2-1-2015

Fault-tolerant fpga for mission-critical applications.

Gehad Ismail Ibrahim Alkady

The American University in Cairo

School of Sciences and Engineering

FAULT-TOLERANT FPGA FOR MISSION-CRITICAL APPLICATIONS

A Thesis Submitted to

Electronics and Communication Engineering Department

in partial fulfillment of the requirements for
the degree of Master of Science

by Gehad Ismail Ibrahim Alkady

under the supervision of Prof. Hassanein H. Amer and Dr. Mohamed Bakr
May/2015

Approval Sheet Goes Here

To my Mum and Dad

ACKNOWLEDGMENTS

I would like to acknowledge both my supervisors: Prof. H.H. Amer and Dr. Mohamed Bakr for their great support throughout my thesis.

I would also like to acknowledge Dr. Nahla Elaraby and Dr. Ahmed Madien for their assistance throughout my research.

I would like to acknowledge my examiners Dr. Ahmed Abou Auf and Dr. Rafik El gendy for their valuable and constructive comments that help me a lot in improving the quality of this thesis.

Finally, I would like to acknowledge the graduate program directors: Dr. Ayman El Ezabei and Dr Karim Seddik.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ASIC	Application Specific Integrated Circuits
CPLD	Complex Programmable Logic Devices
FPGA	Field Programmable Gate Arrays
PR	Partial Reconfiguration
SRAM	Static Random Access Memory
EPROM	Erasable Programmable Read-Only Memory
CLBs	Configurable Logic Blocks
SOC	Systems-On-Chip
SEUs	Single Event Upsets
UART	Universal Asynchronous Receiver/Transmitter
FIFO	First In First Out
CAM	Content Addressable Memory
CLK	Clock
RAM	Random Access Memory
ROM	Read Only Memory
CE	Clock-Enable
SR	Set/Reset
HDL	Hardware Description Language
RTL	Register Transfer Level
NGD	Xilinx native generic database file
NCD	Native Circuit Description file
TDDB	Time-Dependent Dielectric Breakdown
SEEs	Single Event Effects
SEUs	Single Event Upsets
SBE	Single Bit Error
MBU	Multiple Bit Upset
SET	Single Event Transient
SEL	Single Event Latch up
SEFI	Single Event Functional Interrupt
TMR	Triple Modular Redundancy
BIST	Built-In Self-Test

LIST OF SYMBOLS

P_b	Partially Reconfigurable block

ABSTRACT

The American University in Cairo, Egypt

FPGA Testing

Name: Gehad Ismail Ibrahim Alkady

Supervisors: Prof. Hassanein H. Amer and Dr. Mohamed Bakr

One of the devices that play a great role in electronic circuits design, specifically safety-critical design applications, is Field programmable Gate Arrays (FPGAs). This is because of its high performance, re-configurability and low development cost. FPGAs are used in many applications such as data processing, networks, automotive, space and industrial applications.

Negative impacts on the reliability of such applications result from moving to smaller feature sizes in the latest FPGA architectures. This increases the need for fault-tolerant techniques to improve reliability and extend system lifetime of FPGA-based applications.

In this thesis, two fault-tolerant techniques for FPGA-based applications are proposed with a built-in fault detection region. A low cost fault detection scheme is proposed for detecting faults using the fault detection region used in both schemes. The fault detection scheme primarily detects open faults in the programmable interconnect resources in the FPGAs. In addition, Stuck-At faults and Single Event Upsets (SEUs) fault can be detected.

For fault recovery, each scheme has its own fault recovery approach. The first approach uses a spare module and a 2-to-1 multiplexer to recover from any fault detected. On the other hand, the second approach recovers from any fault detected using the property of Partial Reconfiguration (PR) in the FPGAs. It relies on identifying a Partially Reconfigurable block (P_b) in the FPGA that is used in the recovery process after the first faulty module is identified in the system. This technique uses only one location to recover from faults in any of the FPGA's modules and the FPGA interconnects.

Simulation results show that both techniques can detect and recover from open faults. In addition, Stuck-At faults and Single Event Upsets (SEUs) fault can also be detected. Finally, both techniques require low area overhead.

I. INTRODUCTION

After the introduction of transistors, Application Specific Integrated Circuits (ASIC) appeared to combine different types and numbers of transistors, permitting engineers to design complex digital architectures on a single silicon substrate. To avoid the expensive costs associated with custom ASIC design and the unrestrained flexibility of ASICs, Complex Programmable Logic Devices (CPLD) were developed; they are simple programmable logic devices. Moreover, they contain non-volatile memory. To combine the complexity of ASICs with the programmability of CPLDs, Field Programmable Gate Arrays (FPGA) appeared in the 1980's.

Field programmable Gate Arrays (FPGAs) are pre-fabricated silicon devices that can be configured in the field to become almost any type of digital circuit or system. FPGAs are used in designing several applications including data processing, networks, automotive, space and industrial applications. For low to medium volume productions, FPGAs allowed designers to design complex systems in a shorter time which results in decreasing the time-to-market and providing the public with the latest technologies more rapidly. However, the main advantage of FPGAs is flexibility; it is also one of its major drawbacks. The flexible nature of FPGAs makes them significantly larger, slower, and consumes more power compared to ASICs. These disadvantages appear largely because the programmable routing interconnect of FPGAs consumes a large area. Despite this disadvantage, FPGAs are one of the most popular digital circuit implementation media.

There are a number of programming technologies that have been used for FPGAs. Each of these technologies has different features which in turn has a major effect on the programmable architecture. These technologies include flash, anti-fuse and static memory (SRAM). Based on those technologies, there are different types of FPGAs which are Flash-based FPGAs, Anti-fuse based FPGAs and SRAM-based FPGAs. Flash-based FPGAs use flash or Erasable Programmable Read-Only Memory (EPROM) based programming technology. They have several advantages such as being nonvolatile in nature and area efficient compared to other technologies. However, they have some drawbacks including that they cannot be reconfigured or reprogrammed an infinite number of times and use non-standard CMOS process. Anti-fuse based FPGAs include anti-fuses technology which are originally open-circuits and take low on resistance only when programmed. They are appropriate for FPGAs because they can be built using modified CMOS technology. The main advantage of anti-fuse programming technology is that they have low area. Moreover, they have lower on resistance and parasitic capacitance than other programming technologies. Also, they are non-volatile in nature. The disadvantages of using Anti-fuse programming technology in FPGAs are that they do not make use of standard CMOS process and cannot be reprogrammed.

SRAM-based FPGAs use the static memory (SRAM) based programming technology. The static memory cells are found throughout the FPGA to provide configurability. In an SRAM-based FPGA, SRAM cells are primarily used for the following purposes: program the Configurable Logic Blocks (CLBs) and the FPGA programmable interconnections. SRAM-based FPGAs are mainly appealing for several reasons include high performance, in-site re-configurability and low development cost. Also, a portion of the SRAM-based FPGA can be partially reconfigured while the rest of the FPGA is still working by using the Partial Reconfiguration (PR) property in the FPGA. Any future updates in the final product can be easily applied by simply downloading a new bit-stream and can be used also for increasing FPGA-based applications reliability. On the other hand, they have some drawbacks such as the fact that an SRAM cell comprises 6 transistors which consumes a larger area compared to other programming technologies. SRAM-based FPGAs dominate other FPGAs based on other programming technologies and they are used in the implementation of processors and Systems-On-Chip (SOC) in mission-critical applications.

FPGAs are moving to smaller feature sizes; this negatively impacts the reliability of such applications [Psarakis 2012]. Hence, this increases the need for fault-tolerant techniques to improve the reliability and extend system life time of FPGA-based applications.

Fault-tolerance is the property that allows a system to continue operating properly after the occurrence of a single fault or multiple faults or failure within some of its components. Accordingly, the system may work at a reduced level of performance rather than failing completely. Most of the fault-tolerant techniques consist of a fault detection technique and a fault recovery technique. The Fault detection scheme is the scheme that recognizes that a fault has occurred, while the fault recovery technique is the technique of recapturing the operational status via reconfiguration even in the presence of faults.

In this research, two fault-tolerant approaches for FPGA-based applications are proposed with a built-in fault detection region. A low cost fault detection scheme is implemented for detecting faults using the fault detection region used in both techniques. The fault detection scheme mainly detects open faults in the programmable interconnect resources in the FPGAs. In addition, Stuck-At faults and Single Event Upsets (SEUs) fault can be detected. For fault recovery, each approach has its own fault recovery technique. The first approach uses a spare module and a 2-to-1 multiplexer to recover from any fault detected. However, the second approach recovers from any fault detected using the property of Partial Reconfiguration (PR) in the FPGAs. It depends on identifying a Partially Reconfigurable block (P_b) in the FPGA that is used in the recovery process after the first faulty module is recognized in the system.

It will be shown that the main advantages of the second technique developed in this thesis include reduced area overhead, simple recovery hardware and small memory space required to store configuration files which is an important factor in embedded systems. Since both techniques are implemented on FPGAs, a brief description of FPGAs and more specifically the Xilinx Virtex-4 FPGA (XC4VFX12) is given next. Xilinx Virtex-4 FPGA (XC4VFX12) is the implementation platform of this work. Moreover, the design flow of FPGAs is described. A summary of the types of faults occurring in FPGAs is elaborated in the fourth section. Also, it is followed by a description of the Partial Reconfiguration (PR) technique.

I.1. FPGA Architecture

The SRAM-based FPGAs consist of an array of Configurable Logic Blocks (CLBs) and programmable interconnect resources which control the routing between the CLBs. In addition, there are programmable I/O ports to communicate with the outside world as shown in Figure 1 [Brown 1996].

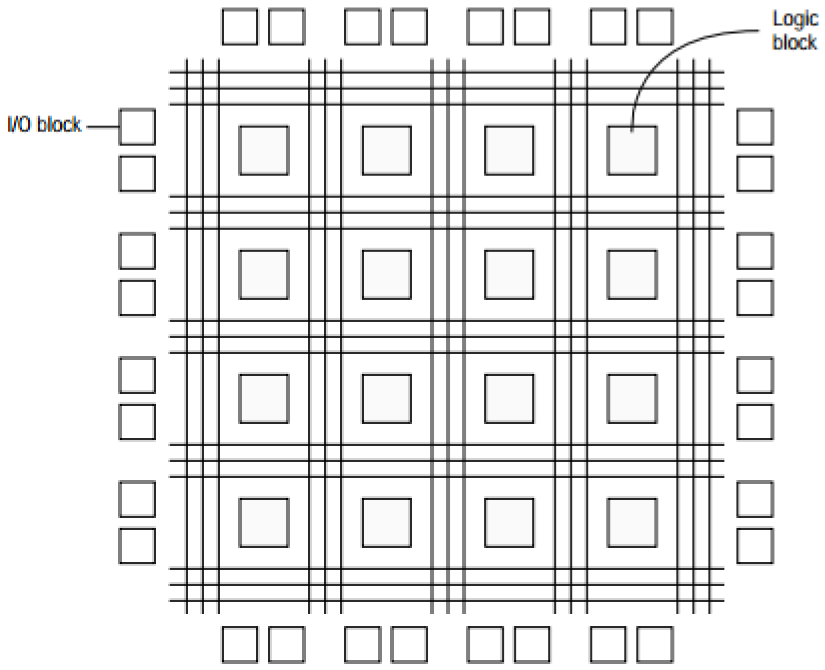


Figure 1: FPGA Architecture [Brown 1996]

CLBs consist of three main subcomponents: Look Up Tables, Multiplexers and Sequential elements such as flip-flops. The main building block of a CLB is a Look Up Table (LUT). LUTs are responsible for the implementation of combinational logic by storing their equivalent truth table through SRAM configuration cells in the FPGA and using the logic inputs as the address into the LUTs to select the logic stored at the appropriate location by using a multiplexer. Figure 2 shows a four-input LUT of Virtex-4 FPGA [Xilinx 2010].

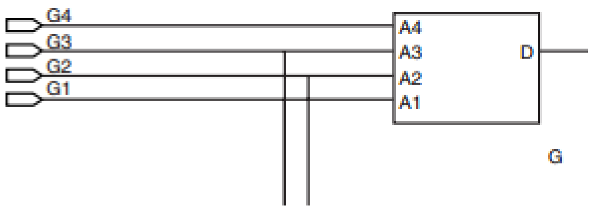


Figure 2: Virtex-4 FPGA four-input LUT [Xilinx 2010]

The second main subcomponent in CLBs is Multiplexers. They are used to determine the connection of signals to one another inside the CLBs. Figure 3 demonstrates a four-input multiplexer with the two select inputs connected to SRAM configuration cells. Accordingly, the inputs of the multiplexer are specified when the configuration is downloaded and remains the same during the run time of the circuit on the FPGA.

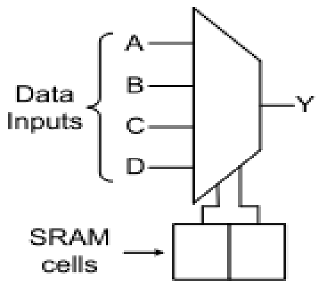


Figure 3: Four-input FPGA MUX

Sequential elements are the third main subcomponent in CLBs. They are crucial for any digital logic design. As a result, they should exist in the FPGA. They are usually located after multiplexers so that any signal coming from the logic portion of a slice can be routed through. Latest FPGAs have sequential elements that can be configured into either a flip-flop or a latch. Accordingly, both edge and level sensitive designs can be implemented.

Each CLB communicates with another through the interconnect network of the FPGA. As a result, any point in the FPGA can be linked to any other point. There are programmable switches between the CLB input/output pins and the wires. They are all clustered together into a so-called Programmable Switch Matrix (PSM). PSM can make connections between several pins connected to it. Accordingly, it connects CLB pins to the FPGA interconnects. The programmable switches inside the PSM are called Programmable Interconnect Points (PIP). A PIP is a CMOS pass transistor switch which is controlled by an SRAM cell as shown in Figure 4 [Godal 2001]. Moreover, Figure 5 shows PIPS inside the Xilinx ISE Simulator ISIM.

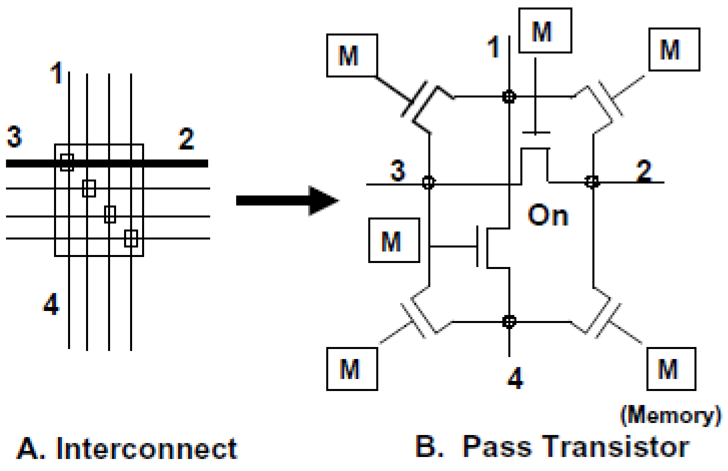


Figure 4: Programmable Interconnect Points (PIP) [Godal 2001]

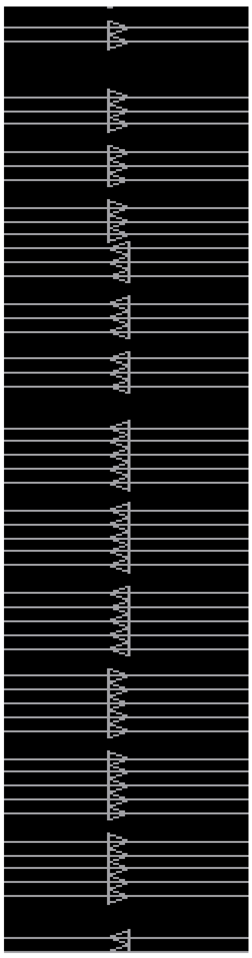


Figure 5: Programmable Interconnect Points (PIP) in Xilinx ISE Simulator.

FPGA interconnections consist of two types which are global routing that connects CLBs to I/O buffers, non-adjacent CLBs and other embedded components. Moreover, local routing connects CLBs to other adjacent CLBs and CLBs to global routing which is done through a switch matrix. The global routing consists of four types of lines such as:

- Long Lines: the routing of the long lines has three connections: beginning (BEG), middle (MID) and end (END). Every direction has 10 BEGs, MIDs, and ENDs. All the directions are bi-directional for a total of 240 wire segments per switch matrix. They span 24 rows/columns of components with a switch matrix connection at every sixth component.

- Double Lines: they have five connections into a switch matrix between beginning and end. They can source in all four directions of the FPGA from a switch matrix. The resources of the double lines span three columns/rows of components with a connection to the switch matrix for each component.
- Hex lines: they have three connections into a switch matrix similar to long lines. They can source in all four directions from switch matrix. They span six rows or columns of components.
- Direct connect lines: they are responsible for routing signals to adjacent blocks: horizontally, vertically, and diagonally.

Figure 6 shows the types of the global routing inside Xilinx ISE Simulator ISIM.

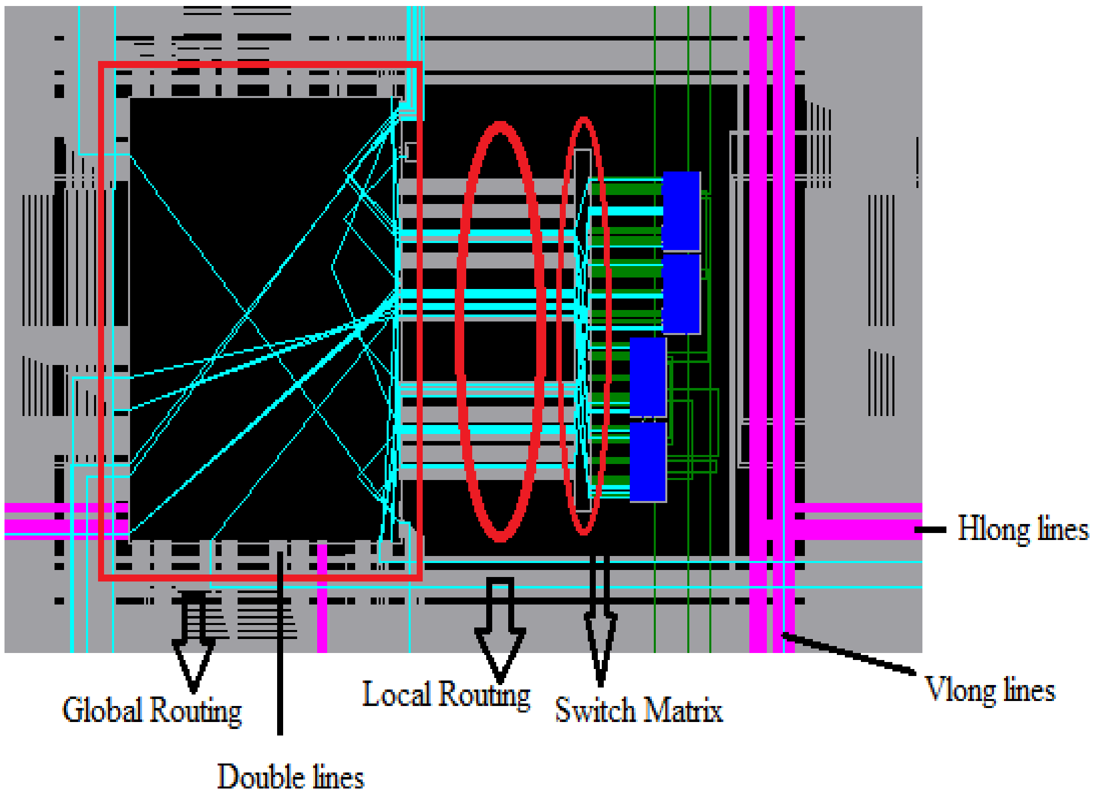


Figure 6: Different types of the global routing inside Xilinx ISE Simulator ISIM.

There are several types of PIPs used [Sterpone 2008]. Figure 7 shows different types of PIPs as follows. They include:

- Cross-point: it connects vertical or horizontal wire segments and allowing turns.
- Breakpoint: it connects or isolates 2 wire segments.
- Decoded MUX: it is a group of 2^n cross-points connected to a single output configured by *n* configuration bits.
- Non-decoded MUX: it is an *n* wire segment each with a configuration bit (n segments).
- Compound cross-point: This type consists of 6 Break-point PIPS. It can isolate two isolated signal nets.

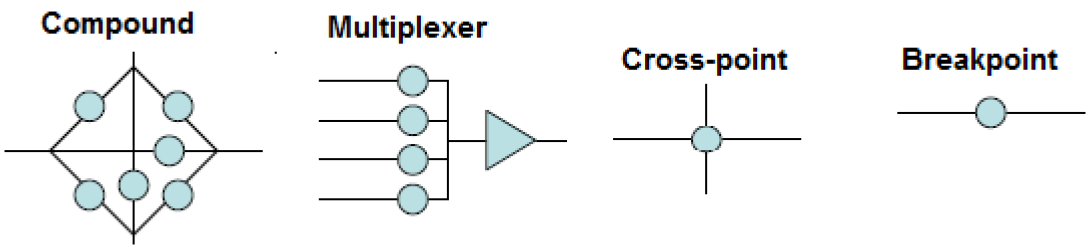


Figure 7: Different types of PIPs

In Figure 8, it is shown how PSM can make any connection between any two adjacent CLBs. It is responsible for routing the signals in their correct path, switching connections on or off and buffering the interconnect wires.

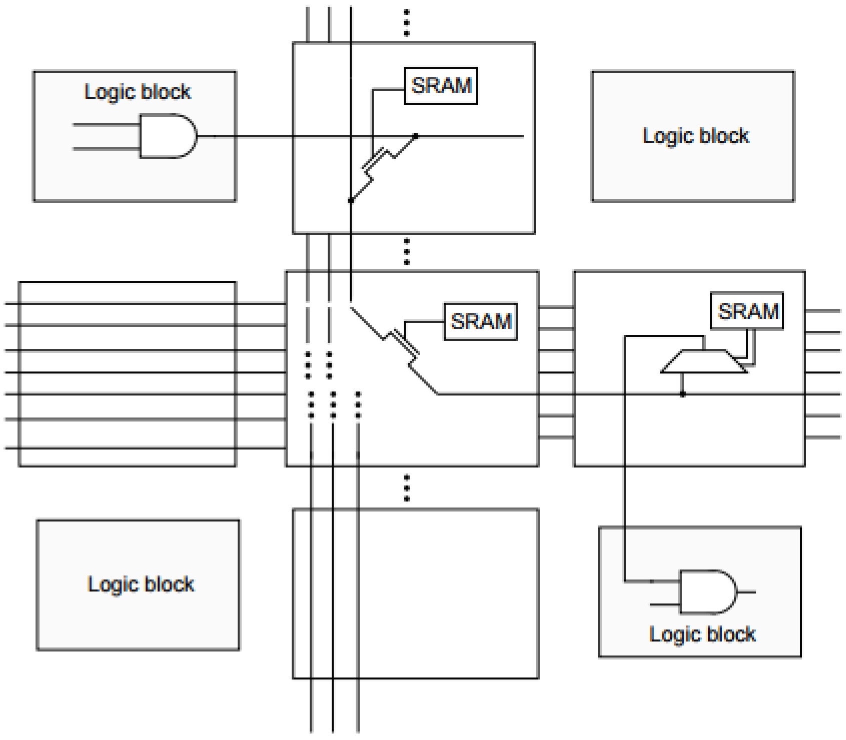


Figure 8: Programmable Switch Matrix (PSM) [Brown 1996]

I.2. XILINX VIRTEX-4 FPGA

The implementation platform of this work is the Xilinx Virtex-4 FPGA [Xilinx 2010]. Virtex-4 FPGA is produced in a technology of 90 nm. The device used in the experimental work is XC4VFX12 which has 5,472 slices. It is capable of several advanced features including Partial Reconfiguration (PR) and memory Readback. Also, it contains various advanced components such as block Random-Access Memory (RAM) and Shift Registers. This section is dedicated to the Virtex-4 FPGA architecture.

I.2.1. CLB ARCHITECTURE

Configurable Logic Blocks (CLBs) are the main logic resource to carry out combinational and sequential circuits. Each CLB block is connected to a PSM to access the general routing matrix as shown in Figure 9.

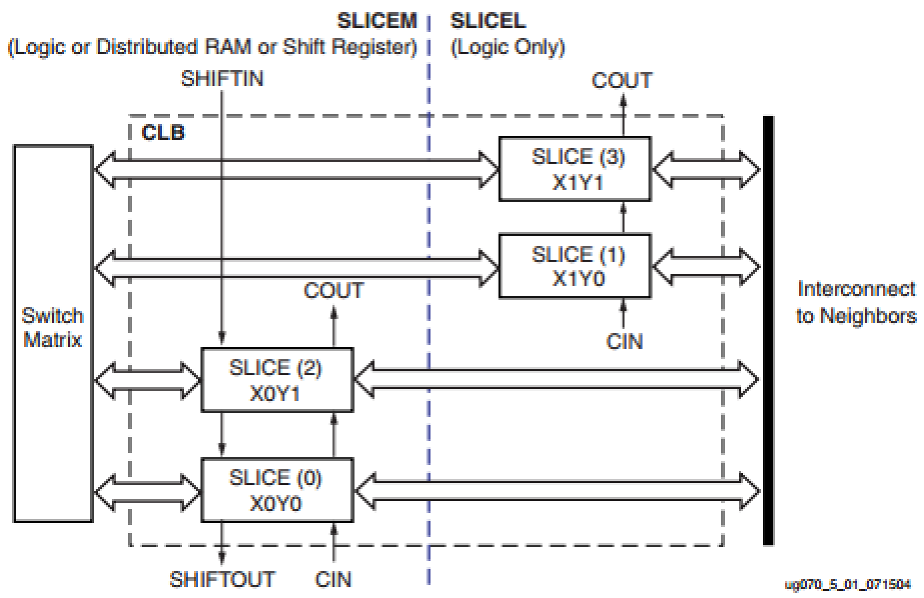


Figure 9: CLB Architecture [Xilinx 2010]

A CLB block is constructed from four interconnected slices. These slices are clustered in pairs. Each pair is structured as a column. SLICEM specifies the pair of slices in the left column and SLICEL indicates the pair of slices in the right column. However, each pair has an independent carry chain; only the slices in SLICEM have a common shift chain. Figure 9 demonstrates the CLB placed in the bottom-left corner of the FPGA die. Slices X0Y0 and X0Y1 create the SLICEM column-pair; however, slices X1Y0 and X1Y1 form the SLICEL column-pair. For each CLB, SLICEM designates the pair of slices labeled with an even number which are SLICE (0) or SLICE (2). In addition, SLICEL indicates that the pair of slices with an odd number includes SLICE (1) or SLICE (3).

The common blocks in both slice pairs (SLICEM and SLICEL) are two logic-function generators which are Look-Up Tables (LUTs), two storage elements, wide-function multiplexers, carry logic and arithmetic gates. These blocks are the basic elements of both SLICEM and SLICEL which are used to implement logic, arithmetic and ROM functions. SLICEM provides two additional functions which are the storage of data using distributed Random Access Memory (RAM) and the shifting of data with 16-bit registers. SLICEM is shown in Figure 10. SLICEM contains a superset of elements and connections found in all slices. SLICEL is shown in Figure 11.

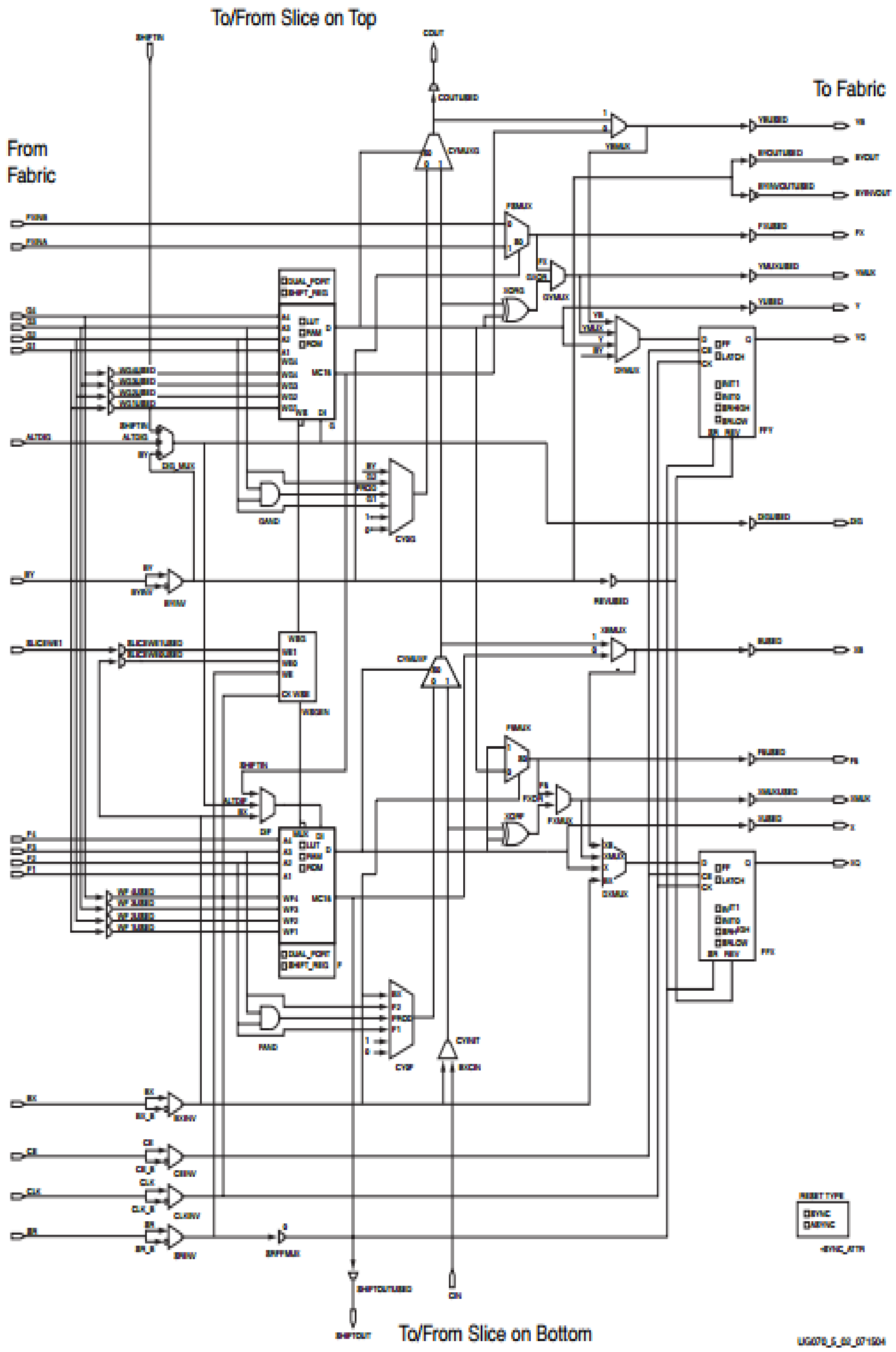


Figure 10: SLICEM [Xilinx 2010]

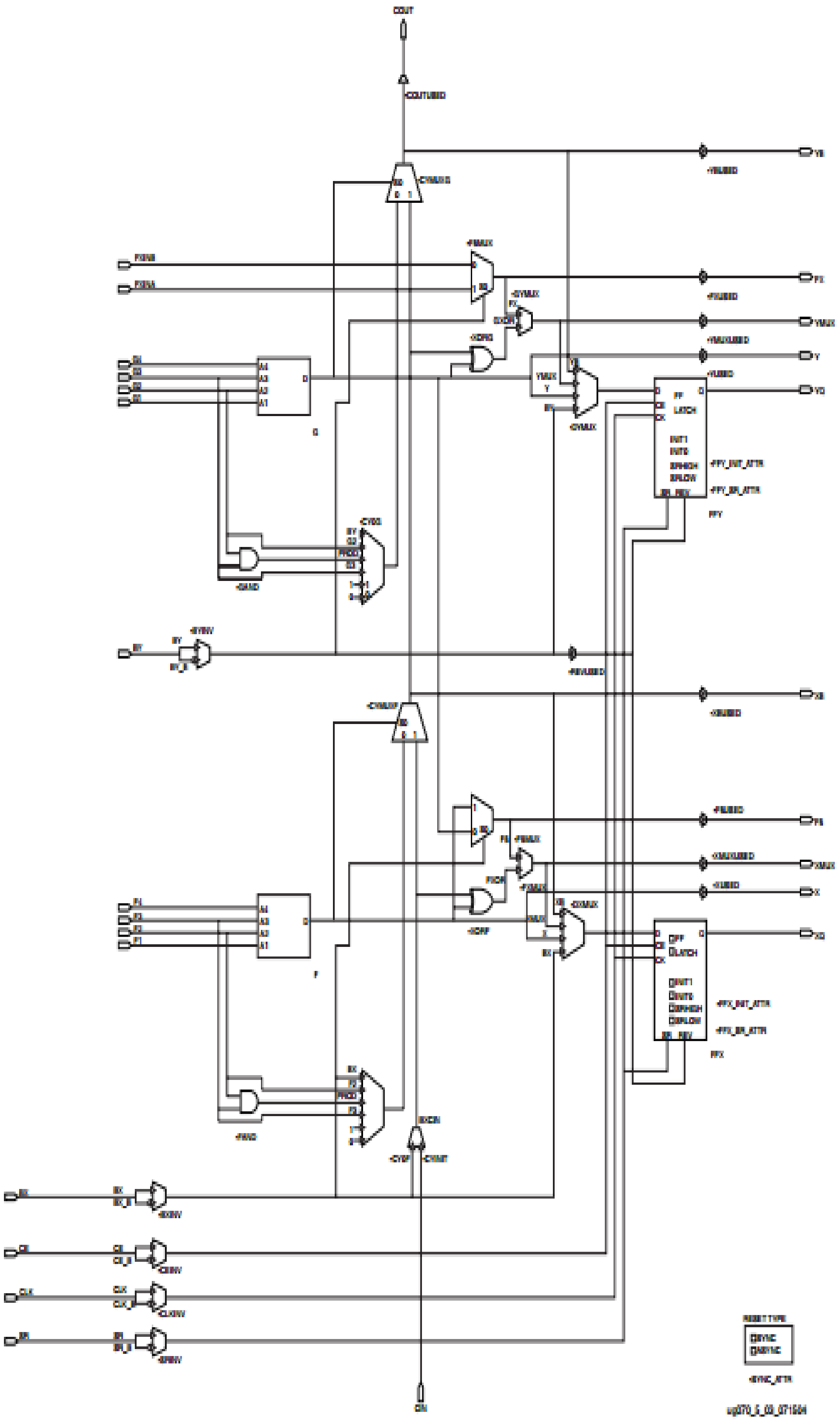


Figure 11: SLICEL [Xilinx 2010]

The logic resources in one CLB are shown in the following table:

Table 1: CLB logic resources

slices	LUTs	Flip-Flops	MUTs-ANDs	Arithmetic & Carry Chain	Distributed RAM	Shifting Registers
4	8	8	8	2	64 bits	64 bits

Virtex-4 FPGA function generators are constructed as 4-input Look-Up Tables (LUTs). Each LUT in a slice (F and G) contains four independent inputs. It is responsible of the implementation of any randomly defined four-input Boolean function. The propagation delay through a LUT is independent of the function performed. Output signals from the LUTs can exit the slice through the X or Y outputs. Firstly, it enters the XOR dedicated gate. Then, it enters the select line of the carry-logic multiplexer and feed the D input of the storage element or it goes to the MUXF5.

In addition, the Virtex-4 FPGA slices contain multiplexers which are MUXF5 and MUXFX. They are used to combine up to eight LUTs to implement any function of five, six, seven or eight inputs in a CLB. The MUXFX can be MUXF6, MUXF7 or MUXF8 which are specified according to the position of the slice in the CLB. In addition, MUXFX can be used to map any function of six, seven or eight inputs and selects wide logic functions. Accordingly, functions with up to nine inputs (MUXF5 multiplexer) can be performed in one slice. The multiplexers with wide functions can efficiently combine LUTs within the same CLB or across different CLBs. As a result, logic functions with even more input variables can be implemented.

The storage elements in a Virtex-4 FPGA slice can be configured as edge-triggered D-type flip-flops or level-sensitive latches. The D input can be motivated directly by a LUT output through the DX or DY multiplexer or by the slice inputs. This occurs by passing the signal through the LUTs by the BX or BY input. Clock (CLK), Clock-Enable (CE) and Set/Reset (SR) are the common control signals for both storage elements in one slice. All of them have independent polarity. Any inverter located on a control input is automatically absorbed. Clock-Enable (CE) signal is active High by default. If CE is left unconnected, it goes to the active state by default. In addition, each slice has other control signals which are Set and Reset signals (SR and BY slice inputs). SR changes the state of storage element into the state identified by the attribute SRHIGH or SRLOW. When SR is asserted, SRHIGH forces a logic High and SRLOW forces a logic Low. An optional second input BY changes the storage element state into the opposite state through the REV pin, when SR is used. Reset condition is prevalent over the set condition. Figure 12 shows how register or latch is configured in a slice.

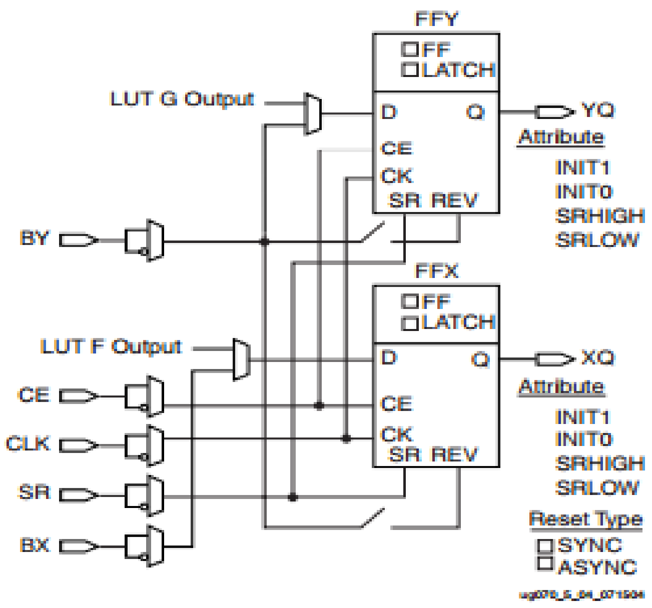


Figure 12: Register/Latch Configuration in a slice [Xilinx 2010]

The Distributed RAM and Memory are only available in SLICEM. Multiple left-hand LUTs in SLICEMs can be joined in several ways to store larger amounts of data. LUTs located in SLICEM can be implemented as a 16x1-bit synchronous RAM resource named distributed RAM block. RAM blocks are configurable within a CLB to implement the following:

• Single-Port 16 x 4-bit RAM

• Single-Port 32 x 2-bit RAM

• Single-Port 64 x 1-bit RAM

• Dual-Port 16 x 2-bit RAM

Shift registers exist only in SLICEM. LUTs in SLICEM can also be configured as a 16-bit shift register without using the flip-flops available in a slice. If an LUT is used to be configured as a shift register, each LUT can delay serial data anywhere from 1 to 16 clock cycles. SHIFTIN and SHIFTOUT line are used to cascade LUTs to implement larger shift registers. The four left-hand LUTs located in SLICEM of a single CLB are consequently cascaded to produce delays up to 64 clock cycles. It is similarly possible to associate shift registers across more than one CLB.

I.2.2. XILINX VIRTEX-4 FPGA INTERCONNECTIONS

The proposed two approaches in this work are implemented using Xilinx Virtex-4 FPGA which has hierarchal routing resources consisting of local and global routing resources [Xilinx 2010]. Local routing resources contain the fast connect lines which are the internal CLB interconnections. The global routing resources (same as other FPGAs) consist of several types of wiring channels which are long lines, hex lines, double lines and direct connect lines as illustrated in section one . Xilinx Virtex-4 FPGAs have 16 direct connections in all directions (E, W, N, and S) between any two adjacent switch matrices.

I.3. XILINX FPGA DESIGN FLOW

FPGA design flow consists of several stages as shown in Figure 13 which are Design entry, Design synthesis, Design implementation and Xilinx Device programming or configuration.

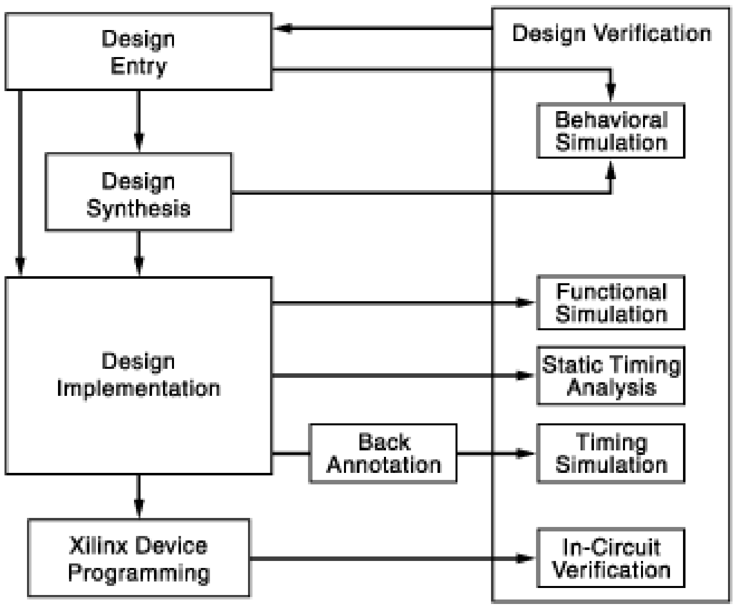


Figure 13: FPGA Design Flow [Official site of Xilinx]

In the Design entry stage, the design files are created by VHDL or VERILOG HDL languages. They can be simulated to check that the design is behaviorally working correctly. Then, there is the stage of Design synthesis that is shown in Figure 14.

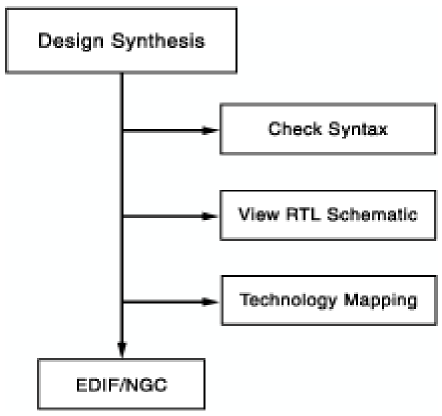


Figure 14: Design synthesis [Official site of Xilinx]

In this stage, the synthesis process checks the syntax of HDL code and analyzes the hierarchy of the design which guarantees that the design is optimized for the design architecture. This happens by changing the HDL code of the design into generic symbols, such as AND and OR gates forming RTL schematic for the design. Then, the Technology schematic is implemented which shows the circuit in terms of logic elements such as LUTs and carry logic optimized to the targeted Xilinx device. The resulting netlist from the synthesis process is saved to an NGC file for XST or an EDIF file for Synplify/Synplify Pro.

Design implementation is the third stage in the FPGA design flow which consists of three main steps: translate, map and place & route as shown in Figure 15.

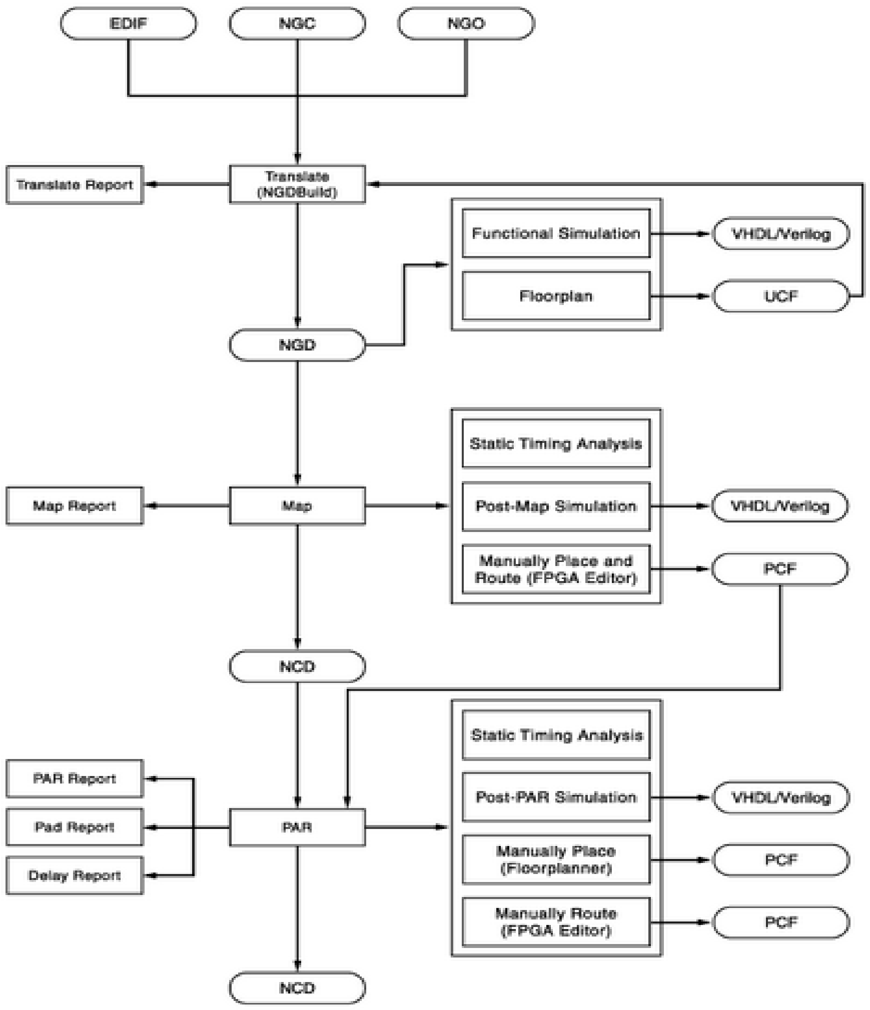


Figure 15: Design implementation [Official site of Xilinx]

The first process in the Design implementation stage is the translate process which combines all of the input netlists, design constraints and outputs into a Xilinx native generic database (NGD) file which defines the logical design reduced to Xilinx primitives. Then, there is the map process that maps the logic defined by an NGD file into FPGA blocks, such as CLBs and IOBs. The output design is a native circuit description (NCD) file that physically characterizes the design mapped to the components in the Xilinx FPGA. The last stage in the Design implementation is the place & route process which takes a mapped NCD file. Then, it produces an NCD file that is used as input for bitstream generation.

Xilinx Device programming or configuration is the last stage in the FPGA design flow. Firstly, generate programming file process is used for producing a bitstream for Xilinx device configuration. After the design is completely routed, the device must be configured so it can execute the desired function. FPGA Configuration Interfaces are classified into several methods:

1. Master (Serial or Parallel): FPGA retrieves configuration from ROM at initial power-up.
2. Slave (Serial or Parallel): FPGA configured by an external source which can be microprocessor or other FPGA. It is used mainly for dynamic partial reconfiguration.
3. Boundary Scan: FPGA uses 4-wire IEEE standard serial interface for testing. Moreover, there is both write and read access to the configuration memory and it interfaces to FPGA via FPGA core internal routing network.

In addition, there are many techniques for configuring FPGAs such as:

1. Full configuration and Readback: it is a simple configuration interface and the internal calculation of frame address is automatic. Large FPGAs require a longer time for downloading than other smaller FPGAs.
2. Compressed configuration: it requires multiple frame write capability because the identical frames of configuration data are written to multiple frame addresses. Frame address of the compressed

configuration is much smaller than the frame of the configuration data. It decreases the download time for initial configuration depending on the regularity of function of the system and the array percent that is used.

Partial Reconfiguration (PR) and Readback: it only changes portions of configuration memory according to reference design and it reduces the download time for reconfiguration. An overview of Partial Reconfiguration (PR) is discussed in section II.5.

I.4. Types of faults in FPGAs

There are two types of faults: transient faults and permanent faults. Permanent faults are the faults that damage the semiconductors permanently. They continue to exist until the faulty component is repaired or replaced. There are three main causes of permanent faults in FPGAs which are degradation phenomena, manufacturing defects and radiation.

The degradation phenomena are hot-carrier effect, Negative-Bias Temperature Instability, Electromigration and Time-Dependent Dielectric Breakdown (TDDB). Hot-carrier effect leads to the buildup of trapped charges in the gate channel interface region [Guerin 2007]. This causes a gradual reduction in channel mobility and increase in threshold voltage in CMOS transistors. The Negative-Bias Temperature Instability effect in the circuit is that the switching speeds become slower which leads to delay faults. It also creates trapped charges as hot-carrier effect [Dieter 2003]. Electromigration is a process by which metal ions migrate over time resulting in voids and deposits in FPGA interconnects. Eventually, these can cause faults due to the creation of open and short circuits [Clarke 1990]. Time-Dependent Dielectric Breakdown (TDDB) degradation affects the gates of transistors causing an increase in leakage current and finally a short circuit. This mechanism creates charge traps within the gate dielectrics weakening the potential barrier it forms [Esseni 2002].

In addition to degradation, there is another type of fault which can affect FPGAs permanently which is the manufacturing defects. Manufacturing defects can be described as circuit nodes which are stuck-at-0 or 1 or switch too slowly to meet the timing specification. Defects also affect the FPGA interconnect network and can cause short or open faults in the circuits and stuck open or closed pass transistors [Harris 2002].

Radiation is the major origin for inducing faults in FPGAs. Single Event Effects (SEEs) occur in most of the cases, when radiation hits the silicon of the FPGA. SEEs may cause permanent faults and transient faults depending on the amount of energy transferred by the charges particles [Baig 2012]. Permanent faults occurring as a result of SEE include Single Event Latch up (SEL) which is a potentially damaging condition connecting parasitic circuit elements, which then stays "shorted" in an effect known as latch up, until the device is power cycled. This effect can happen between the power source and substrate, damagingly high current can be involved and the part may fail.

Transient faults cause no permanent damage and can be recovered from by applying the correct bit streams. There are many transient faults induced as a result of SEEs including Single Event Upsets (SEUs), Multiple Bit Upset (MBU), Single Event Functional Interrupt (SEFI), Single Event Transient (SET), Address decoding fault and Coupling fault. The most frequently considered fault is the Single Event Upsets (SEUs) or Single Bit Error (SBE) which is the flipping of an SRAM cell in the configuration memory. When a SEU occurs, a memory cell is flipped from a logical '1' to a logical '0', or from a '0' to a '1'. This leads the FPGA to operate in an undesired behavior. This persists until the configuration memory is overwritten. Multiple bit upset (MBU) or burst error is one of the transient faults induced as a result of SEEs which is caused by a single high-energy charged particle that forces several adjacent configuration bits to flip from a logical '1' to a logical '0', or from a '0' to a '1'.

Single Event Functional Interrupt (SEFI) is the fault induced when SEUs occurs in the configuration RAM of an active region of an FPGA [Gauer 2010]. Single Event Transient (SET) is a transient fault that happens when impacting ions induce voltage pulses on a combinatorial circuit in a device. If the induced voltage level increases above the level of the switching threshold and is of sufficient pulse-width, invalid data values can be propagated through the circuit. Moreover, Address decoding fault is any fault that affects address decoding with a certain address; no cell will be accessed, a certain cell is never accessed, with anything of the following: with a certain address, multiple cells are accessed simultaneously, and certain cell can be accessed by multiple addresses.

In addition, Coupling fault is the fault in which a write operation to one cell changes the content of another cell.

I.5. PARTIAL RECONFIGURATION

FPGA technology provides several benefits include the flexibility of on-site programming and re-programming with a modified design without re-fabrication. Partial Reconfiguration (PR) increases the flexibility of FPGAs by allowing the modification of an operating FPGA circuit by partially reconfiguring a portion of the FPGA with a partial BIT file. At the beginning, the FPGA is configured with a full BIT file; partial BIT files can be downloaded to modify reconfigurable regions in the FPGA without affecting the static parts of the design that are not being reconfigured. Partial Reconfiguration (PR) is modifying a subset of logic in an operating FPGA design by downloading a partial configuration file [Xilinx 2013].

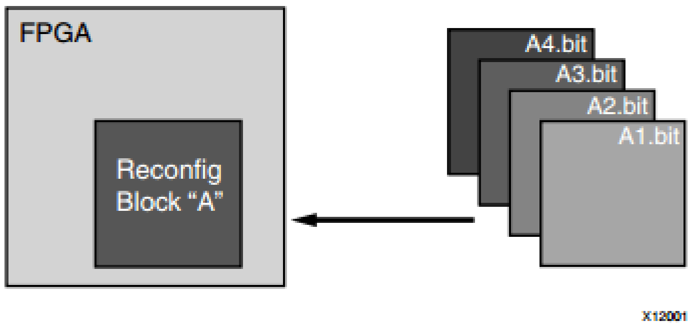


Figure 16: Partial Reconfiguration [Xilinx 2013**]**

As shown in Figure 16, the function implemented in Reconfig Block ”A” is reprogrammed by downloading one of multiple partial BIT files, A1.bit, A2.bit, A3.bit, or A4.bit. The logic in the FPGA design is categorized into two different types, reconfigurable logic and static logic. The gray area of the FPGA block represents static logic; however, the black portion labeled Reconfig Block “A” represents reconfigurable logic. The static logic remains working and is completely unaffected by the loading of a partial BIT file. The reconfigurable logic is reprogrammed by the contents of the partial BIT file.

A full bitstream may be as large as 1,712,614 bytes; however, a partial bitstream may be 2% of this size at 28,306 bytes. This reduction in configuration file size is an example of the several benefits of the PR which include:

- Reduce the size of the FPGA device required to perform the given function, with consequent reductions in cost and power consumption.
- Provide flexibility in choosing algorithms or protocols available to an application.
- Enable new techniques in design security.
- Improve FPGA reliability by applying fault-tolerant techniques.
- Accelerate configurable computing.

In addition to reducing size, weight, power and cost, Partial Reconfiguration allows the evolution of new types of FPGA designs that are impossible to implement without it. There is a software flow for using the Partial Reconfiguration capability in FPGA which is illustrated in the following Figure.

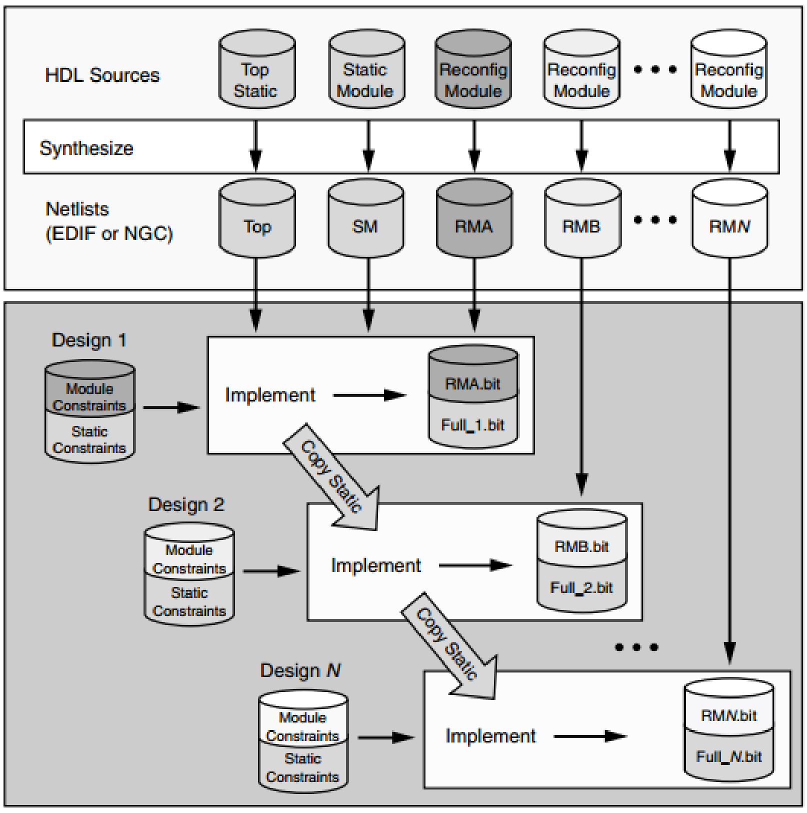


Figure 17: Partial Reconfiguration Software Flow [Xilinx 2013]

I.6. THESIS ORGANIZATION

The thesis organization is described as follows.

Chapter 2 includes the literature review. Fault-Tolerant systems are discussed in section 1. Section 2 includes FPGA Fault-Tolerant Techniques. The recovery techniques for permanent faults are discussed in section 3. The last section describes mission critical applications and their mission time.

Chapter 3 focuses on the first fault-tolerant approach for FPGA-based applications. The fault model is illustrated in the first section. Then, the proposed fault detection scheme is described in the second section. Section 3 discusses a problem occurring while testing the proposed fault detection technique to detect open faults in FPGA interconnections. The fault recovery technique of the first approach is illustrated in the fourth section. In Section 5, solution for the problem in section 3 is identified. In the last section of Chapter 3, a case study using the first approach is described.

Chapter 4 proposes the design of the second fault-tolerant approach which depends on using the Partial Reconfiguration (PR) property of FPGAs. In Section 1, the methodology is discussed while Section 2 contains the case studies in which the second approach is applied. The case studies are the Universal Asynchronous Receiver/Transmitter (UART) and the pacemaker. The last section of Chapter 4 contains the results of the second approach and a comparison to a previous technique. Chapter 5 concludes the work described in this thesis.

II. Literature Review

II.1. Fault-Tolerant Systems

Recent technologies have a great susceptibility to both transient and permanent faults. A transient fault describes a fault resulting from temporary environmental conditions such as radiation or electromagnetic interference and it causes no permanent damage. Accordingly, it can be recovered from by reconfiguring the device with the correct bit-stream. On the other hand, a permanent fault defines a fault that is continuous and stable in hardware until a corrective action is taken. It exists due to the aging of the circuit, electro-migration and Time-Dependent Dielectric Breakdown (TDDB). Both types of faults may cause system failure which has negative impacts in safety critical applications. Therefore, fault tolerant systems are considered necessary for safety critical applications. They were firstly introduced by Avizienis in 1967 [Avizienis 1967]. Moreover, they are currently one of the important long term challenges of the International Technology Roadmap for semiconductors.

II.1.1. Fault-Tolerant Systems

Fault tolerance is the ability of a system to continue operating properly in the presence of faults [Dubrova 2013]. There are two problems that increase the need for fault tolerance in safety critical applications. The first problem is that these kinds of systems are very complex. As the complexity of a system increases, its reliability decreases. Another problem, even after thorough production tests to overcome hardware defects and clean out software bugs, faults can still occur at runtime in situations outside the control of the designers. As a result, nowadays fault tolerant systems are considered a necessity especially for safety critical applications. Fault-tolerant systems are the systems that can continue its intended operation when some part of the system fails. This can be at a reduced level rather than failing completely. The development of a dependable system is one of the goals of having a fault-tolerant system. Dependability means the ability of a system to perform its planned level of performance [Dubrova 2013].

One of the properties that are required in fault tolerant systems is reliability. Reliability R (t) is the probability that the system works without a failure in the interval [0, t], given that the system is operating accurately at time 0. High reliability is required particularly for safety critical applications. It is a function of time and the time varies according to the nature of the system under consideration. Reliability represents the probability of success. On the other hand, unreliability Q (t) expresses the probability of failure which is the probability that the system fails in the interval [0, t], given that the system is operating accurately at time 0.

Failure rate λ is the expected number of failures per unit time. The Bathtub curve is a well-known curve that represents the typical evolution of failure rate over the life time of a system as shown in the following Figure:

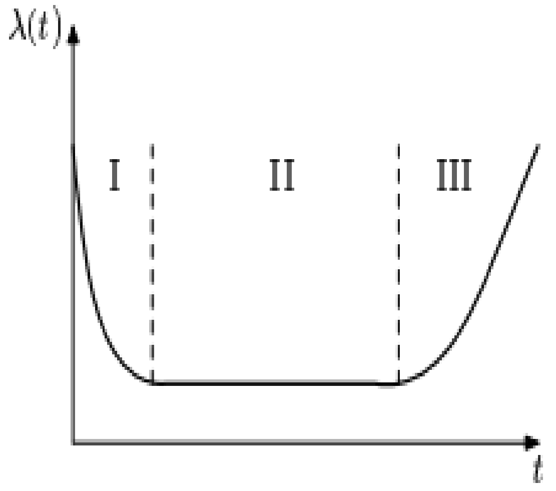


Figure 18: typical evolution of failure rate over the life time of a system [Dubrova 2013].

The bathtub curve has three phases: I is the infant mortality phase, II is the useful life phase and III is the wear out phase. In the infant mortality phase, the system is in the beginning of its life, the failure rate decreases sharply due to the frequent failures of weak components. While in the useful life phase, the failure rate seems to stabilize and remains constant. In the third phase, the components begin to wear out; accordingly, the failure rate increases.

During the useful life phase, failure rate has a constant value λ. Therefore, the system's reliability decreases exponentially with time:

$$R\ (t) = e^{\lambda t}$$

This law is called exponential failure law. It is shown in the following figure.

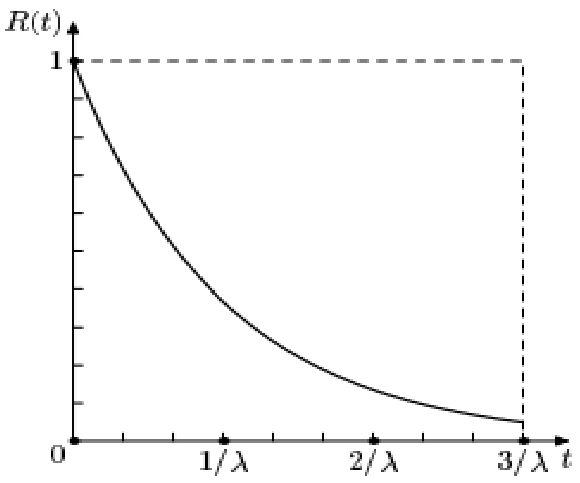


Figure 19: Exponential failure rate [Dubrova 2013]**.**

The exponential failure law is very essential for the analysis of reliability of components and systems in hardware. However, it can only be used in cases when the assumption that the failure rate is constant, is adequate.

There are some measures that are frequently used in measuring the dependability of fault tolerant systems such as Mean Time to Failure (MTTF), Mean Time to Repair (MTTR) and Mean time between Failures (MTBF). Mean Time to Failure (MTTF) is the expected time of the occurrence of the first system failure. Mean Time to Repair (MTTR) is the average time required to repair the system. Finally, Mean time between Failures (MTBF) is the average time between failures of the system and it is equal the sum of (MTTF) and (MTTR) if the repair makes the system perfect.

There are many approaches used in fault tolerant systems design. The mostly common used techniques in fault tolerant design are hardware fault detection techniques, hardware masking redundancy techniques and hardware dynamic redundancy techniques [Siewiorek 1998]. Hardware fault detection techniques are based on redundancy which means adding extra information or resources to those required during normal operation of the system. They include Duplication, Error Detection Codes, Self-Checking logic and Watch-Dog timers. Duplication means two identical copies of the system connected to a comparator. When a fault exists, the two copies become no longer identical and the fault is detected by the comparator. The advantages of duplication are low cost, simplicity and the fact that it can be applied in all types of applications. However, the comparator is a single point of failure. If the comparator fails, the fault cannot be detected.

Error detection codes are systematic applications of redundancy to information. These techniques include M-of-N codes, Checksums, Arithmetic codes and Cyclic codes. Self-checking logic is another fault detection technique that includes two types of circuits: Self-Testing and Totally Self-Checking (TSC). Self-Testing is a circuit that is self-tested; for every fault from a prearranged set of inputs, the circuit produces a non-code output for at least one code input. While Totally Self-Checking (TSC) is the circuit which is both self-testing and fault secure; this means that for every fault from a prearranged set of inputs, the circuit never yields an incorrect output for code inputs. Watch-Dog timers are a simple way of keeping track of process function. A timer is implemented as a process separate from the process which is being checked. If the timer is not reset before it expires, the checked process will be failed.

Hardware masking redundancy techniques are the techniques in which fault masking is used. Fault masking is a static form of redundancy in which the fault is masked without any intervention from the elements outside the targeted module. Some hardware masking redundancy techniques are N-Modular redundancy and Error Correcting codes.

N-Modular redundancy is the redundancy of the targeted module N times. Then, outputs of the N modules are connected to a voting system. The most common example is the Triple Modular Redundancy. Error correcting codes are the most commonly used technique of masking redundancy.

Hardware dynamic redundancy techniques include the reconfiguration of modules in the system in response to failure. There are several approaches of hardware dynamic redundancy techniques including Reconfigurable Duplication and Reconfigurable NMR. In the Reconfigurable Duplication technique, duplication is applied with two enhancements which are the ability to identify a faulty module and the ability to disconnect the faulty module and disable the comparator. Reconfigurable NMR includes the same technique of NMR; however, the faulty module can be replaced by unused spares and the voting system is dynamically modified.

A special type of fault-tolerant systems is the Autonomous Fault-Tolerant System (AFTS) which is discussed in the following section.

II.1.2. AUTONOMOUS FAULT-TOLERANT SYSTEMS

An Autonomous Fault-Tolerant System (AFTS) is a system which has the ability to reconfigure its own resources in the presence of permanent faults and spontaneous random faults in the silicon in order to maintain its functionality [Iturbe 2010]. It can modify its architecture on-the-fly which makes it a fully adaptive system. Moreover, this make a great reduction in the maintenance cost of a system.

SRAM-based Field Programmable Gate arrays (FPGAs) is one of the common devices used in Autonomous Fault-Tolerant System (AFTS) implementation. There are two factors that enable SRAM-based FPGAs to be a suitable platform for this kind of fault-tolerant systems. Firstly, it can deal with faults and defects in a more flexible way compared to other programmable devices. Moreover, the Dynamic Partial runtime Reconfiguration property existing in Xilinx FPGAs allows the user to reconfigure a specific part of the FPGA with a new configuration during runtime while the rest of the FPGA is still working. This maintains the capability of changing the functionality and architecture of the system on-the-fly. This reconfiguration should occur automatically without an external human intervention to better to adapt for fault existence. Accordingly, this supports the implementation of a more advanced Autonomous Fault-Tolerant Systems (AFTSs).

The ideal architecture for building an Autonomous Fault-Tolerant System (AFTS) includes several independently and dynamically reconfigurable areas. These areas are used in the recovery mechanism of AFTS system. If a permanent fault occurs, the faulty module is recovered in one of these areas. There are three technological levels for implementing AFTS in Xilinx SRAM-based FPGAs which are TL0 (Minimal), TL1 (Self-Reconfiguration), TL2 (Dynamic Module Relocation) and TL3 (Online Re-synthesis). They will be discussed in Section II.3 with examples of implemented techniques for each technological level.

II.2. FPGA Fault-Tolerant Techniques

II.2.1. Techniques for Transient Faults

FPGA devices are mostly vulnerable to radiation-induced faults (SEUs) which alter the logic state of the memory cell. As a result, many approaches have been proposed to mitigate SEU errors in FPGAs [Pratt 2007-Yui 2003]. In addition, most of the fault-tolerance techniques are based on redundancy techniques such as Triple Modular Redundancy (TMR) [Kastensmidt 2004]. TMR is a fault-tolerant redundancy technique in which three systems perform a process and that result is processed by a majority-voting system to produce a single output. If any one of the three systems fails, the other two systems can correct and mask the fault. Furthermore, Error Correcting Codes (ECC) are also a commonly used fault tolerant technique in FPGAs [Peterson 1972]. However, some approaches are based on scrubbing and read-back [Carmichael 1999, Carmichael 2000]. Scrubbing is the comparison of the data in the configuration SRAM continually to the golden copy of the design in the external non-volatile device. When any error is detected, it overwrites the configuration values SRAM with original values [Gauer 2010].

II.2.2. Techniques for Permanent Faults

Several techniques have been proposed to detect permanent faults in FPGA-based systems. For example, in [Ramya 2013], a Built-In Self-Test (BIST) technique is used to detect various types of single and multiple faults in both interconnect and logic blocks. BIST technique mainly consists of Blocks Under Test (BUT), and Output Response Analyzer (ORA). This technique has a large area overhead. The techniques used for permanent fault recovery will be discussed in the following section.

II.3. RECOVERY TECHNIQUES FOR PERMANENT FAULTS

Permanent faults induce permanent damages in the silicon of the FPGA; accordingly; the faulty part of the FPGA becomes unusable. This type of fault can be recovered by downloading a new configuration of the circuit in the FPGA avoiding the permanently damaged part of the FPGA. The system which can recover from permanent faults occurring in the silicon of the FPGA to keep operating normally is known as Autonomous Fault-Tolerant System (AFTS).

Nowadays, AFTS implementation is the state of art because the researches presented to recover from permanent faults is insufficient compared to the techniques proposed to recover transient faults. The need for having more researches in hard errors recovery is outlined in the international Road Map for Semiconductors [ITRS 2011]. AFTS differs from other fault-tolerant systems in that it can deal perfectly with permanent faults occurring during runtime which are not predictable at design phase. Hence, AFTS increases the lifetime and availability of the targeted system.

The most suitable devices for Autonomous Fault-Tolerant System (AFTS) implementation in future commercial applications are dynamically and partially self-reconfigurable FPGAs. Modern Xilinx FPGAs include Dynamic Partial runtime Reconfiguration property which plays a great rule in achieving autonomy because it enables the configuration of a portion of the FPGA to be changed during runtime. Moreover, they enable the access to the FPGA configuration memory through Internal Configuration Access Port (ICAP) without the necessity of having external components that control the Partial Reconfiguration (PR) process. Using ICAP enables different parts of the FPGA to be reconfigured sequentially and decreases the reconfiguration speed. Accordingly, a standard configuration takes only milliseconds.

II.3.1. THREE TECHNOLOGICAL LEVELS FOR XILINX FPGAS

Three technological levels for Xilinx FPGAs are proposed to classify the implemented techniques in the literature used for building Autonomous Fault-Tolerant System (AFTS). This classification shows the way that each level produces fault-aware configurations for AFTS implementation. Three technological levels are TL0 (Minimal), TL1 (Self-Reconfiguration), TL2 (Dynamic Module Relocation) and TL3 (Online Re-synthesis) [Iturbe 2010].

II.3.1.1. TL0 (MINIMAL)

In this level, the designer controls both the reconfiguration resources which is ICAP or SelectMap port in Xilinx FPGAs and the fault diagnoses circuitry which is Frame ECC as shown in the following Figure.

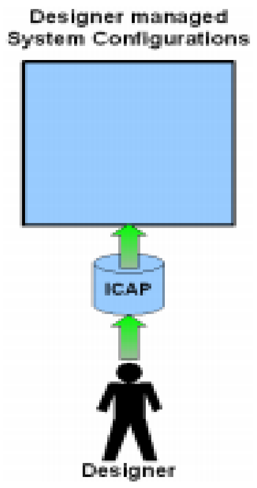


Figure 20: TL0 [Iturbe 2010].

Frame ECC is a built-in diagnosis block in current Xilinx FPGAs which is responsible for automatically checking the ECC bits in the configuration frames of the bitstream. When an error exists, this check identifies the position of corrupted bits within the bit file. Therefore, the affected resources are located. The designer implements both architecture exploration and the updated architecture of the system. TL0 are not autonomous because they are not aware of their architecture and need remote maintenance during runtime. However, it has the minimal of AFTS implementation.

II.3.1.2. TL1 (Self-Reconfiguration)

Since TL1 systems can update their configuration without external interference, they have a minor role in their autonomous adaptation. However, they do not have the ability to generate new configurations on their own. The new configurations are generated by the designer in the design time by taking in consideration the avoidance of the permanently damaged areas.

In TL1, the system is divided into several slots. Several precompiled configurations are generated for each slot; accordingly, the complete configuration of the system is partitioned into several partial partitions as shown in the following Figure.

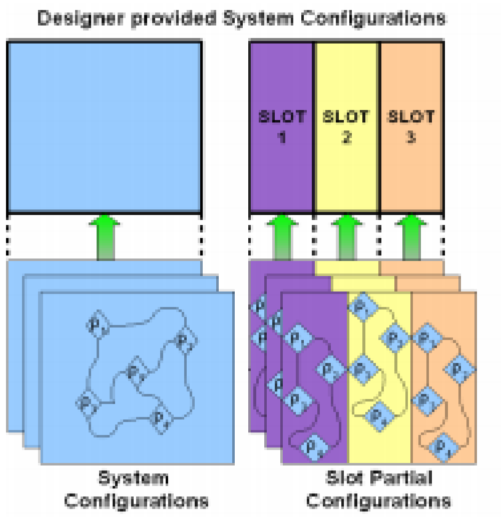

Designer provided System Configurations
SLOT 1
SLOT 2
SLOT 3
System Configurations
Slot Partial Configurations

Figure 21: TL1 [Iturbe 2010].

The precompiled configurations contain different possible combinations which increases the number of permanent faults that can be recovered from. The observable drawbacks of this technological level are that the faulty situations that can be recovered from are limited to those considered at design time because each slot or partial block is constrained to a certain location and boundaries which cannot be changed during runtime. Moreover, as the number of faulty situations considered in the design phase increases, the number of the precompiled configurations generated increases. Therefore, a larger memory space is required for storing them.

II.3.1.3. TL2 (Dynamic Module Relocation)

Module relocation is the basic concept used in TL2 technological level. Accordingly, modules are the primary structural and functional units of this type of Autonomous Fault-Tolerant Systems (AFTSs). The flexibility of module relocation enables the generation of new system bitstreams on-the-fly to recover dynamically from permanent faults. Relocatable bitstreams comprise configuration data which can be reused in multiple reconfigurable regions on the FPGA [Becker 2007]. The faulty modules can be relocated in another position by managing the configuration frame addresses during the reconfiguration process by shifting the configuration frames of the faulty module within the full system configuration bit file [Iturbe 2010]. As result, there is no need for generating several partial bitstream for the same module for each position where it can be placed. Figure 22 shows TL2.

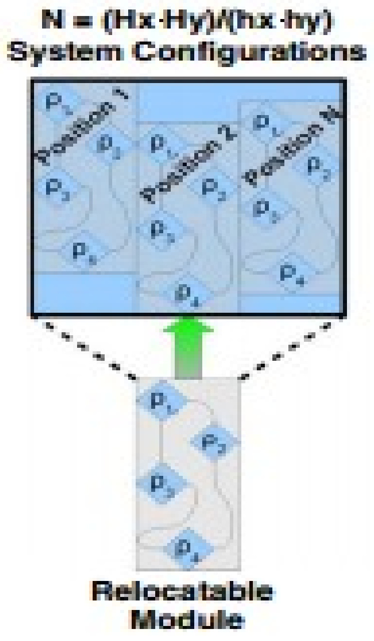


Figure 22: TL2 [Iturbe 2010].

Since any module can be easily placed at any 2-Dimentional position h_x x h_y within the FPGA, $(H_x \, . \, H_y)$ / $(h_x \, . \, h_y) - 1$ extra precompiled configurations for a FPGA which includes $(H_x$ x $H_y)$ resources, can be produced. No redundant partial configurations are generated in TL2 systems so they do not require extra storage requirements compared to TL1 systems. The placement and routing of the modules are carried out online; therefore, the autonomy of fault tolerant systems implemented by this kind of technological level increases.

Although TL2 is the highest technological level that can be applied, there are numerous challenges to be overcome when implementing TL2 systems such as:

- Clock signal distribution related concerns:
  The clock signal must be connected to both the initially configured location of the module and new location used when relocating the module. Unfortunately, the feature of handling the logic resources not used by the original system is not available in Xilinx tools.
- Structural non-regularity related concerns:
  Heterogeneous logic resources by-pass is preferred to improve the relocability of modules. This needs a minor modification in the internal architecture of the modules which is not supported in Xilinx tools. The following Figure contains an example of module relocation.

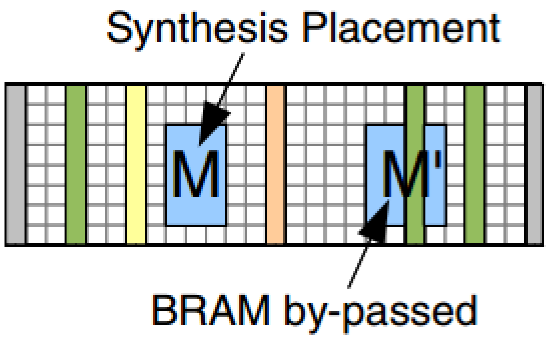


Figure 23: Module relocation [Iturbe 2010].

As shown in Figure 23, module M cannot be relocated in M' which contains BRAM column except if BRAM column is by-passed.

- Relocatable partial bitstreams generation related issues:

  The static routes which pass into the reconfigurable area must be preserved because they operate even after the module is relocated. As an example of static routes preservation, the usage of an Exclusive OR (XOR) to combine the partial bitstream with its corresponding static configuration.
- Configuration granularity related issues:

  One of the important limitations of module relocatabilty is the size of the reconfiguration frame. If the size of the new location does not have the same size of the original module location, the process will not be successful. There are two directions to relocate module which are horizontal relocation and vertical relocation. The horizontal relocation constraint is that the size of the original and new locations must be the same. However, vertical relocation is limited by the amount of clock regions built in the FPGA so it cannot be applied within a clock region during the runtime. Figure 24 shows different proposed zones for module M relocation.

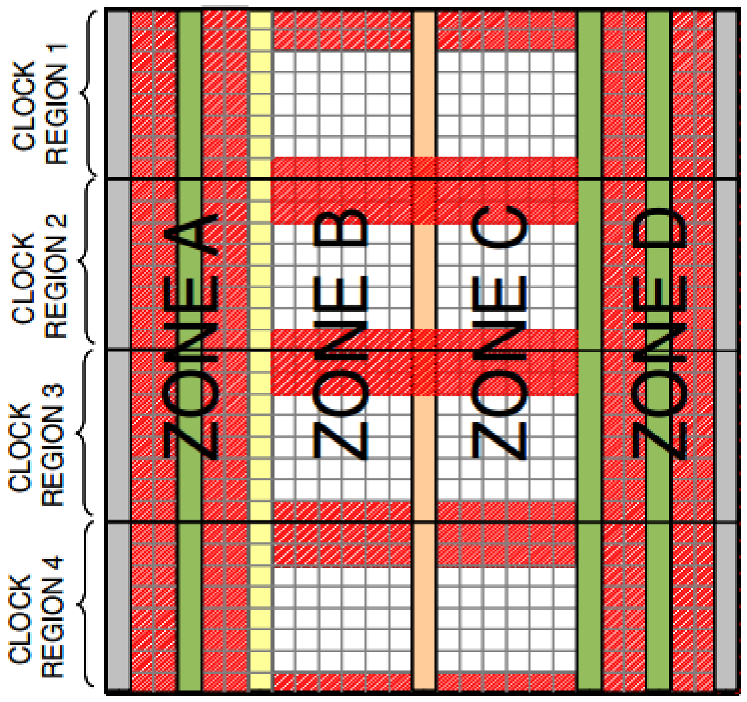


Figure 24: Different proposed zones for Module relocation [Iturbe 2010].

Zone A and Zone D contains heterogeneous logic resources such as BRAM so module M cannot be relocated in these zones. Module M can be located only in the non-shadowed areas of Zones B and C because the vertical placement within a clock region cannot be changed.

II.3.1.4. TL3 (Online Re-synthesis)

The systems implemented using TL3 technological level can recover from permanent faults autonomously by creating a new configuration during runtime for every existing fault. This is implemented by re-synthesizing modules online so TL3 systems can produce an infinite number of different configurations as shown in Figure. 25.

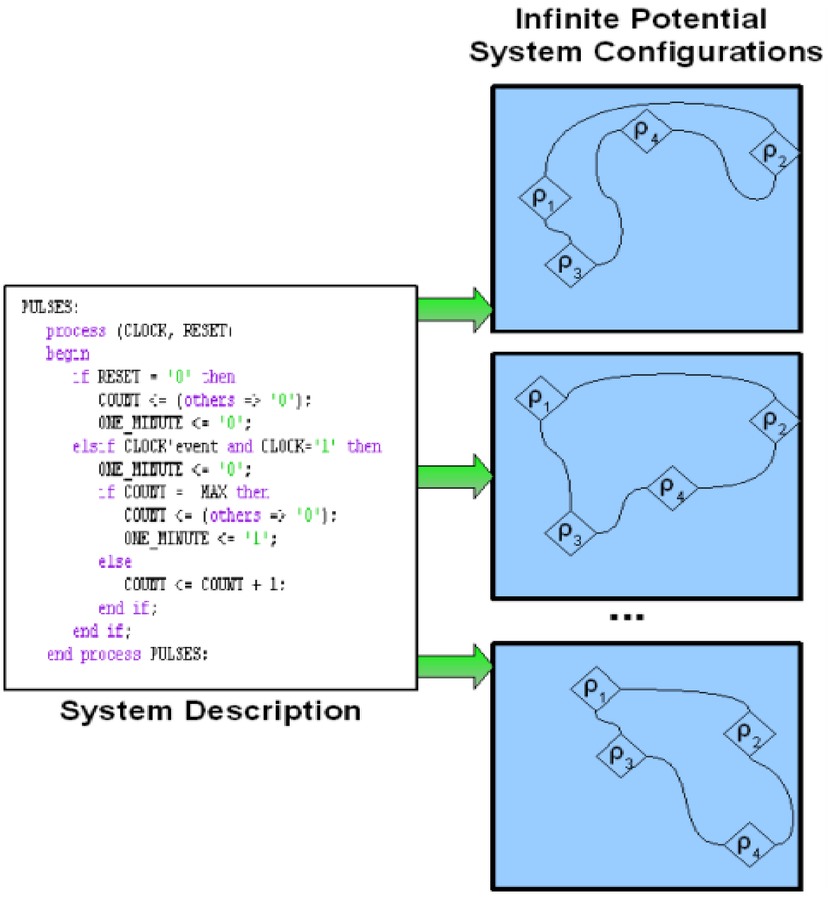

```
PULSES:
   process (CLOCK, RESET)
   begin
      if RESET = '0' then
         COUNT <= (others => '0');
         ONE_MINUTE <= '0';
      elsif CLOCK'event and CLOCK='1' then
         ONE_MINUTE <= '0';
         if COUNT =  MAX then
            COUNT <= (others => '0');
            ONE_MINUTE <= '1';
         else
            COUNT <= COUNT + 1;
         end if;
      end if;
   end process PULSES;
```

Figure 25: TL3 [Iturbe 2010].

The online re-synthesis property works at the finest granularity of the FPGA device. It allows the avoidance of the permanently damaged areas and decreases the waste of FPGA resources.

There are several limitations for TL3 systems implementation which include:

- The generation of the bitstream of the online re-synthesis is exclusively made by high level abstraction synthesis tools such as Xilinx PAR and bitgen tools.
- Time and computational resources needed for applications in which the fault rate is high are unachievable if it is implemented using TL3 technological level.
- The fine-grained access to the bitstream is unfeasible except for the bitstreams that are formerly identified by reverse- engineering or by using JBits API which is not supported in the recent devices of Xilinx FPGAs.

Evolutionary Algorithm (EA) can be used instead of high level abstraction synthesis tools to find the suitable configuration by which the TL3 system can operate normally. It randomly changes the configuration of the system. After that, the behavior of the new configuration of the system is evaluated. If the system operates normally, the configuration is preserved. If it does not work probably, the new configuration will be removed.

II.3.2. DIFFERENT APPROACHES FOR PERMANENT FAULT RECOVERY

According to the literature survey, only the following techniques are found for permanent faults recovery. It is observable that most of the AFTS systems are implemented using TL1 technological level because all the requirements needed for building TL1 systems are available. Moreover, the implementation of both TL2 and TL3 systems has numerous limitations as discussed in subsection II.3.1. In this subsection, permanent fault recovery techniques will be discussed. They are classified according to the technological level used in their implementation.

II.3.2.1. TL0 TECHNIQUES

In [Xu 2003], a TL0 automated fault recovery system for networked FPGA systems is described. The proposed recovery system allows faulty systems (fault tolerant clients) access to computationally-superior processing resources (reconfiguration servers). These servers help in the recovery effort of rebuilding a configuration bitstream. Following reconstruction of programming data, a new configuration is passed to a fault tolerant client via the network environment and remote computation is restarted. To speed the recompilation effort, a self-contained CAD mapping system targeting commercial Xilinx Virtex devices has been developed. A timing driven incremental FPGA router has been developed and integrated into VPR to overcome permanent faults in the FPGA routing fabric.

II.3.2.2. TL1 Techniques

Several techniques are implemented using the TL1 technological level which is based on dividing the system into several partitions and generate several precompiled configurations for each partition. If a fault exists, a new configuration is downloaded to recover from the permanent fault. Some of the TL1 techniques are discussed as follows:

In [Psarakis 2012], the proposed technique creates several precompiled configurations for the design containing different places for spare slices to recover from any faulty module. Accordingly, the system is divided into a set of reconfigurable modules. A reconfigurable partition region is defined in the physical layout of the FPGA devices for each reconfigurable module. This reconfigurable partition region should be large enough to provide sufficient number of spare resources used for the different precompiled configurations. Also, it should include the area needed for Dynamic Partial Reconfiguration mechanism to locate the partition pins and separate the reconfigurable module from the static part of the system. Figure 26 shows four alternative configurations for an 8x4 reconfigurable partition.

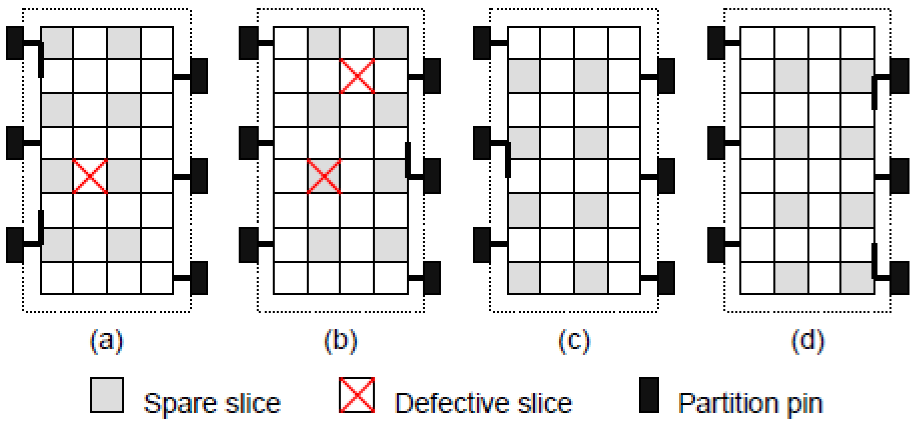


Figure 26: Four alternative configurations for an 8x4 reconfigurable partition [Psarakis 2012].

The fault detection technique used is Duplication With Comparison (DWC). Each reconfigurable module is duplicated. Both duplicated modules are partially reconfigurable and alternative configurations are created for them. When a fault is detected, all alternative configurations are tried until the configuration that tolerates the existing fault is found because the fault detection approach cannot locate the defective slices in the faulty module. Therefore, any single fault existing in the reconfigurable module can be recovered from by an alternative configuration. The runtime swapping of the precompiled configurations of the module does not affect the operation of other modules. It provides single error capability per reconfigurable module.

The advantage of the [Psarakis 2012] technique is that it can recover from several faults so the lifetime of the system will be extended. The disadvantages of the technique in [Psarakis 2012] are that, as the number of the alternative configuration increases, the memory space needed for storing the partial bit files increases. Moreover, the fault repair time will increase because all the precompiled configurations will be tried until the fault is recovered. The technique in [Psarakis 2012] is applied to 5-stage pipelined processor core in which the ALU and the instruction decoder are critical modules.

There is another technique in [Di Carlo 2010] which tolerates both transient and permanent faults. The system comprises Replaceable Functional Units (RFU), a group of Spare Functional Units (SFU), a Reconfiguration Manager and a Reconfigurable Area. The Reconfiguration Manager is responsible for deciding if an SFU is used instead of the corresponding Replaceable Functional Unit according to the detection signal of the fault detection mechanism.

Replaceable Functional modules can be critical or non critical and the Spare Functional modules can be hardware or software. Each type of Replaceable Functional module has its own detection and recovery mechanisms. Critical RFUs use the concurrent error detection technique and can be replaced by a hardware spare unit; however, non critical RFUs may or may not have a BIST capability and can be replaced by software SFU. Critical RFUs are initially placed in static area on the device and when becoming defective, they are replaced by an SFU mapped inside the Reconfigurable Area. On the other hand, non critical RFUs are initially mapped inside the Reconfigurable Area and they are replaced by software SFU if the critical RFU becomes faulty and the critical spare unit is downloaded in the Reconfigurable Area as shown in the following Figure.

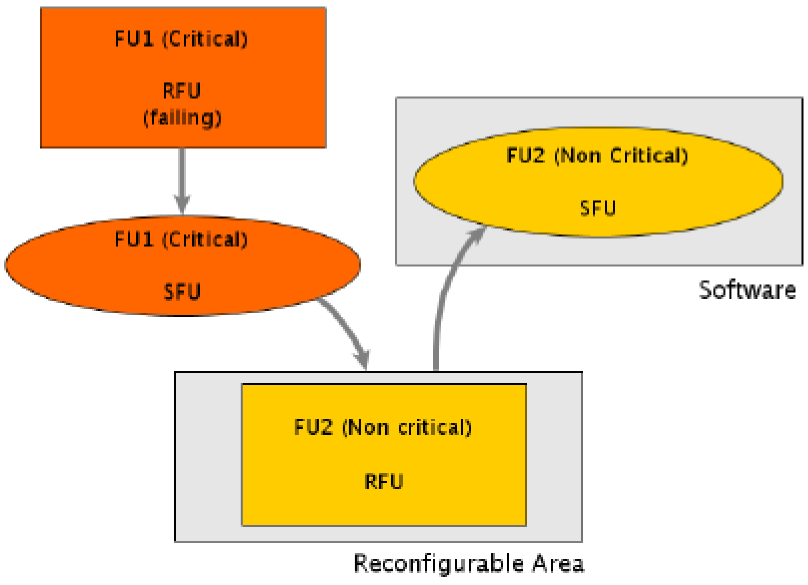


Figure 27: Functional Units swapping architecture [Di Carlo 2010].

Figure 27 shows FU1 which is the critical Functional Unit and FU2 which is the non critical Functional Unit. When FU1 becomes defective, the equivalent SFU is placed inside the Reconfigurable Area. As a result, the previously allocated FU2 is removed and will be executed in software. That results in performance degradation which means that the system will operate with limited functionality.

The Reconfiguration manager is responsible for replacing the RFUs with SFUs according to the fault detection signal of the detection mechanism. Swapping process of a RFU with a SFU happens with the system being totally unaware of it because the Reconfiguration Manager has the facility to freeze the involved circuits while the configuration takes place. The “freezing” technique can be used by the Reconfiguration Manager to differentiate between transient and permanent faults. As a result, unnecessary reconfigurations are avoided. It is implemented by a bus master that is an AMBA AHB master.

There is a case study of a microprocessor implemented using the technique in [Di Carlo 2010] in which the ALU was the critical module and the non-critical module was the DES encryption module. The detection technique used is the online concurrent error detection technique. When the ALU module becomes defective, a new copy of the ALU is downloaded in the reconfigurable area contains the DES encryption module to recover from the fault. After the recovery, the encryption module is executed in software. [Di Carlo 2010] have several drawbacks which include that, after the recovery process, the system will work in graceful degradation so it is operating in constrained functionality. The reconfigurable area is used from the beginning so more static power is consumed. Moreover, the suggested technique in [Di Carlo 2010] uses complicated recovery hardware. Only a specific module in the design (critical module) can be recovered from using [Di Carlo 2010] compared to other techniques that allow the recovery of any faulty module.

In [Pradeep 2014], the proposed system used king spare allocation technique and Dijkstra's shortest path shifting to carry out autonomous repair. In this technique, a spare cell is shared by eight working cells.as shown in the following Figure.

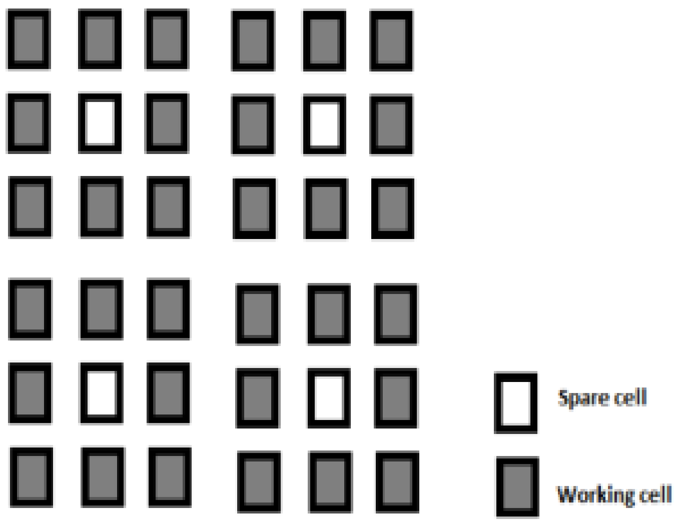


Figure 28: king spare allocation [Pradeep 2014].

As shown in Figure 28, eight working cells with a spare create a section. If a spare is available in a section, then when any working cell in a particular section becomes faulty, it is recovered by the spare cell in the same section regardless whether the fault is transient or permanent. If a spare is unavailable in a particular section, self-test is firstly performed to identify the type of fault. If the fault is permanent, the Dijkstra's shortest path shifting technique is used to shift the faulty working cell to the nearest available spare. When the shortest path is determined, it is checked whether all the cells in that path are fault free. If the shortest path is fault free, the shifting of cells is implemented along that path. On the other hand, if the shortest path contains no fault and the spare is not available, a second shortest fault free path is determined. The permanent fault coverage is constrained by the number of spares available. If the fault is transient, the cell undergoes a delayed transient fault recovery to reduce the routing complexity. The advantage of [Pradeep 2014] is that it has minimum area overhead because it uses only one spare for every eight working cells. However, this limits the number of faults that can be tolerated.

In [Bolchini 2013], an autonomous fault-tolerant system is implemented. It is composed of three main modules: an SRAM-based FPGA hosting the hardened payload application, a rad-hard FPGA hosting the reconfiguration controller and a hardened memory storing the FPGA configurations for recovery as shown in the following figure.

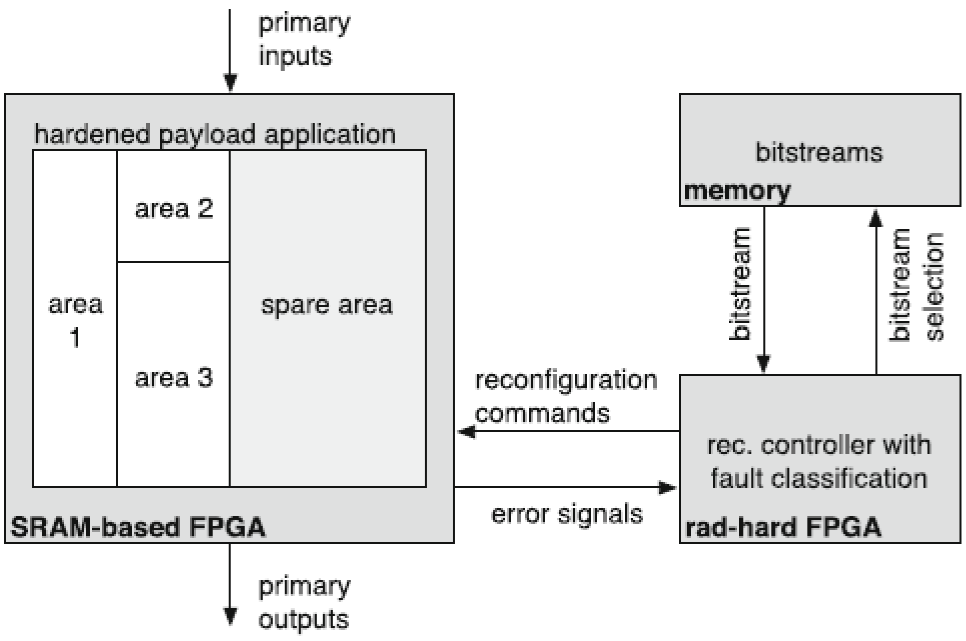


Figure 29: Proposed technique in [Bolchini 2013].

The first module establishes the specific reconfigurable fabric on which the circuit under design is developed and deployed on. As a result, it can be updated during run time. The circuit under design, called payload application, consists of data processing functionality which is the targeted application. Figure 29 shows that the system consists of three areas and there is a spare area used for recovering permanently damaged areas. The other two elements are customized to support the monitoring and reconfiguration features essential to implement an autonomous fault-tolerant system. The entire FPGA is reconfigured with precompiled bit streams, when a permanent fault exists, the circuit will not use the faulty FPGA resources. Each precompiled configuration contains a different mapping of the areas of the system. Each configuration can tolerate one permanent fault as shown in the following figure.

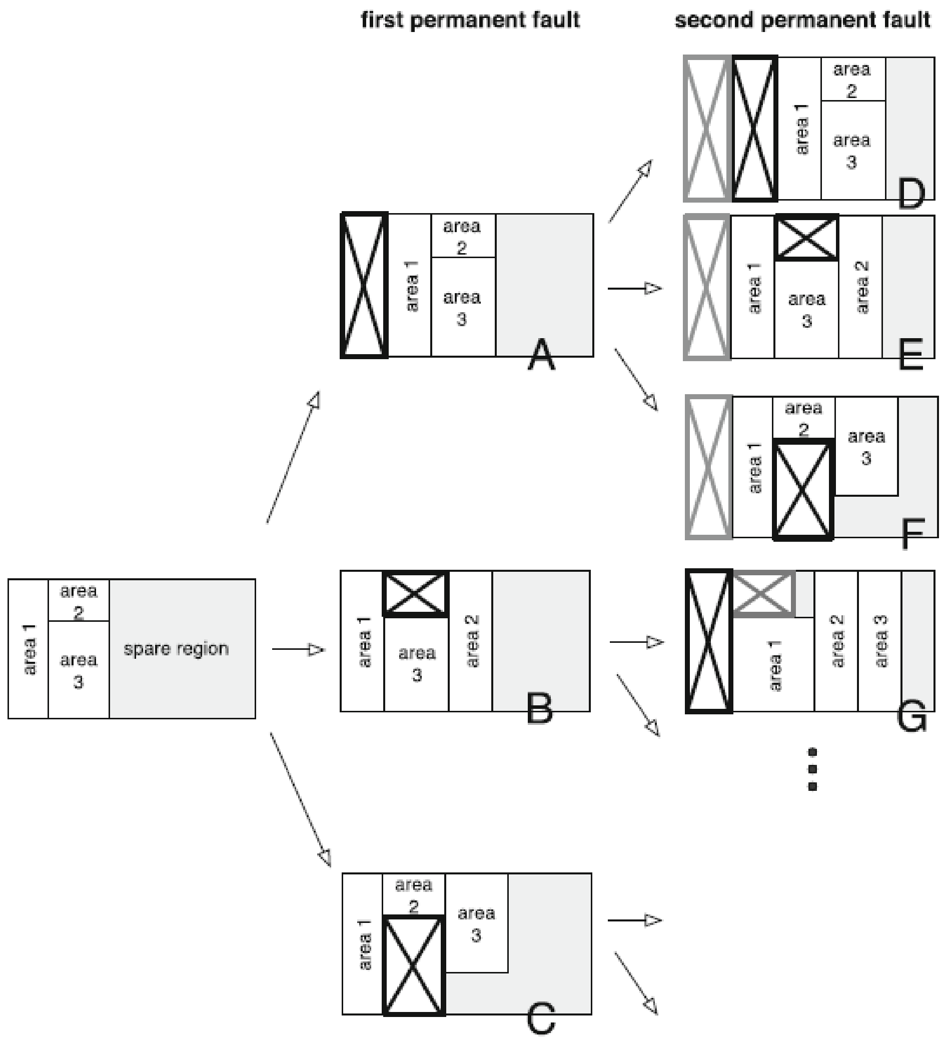


Figure 30: Recovery strategy for permanent fault [Bolchini 2013].

The memory space needed to store the full BIT files will be very large and also the recovery time is increased because the whole FPGA is reconfigured with precompiled configurations not only a portion of it.

There is a different technique in [Zhang 2013] in which the FPGA is divided into cells. The size of each cell is the same and equals the size of an actual module. Also, the targeted application is divided into independent function modules. Each module has the same size and is located in one cell as shown in the following figure.

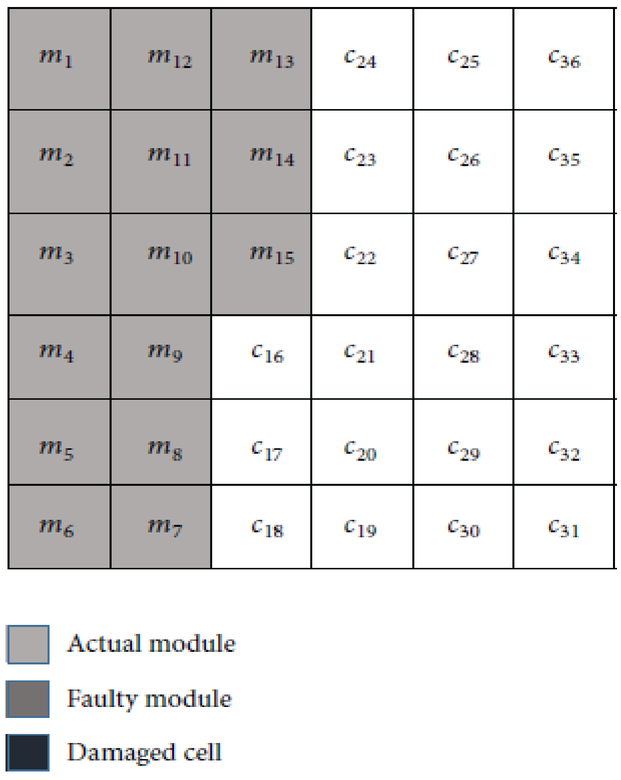


Figure 31: Cells partition and module placement [Zhang 2013].

Each functional module includes a fixed sized fault detection module. If a permanent fault occurs in one of the modules, all the modules will be shifted to the available cells one by one. This mechanism will be repeated, until all available cells in the FPGA have been used. After every reconfiguration, the configuration is checked if there is any routing problem in the new layout and if the system is running well.

Figure 32 shows the proposed recovery technique proposed in [Zhang 2013].

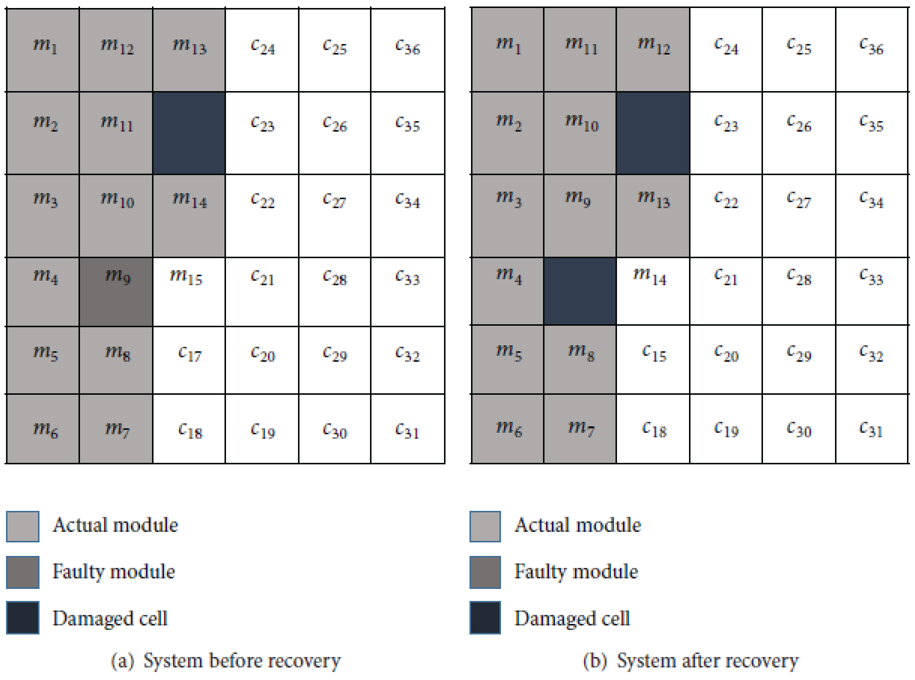


Figure 32: Fault recovery mechanism in [Zhang 2013].

When m_9 becomes faulty, it is shifted to the next available cell as shown in Figure 32. The proposed technique in [Zhang 2013] is not efficient because if a single module becomes defective, all the following modules must be reconfigured.

The approach proposed in [Di Carlo 2014] divides the FPGA into several reconfigurable tiles: logic tiles, interconnection tiles, and recovery tiles. The logic tiles contain the circuits that perform the computation and data processing. The interconnection tiles are added to the design to control communication between design modules in logic tiles. Moreover, the recovery tiles are used as spare tiles. The proposed technique consists of the application FPGA which is the SRAM-based FPGA hosting the hardware functionality, a fault manager which contains the configuration controller and fault classifier as shown in the following figure.

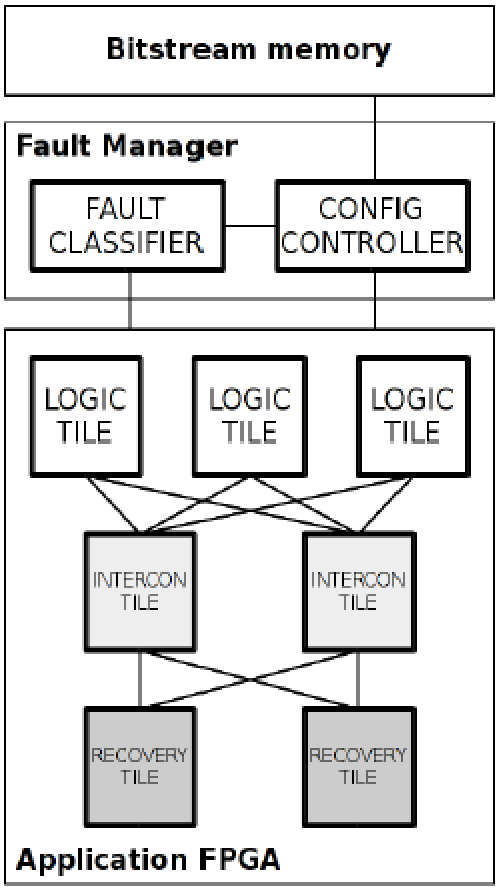
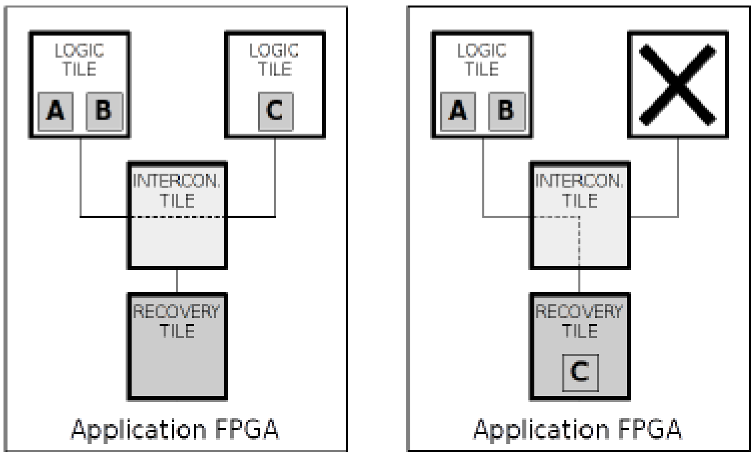
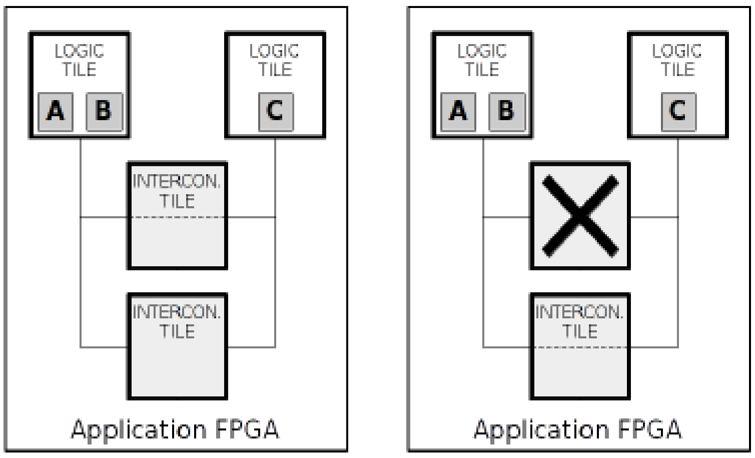


Figure 33: Proposed system architecture in [Di Carlo 2014].

The fault manager monitors the faults that happen in the application FPGA. The fault classifier detects faults and identifies their location. This is implemented by using periodic tests on the application FPGA or by collecting error signals generated by fault detection hardware in the application FPGA. The Configuration Controller runs the recovery operations so the system can be restored back to normal operation. Figure 34 shows the proposed recovery mechanism.



(a) The system without any faulty logic tiles (b) The system recovered from a faulty logic tile



(c) The system without any faulty interconnection tiles (d) The system recovered from a faulty interconnection tile

Figure 34: Proposed recovery mechanism in [Di Carlo 2014].

Assume that a given system consists of three modules A, B and C. The system is partitioned using the partitioning algorithm as in Figure 34 (a). The first logic tile contains module A and B, while the second logic tile contains module C. If the second logic tile becomes defective, it is relocated into a recovery tile and the interconnection tile is also reconfigured as shown in Figure 34 (b). The recovery of faulty interconnection tiles is illustrated in Figure 34 (c) and (d). When the interconnection tile becomes faulty, it is recovered in one of the backup interconnection tiles.

There is a proposed algorithm known as partitioning algorithm which is used to find a feasible partitioning which offers the maximum number of faulty tiles the system can tolerate. The number of configuration files needed for the design according to the number of tiles specified for each type of tiles is calculated using the equation shown in the following figure.

$$n\_conf\_files = n\_logic\_tiles \times n\_faults + \\ + \prod_{i=0}^{n\_faults-1} (n\_logic\_tiles - i) \times (n\_faults + 1) + \\ + n\_logic\_tiles$$

Figure 35: Partitioning methodology equation [Di Carlo 2014].

The first term in the equation resulted from the relocation of functions implemented in logic tiles to recovery tiles. The second term is due to interconnection tiles. There are n faults + 1 interconnection tiles that must offer connection for all the possible combinations of logic tiles relocated to recovery tiles. Finally, the third term of the equation takes into account the fact that faulty logic tiles must be reconfigured with an empty bitstream.

The advantage of [Di Carlo 2014] is that it can recover from more than one fault. However, it has complex recovery hardware and consumes a large area overhead. Additionally, it needs larger memory space to store configuration files compared to other techniques.

II.3.2.3. TL2 Techniques

Most recent software systems in the area require the precompilation of partial bitstreams that are assigned to partially-reconfigurable regions (PRR) which need large memory storage resources. However, this problem is mitigated in TL2 systems; a few works are implemented using TL2 technological level.

In [Drahonovsky 2014], an autonomous fault tolerant system using TL2 technological level is implemented on Xilinx Field Programmable Gate Array (FPGA) where partial bitstream relocation (PBR), configuration memory readback and internal registers restoration techniques are supported. Partial bitstream relocation (PBR) means that the relocation procedure is based on the bitstream major address modifications and design where the relocation of individual modules including their internal states, is supported. Using partial bitstream relocation (PBR) reduces the number of partial bitstreams stored in memory compared to the number of configurations generated by TL1 systems. Moreover, it saves the implementation time and generally increases the flexibility of the reconfigurable system.

II.3.2.4. TL3 Techniques

TL3 systems implementation is based on creating a new configuration during runtime for every existing fault. This is implemented by re-synthesizing modules online so an infinite number of different configurations is generated which is used in permanent fault recovery. Evolutionary Algorithm (EA) can be used instead of high level abstraction synthesis tools to find the suitable configuration by which any fault existing in TL3 system can be tolerated. An example of using Evolutionary Algorithm in TL3 implementation is described as follows:

In [Salvador 2011], a fault-tolerant system is implemented by an autonomous fully embedded evolvable hardware system which uses a combination of partial dynamic reconfiguration and an evolutionary algorithm (EA). The system can self-recover from both transient and permanent faults. The main components of the autonomous self-recover technique are the Reconfigurable Core (RC) and Reconfiguration Engine (RE). RC is the processing array, which is reconfigured by the RE during the adaptation process exploiting Dynamic Partial Reconfiguration.

Using self-reconfiguration by using the internal ICAP configuration port and an embedded Evolutionary Algorithm (EA) provides the system with the required autonomy. Evolutionary Algorithm (EA) is run on the embedded MicroBlaze processor which issues the required reconfiguration commands to the Reconfiguration Engine (RE), which configures the Reconfigurable Core (RC). When a fault exists, a new configuration is resynthesized online by the Reconfiguration Engine (RE) based on the Evolutionary Algorithm to self-recover. The advantage of the proposed technique is that the extra resource requirements are negligible and the recovery time is very small.

II.4. Mission-Critical Applications and Mission Time

Mission time is an important metric in the context of system reliability. The Mission Time is defined as the time at which system reliability falls below a certain pre-determined level R_{min} [Siewiorek 1998].

For mission-critical applications which have a minimum lifetime requirement, the Mission Time is a very appropriate metric. From another point of view, it is known that the exponential distribution is the most commonly used distribution for the time to failure random variable [Siewiorek 1998]. The relation between the reliability and the failure rate can generally be described by the following equation:

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

For short mission times, the probability of more than one failure is low. Hence the focus in this thesis will be on mission-critical applications with short mission times. In these short mission times, it is not expected to have more than one failure. Hence, the proposed techniques will focus on the detection and recovery from the first failure. The advantage of such a philosophy will the low cost required for the addition of the fault-tolerance features.

III. Fault Tolerant Technique for FPGAs without Partial reconfiguration

This chapter talk about the first proposed fault tolerant technique for FPGA.

III.1. Fault Model

The interconnection of an FPGA occupies 80–90% of its total area; however, the logic area occupies only the remaining part [Brown 1992]. Consequently, faults in interconnections of the FPGA are the main motivation of this thesis. The fault model used in this work is based on the one in [Olbrich 1996]. This fault model consists of 5 faults per transistor which are Short Gate-Source fault, Short Gate-Drain fault, Short Drain-Source fault, Open Source fault and Open Drain fault. In this thesis, the focus will be only on two of these faults, specifically the open drain fault and the open source fault along with open wires. This is similar to the stuck open faults (SOP) described in [Li 2005]. It is a complete break among circuit nodes which should be connected [Li 2005]. There are several examples of SOP defects which are missing metal wires, missing contacts or vias contact misalignment and bad transistors.

Since the main building block of the programmable interconnects of the FPGA is the Programmable Switch Matrix (PSM), the fault model focuses on the open faults in the Programmable Interconnect Points (PIPs) of it. PIP is a CMOS pass transistor that is controlled by an SRAM cell. A PIP stuck-open fault causes the PIP to be permanently open regardless of the value of the SRAM cell controlling the PIP. [Tahoori 2002]. It is assumed that only one fault occurs at a time [Nagi 1996].

Figure 36 shows examples of open drain and open source faults in PIP.

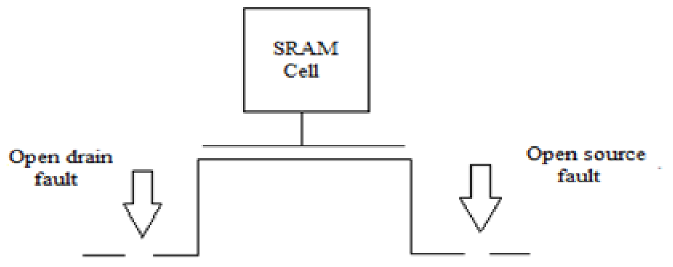


Figure 36: Open faults in pass transistors

III.2. Fault Detection Scheme

Hardware Hardware redundancy techniques are divided into logic-level techniques and architectural techniques. Logic-level techniques are typically implemented in combinational modules using a self-checking circuit such as output parity generation to check the correctness of the results of the circuit. Architectural techniques include duplicated modules or independent hardware to replicate and verify pipeline execution. One of the architectural techniques is Duplication With Compare (DWC). Duplication With Compare (DWC) is one of the simplest error detection techniques for FPGAs. DWC is a hardware redundancy technique that uses two identical replicas of a circuit and then compares the outputs of the circuits to determine if an error has existed. The circuit of the comparator detects differences in the results of the two circuits and notifies the system with an error flag [Johnson 2008]. Using DWC has several advantages including:

- Easy to implement in any circuit.
- Detects errors directly.
- Allows the system to rapidly respond to circuit problems.
- Requires limited external hardware support.

This technique can be used to detect both transient faults (SEU) as in [Johnson 2008] and can be used to detect permanent faults such as in [Psarakis 2012]. However, in [Psarakis 2012], both duplicated modules are partially reconfigurable.

The error detection scheme implemented in this technique uses almost the same technique as DWC; however, the duplicated module will have an inverted output. The outputs of the main module and its inverted version (F & F') are then linked to an internal XOR gate to detect errors. If the output of the XOR gate is '0', an error is detected. A 3-input majority circuit is implemented in a Xilinx Virtex4 FPGA. Figure 19 shows a block diagram for the error detection technique used in detecting faults in the 3-input majority circuit.

Figure 37 shows a block diagram for the error detection technique used in detecting faults in the 3-input majority circuit.

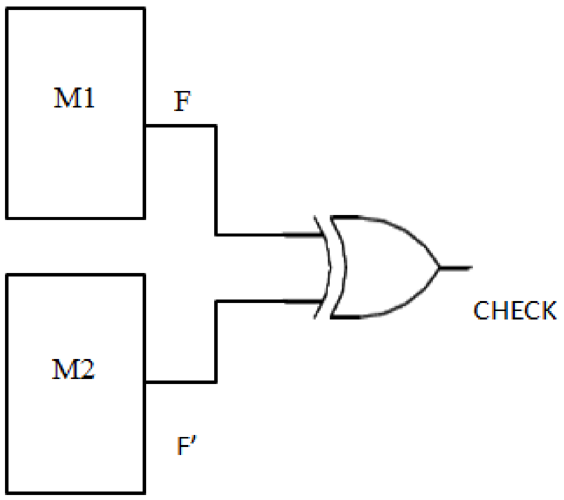
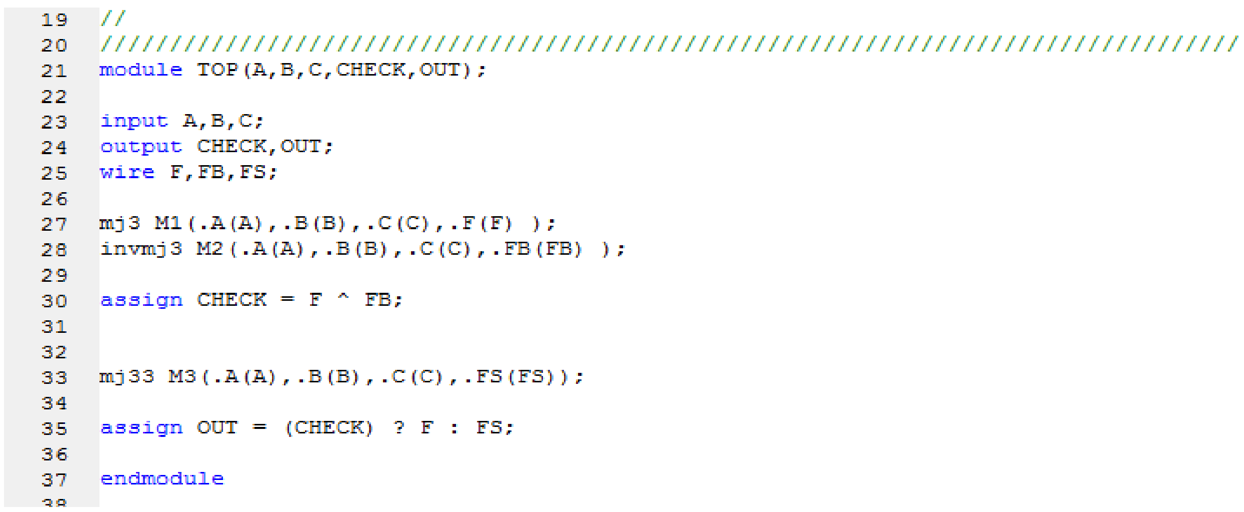


Figure 37: Proposed fault detection technique

The XOR gate can be implemented in the Xilinx ISE by using direct Verilog operator as in Figure 38 showing the Verilog code used to describe the fault detection technique in one of the case studies applied in this work.



```
//
/////////////////////////////////////////////////////////////////////////////////////
module TOP(A,B,C,CHECK,OUT);

input A,B,C;
output CHECK,OUT;
wire F,FB,FS;

mj3 M1(.A(A),.B(B),.C(C),.F(F) );
invmj3 M2(.A(A),.B(B),.C(C),.FB(FB) );

assign CHECK = F ^ FB;


mj33 M3(.A(A),.B(B),.C(C),.FS(FS));

assign OUT = (CHECK) ? F : FS;

endmodule
```

Figure 38: Verilog code for implementing XOR gate using direct Verilog operator.

The post place & route layout of the Xilinx Virtex-4 FPGA (XC4VFX12) slice generated from the Xilinx ISE for the above fault detection technique is shown in Figure 39.

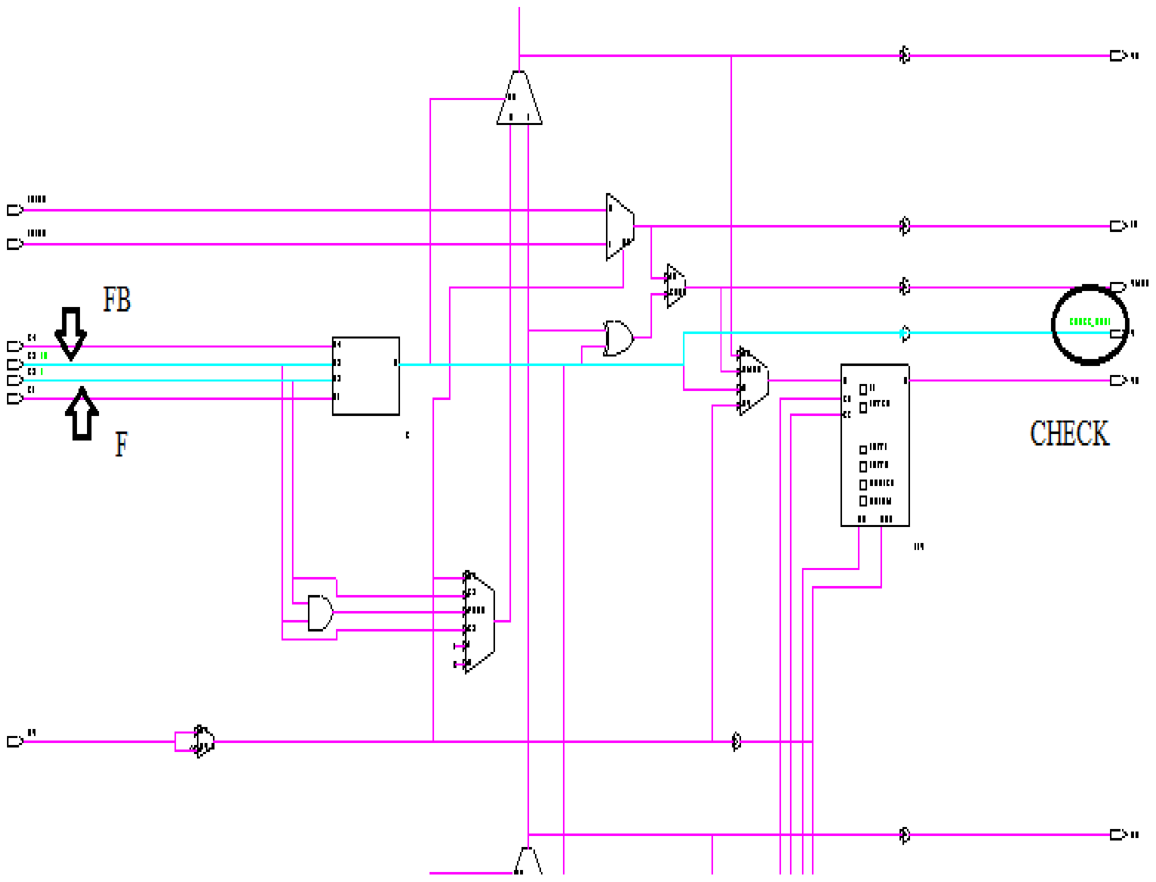
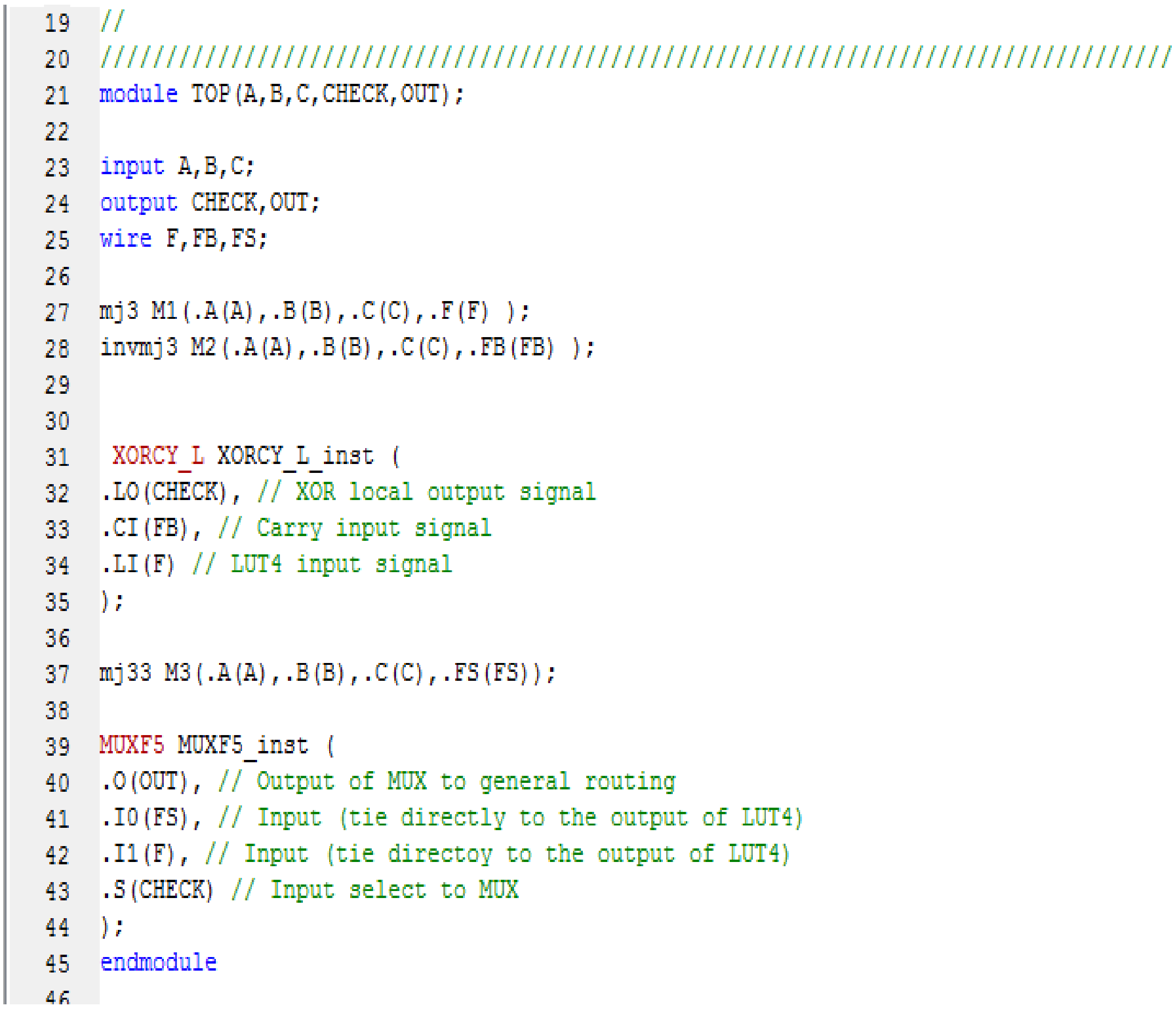


Figure 39: Post place &route layout of Xilinx Virtex-4 FPGA slice generated from Xilinx ISE for implementing XOR gate using direct Verilog operator.

It is observed that the outputs of the main module F and its inverted version FB are connected to the inputs of the 4-input LUT which is used to implement the XOR gate.

The fault detection XOR function can be implemented by accessing the built-in XOR gate in the slice. The slice primitives can be accessed and connected to certain signals in the routed design through inserting a piece of code defined in the Virtex-4 Libraries Guide for HDL Designs [Virtex-4 Libraries Guide for HDL Designs 2010].

The Verilog code describing the fault detection technique accessing the built-in XOR gate in the FPGA slice is shown in Figure 40.



```
//
//////////////////////////////////////////////////////////////////////////////////
module TOP(A,B,C,CHECK,OUT);

input A,B,C;
output CHECK,OUT;
wire F,FB,FS;

mj3 M1(.A(A),.B(B),.C(C),.F(F) );
invmj3 M2(.A(A),.B(B),.C(C),.FB(FB) );


 XORCY_L XORCY_L_inst (
.LO(CHECK), // XOR local output signal
.CI(FB), // Carry input signal
.LI(F) // LUT4 input signal
);

mj33 M3(.A(A),.B(B),.C(C),.FS(FS));

MUXF5 MUXF5_inst (
.O(OUT), // Output of MUX to general routing
.I0(FS), // Input (tie directly to the output of LUT4)
.I1(F), // Input (tie directoy to the output of LUT4)
.S(CHECK) // Input select to MUX
);
endmodule
```

Figure 40: Verilog code for accessing built-in XOR gate in the Xilinx Virtex-4 FPGA slice.

The post place & route layout of the Virtex-4 FPGA slice (XC4VFX12) generated from the Xilinx ISE for the above fault detection technique is shown in Figure 41 where it is clear that routed signals are passing through the internal built in XOR not through the LUT only as in the previous layout in Figure 39.

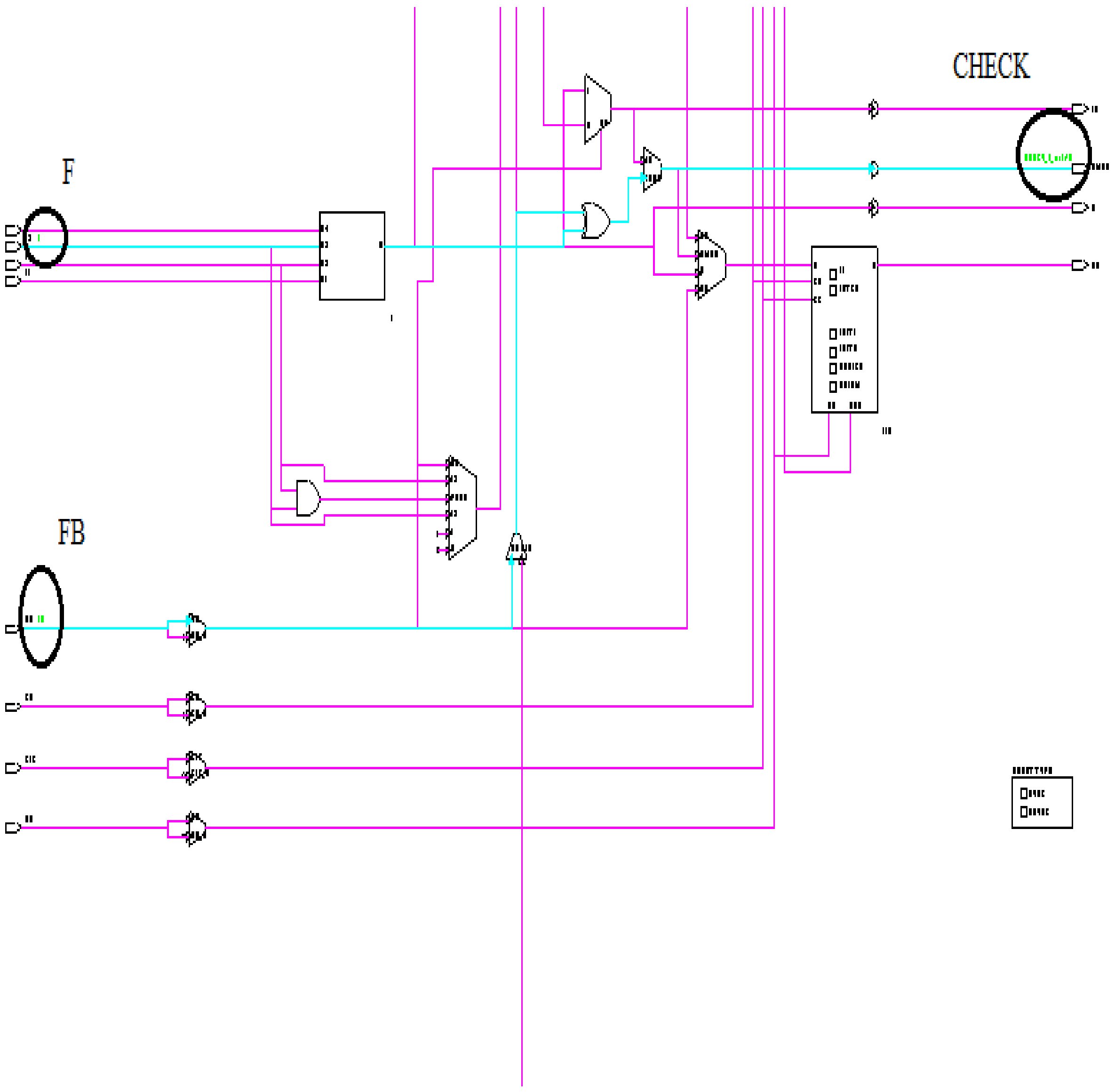


Figure 41: Post place &route layout of Xilinx Virtex-4 FPGA slice generated from Xilinx ISE for implementing fault detection technique through accessing built-in XOR gate in the slice.

It is observed in Figure 41 that the output F of the main module is passing through 4-input LUT to the first input of the built-in XOR gate and the output FB of the inverted version of the main module is connected to the second input of the built-in XOR gate which outputs a signal CHECK for detecting faults.

III.3. HIGH IMPEDANCE PROBLEM

Since A floating input in a CMOS gate may result in an undetermined logic output [Wakerly 2006] [Saba2010] [Stonham1996]. An Open fault results in a floating voltage which is represented in FPGA technologies by ‘Z’. According to [IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164)], an XOR gate will produce an output “X” if either of its inputs are “Z”. Therefore, after the occurrence of an open fault in the interconnections between F or FB and the inputs of the XOR gate, the fault detection signal “CHECK” output turn out to be unknown ‘X’; so the value of the detection signal cannot be defined. Figure 42 shows the high impedance problem.

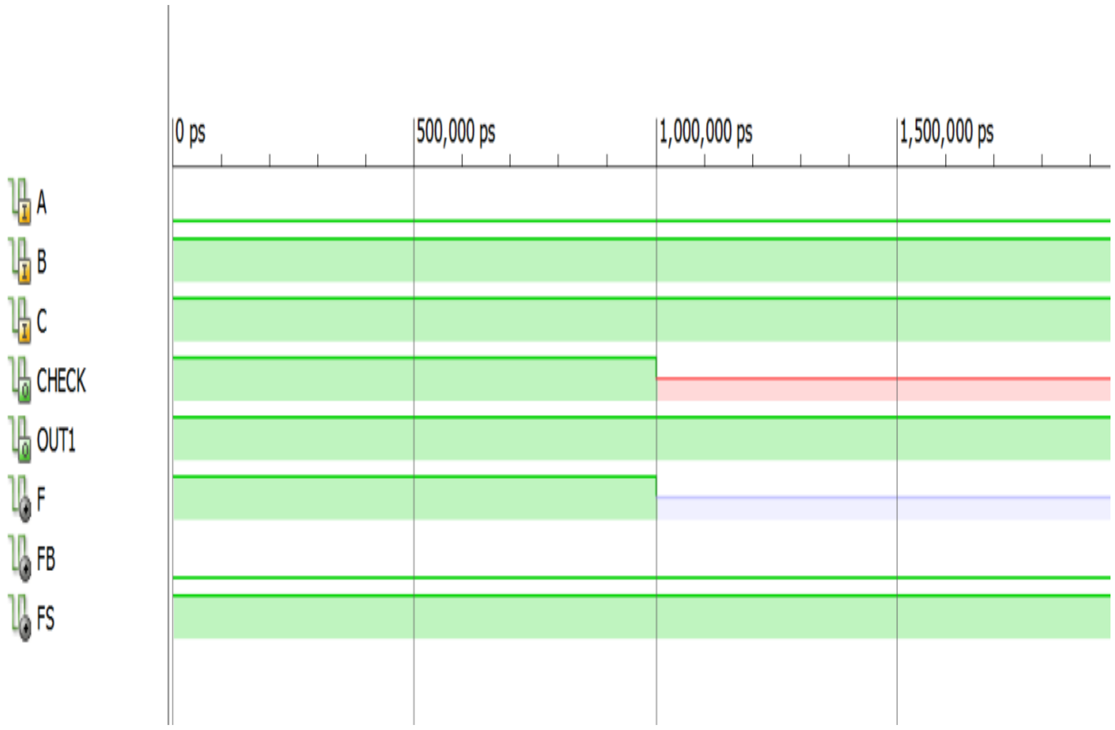


Figure 42: High impedance problem.

To solve this problem, a pull down resistance is added to alter the value of “CHECK” to ‘0’. As a result, a strong zero will be inserted in Xilinx ISE at the faulty input. This will enable Xilinx ISE to correctly simulate the effect of an open interconnect in the FPGA. Figure 43 shows the simulations after adding pull down resistance.

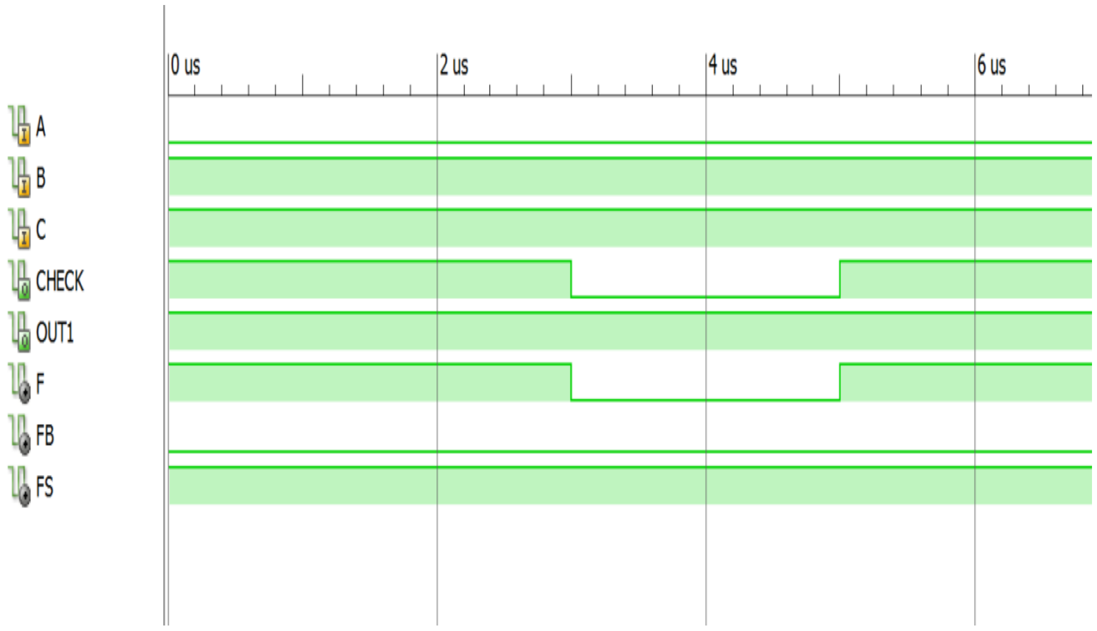


Figure 43: the simulations after adding pull down resistance.

Next is a justification for the use of pulldown resistances. Consider the path between the output F and the XOR gate input it is connected to. This path consists of programmable switch matrices and wires, a pass through LUT and a connection to the XOR input [Xilinx 2010]. If the open fault occurs before the LUT, it is expected that this block will produce a valid logic level but not necessarily the correct value. This value will be XORed with the correct bitstream from the other module FB and the error will be detected. Since the XOR function is the crucial to the correct functionality of the Duplication With Compare scheme, it was preferred to use the actual XOR that already exists in the FPGA. Using an LUT implementing the XOR function resistances at its inputs is not as robust as our scheme.
If the open fault occurs after the LUT, one of the XOR inputs will be floating. While this floating input might be expected to be interpreted as a logic ‘0’, it is indicated in the literature that this value is undefined. To study the effect of this floating input, a SPICE transistor level analysis is conducted next.

In [A'ain 1995], the fault model for an open gate consists of a DC source connected to the floating input to represent the voltage on the gate capacitance produced by circuit noise. This model will be used next. Figure 44 shows the simulation setup.

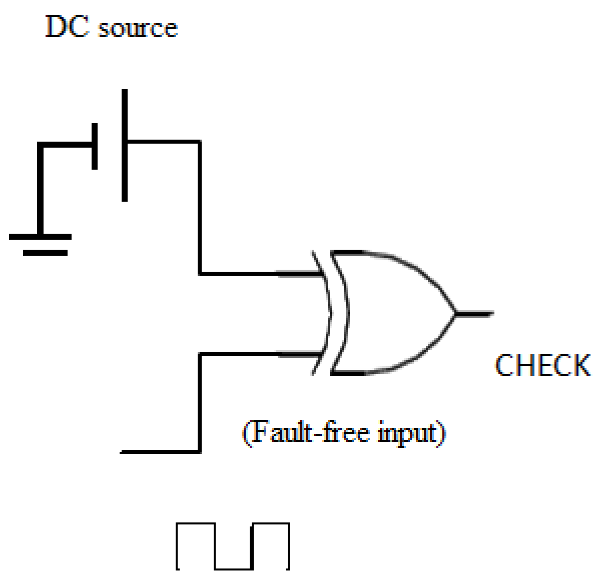


Figure 44: Simulation setup

The value of this source is unknown because it represents the surrounding voltage of the floating gate. This is also mentioned in [Johnson 1994]. It will be shown next that there is a possibility for this voltage to cause the XOR output NOT to detect the open fault. A 1MHz square wave is applied at the fault-free input of the XOR gate; this represents the bitstream produced by the FB. Figure 45 shows the SPICE output.

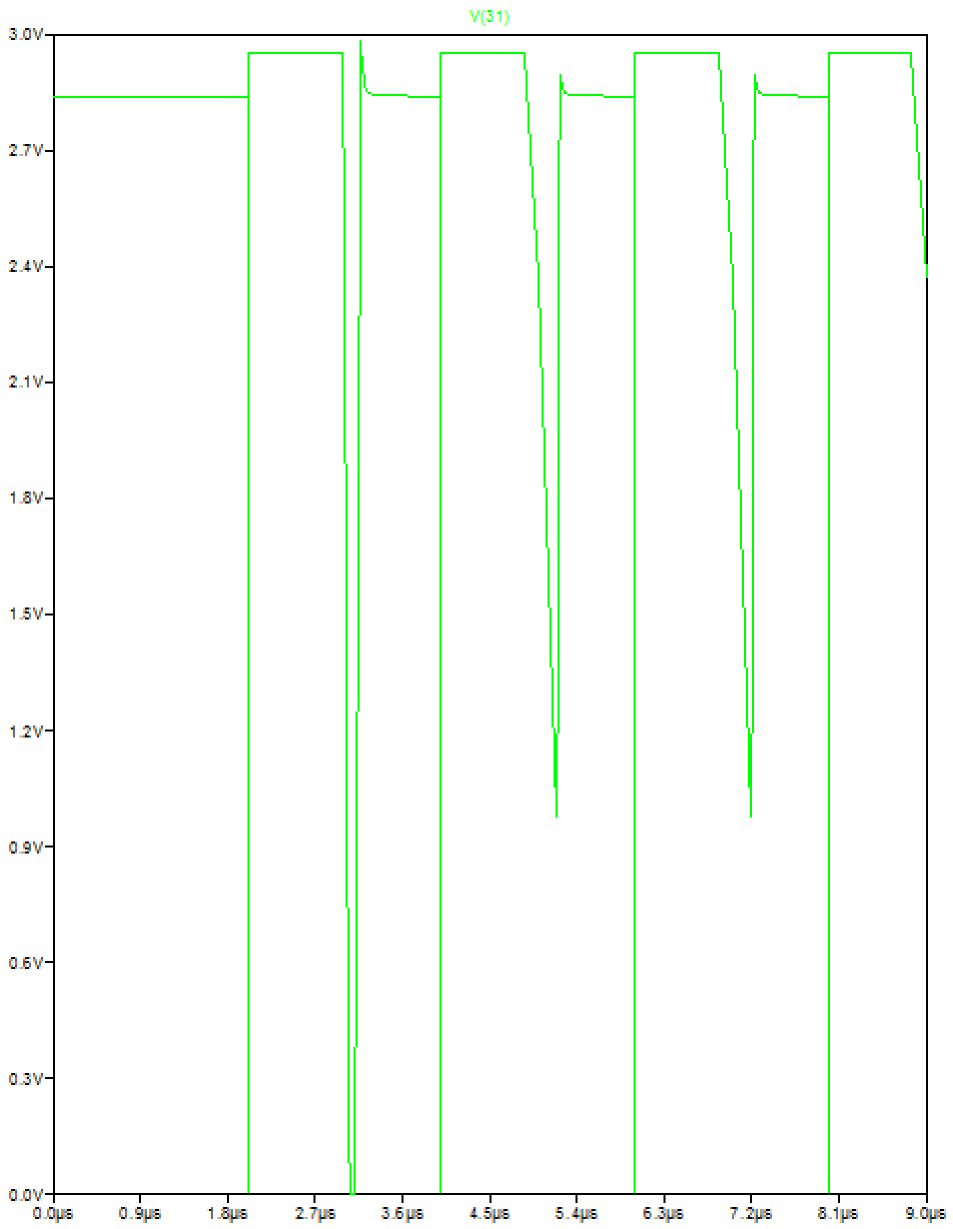


Figure 45: DC source connected to the floating input to represent the voltage on the gate capacitance produced by circuit noise.

This output is Dc and is affected by the surrounding circuitry. This cannot be considered as a valid error detection signal. Now, replace the supply in Figure 44 by a capacitance charged with the same voltage as the source in the previous experiment and connect a resistor in parallel with this capacitor. This simulates the “floating” input with a pulldown resistance. As shown in Figure 46, the output of the XOR gate becomes equal to the FB signal indicating that the XOR gate considers the other input to be equal to a logic ‘0’ and the open fault is detected (the figure shows two superimposed curves).

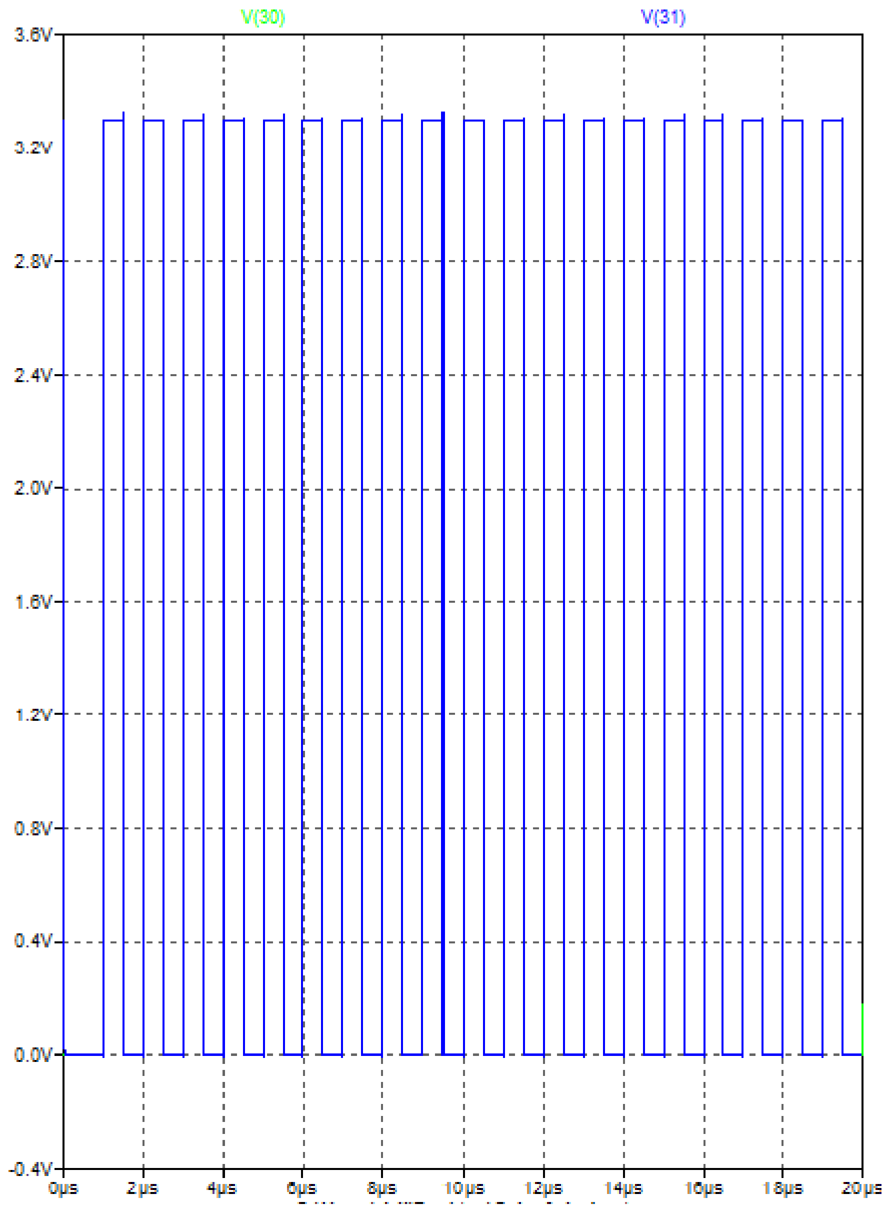


Figure 46: The effect of adding a pull down resistance on one of the inputs of the XOR gate.

In conclusion, SPICE simulations show that a floating XOR gate input may cause the XOR gate output NOT to detect the fault. For mission-critical applications, it is important to have a very robust error detection mechanism. The addition of the pulldown resistance will add to the robustness of the fault detection technique considered in this thesis. It is important to note that a floating gate has also been studied in [Arumi 2008]. In this paper, it is found that the voltage applied to the open gate needs time (sometimes in the order of seconds) to dissipate as shown in the following figure.

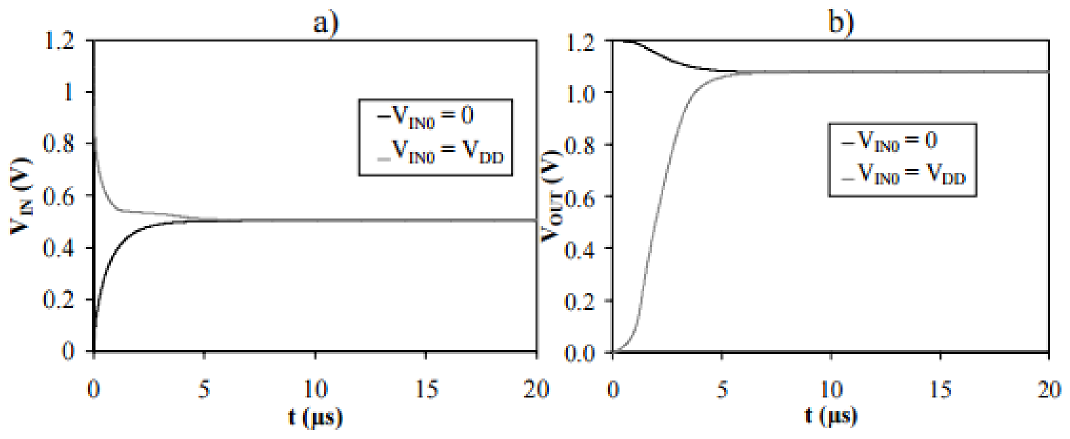


Figure 47: Transient response of the inverter for the 90nm PTM a) Inverter input b) Inverter output [Arumi 2008].

Simulations as well as actual experiments indicate that, in the steady state and due to leakage, the voltage is eventually a logic '0'. However, there is no mention of other charges accumulating on the floating gate during operation. Given that our technique targets mission-critical systems, it is imperative for the XOR gate to detect a fault as soon as it occurs; hence, the need for pulldown resistances.

Of course, these inserted resistances have the disadvantages of increasing area and power. The increase in area is minimized by using NMOS transistors for pull-down instead of PMOS transistors for Pull-up. An NMOS transistor requires less area than a PMOS transistor [Sedra 2004]. The size of the PMOS transistor is about 2.5 times larger than the size of an NMOS transistor [Nam 2005].

A transistor is connected to a gate input as shown in the Figure 48 [Bilotti 1966]. The source is connected to Ground and the gate is connected to 0.26V (Vth). By measuring the current in the transistor (by simulation), it is found that the equivalent resistance is equal to 2.3GΩ. Also, W=180nm and L=360nm. Even if the gate voltage is increased to 0.29V, the equivalent resistance will still be 1GΩ.

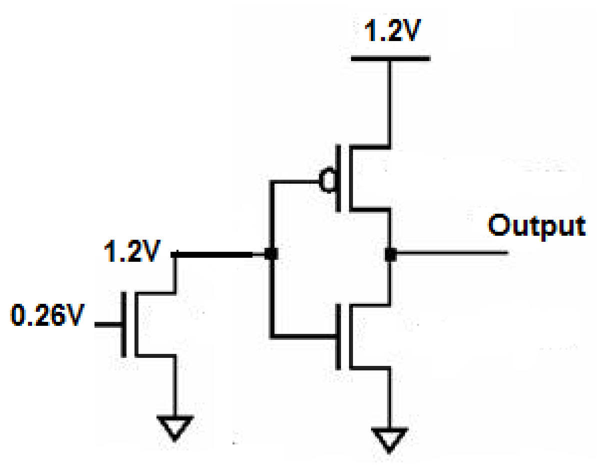


Figure 48: NMOS pulldown transistor connected to a gate input.

In general, the number of resistances inserted is up to the designer who has to make a compromise between increased lifetime vs higher area and higher power consumption.

Of course, the addition of pulldown resistances (implemented by NMOS transistors) has drawbacks in terms of additional area required and extra power consumption. This will be discussed next. Let there be *n* critical modules required to be fault-tolerant in a certain system. For simplicity, let each module has a single output but this argument can be easily extended to multi output modules. Assume that one of these modules consumes an area A in the FPGA. If this module is decided to be made fault-tolerant, it will require the following (at least):

An area for F = A
An area for FB = A
An area for FS (the spare module) = A
One pass through LUT = A_{LUT}
Two pulldown resistances at the inputs of the XOR gate used for detection = $2 \times A_R$

The extra hardware for the *n* critical modules with pulldown resistances will be;

$$\sum_{i=0}^{n}(2 \times A_{i)} + (A_{LUT}) + (2 \times A_R)$$

Note the factor ($2\times A_i$) in the above equation; this is because the original module (producing the output F) is not counted as additional area, only the areas for FB and FS. Given that the modules can be quite complex (hundreds of slices), the value of A_i is expected to be much higher than A_{LUT} or A_R. Consequently, the extra area required for the pulldown resistors compared to the extra area required for the rest of the fault-tolerance requirements is very small.

Furthermore, assume that every slice (worst case) in the FPGA has 2 pulldown resistances connected to its XOR gate. Let the slice consist of (at least): 2×LUT. Each LUT consists of [Mondal 2005]:

- 16 SRAM cells with 6 transistors per cell = 16×6=96 transistors
- 16-input pass transistor multiplexer = 30 transistors
- Set of buffers to amplify the select inputs and generate their complements = 24 transistors

Hence the total number of transistors per LUT is: 150 transitors. For 2 LUTs, the total number of transistors is 300 transistors (at least). Hence the overhead (in terms of transistor area) in a slice is 2/300=0.666%

Note that, in fault-tolerant designs [Siewiorek 1998, Shooman 2002], hybrid redundancy is a well-known and commonly used technique. It often consists of a Triple Modular Redundancy (TMR) core and several spares. Let the number of these spares be *m*. Hence, using the same terminology as above, the extra area required to make a module fault-tolerant using hybrid redundancy with a TMR core is: (2+m)A. In addition, there will be an area required for the error detection and error recovery hardware.

Regarding the extra power consumption of the pulldown resistances, SPICE simulations showed that the static power consumed by a XOR gate in the 90nm technology is equal to 0mw with both inputs high. With a pulldown transistor added to each input, the additional static power consumed by the two resistances is 1.296nw. It is important to note here that the static power consumed by one of the case studies, namely the 16-bit ALU (after applying the proposed fault-tolerant technique), was found to be 166.38mw using the ISE software package. This circuit requires 17 XOR gates and hence 34 pulldown transistors. The extra static power consumed by the transistors is 1.296nw×17=22.032nw.

The pull down resistances should be located in a certain region of the FPGA, called the fault detection region. Additionally, synthesis and mapping algorithms should be adapted to support the selection of the fault detection region. This technique can be modified to other FPGA technologies according to their architectures.

III.4. Fault Recovery Technique

The fault recovery technique consists of a 2-to-1 multiplexer and a copy of the main module which acts as a spare module (output FS). The outputs of the main module and the spare module are the inputs of the Multiplexer (MUX). The selection signal of the multiplexer is connected to the "CHECK" signal as shown in Figure 49.

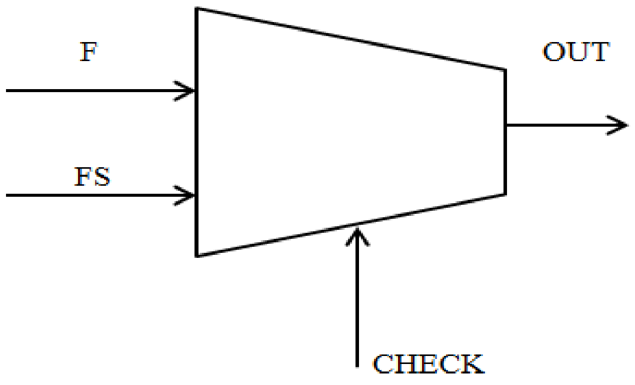


Figure 49: Proposed fault recovery technique.

Consequently, in the fault-free situation, the output of the main circuit (F) will be chosen. In case of an error in the interconnections of FPGA, the output of the spare module will be chosen assuming that the spare module is still operational.

The fault recovery MUX function can be implemented by accessing the built-in MUX by using the code of accessing MUXF5 in the slice which is defined in the Virtex-4 Libraries Guide for HDL Designs [Virtex-4 Libraries Guide for HDL Designs 2010]. The Verilog code describing the fault recovery technique accessing the built-in MUX in the FPGA slice is shown in Figure 40.

The post place & route layout of the Virtex-4 FPGA slice (XC4VFX12) generated from the Xilinx ISE for the above fault recovery technique is shown in Figure 50 where it is clear that the internal built in MUX is used in the fault recovery technique.

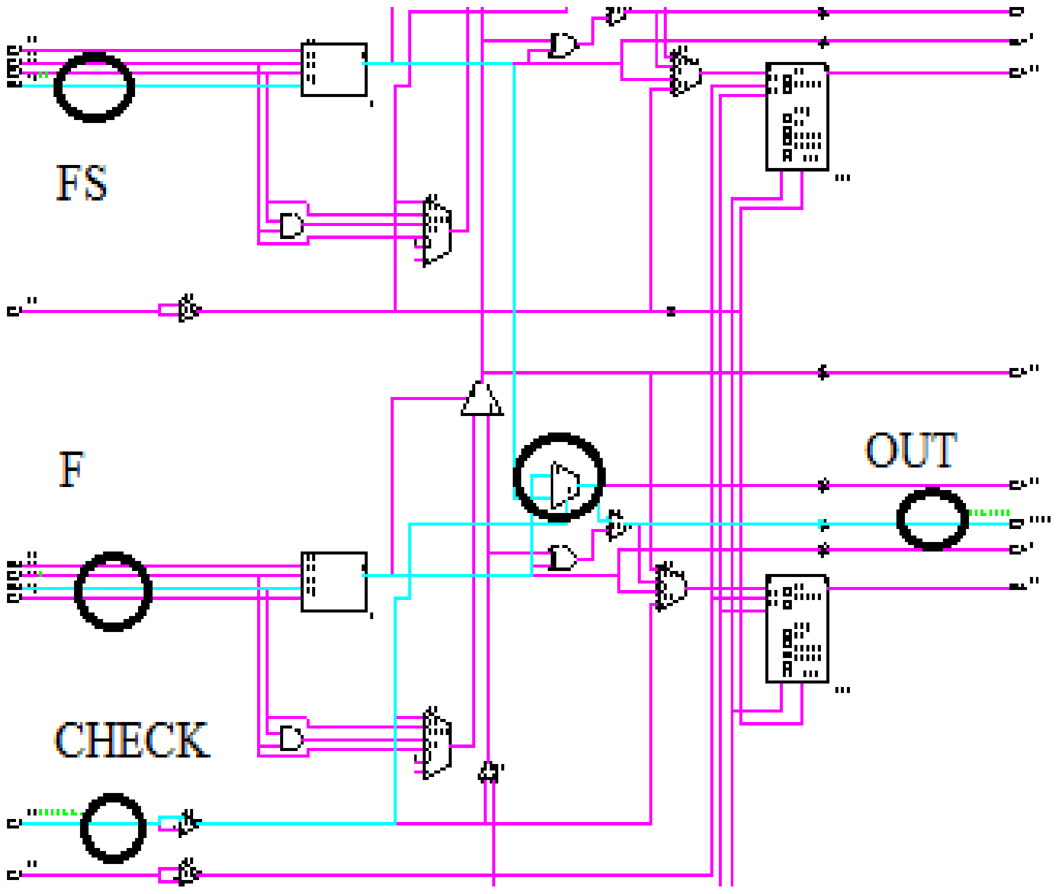


Figure 50: Post place &route layout of Xilinx Virtex-4 FPGA slice generated from Xilinx ISE for implementing fault recovery technique through accessing built-in MUX in the slice.

It is observed in Figure 50 that the output F of the main module is passing through 4-input LUT to the first input of the built-in MUX and the output FS of the spare module is connected to the second input of the built-in MUX. CHECK signal is connected to the select of the MUX. Accordingly, if any fault happened in the interconnections of F, FB or the inputs of the XOR gate, CHECK signal will stimulate the MUX to select the result comes from the FS. As a result, fault is recovered. However, this fault recovery technique is simple; it gives rapid efficient way of recovery and consumes lower area than other complicated recovery techniques.

A User Constraints File (UCF) file is an ASCII file which specifies constraints on the logical design [Constraints guide 2012]. These constraints affect how the logical design is implemented in the target device. Additionally, it can be used to override constraints identified during design entry. UCF files are input to Xilinx Native Generic Database (NGD). The constraints in the UCF files become part of the information in the NGD file produced by NGDBuild. For FPGA devices, some of these constraints are used when the design is mapped by MAP, and some of the constraints are written into the Physical Constraints File (PCF) produced by MAP. Examples of constraints specified in UCF are timing and placement constraints of designs. To confirm that a fault will not affect the spare module, User Constraint File (UCF) is used. Through the UCF file, the spare module can be mapped away from the main module and the fault detection hardware.

III.5. PULL DOWN RESISTANCE LOCATION

The location of the pull down resistance affects area, power and delay. There are different options for adding the pull down resistance. According to the architecture of the Xilinx Virtex4 FPGAs (XC4VFX12), the following three locations are suggested:

1. Switch matrix as shown in Figure 52.
2. Inputs of pass-through LUT as shown in Figure 51.
3. Inputs of the XOR gate as shown in Figure 51.

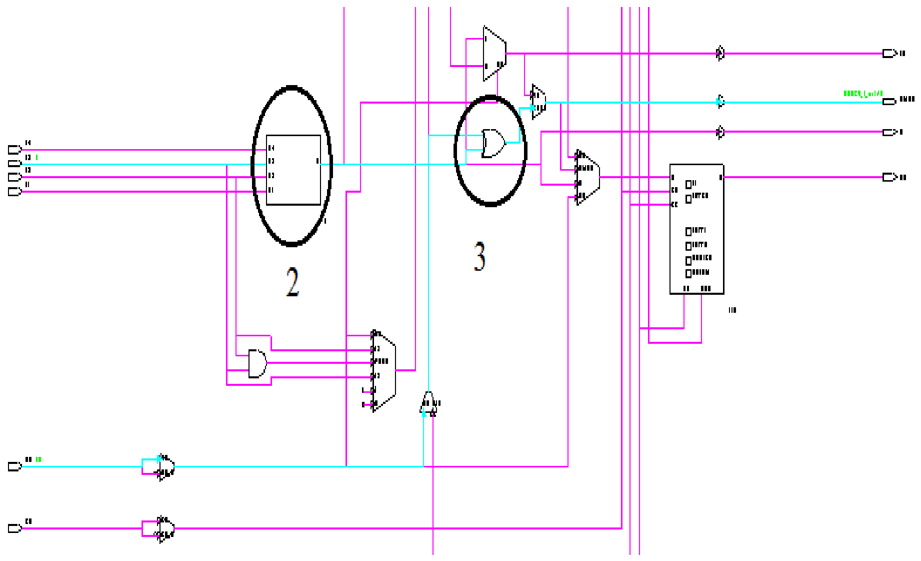


Figure 51: The proposed locations for placing the pull down resistance.

Assume a pull down resistance (Rb) is located in the 16 direct connections surrounding the switch matrix as shown in the block diagram in Figure 52.

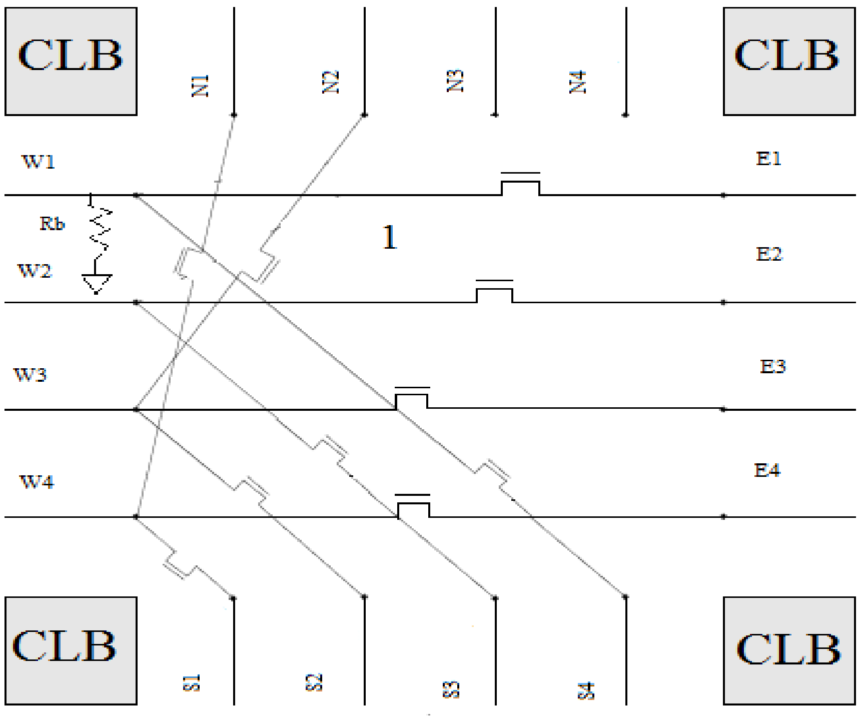


Figure 52: Block diagram represent the switch matrix of Xilinx FPGAs.

Consequently, 16 pull down resistances will be added to cover the entire switch matrix. This consumes a great area in the FPGA. On the other hand, adding the pull down resistances at the inputs of the 4-input LUT will need less area than adding pull down resistances in the switch matrix of the FPGA. Since the number of added resistances will be limited to the number of inputs of the LUTs which are 4 inputs in Xilinx Virtex-4 FPGAs.

In case of adding pull down resistances at the inputs of the XOR gate, only 2 resistances are needed. Accordingly, the third location consumes the least area and it is suggested to add the pull down resistances at the inputs of the XOR gate. Although, the location that adds the least area is chosen, adding the pull down resistance will increase the power consumption in the FPGA. Accordingly, Xilinx vendor has to make a tradeoff between increasing the FPGA reliability and having high power consumption.

III.6. CASE STUDY

III.6.1. 3-INPUT MAJORITY CIRCUIT

A majority circuit is a circuit with an odd number of inputs whose output is a 1 if and only if a majority of its inputs are 1 and output is a 0 if and only if a majority of its inputs are 0. The gate level block diagram of the 3-input majority circuit is shown in Figure 53.

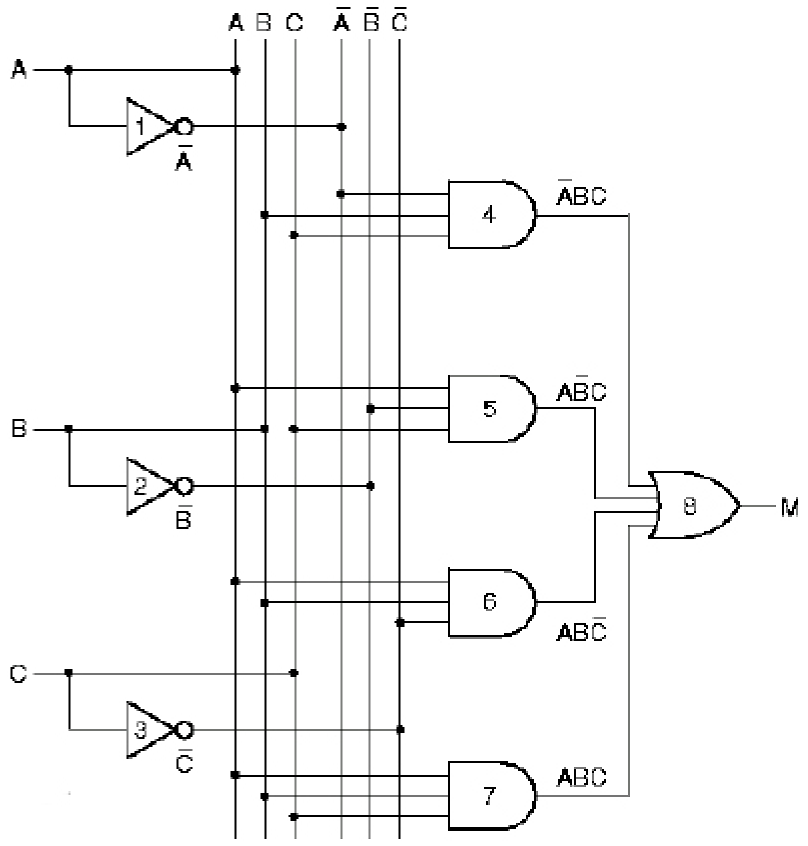


Figure 53: 3-input majority circuit.

The 3-input majority circuit stated above was simulated with the Xilinx ISE Simulator (ISIM). Faults were inserted by injecting a 'Z' value in the location of the open interconnects in the post-synthesis VHDL file of the circuit which is the NCD file. Native Circuit Description (NCD) file is the file that contains the information of the placing and routing of the design. Additionally, is used as input for bit-stream generation. The outcomes show that the proposed technique can detect and recover from open faults in interconnections as shown in Figure 43.

The post place & route layout of the Xilinx Virtex-4 FPGA (XC4VFX12) slice generated from the Xilinx ISE for the fault detection technique implementation in 3-input majority circuit is shown in Figure 41. Moreover, the post place & route layout of the fault recovery technique applied to the 3-input majority circuit is shown in Figure 50. The main module, its inverted version and the spare have the same way of implementation in the post place & route layout. Figure 54 shows how F is implemented using 4-input LUT in the slice of Xilinx Virtex-4 FPGA (XC4VFX12).

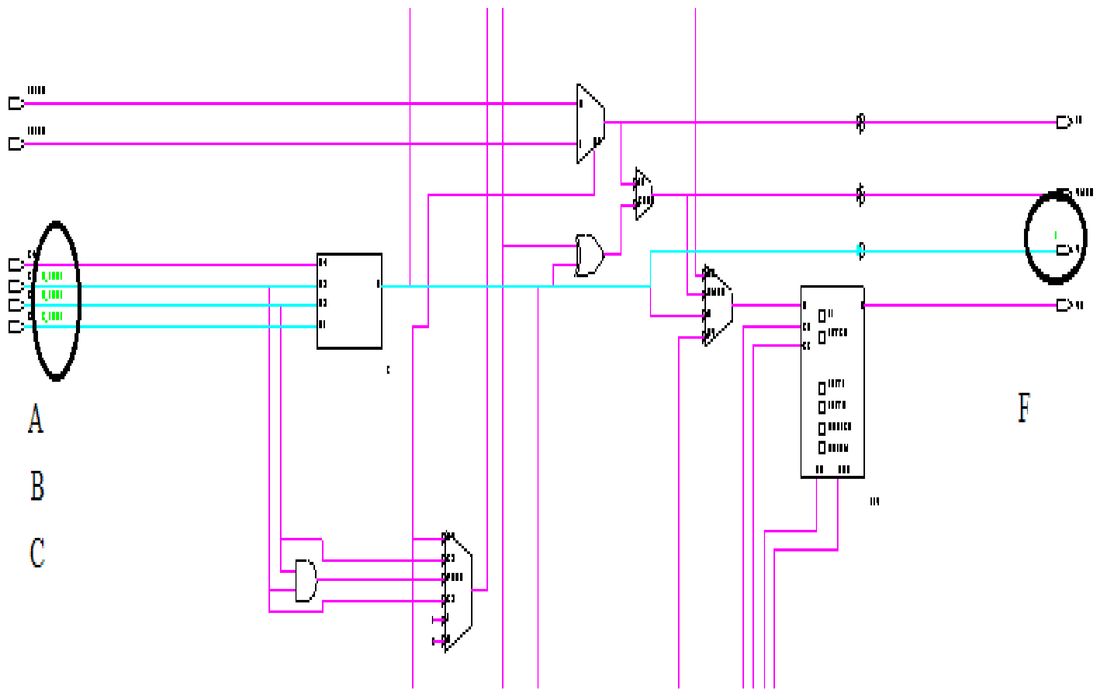


Figure 54: F implementation using 4-input LUT in the slice of Xilinx Virtex-4 FPGA.

III.6.2. 16-BIT ALU

To ensure that the proposed fault detection and reconfiguration techniques can be applied to larger circuits, a 16-bit ALU circuit is implemented. The circuit has 17 outputs. Figure 55 shows the inputs and outputs of the 16-bit ALU.

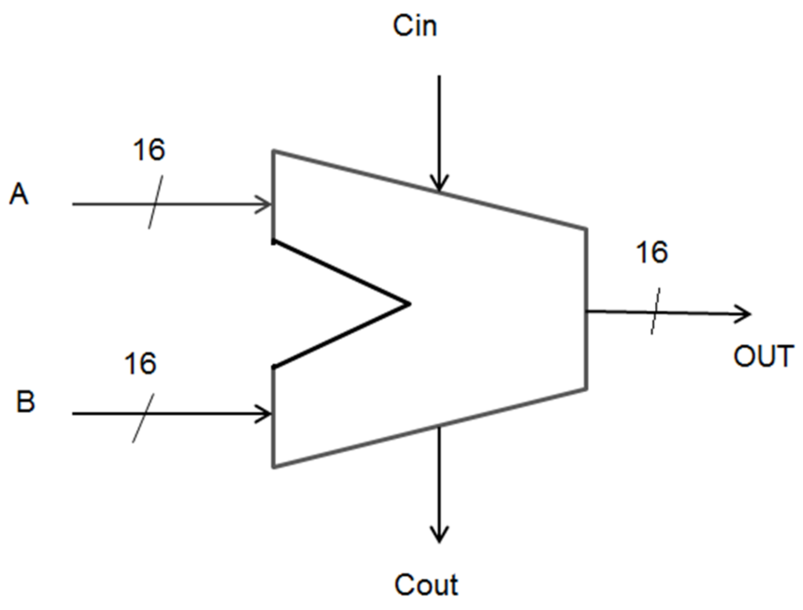


Figure 55: 16-bit ALU

The same fault detection technique is used but it is applied at the level of bits. Every pair of corresponding output bits as well as the COUT bit from the two copies of the 16-bit ALU circuit is compared. Then, all the fault detection signals produced are connected to an AND gate as shown in Figure 56.

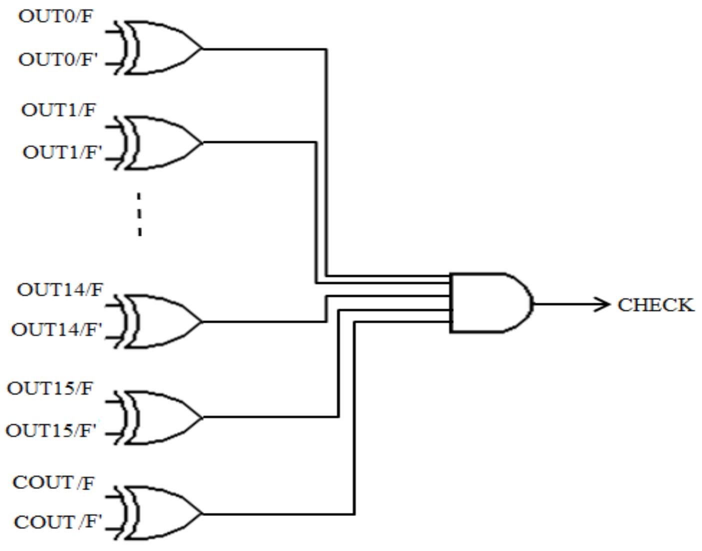
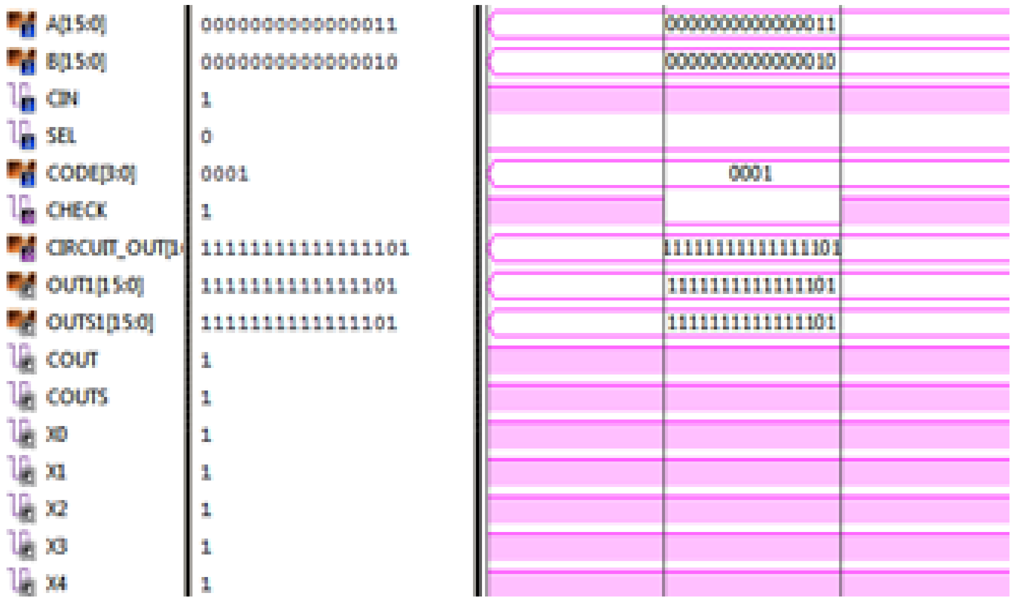


Figure 56: Fault detection technique for 16 bit ALU

As a result, in the event of a fault in any of the output bits, it will be detected. Also, the fault recovery technique described above supports multi outputs. Each output of the16-bit ALU circuit (main module) will be connected to MUX with its corresponding output in the spare module and the check signal for this output will be connected to the select of the MUX. Accordingly, if a fault exists, the check signal will stimulates the MUX to select the output of the spare module.

The 16-bit ALU circuit was simulated with the Xilinx ISE Simulator (ISIM) and the results show that the proposed approach can detect and recover from open faults in the interconnections. Figure 57 displays the wave forms of the fault tolerant 16 bit ALU circuit.



```
A[15:0]          0000000000000011
B[15:0]          0000000000000010
CIN              1
SEL              0
CODE[3:0]        0001
CHECK            1
CIRCUIT_OUT[1    11111111111111101
OUT1[15:0]       1111111111111101
OUTS1[15:0]      1111111111111101
COUT             1
COUTS            1
X0               1
X1               1
X2               1
X3               1
X4               1
```

Figure 57: Fault tolerant 16 bit ALU circuit wave forms

The post place & route layout of the Xilinx Virtex-4 FPGA (XC4VFX12) slice generated from the Xilinx ISE for the fault detection technique implementation in 16-bit ALU is shown in Figure 58.

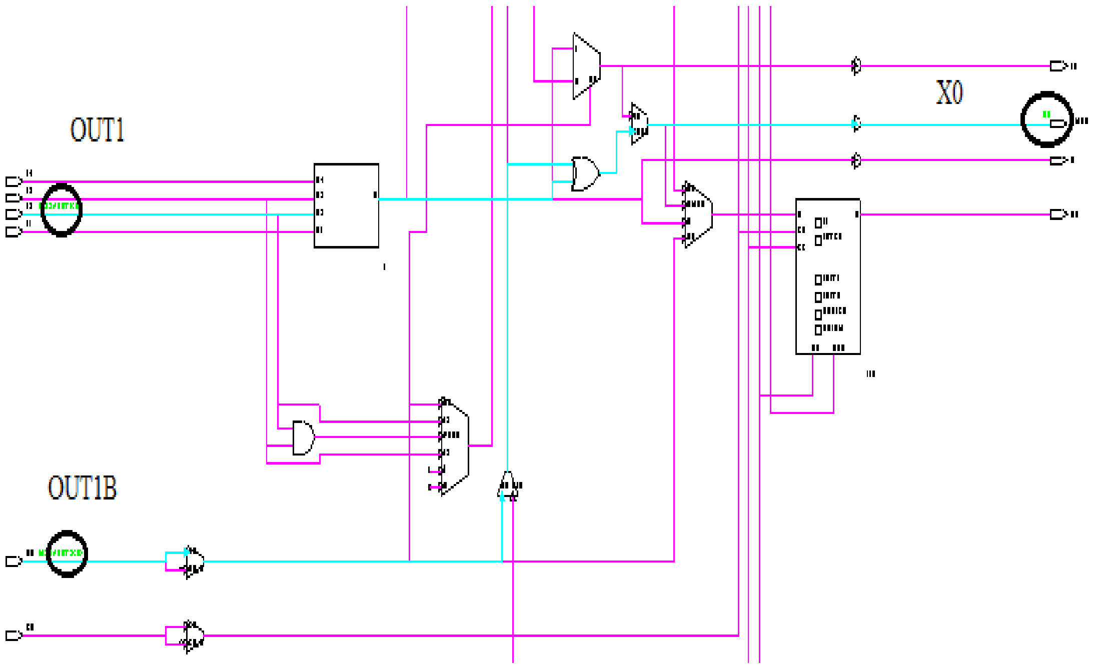


Figure 58: Post place &route layout of Xilinx Virtex-4 FPGA slice generated from Xilinx ISE for implementing fault detection technique of 16-bit ALU.

It is observed in Figure 58 that the output OUT1 of the main module is passing through 4-input LUT to the first input of the built-in XOR gate and the output OUT1B of the inverted version of the main module is connected to the second input of the built-in XOR gate which outputs a signal X0 for the detection of faults.

The post place & route layout of the Xilinx Virtex-4 FPGA (XC4VFX12) slice generated from the Xilinx ISE for the fault detection technique implementation in 16-bit ALU is shown in Figure 59.

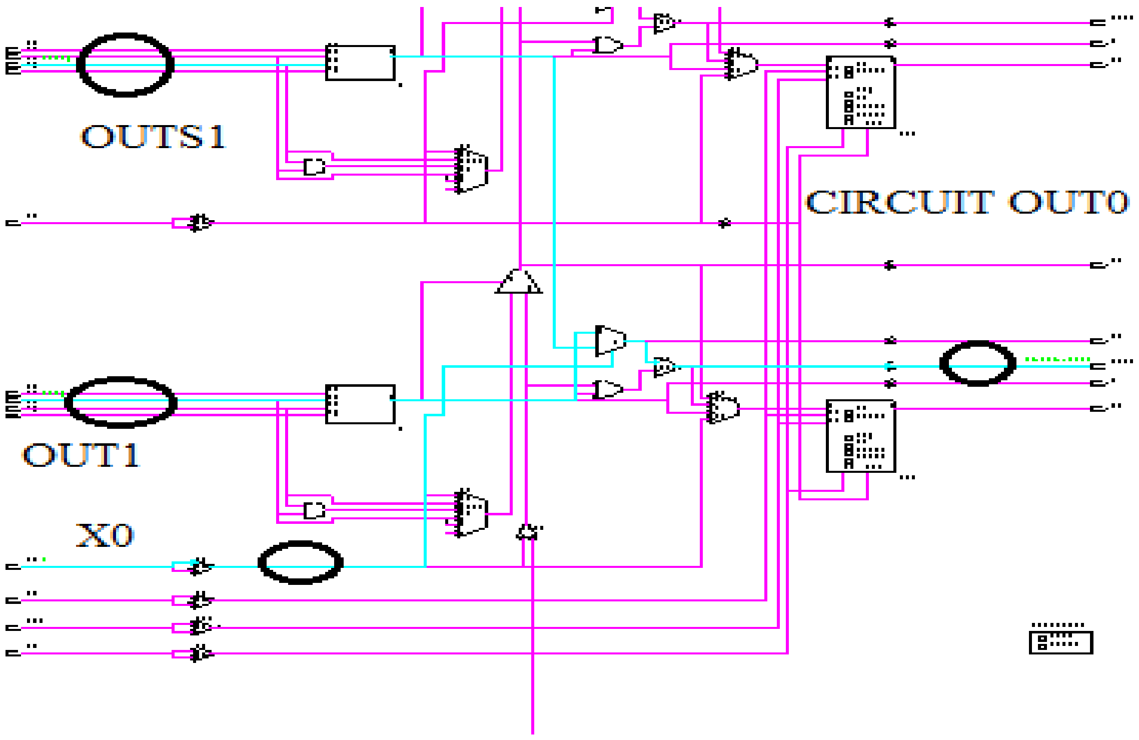


Figure 59: Post place &route layout of Xilinx Virtex-4 FPGA slice generated from Xilinx ISE for implementing fault recovery technique of 16-bit ALU.

It is observed in Figure 50 that the output OUT1 of the main ALU module is passing through 4-input LUT to the first input of the built-in MUX and the output OUTS1 of the spare module is connected to the second input of the built-in MUX. X0 signal is the check signal of output OUT1 and it is connected to the select of the MUX. As a result, if a fault exists, X0 signal will stimulate the MUX to select the result comes from the OUTS1. Therefore, fault is recovered.

III.7. PROS AND CONS OF THE FIRST PROPOSED TECHNIQUE

The pros of the first proposed fault tolerant technique are having simple detection and recovery hardware and consuming low area overhead. However, it can only recover from one fault. Also, it is assumed that the spare will be operational when invoked after the failure of one of the two main modules.

IV. FAULT TOLERANT TECHNIQUE FOR FPGAS WITH PARTIAL RECONFIGURATION

IV.1. METHODOLOGY

In chapter 3, the motivation was detecting open faults in FPGA programmable interconnections using a low-cost fault detection technique. The fault detection technique used is the Duplication With Compare (DWC) technique. Moreover, a fault recovery technique was proposed. The proposed scheme needed the addition of a pulldown resistance to solve the problem of unidentified value of the detection signal that resulted from open fault existing in the FPGA interconnects. Different locations for adding the pull down resistance in the FPGA were proposed and the optimum place was chosen in order to add the minimum additional area to the FPGA.

In this chapter, the same fault model and fault detection scheme are used; however, a new fault recovery technique using the Partial Reconfiguration (PR) property in the FPGA, is proposed. Partial Reconfiguration is a feature of modern FPGAs that allows a subset of the logic fabric of an FPGA to dynamically reconfigure while the remaining logic continues to operate unperturbed [Xilinx 2013]. Partial Reconfiguration can adapt hardware algorithms, share hardware between various applications, increase resource utilization, provide continuous hardware servicing and upgrade hardware remotely. Moreover, it can be used in FPGA fault-tolerant techniques implementation.

There are several techniques used Partial Reconfiguration (PR) feature in designing FPGA fault tolerant techniques such as [Psarakis 2012],[Di Carlo 2010], [Kananaru 2009], [Bolchini 2013], [Zhang 2013] and [Di Carlo 2014]. [Psarakis 2012] used PR in generating the precompiled partial configurations for the design which are used in fault recovery. Also, [Psarakis 2012] implemented the partially reconfigurable duplicated modules using PR. Another technique using Partial Reconfiguration property is [Di Carlo 2010] in which it is used to implement the reconfigured area where the faulty module will be reconfigured.

[Kananaru 2009] benefits from PR to implement the reconfigurable tiles used in the tile-based technique. However, [Bolchini 2013] used PR in generating the precompiled full bit-streams used for reconfiguring the whole FPGA when a fault occurs. The fault recovery technique proposed in [Zhang 2013] uses PR in implementing the reconfigurable cells used in the shifting technique applied once a fault occurs. Finally, Partial Reconfiguration is used by [Di Carlo 2014] in dividing an FPGA in to partially reconfigurable tiles: logic tiles, interconnection tiles and recovery tiles. As a result, when any logic tile becomes faulty, it can be firstly partially reconfigured with its Partial BIT file to recover from any transient fault. If it is not recovered, it will be replaced by one of the recovery tiles and the interconnection tiles updated.

The new fault correction technique presented here is considered as TL1 system. It relies on specifying a Partially Reconfigurable block in the FPGA which is used in system recovery after the first defective module is detected in the running application. When a fault is detected in any one of the interconnections in the circuit, a check bit specifying the identity of the faulty module is sent from the FPGA to the FPGA configuration circuit. Accordingly, the Partial BIT file corresponding to the faulty module is downloaded into the Partially Reconfigurable block which is used for fault recovery using SelectMAP interface.

The Partial BIT file comprises all the configuration commands and data necessary for Partial Reconfiguration (PR) [Xilinx 2013]. The task of loading a partial bitstream into an FPGA does not require knowledge of the physical location of the Partially Reconfigurable Block (P_b) because the configuration frame which addresses the information is included in the partial bitstream. Accordingly, the partial BIT file cannot be sent to the wrong part of the FPGA device. A Partial Reconfiguration controller gets the partial BIT file of the faulty module from an external nonvolatile memory on the FPGA configuration circuit. Then, it delivers a partial BIT file to the configuration port. A user-designed internal PR controller loads the partial BIT file through the Internal Configuration Access Port (ICAP) interface which is essentially an internal version of the SelectMAP interface [Xilinx 2013].

Initially - in the fault free case - the Partially Reconfigurable block is programmed by a blanking bitstream. This means that a blank bitstream file will be written to the region to reduce the static power of the region when it is not needed [Liu 2009]. This is the situation when there is no fault.

Assume that an application consist of H modules. Each module M_N (N=0, 1, …., H-1) contains a main module with its fault detection circuitry as in Chapter 3. The fault detection circuitry consists of the inverted version of the main module and the comparator "XOR gate". The XOR gate detects any fault which may occur in the main module or its inverted copy. Accordingly, the module outputs contain a check bit C_N which defines the status of the module (faulty or not). If the module is fault-free, C_N equals '1'. On the other hand, if the module is faulty, C_N equals '0'. The suggested fault recovery technique is based on a Partially Reconfigurable block (P_b) where the number of inputs and outputs is determined according to the maximum number of inputs and outputs in H modules.

The inputs of the reconfigurable block are controlled by module CTRL. This module chooses the reconfigurable block inputs from the H module inputs. The inputs of the CTRL module are connected to all the inputs of the H modules and all the check bits C_N. The outputs of CTRL are connected to the inputs of the reconfigurable block. When an error is detected using the check bit C_N, the CTRL module sends the inputs of the defective module to the inputs of the reconfigurable block. Simultaneously, check bit C_N acts also as stimuli to the FPGA configuration circuit to download the Partial bit file of the faulty module in the Partially Reconfigurable block. Accordingly, the Partially Reconfigurable block acts as a spare module for the first faulty module.

To control which output is connected to the FPGA output pins, a 2-to-1 multiplexer is implemented for each output. The output of module M_N and the output of the Partially Reconfigurable block are connected to the inputs of the multiplexer and the selection line is connected to the check bit C_N. In the fault-free situation, the output of the M_N module will be selected. But, if an error is detected, the output of the reconfigurable block is selected as shown in Figure 60.

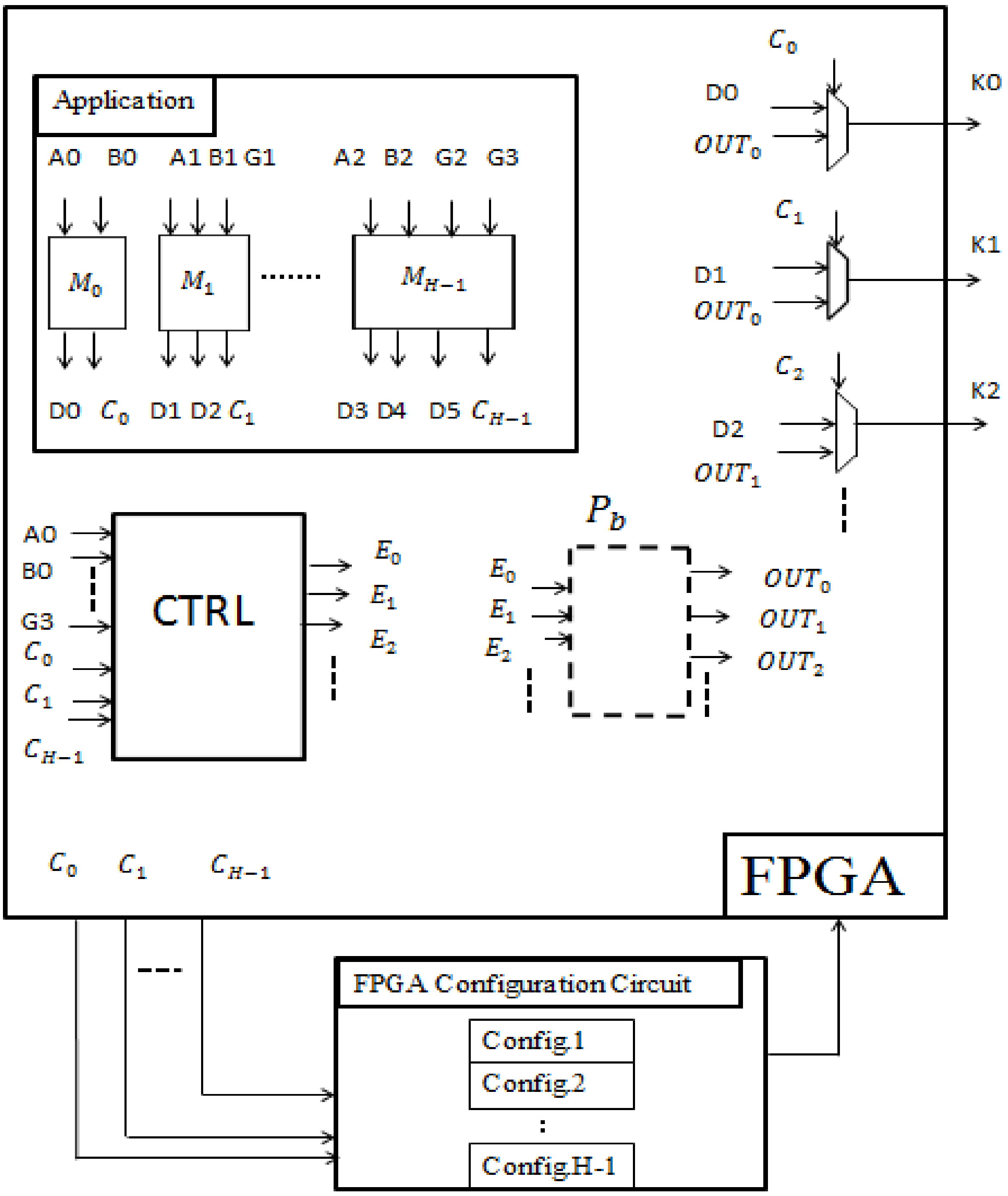


Figure 60: Proposed approach.

To certify that the proposed technique can be implemented on applications that consist of cascaded modules in which the outputs of the modules may feed as an input for another module in the system, a slight modification is implemented. The 2-to-1 multiplexers are placed inside the application, not just before the output pins as shown in Figure 60. The multiplexers are placed in each output of the modules in the application. The output of module M_N and the output of the Partially Reconfigurable block are linked to the inputs of the multiplexer and the selection line is connected to the check bit C_N. In the fault-free situation, the output of the M_N module will be selected. But, if an error is detected and check bit C_N becomes '0', the output of the Partially Reconfigurable block is selected as shown in Figure 68.

IV.2. CASE STUDIES

The new proposed recovery scheme is implemented using the Xilinx Virtex-4 SRAM-based FPGA. It is verified using four case studies. In the beginning of the experimental work, the technique is applied to two simple circuits to test the functionality of the proposed technique which are simple logic gates and a simple 2 bit ALU. Then, the technique is implemented in more complicated case studies which are an UART and a Pacemaker.

Each case study is simulated with the Xilinx ISE Simulator ISIM and faults are injected by inserting a value ‘Z’ in the location of the open interconnects in the post-synthesis VHDL file of the circuit. In addition, partial reconfiguration bitstreams are created using Xilinx PlanAhead tool. Xilinx PlanAhead tool is employed between synthesis and place-and-route in order to enable designers to rapidly analyze, modify and achieve superior performance on each of the individual design blocks. Accordingly, it is used in Partial Reconfiguration through which the Partially Reconfigurable block is defined. Accordingly, it can be partially reconfigured without reconfiguring the whole design. In this section, each case study will be discussed and the results will be presented.

IV.2.1. SIMPLE LOGIC GATES

The first simple design used to test the functionality of the proposed design consists of three modules. Each module has 2 inputs and 1 output. The modules are AND, OR, XOR gates. Each module contains fault detection circuitry as described in the methodology. Accordingly, the module outputs comprise a check bit C_N which defines the status of the module if it is faulty or not. If the module is fault-free, C_N equals '1'. On the other hand, if the module is faulty, C_N equals '0'. The Partially Reconfigurable block (P_b) inputs and outputs are determined according to the maximum number of inputs and outputs in the three modules. As a result, the partially reconfigurable (PR) block will have only 2 inputs and 1 output as shown in Figure 39.

The inputs of the partially reconfigurable block are controlled by module CTRL. This module selects the reconfigurable block inputs from the three module inputs. The inputs of the CTRL module are connected to all the inputs of the three modules and all the check bits C_0, C_1, and C_2. The outputs of CTRL which are E0 and E1 are connected to the inputs of the reconfigurable block. When an error is detected using the check bit C_N, the CTRL module sends the inputs of the faulty module to the inputs of the reconfigurable block. Simultaneously, check bit C_N acts also as stimuli to the FPGA configuration circuit to download the Partial bit file of the faulty module in the Partially Reconfigurable block which is the spare module for the first faulty module.

Three 2-to-1 multiplexer is implemented for each output in order to control which output is connected to the FPGA output pins. The output of each module and the output of the Partially Reconfigurable block (OUT) are associated to the inputs of the multiplexer and the selection line is connected to the check bit C_N of the module. In the fault-free situation, the output of the M_N module will be selected. But, if an error is detected, the output of the reconfigurable block is selected as shown in Figure 61.

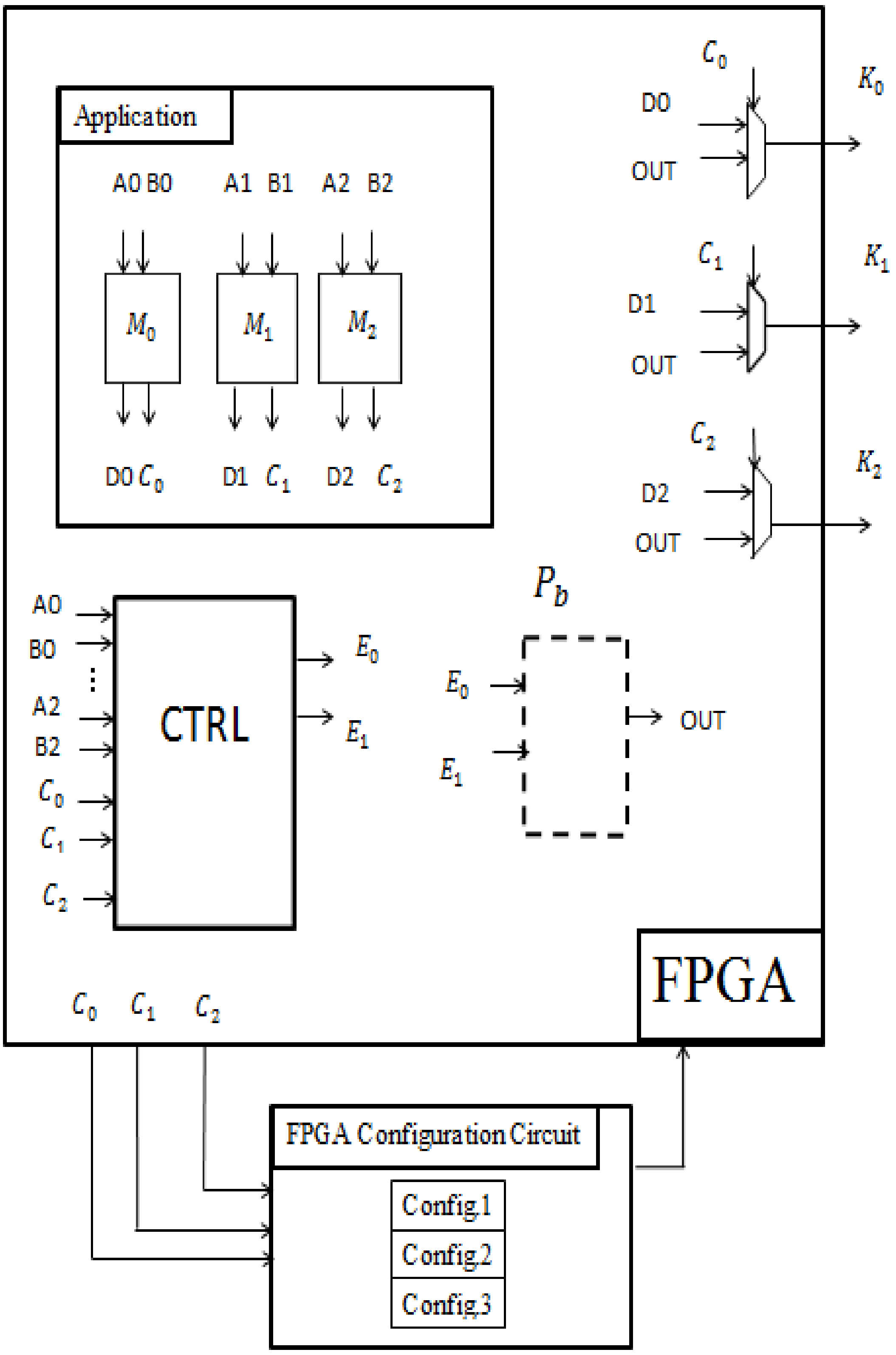


Figure 61: Simple logic gates block diagram

Figure 62 is extracted from the Xilinx PLanAhead tool. It shows how the design is mapped and placed in the FPGA. In addition, it shows how the partial reconfigurable block is specified in the design to be used for partial reconfiguration.

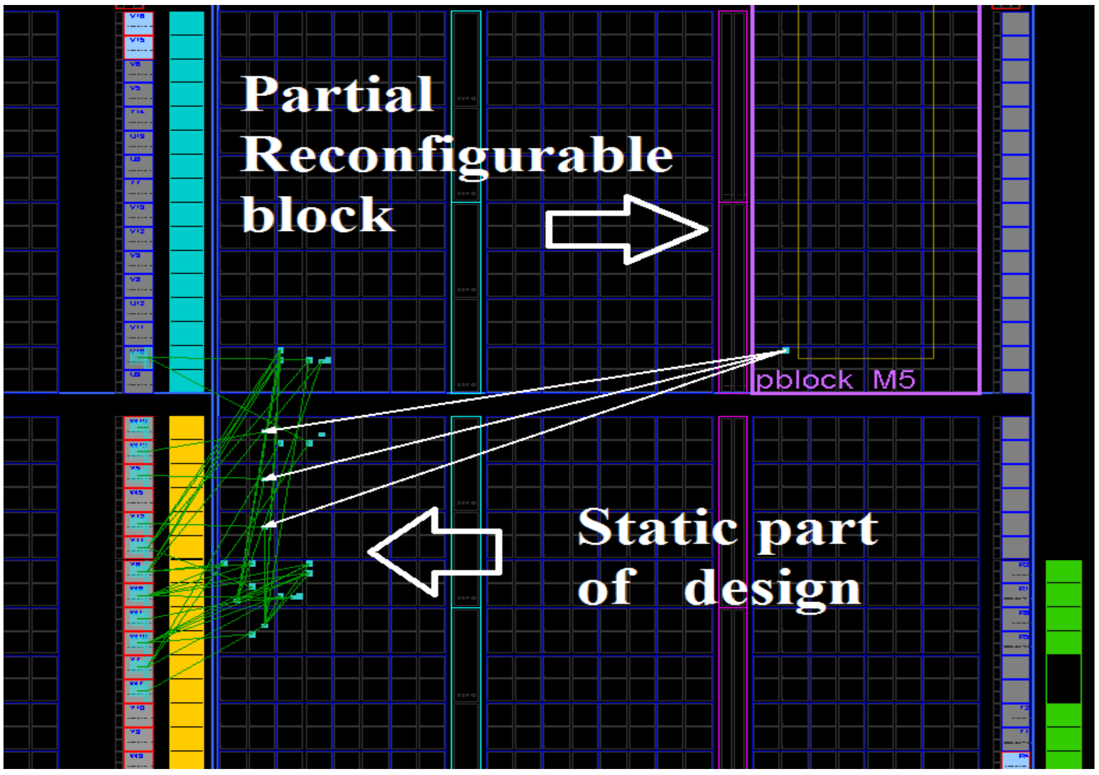


Figure 62: Partially Reconfigurable block implementation in PlanAhead for simple logic gates.

IV.2.2. SIMPLE 2 BIT ALU

The purpose of this design is to ensure that the proposed technique can be applied to designs which consist of modules that have different number of inputs and outputs. The application consists of three modules which are a 2-input AND gate and a 2-input OR gate. In addition, there is a 2-bit ADDER. AND gate and OR gate have the same number of inputs and outputs which is 2 inputs and 1 output. However, the 2-bit ADDER has different number of inputs and outputs which is 5 inputs and 3 outputs.

Each module contains fault detection circuitry as described in the methodology. Accordingly, the module outputs comprise a check bit C_N which defines the status of the module if it faulty or not. If the module is fault-free, C_N equals '1'. On the other hand, if the module is faulty, C_N equals '0'. The Partially Reconfigurable block (P_b) inputs and outputs are determined according to the maximum number of inputs and outputs in three modules. Hence, the partially Reconfigurable (PR) block will have 5 inputs and 3 outputs as shown in Figure 63.

The inputs of the partially reconfigurable block are controlled by module CTRL. This module selects the reconfigurable block inputs from the three module inputs. The inputs of the CTRL module are connected to all the inputs of the three modules and all the check bits C_0, C_1, and C_2. C_2 consists of 3 bits because the 2-bit ADDER has 3 outputs so each bit represents the check bit of one of the outputs . The outputs of CTRL which are E0, E1, E2, E3 and E4 are connected to the inputs of the reconfigurable block. When an error is detected using the check bit C_N, the CTRL module sends the inputs of the faulty module to the inputs of the reconfigurable block. Simultaneously, check bit C_N acts also as indicator through which the FPGA configuration circuit is informed about which module is faulty. Accordingly, the Partial bit file of the faulty module is downloaded in to the Partially Reconfigurable block which is the spare module for the first faulty module.

Five 2-to-1 multiplexer are implemented for each output in order to control which output is connected to the FPGA output pins as shown in Figure 63.

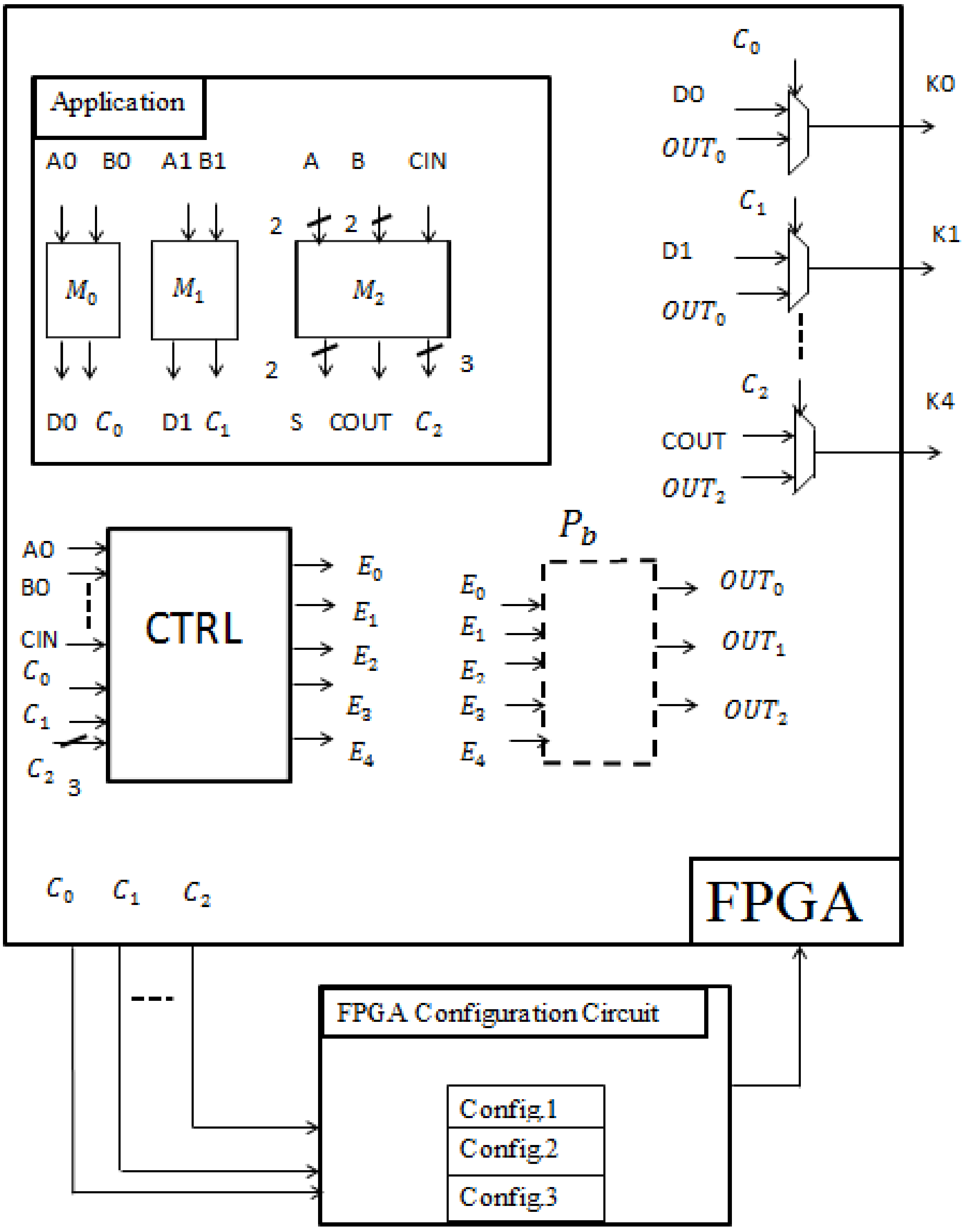


Figure 63: Simple 2-bit ALU block diagram

Figure 64, which is extracted from the Xilinx PlanAhead tool, demonstrates how the simple 2-bit ALU circuit is mapped and routed in the FPGA. In addition, it shows how the partial reconfigurable block, which has 5 inputs and 3 outputs, is specified in the design to be used for partial reconfiguration.

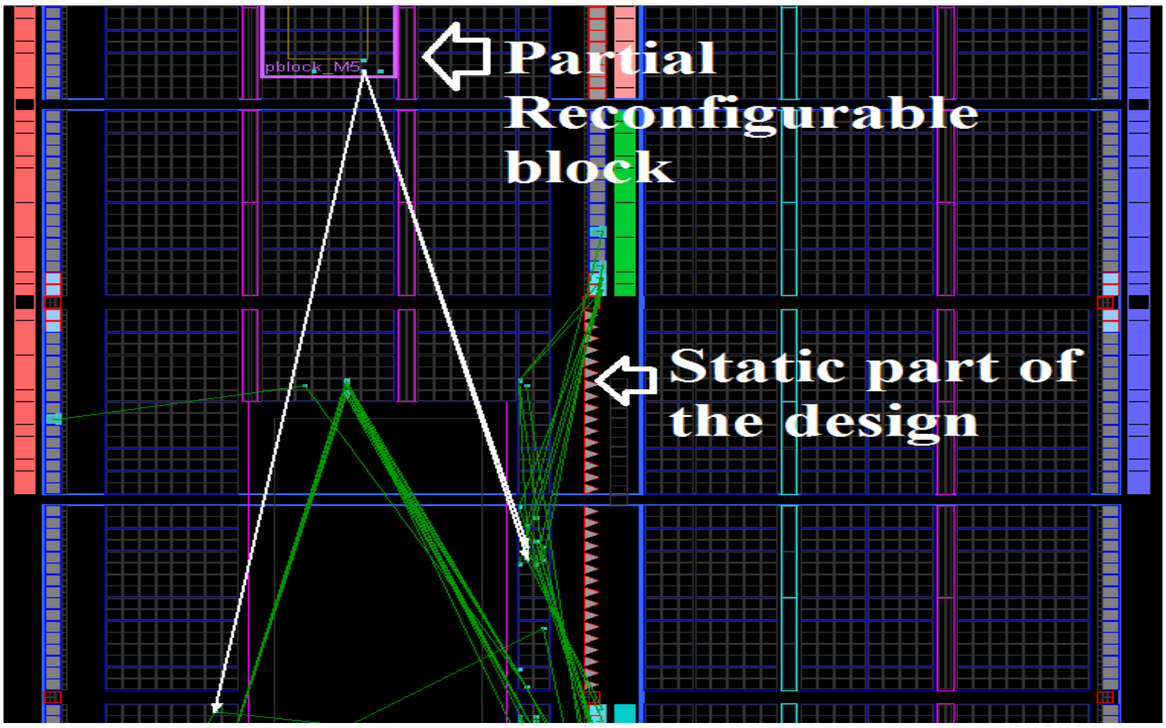


Figure 64: Partially Reconfigurable block implementation in PlanAhead for simple 2-bit ALU.

IV.2.3. UART

A Universal Asynchronous Receiver/Transmitter (UART) is one of the important circuits which are particularly used in systems that exchange information and interact remotely via serial data channels using serializer/deserializer interfaces for the conversion between serial and parallel formats. An UART contains a transmitter and a receiver. They have a different number of inputs and outputs. The purpose of this case study is to guarantee that the proposed technique can be applied to more complex designs. The Transmitter has five inputs: Reset, Tx_CLK, LD_TX, TX_EN, and TX_DATA which consists of 8 bits. The outputs of the transmitter are Tx_OUT and Tx_EMPTY. While the receiver has also five inputs: Reset, Rx_CLK, ULD_RX, RX_EN, and RX_IN. All receivers' inputs consist of 1 bit. The outputs of the receiver are Rx_EMPTY and Rx_DATA which consists of 8 bits.

Each module contains fault detection circuitry as described in the methodology. Accordingly, the module outputs comprise a check bit C_N which defines the status of the module (if it is faulty or not). If the module is fault-free, C_N equals '1'. On the other hand, if the module is faulty, C_N equals '0'. The Partially Reconfigurable block (P_b) inputs and outputs are determined according to the maximum number of inputs and outputs in three modules. Hence, the partially Reconfigurable (PR) block will have 5 inputs and 9 outputs as shown in Figure 65.

The inputs of the CTRL module are connected to all the inputs of the two modules and all the check bits C_0 and C_1. The outputs of CTRL, which are connected to the inputs of the partially reconfigurable block, are E0, E1, E2, E3 and E4. E4 consists of 8 bits so if the transmitter module becomes faulty, TX_DATA input can be easily sent to the partially reconfigurable block inputs.

Eleven 2-to-1 multiplexers are implemented for each output. In the fault free case, the output comes from the main module. On the other hand, if any module of the design becomes faulty, the output comes from the partially reconfigurable block which will be selected as shown in Figure 65.

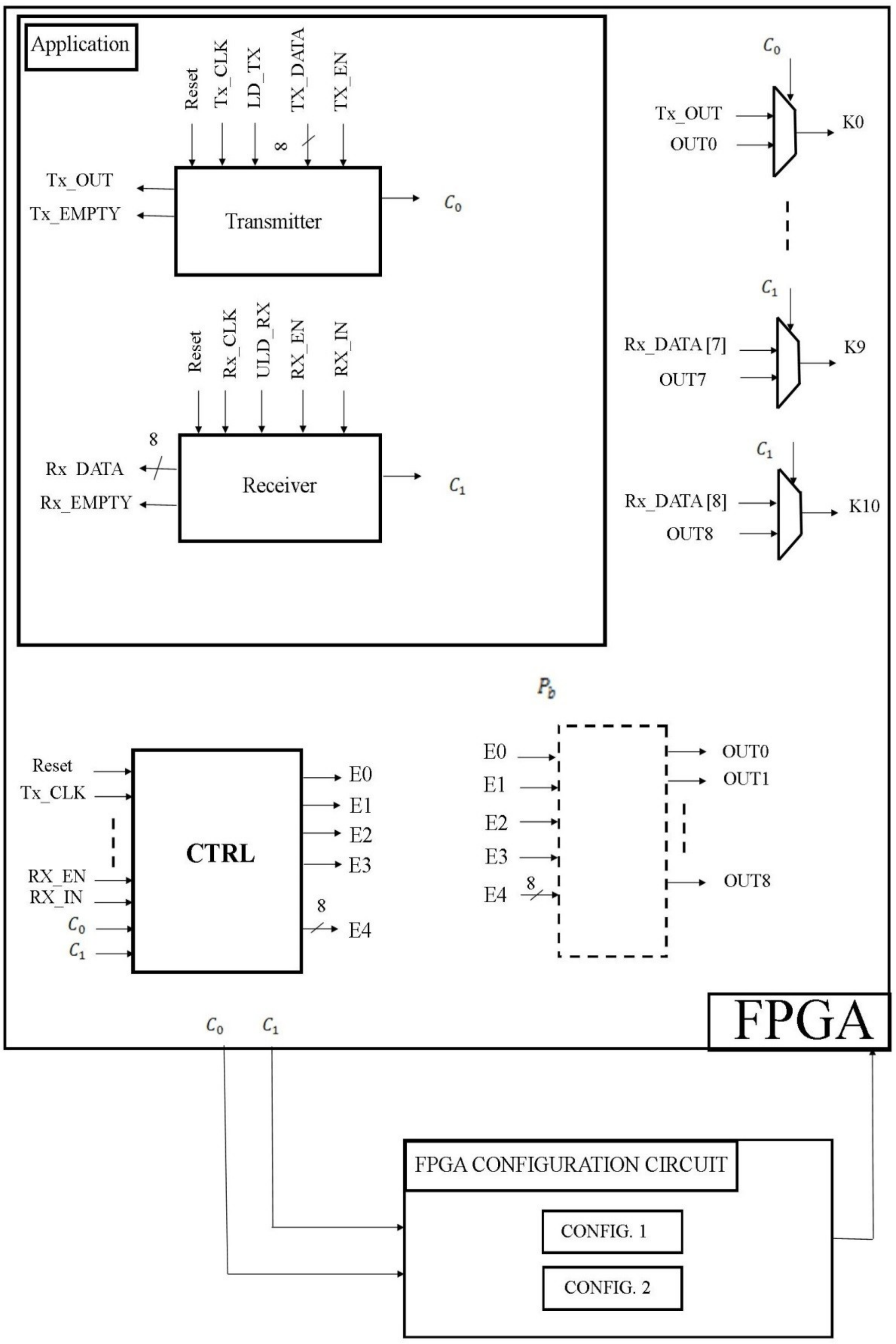


Figure 65: UART block diagram

Figure 66 which is extracted from the Xilinx PlanAhead tool demonstrates how the UART circuit is mapped and routed in the FPGA. Moreover, it shows the Partially Reconfigurable block implementation and how it is connected to the static part of the design. The proposed scheme can recover from faults in either the transmitter or the receiver even though they consume a different number of slices.

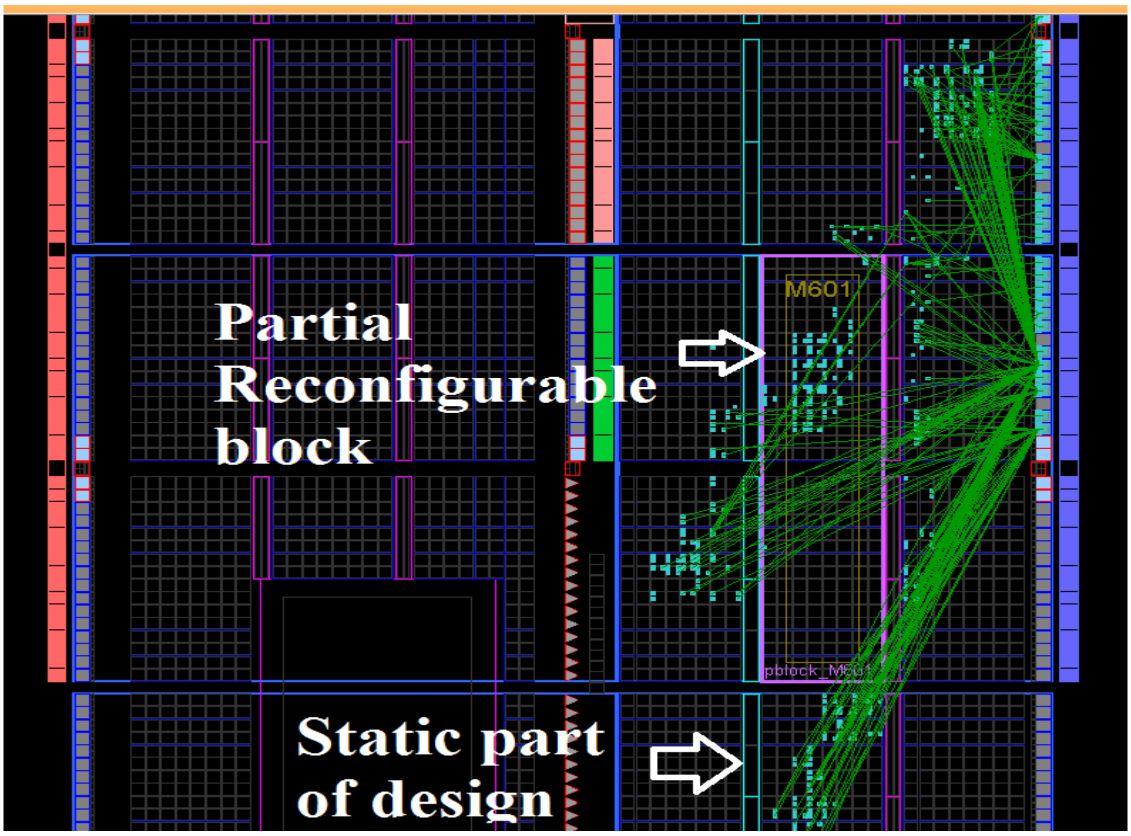


Figure 66: Partially Reconfigurable block implementation in PlanAhead for UART.

IV.2.4. PACEMAKER

A pacemaker is a real time medical device that regulates heart beats and provides electrical stimulation through electrodes to the heart if the heart beats are too slow or if the heart does not beat at all [Kaur 2014]. Accordingly, improving the reliability of these devices and extending their lifetime is crucial. In this case study, the technique is applied to the dual chamber pacemaker.

Pacemakers are classified into two types: single chamber and dual chamber. Single chamber pacemaker means that it controls only one chamber (atrium or ventricle) in the heart but the dual chamber pacemaker controls two chambers in the heart. Dual chamber provides more advantages than the single chamber pacemaker in terms of patient life quality such as reduction in atrial fibrillation, pacemaker syndrome and heart failure. Accordingly, the technique is applied to the dual chamber pacemaker. It consists of a Controller, TimerA, TimerV and an oscillator as shown in Figure 67.

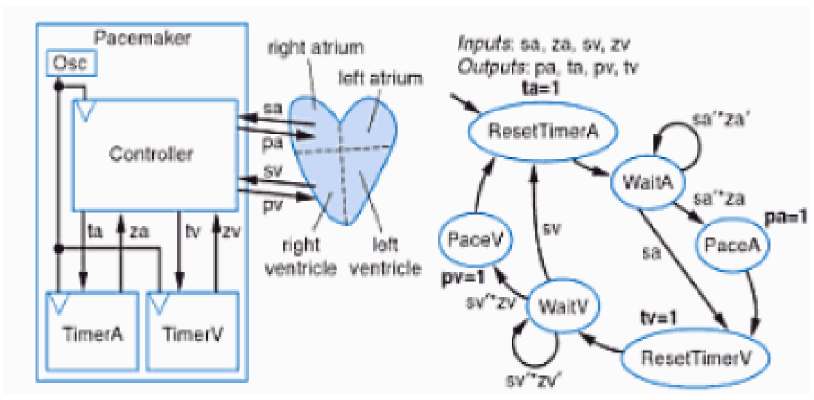


Figure 67: Pacemaker block diagram and Controller state machine [Kaur 2014]

The controller is responsible for detecting the electrical activity of the heart (atrium contractions or ventricle contractions) and for responding accordingly. If the contraction is not sensed, it sends a stimulating impulse to the heart. TimerA and TimerV are the timers of the beats coming from the atrium and ventricle. Figure 67 also shows the state diagram of the dual chamber pacemaker.

The proposed approach is applied to the dual chamber pacemaker. The purpose of this case study is to certify that the proposed technique can be implemented in more complex critical systems with a larger number of modules. The controller of the dual chamber pacemaker has 6 inputs and 4 inputs. The inputs are CLK, reset, sa, sv, za, and zv. The outputs are ta, tv, pa and pv. Timer A module has 3 inputs: CLK, reset and ta1. Additionally, it has 1 output which is za1. While Timer V module has also 3 inputs: CLK, reset and tv1 and it has 1 output which is zv1. Finally, the oscillator has only 1 output.

Each module contains fault detection circuitry as described in the methodology. Accordingly, the module outputs comprise a check bit C_N which defines the status of the module (if it is faulty or not). The Partially Reconfigurable block (P_b) inputs and outputs are specified according to the maximum number of inputs and outputs in three modules. Hence, the partially Reconfigurable (PR) block will have 5 inputs and 4 outputs as shown in Figure 68.

The inputs of the CTRL module are connected to all the inputs of the five modules and all the check bits C_0 , C_1 , C_2, C_3 , C_4, C_5 , C_6 and C_7. The outputs of CTRL which are connected to the inputs of the partially reconfigurable block are E0, E1, E2, E3, E4 and E5.

For each output of a module, nine 2-to-1 multiplexers are implemented in order to control which output is connected to the next module that it feeds as shown in Figure 68.

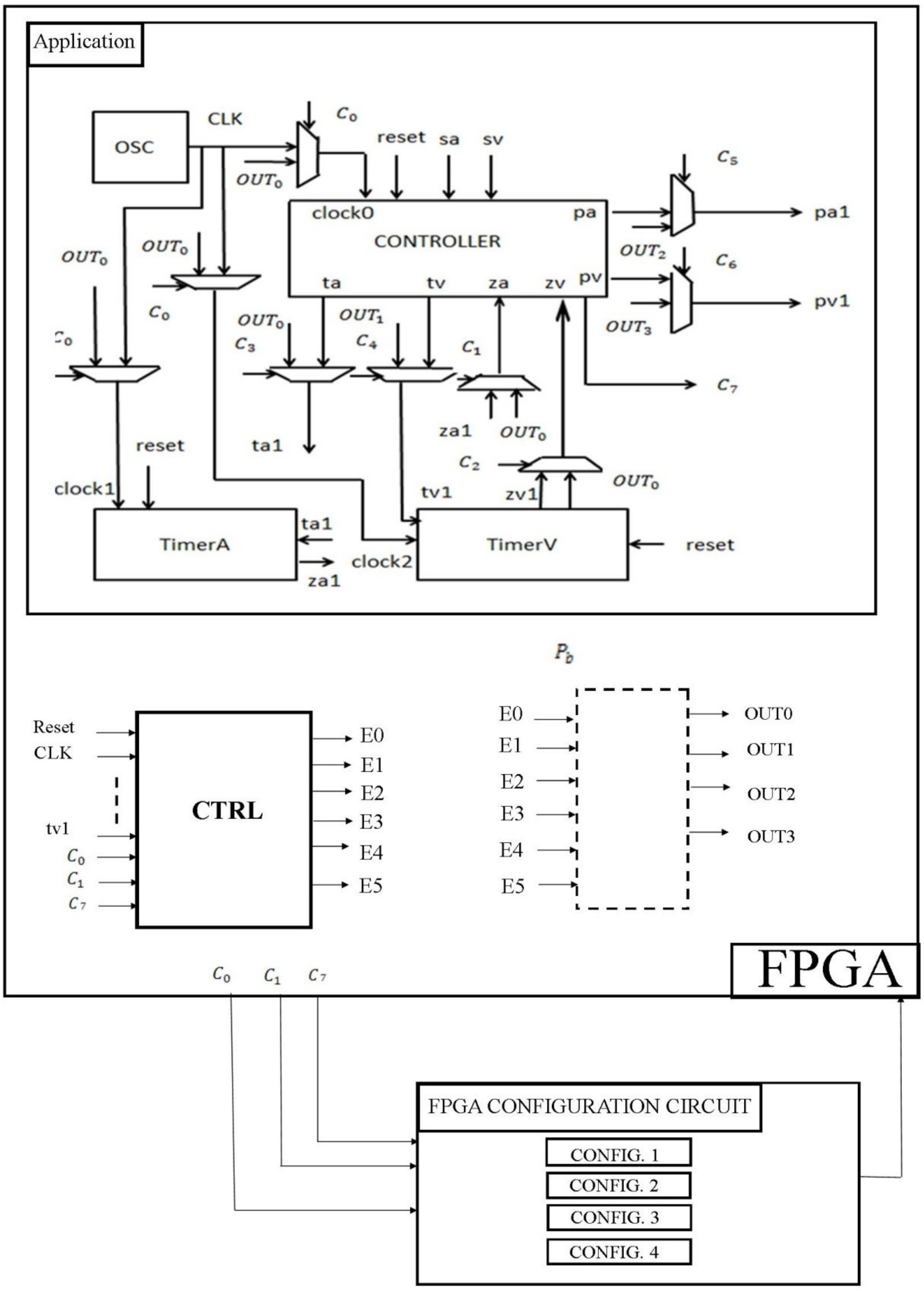


Figure 68: Pacemaker block diagram.

Figure 69 shows the mapping and placement of the pacemaker design in the FPGA. Furthermore, it shows how the partial reconfigurable block is used as a spare for the first defective module.

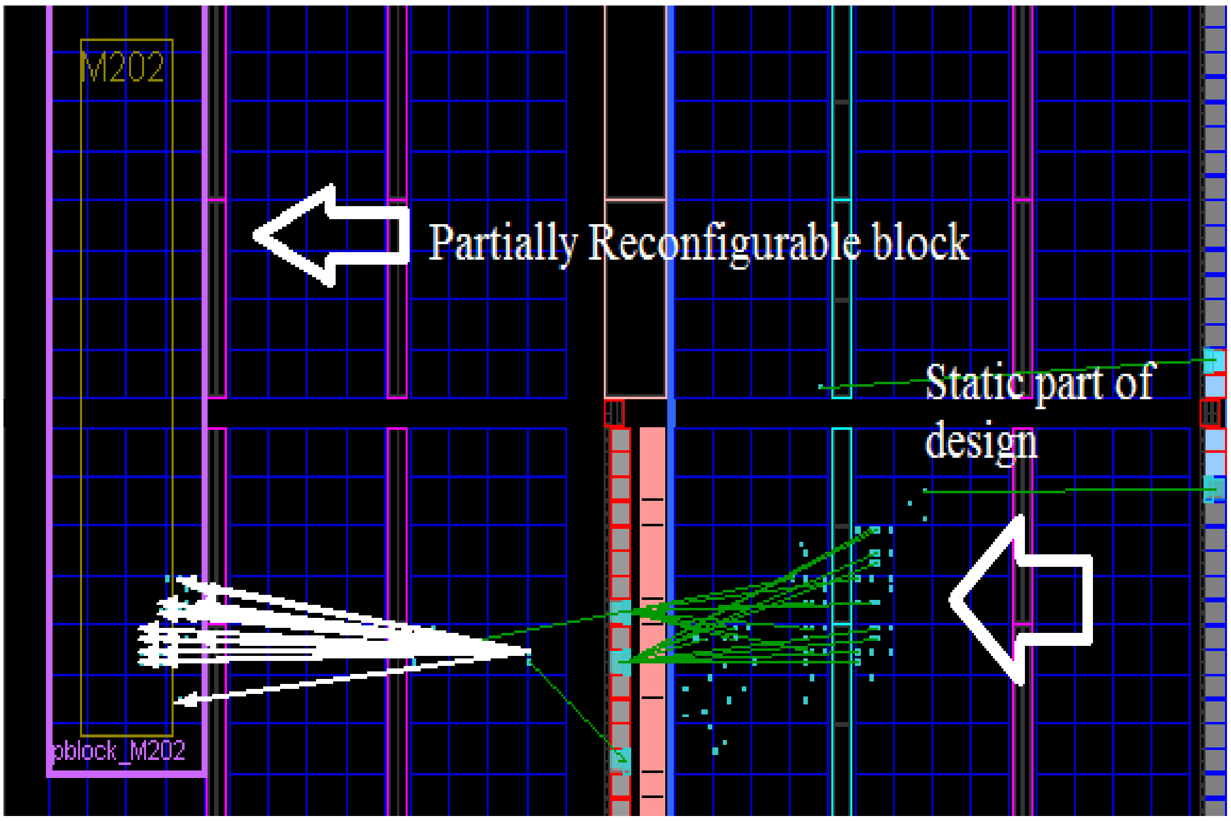


Figure 69: Partial reconfigurable block implementation in PlanAhead for Pacemaker.

IV.3. Results

The results of UART and Pacemaker case studies if it is implemented using the proposed technique are shown in Table 2.

Table2: Results of UART and Pacemaker case studies.

	UART	Pacemaker
no. of slices without approach	46	13
no. of slices with approach	164	51
recovery time	0.15ms	0.15ms
no. of configuration files	3	5

The results of UART and Pacemaker case studies if implemented using the proposed technique are compared to the results if implemented by [Di Carlo 2014] technique because [Di Carlo 2014] is the closest technique to the proposed technique in this Chapter. Equation 1 in [Di Carlo 2014] which is shown in Figure 70 is used to calculate the number of configuration files needed for the implemented design.

$$n\_conf\_files = n\_logic\_tiles \times n\_faults + \\ + \prod_{i=0}^{n\_faults-1} (n\_logic\_tiles - i) \times (n\_faults + 1) + \\ + n\_logic\_tiles$$

Figure 70: Di Carlo 2014 equation.

The second term of the equation represents the number of the interconnection tiles added to the design when [Di Carlo 2014] technique is used.

To have a fair comparison between the proposed technique and [Di Carlo 20l4] technique, the equation is used to calculate how many interconnection tiles will be added to the case studies and the number of configuration files generated for them when they are implemented using [Di Carlo 2014] technique. If UART is implemented using [Di Carlo 20l4] technique, 2 interconnection tiles will be added to the design. If the Pacemaker is designed with the same technique, 2 interconnection tiles will also be added to the design. As a result, this technique will have a larger area overhead compared to the proposed technique in which the interconnections between modules are mainly handled using already fabricated multiplexers. In addition, when the UART is implemented by the proposed technique, it requires only 3 configuration files as shown in Table 2 to tolerate one permanent fault compared to [Di Carlo 20l4] which requires 9 configuration files. For the Pacemaker, only 5 configuration files are needed compared to 17 configuration files when using the technique in [Di Carlo 20l4]. Accordingly, the memory space required for bitstreams using the proposed technique is smaller than [Di Carlo 20l4]. Note that one of the contributions of [Di Carlo 20l4] technique that it requires a smaller memory space for configuration files when compared to other existing techniques.

Accordingly, the advantages of the proposed technique that it has low area overhead because no interconnection tiles are added to the design compared to [Di Carlo 20l4] technique. Also, it has simple recovery hardware compared to complex recovery hardware of [Di Carlo 20l4] technique. The memory space needed to store configuration files in the proposed technique is smaller than [Di Carlo 20l4] technique. However, it can only recover from the first fault in any of the system modules compared to [Di Carlo 20l4] technique that can recover for more than one fault. For critical applications with short mission times, it is not a serious problem in which the probability of more than one fault during a short period is negligible [Siewiorek 1998].

V. CONCLUSIONS

Nowadays, Field programmable Gate Arrays (FPGAs) are widely used in many electronic applications particularly safety-critical design applications. These applications require high performance, reconfigurability and low development cost. In addition, they need high reliability, availability and long system lifetime. As a result, fault tolerant techniques in FPGAs are required in these scenarios.

Recently, several techniques are proposed to mitigate soft errors while just a few researches were presented to handle hard errors. Accordingly, in this thesis, handling permanent faults is the main motivation of the research.

In this thesis, two fault-tolerant approaches for FPGA-based applications are suggested with a built-in fault detection region. For fault detection, a low cost fault detection scheme is proposed for detecting faults using the fault detection region used in both schemes. The fault detection scheme is the Duplication With Compare (DWC) technique. It mainly detects open faults in the programmable interconnect resources in the FPGAs. Furthermore, Stuck-At faults and Single Event Upsets (SEUs) fault can be detected.

For fault recovery, each scheme has its own fault recovery approach. The first technique uses a spare module and a 2-to-1 multiplexer to recover any fault detected. However, the second approach recovers any fault detected using the runtime Partial Reconfiguration (PR) property of FPGAs. It is based on identifying a Partially Reconfigurable block (P_b) in the FPGA which is used in the recovery process after the first defective module is recognized in the system. This technique uses only one location for faults in any of the FPGA's modules and the FPGA interconnects.

Functional simulations indicate that both techniques are operational for circuits with different complexity. Accordingly, it can detect and recover from faults. Xilinx ISE Simulator ISIM was used in the simulations. In addition, Xilinx PlanAhead tool was used in creating partial configuration bitstreams. Both techniques consume low area overhead and have simple recovery hardware.

REFERENCES

[Arumi 2008] D. Arumí, R. Rodríguez-Montañés, J. Figueras, S. Eichenberger, C. Hora and B. Kruseman, "Full Open Defects in Nanometric CMOS", 26th IEEE VLSI Test Symposium, 2008.

[Avizienis 1967] A. Avizienis, "Design of fault-tolerant computers", Fall Joint Computer Conference, 1967.

[A'ain 1995] A.K.B. A'ain, A.H. Bratt and A.P. Dorey," Exposing floating gate defects in analogue CMOS circuits by power supply voltage control testing technique", Proceedings of 8th International Conference on VLSI Design , pp.239-242, New Delhi, India, January 1995.

[Becker 2007] T. Becker, W. Luk, and P. Cheung, "Enhancing relocatability of partial bitstreams for run-time reconfiguration," Annual Symposium on Field-Programmable Custom Computing Machines, 2007.

[Bilotti 1966] A. Bilotti, "Operation of a MOS Transistor as a Varaible Resistor" IEEE Proceedings, August 1966.

[Bolchini 2013] C. Bolchini, A. Miele, and C. Sandionigi, "Autonomous fault-tolerant systems onto SRAM-based FPGA platforms," Journal of Electronic Testing, vol. 29, no. 6, pp. 779–793, 2013.

[Biag 2012] Hassan Baig and Jeong-A Lee, "An Island-style-routing compatible fault-tolerant FPGA architecture with self-repairing capabilities", Int. Conf. on Field-Programmable Technology (FPT), Seoul, South Korea, Dec. 2012.

[Brown 1992] S. Brown, R. Francis, J. Rose, and Z. Vranesic,"Field programmable gate arrays", Kluwer, 1992.

[Brown 1996] Stephen Brown, Jonathan Rose, "FPGA and CPLD Architectues: A tutorial",1996.

[Carmichael 1999] C. Carmichael, E. Fuller, P. Blain, and M. Caffrey,"SEU mitigation techniques for Virtex FPGAs in space application," MAPLD99, Laurel, MD, USA, Sep. 1999.

[Carmichael 2000] C. Carmichael, M. Caffrey, and A. Salazar," Correcting single-event upsets through Virtex partial configuration," Xilinx Application Notes, XAPP216, Vol. 1, 2000.

[Clarke 1990] P.J. Clarke et al, "Electromigration - a tutorial introduction", International Journal of Electronics, 69:3, p333 – 338, 1990.

[Constraints guide 2012] Constraints guide**,** UG625, 2012.

[Di Carlo 2010] Stefano Di Carlo, Andrea Miele, Paolo Prinetto, and Antonio Trapanese, "Microprocessor Fault-tolerance via on-the-fly partial reconfiguration", 15th IEEE European Test Symposium (ETS), Prague, Czech Republic, May 2010.

[Di Carlo 2014] Stefano Di Carlo, Giulio Gambardella, Paolo Prinetto, Daniele Rolfo, Pascal Trotta, and Alessandro Vallero, "A novel methodology to increase fault tolerance in autonomous FPGA-based systems", 20th International IEEE On-Line Testing Symposium (IOLTS), Catalunya, Spain, Jul. 2014.

[Dieter 2003] K. Dieter et al, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing", Journal of Applied Physics, vol. 94, no.1, July 2003.

[Drahonovsky 2014] Tomas Drahonovsky, Martin Rozkovec and Ondrej Novak. “A Highly Flexible Reconfigurable System on a Xilinx FPGA”, ReConFigurable Computing and FPGAs confrence, Cancún, Mexico, Dec. 2014

[Dubrova 2013] Elena Dubrova, “ Fault-Tolerant Design”, KTH Royal Institute of Technology, Krista, Sweden, 2013.

[Esseni 2002] D. Esseni et al, “On Interface and Oxide Degradation in VLSI MOSFETs—Part I: Deuterium Effect in CHE Stress Regime”, IEEE Transactions on Electron Devices, vol. 49, no. 2, February 2002.

[Gauer 2010] C. Gauer, B.J. LaMeres, and D. Racek, "Spatial avoidance of hardware faults using FPGA partial reconfiguration of tile-based soft processors," Proceedings of the IEEE Aerospace Conference, March 2010.

[Godal 2001] Bryan S. Godal, Russell P. Kraft, Steven R. Carlough, Thomas W. Krawczyk Jr and John F. McDonald,” Gigahertz Reconfigurable Computing using SiGe HBT BiCMOS FPGAs”, 11th International Conference on Field Programmable Logic and Applications, Belfast, Northern Ireland, UK, Aug. 2001.

[Guerin 2007] C. Guérin et al, “The Energy-Driven Hot-Carrier Degradation Modes of nMOSFETs”, IEEE Transactions on Device and Materials Reliability, vol. 7, no. 2, June 2007.

[Harris 2002] I.G. Harris et al, “Testing and diagnosis of interconnect faults in cluster-based FPGA architectures”, IEEE Transactions on CAD of Integrated Circuits and Systems, vol. 21, no. 11, p 1337-43 Nov. 2002.

[IEEE 1993] IEEE, “Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164)” IEEE Std 1164-1993, 1993.

[ITRS 2011] ITRS, “ITRS: International technology roadmap for semiconductors” http://www.itrs.net/links/2011itrs/home2011.htm.

[Iturbe 2010] X. Iturbe, K. Benkrid, T. Arslan, I. Martinez, M. Azkarate, and M. Santambrogio, “A roadmap for autonomous fault-tolerant systems,” Design and Architectures for Signal and Image Processing (DASIP) conference, Edinburgh, Scotland, Oct 2010.

[Jing 2011] Naifeng Jing, Ju-Yueh Lee, Weifeng He1, Zhigang Mao, and Lei He, “Mitigating FPGA Interconnect Soft Errors by In-Place LUT Inversion”, IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA, Nov. 2011.

[Johnson 2008] Jonathan Johnson, William Howes, and Michael Wirthlin, “Using Duplication with Compare for On-line Error Detection in FPGA-based Designs”, IEEE Aerospace and Electronic Systems Society Conference, Big Sky, MT, USA, Mar. 2008.

[Kananaru 2009] A.Kanamaru, H. Kawai, Y. Yamaguchi, and M.Yasunaga, “Tilebased fault tolerant approach using partial reconfiguration,” in Proc. of the 5th Int. Workshop on Reconfigurable Computing: Architectures, Tools and Applications (ARC ’09), Karlsruhe, Germany, Mar. 2009.

[Kastensmidt 2004] F.G. de Lima Kastensmidt, G. Neuberger, R.F. Hentschke, L. Carro, and R. Reis, "Designing fault-tolerant techniques for SRAM-based FPGAs", IEEE Design & Test of Computers, vol.21, no.6, pp. 552-562, Dec. 2004.

[Kaur 2014] Amandeep Kaur, “Design and Analysis of A Dual Chamber Cardiac Pacemaker Using VHDL in Biomedical Application”, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), vol. 2, no. 7, July 2014.

[Lach 1998] John Lach, William H. Mangione-Smith, and Miodrag Potkonjak, “Low overhead fault-tolerant FPGA systems”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 6, no. 2, Jun. 1998.

[Lee 2012] Ju-Yueh Lee, Cheng-Ru Chang, Naifeng Jing, Juexiao Su, Shijie Wen, Rich Wong, and Lei He, “Heterogeneous Configuration Memory Scrubbing for Soft Error Mitigation in FPGAs”, International Conference of Field-Programmable Technology (FPT), Seoul, Korea, Dec. 2012.

[Li 2005] James Chien-Mo Li, and Edward J. McCluskey, “Diagnosis of Resistive-Open and Stuck-Open Defects in Digital CMOS ICs“, IEEE Transaction on Computer Aided Design of Integrated Circuits and systems, vol. 24, no. 11, Nov. 2005.

[Mondal 2005] Somsubhra Mondal and Seda Ogrenci Memik, "Fine-grain Leakage Optimization in SRAM based FPGAs", 15th ACM Great Lakes symposium on VLSI (GLSVLSI), pp. 238-243, New York, USA, 2005.

[Nagi 1996] N. Nagi and J. Abraham, "Hierarchical Fault Modeling For Linear Analog Circuits," Journal of Analog Integrated Circuits and Signal Processing, Vol. 10, pp. 89-99, 1996.

[Nam 2005] Ilku Nam, Bonkee Kim, and Kwyro Lee,” CMOS RF amplifier and mixer circuits utilizing complementary Characteristics of parallel combined NMOS and PMOS devices” , IEEE Transaction on Microwave Theory and Techniques, vol. 53, no. 5, May 2005.

[Olbrich 1996] T. Olbrich, J. Perez, I. A. Grout, A. M.D. Richardson and C. Ferrer, “Defect-Oriented VS Schematic-Level Based Fault Simulation for Mixed-Signal ICs”, Proceedings of the IEEE International Test Conference, Washington, DC, USA, October 1996.

[Official site of Xilinx] Official Site of Xilinx, Available: http://www.xilinx.com/itp/xilinx10/isehelp/ise_c_fpga_design_flow_overview.htm

[Peterson 1972] W. Peterson and E. Weldon, Error-correcting Codes, 2nd. ed., MIT Press, Cambridge, 1972.

[Pradeep 2014] C. Pradeep, R. Randahakrishnan and Philp Samuel, “Fault Recovery Algorithm using King Spare Allocation and shortest Path Shifting for Reconfigurable Systems”, Journal of Theoretical and Applied Information Technology, pp. 254-261, Mar. 2014.

[Pratt 2007] Brian Pratt, Michael Caffrey, James F. Carroll, Paul Graham, Keith Morgan, and Micheal Wirthlin, ”Fine-Grain SEU Mitigation for FPGAs UsingPartial TMR“, Radiation and Its Effects on Components and Systems (RADECS) Conference, Deauville, France, Sep. 2007.

[Psarakis 2012] Mihalis Psarakis and Andreas Apostolaki, “Fault-tolerant FPGA processor based on runtime reconfigurable modules”, 17th IEEE European Test Symposium (ETS), Annecy, France, May 2012.

[Ramya 2013] N.Ramya, and N. Kirthika, “A Novel BIST based Diagnosis Technique to Detect Faults in FPGA”, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), vol. 2, no. 3, Mar. 2013.

[Saba2010] A. Saba and N. Manna, Digital principles and Logic Design, Infinity Science Press, Hingham, Massachussetts. USA, 2010.

[Salvador 2011] Ruben Salvador, Andres Otero, Javier Mora, Eduardo de la Torre, Lukaš Sekanina and Teresa Riesgo, “Fault Tolerance Analysis and Self-Healing Strategy of Autonomous Evolvable Hardware Systems”, 2011 International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, Dec 2011.

[Sedra 2004] Adel S. Sedra and Kenneth C. Smith, Microelectronic Circuits, Oxford University Press, New york, USA, 2004.

[Siewiorek 1998] D.P. Siewiorek and R.S. Swarz, "Reliable Computer Systems – Design and Evaluation," A K Peters, Natick, Massachusetts, 1998.

[Shooman 2002] M.L. Shooman, Reliability of Computer Systems and Networks, John Wiley, New York, NY, USA, 2002.

[Sterpone 2008] Luca Sterpone, "Electronics System DesignTechniques for Safety Critical Applications", Politecnico di Torino Torino, Italy,2008.

[Stonham1996] T.J. Stonham, *Digital Logic Techniques: Principles and Practice,* Chapman & Hall, London, UK, 1996.

[Tahoori 2002] Mehdi Baradaran Tahoori, "Diagnosis of Open Defects in FPGA Interconnect", in Proceeding of IEEE International Conference Field-Programmable Technology, pp. 328–331, 2002.

[Virtex-4 Libraries Guide for HDL Designs 2010] Virtex-4 Libraries Guide for HDL Designs,2010.

[Wakerly 2006] [John F. Wakerly, Digital Design – Principles and Practices, Prentice Hall, Upper Saddle River, New jersey, USA, 2006.

[Xilinx 2010] Xilinx, "Virtex-4 Family Overview", DS112 datasheet, Aug. 2010.

[Xilinx 2013] Xilinx, "Xilinx Partial Reconfiguration user guide", UG702, April 2013.

[Xu 2003] Weifeng Xu, Ramshankar Ramanarayanan, and Russell Tessier, "Adaptive Fault Recovery for Networked Reconfigurable Systems", 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'03), Napa, California, April 2003.

[Yui 2003] C.C. Yui, G.M. Swift, C. Carmichael, R. Koga and J.S. George, "SEU Mitigation Testing of Xilinx Virtex II FPGAs", IEEE Radiation Effects Data Workshop, Monterey, CA, USA, Jul. 2003.

[Zhang 2013] J. Zhang, Y. Guan, and C. Mao, "Optimal partial reconfiguration for permanent fault recovery on SRAM-based FPGAs in space mission," Advances in Mechanical Engineering, vol. 2013, 2013.