2011

A slice fault aware tool chain for FPGAs

Adwait Gupte
Iowa State University

A slice fault aware tool chain for FPGAs

by

Adwait Gupte

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Phillip H. Jones, Major Professor
Joseph Zambreno
Alexander Stoytchev

Iowa State University

Ames, Iowa

2011

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to take this opportunity to thank all the people who have made this thesis possible. First and foremost, I'd like to thank my advisor Dr. Phillip Jones. Our discussions on hardware design and research ideas have strengthened and broadened my understanding of hardware design in particular and computer engineering in general. The countless hours he has put into editing my work have transformed my horrendous first drafts into things which I am proud to have my name on. I could have never completed this thesis if it weren't for his constant support and the tremendous patience he has shown from inception to execution to writing of this thesis. I truly appreciate his willingness to find time in his schedule to help me with everything from research to teaching me how to drive on the "other" side of the street. He has truly been a mentor to me through my stay at Iowa State and continues to be one in my professional life.

I'd like to thank Dr. Manimaran Govindrasu for the course project discussions that led to the idea for this thesis. I'd also like to Dr. Joseph Zambreno and Dr. Alexander Stoytchev for their patience in dealing with my wild vacillations in choosing the topic of my thesis. A big thank you to Dr. Stoytchev for pointing out the administrative roadblocks that I would have run into if I had continued on with my earlier thesis plans. He saved me countless hours in wasted effort.

Finally I'd like to thank my family and all my friends for their help and support. I cannot begin to enumerate the ways in which you helped me, so I'll just say - Thank You!

ABSTRACT

Integrated circuits have become more complex over the past few years. As processes have improved, it has become possible to pack an ever increasing number of transistors into an Integrated Circuit (IC). At the same time, the functions being performed in the ICs have increased in complexity many fold. The end result is that modern ICs are not only denser, but also have a larger chip area [1]. As a result of both of these factors, maintaining a high yield has become a problem for the IC industry. This problem is present in the manufacturing process for all integrated circuits. This work, however, only focuses on Field Programmable Gate Arrays (FPGA).

FPGAs are a type of programmable hardware that allow arbitrary digital circuits to be implemented into the fabric after the manufacturing process. In this work I propose a method that could allow manufacturers to increase chip yields by allowing FPGAs with some manufacturing errors to be used. Evaluation experiments suggest that this method can tolerate up to 30% errors in the FPGA's logic fabric without significant reduction in circuit performance as compared to smaller FPGA chips with an equivalent amount of non-faulty logic. This could potentially allow manufacturers to sell larger, faulty chips as a low cost alternative to smaller perfectly manufactured chips.

In brief, the proposed method adds a phase to the tool chain flow that tests and identifies faulty areas in the chip. The information about the faulty areas can then be fed back into the tool flow of the FPGA. This allows the tool chain to avoid the faulty areas of the chip and still place the circuit on the chip in an efficient manner, if possible. The approach aims to reclaim some FPGA ICs that would have ordinarily been classified as unusable and discarded, thus increasing the effective FPGA manufacturing yields.

CHAPTER 1. INTRODUCTION

This chapter describes the motivation for this work. It also gives a brief introduction to the proposed method and discusses the organization of the rest of this thesis.

1.1 Motivation

In VLSI technology, chip yield refers to the ratio of the number of good chips versus the total number of manufactured chips. As yield decreases, the number of chips over which the Non Recurring Engineering (NRE) costs can be amortized reduces, thus increasing the cost per chip. Alternatively stated, the profit per chip is reduced. This makes yield an important consideration for chip manufacturers, one that directly affects their bottom line.

Equation 1.1 [2] gives the relation between different parameters that model the yield of a chip.

$$p(x) = \frac{\Gamma(\alpha + x)}{x!\Gamma(\alpha)}.\frac{(Ad/\alpha)^x}{(1 + Ad/\alpha)^{\alpha+x}} \tag{1.1}$$

where A = Area of the chip

α = Clustering Parameter (the higher the value, the smaller the clustering)

d = Average number of defects per unit of chip area

p(x) = Probability of x defects in the chip

$\Gamma(x)$ = (x-1)!

Thus, the probability of a fault free chip p(0) becomes,

$$Yield = p(0) = (1 + Ad/\alpha)^{-\alpha} \tag{1.2}$$

Technology continues to enable larger and denser FPGAs to host more complex applications. Thus, as the size of the individual chips grows, the probability of having no fabrication faults in the entire chip goes down (Equation 1.2). Therefore methods to make FPGAs usable despite the presence of a few faults could make improvements in the profitability of manufacturers, in addition to extending the life span of the device.

1.2 About proposed method

In this work, a method is proposed to allow FPGAs to be used despite faults. The method makes simple changes to the end user tool chain that can allow both manufacturing defects as well as defects due to aging to be tolerated. Three usage models are discussed below.

1.2.1 Use Case 1: Deploying FPGAs with manufacturing faults

As discussed in Section 1.1, because of the increasing complexity of the chip manufacturing process, keeping yields high is difficult. Challenges emanating from small feature sizes, new processes, and complex circuit designs combine to form a formidable challenge to high yield rates. Let us assume that once a chip is manufactured and found to be fault free, it continues to perform in this manner for a reasonable period of time. If the production volumes are low, then it is feasible to use the method proposed in this thesis to generate individual bitfiles for each "faulty" FPGA that gets used in production. This use case allows small scale users of FPGAs to reduce their material cost by buying faulty chips.

1.2.2 Use Case 2: FPGA used for prototyping

The virtually limitless reconfigurability of FPGAs make them an attractive option for prototyping an Application Specific Integrated Circuit (ASIC) in hardware before spending the large NRE cost required for fabrication. FPGAs are also commonly used in research labs to create prototypes. In such scenarios, a very small number of FPGAs, typically 1, is programmed and reprogrammed to test the impact of different design decisions. Even if the FPGA has some faulty areas, a tool chain using the proposed method can implement the prototype circuit on the FPGA. Post manufacture faults on the FPGA can also be handled in this use case since all faults up to the point of reprogramming are handled by the method.

1.2.3 Use Case 3: FPGA reprogrammed in the field when faults occur

FPGAs deployed in end user systems are programmed once and then continue to work with that bitfile for a long time. If the system fails in the field because of a fault developing on the FPGA, then it is possible to generate a new bitfile that avoids new faults and reprogram the FPGA in the field. This extends the lifetime of the unit, as opposed to replacing it with a new unit.

1.3 Overview

The two main contributions of this work are: 1) a method that leverages existing tools to efficiently implement designs on FPGAs that have faults; and 2) an evaluation of the effectiveness of the proposed approach. The findings presented in this thesis were first reported in a conference paper [3].

The rest of this thesis is organized as follows. Chapter 2 provides background material germane to the topic of this thesis. Chapter 3 covers the literature published on related topics. Chapter 4 provides a detailed description of the proposed method. Chapter 5 discusses the evaluation methodology that was used. Chapter 6 describes the results and their analysis. Chapter 7 closes this thesis with the conclusion and a discussion of potential directions for future work.

CHAPTER 2. BACKGROUND

This chapter introduces some of the basic concepts and terminology necessary to understand the contents of this thesis. Section 2.1 describes the basics of a Field Programmable Gate Array (FPGA). Section 2.2 describes the tools that are used to implement a design on an FPGA.

2.1 About FPGAs

Latency and throughput are constraints that are orthogonal to system flexibility when trying to make a trade-off between implementing a function in software versus implementing it in an Application Specific Integrated Circuit (ASIC). Although software provides the maximum flexibility in terms of changing the system, it's throughput is lower and latency higher than an ASIC. On the other hand, ASICs provide the absolute lowest latency possible and a much higher throughput than software, but cannot be changed easily. Furthermore, ASICs have a much higher Non Recurring Engineering(NRE) cost and a much longer turn around time as compared to software. FPGAs provide a middle ground between these two extremes.

Field Programmable Gate Arrays(FPGAs) are reconfigurable devices for implementing digital hardware, quickly and inexpensively. An FPGA can be reprogrammed virtually limitlessly and in a matter of seconds. This capability makes them suitable for various fields where application functionality is expected to change with time, or in fields where the volumes are not large enough to justify the large initial costs associated with producing custom ASICs. FPGAs are also a useful prototyping tool that can be used to implement a high fidelity model of an ASIC's behavior [4].

Figure 2.2 shows the basic structure of an FPGA. The basic main components of an FPGA are:

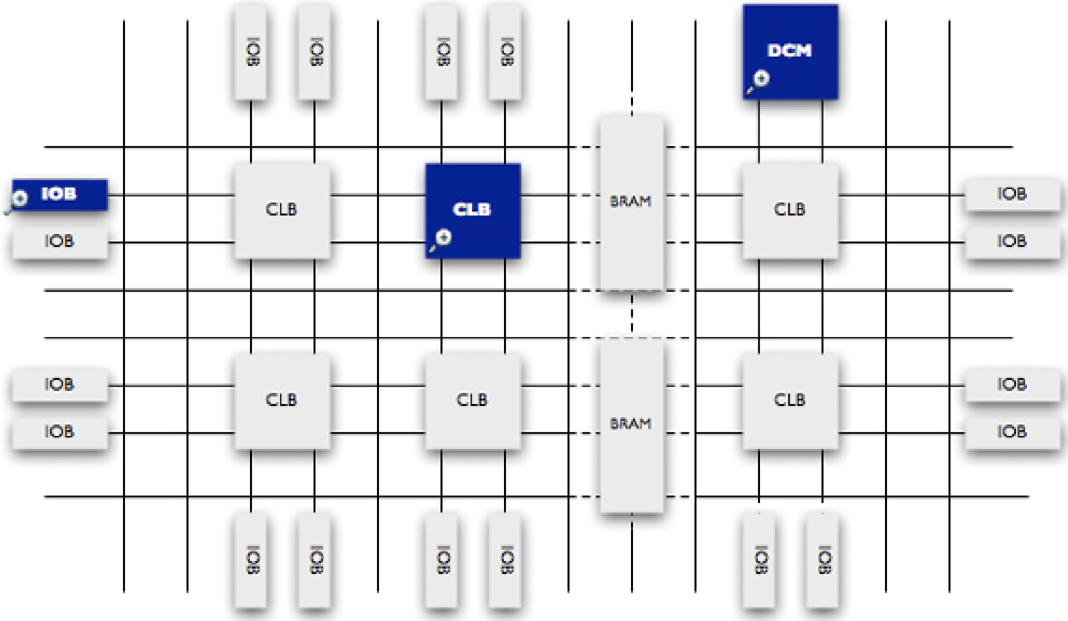


Figure 2.1 Representation of the basic structure of an FPGA. From [5].

- Input/Output Bufers (IOB): These are used for any signals that need to go off chip from the FPGA or need to be brought into the FPGA. These are divided into banks and each bank supports a particular I/O standard.

- Configurable Logic Blocks (CLB): CLBs provide the ability to implement arbitrary digital logic functions on the fabric of the chip. These blocks consist of elements which together can provide an arbitrary combinational logic function. These also contains memory elements so that sequential logic can be implemented.

- Routing: Routing resources provide the link between all the resources present on the FPGA. Each CLB can implement an arbitrary, but small logic function. Routing resources allow the CLBs to be connected to each other to form circuits spanning the entire chip which can implement more complex functions.

Besides these basic elements, moderns FPGAs also contain special elements such as memory arrays, hardware multipliers, hardware PCIe endpoints and hardware Ethernet MACs for performing tasks which are commonly required in applications.

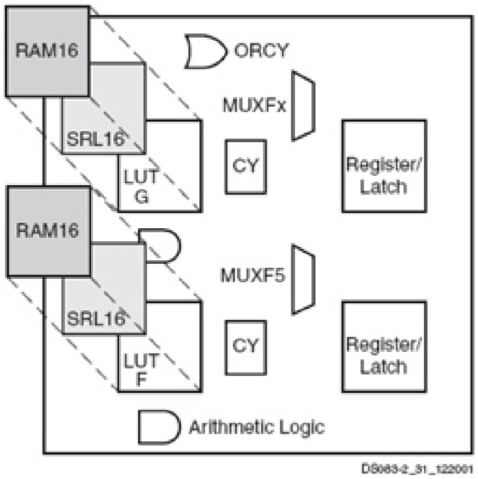


Figure 2.2 Structure of a slice. From [5].

Since the most abundant and used logic resource on an FPGA is the CLB, in this work the focus is on tolerating faults in the CLBs. However, the same logic and method used for tolerating faults in the CLBs could be used to tolerate faults in the other logical elements of the FPGA and routing resources (see Chapter 7).

The proposed method is equally applicable to both Xilinx and Altera FPGAs. However, since the evaluation of the approach has all been done on Xilinx FPGAs, I shall briefly discuss the meaning of a CLB in context of a Xilinx FPGA.

Each CLB in a Xilinx FPGA consist of two slices. Each of these slices are then connected to a switching matrix which connects them to chip-wide routing resources. Each slice consists of four look up tables, four flip-flops, four multiplexers and carry logic. The lookup table (LUT), also known as a “function generator”, is the heart of the FPGA technology. Each of these look up tables output a single bit in response to 4 (more recently 6) input bits. Using these look up tables, an arbitrary 6 input logic function can be implemented. Using the LUTs, the flip flops and the multiplexers as building blocks, more complex functions can be implemented on a slice. These slices then form a CLB which connects to the routing logic of the FPGA.

2.2 About Tool Chains

Besides the developing the architecture of FPGAs, Xilinx and Altera have invested significant time and effort into building software tools that ease the process of implementing designs described in Hardware Description Language (HDLs) on to their chips. The software tools, known as tool chains, assist the engineers in converting the HDL description of a circuit into the corresponding programming and interconnection of CLBs on the FPGA. In this section, I briefly describe the steps involved in implementing designs on an FPGA. This section specifically applies to the Xilinx tool chain, however the steps remain fairly common across FPGA vendors.

2.2.1 Synthesis

Systems implemented on FPGAs are typically first specified in HDLs such as Verilog and VHDL. These languages can describe the design at a much higher level of abstraction than the look up tables or gate level. Synthesis is the first step in implementing a design on to an FPGA, which takes the high level description of the circuit and converts it into technology independent boolean functions. These boolean functions are then minimized and optimized by applying various optimization techniques. A large design may consist of many HDL files, which are individually synthesized into netlists. These netlists are then combined into a single file containing the description of the system in terms of boolean functions. A logical Design Rule Check (DRC) is performed on this representation of the system.

2.2.2 Technology Mapping

The mapping (MAP) phase converts each of the boolean functions into FPGA specific primitives (i.e. lookup tables or an interconnection of lookup tables). The mapping tools are highly device/vendor dependent since they need to know the exact resources available on a chip to efficiently map a design on to the device. In this phase of implementation, all the boolean functions are mapped on the FPGA with no regard to their interactions with each other or timing/performance constraints. Thus, this step will usually succeed unless the design requires more resources (e.g. LUTs) than the device has available.

2.2.3 Place & Route

Place & Route (PAR) is typically the most time consuming part of implementing a design onto an FPGA. The MAP stage simply allocates the LUT representation of the system onto the FPGA without regard to communication delays between various components and the target frequency. The PAR stages takes into account the target frequency of the design, the communication between various parts of the design and moves the parts around the FPGA to try to meet the specified timing requirements. Simulated annealing is a frequently used algorithm in PAR, where the PAR tool begins with a random placement of the design. In every step, it randomly moves different parts of the design around the chip and determines if the change improves or worsens the design. These random changes are initially allowed to move components by large amounts but as iterations increase, the amount a part of the design is allowed to move is successively reduced. This continues until all constraints have been met or until a number of iterations are performed without any improvement to the design.

2.2.4 Configuration File Generation

The behavior of an FPGA is controlled by the way in which resources (e.g. LUTs, interconnects etc.) are configured. The final step in implementation of a design is to convert the placed and routed design into a stream of bits which program each resource available on the FPGA to perform a function. The resulting output is known as a “bitfile”.

2.2.5 Configuration File Deployment

Configuring the FPGA consists of transferring the information contained within the bitfile, generated in the previous step, to the FPGA. The FPGA has configuration pins which allow access to its configuration memory. The bitfile is transmitted into the FPGA over these pins.

CHAPTER 3. RELATED WORK

This chapter is divided into two main parts, both closely related to the presented work. The first discusses previously proposed methods for fault location in FPGAs. The second discusses incorporating fault tolerance in FPGAs. The methods described in the first part can be used to locate faults during the "Test FPGA" phase of the FPGA, discussed in the next chapter. The results of this phase can then be used in future phases of the implementation flow to avoid faults.

3.1 Fault Location Methods

The configurability and inherent parallelism in an FPGA allows for innovative methods of detecting and locating faults. The FPGA can be configured in different ways to detect/locate faults. In their work entitled "Fault Detection and Location of Dynamic Reconfigurable FPGAs", Wu et al. [6] propose one such method in which faults can be detected on an FPGA by programming test circuits on it. They propose using partial dynamic reconfiguration to change the test circuits implemented on different parts of the FPGA, which reduces the amount of time needed to test the entire chip. This is useful in large scale production environments where each chip needs to pass through a testing phase before being approved.

The work "Universal fault diagnosis for lookup table FPGAs" by Inoue et al., [7] proposes another such method for FPGAs, which allows faults to be located up to the granularity of a single CLB. They use the output of each programmed CLB as an input of another CLB. This enables the testing of a number of CLBs by using the same set of I/O pins of the FPGA. By sequentially running this procedure on cascaded rows and then on cascaded columns, they claim to be able to identify individual CLBs that are faulty (e.g., CLBs at the intersection of the faulty row and column).

Wang et al. [8] suggest a method of fault location and also suggest that such a method could allow for faulty chips to be utilized. The method proposed by them is a BIST technique that uses different parts of the FPGA to test other parts of the FPGA. By configuring the FPGA multiple times, the various parts are in turn tested for faults. The PMC model [9] is used to account for potential errors in areas that test other parts.

In “FPGA Interconnect Fault Tolerance”,Campregher [10] discusses the increasing area being occupied by routing resources in modern FPGAs and the consequent need to develop BIST techniques that concentrate on identifying faulty interconnect resources. Although this thesis does not address the problem of faulty interconnect resources, it is an interesting area for further work as discussed in Chapter 7. References [11] and [12] are some papers introducing BIST techniques for fault detection in interconnects of FPGAs.

In “Built in self-test of configurable logic blocks in Virtex 5 FPGAs”, Dutton et al. [13] show a practical implementation of BIST techniques for locating fault on the Virtex 5. They have developed a total of 17 different configurations which together achieve a 100% coverage for logic block faults occuring on the Virtex 5. Similarly reference [14] shows a practical set of test configurations for identify I/O tile faults on a Virtex 5 FPGA.

3.2 Fault Tolerance Methods

Cheatham et al. [15] present an excellent survey of different fault tolerance techniques that have been applied in the field of FPGAs. They have classified the various techniques into two broad types: Device Level and Configuration Level techniques.

3.2.1 Device Level

These techniques are incorporated during the manufacturing stage of a device. For example redundant resources such as routing paths and Programmable Logic Blocks (PLBs) are added. In [16], Hatori proposes a technique that tolerates faults discovered at the manufacturing stage by adding redundant rows and selection logic to routing resources. Faulty rows are made invisible to the user by routing around them.

Kelly [17] proposes a technique that can mask faults even when they occur in the field. An on board router is added which, on the basis of a stored “fault vector”, changes the routing of the design to avoid faulty areas. The advantage of this method is that it remains mostly transparent to the end user software.

Durand [18] proposes a method that can tolerate faults at run time. Extra resources are used to continuously test the LUTs. When a fault is discovered, spare resources are used to mask the fault.

A common drawback of all these techniques is the necessity of additional chip-level or board-level resources to support them. Howard et al. [19] present a method of tolerating faults that works at a higher level of abstraction. Subcircuits of a design are moved from one “block” to another to avoid bad areas. Although this approach does not use extra hardware, due to random movements of subcircuits the modified circuits placement and routing will not be as optimized as using standard place and route tools to re-implement the circuit.

3.2.2 Configuration Level

These techniques incorporate run-time fault tolerance into FPGAs. First they identify faulty areas of the chip. Then, by performing simple shifts of configuration memory, they use alternative resources to implement the design. Methods such as the ones described by Narasimhan [20] and Hanchek [21] rely on shifting the configuration memory, when faults occur, to obtain a fault free implementation. Methods such as the ones described by Emmert et al. [22] try to apply various heuristics to decide the best direction for these shifts. All of these methods have the common drawback of degrading the place and route (PAR) quality of the circuit provided by PAR tools. In other words, given the faulty locations on the chip, there might be more optimized ways to implement the circuit which these methods cannot arrive at. Methods such as the one proposed by Lach et al. [23] rely on grouping FPGA slices into large groupings and using the redundancy within these groupings to maintain an error-free operation. Such methods require all possible configuration files to be generated, which is implausible for large circuits. Emmert et al. [24] present a method for run time fault tolerance by using partial run time reconfiguration capabilities of the FPGA. They require a library of bitfiles avoiding different fault profiles and in case no bitfile is available in the library which avoids the current set of faults, then the method generates one. They also try to use the non-faulty parts of faulty slice so that their method has a higher granularity than a slice.

In summary, current fault tolerance methods try to either mask faults at the manufacturing stage or try to rectify them in the field with (possibly) less optimized implementation decisions than could have been obtained by fully rerunning PAR with standard tools. Though these methods have applications in certain domains, in some other domains where frequent reconfiguration is possible/required, rerunning the tool chain to find optimized implementations is more desirable than the sub optimal results of local changes made by these methods. I propose a zero hardware overhead method that could be used in these situations for fault tolerance.

Current FPGA tool chains do an excellent job of optimizing the way in which a design is implemented. The implementation they arrive at is typically fairly optimized and hence making local changes to this implementation to avoid faults will usually yield degraded results. This work proposes a method that integrates information about faults into standard tool chains thus leveraging their optimization capabilities for fault tolerance.

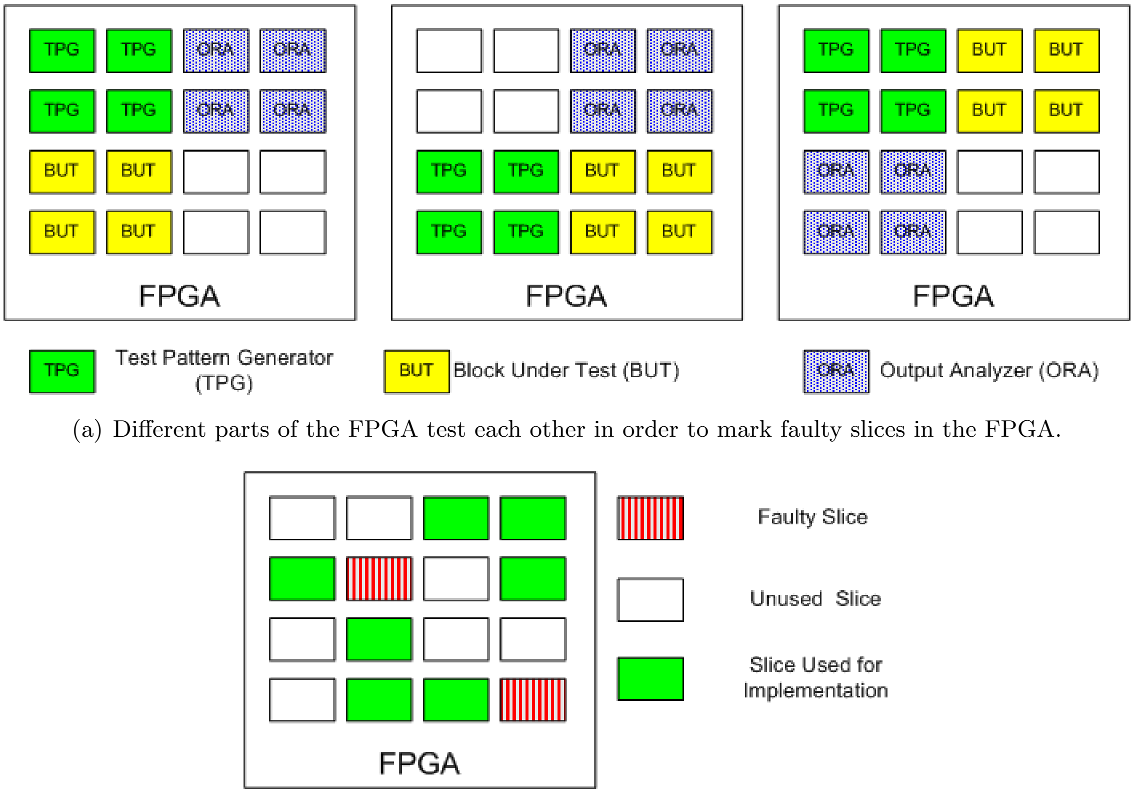
CHAPTER 4. A FAULT AWARE TOOL CHAIN

This chapter provides an overview of the proposed method for integrating fault information into an FPGA implementation tool chain flow. The impact of the proposed approach on the life cycle of an FPGA is discussed, and possible usage models are presented. This is followed by an example of a nomenclature that manufacturers could use to market FPGAs with faults. The chapter ends with a discussion of concerns that need to be considered when using FPGAs with faults.

4.1 Overview

Figure 4.1 succinctly summarizes the essence of this thesis. It shows one of the methods from Chapter 3 being used to identify faulty slices in the FPGA. The lower part of the figure shows how the identified faults can be avoided by using a modified tool chain that is fault aware. Proposing such a tool chain and evaluating its effectiveness is the topic of this thesis.

Figure 4.2 shows a high-level overview of the proposed fault aware FPGA implementation tool chain flow. A design goes through several stages while being implemented on an FPGA. An additional "Test FPGA" stage is proposed here that may use any of the fault location techniques discussed in Chapter 3. The information gained from this test stage can then be fed back into the rest of the tool chain to implement the design in a way that avoids faults. The Synthesis phase involves translating the HDL design description into netlists. This translation tends to take at least a few minutes for any non-trivial design, so the testing of the FPGA can be done in parallel. Since the test stage is performed in parallel with the synthesis step, the additional time overhead to the implementation process will be small (e.g. total testing time of 160 seconds is estimated for a XC4025 device [25]). The information about the faulty areas



(a) Different parts of the FPGA test each other in order to mark faulty slices in the FPGA.

(b) Tools avoid faulty slices while implementing the circuit.

Figure 4.1 Overview of the proposed method.

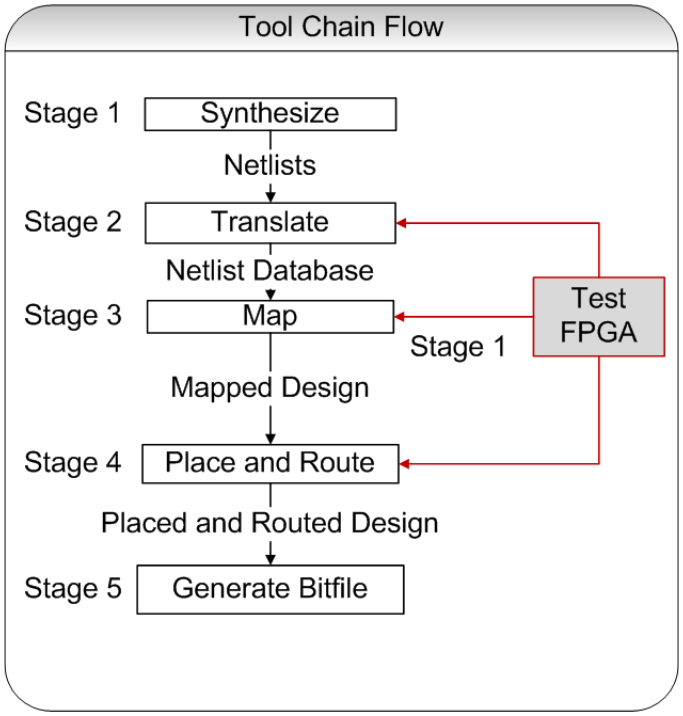


Figure 4.2 Fault tolerant design flow that tests the FPGA, in parallel with synthesis, and feeds this information back to the other stages.

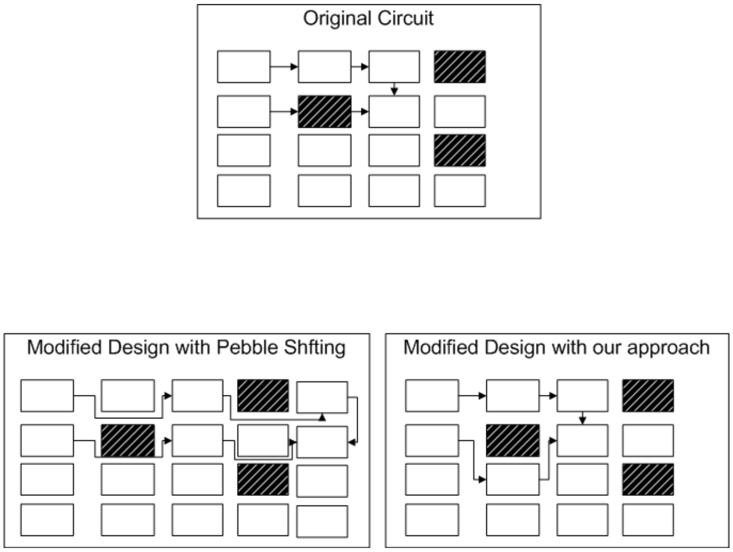

Original Circuit
Modified Design with Pebble Shfting
Modified Design with our approach

Figure 4.3 An example placement obtained by the method proposed here as compared to “Pebble Shifting” [20].

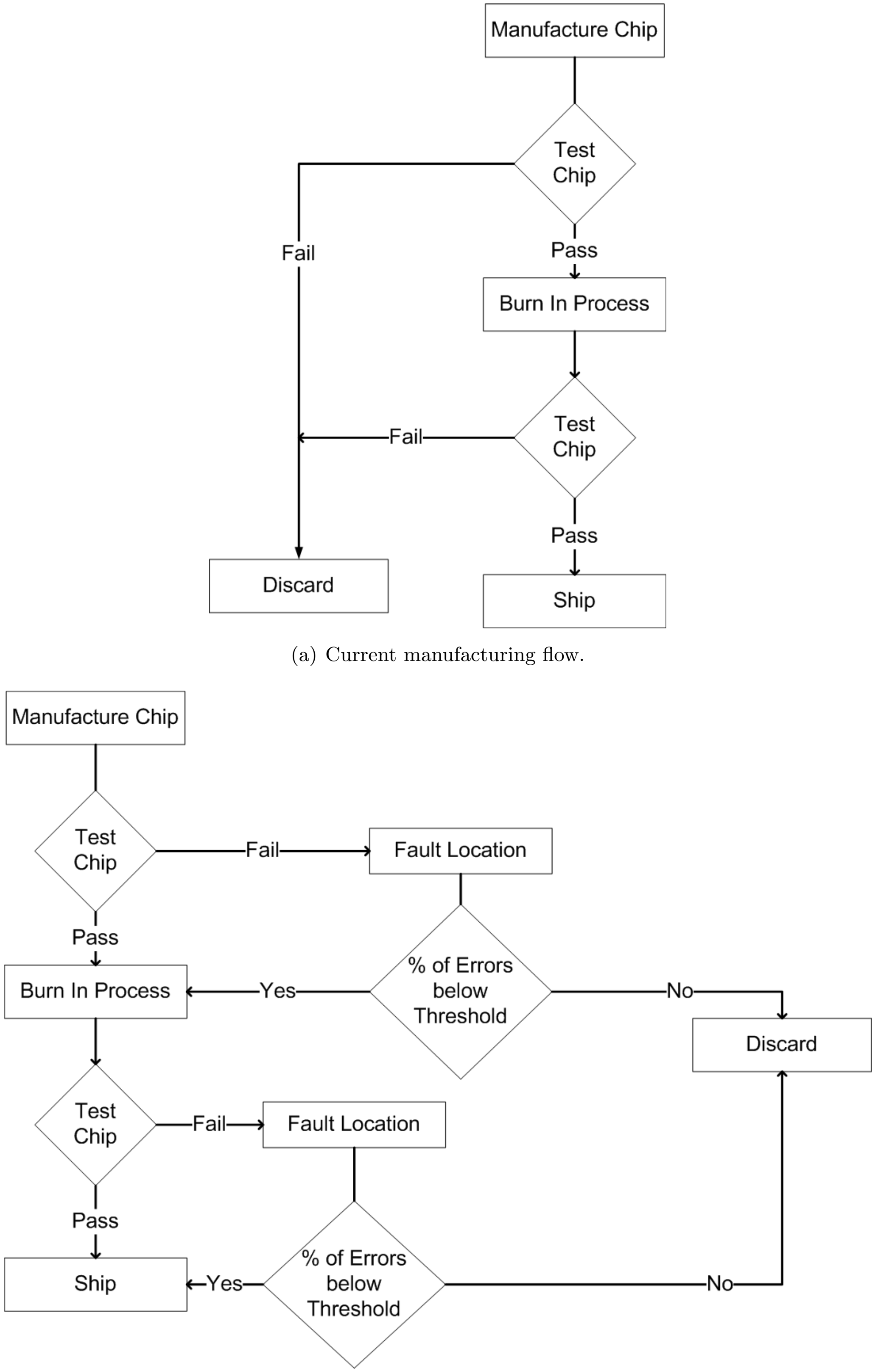
of the chip can be propagated subsequently to allow the tools to implement the design in an efficient manner despite the faults. Thus, as compared to the previous methods discussed in Chapter 3, the proposed method leverages the intelligence of mature place and route tools to provide an efficient implementation of the design. Figure 4.3 gives an example of the placement obtained by the proposed method as compared to one obtained with the "Pebble Shifting" method proposed by Narsimhan et al. [20]. The "Pebble Shifting" method ends up avoiding entire columns even if they have usable slices. The resulting circuit has longer paths than the method proposed here. The method is currently implemented by using the PROHIBIT constraint available in Xilinx tools that allows certain sites to be forbidden to the tools for the purpose of placing components. Although this constraint allows the current tools to support the proposed method without any modifications, a cleaner way of integrating this into the tools is desirable. One simple approach could be to have a separate "error file" that would be consulted during the implementation process.

4.2 Applicability of the Proposed Method

Two main use cases can be considered for the proposed method: 1) using faulty FPGAs to implement new designs; and 2) rectifying FPGAs after a slice fault occurs. These use cases are discussed below.

Consider a company buying faulty FPGAs in order to implement designs required for a product. As opposed to creating a single bitfile for all the FPGAs, the manufacturer would have to run the tool chain for each FPGA individually since each would have errors in different locations. Although this sounds unreasonable at first glance, it might turn out to be economical depending on the savings from buying faulty chips versus the increase in computing costs. In order to support such a use case, the manufacturer would change the manufacturing flow as shown in Figure 4.5. Curve 2 in Figure 4.5 shows the change in the probability of FPGAs being discarded by the manufacturer, if the customer can use such a method.

In the second use scenario, consider a company using fault-free FPGAs to implement a design. These programmed FPGAs are then used in end user products. When an end user



(a) Current manufacturing flow.

(b) Proposed manufacturing flow.

Figure 4.4 Proposed changes in the FPGA manufacturing flow.

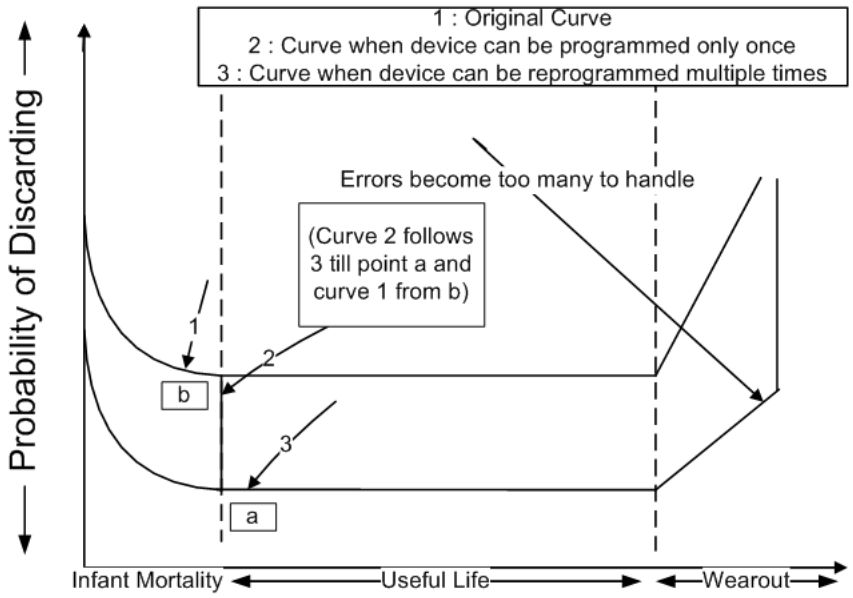

1 : Original Curve
2 : Curve when device can be programmed only once
3 : Curve when device can be reprogrammed multiple times
Errors become too many to handle
(Curve 2 follows 3 till point a and curve 1 from b)
1
2
3
b
a
Probability of Discarding
Infant Mortality
Useful Life
Wearout

Figure 4.5 Changes to the bath tub curve with the method.

product malfunctions because of an error in the FPGA, rather than discarding the FPGA, a technician could run the fault-aware tool chain on the (now) faulty FPGA. This would create a new bitfile of the design that would avoid the faulty areas. Curve 3 in Figure 4.5 shows the conceptual change in the probability of FPGAs being discarded if this approach is used.

4.3 Error Grading

Devices are currently tested after they are manufactured to determine their maximum operating frequency. Due to variances in the manufacturing process, this frequency is not uniform for all chips and hence they are branded with a speed grade to indicate this difference. A similar concept could be used to mark chips with different "Error Grades". For example, the larger the error grade, the more faulty slices in the chip. Also as the error grade increases the maximum frequency of operation will reduce. This is due to the place and route tools having fewer options from which to choose during implementation (see Section 6.2). Thus, either a separate error grade could be constituted or a composite grading system taking both the percentage of errors and the speed into account could be created. Before shipping, the FPGAs could be run through a fault location phase. Then depending on the number of faults detected, the chip could be discarded or branded with an error grade/composite speed-error grade. A consumer could then select a chip based on its error grading, the trade-offs associated with the error grading, and the requirements of the design.

4.4 Some Concerns

Implementing fault tolerance in FPGAs requires certain common obstacles be considered along with issues specific to the nature of the applied approach. In this section both kinds of issues are discussed with respect to the proposed approach.

- *Faulty Routing Resources:* When a certain area of the chip becomes faulty it is likely that the routing resources in that area have become faulty as well. Like many approaches mentioned in Chapter 3, the approach only concentrates on avoiding faulty slices. There are "Device Level" approaches that can tolerate faults in routing resources. There are also fault location methods that can identify faulty routing resources. Such information would have to be integrated into the tool flow in order to make the approach practical. Some possible ideas are discussed in Chapter 7

- *Other Faulty Resources:* Many modern FPGAs have other resources such as Block RAMs, DSP Slices, Clock Management Tiles, hardcore microprocessors, etc. The current approach does not consider fault tolerance for these resources. Extending this work to examine these other resources is possible by using the same logic as used for slices. Chapter 7 discusses some preliminary thoughts on this.

- *Location Constraints:* In some FPGA designs certain logic components are required to be locked to a specific location (LOC), or must be placed at a location relative to another component (RLOC). In the first case, if the location to which the component is LOCed is faulty, then there is nothing this approach or any other approach can do to allow the FPGA to be used for this design. In the latter case, it might be possible to move the RLOC origin to another point to satisfy the constraints. However the operating frequency may be lower than in a design with no RLOC constraints.

- *Mass Production Viability:* When FPGAs are a part of a mass-produced product, using faulty FPGAs would require re-running the tools for each individual FPGA. The amount of time and resources spent on re-running the tools for each FPGA could be offset by the reduced bill of materials resulting from using cheaper, faulty FPGAs.

CHAPTER 5. EVALUATION METHODOLOGY

This section discusses the experimental validation/evaluation of the proposed method. Section 5.1 describes the fault model assumed by this work. Section 5.2 describes the testing methodology and Section 5.3 discusses the benchmark circuits and the device used for these experiments.

5.1 Fault Model

One way to classify faults is based on their effect duration. Soft Errors (SE) are faults that are temporary and cause a circuit to malfunction temporarily. However, the circuit resumes correct operation after a reset. These faults are also known as transient faults. Hard Errors (HE), on the other hand, are faults that cause sections of an IC to permanently malfunction. There can be various causes for such errors, e.g., manufacturing defects, aging, ephemeral high current in a circuit at some point. The proposed method is meant to handle Hard Errors.

Another way to classify faults is based on their origin. In general, this classification method divides faults in to two types: *manufacturing faults* and *faults due to aging*. Manufacturing faults can have many causes such as bad design, bad manufacturing practices, complexity of the lithography process, immature production processes, particles, etc. Faults that occur due to aging, on the other hand, can have different causes as well, such as long term radiation exposure and wearing out of materials. For the purpose of this work, the cause of the fault is not important.

As noted in Section 2.1, modern FPGAs have many different types of logic resources such as slices, multipliers, and hard MACs. They also have interconnects to allow these resources to be combined into circuits that function as required. Faults can occur either in the intercon-

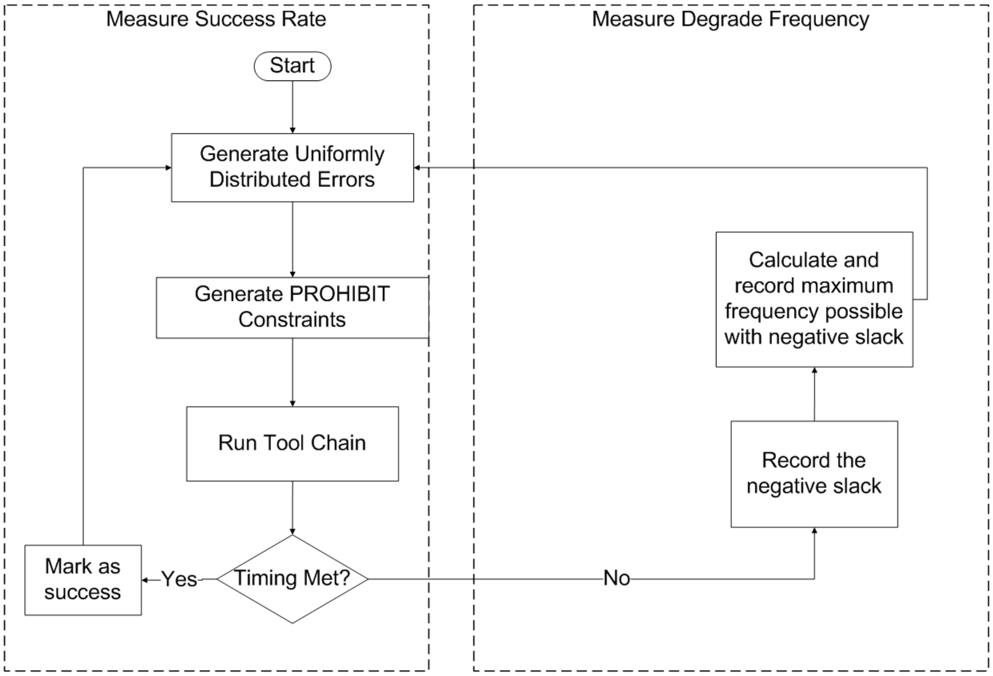

Measure Success Rate
Measure Degrade Frequency
Start
Generate Uniformly Distributed Errors
Generate PROHIBIT Constraints
Run Tool Chain
Timing Met?
Yes
Mark as success
No
Record the negative slack
Calculate and record maximum frequency possible with negative slack

Figure 5.1 The testing flow used to evaluate the proposed method.

nects or in the logic elements. In this work, the proposed method is evaluated only for slice faults. The method can be applied without changes to handle faults in any of the other logic elements. Chapter 7 discusses some work currently in progress to extend the fault tolerance to interconnect faults as well.

The location the faults on a chip also depends on the reason/source of those faults. In this work, the assumption is that faults are randomly distributed across the entire chip.

5.2 Methodology

This section describes the methodology used for testing, validation and qualitative analysis of the proposed method. Section 5.2.1 describes the testing method in greater detail. It also describes how the method was implemented in the current form of the tool chain. Section 5.2.2 describes further experiments to study the effect of loosening the timing constraints of the designs on the success ratio of the proposed method. Finally, Section 5.2.3 gives a description of an experiment to investigate if it is possible to establish an equivalence between larger faulty chips and smaller non-faulty chips.

5.2.1 Error Tolerance

In order to evaluate the proposed method, I ran simulations by varying the numbers of faulty slices in the chip. These slices were then uniformly distributed across the area of the chip by marking them as faulty in an array representing each slice on the chip. PROHIBIT constraints were then generated to restrict the tools from placing any components in the faulty areas. The tools were then run again on the synthesized netlists to obtain the placed and routed designs. Next, PAR design was analyzed with a timing analysis tool "trce" to check that it was still able to meet timing. If it did meet timing, it was marked as a success for the proposed method. If it did not meet timing, it was marked as a failure for the proposed method. Figure 5.1 shows the flowchart of the testing methodology. 100 iterations of the entire tool chain flow were run for each fault level. In order to accomplish this task, a total of 75 computers were used for almost 2 full days. These experiments are meant to investigate the number of faults that the presented approach can tolerate on FPGAs. I discuss the observations on this metric as well as how it relates to chip yields in Sections 6.1 and 6.2.

5.2.2 Performance Degradation

For designs that fail to place and route during the test runs described in section 5.2.1, the negative slack time in the PAR report was recorded. This slack was then used to calculate the maximum frequency at which that design could have been implemented by the tools (see right side of Figure 5.1). These results are used to show that even in cases where it is not possible to implement the design with the specified constraints, a small performance decrease may be enough to make the design implementable on the faulty chip. Thus, in addition to having fewer resources, a faulty chip may also degrade in performance. This metric is used to investigate a threshold for the error rates that can be tolerated with the proposed method in Section 6.2.

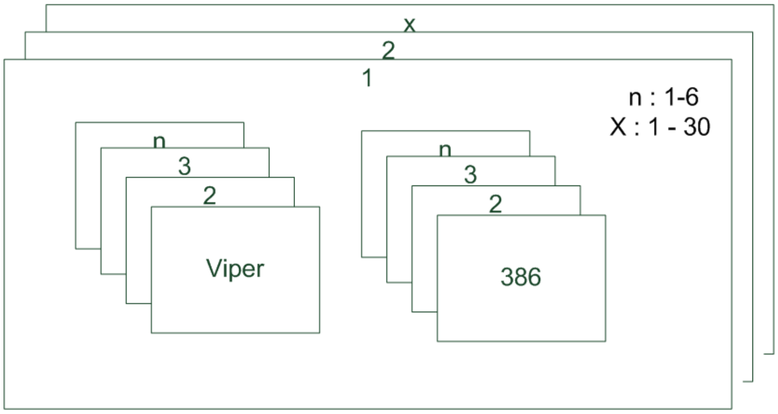


Figure 5.2 Test circuits used in the evaluation experiments for the presented approach.

5.2.3 Comparison with smaller, fault-free chips

The most dominant logic resource in an FPGA are the logic slices. If some of these slices are faulty and hence unavailable for use, a larger FPGA could possibly be thought of as a smaller FPGA. In order to understand the trade-offs involved with this, I implemented benchmarks (described in Section 5.3) on smaller chips and investigated the maximum frequency at which they could be implemented on those chips. These frequencies were then compared to the frequencies possible on a larger reference FPGA (i.e., LX110T) with various error rates. The findings of these experiments are discussed in the Section 6.3

5.3 Circuit/Device Description

Some of the benchmark circuits proposed by Corno et al. [26] were used to verify and validate the proposed approach. These benchmarks were originally meant to evaluate test sequence generation methods for identifying stuck at faults in circuits. These benchmarks were easily scaled to occupy various percentages of the FPGA by replicating the design. These designs are real world circuits, thus they form good test cases to evaluate the proposed method. The benchmarks in [26] were replicated multiple times to get utilizations of approximately 25%, 50% and 75% (See Figure 5.2). The test circuits chosen from among the benchmarks were:

- b17: Three subsets of an Intel 80386 processor.

- b18: Three Viper processors and Six 80386 processors.
- b20: A Viper processor and a modified version of the Viper processor.
- b21: Two Viper processors.
- b22: A Viper processor and two modified versions of the Viper processor.

The benchmarks did not include a constraint file and hence constraints had to be provided for the evaluation. I constrained a few of the I/O ports of the design to specific pins on the device. A feature of these benchmarks is that they use a single clock. This clock was constrained to within 3% of the highest frequency that the tools could successfully place and route a benchmark on a non faulty FPGA. When the benchmarks were replicated, their outputs were ORed together to form the output of the chip in order to have some interconnection between all parts of the circuits.

The Xilinx Virtex-5 LX 110T [5] was used for deploying the test circuits. This device has a total of 17280 slices. The reason this device was chosen is that there are 4 smaller devices available in the same family. This allowed experiments to be run not only for error tolerance, but also allowed running experiments to compare performance with smaller FPGAs, as mentioned in Section 5.2.3.

CHAPTER 6. RESULTS & ANALYSIS

This chapter presents the findings from the experiments described in the previous chapter. Section 6.1 describes the findings on error tolerance obtained by using the proposed method. Section 6.2 then takes into account the percentage of errors that can be tolerated if the frequency is allowed to degrade by a small amount. Section 6.3 presents preliminary findings on the proposed idea of using larger faulty chips instead of small non-faulty FPGAs. Finally, Section 6.4 investigates the sensitivity of the proposed method to some factors like utilization and timing constraints.

6.1 Error Tolerance

In this chapter, experiments were run 100 times for each fault level. This resulted in the plotted data having confidence intervals within 10% at a confidence level of 90%.

Figure 6.1 shows the success rate of the fault aware tool chain at 25% utilization of the LX110T FPGA. As the figure shows, even with almost 75% of the chip empty, the designs have a significant chance of not meeting timing if approximately 10% of the slices are faulty. The success rate is 0% when 30% to 60% of the slices are faulty. This can be explained by the very tightly constrained clock in the original design. The tools were able to barely meet the timing constraints without any faults, so as faults were introduced it became more difficult for the tools to implement the design while still meeting all constraints.

Figure 6.2 shows the success rate when 50% of the FPGA is being utilized. It can be seen that the designs start failing timing much sooner for the higher 50% utilization case as compared to the 25% utilization case. The success rates reach 0% when 20-30% of the slices become faulty.

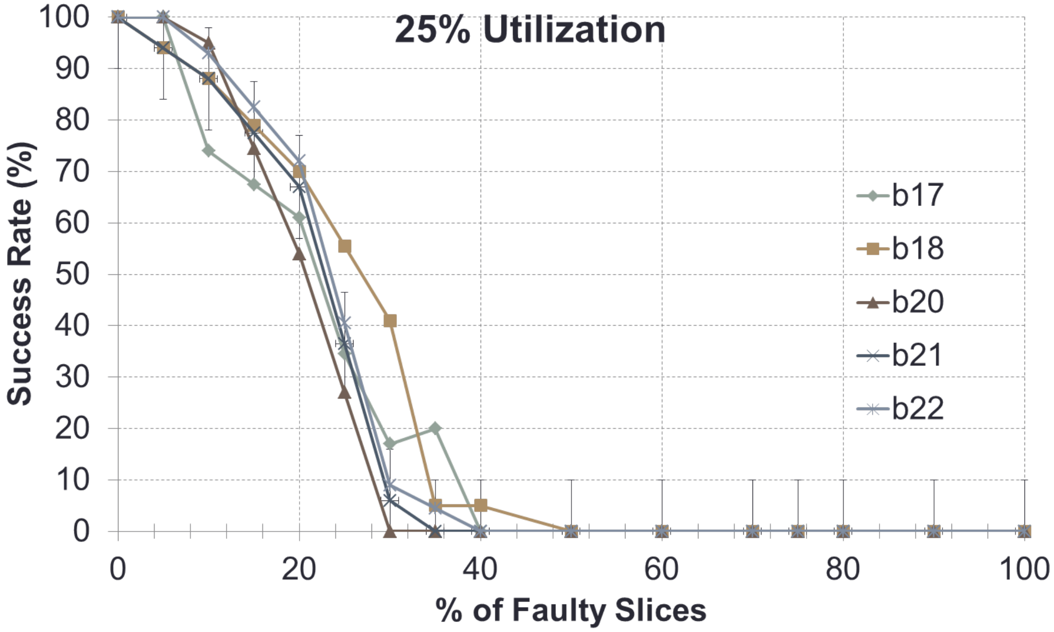


Figure 6.1 Success rate when varying error rates for a device utilization of 25%. All points within 10% confidence interval at 90% confidence level. Error bars shown only on one benchmark for sake of clarity of the figure.

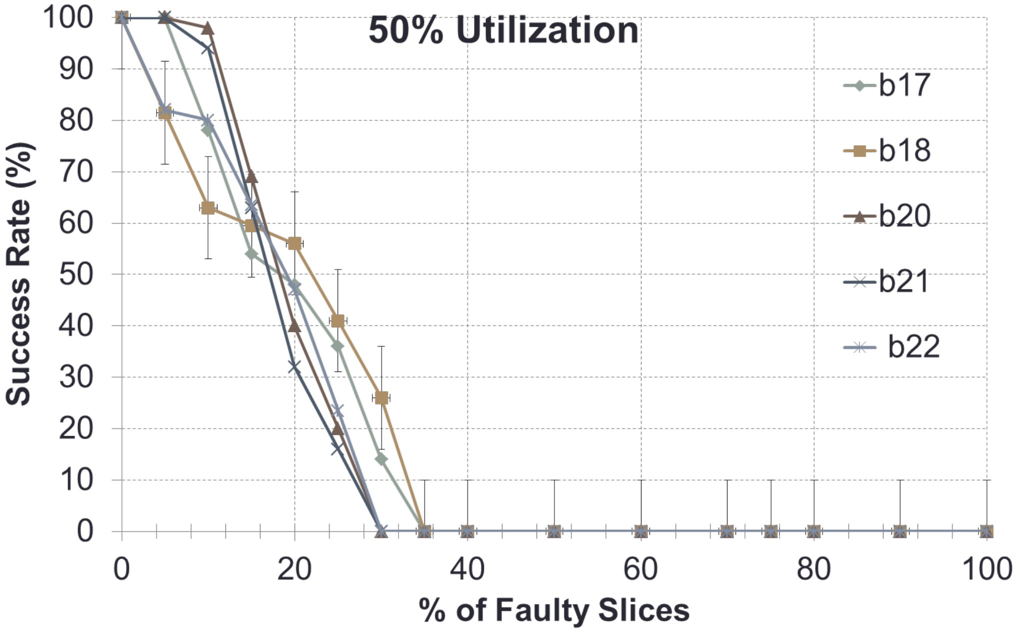


Figure 6.2 Success rate when varying error rates for a device utilization of 50%. All points within 10% confidence interval at 90% confidence level. Error bars shown only on one benchmark for sake of clarity of the figure.

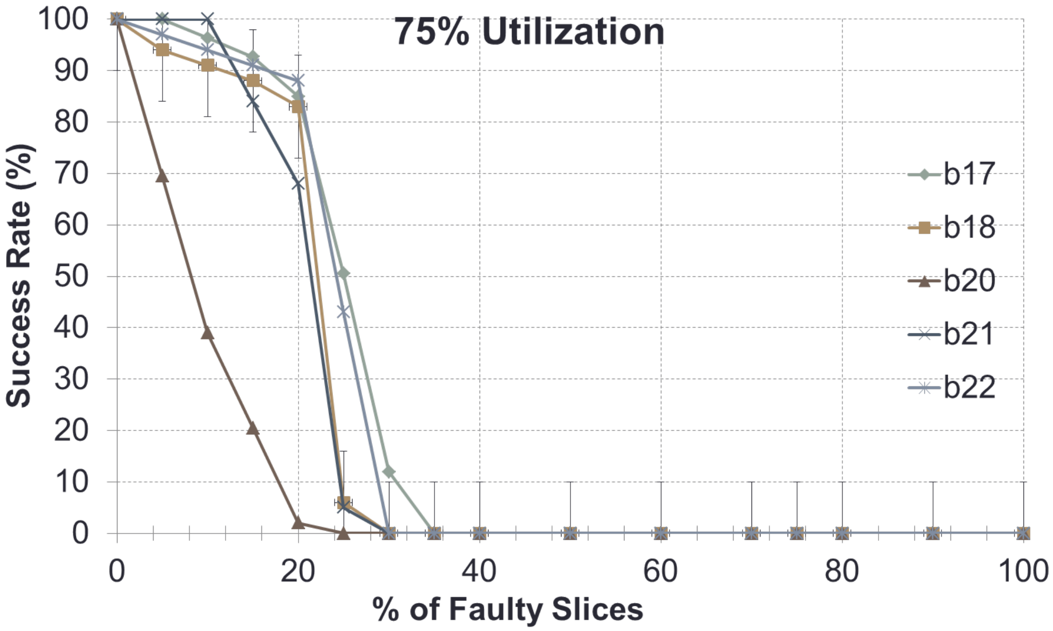


Figure 6.3 Success rate when varying error rates for a device utilization of 75% . All points within 10% confidence interval at 90% confidence level. Error bars shown only on one benchmark for sake of clarity of the figure.

Figure 6.3 shows the success rate for 75% utilization of the FPGA. For the b17 benchmark, it was not possible to achieve an exact 75% utilization so it's utilization was kept at 70%. Thus, the b17 benchmark success rate does not fall to 0% at 25% errors, instead it does so at 30% errors.

The designs experimented with were constrained to within 0.2 ns of the smallest possible period, so these numbers represent close to the worst case success rates of the proposed approach. In many practical cases, a design will not be required to be run at the highest achievable frequency. Thus, if a design is more loosely constrained, then it would be expected that the fault aware tool chain would be able to tolerate more faults.

From the figures it can be seen that the succcess rates are high until about 10% of the slices become faulty. Thus even for designs that cannot compromise on their frequency, it might still be cost effective to buy larger chips with errors present and discard the small percentage of them that cannot meet timing. On the other hand, if frequency degradation is acceptable, then it may be possible to tolerate a given fault level at the cost of performance. The next

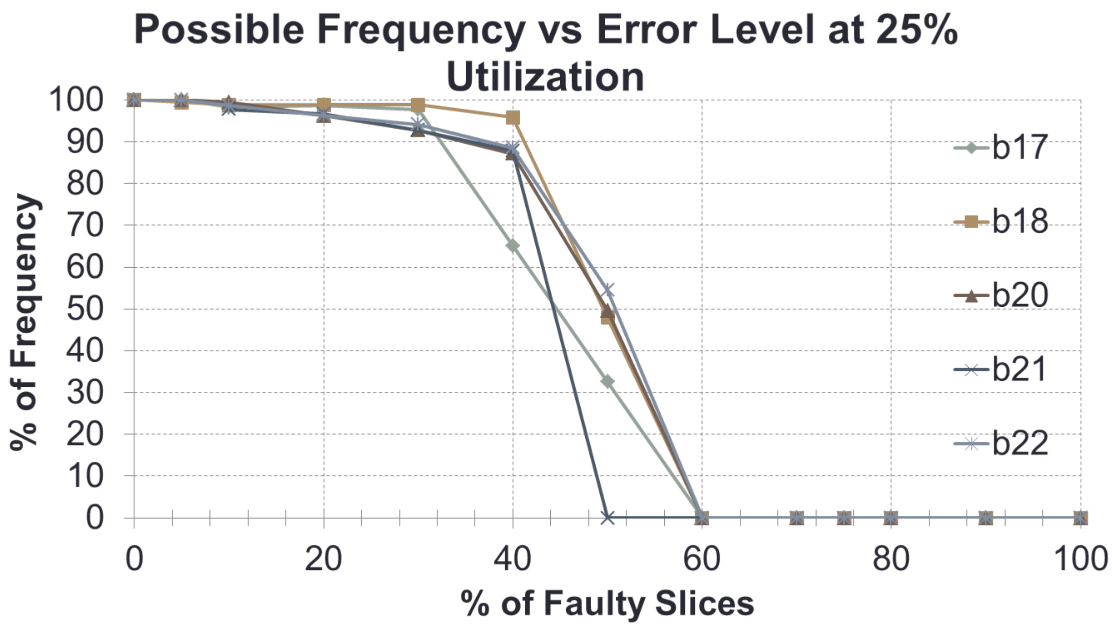


Figure 6.4 Degradation in frequency required for failing designs for a device utilization of 25%. All points within 7% confidence interval at 90% confidence level.

section examines the amount of frequency reduction necessary to implement the experimental runs that failed at various error levels.

6.2 Fault Tolerance when Degraded Performance is Permitted

Table 6.1 Percent of Faulty Slices That Can be Tolerated with a Maximum cost of 10% Performance.

Utilizations	25%	50%	75%
% of Faulty Slices	30%	30%	20%

When the timing constraints could not be met for a given run of a design, it was still possible to implement the design at a degraded frequency. The negative slack times reported by all designs that failed at various error levels were used to calculate the average percentage of the original frequency for which the designs would still meet timing. This section presents the results of that analysis. Table 6.1 summarizes the key observation obtained from this evaluation that gives evidence that the proposed approach is quite tolerant of FPGA slice

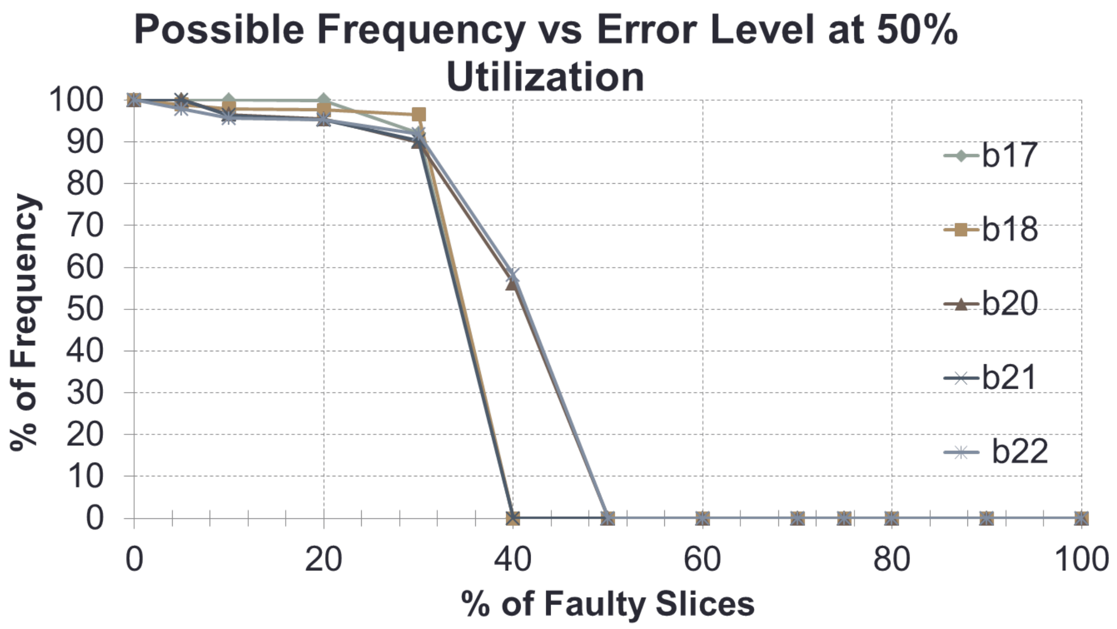


Figure 6.5 Degradation in frequency required for failing designs for a device utilization of 50%. All points within 7% confidence interval at 90% confidence level.

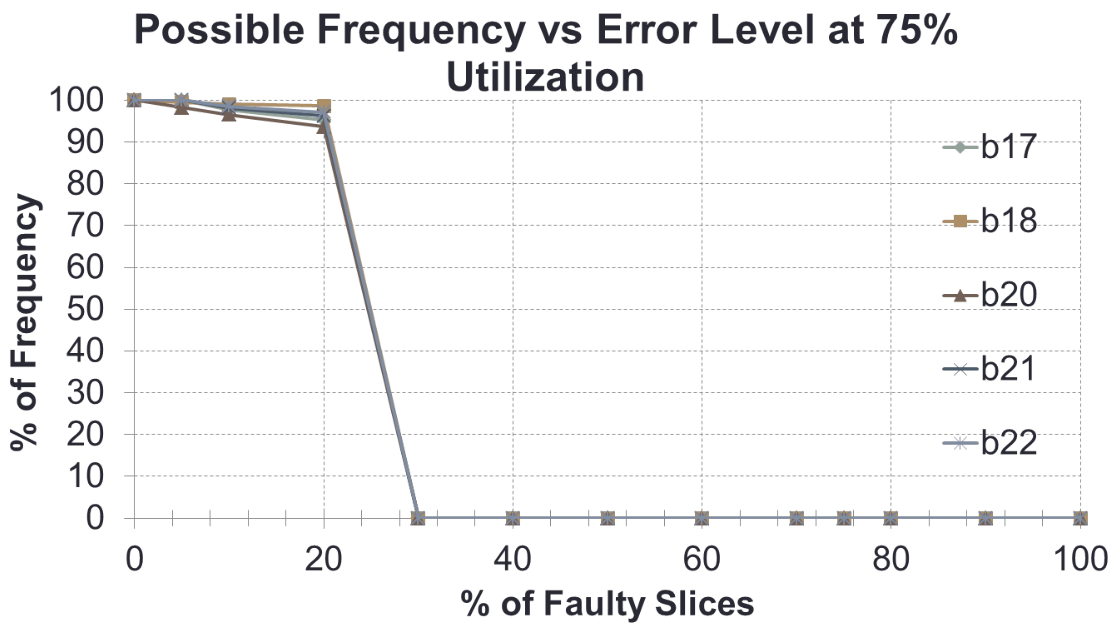


Figure 6.6 Degradation in frequency required for failing designs for a device utilization of 75%. All points within 7% confidence interval at 90% confidence level.

faults. It shows that designs using 25%, 50% and 75% of the chip can tolerate 30%, 30% and 20% slice faults respectively.

The success rate of the proposed approach for a given percentage of errors can be seen in Figures 6.1, 6.2 and 6.3. Figures 6.4, 6.5 and 6.6 must be looked at in the context of Figures 6.1, 6.2 and 6.3 respectively. For example, if for a utilization of 25% and 20% slice errors the success rate in Figure 6.1 is 70%, then 30% of the runs fail at the original frequency. It is these runs that require the frequency degradation shown in Figure 6.4.

As can be seen from Figure 6.4 and Figure 6.1, at a 25% utilization and up to 15% errors, the designs can be implemented in 67.5%-82.5% of the cases. In the cases when this is not possible, they still can be implemented after about a 1% degradation in performance, until about 15% of the slices are faulty. From 15% to 30% errors, all the designs can be implemented with a frequency degradation of between 2%-7%. Thus, the empirical evidence suggests that for a design that utilizes only 25% of the chip, a fault aware tool chain approach can tolerate up to 30% of the chip being faulty at a reasonable performance cost (e.g., less than the performance cost typically associated with stepping down by a speed grade).

Similarly, from Figure 6.2 and Figure 6.5 it can be seen that between 32%-56% of the designs successfully meet timing until about 20% of the slices are faulty. Those that fail can meet timing with a 8%-10% frequency degradation, until about 30% of the slices are faulty. Thus, the experimental results suggest that for a design that utilizes 50% of the chip, the proposed approach can tolerate up to 30% of the chip being faulty at a reasonable performance cost.

From Figure 6.3 and 6.6 it can be seen that there is a knee point at an error level of 20%-25%, below which approximately 90% of the time the designs are successfully implemented. For the remaining 10% of the cases that the designs fail timing, a 3%-5% frequency degradation allows them to be implemented. This knee point varies with utilization. In these graphs the knee point occurs when about 20% of the FPGA is made faulty in the worst case (i.e., at 75% utilization). If the design being implemented is smaller, then up to 30% of the FPGA slices being faulty can be tolerated by the proposed approach for a maximum performance cost of

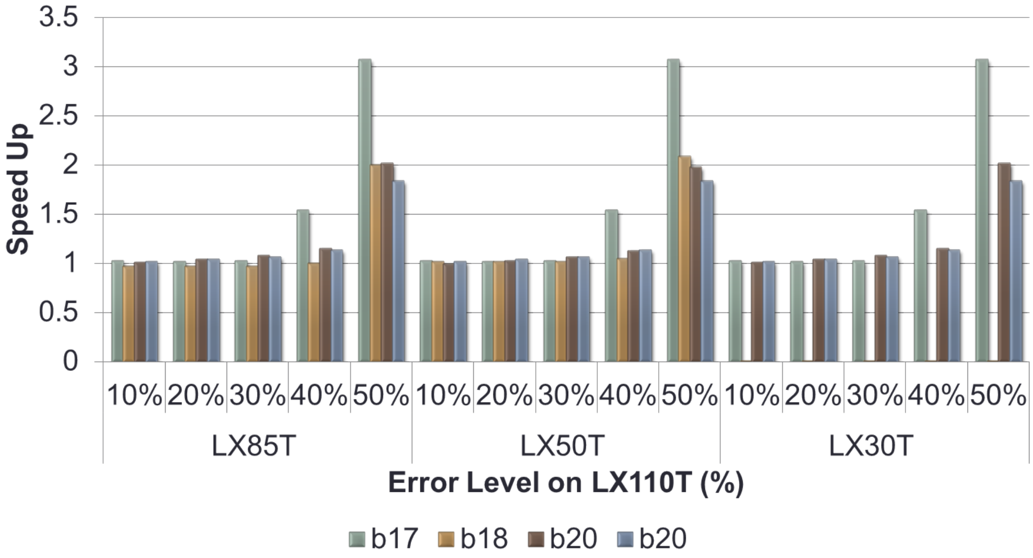


Figure 6.7 Frequency at which the different benchmarks could have been implemented on smaller chips as compared to the maximum frequencies possible at various error rates on the larger chip.

10% (see Table 6.1).

If a manufacturer were to choose an error threshold for which any chip below this threshold was deemed appropriate for market, then the manufacturer's effective yields would increase to that given in Equation 6.1. Where RBT (Rejected Below Threshold) is the % of originally rejected chips that have an error level below a set threshold.

$$Effective\_Yield = Old\_Yield + \frac{(100 - Old\_Yield) * (RBT)}{100} \tag{6.1}$$

6.3 Comparison with smaller, fault-free chips

Figure 6.7 shows the ratio of the achieved frequency at each error rate in the larger chip with respect to the frequency achieved in a smaller, fault-free FPGA. The tests were run for benchmarks that utilized 25% of the LX 110T. Thus, a ratio of 1 indicates the same performance. As can be seen from the figure, up to 30% errors in the larger chip result in approximately the same performance as a smaller fault free chip. Thus, provided enough LUTs are available a design that fits a smaller chip can be implemented on a larger chip with errors such that the fault-free slices are approximately the same.

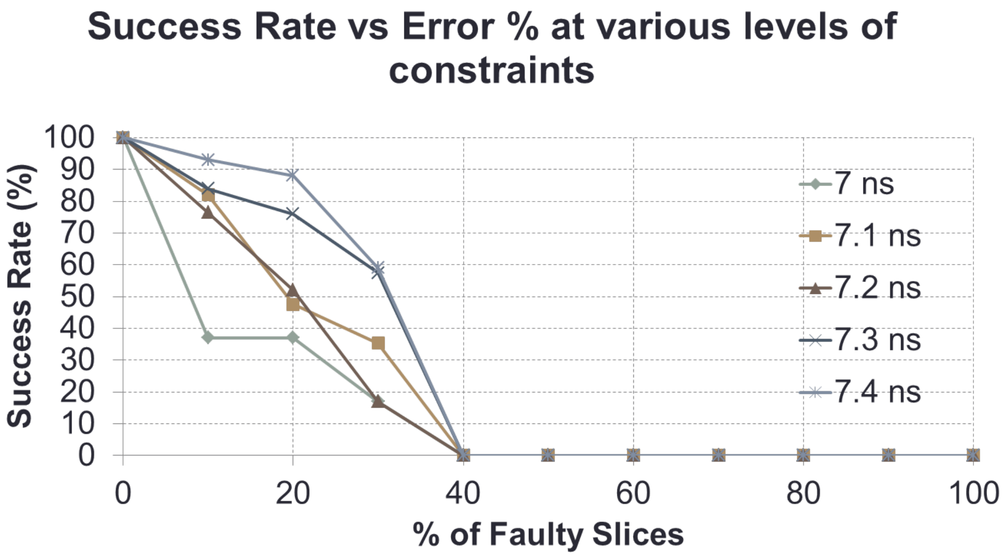


Figure 6.8 Success rate at various error percentages with different timing constraints in nanoseconds.

6.4 Sensitivity Analysis

As mentioned in the earlier sections, all the benchmark designs were constrained to within 0.2ns of their minimum periods. Thus, all results represent the worst case scenarios for those specific benchmarks. However, the success rates were found to vary a lot depending on how tightly the design was constrained. For example the b17 benchmark was tested with a utilization of 25% by constraining it at various periods, the minimum possible being 7 ns. Figure 6.8 shows the findings for this experiment. As seen in the figure, the method is very sensitive to the frequency constraints. Thus, if a design were to be constrained at a frequency lower than the maximum possible, then the success rates for the proposed approach would be much higher than otherwise possible.

As can be seen in Figures 6.1, 6.2 and 6.3, although the success rates are high for all utilization until about 20%, beyond that the rate at which they decrease varies with the utilization. Thus, the proposed method is also sensitive to the percent utilization of the chip.

CHAPTER 7. CONCLUSION & FUTURE WORK

This chapter concludes this thesis and discusses some future work currently under progress as well as other areas that might be interesting to investigate.

7.1 Conclusion

This thesis proposed the idea of a fault aware tool chain that tests and identifies faulty areas of an FPGA. It proposed that this information be fed back into the tool chain to implement around those faulty areas. It then evaluated the approach by sweeping through fault rates from 0% to 100% and utilizations of 25%, 50% and 75% of the FPGA. It also evaluated the approach when slight reductions in frequency are acceptable and investigated sensitivity of the method to various factors. These experiments indicate that the method can tolerate up to 30% slice faults on the Virtex 5 LX110T with a modest decrease in operating frequency (up to 10%). Finally, it proposed the idea of establishing an equivalence between larger, faulty FPGA and smaller, non-faulty FPGAs.

7.2 Future Work

This thesis investigated a limited subset of fault models that are possible on an FPGA. This section explores some interesting ways in which this work could be extended.

7.2.1 Handling faults in other logic resources

This thesis focused on demonstrating and evaluating the proposed method with respect to only slice faults. There are many other logic resources on an FPGA such as multipliers, memory arrays, DSP slices, etc. Any of these is susceptible to faults as well. BIST type techniques suggested for fault location in slices (such as those in discussed Chapter 3) could be extended to identify faults in other logic resources. If these techniques are available, one could use the PROHIBIT statement to avoid using any particular logic resources. In other words, if a fault location technique is present for other logic resources, the method put forward in this work can easily be extended to support those faults. It would be interesting to qualitatively study the effectiveness of the proposed method on handling faults in other logic resources.

7.2.2 Handling faults in interconnects

Along with slices, the largest area of an FPGA is covered by interconnects. Interconnects allow different logic resources to be arbitrarily connected to each other to perform a particular function together. Different BIST techniques have been proposed to identify interconnect faults in an FPGA (see Chapter 3). It would be interesting to investigate if there is a way to extend this work to tolerate interconnect faults as well. A preliminary idea is to create an area group for the faulty area and then declare that area group closed. This, in theory, should stop any routing paths within an area from being used.

7.2.3 Evaluation of these approaches with different clustering parameters

This work assumed a uniform distribution of faults. It might be interesting to compare this model to the real world observations about the clustering factor found in real world fabrication processes. It would also be interesting to evaluate this method with different clustering factors to understand the effects of clustered faults on the efficiency of the method.

Bibliography

[1] G. Moore, “Progress in digital integrated electronics,” in *Proceedings of the 1975 International Electronic Devices Meeting*, vol. 21, Washington, DC, 1975, pp. 11 – 13.

[2] B. Agarwal, “Lecture notes,” in *http://www.ece.rutgers.edu/ bushnell/COURSE/lec4.ppt*, Jan. 2001.

[3] A. Gupte and P. Jones, “An evaluation of a slice fault aware tool chain,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, Dresden, Germany, 2010, pp. 1803–1808.

[4] X. Bing and C. Charoensak, “Rapid FPGA prototyping of Gabor-wavelet transform for applications in motion detection,” in *Proceedings of the 7th International Conference on Control, Automation, Robotics and Vision*, vol. 3, Singapore, 2002, pp. 1653 – 1657.

[5] *Virtex-5 FPGA Datasheet*, Xilinx Inc.

[6] C.-F. Wu and C.-W. Wu, “Fault detection and location of dynamic reconfigurable FPGAs,” in *Proceedings of the International Symposium on VLSI Technology, Systems, and Applications*, Taipei, Taiwan, 1999, pp. 215 –218.

[7] T. Inoue, S. Miyazaki, and H. Fujiwara, “Universal fault diagnosis for lookup table FPGAs,” *IEEE Design and Test of Computers*, vol. 15, no. 1, pp. 39 –44, 1998.

[8] S.-J. Wang and T.-M. Tsai, “Test and diagnosis of faulty logic blocks in FPGAs,” in *Proceedings of the IEEE International Conference on Computers and Digital Techniques*, vol. 146, no. 2, San Jose, CA, mar 1999, pp. 100 –106.

[9] F. P. Preparata, G. Metze, and R. T. Chien, “On the connection assignment problem of diagnosable systems,” *IEEE Transactions on Electronic Computers*, vol. EC-16, no. 6, pp. 848 –854, 1967.

[10] N. Campregher, “FPGA interconnect fault tolerance,” in *Proceedings of the International Conference on Field Programmable Logic and Applications*, Copenhagen, Denmark, 2005, pp. 725 – 726.

[11] C. Stroud, J. Nall, M. Lashinsky, and M. Abramovici, “BIST-based diagnosis of FPGA interconnect,” in *Proceedings of the International Test Conference*, Baltimore, MD, 2002, pp. 618 – 627.

[12] J. Liu and S. Simmons, “BIST-diagnosis of interconnect fault locations in FPGAs,” in *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, vol. 1, Montreal, Canada, May 2003, pp. 207 – 210 vol.1.

[13] B. Dutton and C. Stroud, “Built-In Self-Test of configurable logic blocks in Virtex-5 FPGAs,” in *Proceedings of the 41st Southeastern Symposium on System Theory*, Tullahoma, TN, 2009, pp. 230 –234.

[14] ——, “Built-In Self-Test of programmable input/output tiles in Virtex-5 FPGAs,” in *Proceedings of the 41st Southeastern Symposium on System Theory*, Tullahoma, TN, 2009, pp. 235 –239.

[15] J. A. Cheatham, J. M. Emmert, and S. Baumgart, “A survey of fault tolerant methodologies for FPGAs,” *ACM Transaction on Design Automation of Electronic Systems*, vol. 11, pp. 501–533, April 2006.

[16] F. Hatori, T. Sakurai, K. Nogami, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, H. Muroga, A. Tanaka, and K. Kanzaki, “Introducing redundancy in Field Programmable Gate Arrays,” in *Proceedings of the IEEE Custom Integrated Circuits Conference*, San Diego, CA, May 1993, pp. 7.1.1 –7.1.4.

[17] J. Kelly and P. Ivey, “Defect tolerant SRAM based FPGAs,” in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Cambridge, MA, Oct. 1994, pp. 479 –482.

[18] S. Durand and C. Piguet, “FPGA with self-repair capabilities.” in *Proceedings of the ACM International Workshop on Field Programmable Gate Arrays.*, New York, 1994.

[19] N. Howard, A. Tyrrell, and N. Allinson, “The yield enhancement of Field Programmable Gate Arrays,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 1, pp. 115 –123, Mar. 1994.

[20] J. Narasimham, K. Nakajima, C. Rim, and A. Dahbura, “Yield enhancement of programmable ASIC arrays by reconfiguration of circuit placements,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 8, pp. 976 –986, Aug. 1994.

[21] F. Hanchek and S. Dutt, “Design methodologies for tolerating cell and interconnect faults in FPGAs,” in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Austin, TX, Oct. 1996, pp. 326 –331.

[22] J. M. Emmert and D. Bhatia, “Partial reconfiguration of FPGA mapped designs with applications to fault tolerance and yield enhancement,” in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, London, UK, 1997, pp. 141–150.

[23] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, “Efficiently supporting fault-tolerance in FPGA,” in *Proceedings ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, Monterey, CA, 1998, pp. 105–115.

[24] J. Emmert, C. Stroud, and M. Abramovici, “Online fault tolerance for FPGA logic blocks,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 2, pp. 216 –226, 2007.

[25] N. Itazaki, F. Matsuki, Y. Matsumoto, and K. Kinoshita, “Built-In Self-Test for multiple CLB faults of a LUT type FPGA,” in *Proceedings of Seventh Asian Test Symposium*, Singapore City, Singapore, Dec. 1998, pp. 272 –277.

[26] F. Corno, M. Reorda, and G. Squillero, “RT-level ITC’99 benchmarks and first ATPG results,” *IEEE Design and Test of Computers*, vol. 17, no. 3, pp. 44 –53, 2000.