in the system, its mitigation or system maintanance of any kind. Thus, it does not just incorporate the frequency of fault occurence but also the time needed for its repair and for the system maintanance.

The availabilty function $A(t)$ for repairable (and non-repairable) system for given time t, last repair time u with renewal density function $m(u)$ can be derived as shown in 2.13.

$$A(t) = R(t) + \int_0^t R(t-u)m(u)du \tag{2.13}$$

In practice, operational availability A_O as the measure of system availability that includes all possible sources of downtime (failure repair, maintenance, etc.) is frequently used (2.14).

$$A_O = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR} = \frac{\mu}{\mu + \lambda} \tag{2.14}$$

Availability is frequently used as a measure of dependability for systems where short out-of-order states can be tolerated.

2.4.3 Safety

To define safety, we need to split possible failure states in system to fail-safe and fail-unsafe ones according to fact if they create safety hazards to the system or its environment. Then, the safety $S(t)$ of a system at time t can be described as the probability that the system is not in failure state or it is in its fail-safe state in the interval $[0, t]$. The non-fail state of system at time 0 is a premise.

Safety is required in safety-critical applications such as medical, transportation or military systems where a failure can result in human injury, loss of life, or environmental disaster [11].

2.4.4 Maintainability

The maintainability is dependability attribute concerned with the ease of repairing the system after a failure occurence or changing the system to include new features. It is distinct from other dimensions of dependability because it is a static and not a dynamic system attribute. High maintainability means a short downtime for the system repair or its upgrade to new version.

Estimation of maintainability can highlight if the predicted maintanance factors such as downtime, the quality and quantity of maintanance staff or tools are adequate and consistent with the needs of the system operational requirements.

2.4.5 System dependability analysis

To analyse the system dependability parameters, many methods were introduced. From the most popular ones, fault tree analysis, reliability block diagram, reliability graph, Markov chain and Monte Carlo simulation can be mentioned.

- *Fault tree analysis (FTA)* is widely used method for the analysis of system reliability, safety and maintainability. It can be used to determine the cause of undesired event at system level such as hardware failure, human errors, etc. An undesired state of a system is further analyzed using Boolean logic to combine series of lower-level events. Failure states are depicted by square signs in diagram and states where system is operating properly are depicted by circles. Only one fault event is analyzed by single fault tree. FTA has expression power but its construction is not an intuitive process and it requires skilled designer. Enhanced techniques based on FTA such as dynamic FTA were introduced to enable dependability attributes analysis of complex and dynamic systems [15].

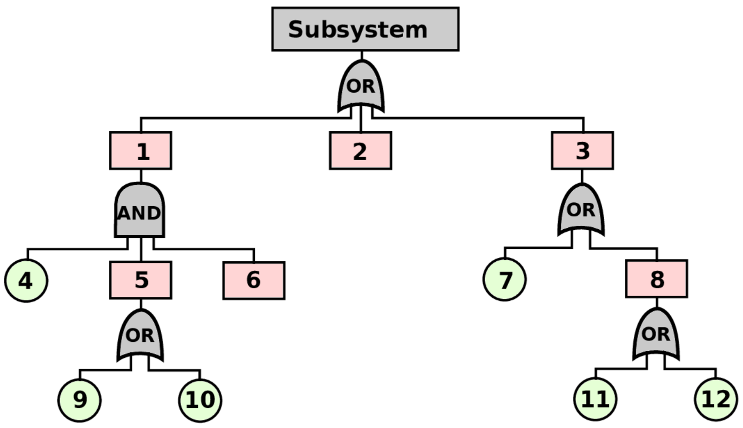
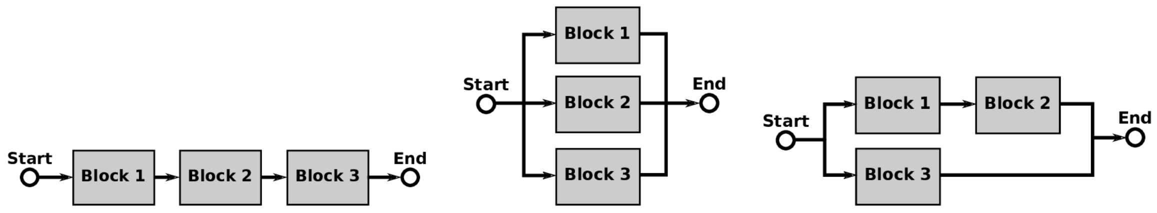


Figure 2.12: Fault tree analysis method

- *Reliability block diagram (RBD)* enables the system reliability and availability analysis for large and complex systems using block diagrams which show the relationship between the components of system. System components can be displayed as series of



(a) Series systems (b) Parallel systems (c) Combinational systems

Figure 2.13: Reliability block diagrams

blocks which can be connected in series, in parallel connection or as the combination of series and parallel. Each block can be assigned by rates of desired dependability attributes such as reliability or availability value, failure or repair rates, etc.

RDB is frequently used to evaluate how much the reliability (availability) of each component contributes to the overall value for entire system. On the basis of RBD, enhanced method such as dynamic RBD were introduced enabling the analysis of such complex systems as multiprocessors [15].

In RBD, system can be modelled as series of connections (2.13a) if it consists of components where a failure of each component will cause the failure of entire system and their MTTF rates are mutually independent. In the series connection, the most important component to the overall system reliability is that one with the least reliability because the overall reliability is always less than the reliability of this component. Overall reliability can be evaluated as a product of all component reliability values (R_i).

To be able to model a system as a set of components connected in parallel, the condition that the system has correct output is met in case when at least one of the component in parallel connection has correct output (2.13b). A parallel connection is used to show redundancy of components and the fact that several paths from start node to end node exist. Overall reliability can be evaluated as the unity complement to the product of all component unreliability values (Q_i).

More complex system can be represented by components connected by combination of series and parallel (2.13c). To create this model, the appropriate decomposition of entire system to subsystems which can be represented either in a series connection or in a parallel connection is needed. Overall reliability can be gained by evaluating partial reliabilities of these subsystems.

- *Reliability graph* model is composed of nonempty sets of nodes and arcs. A node can be used to model a component in a system and an arc can be used to model the transition between two components. A system reliability graph fails when there is no path from the start node to the end node. It is considered as intuitive method for analyzing system reliability because the bijection between the actual structure of the system and the system model can be stated. The drawback of this method is its limited expression power. In Figure 2.14, the sample system delivering data from node 1 to node 4 with 5 transmission lines modelled by 5 arcs is shown [29].

- *Markov chain models* are used to model random processes with the Markov property. This property refers to the memoryless property of the stochastic process. The modelled random process can be described by a set of states and transitions with their

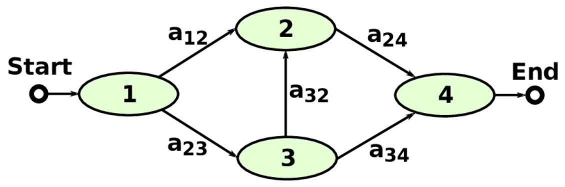


Figure 2.14: The reliability graph of system delivering data from node 1 to node 4

probabilities where the next state only depends on the current state but not on the past state. Markov chain models can be graphically shown as the state transition diagrams comprised of various possible states of the system where the transitions between various states are described in terms of the rates of transition probabilities. For a state, the transition probabilities must be positive and they must sum to unity. The state model can be solved by state space approach predicting the probability of system ending in various states after specified time interval. The availability of such system can be evaluated as the sum of the probabilities of system ending in non-failure states. Several techniques for creating Markov models for the reliability analysis of system exist [30].

The states in Markov chain model reflects the operability of the modelled system in the same way as reliability graphs. The circle sign is used to model non-failing states of system, the square sign depicts the failure of system.

In Figure 2.15, the Markov chain model for system with three replicated modules and with repair done by bitstream scrubbing technique is presented [32]. The directed edges between the states are marked either with failure rate (λ) or repair rate (μ).

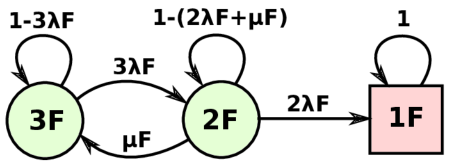


Figure 2.15: Markov chain for TMR system with bitstream scrubbing repair [32]

- *Monte Carlo* is a simulation method useful for modeling phenomena where uncertainty in inputs is significant. The properties of desired phenomena are determined by repeated sampling. Instead of studying few discrete scenarios, Monte Carlo method uses random sampling by chosen probability distribution function to create inputs and acquire many possible outputs which can be further analyzed. The reliability analysis of complex system based on Monte Carlo method was studied in [39].

2.4.6 Dependable system design

Several strategies can be adopted while designing the digital system to make it (more) dependable:

- *Fault prevention* is a set of techniques aiming at careful design based on approved development methodologies in order to prevent incorporation of faults into system. These methodologies typically contain processes such as design reviews, component screening and testing [60].

- *Fault forecasting* is a set of techniques focused on the estimation and prediction of system reliability to determine whether the effort to increase dependability of system will be needed. This includes the estimation of fault presence and the occurence of failures and their consequences in a system. To be able to do the estimation, the relation between the faults and failures has to be known, the reliability models have to be designed and applied to gathered failure data and analysed. It can lead to decision about making the system fault tolerant to meet the system dependability requirements.

- *Fault tolerance* is the ability of a system to continue to perform its specified tasks even if the fault is present. The main goal is the masking of fault what means that the error caused by the occured fault is not propagated though the system to its outputs. These techniques work in real time. As an example, the system with replicated unit and voting can serve. This strategy is studied more deeper in the next chapter.

- *Fault removal* is a set of techniques aiming at reducing the number of faults which are present in the system. This can be done in development phase by verification. The verification of design checks if the system meets the given conditions. If it does not, the fault causing the incorrectness has to be diagnosed and corrected. The fault can be removed also during the operation phase by maintanance in two ways. Preventive maintanance is based on removing possible damaged parts of system before the fault appears. When this approach is not used or it fails to avoid the fault occurence, the corrective maintanance is applied removing the fault at as short time as possible.

In this work, the fault tolerant design will be the main topic.

Chapter 3

Related areas of topic

In this chapter other approaches in the field of fault tolerant system design are presented.

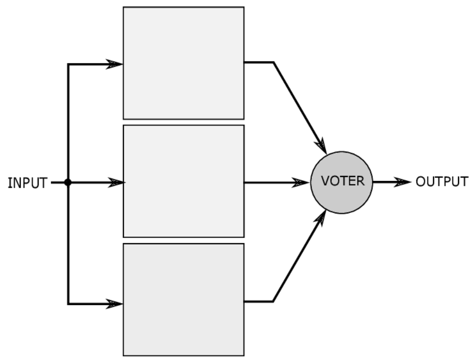
3.1 Fault tolerant systems

A fault tolerant system is that one which can perform its function and produce correct outputs even when it is affected by a hardware or software fault. Various conditions can exist that tell us whether the system is working correctly. In [25], three condition to state that system is fault tolerant are considered:

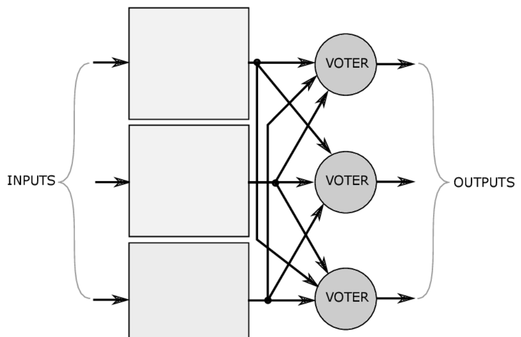
- The system computation for given dataset was not interrupted when a fault occured and complete batch of input data was processed.
- The outputs produced by the system are correct.
- The length of computational process did not exceed the predefined time limit.

In fault tolerant systems, the key goal is to prevent the errors from propagating to observable outputs of computation process. To achieve this behaviour, some kind of redundancy is a mandatory prerequisite. According to [27], the redundancy used for tolerating faults can be classified into four categories.

- Space (hardware) redundancy is probably the most frequently used one because of its easy use. Components of original systems are duplicated and special logic implementing some kind of voting strategy is added. This approach can be used for both fault detection and localization. This category will be discribed later in this text.
- Time redundancy is based on several repetitions of unit computation with the same input data. The interval between two repetitions can vary. This type of redundancy can be used to distinguish between the transient fault which can disappear after the execution of recovery process and permanent fault which persists over time. The input data used for several computational repetitions can be encoded in different ways to enhance the error detection capability.
- Information redundancy is based on the addition of information to the basic data structure. These additional data (referred to as control bits) can be used to check the validity of received data and can be used to restore the original data if maximal number of tolerable faults is not exceeded. From among the most popular approaches, the



(a) TMR scheme with one voter



(b) TMR scheme with tripled voter

Figure 3.1: TMR scheme with single and triplicated voter

single parity check, M-out-of-N coding, computing checksum and codes with residue and inverse-residue can be mentioned [71].

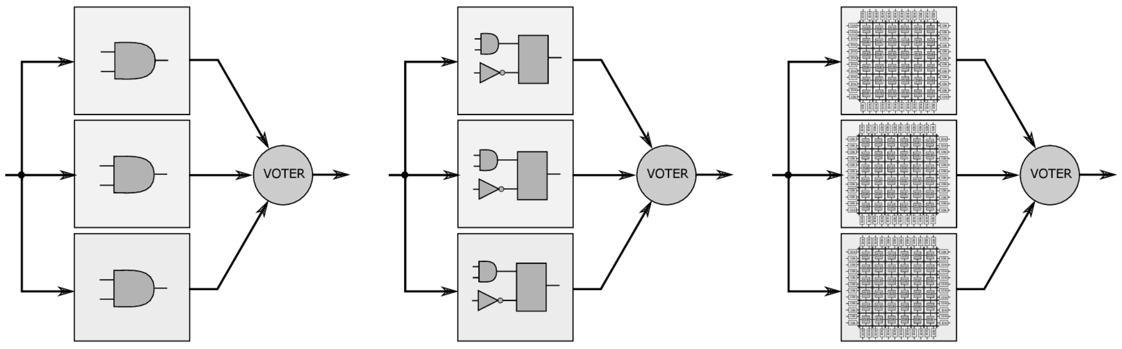
- Software redundancy is combined with the expansion of the use of microprocessors. It is based on the implementation of some additional software to provide FT features such as validity check of operational unit outputs or periodical self-test which exercises the hardware units repeatedly after a certain period.

Very often the fault tolerance techniques based on space redundancy are classified into two categories - static and dynamic.

3.1.1 Static fault tolerance

Because the size, energy consumption and price of hardware components is being decreased during the time, it is acceptable in most cases to increase the level of hardware redundancy to provide the possibility to detect and overcome one or more occurred faults. The majority voting mechanism is provided by n identical modules (n is always odd) and one or more voting units (voters).

The simplest example with 3 identical modules is called Triple Modular Redundancy (TMR) (see Figure 3.1). TMR is able to detect one affected module and tolerate its error output. Maximal number of tolerated faults for n-modular redundancy is $\frac{n-1}{2}$. The weak point of using this FT scheme is the voter which is not usually secured against the presence of fault. Several strategies exist how to make the probability of system failure smaller. The voter unit can be also replicated and thus have better chance to survive the fault occurence. This situation is shown in Figure 3.1b where all voters examine all results. All data paths remain triplicated in as many system units as possible. At the end of the chain of TMR architecture single result has to be reached. Another method for improving the voter resistance against faults is to implement it into more resistable fabric or by more reliable technology. Also the granularity of system plays an important role. When the coarse-grained strategy is used, the voter size is relatively much smaller and therefore the chance that fault will affect it is also much smaller. These TMR schemes with different granularity are shown in Figure 3.2a, Figure 3.2b and Figure 3.2c. It is clear that the relative probability of fault occurence in voter is biggest in case of single unit replication because of its size ratio.



(a) Unit replication (b) System replication (c) FPGA replication

Figure 3.2: TMR scheme with different level of granularity

3.1.2 Dynamic fault tolerance

Active replication methods are not based only on masking faults but they also provide fault detection, its localization and recovery from fault ability. The switching faulty components with spare ones or some kind of system reconfiguration can be used by this approach. The full operational capability of system can be brought back but a disruption in processing may be necessary during the reconfiguration process. Hot sparing techniques can be used to eliminate the disruption. In this case the spare unit is operating in parallel to original unit and it is ready everytime to substitute the unit affected by fault. The configuration with spare units operating in parallel together with n-to-1 switch is shown in Figure 3.3.

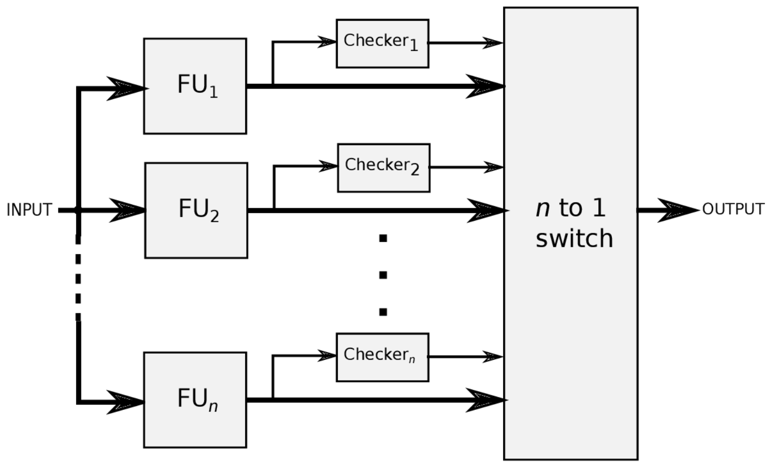


Figure 3.3: Active redundancy based on hot sparing

3.2 Fault detection and localization techniques

For fault detection, the capability of systems to mitigate faults which appear during their operation is an important feature. It should be signaled in some way to the supervising process running the application that a problem exists which needs to be solved. In most cases, the next step in mitigation process, the fault localization, is done together with fault detection. Although several fault mitigation techniques exist which do not need these two steps to be performed, as e.g. bitstream scrubbing (see section 3.3.1), the fault localization is a very important prerequisity for starting the fault recovery process.

The fault detection always requires some kind of redundancy. In last decades, many techniques were introduced and many of them are used till nowadays. Many of these techniques were based on increasing information redundancy. The main principle is to encode the given data vector to the form which will allow error detection, its localization or even its mitigation by some control logic. These techniques are still frequently used mainly during data transfers between two logic devices in non-reliable environment.

From among the most widely used techniques based on information redundancy, the parity code, cyclic redundancy check and checksum methods can be mentioned.

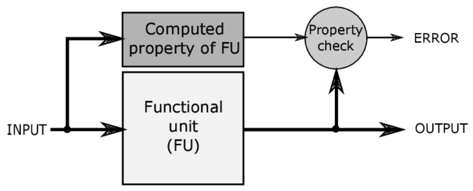
- *The parity code* is based on adding extra information, mostly a single bit, to each input data vector. The value of this bit is determined to respect the required parity for the resulting code. When even parity is used the number of logical ones in code has to be even. For the odd parity the code has to contain the odd number of logical ones. This technique is able to detect the odd number of errors in code. When the even number of errors appear the error detection is impossible and the received code is considered as non-faulty.

- *The checksum* can be counted by some aggregation function both on side of transmitter and the receiver and then these two values are compared. If they disagree the error is present. The bit size of the checksum when compared with data vector has to be bigger because of the carry bit which can appear when the sum is counted.

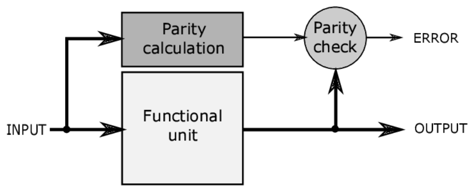
- *CRC* is the method which uses the generator polynomial to count the check code for the input data vector. It is the enhancement of the checksum counting based on cyclic codes. CRC can be easily implemented by blocks with XOR logic function. The code word consists of original data vector and the counted code. CRC is based on dividing the original data vector by the generator polynomial in a polynomial long division. The remainder of the computation becomes the result. The polynomial coefficients are calculated by a finite field arithmetics and thus the addition operation can be performed bitwise-parallel. The ability to detect errors depends on the proper choice of the generator polynomial. [40]

When the information redundancy is increased the Hamming distance between the data word and encoded word can be counted and can be used to find the detection parameters of used method such as maximum number of errors which can be detected or repaired. The codes with the ability to detect errors are reffered to as Error Detection Codes (EDCs). If they can also repair the error, they are reffered to as Error Correction Codes (ECCs). Fault detection and localization methods can be also based on space redundancy. These are also very popular widely used when dealing with FPGAs. The main techniques from this group can be roughly divided into three categories:

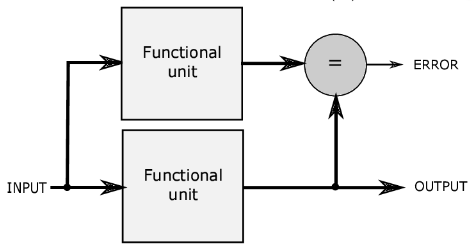
- Methods based on replication and concurrent error detection
- Methods for off-line testing
- Methods based on roving areas in FPGA



(a) Checking function property



(b) Checking computed parity



(c) Comparing output of replicated FUs

Figure 3.4: CED based on different techniques

3.2.1 Replication and concurrent error detection

These methods are based on the fact that for fault detection some kind of space redundancy is needed.

The simplest form is n-modular redundancy with n replicated modules connected in parallel. Their outputs are compared and any difference indicates the presence of fault. As an example, TMR system can be mentioned (see Figure 3.4c). The redundancy of this system is always more then $(n-1)*100\%$. As its benefit, the ability to detect every distinct error can be seen. The problem can arise only when the same error will be produced by all replicated units.

In many solutions, the technique of Concurrent Error Detection (CED) is adopted because of the minimization of performance impact. Parallel to operational unit, there is a unit computing some function as can be seen in Figure 3.4a. Its output can be continuously used to check the correctness of the output. When the outputs of the units are not equal, it means there is a fault in the system which caused an error to occur. The drawback of this approach is that we cannot distinguish whether the error is produced by operational unit or its concurrent unit. One of the most popular CED techniques is the computation of parity code (see Figure 3.4b).

If the functional unit implements bijective function, the unit implementing inversion function can be added to compute back the input vector value and compare it with the original one. This approach can be seen in Figure 3.5. This is a more reliable technique than replication of functional units because there is no risk that the same fault will appear in more replicated units at the same time. Another drawback besides the limitation that the inversion function cannot be created for non-bijective function is the delay extension of resulting system when compared with the original system. This is caused by the fact that the new system is consisting of two units, the first is implementing the desired function and the second its inversion, and they are connected in series.

In common, the methods based on redundancy and CED provide a very fast solution of fault detection, as the fault can be detected immediately or very early after the functional unit finishes the computation. There is just the latency of voting, parity or similar logic. Besides the extension of space overhead, their drawback is also the resolution of detecting

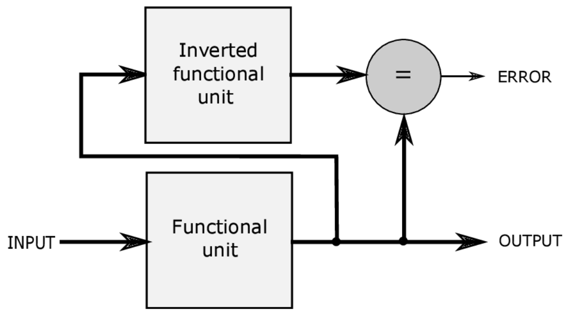


Figure 3.5: Fault detection based on the use of function inversion

and localizing the fault in system. Only entire replicated unit can be localized, not particular affected resource inside this unit.

3.2.2 Off-line testing

Off-line fault detection is a widely used technique which is checking the fault occurrence while the application in the FPGA is not running. This method can be based on external testing equipment outside FPGA or the test equipment can be configured into FPGA. The second approach is known as Built-In Self-Test (BIST). For configuring internal test equipment the complete reconfiguration of FPGA is frequently used.

The offline testing methods for FPGA can be divided into two categories according to their relation to currently configured application:

- The objective of application-independent testing is to check as great number of possible configurations of programmable logic components and their connections as possible. No information about application which will be run in FPGA is available. This type of testing is very often employed by chip manufacturers. The main drawback of this approach is low efficiency of detection of timing-related faults because during this testing it is not possible to examine all possible interconnection patterns which can be designed by user.

- The objective of application-dependent testing is only to test the resources used by a specific application implemented in the FPGA. This is done by decomposing the user designed system into blocks and exercise each block by using the another ones. These blocks have to provide the data path between tested block and the external interface of FPGA in case of using external test equipment or they have to implement the test equipment by themselves. As the configuration memory has low dependence on the structure of the FPGA, this testing scheme can be often used for the same design in different FPGAs. To implement reliable BIST is more difficult than the use of the external testing equipment because the test equipment such as test vector generators and response analyzers are not implemented in fault-free fabric.

Firstly, BIST technique for FPGAs was introduced for testing PLBs [58], and then it was extended for testing programmable interconnect [57]. It uses several test configurations which consist of three main blocks - Test Pattern Generator (TPG), Block Under Test (BUT) and Output Response Analyzer (ORA). The main principle of BIST methods is that one part of the FPGA is configured to be under test (BUT) and the other parts are configured to generate testing vectors for it (TPG) and to analyze the results (ORA). When the testing is finished, the resources of the FPGA change roles and the other part of FPGA is tested. Thus, in several steps the entire FPGA can be fully tested. This technique can be seen in Figure 3.6. The start of BIST can be triggered either during the system start-up, as part of a maintenance schedule or in response to detected error.

The main advantage of BIST in comparison with other fault detection techniques is that there is no impact on FPGA during the normal operation. Just the storage for test configurations which are not typically very big must be provided. After the manufacturing phase, testability can be achieved without any cost, because the BIST can be configured into the FPGA at the beginning and then reconfigured by the desired design. When BIST is complete, an FPGA needs to be reconfigured for its normal operation. For BIST in FPGA, the only cost is the additional memory required for the BIST configurations which are not typically very big. Another benefit of BIST is the fact that it allows the complete coverage of the FPGA fabric [16].

There are also several drawbacks of BIST technique. One of them is its limitation that it can only detect faults during the test mode when the FPGA is not operating. Thus, some timing-dependent faults or similar may not be detected.

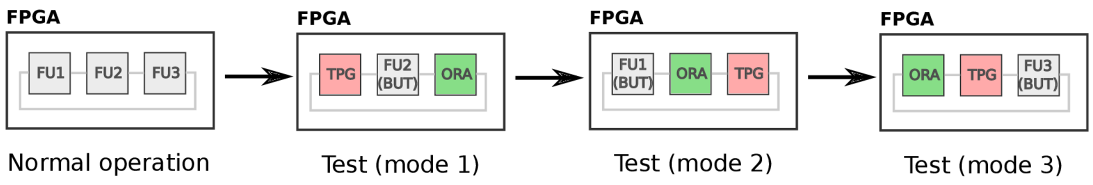


Figure 3.6: Testing of entire FPGA using BIST and PDR

3.2.3 Bitstream readback

Present FPGAs offer the possibility of reading the actual contents of their configuration memories as well as the contents of flip-flops in CLBs in form of configuration bitstream. This process can be considered as the inversion of FPGA configuration. To perform bitstream readback, external controller outside FPGA under test is needed which uses some FPGA configuration interface. Bitstream readback is available in two modes - readback verify and readback capture [69].

In readback verify mode, the controller reads the configuration of memory cells. This mode is mainly used to verify the success of previously done configuration. It is performed by comparing the original bitstream used for configuration and the bitstream read back from FPGA.

Readback capture mode also reads configuration memory cells data but in addition to that it also acquires the current states of all internal flip-flops inside CLBs and the state of IOBs. With these gained data from FPGA and the knowledge of data which are expected to be in the configuration memory of FPGA and other resources in the moment of readback, the diagnosis algorithms can be used to detect and localize faults in FPGA [52] [53]. Possible test system based on bitstream readback is shown in Figure 3.7.

3.2.4 Roving STAR

This technique capable to detect and localize faults in FPGA is based on the dividing the array into tiles with the same number of resources and their structure. Some techniques for off-line testing are then used on tiles which are not used by the current design for performing

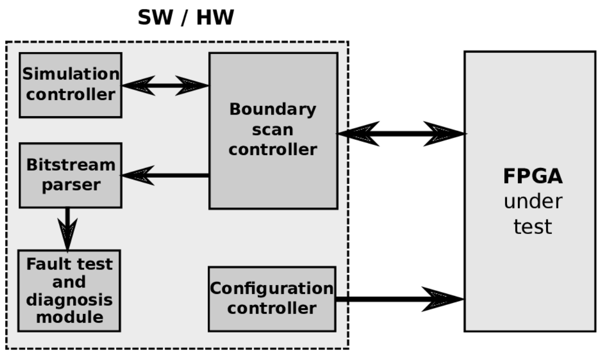


Figure 3.7: Test system of FPGA based on bitstream readback [53]

the function. When the testing is finished the partial reconfiguration of FPGA is used to change the layout of design and another tiles previously used to implement the function are configured as BIST and tested. This technique and its extensions have been described in many publications [3] [2]). The tiles of FPGA which are currently implemented by BIST and which are performing the fault detection and localization are referred to as Self-Testing AReas (STAR). While one STAR is tested off-line the remaining blocks of system which are not utilizing resources from actual STAR continue in run and the aplication in FPGA is not interrupted. Testing is focused on logic blocks and connecting wires. If fault is present in the implemented circuit, the greatest possible latency of its detection will be equal to time needed to test a number of blocks STAR throughout the FPGA.

Figure 3.8 shows how the circuit is partitioned between the STAR areas and areas where the desired function of the design is performed. In the first version of this technique, only 1-dimensional STAR areas were implemented. Nowadays, typically 2-dimensional areas are used - vertical STAR (V-STAR) and horizontal (H-STAR). Thus, the working area may be contiguous or it may be divided into two or four disjoint regions. All horizontal wire segments in H-STAR and all vertical segments in V-STAR are reserved for testing. The interconnection signals connecting separated parts of working areas can pass STARs. But only horizontal wires can go through H-STAR and only vertical wires through V-STAR. A STAR tests both the PLBs and the programmable interconnect within its area. The size of the STAR depends on the number of resources needed to implement BIST circuit which will be used for testing. The testing of PLBs is performed in cells of FPGA in the intersection of H-STAR and V-STAR. They are tested using off-line BIST which was configured to this tile by partial dynamic reconfiguration. When testing interconnection wires, the whole length of H-STAR is used to test horizontal wires and length of V-STAR to test vertical wires. At the same time when STARs are testing logic blocks and interconnection wires the working area of FPGA continues in work without interruption. After the testing of a STAR has been completed, the STARs are reconfigured back to the original function and testing areas are moved to the following area which was previously used as a part of working area.

The fault coverage of this approach can be 100% because every tile of FPGA is tested. The maximal latency of fault detection for the worst case can be counted by multiplying the number of possible STAR locations and the time needed to complete the tests in STARs (plus time for the reconfiguration of STARs to new location). Hardware overhead of this approach is formed by tiles needed for STARs and the reconfiguration controller logic which is used for roving the STAR through the FPGA. The fault localization of this method is scalable depending on the size of the FPGA. The space overhead for FPGA divided into tiles with n columns and n rows can be counted by formula $\frac{1-(n-2)^2}{n}$ [2], which is usually less than for techniques based on unit replication.

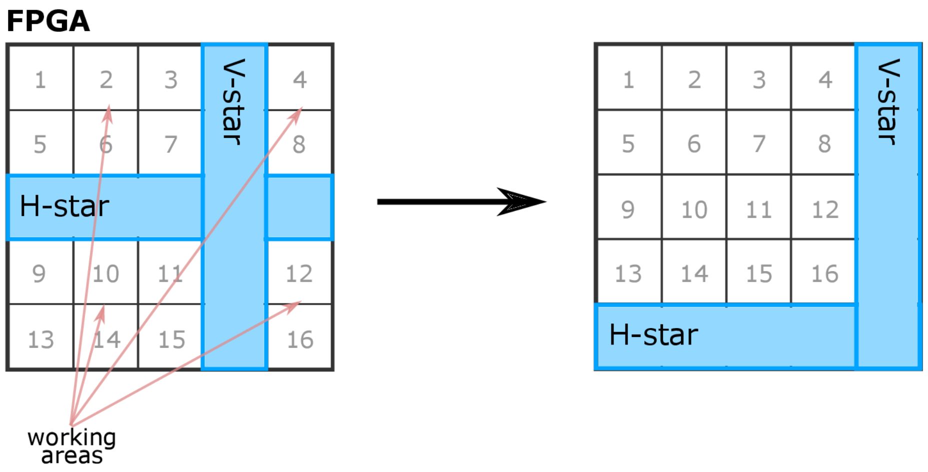


Figure 3.8: Configurations applied by roving star technique

3.2.5 Summary

All presented fault detection methods have both positive and negative features. Table 3.1 is showing the results of detection methods when they are evaluated by several criteria. Granularity criterion means how specific the method can be when detecting an error. Space overhead shows how much additional resources will be needed when the method will be applied. The additional latency of computing caused just by the fault detection is evaluated by criterion named performance overhead. Detection speed is evaluating the time needed from fault occurence to its detection. Finally, criterion fault coverage shows how many faults from all possible faults can be detected by the method.

Detection method	**Granularity**	**Detection speed**	**Fault coverage**
Unit replication	**coarse** only module can be detected	**fast** with error occurence	**good** all error occurences
CED	**coarse** only module can be detected	**fast** with error occurence	**medium** can be impractical for some functional units
Off-line methods	**fine** exact error can be detected	**slow** during periodical test	**very good** can detect also faults not manifested by error
Bitstream readback	**fine** exact error can be detected	**slow** after bitstream is read, parsed and analysed	**very good** can detect also faults not manifested by error
Roving star	**fine** exact error can be detected	**medium** after STAR moves to faulty tile	**very good** can detect also faults not manifested by error

Detection method	Space overhead	Performance overhead
Unit replication	**large** resources for n-1 modules + voter needed	**small** voting logic latency
CED	**medium** trade-off with coverage	**small** additional latency of checker unit
Off-line methods	**small** testing controller	**small** just start-up delay
Bitstream readback	**small** readback and testing controller	**small** just start-up delay
Roving star	**medium** resources for STARs + testing controller	**large** latency of switching tiles, can cause long critical paths

Table 3.1: The comparison of fault detection methods

3.3 Transient fault mitigation

The use of FPGAs in harsh conditions has significantly risen the number of transient faults mainly caused by ionization radiation. These faults can be mitigated but this requires additional logic.

The susceptibility to these kinds of faults can be lowerred by the special fabrication design to produce radiation-hardened FPGAs. This radiation hardened design is based on protecting the configuration cells at transistor or silicon level. As FPGAs become more and more complex with large number of resources and processing capabilities, the radiation-hardenning becomes excessively expensive in comparison with non-protected ones. Radiation hardened FPGA has slower operating frequency and increased power consumption when compared with its commercial off-the-shelf FPGA counterpart [41].

When a transient fault occurs in FPGA it can be repaired by reconfiguration of affected part of configuration memory. This can be done by complete (static) reconfiguration of FPGA or by PDR of affected reconfigurable region. Static reconfiguration causes the stopping of running design in FPGA and possible loss of current status information of implemented modules. Due to complete reconfiguration of FPGA, this technique does not require the localization of the affected part of FPGA. Nowadays in most cases, the application running in FPGA cannot be stopped during the recovery process and therefore techniques based on PDR are preffered.

3.3.1 Configuration bitstream scrubbing

Configuration bitstream scrubbing was introduced to correct configuration memory after SEU occurences. This method is based on periodical reconfiguration of PRM by correct partial bitstream while the FPGA is in operation. The scrubbing approach is typically combined with some fault tolerant technique which ensures that the implemented application can stay operating properly even during and after scrubbing process.

There are two common configuration scrubbing strategies:

- *Blind scrubbing* stategy is based on periodical reconfiguration of PRM by golden copy of designated partial configuration bitstream. The reconfigurated region is not searched for the SEU occurences and reconfiguration is done also in case when PRM is working correctly. This method is easy and fast to implement due to its simplicity. The drawback of it is its waste of processing time because it performs scrubbing also in times when it is not needed.

- *Scrubbing with configuration readback* stategy is based on the use of read bitstream to detect SEUs and on reconfiguring the PRM only in case when it is affected by SEUs. According to the type of error, the detection and the source of configuration data for reconfiguration, three variations of configuration scrubbing with readback can be identified.
  - *Readback and reconfiguration with golden copy of configuration bitstream* - this method uses the comparison between the read bitstream data and the golden copy from external memory to detect and optionally localize the errors.
  - *Readback with ECC check and reconfiguration with golden copy configuration bitstream* - this method is based on error detection by parsed ECC from read bitstream and reconfiguration by golden copy of bitstream.
  - *Readback with ECC check and reconfiguration with corrected configuration bitstream* - this method uses the read bitstream where the error is corrected by the ECC technique. This option can be used only when no more than maximum number of faults is present. On the other hand, this approach eliminates the need of external memory with golden copies of partial bitstreams.

Configuration scrubbers can be also divided into two groups according to their implementation:

- *Internal scrubbers* are implemented inside the FPGA where the scrubbing is performed. They are also reffered to as self-scrubbers. They have to use the internal configuration interface as the access to configuration memory. This solution does not require any external device to host the scrubber but the recovery process is limited. The internal scrubber cannot reconfigure the reconfiguration region where it is implemented or where the storage for golden copies of partial configuration bitstreams or its memory driver is placed.
- *External scrubbers* are implemented in different devices and they access the configuration memory of FPGA by external configuration interface such as JTAG or SelectMAP. With this approach, each part of configuration can be scrubbed. The drawbacks are: 1) increased power consumption, and 2) lower possible frequency due to fact that external scrubbers are connected by longer wire connections than internal scrubbers.

The main drawback of configuration scrubbing method is the need of continual use of the configuration port. This prevents other forms of configuration port using such as its utilization for dynamic design change by partial reconfiguration. In [20], the technique to overcome this issue via integrating partial dynamic reconfiguration into configuration scrubbing is presented.

The configuration scrubbing works only with SEUs causing a bitstream corruption, it is unable to determine if a SEU has occurred in the memory used by implemented design for computation. The scrubbing period should be stated according to failure rate of system. This period implies a fault detection latency.

3.3.2 Recovery by partial dynamic reconfiguration

Methods based on partial dynamic reconfiguration are dependent on some kind of detection and localization technique implemented in design which in case of fault detection triggers the process of recovery. Unlike in the configuration scrubbing, this process is started only in case of fault detection event. The process of recovery is shown in Figure 3.9. The detection and localization of faulty module is typically done by the design itself implementing techniques such as CED or unit replication with checker units. The error signals are supplied by some kind of PDR controller which will trigger the reconfiguration process using appropriate configuration bitstream downloaded from configuration bitstream storage.

PDR controller can be implemented in the same FPGA where it performs the mitigation process or it can be implemented in external reliable fabric. This controller is often implemented in FPGA as an IP core processor such as MicroBlaze or LEON. It can be also implemented as dedicated module and optimized to achieve better performance and lower space overhead.

In [9], mitigation technique able to cope with the SEU effects is presented. Errors in PRMs are detected by CED technique and they are mitigated by PDR. The effects of SEU in configuration memory and also in user SRAM memory can be mitigated.

In [10], the design with enhanced TMR implementation providing error localization is used and the faulty module is reconfigured by golden copy of configuration bistream. The implemented TMR scheme ensures the fault tolerance of the system.

The scheme with TMR and utilization of PDR to mitigate the effects of SEUs can be also applied to entire softcore processor [23].

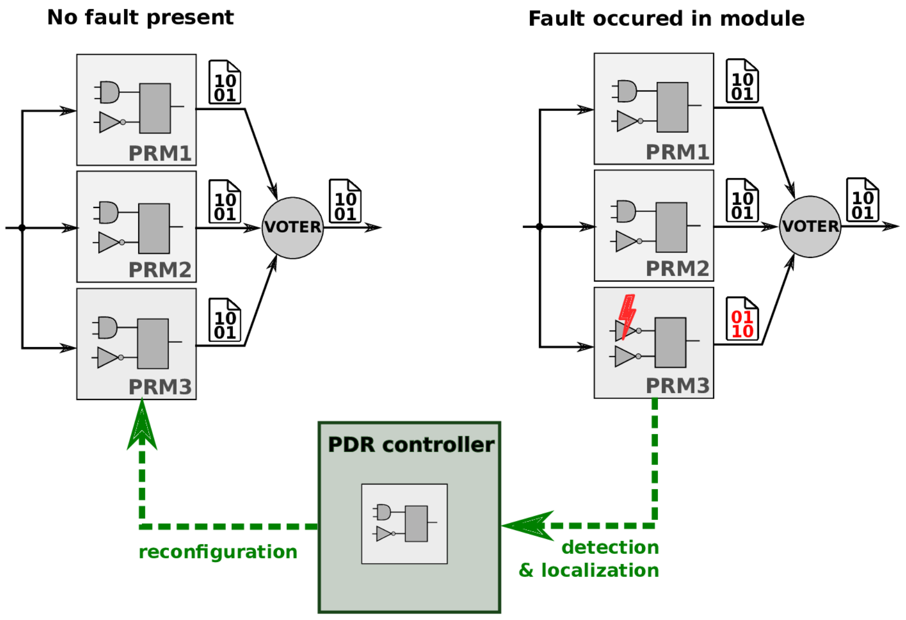


Figure 3.9: Using PDR to recover system after SEU occurence

3.4 Techniques for system recovery after permanent fault occurence

In this work, as the permanent fault is considered, each fault causes a damage of FPGA resource in that way that it cannot be used in FPGA design anymore. This happens mostly by outside damaging or during the wear-out phase of FPGA or by the impact of harsh environment on FPGA. The recovery of this fault can be done on various levels:

- Hardware level recovery - the FPGA fabric can be designed and manufactured with spare resources which can be utilized in case of fatal fault occurence in currently used set of resources. This approach for array based resources (i.e. CLBs) is based on using multiplexers or other switching logic at the ends of lines of cells. This allows the remapping of a row or a column with damaged component into some spare row or column [19]. For hardenning the interconnection of CLBs and other hard blocks in FPGA, the fine-grain redundancy in the interconnect blocks can be introduced [72].

  All hardware level recovery approaches based on hardware switching of connected blocks or connection routing guarantees that the recovery is transparent to the configuration. This means that the current design loaded in configuration memory can be used further. Also the timing performance of the design before the permanent fault occurence will be not changed as the physical layout and connections of resources are replaced with the new ones utilizing the spare resources with predefined timings.

  The drawback of the approach is in the limited number of faults which can occur in distinct rows or columns in the FPGA. These spare rows or columns create the necessary area overhead. When all these spare lines of resources are utilized another approach has to be used.

- Configuration level recovery - the damaged resource causing permanent fault is still accesible for reconfiguration controller but it is excluded from the further utilization. This approach needs the existence of reconfiguration controller which will control the mitigation process. From the most known techniques, the use of precompiled alternative configurations, evolutionary algorithms and incremental mapping and routing of design can be mentioned.

- Application level recovery - this category incorporates all higher level aproaches where the fault is mitigated without using different physical resources or other design configuration. These approaches can be commonly used and they are not limited only to FPGAs. The fault can be overcome by the system design which can be able to use spare functional block instead of the faulty one without the need to modify design configuration or continue in computation with some performance degradation [70].

Since hardware level recovery is dependent on the manufacturer design of each single type of FPGA and the application level recovery is a wide problem not focused only to FPGA, this thesis is aiming at implementing the approaches manipulating with FPGA configuration to mitigate the occured fault. From these approaches, three categories can be identified - incremental rerouting, the use of alternative configurations and evolutionary algorithms.

3.4.1 Incremental mapping and design routing

One possible approach to deal with permanent fault occurence is the modification of current design configuration to exclude the affected resources when a fault occurs. The incremental change of configuration may consist of several steps: re-mapping, re-rerouting and bitstream generation. Some methods using this approach do not require to perform all these steps. This approach can in theory utilize all spare resources which are currently not used by implemented application for logic or routing affected by faults. It can also handle occured faults in various patterns in FPGA where other approaches not using online change of configuration will fail. The drawback of this method is the need of adaption of FPGA mapping, placement and routing tools to operate autonomously with considering existing faults in implementation area. The incremental change of design requires not negligible time for processing, it can increase power consumption and area overhead.

The incremental change of design can be computed remotely and then downloaded to FPGA or it can be performed by tools implemented in the device which is under repair.

One of possible fault recovery methods mentioned in [37] focus on swapping faulty resource to non-utilized one within single logical block. This repair has typically only small impact on the global routing or other resources outside the block. It is not always possible to use this approach.

When the logic block is faulty and it needs to be replaced by a spare one, the pebble shifting technique can be used [45]. The allocated blocks are shifted according to cost of shifting move. The basic principle is shown in Figure 3.10.

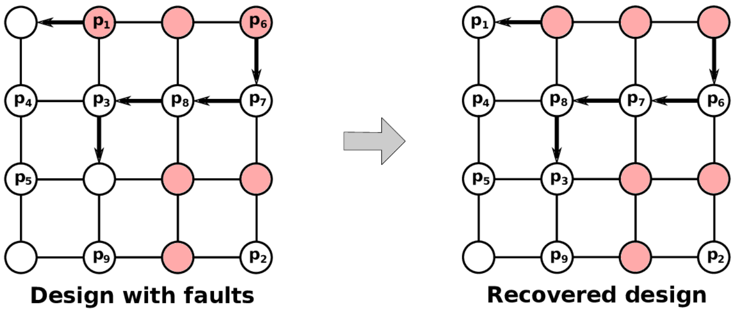


Figure 3.10: Pebble shifting approach

In [47], the approach consisting of the instantiation of logic and routing resources at run-time according to application data-flow graph and constraints, such as fault regions that cannot be allocated, is presented. It uses implementation tools run on the host system. The basic principles of this approach are shown in Figure 3.11 and Figure 3.12.

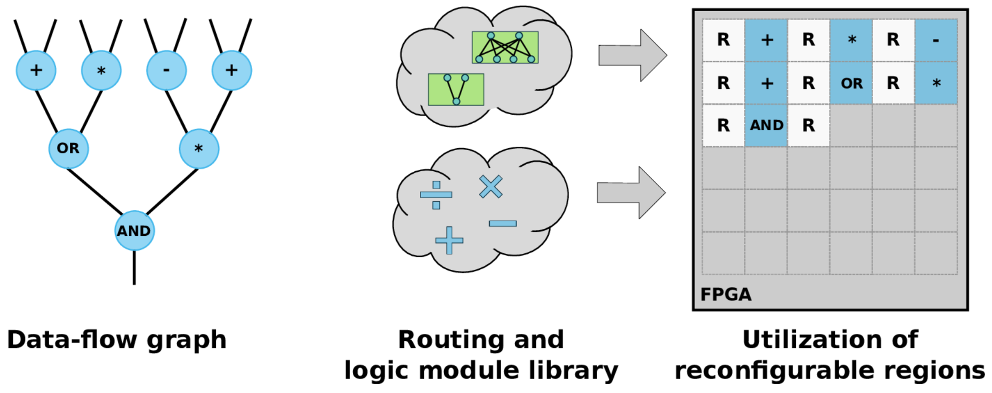

+
*
-
+
OR
*
AND
R
+
R
*
R
-
R
+
R
OR
R
*
R
AND
R
FPGA
Data-flow graph
Routing and logic module library
Utilization of reconfigurable regions

Figure 3.11: Resource instantiation in FPGA without faults [47]

3.4.2 Approach based on precompiled alternate configurations

While the previous approach is designed to be able to solve the fault presence in system after it really occurs, some other approaches are trying to prepare the possible solutions before the fault appears, in the design phase. The basic requirement is to exclude the damaged resource from the further use. There is a possibility to divide the complete

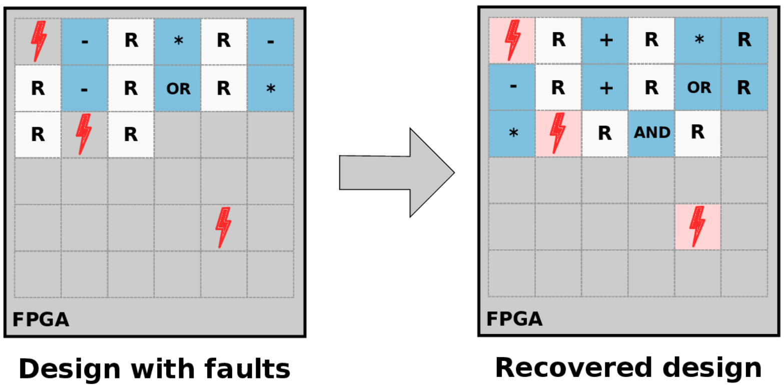


Figure 3.12: Resource instantiation considering faults[47]

implementation space in some reconfigurable region into several tiles, split the desired design into modules which will be configured into different tiles leaving one or more tiles unused. The configurations with these alternative implementations are precompiled and created partial configuration bitstreams are stored in some type of memory. The basic principle of generating alternative configurations implementing different module layouts is shown in Figure 3.13. When a fault is detected and localized in some tile, the reconfiguration of the entire design is performed with this precompiled configuration which does not utilize the resources from this tile. Since each configuration contains the implementation of the same function and the interface between the entire reconfigurable module and the rest of design is fixed and the same in all cases, all partial bitstreams are interchangeable and can be configured to this partial reconfigurable region. With this approach, a fault in logic block and in local interconnections can be handled. Recovering from faults to global and overlapped segmented interconnect is discussed in [36].

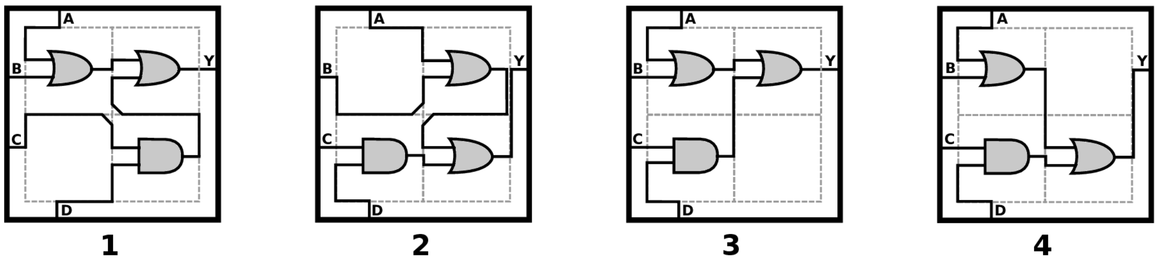


Figure 3.13: The alternative layouts of sample system with 4 tiles

This technique minimizes the recovery time since the process consists of alternative configuration selection and PDR with its precompiled bitstream.

One of the first implementations of this approach is presented in [36]. The FPGA implementation area is segmented into static tiles at design phase.

In [73], the method using alternative configuration and degradable design to recover from multiple permanent faults is presented. When there are no fault-free tiles available in the design, the design is degraded in terms of its fault tolerance to achieve lower number of utilized tiles.

The drawbacks of this approach can be seen in its poor area efficiency and complicated mitigation of mutliple faults but the main one is the requirement of external storage for precompiled partial configuration bitstreams. This can be reduced by some techniques such as bitstream compression but there is always a trade-off with increased time and complexity of recovery process.

3.4.3 Evolutionary algorithms

The ability of modern FPGAs to be reconfigurated dynamically can be used by evolutionary methods. They can recover the system correct operation through evolution when faults occur. These methods offer a large degree of flexibility in the number and distribution of faults which can be mitigated. There is no need to precisselly localize the fault. Evolutionary methods attempt to facilitate repair through the reuse of damaged resources. The fitness function of implemented Genetic Algorithm (GA) is able to internally evaluate the residual functionality of the design in FPGA and assess the fitness value. This value is used for the upcoming selection phase.

In [14], the fault recovery method referred to as competitive runtime reconfiguration with competing two half-configurations (left and right) gradually modified by GA is used.

The drawback of this method is complexity and flexibility what can result in very time-demanding search of satisfactory design with unpredictable duration and its result. The logic for evolutionary algorithms can cause unnegligble area overhead.

3.4.4 Summary

Although many different approaches to system recovery after permanent fault occurence exist, none of them is considered as universally applicable. Frequently used method based on spare resources or predefined alternatives are demanding device resources, other methods with lower requirements can have problem with speed of recovery process or can cause performance overhead.

Table 3.2 is showing the comparison of recovery methods from different aspects. Transparentness to configuration means the fact that recovery is done in device by itself and user does not have to change the configuration in FPGA. Recovery speed determines the required time for permanent fault recovery. Resource overhead shows how much additional resources (physical or logical/configurable) will be needed when the method is applied. Performance overhead describes the additional latency of recovered system. Finally, criterion flexibility of recovery highlights how flexible and effective these methods are.

Recovery method	**Transparent to user design**	**Recovery speed**	**Resource overhead**
Hardware level	**yes**	**very fast** just switching lines in hardware	**low** spare physical resources needed, no requirements for implementation space
Alternative configurations	**no**	**medium** configuration selection and reconfiguration delay	**very high** reconfiguration controller and configuration selector needed & storage for configurations
Incremental remapping & rerouting	**no**	**poor** time demanding remapping and rerouting	**high** design implementation and reconfiguration controller
Evolutionary algorithms	**no**	**poor** may take long time to evolve	**high** after bitstream is read, parsed and analysed

Recovery method	Performance overhead	Flexibility of recovery
Hardware level	**low** no design change	**low** big loss of usefull physical resources while switching from affected row/column to spare one
Alternative configurations	**low** alternative configuration can be optimized	**medium** the loss of usefull configurable resources when excluding the affected part, trade-off with the number of configurations (depends on the granularity of partitioning)
Incremental remapping & rerouting	**medium** trade-off with implementation controller complexity	**high** non-faulty resources can be effectively utilized
Evolutionary algorithms	**low** can be optimized by setting fitness function	**high** non-faulty resources can be effectively utilized

Table 3.2: The comparison of permanent fault recovery methods

3.5 Fault injection techniques

With the introduction of new fault detection, localization and mitigation techniques, effective techniques and methods for debugging and verification of the correctness of their fault tolerant design and their performance is needed. This is done by purposely injection of faults into implemented design or emulation of its impact. Fault injection can be done on different levels in FPGA. The fault occurence can be emulated by implemented software (e.g. by simulating the altering of inputs of system components) or it can be injected as a bit flip of configuration memory cell on hardware level.

In FPGA, the fault emulation in application is not sufficient since it is only able to test the faults that occur on the design level rather than the hardware level such as faults in routing. The faults also cannot be emulated in locations of FPGA not utilized by design.

The fault injection on the lowest hardware level can be done by the exposure of FPGA with implemented system design to radiation with heavy ion beams. This is the most realistic way of injecting faults since it simulates the physical phenomemon occuring in real application. The drawback of this approach is its price and time demands when compared with other approaches. Another important disadvantage is the fact that it is not possible to precisely inject a fault into desired destination such as to flip only single bit in configuration memory [22].

Another approach aims at manipulation of configuration bitstream loaded in FPGA. The scheme of this approach is shown in Figure 3.14. The correct configuration bitstream is altered with a single or multiple bit flips driven by some injection controller (injector). Typically, the injector reads a correct configuration bitstream of the FPGA and creates a bit flip in the stream at the desired bit in the bitstream. Due to fact, that the injection of every possible bit flip can be very time demanding (up to several tens of minutes [18]) and it can cause cause peformance bottleneck in FPGA, the FPGA configuration bistream can be divided to the important bits of design and *don't care* bits. The list of important bits to be flipped can be generated by some software tool incorporating the knowledge of relation between the utilized resources and their real position inside configuration bitstream. The manipulated configuration bitstreams are one by one loaded via PDR into FPGA and the simulation is done by providing input stimuli. The outputs of the system to each configuration with specified placement of fault can be gathered and compared with the outputs of the correctly operating system (the golden copy of system configuration). The sensitive bits can be coupled with the parts of system design which they implement and reliability parameters of these parts and entire system can be evaluated. The fault injector can be implemented as external tool and utilizing external configuration interface of FPGA such as JTAG or SelectMAP [18] or it can be instantiated in FPGA which is beeing injected using the ICAP interface [12].

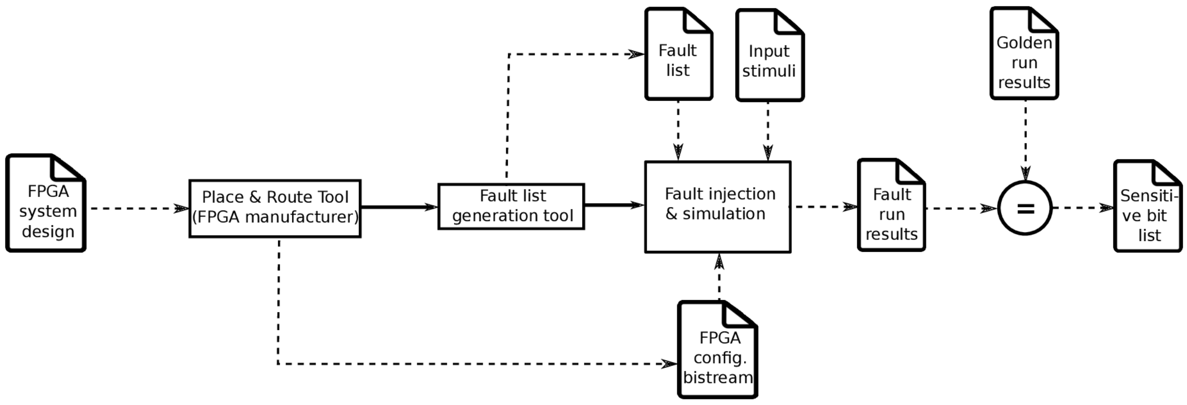


Figure 3.14: The scheme of fault injection approach based on configuration modification

In [74], the fault injection combined with fault emulation to reduce the performance bottleneck caused by injecting faults by frequent reconfiguration of FPGA is presented.

Chapter 4

Motivation and goals of the research

Previous chapters presented the topics of fault tolerant system design in FPGA and some open issues were mentioned. This information can be seen as the key inputs of this thesis.

4.1 Motivation

The scaling of electronic devices and still less robustness of components bring the strong need for more complex securing against the occurence of faults. The use of electronic devices in new rough and noisy environment is also another source of problems. For example, in the aerospace industry there are requirements on electronic devices for their resilience against radiation and on hardenning them against negative effects of material aging during long term missions.

In recent decades, new possibilities and new challenges in the area of system design appeared. Programmable electronic devices such as CPLDs and FPGAs allowed rapid prototyping and started the era of reconfigurable computing. Faulty design can be easily fixed after the first deployment and the same hardware can be also used to perform various tasks during the lifetime where some of these can be unforeseen. The FPGAs came up with new possibilities in the field of fault tolerant hardware design. The dynamic reconfiguration can be now used for changing the mapping and routing inside FPGA in order to mitigate the faults which have occurred. The new challenges with fault tolerance in FPGAs are connected with their configuration saving. Very often the FPGAs which have configuration stored in SRAM memory are used [28]. They are popular because of their lower price and easy use they offer. Higher susceptibity to SEU faults in comparison with other FPGA types can be seen as their drawback.

Many approaches for making digital systems more dependable were presented. Fault tolerant system design offers the possibility to overcome the impact of fault occurence while the use of detection and localization methods together with fault mitigation based on PDR can offer to restore the fully operational state of system. This can be done autonomously without the need of user intervention and without stopping system operation. Nowadays, the utilization of FPGAs is not only in rapid prototyping but they are used frequently also in long term missions. Thus, the study of system dependability has to focus also on permanent faults which occur more likely with the increasing age of FPGA. Many techniques for mitigation of SEU effects in FPGA and also several mitigation techniques for permanent damage of resources in FPGA are available. None of them is universally applicable due to their high demands on memory (e.g. precompiled alternative configurations), time-demanding fault recovery (e.g. evolutionary algorithms), area overhead (e.g. incremental change of design), etc. Thus, it makes sense to focus on optimization of these techniques and creating such methodology which will describe how to create design with effective fault recovery ability.

4.2 Goals of the research

In various applications of fault tolerant systems, different levels of dependability are required to be achieved by the implementation. It can be stated that the range of hardware to which the failing design can be implemented plays an important and maybe even decisive role. This holds especially for longtime missions in which it is important the hardware to operate correctly for long periods (in terms of years).

The effort to develop a methodology for fault tolerant systems design was driven by the goal to satisfy the following aspects.

- The localization of the FPGA part (PRM) affected by fault.
- The determination of the fault type and its classification according to considered fault model.
- The driving of repair proccess to return the system
  - to the exactly same state as there was before - in case of transient fault,
  - to the state when the functions of system are producing correct outputs - in case of permanent fault.
- Keeping the design running during the reconfiguration process if it is possible.
- Enabling the support for synchronization process after reconfiguration is completed.
- The effort to shrink the number of the FPGA resources needed as hardware overhead because of the system design according to proposed methology.

The goal of this thesis is to combine the existing well known techniques together with new approaches. As an example, the CED technique together with online checkers can be used not only to ensure the fault tolerance in system but also to localize the module affected by a fault in FPGA if it is possible. This localization information will point at specific reconfigurable module of FPGA which is faulty. Then some reconfiguration controller will use this information to process fault mitigation in it.

The key goal of the research is to develop this specific controller which will control the reconfiguration of FT architecture when the fault will be localized in it. The process of fault mitigation will vary for different types of faults specified in the fault model. The determination of the type will be also the task for reconfiguration controller and it will be based on the knowledge of the history of the reconfigurable module errors.

The fault mitigation process will be limited in such way that it can use only the resources of FPGA which were reserved for the implementation at the beginning of system lifetime. Before the first configuration of system into FPGA the implementation area will be defined by the user. During the system lifetime the permanent fault can be detected and localized and then the need of resource mapping change can arise. Since the limitation of FT system implementation to the allocated area is needed, the new resource mapping can only utilize the resources from this constrained area. These resources will be called a limited implementation area.

So the two requirements for new system which will be configured in FPGA after fault mitigation procedure are as follows:

- System is utilizing the resources in given implemementation area only.
- System is producing correct outputs - the implemented function of original system was not changed.

In case of transient fault mitigation the third requirement can be to preserve the capability of fault tolerance on the same level as in the original system. This means when for example the original system is capable to cope with two indepedent faults inside it, then the new system has to be able to overcome the same two faults too. Permanent fault mitigation can cause the reduction of fault tolerance capability to the situation when the system will cease to be fault tolerant.

Another goal of the research will be to enable the synchronization of reconfigurable modules in which the FT system designed by methodology is implemented. Since immediatelly after the partial reconfiguration of reconfigurable module in FPGA its state is undefined, its synchronization with other untouched modules can be required. Thus in such cases, the synchronization has to be the last phase of fault mitigation process. The synchronization procedure itself is not defined by the methodology since many approaches for it exist [48], [59], [35]. The designed FT architecture with reconfiguration controller will be able to cooperate with external synchronization controller.

The chosen strategy of fault mitigation in the methodology requires to store the configuration bitstreams needed for dynamic reconfiguration in external storage outside the FPGA. Since the size of bitstreams for modern FPGAs with many resources can be very large, it is crucial to bring some strategy to reduce their aggregated size. Not only the requirement on sparing the space must be taken into account but also on reduction of reconfiguration time, since the reconfiguration controller is loading the bitstream through the reconfiguration interface of FPGA serially. Thus, the total count of bitstreams and also the size of each bitstream has to be reduced to minimum.

The goals of the research can be summarized in the following way:

1. To propose the methodology for the FT design of digital system into FPGA with the ability to recover after transient and permanent fault occurence which satisfies these conditions:
    - The designed architecture of system is operating in limited implementation area which means that it can only utilize the resources from the area of FPGA which was designated for the system at the begining of its lifetime.
    - The occured transient fault in one system module is mitigated while the rest of modules in FPGA are not affected by it.
    - If the architecture of implemented system has to be modified to recover after permanent fault occurence, the new one has to keep producing correct outputs and it should remain fault tolerant if it is possible.

2. To design the reconfiguration controller which will control the mitigation process in FPGA after fault occurence done by PDR. It supplies the information about the detection and localization of fault, it determines its type and controls the reconfiguration process. Alternatively, it can also trigger the sychronization process when it is needed.

3. To create test platform which will enable the evaluation of methods and procedures described by the proposed methodology. For the FT architectures designed by means of methodology principles, the ability to survive will be tested by fault injection.

The proposed methodology covering these points is described in the following chapter.

Chapter 5

Methodology for fault tolerant system design into limited implementation area in FPGA

In this chapter the principles of the proposed methodology which aims at securing system by implementing its parts as fault tolerant systems into the limited implementation area in FPGA are described. The methodology incorporates fault mitigiation technique based on the use of several sets of precompiled configurations which are developed for this purpose.

The limited implementation area from the perspective of this research means the set of FPGA resources assigned for implementation of some system parts which are important from the dependability point of view. This implementation area is specified during the design phase of system implementation and it cannot be modified during system lifetime. This assessment limits the fault mitigation technique during permanent fault recovery process. The statement of limited implementation area simplifies the fault mitigation procedure for both transient and permanent faults based on precompiled configurations due to the possibility of building a deterministic fault mitigation scenarios. Thus, it can decrease hardware and performance overhead of fault mitigation techniques application due to enabling optimization efforts during the offline implementation.

5.1 Methodology basic principles

The proposed methodology defines the process of securing digital system designed and implemented in FPGA. In other words, it can be understood as the recipe how to redesign the given architecture of a system in FPGA and how to prepare the system for recovery after fault attack and thus make its lifetime longer. Such methodologies have their justification e.g. in long term missions where the implementation area becomes smaller after every permanent fault which occurs in the design. In this methodology, the desired goal is achieved by combining the approaches for detection, localization and mitigation of the occured faults.

The original system design is splitted into several parts from which the chosen ones are redesigned as FT architectures. The detection and localization process is then based on the comparison of replicated functional units in FT architectures and on other CED techniques. No specific methods are intended. The mitigation technique requires the localization on the PRM level. When the faulty PRM is localized, it must be determined to which type of fault defined by fault model this particular fault belongs. Mitigation process is different for both types of fault - transient and permanent. Both of them are driven by developed controller unit - Generic Partial Dynamic Reconfiguration Controller (GPDRC). This unit has a crucial role in the system because it is responsible for the task of fault mitigation and is able to control the reconfiguration performed through ICAP interface. It will be described in details in Section 5.2.

The developed methodology allows to detect and repair transient faults caused by SEU occurrence in FPGA configuration memory which can cause an incorrect operation of the system implemented into FPGA. The transient fault mitigation is based on the existence of relocatable golden copy of bitstream for affected PRM with the given unit type. When the fault is localized in one of the FT architecture units, GPDRC will execute the reconfiguration of PRM corresponding to this unit. After this process and unit synchronization process (when it is needed), the system is back to fully operational state as it was before the fault occurence.

The methodology also describes the process of detection and repair of permanent faults occurring in the configuration memory or in the physical resources (CLBs, interconnection resources, etc.) of FPGA. The main idea of permanent fault mitigation is based on the existence of alternative FT architecture sequences, all of them covering the same function. The functional units and the components supporting fault tolerant features of the design are implemented as single PRMs which are interconnected by means of a predefined interface. The repair mechanism in case of permanent fault occurence is based on downloading the configuration of another FT architecture covering the same function but with reduced range of support diagnostic circuitry into the same location of FPGA. With this new FT architecture the part of FPGA affected by fault is excluded from further use. The selection of FT architectures which implements one system unit in different number of PRMs and thus enabling the exclusions of the remaining PRM is called *a degradation strategy* of this unit in the following text.

5.1.1 System design based on the methodology

To be able to apply the methodology to system design, it has to be described in a hardware description language (VHDL) and its source codes have to be available. The output of this process is a new system which is secured by means of FT architecture to guarantee the resilience against both independently occurring transient faults and given number of permanent faults which affect the FT system correct operation.

Inputs

On the input of securing process the designer has to specify the following informations:

1. The design of system described by hardware language.
2. The definition of desired FPGA.
3. The user constraints for the implementation.
4. The assignment of implementation area in FPGA for given system.

Outputs

The outputs of the securing process are these:

1. The FT system design of the secured system described by hardware language with all neccessary equipment.
2. The complete configuration bitstream for initial configuration of FPGA.
3. The set of partial configuration bitstreams used by fault mitigation process.

The process of FT system design by means of the proposed methology is a complex process starting with the selection of parts intended to be secured against transient or permanent faults, the allocation of implementation area for secured parts, the decision about the strategy for permanent fault recovery and ending with the set of generated PRBs to be stored in external memory and to be used by GPDRC during the fault mitigation process. This process is described in details in Chapter 6.

5.1.2 Structure of fault tolerant system under design - basic principles

In the developed methodology, the design is protected by means of FT architecture to guarantee the resilience against both independently occurring transient faults and given number of permanent faults which affect the FT system correct operation. The methodology suggests to divide the implementation into certain number of PRMs. This set of PRMs put together is called *a configuration* in the following text. Each unit of FT system is placed in one PRM in a uniform way which means that relative position of all sources, connections and proxy logic inside PRM was identical for the particular type of the unit. This is required for relocation process, which will be described later in Section 6.1.2.

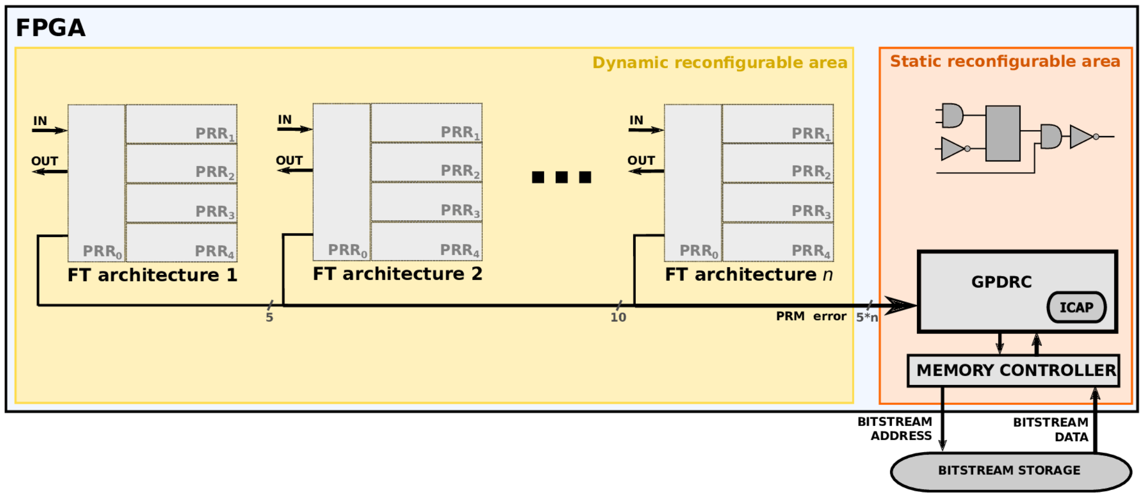


Figure 5.1: The main structure of the proposed methodology

In Figure 5.1, an example of the complete FT system design in FPGA based on the principles of methodology is shown. It consists of dynamic part in which FT architectures are placed and static part which contains GPDRC. The GPDRC utilizes the information about detection and localization of faults from the CED logic units of FT architectures. The set of error signals from PRMs (assigned in PRR1 - PRR4) are the inputs to GPDRC. Splitting FT architecture into several PRMs gives the possibility to exclude from the implementation one or several PRMs when they are affected by permanent fault. The interconnection signals between modules and the connections between the particular module and the rest of FPGA pass through single PRM assigned to PRR0 which is neighbouring with all other

PRRs. The other 4 PRRs can be assigned by PRMs of different units of the selected FT architecture. The number of these uniformly sized and structured PRRs can vary. The small count of PRR (3,4) can be used to implement only simple FT architectures (e.g. TMR with simple voter, duplex with checker) and can be used to recover system after permanent faults only very few times. On the other hand, the number of alternative configuration is smaller. The higher number of available PRRs allows to overcome more faults but brings much more alternative configurations.

To illustrate the application of methodology for securing a real system, several FT architectures were proposed which can be used in degradation strategy of some system unit. The first, most robust FT architecture, is TMR architecture with doubled voter which enables the detection of errors also in the voter. The next TMR architecture uses just simple non-protected voter unit. The last architecture is based on Duplex system with a comparator. This architecture is not fault tolerant since there is no possibility to distinguish which output of two replicated units is incorrect. But this system can run correctly until the first fault occurs and then it is detected by compare unit. The exact implementation of these architectures will be presented in Section 7.2. The utilization of them and assignment of their PRMs into allocated PRRs in system design by means of proposed methodology is shown in Figure 5.2. Each replicated functional unit is implemented in single PRM referred to as PRM_FU, complex voter unit is implemented in its own PRM referred to as PRM_VOTER and the routing between replicated units and the FT architecture external interface is constrained into PRM referred to as PRM_ROUTE.

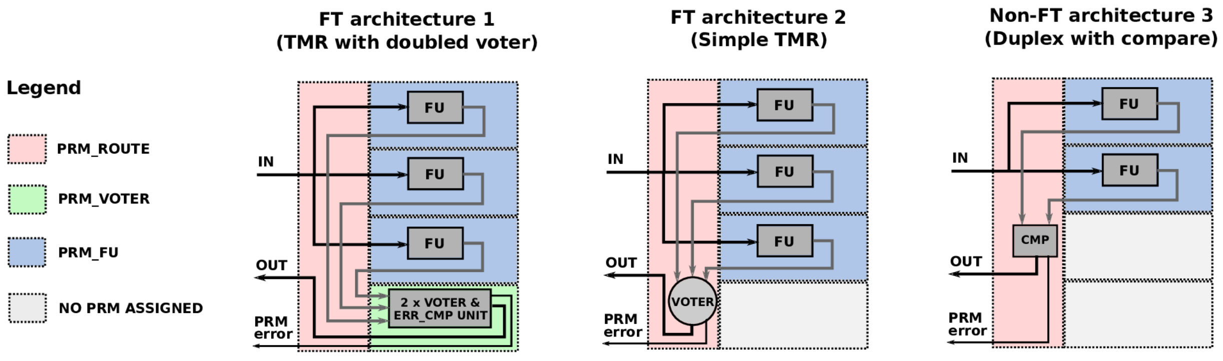


Figure 5.2: The assignment of PRRs by different PRMs

5.1.3 Generations of alternative FT architecture configurations

The methodology is based on the existence of precompiled configurations of an FT design which are applied when a permanent fault occurs. These configurations are divided into several generations. Configurations from one generation contain the same FT architecture but with different PRM placement. The enumeration of all possible generations for such FT architectures is shown in Figure 5.3.

The number of unused PRMs (PRMs excluded from use) in configurations of each generation reflects the generation number. The code of configuration is assembled from flags indicating if the corresponding PRR is assigned by PRM (see legend in Figure 5.3). The configuration with code 1111 from generation 0 represents the starting configuration for this system part. After the first permanent fault is detected and affected PRM is localized, the new configuration excluding the faulty PRM from the next generation is

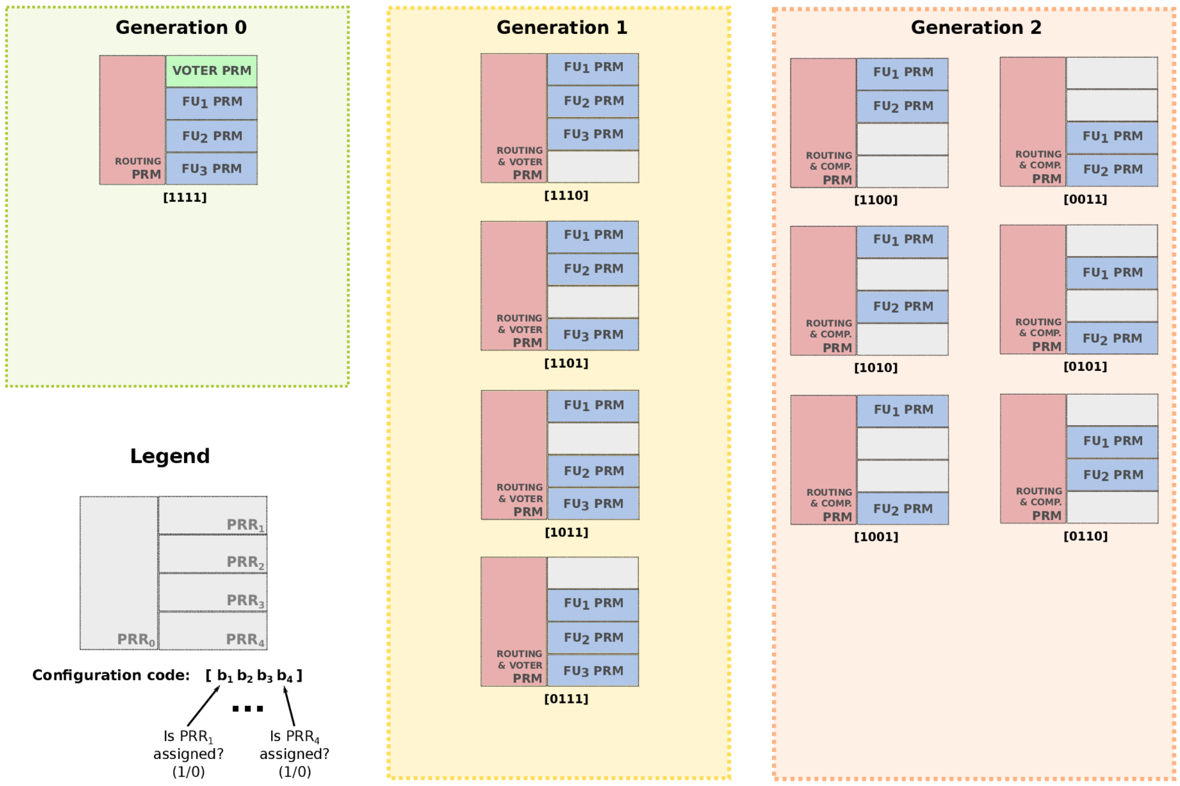


Figure 5.3: The generations of FT architectures and their alternative configurations

chosen to be used for system implementation. This principle is applied again when a new fault affects another PRM. The number of possible variants of configurations is rising with the number of PRMs affected by fault. To reduce the memory requirements for the configuration, bitstream relocation method is used to avoid the existence of several copies of PRM containing the same type of unit. Only one copy of PRM bitstream for each type of PRM except PRM_ROUTE is needed. Only the bitstream designated as PRM_ROUTE is stored for each configuration in the memory.

5.1.4 Reducing the number of configuration bitstreams via bitstream relocation technique

Due to the specifics of design and implementation flow adopted by Xilinx tools, the generated partial configuration bistream of PRM cannot be assigned to different PRR than it was originally designated to. One PRB has to be generated for each PRR where the PRM will be configured. Thus, if there is a need to apply N PRMs of different types to any of M PRRs, $N * M$ PRBs have to be produced and stored in external memory for run-time partial reconfiguration.

With the adoption of bitstream relocation technique, the number of generated PRBs is reduced to N. These PRBs can be used then for reconfiguration of all PRRs satisfying the conditions for the application of relocation technique. These conditions are applied in the design phase and the implementation phase. One of the main limitations of this technique is the need to have all PRRs with identical FPGA resources.

In common, this technique always starts by generating the PRBs for all types of PRMs in one chosen location of PRR. Before the run-time reconfiguration, the bitstream manipulation modifying the information related to its location to apply it into other different PRR is needed.

The implementation of this technique in proposed FT system design process is described in details further in the text of the Section 6.1.2.

5.1.5 Synchronization issues

When the reconfiguration of a faulty PRM in an FT architecture with replicated units is done, the problem with synchronization between these units can arise. While the state of reconfigured unit is initialized after reconfiguration, the other units operate and their states are different from the reconfigured ones. This issue is typical for PDR of PRM implementing sequential circuits which is done during the transient fault mitigation process. Another problem with the synchronization can arise when the change of FT architecture configuration is done and all its PRMs are reconfigured to overcome the permanent fault occurence. The state of its functional units is reset while the other system parts connected to input or output of this architecture are stopped in some accumulated state. Solving of the synchronization issues was not the goal of the research but it was also studied as a consequence of using PDR in fault mitigation process.

Several synchronization techniques were presented in order to restore the reconfigurated unit in TMR scheme. They differ in the type of target system. When a sequential circuit is implemented as a Finite State Machine (FSM), checkpoint stategy presented in [49] can be used. It is based on setting the state of the reconfigured unit to often reached one and holding it until other units reach this state again. Another method for complex sequential circuits (i. e. softcore processors) was presented in [24]. The state of all internal registers of the chosen correctly working unit can be saved to memory after reconfiguration, all units stopped then and the content of registers from memory writen back to all units. Other techniques of synchronization are designated for the packet processing circuit. In this case it is sufficient to mask the output of reconfigured unit until the next packet comes to process. In [13], a method is presented where for the stated minimal time (longest period between 2 arrivals of consecutive packets) the function of fault detection unit of the system is disabled, so the incorrect outputs of the reconfigured unit do not cause the new reconfiguration of the unit. During this minimal time, the local reset will be generated which will result in the synchronization of units.

In this methodology, a specific synchronization method for complex sequential circuits based on copying the state of other replicated unit was tested. The architecture with synchronization capability incorporates voter unit with fault detection and localization ability and implements simple control mechanism of unit synchronization. The proposed architecture for TMR scheme is shown in Figure 5.4.

Because the used synchronization technique is based on copying the state from correctly working unit to the reconfigured one, all units are connected by oriented point-to-point (source-destination) connections to the ring. Functional units are modified in such a way, that they expose synchronously the values of state registers one by one starting when enable signal is disabled. On the other side, the unit with enabled receive signal is saving at the same time the values from antecedent unit into its state registers. When all register values are saved, the unit with enabled receive signal triggers the sync end signal to inform the voter unit about the end of synchronization process. To prevent the new reconfiguration of

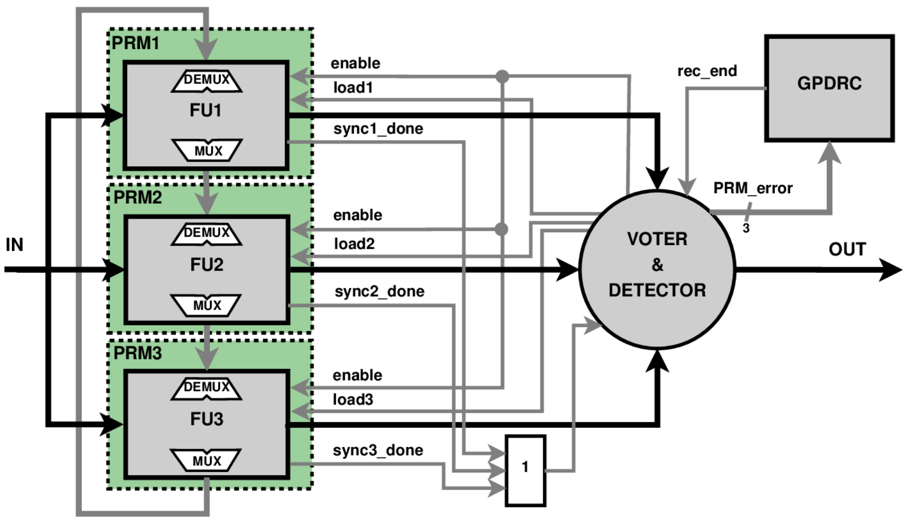


Figure 5.4: FT architecture with the unit synchronization

unit which is not yet synchronized and can be considered as faulty due to its undefined state, the error signals of this unit connected to GPDRC are masked by voter until the impulse on sync end signal appears. During the synchronization process all units are stopped.

When needed, this synchronization method can be applied to FT architectures which we propose later in the text. The implementation for FT systems based on duplex scheme which was not mentioned in previous paragraph can be designed in a similar way as for the TMR scheme. Control mechanism for state copying can be implemented in detection logic in PRM_ROUTE instead of voter unit.

The overhead which this technique brings, depends on the number and the width of FUs state registers. This technique was presented in [33].

5.2 Generic partial dynamic reconfiguration controller

Many works dealing with partial dynamic reconfiguration are using microprocessor to control this process. There could be a significant waste of computational power when such universal processor is used to drive the reconfiguration. Moreover, the wasted performance causes higher power consumption and higher complexity of the solution which increases the probability of failures. The microprocessor can perform another computation, however, an error in any software module may delay or even stop the reconfiguration process. Therefore, the decision to implement generic reconfiguration controller of PDR as a hardware unit to reduce resource utilization and thus reduce the failure probability in the controller was made.

The concept of the first GPDRC for transient fault mitigation was presented in [56]. The first implementation within system with counter and SEU injection was presented in [26]. Previous GPDRC design has been extended to be able to perform reconfiguration of entire FT system (several PRMs) when the permanent fault occurs in its PRM. New issues such as choosing the proper configuration from the next generation of configurations, performing the relocation process on loaded PRBs and the synchronization of the complete FT system were solved and implemented into controller. The GPDRC for transient and permanent fault mitigation was presented in [34].

5.2.1 The design goals of GPDRC

Before the development of GPDRC, several design goals to be achieved were defined:

- The resource utilization of new controller has to be lower than the standard controller units implemented by universal softcore processors.
- The controller must be built in generic way to be able to perform PDR in the systems with the different number of PRMs. The size of controller should be related to the size of the system - if the system is small and it consists of few PRMs only, the controller should be much smaller than for complex systems with plenty of PRMs.
- The controller should be autonomously able to determine the type of fault which occured in a PRM, whether it is a transient or a permanent one - for this purpose the information on whether the fault occurred during n successive reconfiguration cycles (the reconfiguration cycle consists of faulty PRM detection, PRM reconfiguration, PRM synchronization) can be used. If the fault occurrence is equal or lower than n, the fault is seen as a transient one, otherwise it is concluded that the fault is a permanent one.
- The PDR will be done via internal reconfiguration interface (ICAP in Xilinx FPGAs) and utilize its full speed (up to 100MHz).
- To reduce the number of needed precompiled PRBs the controller has to implement the technique to use the same PRB for the PDR of several PRMs where it is possible (e.g. the same type of PRM but different physical assignment to PRR).
- The controller should allow the synchronization of reconfigured PRMs by ignoring the error output of reconfigured PRM until it is synchronized. The synchronization can be done autonomously in the concerned FT architecture or it can be driven by external sychronization controller.
- The controller should be able to cooperate with different types of external memories serving as a bitstream storage. The bitstream data transfer interface must be so general as to allow the connection to different external memory controllers.

5.2.2 GPDRC unit design

The detailed architecture of GPDRC can be seen in Figure 5.5. Its interface contains an error vector of FT architectures as input. Its width depends on the number of FT architectures and the available number of PRMs for each of them. The next interface signals such as bitstream address, bitstream data and their validity indicators are designated to communication with external bitstream storage via connected memory controller when bitstream is transported through ICAP interface of FPGA. The *sync done* and *rec done* signals are intended mainly for controlling the synchronization of reconfigured PRMs in FT architectures. While the *arch. index* vector contains the index of current FT architecture where the fault mutigation process is done, *PRM error index* vector expose the specific PRM which is currently handled by GPDRC. The *hard* signal indicates, if current PRM is affected by permanent. The *fatal* signal announces the situation when the FT architecture cannot be repaired by GPDRC because the number of available PRMs has fallen below the required minimum.

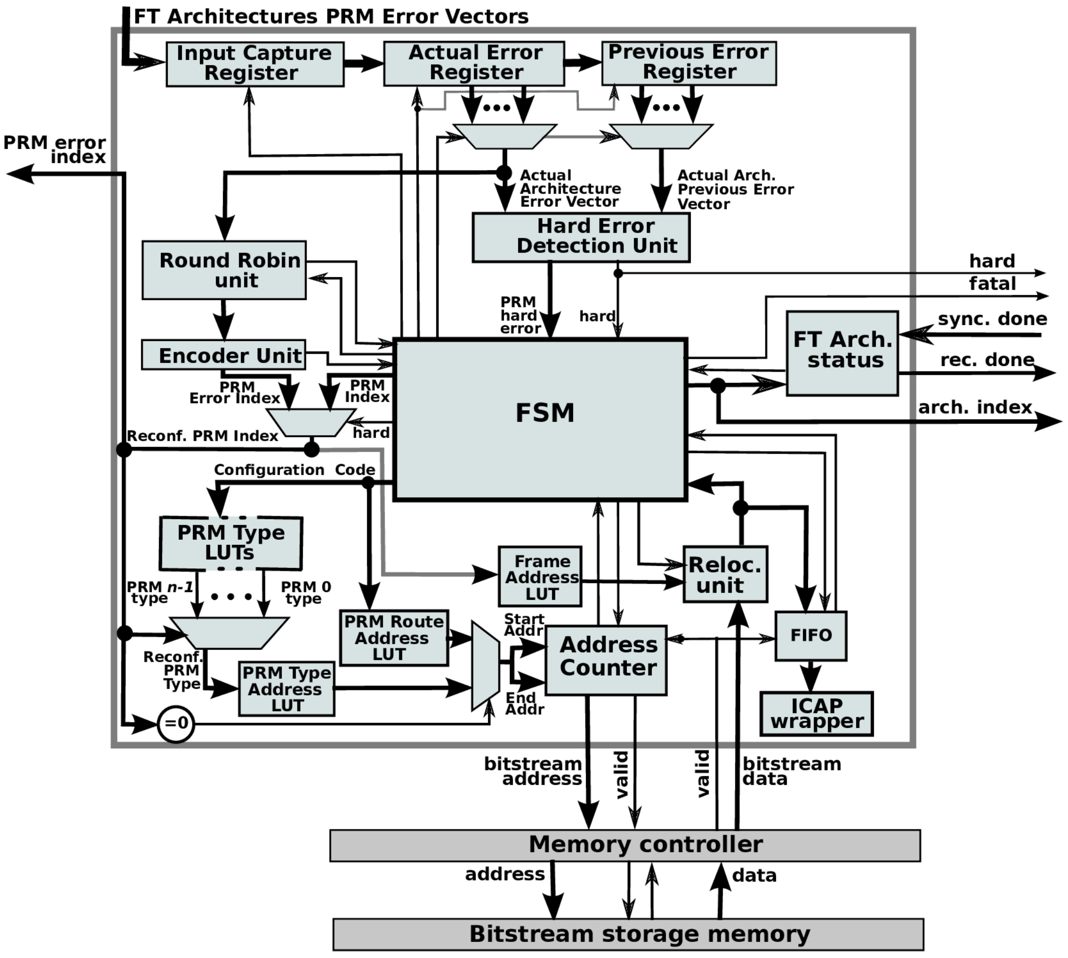


Figure 5.5: Fault tolerant system structure for SRAM based FPGA

The GPDRC contains nine main units, one FIFO unit and several Look Up Tables (LUT) and multiplexers (MUX). The information about errors on input vector is captured during one reconfiguration cycle in input register and then loaded into error register. The GPDRC works sequentionally, thus one reconfiguration cycle is composed of looping over FT architectures error signals. To be able to determinate the type of fault, the value of error vector from previous reconfiguration cycle is also kept available. Hard error detection unit can designate the fault as permanent on the basis that the same error was detected during two consequent cycles.

When there is no permanent fault detected, the roundrobin unit looks for any transient error in register and together with encoder unit returns its index when some is found. This index is used to choose the type of unit in the selected faulty PRM. The relationship between PRM index and its type for each configuration and FT architecture is known because it is stored in PRM type LUT. After resolving the PRM type, the address of its bitstream in memory is looked up and used in address counter. When the index is equal to 0, the address of bitstream of appropriate PRM with routing is used.

If a permanent fault was detected, each PRM of actual FT architecture will be reconfigured. Thus, the value of PRM index signal loops from *0* to *PRM number - 1.* The configuration code supplied by LUTs is formed from error signals of actual FT architecture in this way: *error n-1 error n-2 ... error2 error1*. The *error 0* signal is generated by PRM which contains routing and does not affect the choice of FT architecture configuration. The resolving of indexed PRM bitstream address follows the same steps as it was described in the previous paragraph.

The address counter unit is used to address data words in bitstream storage from the starting to the ending address. Due to the use of one copy of the bitstream for all units of the same type, the relocation process of bitstream must be performed. This is done by relocation unit, which uses the address of actual reconfigured PRM from Frame Address LUT to modify the value of frame address in bitstream.

When the PDR of all faulty PRMs in one FT architecture is finished, the value of corresponding *rec done* signal in FT Architecture Status unit is set. This means, that GPDRC will ignore the errors coming from this architecture, until its PRMs are synchronized by some external unit or mechanism. After activating the appropriate *sync done* signal, the error signals from FT architecture will be no more ignored by GPDRC. This approach based on excluding the synchronization from GPDRC allows to use different and more suitable techniques of synchronization for each FT architecture.

5.2.3 Implementing GPDRC unit as fault tolerant

The GPDRC as the important part of this reconfigurable architecture which should be protected from the impact of SEUs. One solution is to move it outside FPGA to radiation hardened fabric. Other possibility is to implement it as FT system. In this case, the GPDRC must be moved into dynamic part of FPGA and its units will be designed as FT architectures and divided into PRMs and their error signals connected to the error input of the GPDRC. Then, the fault mitigation of GPDRC itself is possible because of its FT design. This process will have higher priority than the fault mitigation in other PRMs. This approach requires excluding the instance of ICAP outside of the GPDRC instances because only one ICAP instance is available.

5.3 Fault mitigation procedure

In Figure 5.6 the behavior of the system after a fault is detected in PRM is shown in flow diagram. The fault is detected by the FT architecture. The FT architecture generates a set of error signals which identify the faulty PRM (step 0). This is possible due to the fact that the functional units and voters are implemented into separate PRMs and the relation between the units and PRMs where they are placed is known.

When the faulty PRM is localized, the GPDRC determines, if the occured fault will be considered as transient or permanent one. The solution used in the case of transient fault occurence is denoted as the option A further in this text. If the fault is seen as a permanent one, then the subsequent steps depend on whether the current configuration comes from the final generation (Generation 2 in this case). The GPDRC stores the configuration code of actual configuration so it is able to identify that it is from final generation. If it is from final generation, there is no additional option to continue in mitigation of this new permanent fault and the FT architecture will indicate this to GPDRC unit. Then, the intervention from outside is needed (e.g. physical placement of configuration is moved to another locality of FPGA or the FPGA is replaced with a new one). In the situation when actual configuration is not from final generation, it is possible to mitigate the occured fault and the solution is denoted as the option B.

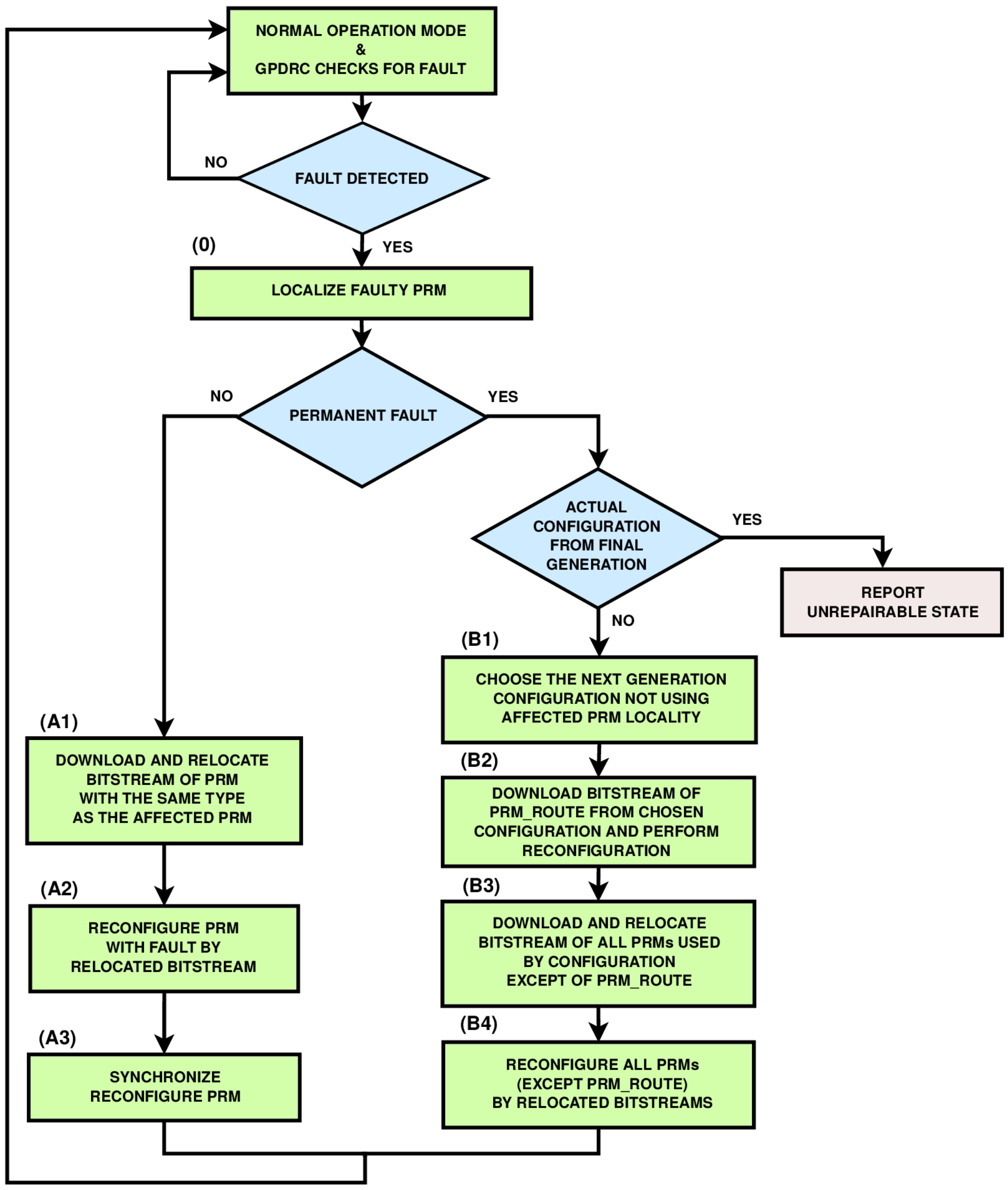

NORMAL OPERATION MODE & GPDRC CHECKS FOR FAULT
FAULT DETECTED
NO
YES
(0)
LOCALIZE FAULTY PRM
PERMANENT FAULT
NO
YES
ACTUAL CONFIGURATION FROM FINAL GENERATION
YES
REPORT UNREPAIRABLE STATE
NO
(B1)
CHOOSE THE NEXT GENERATION CONFIGURATION NOT USING AFFECTED PRM LOCALITY
(A1)
DOWNLOAD AND RELOCATE BITSTREAM OF PRM WITH THE SAME TYPE AS THE AFFECTED PRM
(B2)
DOWNLOAD BITSTREAM OF PRM_ROUTE FROM CHOSEN CONFIGURATION AND PERFORM RECONFIGURATION
(A2)
RECONFIGURE PRM WITH FAULT BY RELOCATED BITSTREAM
(B3)
DOWNLOAD AND RELOCATE BITSTREAM OF ALL PRMs USED BY CONFIGURATION EXCEPT OF PRM_ROUTE
(A3)
SYNCHRONIZE RECONFIGURE PRM
(B4)
RECONFIGURE ALL PRMs (EXCEPT PRM_ROUTE) BY RELOCATED BITSTREAMS

Figure 5.6: Reconfiguration flow diagram

Option A - recovery from a transient fault: After a transient fault is detected, GPDRC reads from external memory the PRB which responds to the type (PRM_FU, PRM_VOTER, PRM_CHECKER, etc.) of the identified faulty PRM. The type of the unit is known because the GPRDC knows which configuration is configured actually and the distribution of PRMs in it. The downloaded PRB is originally designated to the first suitable PRR (typically to PRR1, see Figure 5.7). Therefore, the next step of mitigation process (step A1) will be the relocation of this bitstream in such way that it can be used for reconfiguration of the affected PRM. The reconfiguration process of this PRM with the relocated PRB is driven by GPDRC (step A2).

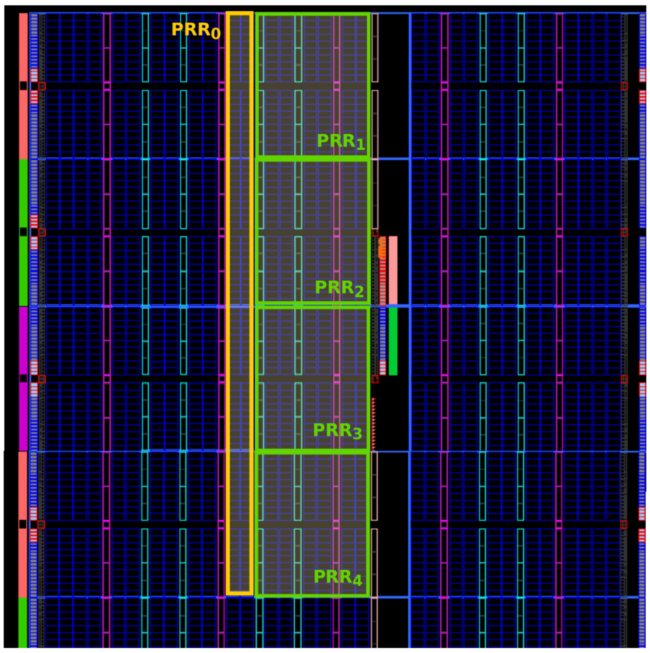


Figure 5.7: The dislocation of partial reconfiguration regions belonging to one configuration of FT architecture in the FPGA

After the reconfiguration is finished, in some cases the PRM must be synchronized with other components of FT architecture. The synchronization can be also controlled by GPDRC (step A3).

Option B - recovery from a permanent fault: After a permanent fault is detected in PRM and the actual configuration does not belong to the final generation, new configuration from the following generation is selected. This configuration will not use the faulty PRM. The GPDRC will choose configuration according to configuration code which will respond to bitwise negation of the vector of error signals from FT architecture (B1 step).

An example: let us say that the currently used configuration has the 1111 code (generation 0) and permanent fault affecting the operation of the voter unit occurs. The vector of error signals from FT architecture appearing on the PRM error signal vector on GPDRC will be 1000. The bitwise negative of this value will be the code 0111 of the new selected configuration from generation 1.

The PRB for PRM_ROUTE (PRM with the interconnections) of selected configuration is stored in the external bitstream storage. This bitstream is designated to reconfigure resources of PRR0 (the only PRR of FPGA where this bitstream of PRM_ROUTE can be assigned). This implies that there is no need to relocate this PRB (step B2).

The downloading of PRB copies implementing all remaining PRMs will be the next action. The number of needed bitstream copies and their type (if it is implementing PRM_FU, PRM_CHECKER or PRM_VOTER) is determined by the selected configuration. The PRBs of all PRM types are downloaded from the same destination, as in the case of reconfiguration after transient fault. Each of these downloaded PRBs will go through relocation process which will make them suitable for appropriate PRRs (step B3).

An example: if the configuration with 0111 code is selected as the new one, as the first step the PRB of PRM_CHECKER will be downloaded and because all of these PRB types are designated to PRR1, it has to be relocated to PRR2 and then used for PDR.

Subsequently, PRB of PRM_FU is downloaded, relocated to PRR3 and used for PDR. The final step will be similar as the previous one with the only difference that the new destination will be PRR4.

The downloaded and relocated PRBs are used for the reconfiguration of PRMs, which are used in the configuration (step B4). After completion of the reconfiguration, local reset of units in newly configured PRMs is performed. Also some kind of synchronization (state recovery of all units in affected PRMs) can be performed in this step.

5.4 Summary

In this chapter, the basic principles of methodology were presented. It describes the process of securing a given system design to become less vulnerable to fault occurences and the steps of fault mitigation when some fault is detected and localized.

The system is divided according to designer decision to several parts which are implemented as FT architectures. Fault detection is mainly based on CED and requires additional logic to localize faulty PRM. The fault mitigation is driven by dedicated reconfiguration controller (GPDRC) which determines the scenario of fault mitigation process according to the fault type and taking into account the previously occurred faults.

The transient fault is mitigated by PDR and during its performance the FT architecture ensures the production of correct outputs. To reduce the number of stored precompiled PRBs, the relocation of the golden copy of the PRB for the needed PRM type is performed. When a permanent fault is identified and some alternative configuration which does not utilize the resources from faulty region of FPGA exists, then the reconfiguration of entire FT architecture is performed. With this approach many alternative configurations can arise. Therefore they are grouped into the generations of configurations according to the number of faulty PRM they can handle. With their setup, GPDRC can determine which configuration will be used to recover the system after permanent fault occurence.

Chapter 6

Design of fault tolerant architecture by means of developed methodology principles

The process of FT architecture system design to meet requirements defined by the proposed methodology is described in this chapter. The tool developed for automatic FT architecture generation is presented in this chapter as well.

6.1 Prerequisities

This section describes the methods which are used or can be alternatively used during design phase.

6.1.1 PRM isolation

The modules of FT architecture have to be implemented into separate PRMs according to the proposed design principles. This means that FPGA resources from given PRM can be used only to implement the function of designated FT architecture module. At the same time, all pieces of module logic and connections (except of connections between the module and the rest of design) have to be implemented by these resources only. This is not ensured by current design tools which are processing optimizations such as redundacy removal what can lead to unwanted utilization of FPGA resources in non-designated areas of FPGA (some part of module is implemented outside the restricted area for its designated PRM) or leading crossing wires through the area restricted for some PRM which is not using them.

When the modules are implemented into separate PRMs as isolated partitions of FPGA, it enables later partial reconfiguration in single PRM where the implemented module can be changed while the remaining modules remain unchanged and correctly working. To ensure this, several design strategies can be adopted:

- *Post-implementation adjustment of implemented design* created by automated tool can be done by designer. The output of synthesis and implementation tool can be modified in such way that the PRM partitions are isolated by remapping conflicting resources and rerouting conflicting connections. This method can be very costly and time demanding.

- *Custom built PAR tool* can be created to replace the entities of static design from PRM and reroute the connections between static logic going through PRM. This tool can be a part of standard design flow or it can be used to preprocessing design implemented by some other design implementation tool. This tool will need a set of constraints specifying the PRM partitions location.

- *Blocking macros* can be added to design to prevent the utilization of PRM resources while implementing the static design. This hard macro inserted to design utilizes all resources and connections within the stated area in FPGA which is reserved for PRM. The GoAhead tool which is implementing this strategy can be taken as an example [7]. The steps of synthesis and implementation are as follows. First, the hard macros generated by GoAhead tool for desired PRM partitions are added to the implementation and the static design configuration is generated. This ensures the logic and connections of static design will not use the resources from PRM partitions because they are utilized by dummy logic and connections of generated hard macros. The next step consists of generating configuration of the design with proper PRMs included. The main drawback of this method is its binding to specific FPGA type. For each FPGA family, the complete description of FPGA resources has to be delivered.

- *Isolation Design Flow (IDF)* is a strategy available for FPGAs from Xilinx to ensure the isolation of logic, routing and utilization of IOBs of the specified module design. The designer has to consider floorplanning earlier than in standard design flow. The design has to be partition based. This involves several goals which have to be achieved [68]:

  – each module which should be isolated must be in its own partition,
  – each partition must consist of a single module instantiation,
  – a special restricted area (fence) must be used to separate isolated partitions within a single chip. The fence is a set of unused tiles in which no routing or logic is present,
  – IOBs used by isolated partition must instantiated inside this partition (not in top-level entity of design),
  – communication between isolated functions in FPGA is achieved through the use of trusted routing (user defined route between isolated partitions) or through off-chip communication.

  To ensure that unwanted optimization will not appear, IDF consists of two steps. First, each isolated module is synthesized and implemented independently of the other partitions. After this step is done for all partitions, the design is merged into a flattened FPGA design for device configuration.

6.1.2 Bitstream relocation technique

The PDR design flow adopted by Xilinx design tools determines that each PRM has to be designed before PAR phase is performed and also dynamic reconfigurable modules have to be dedicated to specified PRRs in this phase. This flow implies that the partial configuration bitstream of PRM is designed specifically for a certain location in the FPGA which is defined by the PRR. This complicates the use of one partial bitstream of one PRM for the configuration of different PRRs. This is caused due to the fact that the bitstream contains information about the starting position of the block of resource which it configures. This position is written in frame address part of bitstream (see Section 2.2.7). The modification of this value requires the change of Cyclic Redundancy Check (CRC) code in the footer of bitstream. CRC can be recalculated again when the frame address is changed or it can be invalidated.

To be able to perform the relocation, the design of PRMs has to satisfy the following conditions for all PRRs where the given bitstream will be used for their reconfiguration:

1. The amount and the relative layout of reconfigurable resources is identical. The restriction to amount of resources means that the number of each resource type (e.g. CLBs, BlockRAMs, DSPs, etd.) as well as the number of allocated rows and columns of FPGA resources has to be same in all PRRs. The layout of all different types of FPGA resources in these allocated regions has to be the same, too.

2. The relative placement of proxy logic has to be the same. The proxy logic is automatically added to each signal of the design which is crossing the PRM and its placement is typically not the same in all PRRs where the relocatable PRM can be placed.

3. The routing path between proxy logic of each PRM and the static region have to cross the boundary between PRR and the static part of design in the same relative position to current PRR layout. Even in the case when the proxy logic is placed in all PRRs in the same relative location the wires connecting them with the static area can be led along with different routing in each PRR.

4. The wires of static part crossing the PRR (without any junction inside) have to be excluded.

To satisfy condition 1, the PRRs are contrained to areas which consist of the same resources. This is done by setting AREA_GROUP constraint to the set of resources (each resource type are constrained separately) for top-level entity of each PRM. The setting of this constraint to region bounded by rectangle defined by the position of two oposite corner points ([a,c] and [b,d]) is shown by in the following code.

```
AREA_GROUP "entity_name" RANGE=<logic_resource1>_XaYc:<logic_resource1>_XbYd
AREA_GROUP "entity_name" RANGE=<logic_resource2>_XaYc:<logic_resource2>_XbYd
...
```

This constraint can be applied to all types of FPGA logic such as SLICEs, RAMB16s, IOBs, PLLs, DCMs, BUFGs, DSP48Es, etc. According to Xilinx application note [63], it is recommended to include (constrain with AREA_GROUP) also all unused resources of any type if they are physically located within the same area as the utilized resources to the region constrained with AREA_GROUP .

The assignment of PRM top-level entity to the set of resources by AREA_GROUP constraint which will create the PRR suitable for relocation is shown in Figure 6.1.

For the correct placement of proxy logic on the boundary of PRM (condition 2), the several placement constraints can be used. They can be specified in User Constraints File (UCF) which is supplied by Xilinx ISE implementation tool. The useful constraints for this task are as follows:

- **PIN** - the constraint is applied to nets connecting the proxy logic of PRM and the rest of design. As the PRMs are placed in the successive columns of FPGA, only

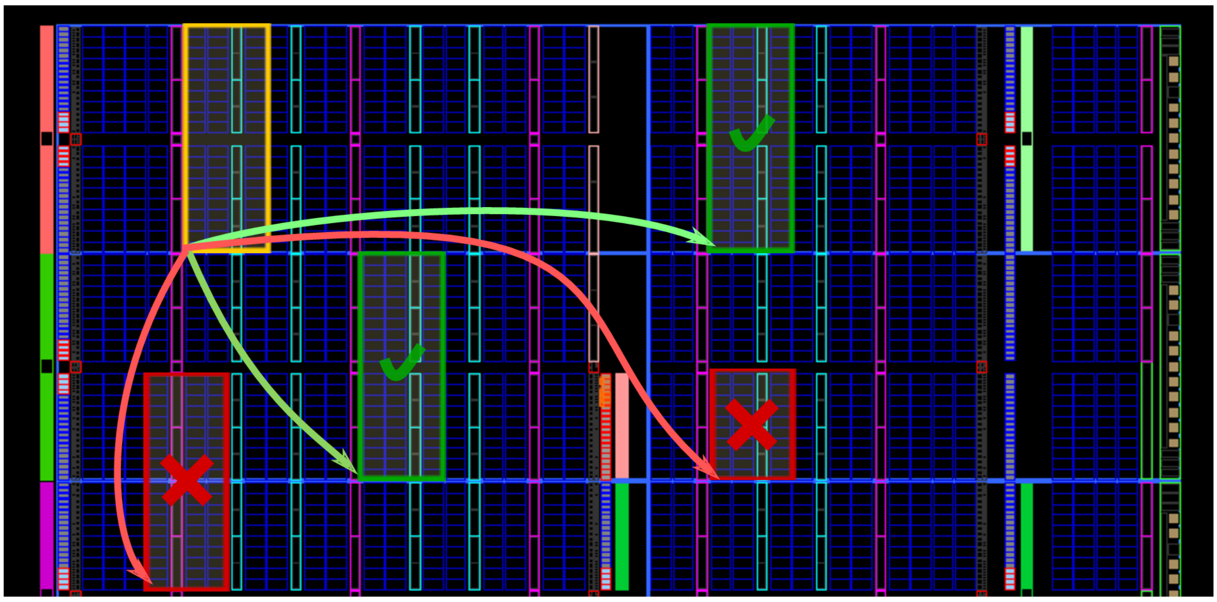


Figure 6.1: The example of regions suitable to host the same relocatable PRM

the location of proxy logic in first PRM is selected by designer (or design tool). The position of proxy logic and its connecting nets for the remaining PRMs can be determined by adding the constant value. This increment is evaluated from the difference of the position of two successive columns.

- **LOC** - the constraint is needed to specify location of entity in some FPGA resource component (e.g. SLICEs, BRAMs, IOBs, etc.).
- **BEL** - the constraint is used to specify the utilizitation of a particular slice or part of components (SLICEs, BRAMs, IOBs, etc.). As an example, it is possible to specify the specific LUT inside CLB in specific SLICE.
- **LOCK_PINS** constraint is used to force the utilization of the same inputs in all proxy by implementation tool.

The use of these constraints to bind single signal to specific LUT in specific SLICE is shown by the following code.

```
PIN "PRM_entity_name.signal" LOC=SLICE_X1Y2;
PIN "PRM_entity_name.signal" BEL=A6LUT;
...
INST "PRM_entity_name" LOCK_PINS=ALL;
```

The example of constrained proxy logic in several PRRs which is possible to be configured by one relocatable PRM is shown in Figure 6.2.

The same (relatively to PRR layout) routing between proxy logic and static area (condition 3) can be enforced by adding single LUT to static area for each proxy logic block implemented in all PRRs. This technique simulating the use of bus macros in older FPGAs was presented in [17]. The added LUT is placed on the opposite side of PRR boundary very close to proxy logic I/O to compell the router to route the connection in all PRRs in the same way (see Figure 6.3).

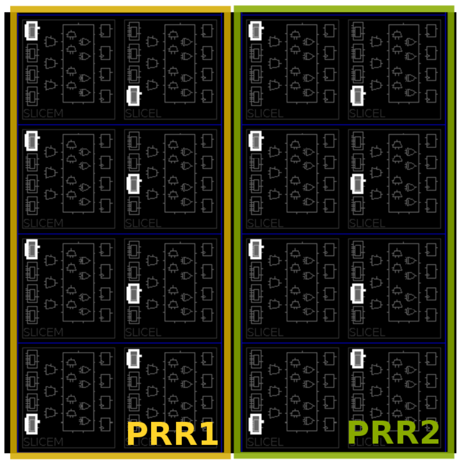


Figure 6.2: The constraining of proxy logic inside PRRs

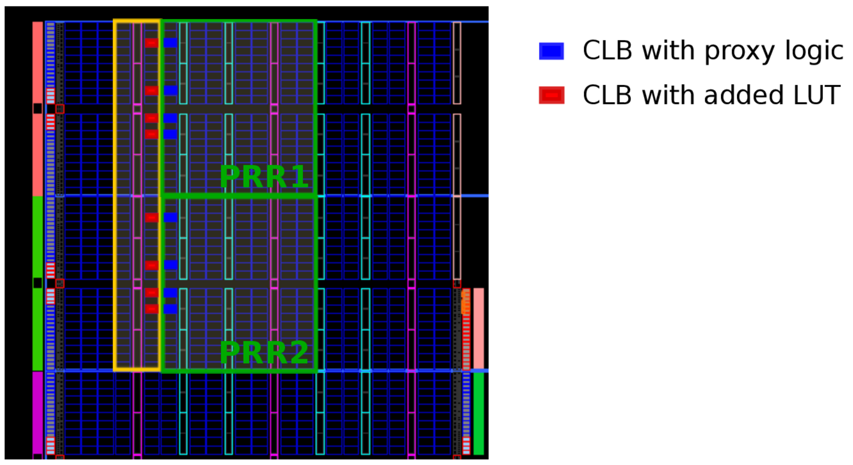


Figure 6.3: The addition of LUT to each proxy logic block

The probably most complicated is the satisfaction of condition 4. The wires of static area crossing the reconfigurable region can be solved by PRM isolation techniques described in previous section. First, the blocking macros created by GoAhead tool [7] were used in this work to implement the static logic without interfering the stated PRMs. This technique requires the specific implementation of blocking macros for each type of FPGA and therefore this cannot be transferred between FPGAs. Another drawback is the need to convert files in NCD format (generated from Xilinx ISE implementation tool) to XDL format (its alternative in human readable format) and back repeatedly. This conversion can be very time demanding when implementing design for large devices. The IDF strategy is a new concept for newer FPGAs and therefore it was not addressed by this work. The adoption of this strategy in older FPGAs and older design tools is possible but there are some bugs which complicate its use. With the careful placement of the entities from the static part of the design, the probability of the occurence of these unwanted wire crossings can be dramatically reduced. The remaining crossings can be mitigited by designer intervention in tools like FPGA Editor from Xilinx ISE toolkit or RapidSmith [38]. RapidSmith is an open-source design CAD tool, but it also requires the conversion to XDL format to perform changes on the design. After the change of static design is done, the bitstream can be generated from the modified design implementation.

When the design of PRMs is following these four defined restrictions, the relocation of final partial configuration bitstreams is possible. There is no need to be familiar with the device bitstream compositions. The bitstream relocation can be impossible in some cases. This can happen for example in situation when the PRMs are too large and there is a lack of similar regions with the same set of resources and their relative position inside the region.

6.2 Fault tolerant architectures design

The application of the methodology requires the specific process of system design. When this design is adopted, it is ensured that faults appearing subsequently in functional modules or other FT modules (containing voters, checkers, etc.) of design can be mitigated.

The system design according to the methodology contains several required steps. First, the user system design on the input is splitted into the set of important parts designated for securing and the set of remaining parts to be left as they are (i.e. unsecured). For each important part, the degradation strategy to overcome permanent fault occurence is chosen and some part of the implementation area of FPGA is allocated. Finally, the GPDRC unit is added into modified system design to provide the control of fault mitigation process. These steps are described in more details in the following sections.

6.2.1 System design partitioning

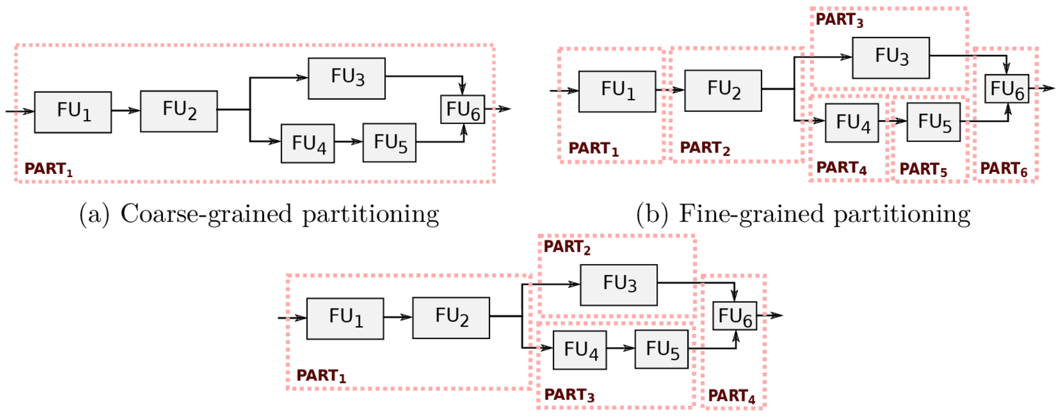
The original system design delivered from a designer for securing has to be divided into important parts in terms of required dependability and the remaining parts which may remain unsecured (from the methodology point of view) or they are secured in some other way. From the chosen important system parts every single part will be secured as single FT architecture with fault mitigation capability according to the methodology.

The process of partitioning has to be driven by designer knowledge of importance of each system part. This can be gained as the result of modelling reliability of system parts and the impacts of faults occured in specific system part to entire system. In FPGA, fault injection is frequently used to examine the system endurance to impacts of possible faults.

The partitioning can be done with different granularity as shown in Figure 6.4. These approaches can be categorised into 3 groups.

- Coarse-grained partitioning - One possibility of system partitioning is to take the system as one single part. It is the easiest option of securing the system when just one set of FT architectures will be created. Therefore, the resulting replicated modules and the other FT modules of FT architectures will be very big as well as the partial configuration bitstreams of these modules. For these reasons, this option should be preffered only in some cases, such as the situation when the original system is small sized or when the system has to be taken only as a black box (e.g. hard macro).

- Fine-grained partitioning - Other option in partitioning is dividing the system into more smaller parts. With this approach for each single part custom set of FT architecture can be chosen and the resulting modules of FT architectures will be smaller (see Section 7.3). The time needed to fully recover after fault occurence (whether transient or permanent) will be lower since it is mostly affected by partial configura-



(a) Coarse-grained partitioning

(b) Fine-grained partitioning

(c) Mixed partitioning

Figure 6.4: Design partitioning with different granularity

tion bitstream size. The drawback of this approach is the slightly increased size of GPDRC due to increased number of PRMs which it controls (see Section 7.1.1).

- Mixed partitioning - The possibility to combine two previous approaches and divide the system into some small parts small and some bigger parts also exists. These bigger parts can consist of several (or many) small units. The application of methodology demands the wrapping of these smaller units by top-level entity (in VHDL) to create the mentioned bigger part.

6.2.2 Selection of the degradation strategy for recovering from permanent faults

From the dependability point of view, not all of the chosen important parts must be categorized to the same level of importance. Permanent fault occurence in system is mitigated by downgrading the FT architecture from the robust one to less robust one. This step is required every time the permament fault occurs in currently occupied PRR containing the PRM of FT architecture. The less robust FT architecture will exclude this PRR from the further use. The number of PRRs which can be excluded at the same time then specifies the number of permanent faults which can be handled by this secured part of the system.

With the increasing number of permanent faults which can be handled, the number of posibble variations of PRMs dislocation into available PRRs is increasing almost twice (see generations of configurations in 5.1.3). This results in higher demand on external bistream memory capacity. Thus, it makes sense to research the probability of permament fault occurence in current secured system part and its impacts and based on this consideration choose the sequence of FT architectures for the degradation strategy.

As an example, the key part of system can be secured by implementating it as TMR with triplicated voter unit. When a permanent fault appears, the TMR with single voter which excludes one PRR from utilization will be used. The next step will be the degradation to the duplex architecture with checker units and finally it will end as simple duplex architecture without the possibility of recovering from the next permanent fault occurence. Another part of the system which is not so crucial for securing from any reason (i.e. the low probability of fault occurence due to its small size), can start with the TMR architecture with simple voter. The rest of degradation strategy will be the same as in previous case. The ommiting of one step in degradation strategy for non key part of system will reduce the number of partial bitstreams to be stored and reduce the size of GPDRC (because its size is dependent on the number of error inputs).

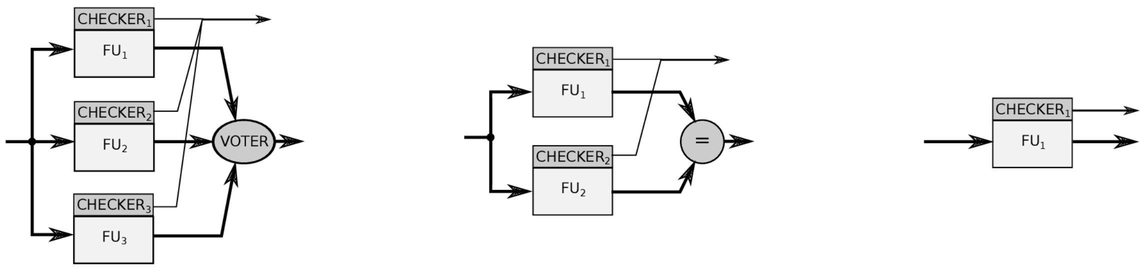


Figure 6.5: The set of FT architectures as a sample of degradation strategy

6.2.3 The allocation of implementation area for FT architectures

From the previous steps the set of important parts together with their chosen possible degradation strategies for permanent fault recovery was adopted for securing. From the reconfigurability point of view, the implementation area in FPGA can be divided into a dynamic reconfigurable area and a static area without possibility to be modified by PDR. The set of important parts of original system (chosen in previous steps) will be implemented in dynamic area to be able to be modified by PDR and the remaining parts will be placed in static area. For each chosen important part of system, several PRRs will be created. To these PRRs, the PRMs of currently used FT architecture will be assigned according to stated procedure.

The location and the size of PRRs for implementing one system part has to be chosen with respect to this conditions.

- The number of PRRs is the same or bigger than the number of PRMs of the starting (the most robust) FT architecture for given system part. When the number of PRRs is bigger than the number of PRMs, the remaining PRRs can serve as spares and it will increase the number of tolerable permanent faults. In this case, the degradation strategy has to be modified in that way, that in the case of permanent fault occurence, these spare PRRs to substitute the excluded PRR will be utilized first.

- The set of created PRRs will contain one specific PRR for PRM with routing (PRM_ROUTE). This PRR has to be located in the neighbourhood of all other PRRs. This means that there is a direct interconnection between this PRR and the other PRRs not utilizing resources from the remaining part of FPGA outside these two PRRs.

- Every PRR from the set of created PRRs (except of the PRR designated to be configured by PRM with routing) has to have the same size, the same structure and the same local placement of the FPGA resources. This requirement must be satisfied to be able to apply the bitstream relocation technique.

- The placement of PRR and also the size of the smallest possible PRR (PRR_{min}) is limited by the fact that the reconfiguration is done per configuration frames (see

Section 2.2.7). As the configuration frame is modifying the configuration of specified number of resources at once, the location and the size of PRR has to respect these principles and can only allocate resources corresponding to one PRR_{min} or its multiples.

The variants with different number of PRRs allocated for simple TMR architecture is shown in Figure 6.6.

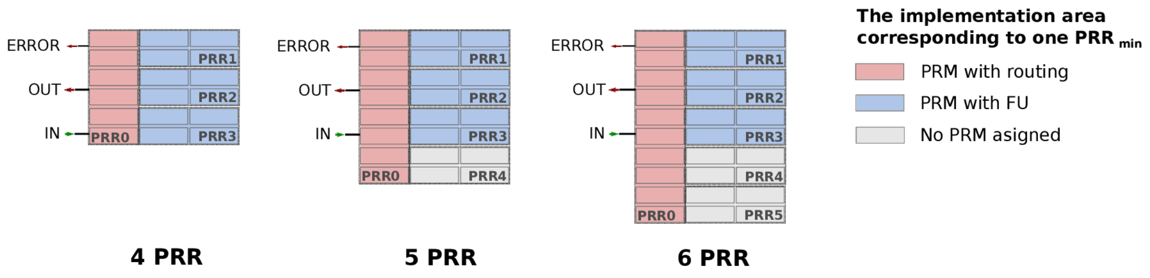


Figure 6.6: Several possibilities with area allocation for simple TMR architecture

6.2.4 The instantiation of GPDRC

The process of transient fault mitigation and recovery from permanent fault relies on GPDRC unit. This controller supplies the error signals from all PRMs of implemented FT architectures, it reads the configuration data from the external bitstream storage and controls the PDR process via instantiated ICAP interface. The GPDRC has to be placed into static part of the design because it cannot be reconfigurated by PDR (PDR is driven by GPDRC and it cannot reconfigure itself).

With the instantiantion of GPDRC, some external memory controller has to be instantiated too. The GPDRC is independent on the type of configuration bitstream storage and its controller. It only provides interface for configuration bistream read operation.

Alternatively, when the synchronization of replicated modules in some implemented FT architecture is needed, the separate synchronization controller can be instantiated and connected via the synchronization interface of GPDRC which provides information about the end of reconfiguration process and the index of reconfigured module (the one which has to be synchronized).

6.3 Design tool for automatic generation of fault tolerant architectures

The process of creating FT architectures based on the described methodology can be automated. This tool can faciliate the work of system designer who can focus on tasks such as design partitioning, chosing the degradation strategies or the optimization of implementation area allocations. The design tool is able to process the given VHDL entity and create another entity described in VHDL which will contain the definition of FT architecture where the input entity will be replicated to several functional modules. This output FT architecture will have the same interface as the original system with addition to output signals needed for GPDRC to be able to detect faults in its modules and perform their reconfiguration.

6.3.1 Inputs

In this paragraph, the required inputs for the developed design tool securing the given system entity are summarized.

1. The design of system described as an entity written in VHDL. This will allow to describe any system design from the simple entity with behavioral description to the top-level entity of complex processor.

2. The type of FT architecture to be created. It can be chosen from existing template or the new template.

3. The location of desired PRRs (placement constraints).

4. The implementation script for the given design written in Tool Command Language (TCL) for PlanAhead tool from Xilinx ISE Design Suite.

6.3.2 Outputs

In this paragraph, the outputs of developed design tool are summarized. These files are further used for implementing the secured system and for creating the partial configuration bitstreams.

1. The set of VHDL files with all entities of FT architecture.

2. The set of VHDL files with entity wrappers which have the required interface and can be used as PRMs.

3. The modified implementation script in the TCL format. This script contains the commands for the creation of PRR according to stated locations (in constraints file), their assignment by implemented entities and commands for starting runs which will create partial configuration bitstreams for all PRMs.

6.3.3 The process of FT architecture generation

In this section, the process performed by the developed design tool will be described.

The main principle of creating FT architecture from the input entity and a set of template files is shown in Figure 6.7. In this case, the building modules for TMR architecture with duplicated voter will be created. The use of generated PRMs to create the desired FT architecture is shown in Figure 6.8.

The input entity provided by designer is wrapped into new entity from which the PRM will be created. This PRM is referred to as Function Unit (FU) PRM as only this unit performs the original function of system. This PRM entity has input defined as a single input vector which was gathered by joining the inputs of original FU. The same gathering to single vector is done also with its outputs. This allows to define the templates of FT architecture modules in generic way as it can be seen in Figure 6.7 in case of PRM with voter.

Due to the fact that the interface of original FU should be retained in the final FT architecture entity, the template of routing for current architecture has to be wrapped. It is done in the opposite way, the joined input and output vectors for replicated FUs in FT architecture are scattered to form the same set of inputs and outputs as the original FU. Except these vectors, the interface of FT architecture also contains the error vector where

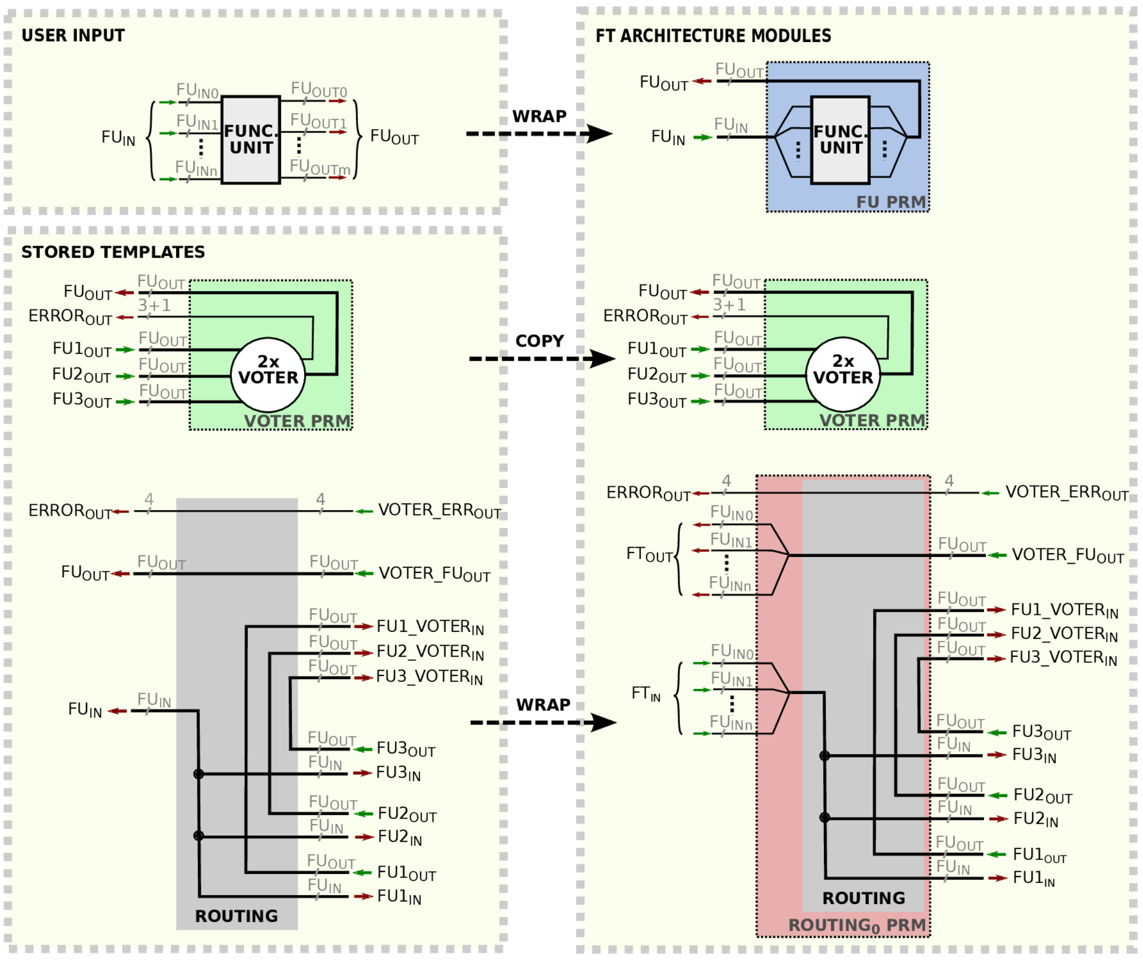


Figure 6.7: Generating FT architecture of Generation 0 from the unit designated for securing

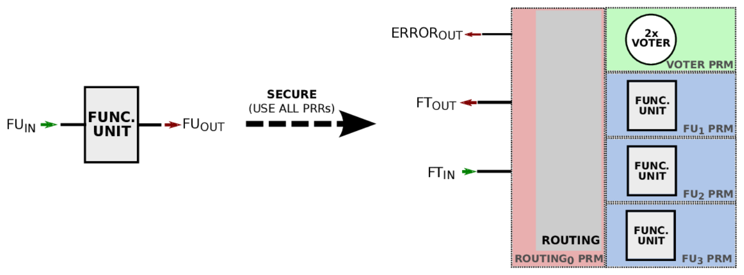


Figure 6.8: The scheme of final FT architecture with generated modules in Generation 0

each logic signal carries the information about error status of specific PRM to input error vector of GPDRC (see Section 5.2).

In Figure 6.9, the process of creating different FT architecture is shown. This architecture is less robust then the previous one and uses different routing scheme with the addition of simple voter. In this case, the voter is not secured against the impact of fault occuring in it. Therefore there is no designated PRM with voter, one PRR of the PRRs designated for the given FT architecture can remain unused. All variations of the FT architecture PRRs assignment are shown in Figure 6.10.

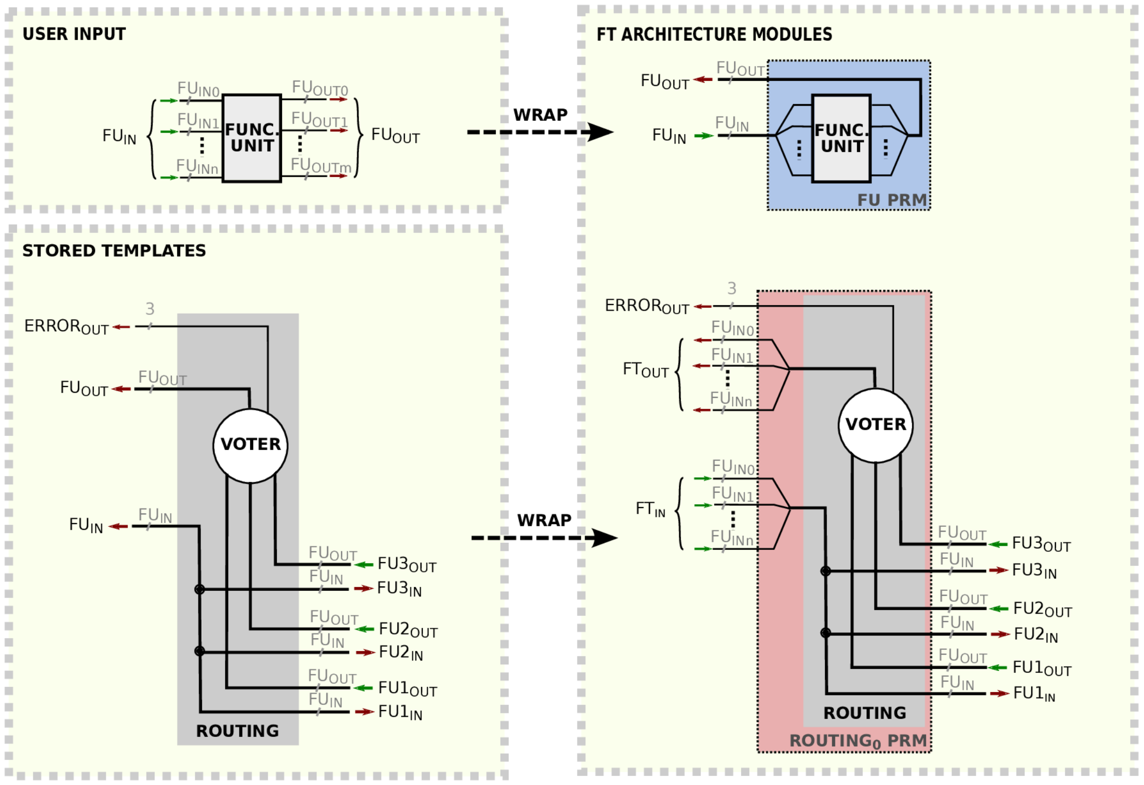


Figure 6.9: Generating FT architecture of Generation 1 from unit designated for securing

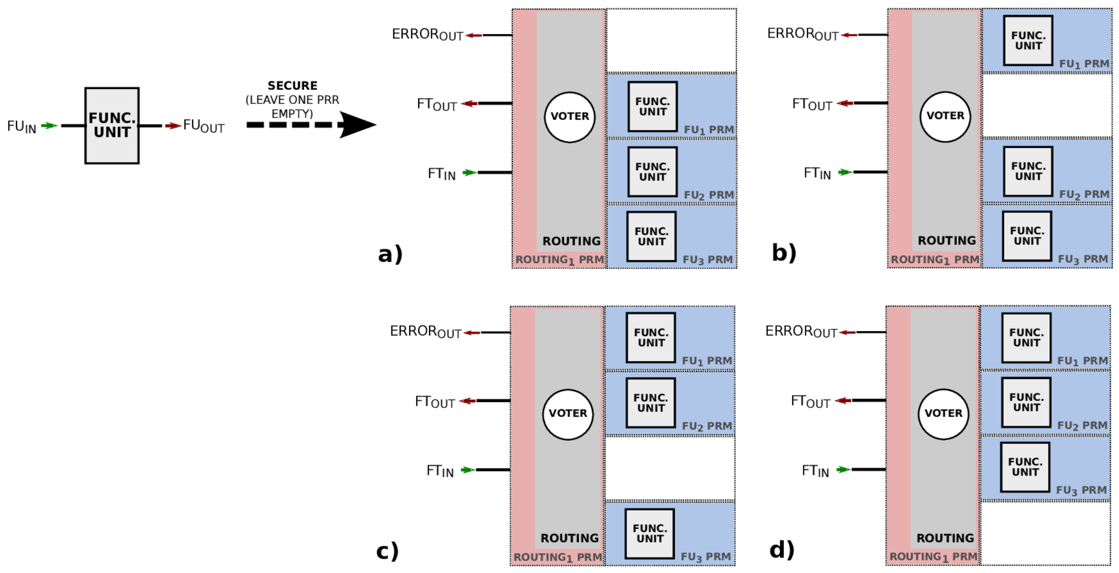


Figure 6.10: The scheme of final FT architecture with generated modules in Generation 1

6.4 The implementation of generated FT architectures

The complete process starting with the entry of unsecured system design to the final step of configuration of FPGA with the equipment to tolerate the fault impacts and their mitigation consists of several steps. These steps are shown in Figure 6.11.

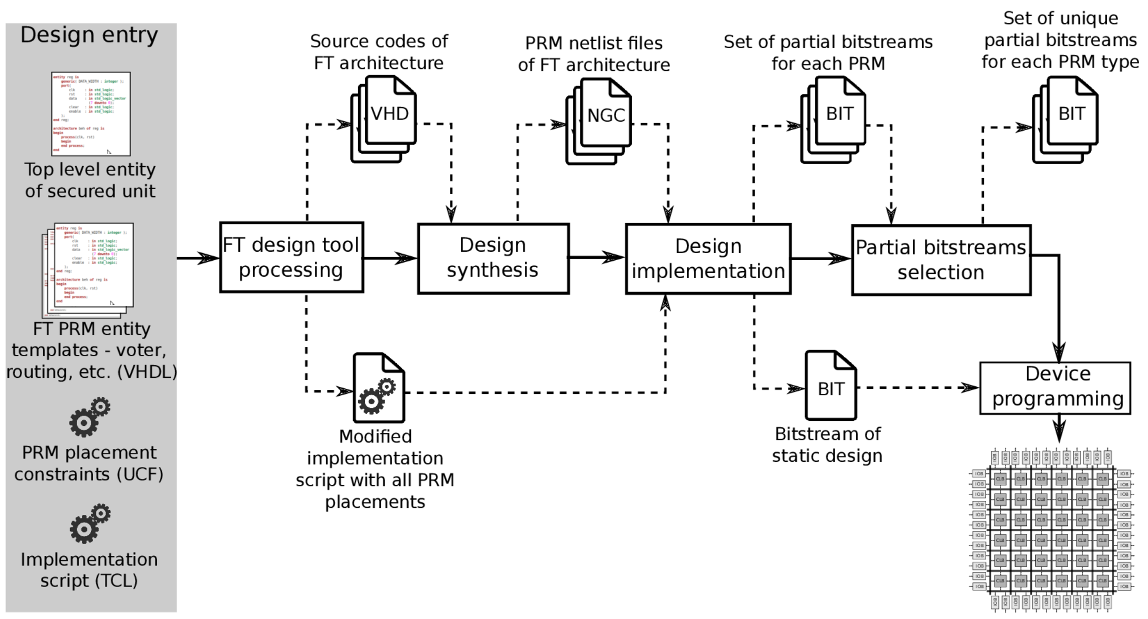

Design entry
Top level entity of secured unit
FT PRM entity templates - voter, routing, etc. (VHDL)
PRM placement constraints (UCF)
Implementation script (TCL)
Source codes of FT architecture
VHD
PRM netlist files of FT architecture
NGC
Set of partial bitstreams for each PRM
BIT
Set of unique partial bitstreams for each PRM type
BIT
FT design tool processing
Design synthesis
Design implementation
Partial bitstreams selection
Modified implementation script with all PRM placements
BIT
Bitstream of static design
Device programming

Figure 6.11: The process of generating FT architecture and its implementation for given system unit

The steps of this process have the following meaning:

1. Design entry - the designer describes the desired system in VHDL and sets his constraints for the design.

2. The specification of the system parts to be secured - the designer chooses the parts of the system which is implemented in entered VHDL files to be secured and chooses the degradation stategy for each of these parts. Each selected part has to be described as a single entity in VHDL and none of these parts can be included inside another of these parts.

3. The generation of FT architectures for the use in degradation strategies - The developed tool for designing system part as FT architecture is executed for each system part and each FT architecture from its degradation strategy.

4. The creation of secured FT system - the original system design is modified by replacement of selected parts by their implementations as FT architectures. This can be done without much effort because the interface of original part (unit) is a subset of the interface of the generated FT architecture. Further, the GPDRC instance has to be added and the error signals from all FT architectures have to be gathered and connected to its error input. The controller for some external memory device (e.g. the developed SD card controller) has to be added, too. This unit is needed to provide the configuration bitstream data for GPDRC. Alternatively, the synchronization

controller and logic to perform synchronization of the modules of FT architectures can be added as well in this step.

5. The implementation of static design with the starting configuration - for the complete (static) reconfiguration of FPGA, the system design where all chosen important parts are secured with most robust FT archictures from generation 0 is used. This implementation run is also used for generating partial bitstreams for all PRMs utilized by FT architectures in generation 0 (Xilinx implementation tool implements both static design and PRMs in one batch). From these partial bitstreams, one from each PRM type is chosen as golden copy to be stored in external memory storage. These bitstreams can be later relocated and used during fault mitigation process.

6. The implementation of all partial configuration bitstreams - to create partial bitstreams which can be used by GPDRC for recovery from permanent fault, PRBs for each PRM with routing for all possible alternative configurations in each FT architecture must be created. The number of needed implementation runs to create all neccessary PRBs does not depend on the number of implemented FT architectures but on the maximal number of alternative configurations existing for some FT architecture implemented in the system. This is caused by the fact that one alternative configuration is generated in each FT architecture per single implementation run.

   Table 6.1 shows the number of design runs for the FT architectures consisting of different number of PRMs (excluding PRM_ROUTE) and available PRRs to be assigned by these PRMs. Note that this table can be only used for the degradation strategies which ends with only one correctly operating PRM.

6.5 Summary

In this chapter, the process of the design of secured system by means of proposed methodology was described in a series of neccessary steps. Altough some steps such as the generating of FT architectures or the implementation of final design and partial bitstreams is automated, the most important tasks in the process must be done by a designer.

The original system design which is intended to be more secured against transient and permanent fault occurences must be divided into the set of entities (parts) which affect the overall system dependability at most and therefore they will be implemented as FT architectures. For these parts, the strategies based on degradation of FT architecture robustness are stated and the chosen FT architectures are generated by developed design tool. For each part, the area of FPGA for its implementation is allocated. The final secured system where the chosen parts are substituted with FT architectures implementing same function and which are stated in a degradation strategy is implemented in a series of implementation runs to acquire PRBs of all alternative configurations. The resulting set of PRBs contains only one implementation of each PRM type (except PRM with routing) and one PRB of PRM with routing for each alternative configuration in each FT architecture.

The output of this process is the bitstream for static configuration of FPGA which is done at the beginning and the set PRBs which are stored in external memory and used during fault mitigation process.

Design run [#]	3-PRM achitecture (3 PRRs) [config. code]	4-PRM achitecture (4 PRRs) [config. code]	5-PRM achitecture (5 PRRs) [config. code]
1 (static design)	[111] (Gen. 0)	[1111] (Gen. 0)	[11111] (Gen. 0)
2	[110] (Gen. 1)	[1110] (Gen. 1)	[11110] (Gen. 1)
3	[101] (Gen. 1)	[1101] (Gen. 1)	[11101] (Gen. 1)
4	[011] (Gen. 1)	[1011] (Gen. 1)	[11011] (Gen. 1)
5	[100] (Gen. 2)	[0111] (Gen. 1)	[10111] (Gen. 1)
6	[010] (Gen. 2)	[1100] (Gen. 2)	[01111] (Gen. 1)
7	[001] (Gen. 2)	[1001] (Gen. 2)	[11100] (Gen. 2)
8	- (don't care)	[0011] (Gen. 2)	[11010] (Gen. 2)
9	-	[0110] (Gen. 2)	[11001] (Gen. 2)
10	-	[1010] (Gen. 2)	[10110] (Gen. 2)
11	-	[0101] (Gen. 2)	[10101] (Gen. 2)
12	-	[1000] (Gen. 3)	[10011] (Gen. 2)
13	-	[0100] (Gen. 3)	[01110] (Gen. 2)
14	-	[0010] (Gen. 3)	[01101] (Gen. 2)
15	-	[0001] (Gen. 3)	[01011] (Gen. 2)
16	-	-	[00111] (Gen. 2)
17	-	-	[00011] (Gen. 3)
18	-	-	[00110] (Gen. 3)
19	-	-	[00101] (Gen. 3)
20	-	-	[01100] (Gen. 3)
21	-	-	[01010] (Gen. 3)
22	-	-	[01001] (Gen. 3)
23	-	-	[11000] (Gen. 3)
24	-	-	[10100] (Gen. 3)
25	-	-	[10010] (Gen. 3)
26	-	-	[10001] (Gen. 3)
27	-	-	[00001] (Gen. 4)
28	-	-	[00010] (Gen. 4)
29	-	-	[00100] (Gen. 4)
30	-	-	[01000] (Gen. 4)
31	-	-	[10000] (Gen. 4)

Table 6.1: Planning design runs to create all neccessary PRBs for FT architectures with different number of allocated PRRs

Chapter 7

Implementation and experimental results

This chapter describes the implementation results of systems where the methodology was applied to their design and implementation phase. It reveals the implementation specifics and the hardware overhead added to unsecured design or design secured by known approaches such as TMR implementation. Separately, the implementation results of GPDRC for different secured systems are examined.

For implemented FT architectures and the possibility of the secured system to reconfigure its implementation to a different one (using the stated degradation strategy), the dependability of the system is modelled by Markov models.

The ability to detect, localize and mitigate transient faults is examined in developed test platform where the SEU faults are simulated by configuration manipulation performed by external SEU injector. Finally, the permanent fault occurences in implemented system are simulated and the ability to avoid the use of faulty modules by reconfiguration to different FT architecture is tested.

All experimental systems were implemented in VHDL and synthetised and implemented by tools from Xilinx ISE Design Suite 14.7. The targeted FPGA was XC5VSX50T from Xilinx Virtex 5 family which is a component on ML506 development board.

7.1 Implementation of GPDRC

In the secured system design, a very important role is designated to GPDRC unit. The reason for its development as the alternative to controllers implemented into softcore processor is its smaller size and lower reconfiguration latency due to its specialization. Its size (the number of utilized FPGA resources) is mainly affected by the number of PRMs into which the system is implemented.

7.1.1 Generic implementation of GPDRC and its scaling

The GPDRC can be used in systems with different types of partitioning. To be able to instantiate it in different designs, it was developed as a generic unit with the possibility to define several of its attributes.

The number of system parts which are implemented as single FT architecture is denoted as $ARCH\_COUNT$. This is the most important generic parameter which affects the number of PRBs to be stored. The number of available PRMs in each FT architecture for implementing its units is denoted as *PRM_COUNT*. The number of all PRMs in system design and also the width of error signal vector entering the GPDRC is the product of these generic values. The GPDRC can handle different types of FT architecture with various PRM types. The number of specific PRM types is important to fault mitigation process as only one copy with each PRM type must be stored. To be able to choose the proper type of PRM, the width of the vector with type index must be stated. It can be counted as the square root of the number of PRM types and it is denoted as *PRM_TYPE_WIDTH*. The last generic parameter is the width of bitstream address vector (*ADDRESS_WIDTH*).

For the evaluation of GPDRC resource utilization results for different system partitioning approaches, a design with counters, registers, decoders and other logic was created. The complexity of this implemented system does not play any role in the evaluation of GPDRC size. It is mainly influenced by the overall number of PRMs and other attributes mentioned in above paragraph. Thus, the entire design in FPGA was divided into several FT architectures and they were divided into the same number of PRMs. The experiments were done for 3 to 6 PRMs. When the number of 5 or 6 PRMs per architecture was used, then the starting configuration of FT architecture was the one with TMR scheme with duplex voter. For 4 PRMs, the TMR scheme with simple voter was used and for 3 PRMs the duplex alternative with checker was chosen as the starting configuration. The size of GPDRC for various numbers of FT architectures and the number of PRMs is presented in Figure 7.1.

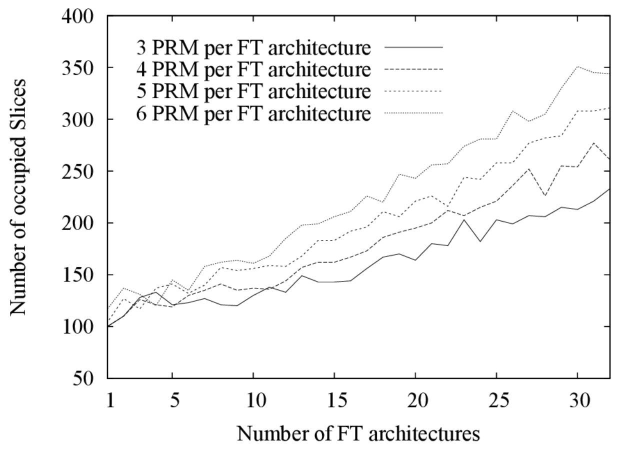


Figure 7.1: GPDRC size vs. the number of FT architectures for various numbers of PRMs per FT architecture

The size of GPDRC and its units together with the comparison with the size of MicroBlaze IP core used as PDR controller is shown in Table 7.1. These results are valid for 32 FT architectures with 6 PRMs per each controlled by the GPDRC. The meaning of the columns is as follows: the name of unit (column 1), the size of unit in slices (2), the number of occupied LUTs (3) and FlipFlops (4) and the size of TMR alternative (5).

7.1.2 The reconfiguration time of PRM

The time needed for the reconfiguration of one PRM is an important metric of the controller. In Table 7.2, the measured values for the GPDRC according to the PRM bitstream size are presented. The reconfiguration time is related to the size of PRB which reconfigures the PRM and the utization of the resources inside PRM does not influence this time. Thus, the

ML506 - Virtex5 **192 PRMs**	Size [slices]	LUTs [#]	F/Fs [#]	TMR [slices]
Input Capture Register	49 (0,6%)	97	192	127 (2,6x)
Actual Error Register	48 (0,6%)	101	101	124 (2,6x)
Previous Error Register	48 (0,6%)	192	192	124 (2,6x)
Hard Error Unit	3 (0,1%)	4	0	9 (3,0x)
Round Robin Unit	5 (0,1%)	6	6	14 (2,9x)
Error Encoder	3 (0,1%)	3	0	6 (2,0x)
Relocation Unit	7 (0,1%)	16	1	20 (2,9x)
Architecture Status Unit	2 (0,1%)	49	32	6 (3,0x)
Address Counter	22 (0,3%)	52	21	56 (2,5x)
FSM	22 (0,3%)	48	17	59 (2,7x)
Others (LUTs, MUXs...)	135 (1,7%)	317	186	414 (3,1x)
GPDRC total	**344 (4,2%)**	**885**	**748**	**959 (2,8x)**
MicroBlaze	**628 (7,7%)**	**1414**	**1491**	**1664 (2,8x)**

Table 7.1: The numbers of FPGA resources for GPDRC (32 FT architectures, 6 PRM per FT architecture)

table contains the multiples of the smallest possible PRM (the set of CLBs in one FPGA column and in the same FPGA row) which can be created in Virtex 5 FPGA and contains 20 CLBs. The meaning of the table columns is as follows: the multiple of smallest PRM (column 1), the size of PRM in CLBs (2), the size of PRM partial bitstream in kBs (3) and reconfiguration time in miliseconds (4).

ML506 - Virtex5 **multiples of the size of** **smallest PRM with CLBs**	CLBs [#]	Bitstream size [kB]	Reconfiguration time [ms]
1x	20	6,6	0,35
2x	40	12,7	0,67
3x	60	18,8	0,99
4x	80	25,0	1,33
5x	100	30,5	1,67
6x	120	36,5	2,02
7x	140	43,1	2,35
8x	160	49,5	2,74

Table 7.2: The reconfiguration time of one PRM according to its bitstream size

7.2 Evaluation of hardware overhead of FT architectures developed to secure a given part of system

This section presents the basic features of FT architectures which were developed for each generation (0, 1 and 2) together with the specification of their properties and constraints. Please note that these architectures serve as models for the description of the methodology application to some system. Different FT architectures which have the ability to detect and localize faults on PRM level can be used.

In this case, the proposed FT architectures utilize 5 PRMs and thus 5 error signals can be identified on the output of PRM_ROUTE block. These signals are connected to the inputs of GPDRC where they indicate the occurrence of a fault.

7.2.1 FT Architecture of Generation 0

The initial FT architecture of Generation 0 is based on TMR scheme in which the outputs of all FUs are checked by the majority element (voter). This architecture consists of 5 PRMs (3 PRM_FUs, PRM_VOTER and PRM_ROUTE). Figure 7.2 presents the proposed structure of this architecture.

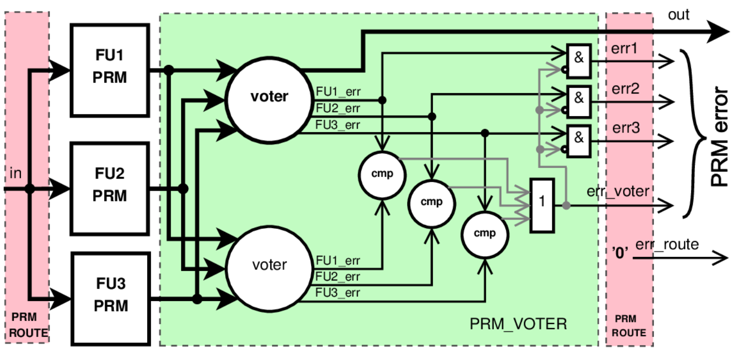


Figure 7.2: The FT Architecture of Generation 0 based on TMR

Each FU of the architecture is implemented as a standalone PRM without any additional diagnostic logic. The outputs of all PRM FUs are connected into PRM_VOTER block which is implemented as a PRM as well. PRM_VOTER block contains voter and additional diagnostic logic (e.g. comparators and logic gates) for the fault detection in FUs. The voter is implemented as a duplex architecture because of the need to detect fault occurrence in its structure, this information is used for the reconfiguration of PRM_VOTER block. Note that the voter architecture can be implemented also in two-rail logic instead of duplex architecture. The error outputs from PRM_VOTER block are processed by GPDRC which selects the appropriate bitstream from bistream storage and performs the reconfiguration of specific PRM. The PRM_ROUTE block provides the interface between PRMs in FT architecture and other parts of FPGA. In this architecture, the PRM_ROUTE block does not contain any additional diagnostic logic or FPGA logic elements, therefore this block is not protected against fault occurrence and the error signal *err_route* is set to logic zero value permanently.

Therefore, if a fault occurs in the detection logic of duplex architecture in majority element then the architecture may become inoperable. This threat is lowered to minimum due to the size of the logic which is much smaller than the size of other units.

During PRM_VOTER and PRM_ROUTE reconfiguration, incorrect values can appear on the outputs of FT system. They must be ignored.

7.2.2 FT Architecture of Generation 1

The FT architecture of Generation 1 is based on a duplex scheme with the addition of one PRM with CHECKER unit (PRM_CHECKER). As can be seen in Figure 7.3, this architecture consists of four PRMs (2 PRM_FU, PRM_CHECKER and PRM_ROUTE). Each FU of the architecture is implemented as a single PRM and their outputs are switched by output multiplexor which is controlled by error signal from diagnostic logic. The checker unit is implemented as a standalone PRM as well.

In this architecture, the PRM_VOTER block is missing so that the additional diagnostic logic and the output multiplexor are relocated into PRM_ROUTE block. Diagnostic

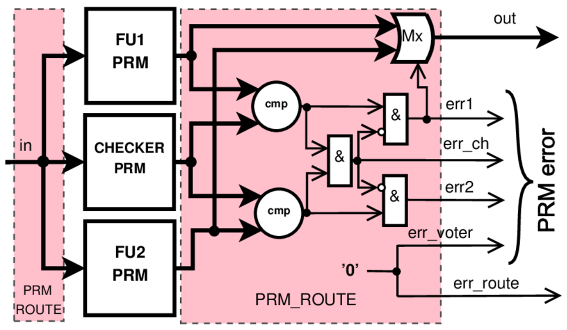


Figure 7.3: The FT Architecture of Generation 1 based on Duplex with checker

logic consists of two comparators and three AND logic gates, the area overhead is smaller than the area overhead in PRM_VOTER block of Generation 0. The outputs from units implemented in both PRM_FUs and PRM_CHECKER are compared by comparators and the output logic gates uniquely identify the faulty PRM. If a failure is recognized in PRM implementing the first FU then the output multiplexer is switched on the output of the second FU. In this architecture, the PRM_VOTER block is missing, therefore the error signal *err_voter* is set to logical zero value permanently. The error signal *err_route* is set to logical zero value permanently as well because in this architecture the possibility to detect any fault in the PRM_ROUTE block is not available.

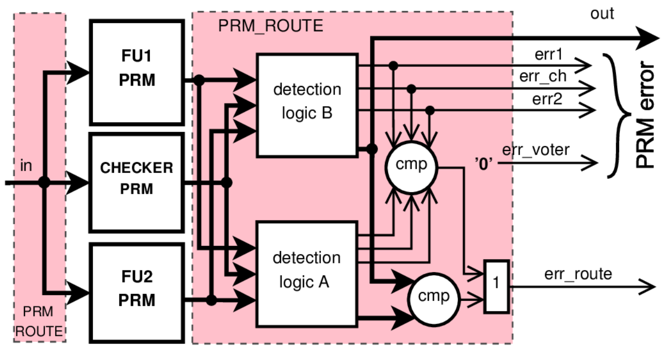


Figure 7.4: The alternative FT Architecture of Generation 1

In order to detect any fault in PRM_ROUTE block, this block is supposed to be implemented as duplex architecture with comparator. The alternative of FT architecture of Generation 1 can be seen in Figure 7.4. The comparator output is connected to error signal *err_route*, the occurrence of logical one value on error signal will cause the start of PDR process.

7.2.3 Non-FT Architecture of Generation 2

The final architecture of Generation 2 is based on classical duplex scheme. This architecture is not FT and it has only 3 PRMs (2 PRM_FU and PRM_ROUTE). Each FU of this architecture is implemented as a standalone PRM without additional diagnostic logic. The structure of this architecture can be seen in Figure 7.5.

As can be seen, PRM_ROUTE block with additional diagnostic logic for fault detection is included in the architecture. Because it is not known if the fault occured in one of the two PRM implementing FUs or in PRM with routing, the reconfiguration process is

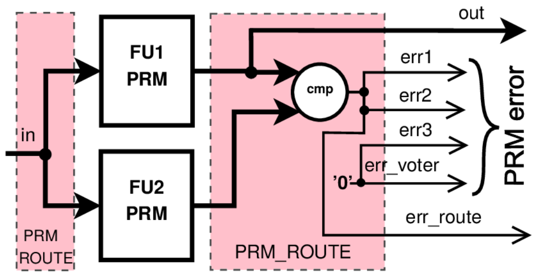


Figure 7.5: The architecture of Generation 2 based on Duplex

applied to both of FU PRMs or to PRM with routing. Therefore, the output from the comparator is connected to error signals *err1*, *err2* and *err_route*. In this architecture, the PRM_VOTER block and FU3 PRM block are missing, the error signals *err_voter* and *err3* are set to logical zero values permanently.

7.2.4 Evaluation of resource overhead

The sizes of FT architecture components which cause hardware overhead in FPGA are shown in Table 7.3. In this table, the overhead of only those units which were utilized and extended by our methodology when compared to the standard use of these units are taken into account. For the generation 0, the overhead includes the size of PRM_ROUTE and PRM_VOTER units. The sizes of any of three FUs were not considered into overhead as they are present also in the standard TMR scheme. The size of PRM_VOTER unit was decremented by the size of standard majority voter unit without the ability of faulty unit localization to get only the overhead caused by the use of our methodology. For both types of the generation 1 and for the generation 2, the overhead includes only PRM_ROUTE unit for the same reasons as for the generation 0. The meaning of the columns is as follows: column 1 - the width of each FU output in bits; column 2 - the overhead in slices in the generation 0 configurations; column 3 - the overhead in slices in the generation 1 configurations; column 4 - the overhead in slices in the generation 1 (variant with duplex logic) configurations; column 5 - the overhead in slices in the generation 2 configurations.

XC5VSX50T data width [bits]	Generation 0 [slices]	Generation 1 [slices]	Generation 1-variant [slices]	Generation 2 [slices]
2	12	5	12	1
4	22	11	24	2
8	36	17	39	3
16	68	31	68	7
32	126	57	122	12
64	206	111	210	23

Table 7.3: The overheads of Generations in slices

7.2.5 Modelling reliability of proposed FT architectures

This section contains the description of Markov models which can be further used in the evaluation of reliability for FT architectures which were described in previous sections (Section 7.2.1, 7.2.2 and 7.2.3). Due to their complexity and big number of states, they have been simplified to be able to draw them. The situation is explained and discused by each figure of Markov model.

In all graphs showing Markov models, the name of state is usually prefixed by S and followed by the number of correctly operating functional units. This number is followed by a set of characters, each character is indicating that the unit encoded under this character works correctly. The codes of units are as follows: voter unit (V), checker unit (C), GPDRC unit (G) or routing logic (R). For example, the starting state of the model from Figure 7.6 is S_3VG. It means that the FT architecture consists of 3 functional units, a voter and a GPDRC. Each unit which has error signal connected to error input vector of GPDRC can be distinguish as faulty and its state can be shown in Markov model. In addition to these units, the fault occurence in GPDRC can be modelled as the loss of repair ability of the secured system.

The oriented edges corresponding to failures caused by transient fault are labelled by λ and these caused by permanent faults by Λ. Both of these labels are followed by the symbol of the failing unit. As an example, in the model in Figure 7.6 the edge going from state S_3VG to state S_2VG with label $3\lambda F$ is showing the transition between these states in case that one of three functional units has been affected by transient fault. The oriented edges describing the transient fault repair are labeled by μ and edges designated to the process of the recovery from permanent fault occurence are labeled by M. Since the time needed for PDR depends on the size of configured PBR there are two different values of repair rate in graphs. The first rate μ is dedicated to the reconfiguration of one PRR from the set of relocatable PRRs where replicated FUs, checker or voter unit are implemented. These PRRs have the same sized PBR and thus the repair rate is the same for all of them. The second repair rate μR is dedicated to PRR where the routing is implemented (PRM_ROUTE).

The states in which the system is producing correct outputs are shown as circles and these ones which not are shown as rectangles. There are 3 special states connected with permanent fault recovery process. The *permanent fault (correct operation)* state is describing the situation when permanent fault appears in one unit but at the same time it is still able to produce correct outputs and a possibility to recover from this fault still exists. When the system is in the *permanent fault (failure)* state, it can still recover from the fault but it is not producing correct output in that moment. The special double circle shape is dedicated to the starting state of the FT architecture from the next generation. The entire state graph describing Markov model of this FT architecture can be imagined in this place instead of the *next generation* state. This simplification was not only used to simplify the state graph drawing but also to show that the final reliability of secured system by means of proposed methodology is influenced by the selection of each FT architecture in its degradation strategy.

To be able to model the reliability by the Markov model, several conditions have to be defined:

- The priority of reconfiguration (repair) by the GPDRC is as follows from the biggest priority: PRM with routing, PRM with voter, PRM with checker and PRM with functional unit.

- The fault type (transient or permanent) is determined by the GPDRC according to the success of the first repair of affected PRM. If it fails, the occured fault is considered as permanent.

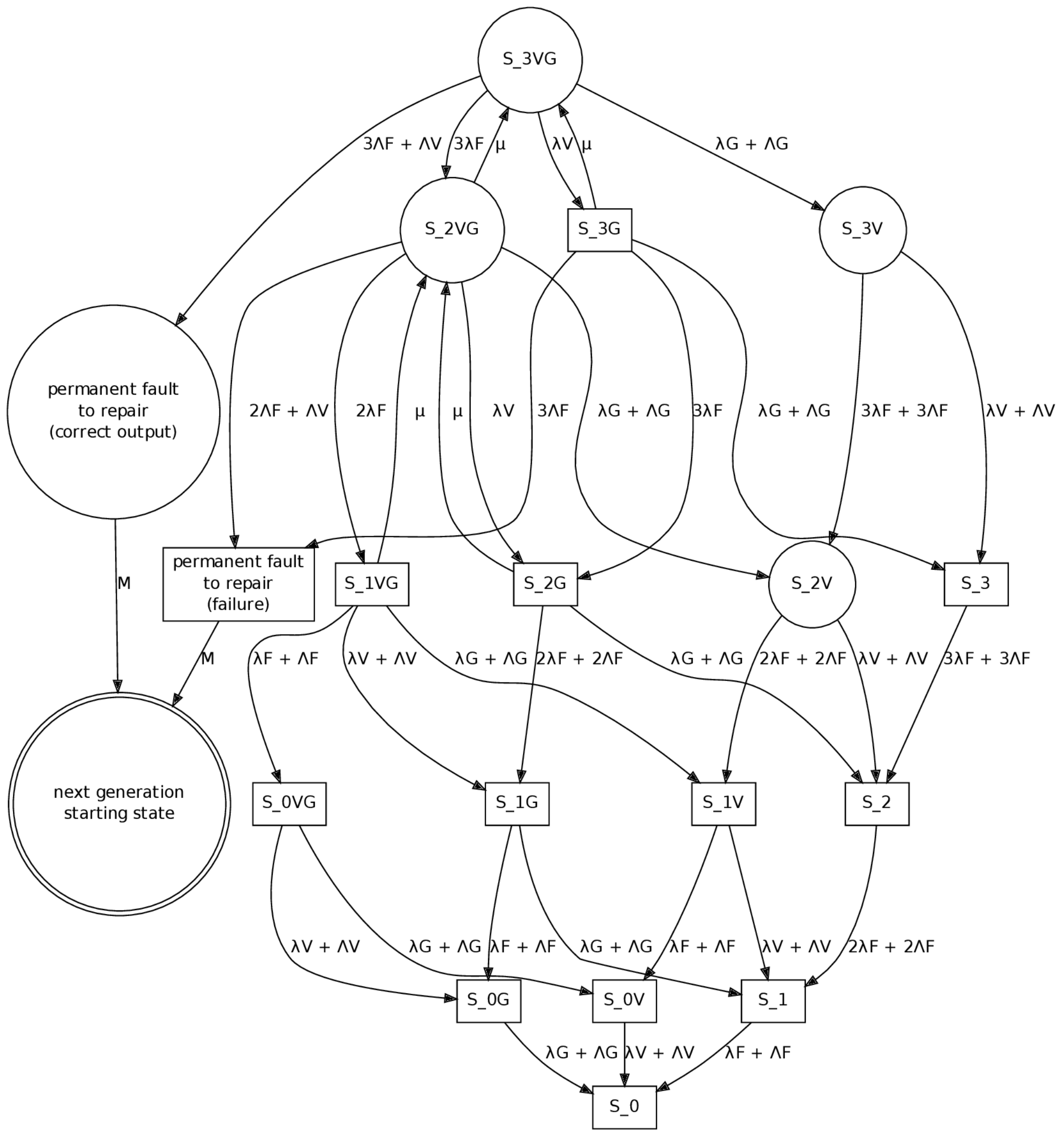


Figure 7.6: The simplified Markov model of the FT architecture of the Generation 0 with TMR scheme

- The GPDRC will not trigger the repair process if it is meaningless. This means that the repair would not bring the system to the state where it will produce correct outputs or to the state where further repair to bring the system to this correct state will be possible.
- Any repair is possible when the GPDRC is affected by a fault.

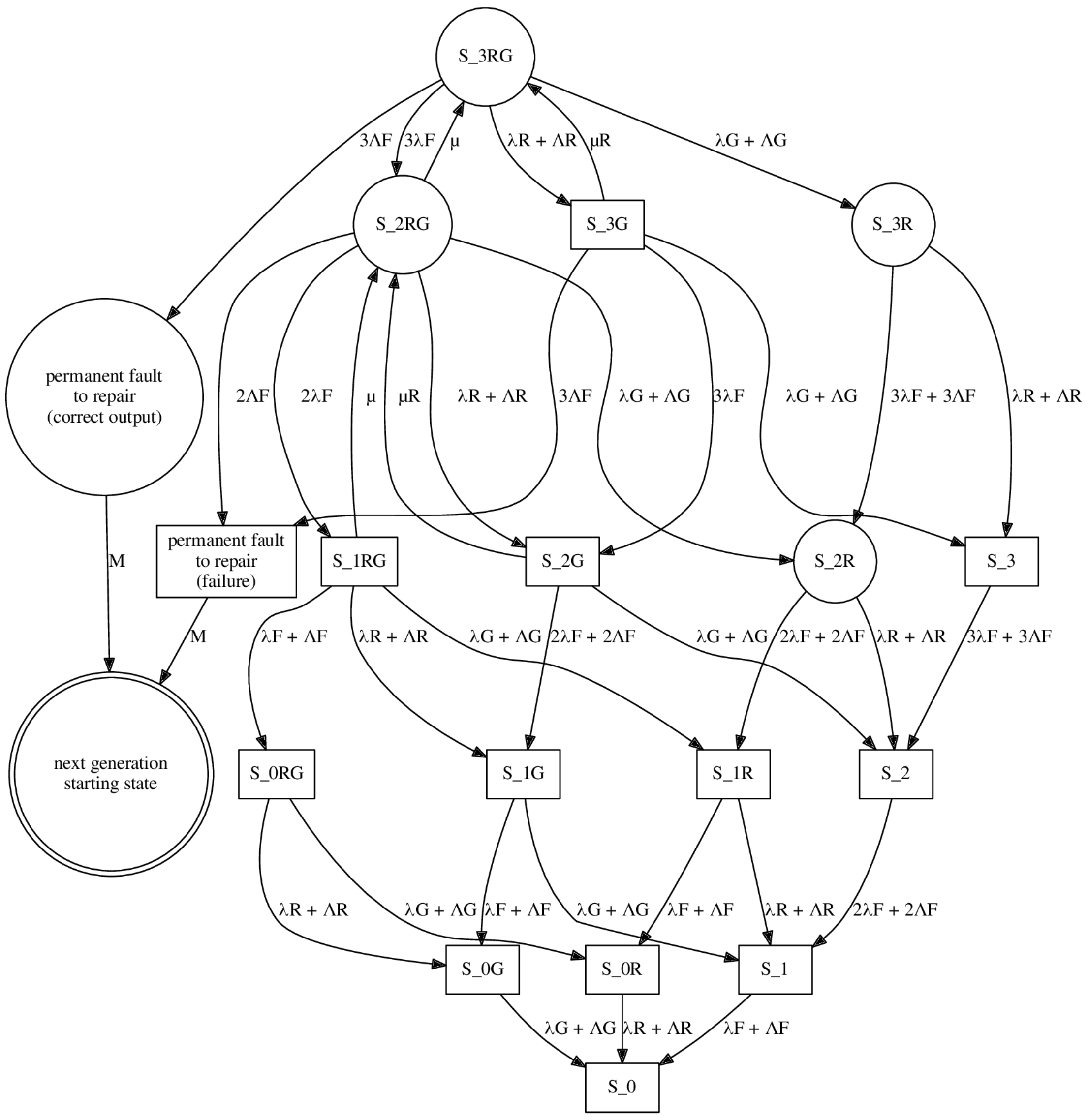


Figure 7.7: The simplified Markov model of the FT architecture of the Generation 1 with duplex scheme with simple voter unit

The simplified Markov model for FT architecture from generation 0 based on TMR architecture with doubled voter unit (see Section 7.2.1) is shown in Figure 7.6. As the correct states are considered these ones which certainly produce correct outputs. To ensure this, at least two functional units and the voter have to be working without errors. This state graph was simplified by not considering the faults occuring in PRR implementing routing between modules. If these states are shown, the overall number of states will be approximately doubled. Due to the fact that the faults in routing are considered as undetectable, the recovery from this state is not possible. On the other hand, the failure intensity of this module (λR) is negligible small since the routing is defined by very few configuration bits setting just several PIPs.

The simplified Markov model for FT architecture from generation 1 based on duplex architecture with checker unit and voter unit implemented in the same PRR as the routing (see Section 7.2.2, the alternative version) is shown in Figure 7.6. In Figure 7.4 there are 3 types of PRMs: PRMs with FU, PRM with checker unit and PRM with routing and voter (compare) logic. As there is no difference (for the modelling of dependability) between checker unit and function unit (it does not matter if error apears in FU or checker unit), these units are counted together in state label (e.g. state S_3 describes situation when both FUs and checker unit are working correctly). This simplification was used to avoid unnecessary increase of the number of system states. Final model is quite similar to the previous one with TMR but it differs in the toleration of permanent fault occurence. In this architecture, the voter logic is implemented in PRR with routing and it is not possible to recover the system when a permanent fault appears here. The increase of the utilized resources in this PRR is also the reason to explicitly model the fault occurences in it.

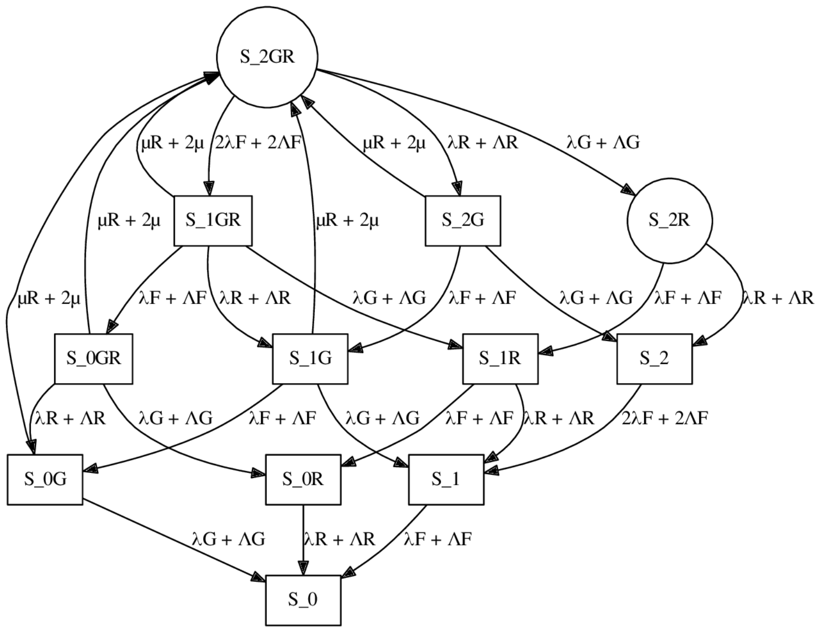


Figure 7.8: The Markov model of the FT architecture of the Generation 2 with simple duplex scheme

The Markov model for FT architecture from generation 2 based on simple duplex architecture with simple compare logic (see Section 7.2.3) is shown in Figure 7.8. Since there is no way to distinguish in which PRR the fault appears, each fault triggers the reconfiguration of all PRMs. No permanent fault repairs are considered in this model.

7.3 Implementation results of different approaches to the partitioning of original system

The key step in design process of securing a given system is its partitioning into parts which will be implemented as standalone FT architectures. To examine the properties of a secured system such as hardware overhead or the size of PRBs used for the reconfiguration after fault occurence for different types of its partitioning, the test design of system with MB-LITE softcore processor (see [31]) was developed.

The MB-LITE processor is the light-weight implementation of MicroBlaze processor [65]. The MicroBlaze processor is a RISC architecture with 32-bit wide instruction and data words. It is also based on MIPS architecture and implements a pipeline with 5 stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM) and Write-back (WB). Due to this pipelining in the processor, the most instructions have a latency of one cycle. Thanks to additional prefetch buffer, the MB-LITE processor has reduced the rate of instructions which can be fed into it. Therefore, the execution time was dropped by 10% when compared with MicroBlaze [31]. The design with MB-LITE was chosen for the reason that thanks to its structured and straightforward design it can be easily partitioned in different ways.

The test design consists of MB-LITE processor connected to Wishbone bus by Wishbone adapter as shown in Figure 7.9. Altough the top-level design contains only two main units - the instance of MB-LITE and Wishbone adapter, the processor can be further divided into 4 functional units - IF, ID, EX and MEM.

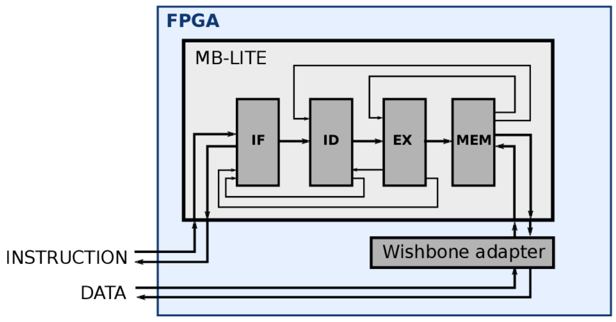


Figure 7.9: The original system design

This design was able to achieve the operating frequency of 229 MHz in Virtex 5 (XC5VSX50T) FPGA. The implementation results for the same FPGA used for the decision about the partitioning of design is shown in Table 7.4. The meaning of the table columns is as follows:

XC5VSX50T Design FU	Slices # (% of design)	LUTs # (% of design)	F/Fs # (% of design)
MB-LITE - IF	20 (3%)	45 (3%)	16 (4%)
MB-LITE - ID	94 (13%)	185 (12%)	191 (50%)
MB-LITE - EX	532 (71%)	1203 (76%)	93 (25%)
MB-LITE - MEM	55 (8%)	110 (7%)	44 (12%)
Wishbone adapter	21 (3%)	35 (2%)	33 (9%)
$\sum$	723 (9% of all in FPGA)	1579 (4% of all in FPGA)	377 (1% of all in FPGA)

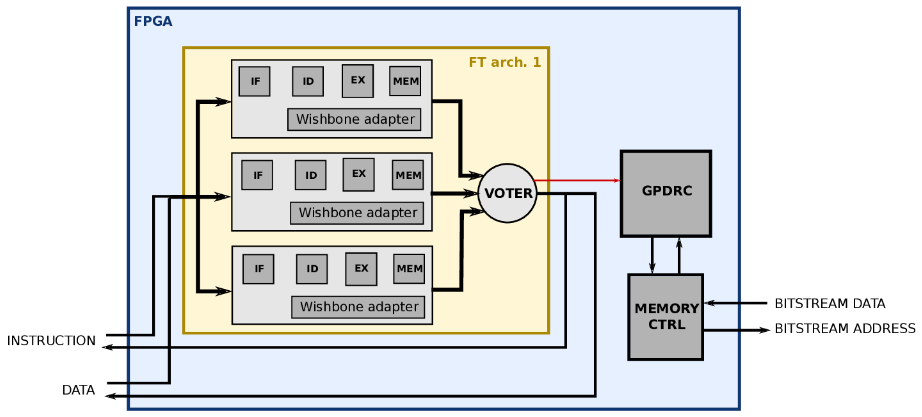
Table 7.4: The resource utilization of original system with MB-LITE processor and Wishbone adapter

the unit of design (column 1), the size of unit in slices (2), LUTs (3) and flip-flops (4).

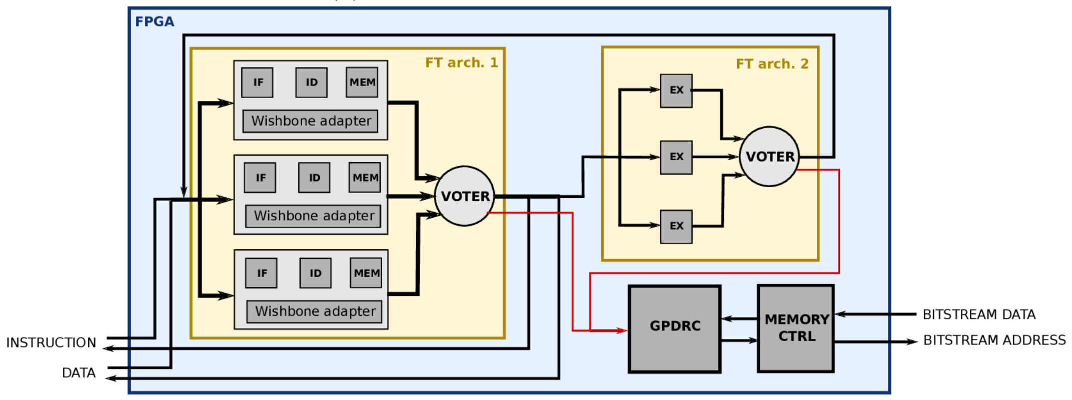
The implementation results show that the unit performing the execute stage of pipeline (MB-LITE - EX) utilizes much more slices than other units which is caused by a big number of used LUTs. Therefore, the following possibilities for partitioning based on different granularity were proposed:

- 1 FT architecture - all functional units are grouped together and replicated (coarse-grained partitioning), see Figure 7.10a.
- 2 FT architectures - EX unit of MB-LITE processor is implemented as one FT architecture, the remaining units are grouped together and implemented as the second FT architecture, see Figure 7.10b.
- 5 FT architectures - each unit mentioned in the above provided table is implemented in a single FT architecture (fine-grained partitioning), see Figure 7.10c.

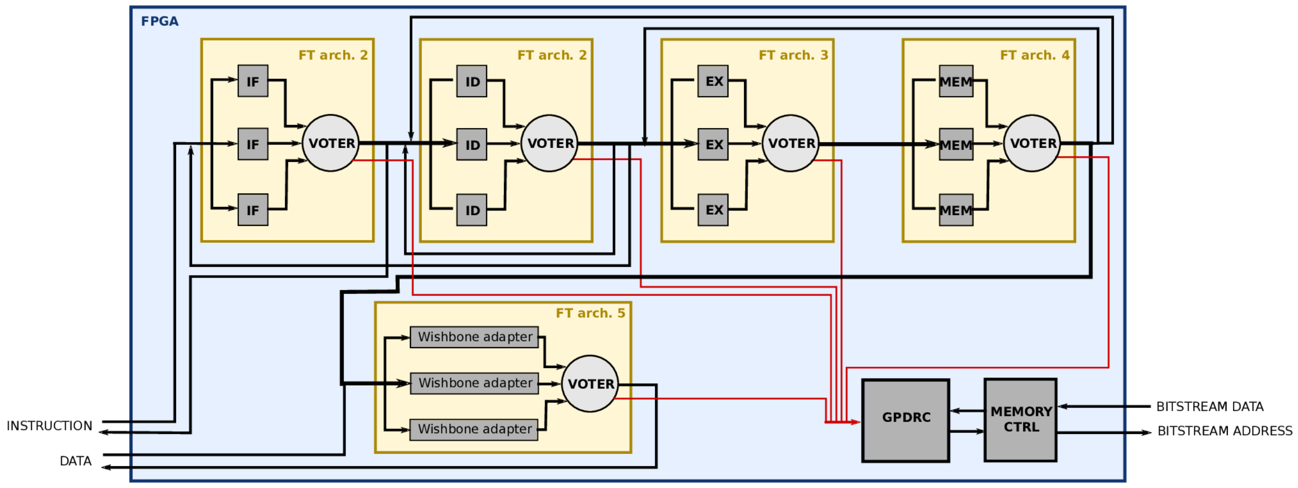
Table 7.5 shows the implementations of all variants of the secured system. They were compared by their resource utilization (column 1), hardware overhead in comparison to original design (column 2) and the sizes of their PRBs (column 3).



(a) All units in one FT architecture



(b) The ID unit of MB-LITE in standalone FT architecture



(c) Each unit in single FT architecture

Figure 7.10: The variants of differently partitioned secured systems

XC5VSX50T The variant of secured system	PRMs \#	Slices \#	HW overhead %	Bitstream sizes [kB]
Original design	0	723	0	-
TMR design (without PDR)	0	2287	216	-
1 FT architecture	5	2421	235	*PRM_ROUTE*: 6,6 *PRM_FU*: 408
2 FT architectures	10	2484	244	*PRM_ROUTEs*: 6,6; 6,6 *PRM_FUs*: 92,4; 39,6
5 FT architectures	25	2572	256	*PRM_ROUTEs*: 6,6; 6,6; 6,6; 6,6; 6,6 *PRM_FUs*: 6,6; 19,8; 92,4; 13,2; 6,6

Table 7.5: The comparison of resource utilization and hardware overhead for different implementations of the given system

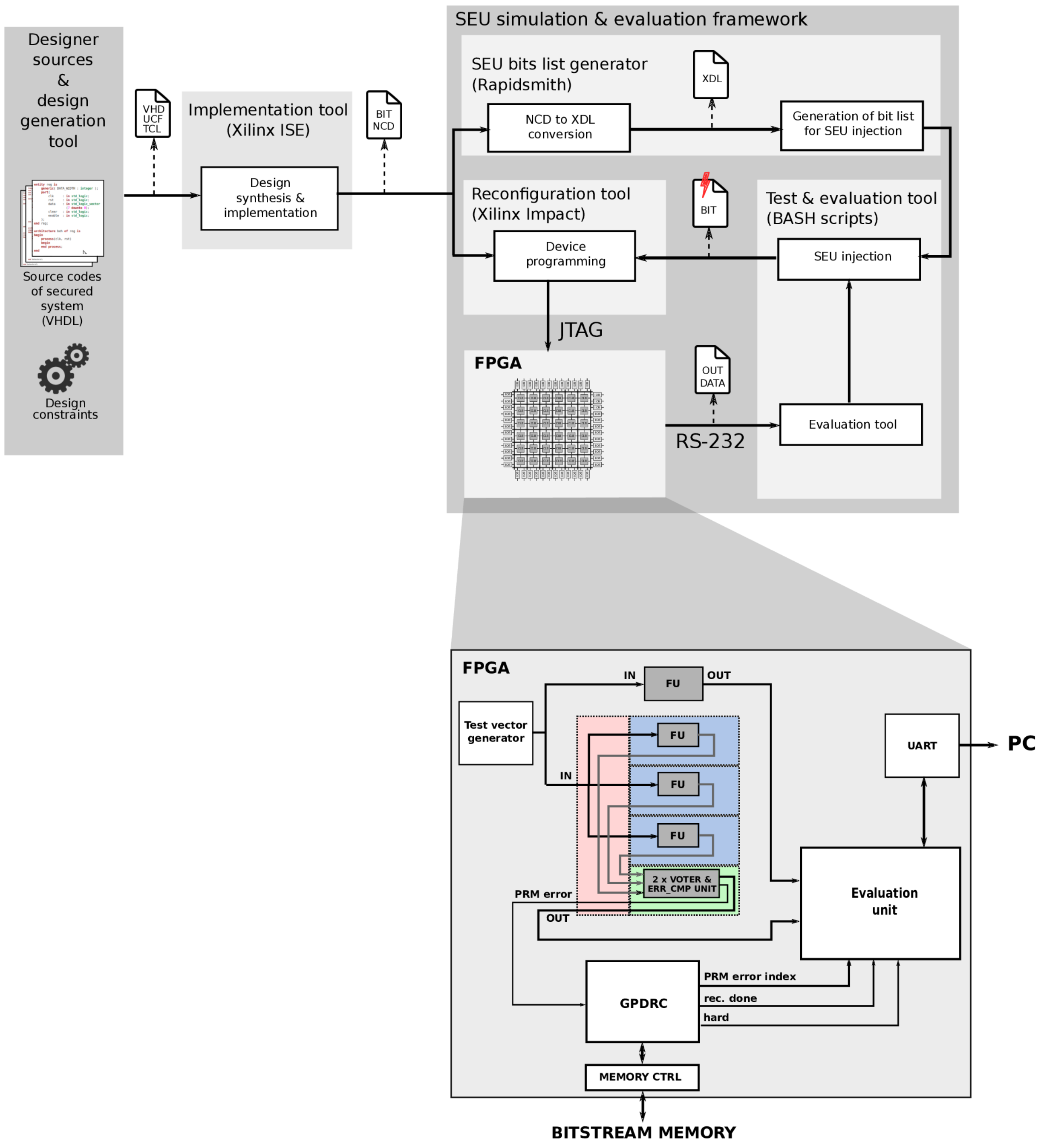
The results summarized in the table show that the HW overhead is slightly lower for the variant with 1 FT architecture. This is caused by the smaller GPDRC unit due to lower number of PRMs. On the other side, this is degraded by bigger PRB size which causes longer reconfiguration time. The table shows that there could be a tradeoff between HW overhead and the overall size of all PRBs (and the time of reconfiguration).

7.4 SEU testing platform for the evaluation of FT system design by means of methodology principles

To evaluate the quality of secured FPGA to cope with transient and permanent fault occurence, the special test platform was developed. The testing was based on fault injection into configuration bitstream to simulate an SEU fault occurence. The platform allows to observe the behaviour of entire secured system implemented in FPGA when a fault occurs. All components of the test platform are shown in Figure 7.11.

7.4.1 The implemented test and evaluation equipment

The test platform contains several parts which are creating together the necessary test and evaluation equipment. The FPGA is configured by the implementation of system secured by



Designer sources & design generation tool
Source codes of secured system (VHDL)
Design constraints
VHD UCF TCL
Implementation tool (Xilinx ISE)
Design synthesis & implementation
BIT NCD
SEU simulation & evaluation framework
SEU bits list generator (Rapidsmith)
XDL
NCD to XDL conversion
Generation of bit list for SEU injection
Reconfiguration tool (Xilinx Impact)
BIT
Device programming
Test & evaluation tool (BASH scripts)
SEU injection
JTAG
FPGA
OUT DATA
RS-232
Evaluation tool
FPGA
IN
FU
OUT
Test vector generator
FU
IN
FU
FU
2 x VOTER & ERR_CMP UNIT
PRM error
OUT
UART
PC
Evaluation unit
GPDRC
PRM error index
rec. done
hard
MEMORY CTRL
BITSTREAM MEMORY

Figure 7.11: Dependability evaluation platform with SEU injection test platform

the means of the methodology. The remaining parts of the test platform are implemented and run on PC.

The FPGA device under test (DUT) contains these units:

- The functional unit (FU) - The unit which is implementing the given function (the same function as original unsecured system).

- The FT architecture - The fault tolerant implementation of original system. It is implemented according to methodology principles with error output for each of its PRM.
- The test vector generator - This unit generates inputs for tested FT architecture as well as for the standalone FU.
- The GPDRC unit - The controller unit driving the fault mitigation process and the PDR.
- The memory controller - The SD card controller which is creating an interface between the GPDRC and SD memory card with stored partial bitstreams.
- The evaluation unit - In this component the outputs from FU and FT architecture are gathered and together with GPDRC status information they are sent to PC for further analysis.
- The UART controller - The unit interfacing the evaluation unit and serial interface of FPGA.

The test and evaluation equipment in PC consits of several tools:

- The FPGA reconfiguration tool - The script using Impact tool from Xilinx ISE toolkit.
- The XDL conversion tool - The tool for the conversion of the fully implemented design from NCD to XDL format.
- The PRM bit-list generator - The application based on RapidSmith framework for generation of the list of bits to be injected (reversed) by SEU injector.
- The SEU injector - The tool for manipulating the configuration. The SEU is simulated by making bit-flip. The process of injection is described in following section.
- The evaluation tool - The script collecting the results from DUT for further processing.

7.4.2 Process of SEU faults injection

For the SEU injection into configuration memory, the external SEU injector presented in [32] was used. It uses PDR to simulate the radiation-induced upsets by artificially changing the contents of the configuration memory. This injector is written as TCL script and is run on PC. It accesses the JTAG external reconfiguration interface of FPGA. It uses the ChipScope library function of Xilinx ISE toolkit to perform the upset in configuration memory by toggling some bit value in the configuration bitstream.

Due to the fact that the relation between the FPGA resource and the configuration bits which configure its settings is not known in general, all bits in PRM should be tested by injecting the SEU into it. In the Department of Computer System at Brno University of Technology, the relation between utilized LUT resources and configuration bits for Virtex 5 was uncovered. According to this fact, the application for bit-list generation was extended to be able to generate the list of configuration bits which is used to set the function of utilized LUTs. Although the SEU injection to only these bits is not sufficient for complete test of unit dependability, it can be used to reduce the time of SEU injection campaign to test the detection, localization or fault mitigation ability of the secured system.

One simulation step for testing a secured design consists of one SEU injection into one of FT architecture PRMs and checking its output for error. When the fault is detected by the compare logic in the evaluation unit or by detection and localization logic implemented in FT architecture, the status message is sent via RS-232 to the evaluation tool in PC. If the reconfiguration is performed, the GPDRC status is observed and after *rec. done* signal is set, the next status message is sent to the evaluation tool.

7.4.3 Experimental results of GPDRC transient fault mitigation process

The test platform described in the above section was implemented and tested with Virtex 5 FPGA (XC5VSX50T) on an ML506 development board. To implement the system design for FPGA and for bitstream generation, the Xilinx ISE 14.7 toolkit was used. The FU contains several 8-bit counters, decoders and multiplexers, the data width of input was 6 bits and the data width of output was 16 bits. The design contains one FT architecture with 5 PRMs and the FT architectures described in Section 7.2 was used in the degradation strategy. The size of a PRB for PRMs with FU, PRM with doubled voter and PRM with checker unit was 6632 bytes, the sizes of PRBs for each PRM with routing were 26582 bytes.

The meaning of the columns in Table 7.6 is as follows: column 1 - the type of PRM; column 2 - the utilization of PRM; column 3 - the number of the detected SEUs in FU; 4 - the number of SEUs detected by checkers in FUs; column 5 - the number of incorrect data on the outputs of FT architecture; column 6 - the number of missed SEU faults by the detection logic of FT architecture, column 7 - the number of successfully performed reconfigurations of PRM performed by the GPDRC.

XC5VSX50T PRM type	PRM utiliz. %	SEU injected in PRM #	SEU detected by FT arch. #	FT arch. output data errors #	SEU missed by FT. arch #	GPDRC reconf. #
PRM with FU (all generations)	45%	47232	7806	552	25	7826
PRM with 2xVOTER (generation 0)	30%	47232	3345	1571	105	3345
PRM with CHECKER (generation 1)	45%	47232	6901	552	11	6900
PRM with routing (generation 0)	1%	188928	0	21	21	0
PRM with routing (generation 1)	12%	188928	3542	1243	234	3541
PRM with routing (generation 2)	6%	188928	2432	1056	351	2430

Table 7.6: The number of detected SEUs in FUs of the architecture

From the results, it can be seen that the FT architecture of generation 0 and 1 can detect and repair more than 97% SEUs in PRM with FU, voter or checker unit. Except the FT architecture from generation 0 which do not have ability to detect faults in PRM with routing, this PRM type in the FT architectures from other generations was able to detect faults in more than 85% cases. Almost all detected faults have triggered the mitigation process done by GPDRC. In all cases, the PRM with routing was able to survive most of the SEU faults injected inside it due to very low utilization of FPGA resources.

7.4.4 Testing and evaluating recovery from permanent fault occurence

The developed evaluation platform was also used to test and evaluate the process of recovery from permanent fault occurence. For its simulation, the above described fault injector was used again. According to the results of SEU injection campaign during transient fault simulation process only some configuration bits of FT architecture PRMs were used for permanent fault simulation. A specific bit was chosen for permanent fault injection campaign, if the fault that it creates in the unit implemented in PRM was detected by the detection logic of FT architecture or it has been manifested as an error on FT architecture output during the transient fault simulation campaign.

The process of the simulation consists of these steps:

1. The first (next) bit from the set of SEU sensitive bits of FT architecture PRMs is taken and inverted.

2. When the GPDRC finishes the reconfiguration and indicates the end of reconfiguration by setting *rec_done* signal, the same bit is inverted again.

3. The GPDRC should localize the fault again and determine it as permanent fault and start the permanent fault recovery process. This action can be identified by setting *hard_error* sig while the index of PRM with the fault can be observed on the *PRM_index* signal. The operation of all units in PRMs are stopped during this step.

4. The end of a permanent fault recovery is indicated by the combination of active *rec_done* and *hard* signals and the set values of *arch. index* and *PRM error index* vectors.

5. The impulse on *rst_sig* is done to reset and synchronize the reconfigured FT architecture.

6. The output of FT architecture is compared with the stored correct value and the result is sent via UART to the evaluation tool.

7. When the simulated permanent fault is not repaired the starting FT architecture design is configured back to the appropriate PRMs.

The experimental results for a permanent fault injection campaign to the same FT architectures as in previous experiment are shown in Table 7.7. The meaning of the columns in the table is as follows: column 1 - the type of FT architecture and which generation it belongs to; column 2 - the number of injected SEU faults; column 3 - the number of incorrect data on the outputs of FT architecture; column 4 - the number of permanent faults detected; column 5 - the number of performed permanent fault recoveries by the reconfiguration to a different FT architecture; column 6 - the mean time to repair the system to the correctly operating state.

The results summarized in Table 7.7 show that the number of detected faults is decreasing with the number of PRMs which are used by FT architecture. This is caused mainly by masking the faults which are injected into excluded PRMs. The repair process is shorter for less robust FT architectures due to the fact that the reconfiguration of fewer PRM is performed.

XC5VSX50T Generation	Injected faults \#	FT arch. output data errors \#	Permanent fault detected \#	Recovery done \#	MTTR [ms]
Generation 0 (TMR-2xVOTER)	12768	2144	12515	12480	512
Generation 1 (TMR-simple)	12575	1796	12287	9802	351
Generation 2 (Duplex)	7987	708	156	0	-

Table 7.7: The number of successfully detected permanent faults and the MTTR for permanent fault recovery

7.5 Summary

This chapter summarized the implementation results of the securing system according to the methodology. The possible implementation of FT architecture was presented in details and the hardware overhead was compared with the TMR solution. For these FT architectures used in the proposed fault recovery mechanism, the Markov models for the evaluation of the system dependability were derived. The experiment with the partitioning the system with pipelined microprocessor was done to compare three different approaches in terms of hardware overhead and the size of all necessary PRBs which have to be stored. The last experiments were focused on testing the given system by injecting SEU faults by external SEU injector and evaluating the fault mitigation ability of the proposed method in the developed test and evaluation platform.

Chapter 8

Conclusions

In this work, the methodology of FT system design with the ability to mitigate transient faults caused by SEUs and to recover from several permanent fault occurences was proposed as the alternative to existing methods or methodologies. This methodology benefits from the ability to PDR in modern FPGAs which can be used for the run-time repair or the change of current FPGA configuration. The production of correct outputs from the system implemented in FPGA even during its PDR is ensured by its designing as an FT architecture.

At the beginning of this thesis, the basic principles, technologies and methods important for the methodology proposal are described. As the FPGAs themselves, their manufacturing process and technologies around them are still evolving very fast, some principles and techniques can become early outdated and overcomed. In this work, this part served to describe the state-of-the-art and the motivation at the beginning of the work on the methodology.

The main part of this paper presents the basic principles of the proposed methodology. The methodology was developed to satisfy the conditions stated in the goals of the research part. It is based on the designing of FT system into SRAM-based FPGA with the use of PDR and limited into specified implementation area. The next chapter then describes the entire procedure of designing this FT system step by step from the entering the original system design and finishing with the generation of the configuration bitstream of implemented FT system design which is ready to configure the FPGA device. Together with this bitstream, the set of PRBs which are designated to be stored in external memory is generated. This final system implementation is then ready to survive many transient and several permanent fault occurences.

At the end of the thesis, the implementation and experimental results are presented.

8.1 Benefits of this research

As the main benefit of this research, the proposal of alternative methodology for the FT system design with the ability of fault mitigation can be mentioned. This methodology brings some new features such as the use of dedicated reconfiguration controller or the application of the relocation technique in transient fault mitigation and also in the recovery process from permanent fault occurence. This greatly suppresses the main disadvantage of the use of precompiled configurations to mitigate faults which is space demanding storing of many configuration bitstreams.

The benefits of the use of the proposed methodology for FT system design can be summarized into several points.

- In the proposed methodology, the exact procedure of transformation the system design entered by designer to secured system where selected important parts are implemented as FT architectures with the mechanism of transient fault mitigation and recovery from permanent fault was described. The standard PR design flow defined by Xilinx as well as its standard design and implementation tools were used and thus the methodology is applicable to system design for all its FPGA families adopting the same design flow.
- The key component for this methodology, the dedicated reconfiguration controller (GPDRC) with the ability to determine the type of fault and perform the correct mitigation procedure, was developed. It was designed with the effort to reduce the necessary area and performance overhead. The overhead of final GPDRC design was compared with other alternative controller.
- When the system is designed by the means of the proposed methodology, it is possible to define the level of importance for every system part (which is designated to be secured) by specifying the degradation strategy. This strategy is used when the recovery after permanent fault occurence is performed. Several types of FT architecture possible to be used in some degradation strategy was presented. The methodology is not bounded just to these particular ones but many other architectures satisfying the stated conditions can be used.
- The final system design can be extended with synchronization mechanism for reconfigured units. The GPDRC is designed to cooperate with the synchronization controller.
- To examine the proposed methodology, complex FT system was designed according to its principles and included into the test platform to prove its ability to repair the modules after transient fault occurence and the recovery from permanent fault occurence by degrading the affected FT architecture to less robust one. The fault injector inserting the SEU faults into configuration was used to test the fault detection, localization and mitigation process. The experiments proved that the final system design is able to work correctly after transient fault occurence thanks to the FT architecture design and that the recovery after permanent fault is possible by excluding the affected PRM from further use.

8.2 Possible enhancements of methodology

This complex methodology is based on many principles and incorporates many methods which can be further enhanced to achieve better performance of final secured system, to lower the neccessary area overhead in FPGA or to lower the necessary capacity of external bitstream storage. In next sections, some of them are discussed.

8.2.1 Bitstream compression

Data compression is known approach to reduce the space for storing the data. On the other hand, it can add area overhead due to the need of compressor and decompressor implementation. Moreover, the performance overhead can be caused by the compression and decompression process. Some methods for data compression can be also applied to configuration bitstream data. When the bistream compression is applied to PRBs and used together with the PDR the requirements for compression and decompression will be different. While the bitstream compression is typically done offline immediately after its generation by some software tool in computer the decompression has to be done online directly in the FPGA (in the case when ICAP is used for PDR) or in the external device where the reconfiguration controller is located (when some external configuration interface is used). Thus, the main focus when chosing the compression method should be put among the compression ratio also on the simplicity of the decompressor. Although the throughput of the decompressor is also important, frequently the overal throughput between memory and reconfiguration controller is more limited due the low speed of reading the external memory.

The application of well known compression techniques such as arithmetic or Huffman encoding, LZ78, LZW, RLE and other for configuration bitstream data was presented in [55]. According to this paper, the compression ratio for bitstream data typically vary from 40% to 60%. The througput of decompressor (implemented in Virtex 4 device) can from several hundreds be up to several thousands Mbps. In [46], the method designed specially for bitstream data which is taking into account its composition of frames and the correlation between them is presented. Often, there are only small difference between the neighbouring frames configuring the same CLB column. The next frame can by typically derived from the previous one by applying relatively small number of bit-flips. Therefore, this approach is based on the application of distance vector technique.

The bitstream compression can be incorporated into final FT system designed by means of proposed methodology to achieve lower capacity demands for the external bitstream storage. Due to the fact that the GPDRC unit does not implement the direct read from the memory but it uses external memory controller unit, the decompressor unit for processing the compressed bitstream can be inserted to the data path between these units as it is shown in Figure 8.1. For some compression methods which do not work with the same-sized words of data as the memory and reconfiguration controller additional buffers may be required.

8.2.2 Adoption of isolation design flow

IDF would require for each top-level entity of PRM to have its own level of hierarchy (this is already achieved in current state) and to be implemented independently from the other entities. The final implemented design for entire FPGA will be created by merging this partial implementations. IDF also requires to create regions for all static units of the system. Assigning the units to defined regions will allow to create the fences between the isolated parts of the system design. The fences will be created in places which do not belong to any of defined regions. The connections between the regions in IDF are possible only via trusted routing or by the utilization of IOBs and going off-chip in one region and back in the second one. The creation of trusted routing is quite complicated process (see [63]) but for the replicated units of FT architecture the creation of required constraints can be (at least partially) automated because the relative routing is the same in all these units with the exception of the absolute address of their starting point which is different.

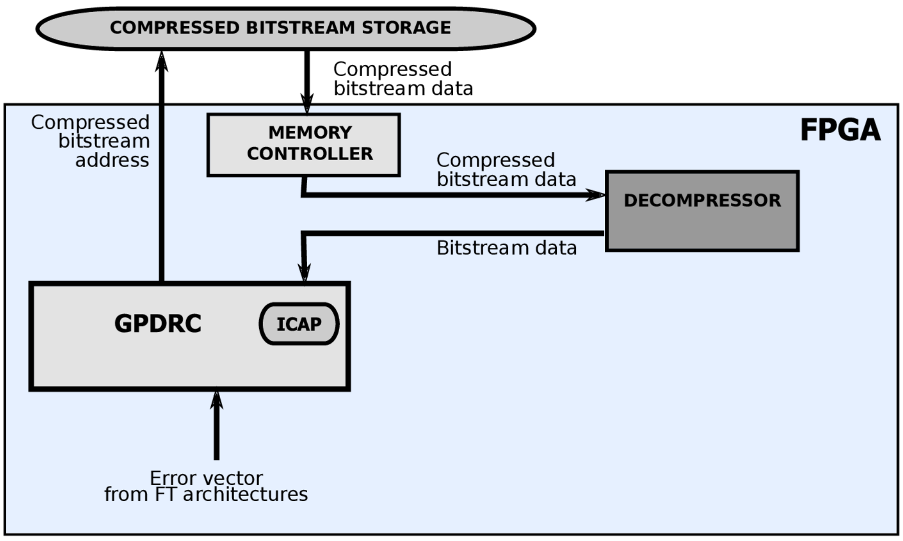

COMPRESSED BITSTREAM STORAGE
Compressed bitstream data
Compressed bitstream address
MEMORY CONTROLLER
FPGA
Compressed bitstream data
DECOMPRESSOR
Bitstream data
GPDRC
ICAP
Error vector from FT architectures

Figure 8.1: Incorporating bitstream compression into FT system design based on the proposed methodology

8.2.3 Other possible enhancements

Several enhancements would be also possible in the GPDRC unit. One of the current issues in this unit is its permanent occupation of ICAP. If the system entered by designer would like to use the PDR ability of FPGA for its reconfiguration it will not be possible because only one instance of this unit can be used. This can be solved by excluding the ICAP instance from the GPDRC unit and using it externally. Then some multiplexing logic can be added and this one ICAP instance can be shared by the original system and by the GPDRC. Because the final secured system (designed by means of the methodology) is based on the set of FT architectures and thus the fault can be masked by them the instant fault mitigation is not necessary. The GPDRC unit can wait until the ICAP instance is not used and then finally perform the mitigation process.

To make the procedure for the transformation from entered system design to secured system design with all needed FT architectures easier, the transformation tool can be implemented. As the steps in this procedure contain many similar tasks (e.g. generating the set of FT architectures for each important system part, constraining each PRM top-level entity, etc.), it would be possible to automate it. The final solution can be the execution of the single script for processing where among the original system design and its original constraints also its important parts with degradation strategies are specified as its input. This script would create all necessary FT architectures, connect them with the static part of design, create the PRMs, constrain their top-level entities, etc. In addition to this, it can also create the implementation script (TCL) for the implementation tool with the plan of all necessary runs to gain all PRBs needed by fault mitigation procedure performed by GPDRC.

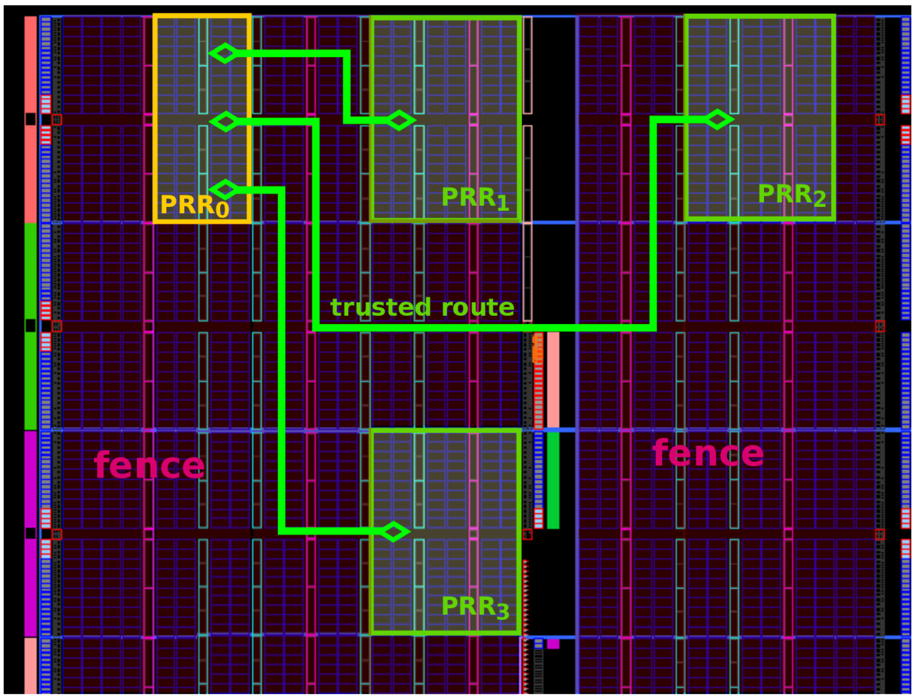


Figure 8.2: The application of IDF in the design phase of FT system design by means of developed methodology

Bibliography

[1] Ieee standard multivalue logic system for vhdl model interoperability (std_logic_1164). *IEEE Std 1164-1993*, pages 1–24, 1993.

[2] M. Abramovici, J. M. Emmert, and C. E. Stroud. Roving stars: an integrated approach to on-line testing, diagnosis, and fault tolerance for fpgas in adaptive computing systems. In *Evolvable Hardware, 2001. Proceedings. The Third NASA/DoD Workshop on*, pages 73–92, 2001.

[3] M. Abramovici, C. Strond, C. Hamilton, S. Wijesuriya, and V. Verma. Using roving stars for on-line testing and diagnosis of fpgas in fault-tolerant applications. In *Test Conference, 1999. Proceedings. International*, pages 973–982, 1999.

[4] Actel. Ds0095: Igloo low power flash fpgas datasheet. URL: `<https://www.microsemi.com/document-portal/doc_download/130694-ds0095-igloo-low-power-flash-fpgas-datasheet>`. Accessed: 2016-09-30.

[5] Actel. Ds0097: Proasic3 family flash fpgas datasheet. URL: `<https://www.microsemi.com/document-portal/doc_download/130704-ds0097-proasic3-flash-family-fpgas-datasheet>`. Accessed: 2016-09-30.

[6] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[7] C. Beckhoff, D. Koch, and J. Torresen. Go ahead: A partial reconfiguration framework. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, pages 37–44, 29 2012-May 1.

[8] Patrick Blau. Juno: Spacecraft information. URL: `<http://spaceflight101.com/juno/spacecraft-information/>`. Accessed: 2016-07-06.

[9] Cristiana Bolchini, Antonio Miele, and Marco D. Santambrogio. Tmr and partial dynamic reconfiguration to mitigate seu faults in fpgas. In *22nd International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT '07)*, pages 87–95, Washington, DC, USA, 2007. IEEE CS.

[10] Cristiana Bolchini, Antonio Miele, and Marco D. Santambrogio. Tmr and partial dynamic reconfiguration to mitigate seu faults in fpgas. In *22nd International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT '07)*, pages 87–95, Washington, DC, USA, 2007. IEEE CS.

[11] J. Bowen and V. Stavridou. Safety-critical systems, formal methods and standards. *Software Engineering Journal*, 8(4):189–209, 1993.

[12] Mustafa Ali Bradley Dutton and John Sunwoo Charles Stroud. Embedded processor based fault injection and seu emulation for fpgas. In *ESA '09: Proceedings of the 2009 International Conference on Embedded Systems & Applications*, pages 183–189, Las Vegas, Nevada, USA, 2009. CSREA Press.

[13] Jason A. Cheatham, John M. Emmert, and Stan Baumgart. A survey of fault tolerant methodologies for fpgas. *ACM Trans. Des. Autom. Electron. Syst.*, 11(2):501–533, 2006.

[14] R. F. DeMara and Kening Zhang. Autonomous fpga fault handling through competitive runtime reconfiguration. In *2005 NASA/DoD Conference on Evolvable Hardware (EH'05)*, pages 109–116, 2005.

[15] S. Distefano and A. Puliafito. Dependability evaluation with dynamic reliability block diagrams and dynamic fault trees. *IEEE Transactions on Dependable and Secure Computing*, 6(1):4–17, 2009.

[16] A. Doumar and H. Ito. Testing approach within fpga-based fault tolerant systems. In *Test Symposium, 2000. (ATS 2000). Proceedings of the Ninth Asian*, pages 411–416, 2000.

[17] T. Drahonovsky, M. Rozkovec, and O. Novak. Relocation of reconfigurable modules on xilinx fpga. In *2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 175–180, 2013.

[18] Nathan A. Harward, Michael R. Gardiner, Luke W. Hsiao, and Michael J. Wirthlin. *A Fault Injection System for Measuring Soft Processor Design Sensitivity on Virtex-5 FPGAs*, pages 61–74. Springer International Publishing, Cham, 2016.

[19] F. Hatori, T. Sakurai, K. Nogami, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, H. Muroga, A. Tanaka, and K. Kanzaki. Introducing redundancy in field programmable gate arrays. In *Custom Integrated Circuits Conference, 1993., Proceedings of the IEEE 1993*, pages 7.1.1–7.1.4, 1993.

[20] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb. Fpga partial reconfiguration via configuration scrubbing. In *Field Programmable Logic and Applications (FPL '09)*, pages 99–104, Washington, USA, 2009. IEEE CS.

[21] P. Horowitz and W. Hill. *The Art of Electronics*. Cambridge University Press, New York, NY, USA, 1989.

[22] Mei-Chen Hsueh, T. K. Tsai, and R. K. Iyer. Fault injection techniques and tools. *Computer*, 30(4):75–82, 1997.

[23] Yoshihiro Ichinomiya, Shiro Tanoue, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga, and Toshinori Sueyoshi. Improving the robustness of a softcore processor against seus by using tmr and partial reconfiguration. In *Proceedings of the 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, FCCM '10, pages 47–54, Washington, DC, USA, 2010. IEEE Computer Society.

[24] Yoshihiro Ichinomiya, Shiro Tanoue, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga, and Toshinori Sueyoshi. Improving the robustness of a softcore processor against seus by using tmr and partial reconfiguration. In *Proceedings of the 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, FCCM '10, pages 47–54, Washington, DC, USA, 2010. IEEE Computer Society.

[25] Hlavicka J. *Cislicove systemy odolne proti porucham.* CVUT, 1992.

[26] Straka M., Miculka L., Kastil J. and Kotasek Z. Test platform for fault tolerant systems design qualities verification. In *15th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 336–341. IEEE Computer Society, 2012.

[27] B.W. Johnson. Fault-tolerant microprocessor-based systems. *Micro, IEEE*, 4(6):6–21, 1984.

[28] F. Kastensmidt, L. Carro, and R. Reis. Designing fault tolerant systems into sram-based fpgas. In *Design Automation Conference, 2003. Proceedings*, pages 650–655, 2003.

[29] Man Cheol Kim and Poong Hyun Seong. Reliability graph with general gates: an intuitive and practical method for system reliability analysis. *Reliability Engineering and System Safety*, 78(3):239–246, 2002.

[30] J. F. Kitchin. Practical markov modeling for reliability analysis. In *1988. Proceedings., Annual Reliability and Maintainability Symposium,*, pages 290–296, 1988.

[31] T. Kranenburg and R. van Leuken. Mb-lite: A robust, light-weight soft-core implementation of the microblaze architecture. In *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, pages 997–1000, 2010.

[32] Kastil J., Straka M., Miculka L. and Kotasek Z. Dependability analysis of fault tolerant systems based on partial dynamic reconfiguration implemented into fpga. In *15th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, pages 250–257. IEEE Computer Society, 2012.

[33] Miculka L. and Kotasek Z. Synchronization technique for tmr system after dynamic reconfiguration on fpga. In *The Second Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN 2013)*, pages 53–56. Politecnico di Milano, 2013.

[34] Miculka L. and Kotasek Z. Generic partial dynamic reconfiguration controller for transient and permanent fault mitigation in fault tolerant systems implemented into fpga. In *17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 171–174. IEEE Computer Society, 2014.

[35] Szurman K., Miculka L. and Kotasek Z. Towards a state synchronization methodology for recovery process after partial reconfiguration of fault tolerant systems. In *9th IEEE International Conference on Computer Engineering and Systems*, pages 231–236. IEEE Computer Society, 2014.

[36] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. Enhanced fpga reliability through efficient run-time fault reconfiguration. *IEEE Transactions on Reliability*, 49(3):296–304, 2000.

[37] Vijay Lakamraju and Russell Tessier. Tolerating operational faults in cluster-based fpgas. In *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays*, FPGA '00, pages 187–194, New York, NY, USA, 2000. ACM.

[38] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings. Rapidsmith: Do-it-yourself cad tools for xilinx fpgas. In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 349–355, 2011.

[39] E. E. Lewis, F. Boehm, C. Kirsch, and B. P. Kelkhoff. Monte carlo simulation of complex system mission reliability. In *1989 Winter Simulation Conference Proceedings*, pages 497–504, 1989.

[40] S. Lin and D. J. Costello. Error control coding-fundamentals and applications. *International Journal of Satellite Communications*, 2(2):139–139, 1984.

[41] Tyler M. Lovelly and Alan D. George. Comparative analysis of present and future space processors with device metrics. *Journal of Aerospace Information Systems*, 14(3):184–197, 2017.

[42] Fayneh E., Yuffe M. and Knoll E. 4.1 14nm 6th-generation core processor soc with low power consumption and improved performance. In *2016 IEEE International Solid-State Circuits Conference, ISSCC 2016, San Francisco, CA, USA, January 31 - February 4, 2016*, pages 72–73, 2016.

[43] K. T. Moon. *Error Correction Coding: Mathematical Methods and Algorithms.* Wiley, Logan, UT, USA, 2005.

[44] Kevin Morris. Xilinx vs. altera: Calling the action in the greatest semiconductor rivalry. URL: `<http://www.eejournal.com/archives/articles/20140225-rivalry/>`, 2014. Accessed: 2015-08-27.

[45] J. Narasimham, K. Nakajima, C. S. Rim, and A. T. Dahbura. Yield enhancement of programmable asic arrays by reconfiguration of circuit placements. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(8):976–986, 1994.

[46] Ju Hwa Pan, T. Mitra, and Weng-Fai Wong. Configuration bitstream compression for dynamically reconfigurable fpgas. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 766–773, 2004.

[47] M. M. Pereira, L. Braun, M. Hübner, J. Becker, and L. Carro. Run-time resource instantiation for fault tolerance in fpgas. In *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 88–95, 2011.

[48] Conrado Pilotto, Jose Rodrigo Azambuja, and Fernanda Lima Kastensmidt. Synchronizing triple modular redundant designs in dynamic partial reconfiguration

applications. In *SBCCI '08: Proceedings of the 21st annual symposium on Integrated circuits and system design*, pages 199–204, New York, NY, USA, 2008. ACM.

[49] Conrado Pilotto, José Rodrigo Azambuja, and Fernanda Lima Kastensmidt. Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications. In *SBCCI '08: Proceedings of the 21st annual symposium on Integrated circuits and system design*, pages 199–204, New York, NY, USA, 2008. ACM.

[50] Jakub Podivinsky, Ondrej Cekan, Marcela Simkova, and Zdenek Kotasek. The evaluation platform for testing fault-tolerance methodologies in electro-mechanical applications. *Microprocess. Microsyst.*, 39(8):1215–1230, 2015.

[51] John Rhea. Bae systems moves into third generation rad-hard processors. *Military & Aerospace Electronics*, 13(5), 2002.

[52] Aiwu Ruan, Bairui Jie, Li Wan, Junhao Yang, Chuanyin Xiang, Zujian Zhu, and Yu Wang. A bitstream readback-based automatic functional test and diagnosis method for xilinx fpgas. *Microelectronics Reliability*, 54(8):1627–1635, 2014.

[53] T. D. A. W. Ruan and P. L. B. R. Jie. A bitstream readback based fpga test and diagnosis system. In *2014 International Symposium on Integrated Circuits (ISIC)*, pages 592–595, 2014.

[54] L. Sekanina, L. Starecek, and Z. Kotasek. Polymorphic gates in design and test of digital circuits. *International Journal of Unconventional Computing*, 4(2):125–142, 2008.

[55] R. Stefan and S. D. Cotofana. Bitstream compression techniques for virtex 4 fpgas. In *2008 International Conference on Field Programmable Logic and Applications*, pages 323–328, 2008.

[56] M. Straka, J. Kastil, and Z. Kotasek. Generic partial dynamic reconfiguration controller for fault tolerant designs based on fpga. In *NORCHIP '10*, pages 1–4, Washington, DC, USA, 2010. IEEE CS.

[57] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici. Built-in self-test of fpga interconnect. In *Test Conference, 1998. Proceedings., International*, pages 404–411, 1998.

[58] Charles Stroud, Ping Chen, Srinivasa Konala, and Miron Abramovici. Evaluation of fpga resources for built-in self-test of programmable logic blocks. In *Proceedings of the 1996 ACM Fourth International Symposium on Field-programmable Gate Arrays*, FPGA '96, pages 107–113, New York, NY, USA, 1996. ACM.

[59] S. Tanoue, T. Ishida, Y. Ichinomiya, M. Amagasaki, M. Kuga, and T. Sueyoshi. A novel states recovery technique for the tmr softcore processor. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 543 –546, 2009.

[60] Irem Y. Tumer. Design methods and practices for fault prevention and management in spacecraft, 1999.

[61] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham. The reliability of fpga circuit designs in the presence of radiation induced configuration upsets. In *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on*, pages 133–142, 2003.

[62] Xilinx. Axcelerator family fpgas. URL: <`https://www.microsemi.com/document-portal/doc_view/130669-axcelerator-family-fpgas-datasheet`>. Accessed: 2016-09-30.

[63] Xilinx. Developing secure designs using the virtex-5 family (xapp1134). URL: <`https://www.xilinx.com/support/documentation/application_notes/xapp1134-developing-secure-designs.pdf`>. Accessed: 2016-09-30.

[64] Xilinx. Spartan-3an fpga family data sheet. URL: <`https://www.xilinx.com/support/documentation/data_sheets/ds557.pdf`>. Accessed: 2016-09-30.

[65] Xilinx. Ug081: Microblaze processor reference guide. URL: <`https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/mb_ref_guide.pdf`>. Accessed: 2016-09-30.

[66] XILINX. Ug702: Partial reconfiguration user guide. URL: <`https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug702.pdf`>. Accessed: 2015-10-12.

[67] Xilinx. Virtex-5 family overview. URL: <`https://www.xilinx.com/support/documentation/data_sheets/ds100.pdf`>. Accessed: 2016-09-30.

[68] XILINX. Xapp1086: Isolation design flow for xilinx 7 series fpgas or zynq-7000 ap socs (ise tools). URL: <`https://www.xilinx.com/support/documentation/application_notes/xapp1086-secure-single-fpga-using-7s-idf.pdf`>. Accessed: 2015-10-12.

[69] Xilinx. Xapp138: Virtex fpga series configuration and readback. URL: <`https://www.xilinx.com/support/documentation/application_notes/xapp138.pdf`>. Accessed: 2016-09-30.

[70] Gulay Yalcin, Osman Sabri Unsal, and Adrian Cristal. Fault tolerance for multi-threaded applications by leveraging hardware transactional memory. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '13, pages 4:1–4:9, New York, NY, USA, 2013. ACM.

[71] Lie-Liang Yang and L. Hanzo. Redundant residue number system based error correction codes. In *IEEE 54th Vehicular Technology Conference. VTC Fall 2001. Proceedings (Cat. No.01CH37211)*, volume 3, pages 1472–1476 vol.3, 2001.

[72] A. J. Yu and G. G. F. Lemieux. Defect-tolerant fpga switch block and connection block with fine-grain redundancy for yield enhancement. In *International Conference on Field Programmable Logic and Applications, 2005.*, pages 255–262, 2005.

[73] Shu-Yi Yu and Edward J. McCluskey. Permanent fault repair for fpgas with limited redundant area. In *DFT '01: Proceedings of the 16th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, pages 125–133, Washington, DC, USA, 2001. IEEE Computer Society.

[74] Velazco R. Ziade H., Ayoubi R. and Idriss T. A new fault injection approach to study the impact of bitflips in the configuration of sram-based fpgas. *the International Arab Journal of Information Technology*, 8(2):155–162, 2011.

List of Abbreviations

ASIC	Application Specific Integrated Circuit
BIST	Built-In Self-Test
BPI	Byte-wide Peripheral Interface
BRAM	Block Random Access Memory
BUT	Block Under Test
CED	Concurrent Error Detection
CLB	Configurable Logic Block
CPLD	Complex Programmable Logic Device
CRC	Cyclic Redundancy Check
DCM	Digital Clock Manager
DSP	Digital Signalling Processor
ECC	Error Correction Code
EDC	Error Detection Code
EDIF	Electronic Design Interchange Format
FAR	Frame Address Register
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
FT	Fault Tolerant
FTA	Fault Tree Analysis
GA	Genetic Algorithm
HDL	Hardware Definition Language
IC	Integrated Circuit
ICAP	Internal Configuration Access Port

IEEE	Institute of Electrical and Electronics Engineers
IOB	Input/Output Block
JTAG	Joint Test Action Group
LUT	Look-Up table
MAP	Map
MTBF	Mean Time Between Failures
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
MUX	MUltipleXor
NCD	Native Circuit Description
NGD	Native Generic Database
NRE	Non-Recurring Engineering
ORA	Output Response Analyzer
PAL	Programmable Array Logic
PAR	Place And Route
PDR	Partial Dynamic Reconfiguration
PIP	Programmable Interconnect Point
PLA	Programmable Logic Array
PLD	Programmable Logic Device
PRM	Partial Reconfiguration Module
PRR	Partial Reconfiguration Region
PRR_{min}	The smallest possible PRR
RBD	Reliability Block Diagram
RTL	Register Transfer Level
SET	Single Event Transient
SEU	Single Event Upset
SPI	Serial Peripheral Interface
SPLD	Simple Programmable Logic Device
SRAM	Static Random Access Memory

STAR	Self-Testing AReas
TMR	Triple Modular Redundancy
TPG	Test Pattern Generator
VLSI	Very Large Scale Integration

List of Figures

2.1 Simple programmable logic devices 9
2.2 CPLD scheme 10
2.3 FPGA architecture 11
2.4 LUT and flip-flops connections inside CLB (Xilinx Virtex 7) 12
2.5 The configuration interfaces of FPGA 15
2.6 Design workflow by Xilinx ISE tool 17
2.7 Design workflow with PDR by Xilinx 18
2.8 The configuration of the smallest possible PRM by configuration frames . . 19
2.9 Bits in configuration memory and resources of FPGA sensitive to SEUs (Xilinx Virtex FPGA family) [28] 22
2.10 The chain of system states during its lifetime 24
2.11 The failure rate of a system during its lifetime 25
2.12 Fault tree analysis method 27
2.13 Reliability block diagrams 28
2.14 The reliability graph of system delivering data from node 1 to node 4 . . . 29
2.15 Markov chain for TMR system with bitstream scrubbing repair [32] 29

3.1 TMR scheme with single and triplicated voter 32
3.2 TMR scheme with different level of granularity 33
3.3 Active redundancy based on hot sparing 33
3.4 CED based on different techniques 35
3.5 Fault detection based on the use of function inversion 36
3.6 Testing of entire FPGA using BIST and PDR 37
3.7 Test system of FPGA based on bitstream readback [53] 38
3.8 Configurations applied by roving star technique 39
3.9 Using PDR to recover system after SEU occurence 42
3.10 Pebble shifting approach 44
3.11 Resource instantiation in FPGA without faults [47] 44
3.12 Resource instantiation considering faults[47] 45
3.13 The alternative layouts of sample system with 4 tiles 45
3.14 The scheme of fault injection approach based on configuration modification 48

5.1 The main structure of the proposed methodology 55
5.2 The assignment of PRRs by different PRMs 56
5.3 The generations of FT architectures and their alternative configurations . . 57
5.4 FT architecture with the unit synchronization 59
5.5 Fault tolerant system structure for SRAM based FPGA 61
5.6 Reconfiguration flow diagram 63

5.7 The dislocation of partial reconfiguration regions belonging to one configuration of FT architecture in the FPGA 64

6.1 The example of regions suitable to host the same relocatable PRM 69
6.2 The constraining of proxy logic inside PRRs 70
6.3 The addition of LUT to each proxy logic block 70
6.4 Design partitioning with different granularity 72
6.5 The set of FT architectures as a sample of degradation strategy 73
6.6 Several possibilities with area allocation for simple TMR architecture 74
6.7 Generating FT architecture of Generation 0 from the unit designated for securing 76
6.8 The scheme of final FT architecture with generated modules in Generation 0 76
6.9 Generating FT architecture of Generation 1 from unit designated for securing 77
6.10 The scheme of final FT architecture with generated modules in Generation 1 77
6.11 The process of generating FT architecture and its implementation for given system unit 78

7.1 GPDRC size vs. the number of FT architectures for various numbers of PRMs per FT architecture 82
7.2 The FT Architecture of Generation 0 based on TMR 84
7.3 The FT Architecture of Generation 1 based on Duplex with checker 85
7.4 The alternative FT Architecture of Generation 1 85
7.5 The architecture of Generation 2 based on Duplex 86
7.6 The simplified Markov model of the FT architecture of the Generation 0 with TMR scheme 88
7.7 The simplified Markov model of the FT architecture of the Generation 1 with duplex scheme with simple voter unit 89
7.8 The Markov model of the FT architecture of the Generation 2 with simple duplex scheme 90
7.9 The original system design 91
7.10 The variants of differently partitioned secured systems 93
7.11 Dependability evaluation platform with SEU injection test platform 94

8.1 Incorporating bitstream compression into FT system design based on the proposed methodology 102
8.2 The application of IDF in the design phase of FT system design by means of developed methodology 103

List of Tables

2.1 The comparison of Virtex FPGA families 13

3.1 The comparison of fault detection methods 40
3.2 The comparison of permanent fault recovery methods 47

6.1 Planning design runs to create all neccessary PRBs for FT architectures with different number of allocated PRRs 80

7.1 The numbers of FPGA resources for GPDRC (32 FT architectures, 6 PRM per FT architecture) 83
7.2 The reconfiguration time of one PRM according to its bitstream size 83
7.3 The overheads of Generations in slices 86
7.4 The resource utilization of original system with MB-LITE processor and Wishbone adapter 91
7.5 The comparison of resource utilization and hardware overhead for different implementations of the given system 93
7.6 The number of detected SEUs in FUs of the architecture 96
7.7 The number of successfully detected permanent faults and the MTTR for permanent fault recovery 98

Appendix A

Author's publications

1. Towards a State Synchronization Methodology for Recovery Process after Partial Reconfiguration of Fault Tolerant Systems. In *4th Prague Embedded Systems Workshop*. Proceedings of 4th PESW, 2016 (20%)

2. Miculka L. and Kotasek Z. Generic partial dynamic reconfiguration controller for transient and permanent fault mitigation in fault tolerant systems implemented into fpga. In *17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 171–174. IEEE Computer Society, 2014 (70%)

3. Szurman K., Miculka L. and Kotasek Z. State synchronization after partial reconfiguration of fault tolerant can bus control system. In *17th Euromicro Conference on Digital Systems Design*, pages 704–707. IEEE Computer Society, 2014 (30%)

4. Szurman K., Miculka L. and Kotasek Z. Towards a state synchronization methodology for recovery process after partial reconfiguration of fault tolerant systems. In *9th IEEE International Conference on Computer Engineering and Systems*, pages 231–236. IEEE Computer Society, 2014 (20%)

5. Miculka L. and Kotasek Z. Synchronization technique for tmr system after dynamic reconfiguration on fpga. In *The Second Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN 2013)*, pages 53–56. Politecnico di Milano, 2013 (80%)

6. Miculka L. Metoda návrhu systémů odolných proti poruchám do omezeného implementačního prostoru na bázi FPGA. In *Počítačové architektury & diagnostika 2013*, pages 63–68. University of West Bohemia in Pilsen, 2013 (100%)

7. Miculka L., Straka M. and Kotasek Z. Methodology for fault tolerant system design based on fpga into limited redundant area. In *16th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, pages 227–234. IEEE Computer Society, 2013 (50%)

8. Straka M., Kastil J., Kotasek Z. and Miculka L. Fault tolerant system design and seu injection based testing. *Microprocessors and Microsystems*, 2013(37):155–173, 2013 (10%)

9. Kastil J., Straka M., Miculka L. and Kotasek Z. Dependability analysis of fault tolerant systems based on partial dynamic reconfiguration implemented into fpga. In *15th*

Euromicro Conference on Digital System Design: Architectures, Methods and Tools, pages 250–257. IEEE Computer Society, 2012 (15%)

10. Miculka L. and Kotasek Z. Design sychronization after partial dynamic reconfiguration of fault tolerant system. In *15th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, pages 20–21. IEEE Computer Society, 2012 (70%)

11. Miculka L. Metoda návrhu systémů odolných proti poruchám do omezeného implementačního prostoru na bázi FPGA. In *Počítačové architektury & diagnostika 2012*, pages 109–115. Faculty of Information Technology, Czech Technical University in Prague, 2012 (100%)

12. Straka M., Miculka L., Kastil J. and Kotasek Z. Test platform for fault tolerant systems design qualities verification. In *15th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 336–341. IEEE Computer Society, 2012 (30%)

13. Miculka L. Metoda návrhu systémů odolných proti poruchám do omezeného implementačního prostoru na bázi FPGA. In *Počítačové architektury & diagnostika 2011*, pages 61–66. Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava, 2011 (100%)

Appendix B

Publications cited by other authors

- Miculka L. and Kotasek Z. Generic partial dynamic reconfiguration controller for transient and permanent fault mitigation in fault tolerant systems implemented into fpga. In *17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 171–174. IEEE Computer Society, 2014
  - B. H. Krishna and C. A. Kumar. A novel method of reconfigurable image processing using fpga. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 3784–3789, 2016
  - S. Di Carlo, P. Prinetto, P. Trotta, and J. Andersson. A portable open-source controller for safe dynamic partial reconfiguration on xilinx fpgas. In *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pages 1–4, 2015
- Miculka L. and Kotasek Z. Synchronization technique for tmr system after dynamic reconfiguration on fpga. In *The Second Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN 2013)*, pages 53–56. Politecnico di Milano, 2013
  - J. Jiménez, U. Bidarte, C. Cuadrado, E. García, and J. Lázaro. Safesoc: A fault-tolerant-by-redundancy evaluation card for high speed serial communications. In *2016 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–4, 2016
- Miculka L., Straka M. and Kotasek Z. Methodology for fault tolerant system design based on fpga into limited redundant area. In *16th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, pages 227–234. IEEE Computer Society, 2013
  - A. S. B. Lopes, E. Santos, M. Kreutz, and M. Pereira. A runtime mapping algorithm to tolerate permanent faults in a cgra. In *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 63–70, 2016
  - R. Backasch, G. Hempel, S. Werner, S. Groppe, and T. Pionteck. Identifying homogenous reconfigurable regions in heterogeneous fpgas for module relocation. In *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, pages 1–6, 2014

- Straka M., Kastil J., Kotasek Z. and Miculka L. Fault tolerant system design and seu injection based testing. *Microprocessors and Microsystems*, 2013(37):155–173, 2013
  - P. H. W. Leong, H. Amano, J. Anderson, K. Bertels, J. M. P. Cardoso, O. Diessel, G. Gogniat, M. Hutton, J. Lee, W. Luk, P. Lysaght, M. Platzner, V. K. Prasanna, T. Rissa, C. Silvano, H. So, and Yu Wang. Significant papers from the first 25 years of the fpl conference. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–3, 2015
  - Thomas E. Carney, Richard P. McWilliam, and Alan Purvis. Modelling electronic circuit failures using a xilinx fpga system. *Procedia CIRP*, 38:277 – 282, 2015
  - Deepa Jose, P. Nirmal Kumar, Arfath Hussain, and Prabhu Shanker. Vlsi circuit partitioning using ant colony optimisation to yield fault tolerant testable systems. *Arabian Journal for Science and Engineering*, 39(12):8709–8729, 2014
  - Antonio da Silva, Pablo Parra, Óscar R. Polo, and Sebastián Sánchez. Run-time instrumentation of systemc/tlm2 interfaces for fault tolerance requirements verification in software cosimulation. *Model. Simul. Eng.*, 2014:42:42–42:42, 2014
  - Reza Omidi Gosheblagh and Karim Mohammadi. Article: Dynamic partial based single event upset (seu) injection platform on fpga. *International Journal of Computer Applications*, 76(3):19–24, 2013
  - D. Jose, P. N. Kumar, and A. David Naveen Dhas. Implementation of power optimized vlsi designs for reliable processing using majority circuit. In *2013 Annual IEEE India Conference (INDICON)*, pages 1–6, 2013
  - Reza Omidi Gosheblagh and Karim Mohammadi. New approach to emulate seu faults on sram based fpgas. *Journal of Electronics (China)*, 31(1):68–77, 2014
  - Reza Omidi Gosheblagh and Karim Mohammadi. Seu-secure parity prediction multiplier on sram-based fpgas. *Journal of Circuits, Systems and Computers*, 23(06):1450081, 2014
  - Xiuhai Cui, Haigang Yang, Yu Peng, and Xiyuan Peng. Research on the packing algorithm for anti-seu of fpga based on triple modular redundancy and the numbers of fan-outs of the net. *Journal of Electronics (China)*, 31(4):284–289, 2014
  - M. Psarakis, A. Vavousis, C. Bolchini, and A. Miele. Design and implementation of a self-healing processor on sram-based fpgas. In *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 165–170, 2014
  - Shobana. M and Senthil Murugan. S. Reconfigurable data processing using duplex fault tolerance system. In *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–5, 2015
  - R. Santos, S. Venkataraman, and A. Kumar. Generic scrubbingbased architecture for custom error correction algorithms. In *2015 International Symposium on Rapid System Prototyping (RSP)*, pages 112–118, 2015
  - I. Villalta, U. Bidarte, J. Gomez-Cornejo, J. Lazaro, and C. Cuadrado. Dependability in fpgas, a review. In *2015 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6, 2015

- D. Agiakatsikas, N. T. H. Nguyen, Z. Zhao, T. Wu, E. Cetin, O. Diessel, and L. Gong. Reconfiguration control networks for tmr systems with module-based recovery. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 88–91, 2016
- M. Vavouras and C. S. Bouganis. Area-driven partial reconfiguration for seu mitigation on sram-based fpgas. In *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–6, 2016

• Kastil J., Straka M., Miculka L. and Kotasek Z. Dependability analysis of fault tolerant systems based on partial dynamic reconfiguration implemented into fpga. In *15th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, pages 250–257. IEEE Computer Society, 2012

- Khaza Anuarul Hoque, Otmane Ait Mohamed, and Yvon Savaria. Formal analysis of SEU mitigation for early dependability and performability analysis of fpga-based space applications. *Journal of Applied Logic*, 21:–, 2017
- V. Simek and R. Ruzicka. Reconfigurable platform with polymorphic digital gates and partial reconfiguration feature. In *2014 European Modelling Symposium*, pages 501–506, 2014
- K.A. Hoque, O.A. Mohamed, Y. Savaria, and C. Thibeault. Probabilistic model checking based dal analysis to optimize a combined tmr-blind-scrubbing mitigation technique for fpga-based aerospace applications. In *Formal Methods and Models for Codesign (MEMOCODE), 2014 Twelfth ACM/IEEE International Conference on*, pages 175–184, 2014
- B. Navas, J. Oberg, and I. Sander. The upset-fault-observer: A concept for self-healing adaptive fault tolerance. In *Adaptive Hardware and Systems (AHS), 2014 NASA/ESA Conference on*, pages 89–96, 2014
- Felix Siegle, Tanya Vladimirova, Jorgen Ilstad, and Omar Emam. Mitigation of radiation effects in sram-based fpgas for space applications. *ACM Comput. Surv.*, 47(2):37:1–37:34, 2015
- F. Siegle, T. Vladimirova, C. Poivey, and O. Emam. Validation of fdir strategy for spaceborne sram-based fpgas using proton radiation testing. In *2015 15th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, pages 1–8, 2015
- F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam. Availability analysis for satellite data processing systems based on sram fpgas. *IEEE Transactions on Aerospace and Electronic Systems*, 52(3):977–989, 2016
- L. Sterpone, L. Boragno, and D. M. Codinachs. Analysis of radiation-induced seus on dynamic reconfigurable systems. In *2016 11th International Symposium on Reconfigurable Communicationcentric Systems-on-Chip (ReCoSoC)*, pages 1–6, 2016

• Straka M., Miculka L., Kastil J. and Kotasek Z. Test platform for fault tolerant systems design qualities verification. In *15th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 336–341. IEEE Computer Society, 2012

– Felix Siegle, Tanya Vladimirova, Jorgen Ilstad, and Omar Emam. Mitigation of radiation effects in sram-based fpgas for space applications. *ACM Comput. Surv.*, 47(2):37:1–37:34, 2015