A Fault-Aware Toolchain Approach for FPGA Fault Tolerance

ADWAIT GUPTE, SUDHANSHU VYAS, and PHILLIP H. JONES, Iowa State University

As the size and density of silicon chips continue to increase, maintaining acceptable manufacturing yields has become increasingly difficult. Recent works suggest that lithography techniques are reaching their limits with respect to enabling high yield fabrication of small-scale devices, thus there is an increasing need for techniques that can tolerate fabrication time defects. One candidate technology to help combat these defects is reconfigurable hardware. The flexible nature of reconfigurable devices, such as Field Programmable Gate Arrays (FPGAs), makes it possible for them to route around defective areas of a chip after the device has been packaged and deployed into the field.

This work presents a technique that aims to increase the effective yield of FPGA manufacturing by reclaiming a portion of chips that would be ordinarily classified as unusable. In brief, we propose a modification to existing commercial toolchain flows to make them fault aware. A phase is added to identify faults within the chip. The locations of these faults are then used by the toolchain to avoid faults during the placement and routing phase.

Specifically, we have applied our approach to the Xilinx commercial toolchain flow and evaluated its tolerance to both logic and routing resource faults. Our findings show that, at a cost of 5–10% in device frequency performance, the modified toolchain flow can tolerate up to 30% of logic resources being faulty and, depending on the nature of the target application, can tolerate 1–30% of the device's routing resources being faulty. These results provide strong evidence that commercial toolchains not designed for the purpose of tolerating faults can still be greatly leveraged in the presence of faults to place and route circuits in an efficient manner.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and routing*; B.8 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms: Reliability, Design, Experimentation, Performance

Additional Key Words and Phrases: Fault tolerance, reconfigurable hardware, design automation

ACM Reference Format:
Adwait Gupte, Sudhanshu Vyas, and Phillip H. Jones. 2015. A fault-aware toolchain approach for FPGA fault tolerance. ACM Trans. Des. Autom. Electron. Syst. 20, 2, Article 32 (February 2015), 22 pages.
DOI: http://dx.doi.org/10.1145/2699838

1. INTRODUCTION

The computing industry has been able to leverage the scaling of transistors to ever-smaller dimensions in accordance to Moore's law for decades. The amazing ability to keep pace with this law motivated the reprinting of Gordon Moore's original 1965 paper [Moore 1998, 2006]. However, there is strong evidence that suggests this era of increasing computing performance by packing more transistors onto a device is coming to an end. Typically, as the density and size of chips increase, their yields tend to decrease [Gupta and Lathrop 1972]. The 2013 International Technology Roadmap for Semiconductors (ITRS) suggests that, as fabrication technology continues to scale

Authors' addresses: A. Gupte, S. Vyas, and P. H. Jones (corresponding author), Electrical and Computer Engineering Department, Iowa State University, Ames, IA 50011; email: phjones@iastate.edu.

DOI: http://dx.doi.org/10.1145/2699838

downward, detecting defects will become increasingly challenging [ITRS 2013b], and that the boundary between what is considered process variation and defects is blurring [ITRS 2013c]. This will result in larger numbers of undesirable structural defects, such as open and short circuits, which in turn challenges the maintaining of high chip yields.

Additionally, evidence has shown that chips associated with low yield batches have an increased likelihood of having reduced lifetimes due to phenomena such as oxide puncture [Kim et al. 2005]. Overall, lower chip yields and reduced device reliabilities (e.g., reduced lifetimes) negatively impact the computing industry as a whole. From a research perspective, these factors bottleneck the degree to which transistors can be scaled, thus constraining the raw computing power that can be leveraged to solve computationally intensive problems. From a business perspective, lower yields mean more chips discarded, resulting in lower profit margins.

Solutions are being pursued on several fronts to help maintain high yields as fabrication scales continue to decrease. Fabrication engineers are exploring new procedures for fabrication [Khan et al. 2008], technology developers are experimenting with new materials to replace or enhance the traditional metal oxide that is primarily used today [Robertson 2007], and design architects are developing techniques to implement fabrication time redundancies to combat increasing defect rates [Vial et al. 2008].

Combating Defects with Reconfigurability. Reconfigurable hardware technology, such as Field Programmable Gate Arrays (FPGAs), shows promise in complementing some of these solutions. FPGAs are devices that can be used to implement 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 where volumes are not sufficient to justify the large initial costs associated with producing Application-Specific Integrated Circuits (ASICs). FPGAs are also a useful prototyping tool that can be used for high-fidelity modeling of an ASIC's functional behavior [Bing and Charoensak 2002]. Since the FPGA fabrication process has many similarities to the fabrication of any other digital silicon-based device, they naturally also share many of the same challenges associated with maintaining yields and reliability as fabrication scales continue to decrease [Gupta and Lathrop 1972; ITRS 2013b, 2013c]. However, unlike ASICs, the reconfigurable nature of FPGAs intrinsically supports redundancy that can be leveraged to tolerate defects that occur at fabrication time or in the field. Their symmetric architecture and reconfigurability can allow designs to be placed and routed around defective areas of the chip after the device has been fabricated, packaged, and deployed into the market.

Techniques to leverage the reconfigurability of FPGAs to allow them to be used despite the presence of defects could help mitigate increasing defect rates in silicon devices by: (1) encouraging industry to migrate more ASIC applications to FPGAs, and (2) integrating aspects of FPGA architectures into their designs. More immediately, such techniques would impact FPGA manufacturers by allowing them to increase their effective chip yields and extending device lifespans.

Contributions. In our work, we evaluate a method that steps toward reclaiming some fraction of FPGAs that would currently be deemed defective. Our approach introduces lightweight modifications to the end-user FPGA toolchain flow to tolerate both manufacturing defects as well as those due to aging. The three core contributions of our work are (1) a technique for leveraging existing commercial FPGA toolchains to make them fault aware, (2) quantifying to what degree existing tools can tolerate both logic [Gupte and Jones 2010] and routing faults, and (3) recognizing and quantifying the trade-off

between the tool's tolerance to faults and the frequency performance of the circuits it can implement.

Organization. The remainder of this article is organized as follows. Section 2 gives related work from the areas of defect quantification, fault location, and fault tolerance. Section 3 introduces our proposed approach. Section 4 then describes the evaluation methodology used to quantify the effectiveness of our approach. Section 5 discusses the results of our evaluation experiments, and Section 6 presents our conclusions and avenues for future work.

2. RELATED WORK

Fault location and fault tolerance in FPGAs are closely related to the work presented in this article. This section first discusses previously proposed methods for fault location in FPGAs. Some of the methods introduced are potential candidates for use during the "test FPGA" phase we propose in our work (see Figure 2). The second part of this section reviews fault tolerance techniques for FPGAs, and places our work into context with respect to this existing body of research.

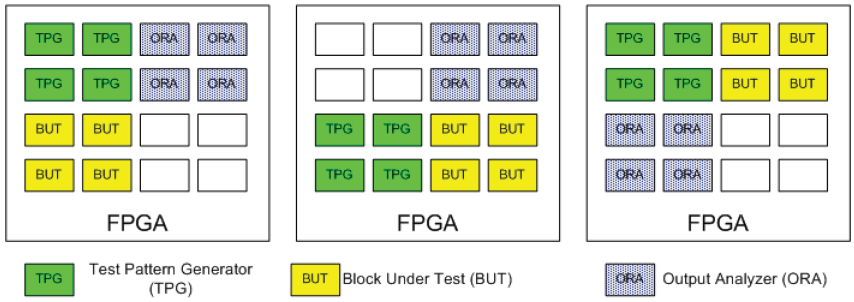
2.1. Fault Location Methods

The configurability and inherent parallelism of FPGAs allow for innovative methods of detecting and locating faults, as compared to standard ASICs. For example, an FPGA can be configured with circuits for detecting/locating faults and then, if deemed usable, can be reconfigured to implement logic for a target application.

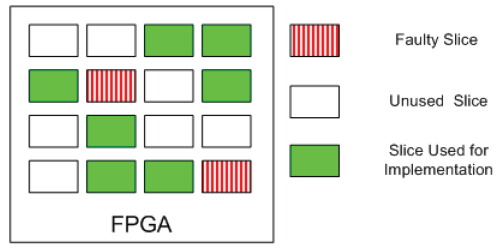
Logic Faults. In Wu and Wu [1999], the authors propose a method in which faults can be detected on an FPGA by programming test circuits on it. They make use of partial dynamic reconfiguration features to reconfigure different portions of the FPGA to act as test circuits. This reduces the amount of time needed to test the entire chip as opposed to reconfiguring the entire FPGA for each configuration of the test circuits they wish to deploy. Reducing test time is especially important in large-scale production environments, where each chip needs to pass through a testing phase before being shipped.

In Inoue et al. [1998], the authors propose another method for testing FPGAs that allows faults to be located at the granularity of a single configurable logic block (CLB). The output of each programmed CLB is used as the input to another. This daisy chaining of CLBs allows the relatively small number of FPGA I/O pins to be used to test a large number of CLBs. By sequentially running this procedure on cascaded rows and then on cascaded columns, they can identify individual CLBs that are faulty (e.g., CLBs at the intersection of a given faulty row and column). In Lala and Burress [2003], the authors present an approach for which, at synthesis time, CLB-level fault detection is infused into a specific design for online fault detection. The authors of Tzilis et al. [2010] locate logic faults at a finer than CLB granularity. By breaking a CLB into finer-grain faults (in this case, 150), faulty CLBs can still be used in a gracefully degraded manner.

Wang and Tsai [1999] present a method of fault location and suggest that their method could allow for faulty chips to be utilized in the field. A built-in self-test (BIST) technique is proposed that uses some regions of the FPGA to test others. As illustrated in the top portion of Figure 1, the FPGA is reconfigured multiple times so that various parts of the chip take turns acting as the testing circuit versus acting as the circuit under test. The classical Preparata, Metze, and Chien (PMC) model [Preparata et al. 1967] is used to account for potential errors in areas configured as test circuits. In Dutt [2006], Dutt presents a method for detecting faults without the assumption of fault-free resources.



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



(b) tools avoid faulty slices during implementation

Fig. 1. Overview of the proposed method.

In Dutton and Stroud [2009a], the authors present a practical implementation of BIST techniques for locating faults on the Xilinx Virtex-5 FPGA. A total of 17 different configurations were developed that, together, achieve 100% coverage of the logic faults that can occur. Similarly, Dutton and Stroud [2009b] identify a practical set of test configurations for identifying I/O tile faults on a Virtex-5 FPGA.

Both logic and interconnect faults are detected by the approaches used in Hsu and Chen [2009] and Tahoori [2011]. An overview of interconnect fault detection research is presented next.

Routing Faults. In Campregher [2005], Campregher discusses the trend of commercial FPGA architectures dedicating increasing amounts of silicon area to routing resources, and indicates a consequent need to develop BIST techniques that concentrate on identifying faulty interconnect resources in addition to faults in logic resources. Almurib et al. [2014] provide a nice overview of research in this area. Most work in the area of locating interconnect faults can be classified as either application-independent or- dependent approaches.

Application-independent approaches [Renovell et al. 1999; Sun et al. 2000; Stroud et al. 2002; Tahoori and Mitra 2005] are often referred to as manufacturing tests and can be used by a manufacturer to check an entire device for faults. An emerging area in this field is locating faults within FPGAs that have 3D interconnects [Peng et al. 2014]. The authors of Ruan et al. [2013] propose an abstract interconnect resource model to help move interconnect fault location research from targeting a single specific FPGA architecture to evaluating a given approach across many interconnect architectures.

A major concern in the testing of interconnect faults is the time required to load test configurations to the FPGA under test. In a manufacturing usage model or when trying to perform online fault tolerance, this can be a significant—if not primary—fraction of fault location time. Application-dependent fault location [Das and Touba 1999; Tahoori 2011; Kumar and Lombardi 2013; Almurib et al. 2014] is one direction

that has been taken to help decrease this overhead. By only testing those resources that will be used by an application, which is often a small fraction of the available interconnect resources, the number of test configurations and/or test vectors run on each configuration can be reduced.

Performance Degradation. Work by Stott et al. [2010b, 2010c] used experimental data to help quantify the rate at which FPGAs can degrade in performance. This degradation can cause new faults to manifest over time. In Keane et al. [2010], the authors propose the use of ring oscillators to act as a means of monitoring silicon device performance degradation in real time. This type of monitor could be implemented using FPGA resources (e.g., CLBs) and used to trigger a fault detection technique when a given application-specific performance degradation threshold is surpassed. Amouri et al. [2012] have explored FPGA aging dependence on device-level architecture and on how a design is mapped [Amouri and Tahoori 2012]. In Amouri and Tahoori [2013b], the authors propose sensors for monitoring FPGA aging, and in Rao et al. [2013] and Amouri and Tahoori [2013a] the authors additionally examine mitigating the impact of aging.

2.2. Fault Tolerance Methods

The researchers Stott et al. [2008, 2010a] and Cheatham et al. [2006] present excellent surveys of different FPGA-based fault tolerance techniques. The latter classify the various techniques into two broad types: device- and configuration-level techniques.

2.2.1. 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 are added. One could further classify these into what this article will refer to as direct and indirect fault-tolerant methods where, in a direct method, redundant resources are only used when a fault occurs, while in an indirect method fault tolerance is indirectly achieved through the innate structure of an architecture.

Direct Methods. In Hatori et al. [1993], the authors propose 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 and Ivey [1994] propose a technique that can mask faults even when they occur in the field. An on-board router is added that, 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 and Piguet [1994] propose a method that can tolerate faults at runtime. Extra resources are used to continuously test logic resources. When a fault is discovered, spare resources are used to mask the fault. In Emmert et al. [2007], an online technique that additionally allows spares to be faulty lookup tables used in nonfaulty modes, and in Agarwal et al. [2013] the authors propose allocating unused resources as spares in a nonuniform application-specific manner.

Indirect Methods. In Roy and Nag [1995], routing architectures were evaluated for their intrinsic routablity in the presence of defects. In Huang et al. [2005], models of commercial and academic FPGA switch matrices were developed. Simulations were run with these models to evaluate the effect of routing faults on a circuit's routablity. In Maidee and Bazargan [2006], various amounts of redundancy were added to an FPGA architecture to quantify the impact of redundancy on a circuit's routablity in the presence of faults. FPGA logic blocks and switch matrices were modeled and Versatile Place and Route (VPR) was run to quantify this relationship. The results of their experiments showed that adding one extra interconnect per switch lane was the

most effective amount of redundancy to add. They found this to be true across several generations of fabrication technology.

A common drawback of device-level techniques is the necessity of additional chip or board-level resources to support them.

2.2.2. Configuration Level. These techniques first identify faulty areas of the chip and then, by performing simple shifts of configuration memory, use alternative resources to implement the design. Methods such as those described by Narasimham et al. [1994] and Hanchek and Dutt [1996] rely on shifting the configuration memory when faults occur to obtain a fault-free implementation. Methods such as described by Emmert and Bhatia [1997] apply heuristics to decide the best direction for these shifts. Howard et al. [1994] present a configuration-level approach that works at a higher level of abstraction than shifting bits. Subcircuits of a design are moved from one "block" location to another to avoid faulty areas.

These configuration-level methods have the common drawback of degrading the place and route (PAR) quality of the original circuit provided by PAR tools. In other words, since these techniques typically only use local information to adjust a design's configuration, there may be more optimal PAR solutions that could be achieved if global information were utilized.

2.2.3. A Classic Work. In Culbertson et al. [1997], the authors present a classic work closely related to our research. They design and implement a custom reconfigurable platform composed of a hierarchy of elements that were engineered to efficiently support defect tolerance. The platform, called Teramac [Amerson et al. 1995], was developed to accelerate large-scale architectural design exploration [Culbertson et al. 1996]. A driving design principle of their system was Rent's Rule Landman and Russo [1971], which relates the number of gates in a partition to the number of signals that interconnect this partition with others. Richard Rent observed that, for a partition consisting of N gates, roughly $\sqrt{N}$ (i.e., $\sim N^e$, where e is typically between .4 and .7) signals interconnected this partition with others.

The primary components of their system were: (1) custom fabricated FPGAs mounted to (2) custom Multichip Modules (MCMs) that were mounted to (3) custom printed circuit boards (PCBs), which were interconnected using ribbon cables. At each level of the hierarchy, they conformed to Rent's Rule in terms of routing resources. Additionally, for the purpose of tolerating defects, they identified and attempted to minimize the area of what they call critical areas at each hierarchical level. For example, within their custom fabricated FPGA, state machines responsible for configuring the FPGA were deemed critical. At manufacturing time, the critical areas for each component were thoroughly tested before being assembled into the system.

Conceptually, the toolchain flow Culbertson et al. implement to automate defect-tolerant mapping of user applications to the Teramac platform matches our approach. In summary, this flow consists of: (1) detecting defective resources, (2) storing these resources into a database, and (3) removing them from the set of resources available to the toolchain. Our work differentiates from that one in the following ways: (1) while their work is an excellent case study of their custom end-to-end operational system, we have focused on quantifying the defect tolerance of a widely used commercial toolchain over a wide range of defect levels, and (2) we analyze the trade-off between the toolchain's defect tolerance and the frequency performance of the circuits it can implement.

Summary. In summary, current FPGA fault tolerance methods typically try to either mask faults, by adding redundancy during the manufacturing stage (device level) at the cost of resource overhead, or try to rectify them in the field by updating the FPGA's

configuration (configuration level) at the cost of circuit performance. In domains where low-latency recovery is needed, many of these approaches are a good fit however, for use-cases where recovery time is not a factor, we propose an approach with zero device overhead that provides a routed circuit of high quality. Current FPGA toolchains do an excellent job of optimizing the way in which a design is implemented. The solutions they arrive at are typically fairly optimized. This work proposes a method that integrates information about faults into standard FPGA toolchains, thus leveraging their optimization capabilities for the purpose of fault tolerance. This article is an extension of our previous work [Gupte and Jones 2010], that only considers logic faults. In this work, we additionally consider routing faults and present an approach for emulating routing faults for the purpose of fault tolerance evaluation.

It should be noted that Xilinx Corporation provides a service called EasyPath [Xilinx 2011] that is closely related to our approach. A fundamental difference between EasyPath and our approach is that ours allows the toolchain to avoid faults, while EasyPath checks whether a given design will work on a given FPGA without knowledge of fault locations. In short, with EasyPath, if a given design works on a given FPGA, then it ships and, not, then the FPGA is not used. There is no attempt to replace and reroute the design to account for errors on the chip.

3. FAULT-AWARE TOOLCHAIN

This section provides an overview of our proposed method for integrating fault information into an FPGA toolchain 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 nomenclature that manufacturers could use to market FPGAs with faults. The section concludes with a discussion of some concerns that need to be considered when using our approach in practice.

3.1. Overview

Figure 1 succinctly summarizes the essence of this article. One of the methods from Section 2 can be used to identify faulty sections of the FPGA. The lower part of the figure shows how the identified faults can be avoided by using a modified toolchain that is fault aware. Proposing such a toolchain and evaluating its effectiveness is the primary thrust of this article.

Figure 2 shows a high-level overview of the proposed fault-aware FPGA toolchain flow. A design goes through several standard stages while being implemented on an FPGA. An additional "test FPGA" stage is proposed that may use any of the fault location techniques discussed in Section 2. The information gained from this test stage can then be fed back into the rest of the toolchain 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 nontrivial 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., a total testing time of 160 seconds is estimated for an XC4025 device [Itazaki et al. 1998]). As compared to the previous configuration-level methods discussed in Section 2, our proposed method leverages the intelligence of mature place-and-route tools to provide an efficient implementation of designs in the presence of faults.

Figure 3 gives an example of the placement obtained by the proposed method as compared to one obtained with the "pebble shifting" method presented by Narasimham et al. [1994]. The "pebble shifting" method avoids an entire column even if only a single logic slice in that column is faulty. The resulting circuit has longer paths than those of the method we propose. To inform the toolchain of slice logic faults, our approach

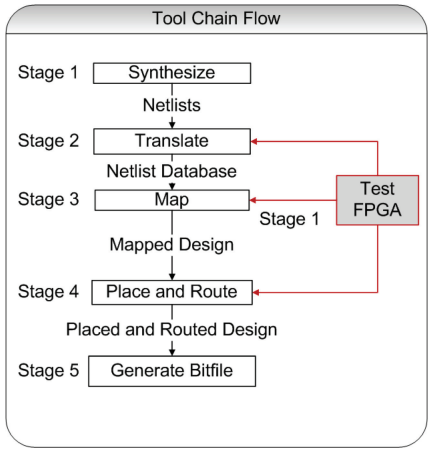


Fig. 2. A fault-aware toolchain flow that tests the FPGA in parallel with the synthesis stage, and then feeds fault information to the remaining stages.

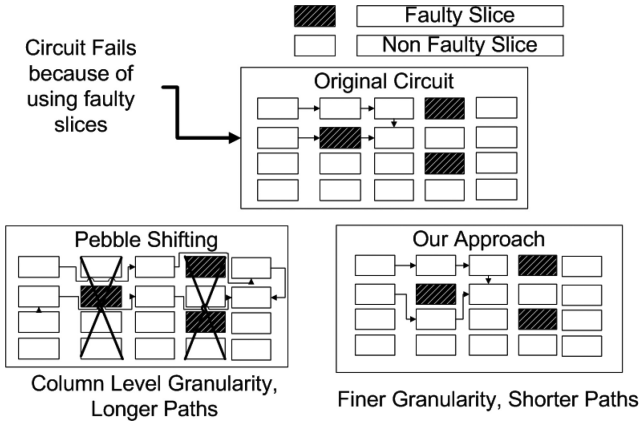
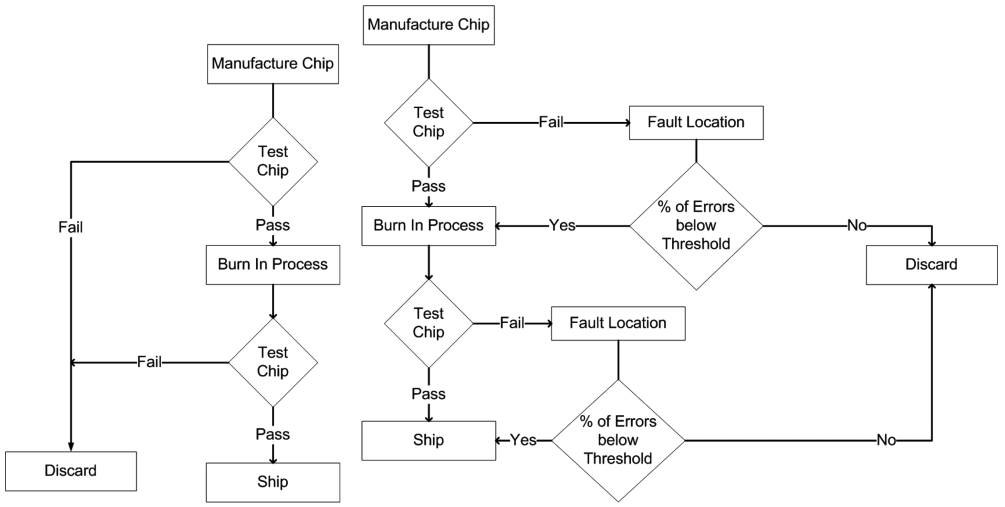


Fig. 3. An example comparison to an existing fault tolerance technique with respect to a circuit's critical path length. We illustrate the benefit of using existing place-and-route tools for fault tolerance as opposed to larger-granularity approaches. In this case a conceptual comparison is made with the pebble shifting technique [Narasimham et al. 1994] that reconfigures at a column granularity.

uses the PROHIBIT constraint available in Xilinx tools that allows certain logic sites to be forbidden for the purpose of placing components. For informing the toolchain of routing faults, a more involved approach is needed as described in Section 4.4. In short, low-level design mechanisms are used to effectively block the tools from using targeted switch matrices. Although our two techniques for passing fault information to the toolchain are effective, a more integrated method of integrating fault information into the toolchain is desirable. One simple approach could be to have a separate "error file" that would be consulted during the implementation process.

3.2. Applicability of the Proposed Method

Two main use-cases can be considered for the proposed method: (1) using FPGAs with defects in mass production and (2) rectifying FPGAs after a fault occurs. These use-cases are discussed next.



(a) current manufacturing flow (b) proposed manufacturing flow

Fig. 4. Comparison between current manufacturing flow and our proposed flow.

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 toolchain 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. Curve 2 in Figure 5 shows the conceptual change in the classic probability-of-failure bathtub curve (i.e., top curve) [Klutke et al. 2003], if the customer implemented our approach.

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 product malfunctions because of an error in the FPGA, rather than discarding the FPGA, a technician could run the fault-aware toolchain on the (now) faulty FPGA. This would create a new bitfile of the design that would avoid the faulty areas. Curve 3 in Figure 5 shows the conceptual change in the probability of FPGAs being discarded if this approach is used.

3.3. 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 faults in the chip. Also as the error grade increases, the maximum frequency of operation decreases. This is due to the place-and-route tools having fewer options from which to choose during implementation (see Section 5.2). 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

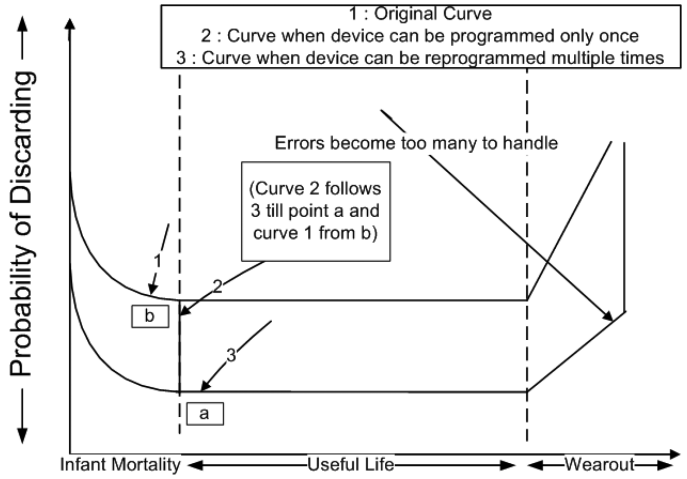


Fig. 5. The top curve (Curve 1) depicts the classic probability-of-failure bathtub curve [Klutke et al. 2003] of a device. Curves 2 and 3 illustrate conceptually how this curve would change if a fault-tolerant toolchain was used. Curve 2, shows when the device is only programmed before deployment, and Curve 3 shows a usage model where the device can also be reprogrammed after deployment.

based on its error grading, the trade-offs associated with the error grading, and the requirements of the design.

3.4. Some Concerns

Implementing fault tolerance in FPGAs requires that 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 our proposed approach.

Critical Resources. The scope of this work is to faults for which place-and-route tools are inherently tolerant. However, as the percentage of these types of faults increases, so too does the chance, a critical resource (e.g., that power line or clock generator) has a fault which could render a device inoperable. One approach to address this issue, taken by the Teramac custom computer in Culbertson et al. [1997], is to thoroughly test for critical faults before spending time locating tolerable ones. In a case study of the Teramac platform [Culbertson et al. 1997], the average number of logic cell faults in their 865 FPGA prototype was about 10%, however, they do not report the number of rejected FPGAs due to critical faults.

Other Faulty Resources. Many modern FPGAs have other resources such as block RAMs, DSP slices, clock management tiles, hardcore microprocessors, etc. Our approach does not evaluate fault tolerance for these resources. In many cases, extending this work to examine these other resources is possible by using the same approach we use for emulating logic slice faults.

Location Constraints. In some FPGA designs, certain resources 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, nor 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.

Mass Production Viability. When FPGAs are a part of a mass-produced product, using faulty FPGAs would require rerunning the tools for each individual FPGA. Cost

analysis must be performed to determine whether the amount of time and resources spent rerunning the tools for each FPGA is offset by the reduced bill of materials resulting from using cheaper, faulty FPGAs.

4. EVALUATION METHODOLOGY

In this section we describe the methodology used to evaluate the effectiveness of our proposed fault-aware toolchain. First, the fault model assumed by this work is given. Section 4.2 then presents our test flow and the metrics used to evaluate our approach. Section 4.3 describes the benchmark circuits that were used, and in Section 4.4 we discuss how fault emulation was implemented.

4.1. Fault Model

Faults occurring in a device can be divided into two main types: those that occur as a result of the fabrication process, and those that occur after the chip is manufactured due to aging. These faults can occur either in the interconnects or the logic elements. Further, when a fault occurs in a logic element, it can occur in a LUT, a flip-flop, a multiplexer, or the combinational carry-chain logic. In this work, we have abstracted away the details of the exact point of failure of the chip. For the case of logic faults, we consider a slice as a whole to be either working correctly or as faulty. For the case of routing faults, we take a pessimistic view that a given switch matrix is, as a whole, functional or not.

Another common classification of faults is as either permanent or transient. This work assumes permanent faults. In addition, the positioning and clustering of faults depend on the mechanism causing these faults (e.g., dust particles during lithography, radiation exposure, thermal cycling, electron migration). In this work, we have evaluated our approach under pessimistic conditions for which fault sizes and rates are much greater than typically found in industry, thus we have used a simple uniformly random distribution of faults. There are a number of works that focus on accurately modeling fault sizes, rates, and distributions [Stapper 1983; Campregher et al. 2005a, 2005b; ITRS 2013a] showing that, under nominal conditions, defects tend to cluster.

4.2. Methodology

Figure 6 illustrates our evaluation flow. For each experimental run, the first step "generate uniformly distributed errors" creates a defect map. This defect map is generated using a seeded random number generator, allowing us to reproduce fault patterns across benchmark circuits. The fault types and locations are then passed to the Xilinx toolchain via its user-constraint-file (UCF). In general, the UCF file is a mechanism that allows user-specific constraints to be defined. In our case, we use this file to keep the tools from using those resources that our defect map indicate are faulty.

From here, the toolchain runs as normal. At the end of each run, the toolchain generates its standard reports that indicate whether it was able to route the design and meet the specified timing requirements.

This test procedure was repeated for various fault levels, FPGA utilization levels, and benchmarks (see Section 5). Each combination of the parameters tested were run approximately 200 times for different random fault patterns. A set of 200 runs took approximately 9 hours to complete, typically spread over 75 CPU cores.

Next we describe the metrics used to evaluate our proposed fault-aware toolchain approach.

Error Tolerance. This metric is used to evaluate the degree to which our approach can tolerate faults. In order to measure this metric, various percentages of the FPGA's resources were marked as faulty. A normally distributed random variable was

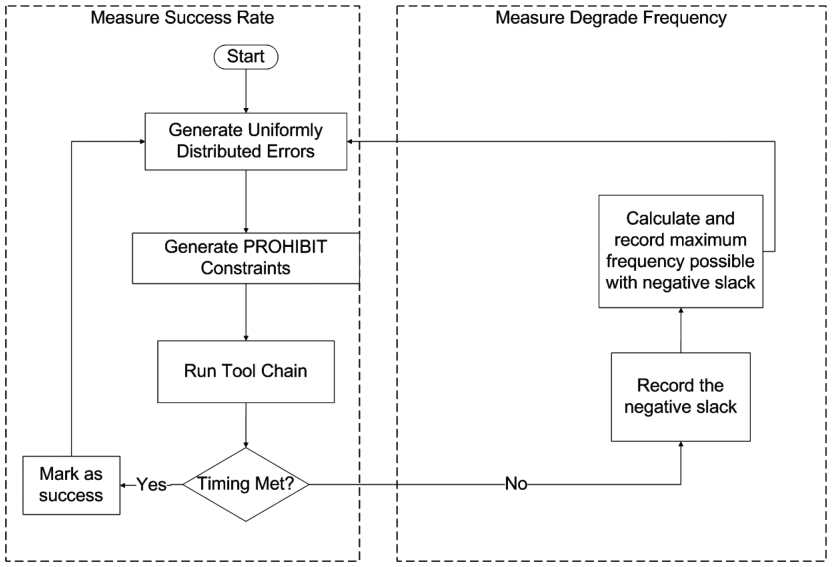


Fig. 6. The testing flow used to evaluate our proposed approach.

generated to represent the number of faulty resources, and these faults were then distributed across the chip as described in Section 4.4.

The tools were then run on the synthesized netlists to obtain a placed-and-routed (PAR) design. The resulting implementation was analyzed to check that it was still able to meet timing. If so, the run was marked as a success for our proposed approach and if not, then the run was marked as a failure. We discuss the results of these experiments as well as how this metric relates to chip yields in Sections 5.1 and 5.2.

Performance Degradation. We use this metric to investigate the relation between the percentage of faults on the FPGA and the performance degradation needed by the tools to successfully implement a design. For designs that fail to place and route during the "error tolerance" evaluation, 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 6).

These results are used to show that, even in cases where it is not possible to implement the design with the specified timing 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. Section 5.2 evaluates our approach with respect to this metric to determine the error threshold that our approach can tolerate for given performance degradation levels.

Comparison with Smaller, Fault-Free Chips. In order to understand the trade-offs between using a large FPGA with faults and a smaller fault-free FPGA, we implemented benchmark circuits targeting smaller fault-free chips and measured the maximum frequency at which they could be implemented. These frequencies were then compared to those obtained for the larger faulty FPGA. Our findings are discussed in Section 5.4.

4.3. Circuit/Device Description

A subset of the benchmark circuits proposed by Corno et al. [2000], was used to evaluate our approach. These benchmarks were originally created to evaluate test pattern generation methods for identifying stuck-at faults and easily scale to occupy various

percentages of an FPGA by replicating the design. Since these designs are based on real-world circuits (e.g., processor cores), they form good test cases for evaluating our approach. The selected benchmarks were replicated multiple times to obtain utilizations of approximately 25%, 50%, and 75%. The benchmarks chosen were:

(1) *b17*—three subsets of an Intel 80386 processor;
(2) *b18*—three Viper processors and six 80386 processors;
(3) *b20*—a Viper processor and a modified version of the Viper processor;
(4) *b21*—two Viper processors; and
(5) *b22*—a Viper processor and two modified versions of the Viper processor;

For evaluating slice logic faults, all five of the prior benchmarks were used. For evaluating the impact of routing faults, only benchmarks b17, b18, and b20 were used. For evaluating routing faults, benchmark b20 provided a test case made up of many small and relatively loosely coupled cores, and b17 provided a test case for larger cores (i.e., the 80386 cores are much larger than the Viper cores). The implications of these different characteristics are discussed in our results section with respect to the tool's ability to tolerate routing faults. Additionally, it should be noted that, since the results for b17 and b18 were similar, only b17 is discussed in detail.

The benchmarks did not include a constraint file and hence constraints had to be provided for the evaluation. A feature of these benchmarks is that they use a single clock constrained to within 3% of the highest frequency that the tools could successfully place and route a benchmark on a nonfaulty FPGA. When the benchmarks were replicated, their outputs were ORed together to deal with mapping them to the limited number of I/O ports available on the actual FPGA device.

The test circuits targeted a Xilinx Virtex-5 LXT 110 [Xilinx 2014]. This device has a total of 17280 slices and was chosen because there are four smaller devices available in the same family. This allowed us to compare a large faulty FPGA's performance against smaller fault-free FPGAs (see Section 5.4).

4.4. Fault Implementation

Here we describe how logic slice and routing faults were emulated.

Logic slice faults. Emulation of logic slice faults was a fairly straightforward process. The Xilinx toolchain provides an attribute called PROHIBIT that can be associated with a specified logic slice. This attribute indicates to the toolchain that a particular logic slice is not allowed to be used in the design being implemented. Based on the defect map generated by the "generate uniformly distributed error" step of Figure 6, PROHIBIT attributes are appropriately added to the tool's user constraint file (UCF).

Routing faults. Emulating routing faults was more challenging. We used a combination of Xilinx FPGA Editor, XDL (Xilinx Design Language), and DIrect RouTing (DIRT) constraints to generate a circuit that utilized all of the output ports of a single switching matrix (see Figure 7). With all the output ports used, the tool was kept from routing through the targeted switch matrix, thus emulating a faulty switch matrix. We then replicated and instantiated this fault as directed by our generated defect map.

5. RESULTS AND ANALYSIS

This section presents the findings from the experiments described in Section 4. First, we examine the error tolerance of our proposed method. Next, we quantify the percentage of errors that can be tolerated if the frequency of a design is allowed to degrade by a given amount. We then discuss differences between the tool's tolerance to logic slice faults versus routing faults. Section 5.4 presents our findings with respect to the idea of using larger faulty FPGAs as equivalent small nonfaulty FPGAs, and this section

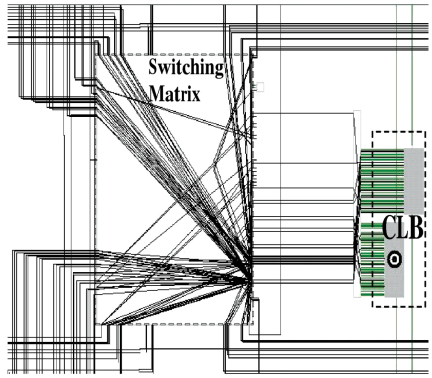


Fig. 7. Routing fault emulation shows the FPGA Editor view of a Virtex-5 switching matrix and associated CLB. The darkened traces indicate the switch matrix output ports, all of which were manually blocked in order to emulate a faulty switch matrix.

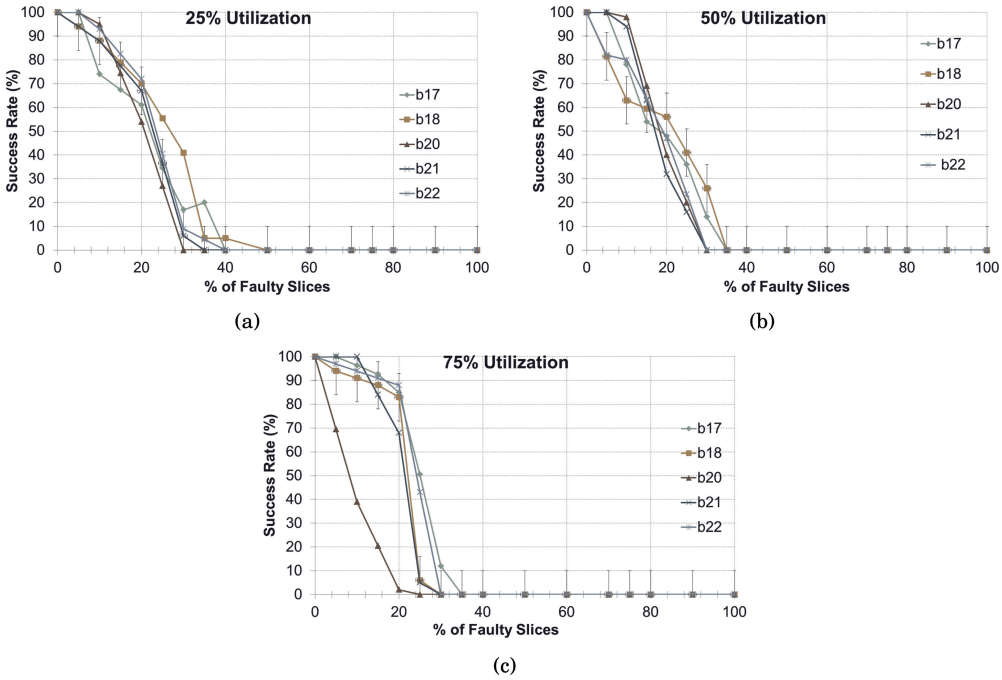


Fig. 8. Success rate when varying error rates for a device utilization of: (a) 25%; (b) 50%; (c) 75%. All data points are within a 10% confidence interval at a confidence level of 90%.

concludes with a discussion of the sensitivity of our approach to the tightness of timing constraints.

5.1. Logic Slice Error Tolerance

Figure 8(a) shows the success rate of the fault-aware toolchain 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 logic slices are faulty. The success rate is 0% when 30–60% of the logic slices are faulty. This can be explained by the tightly constrained clock in the original design. The tools

Table I. Percent of Faulty Logic Slices that can be Tolerated for a Maximum Frequency Performance Cost of 10%

Utilizations	25%	50%	75%
Percent of Faulty Logic slices	30%	30%	20%

were barely able to 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 8(b) 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% one. The success rates reach 0% when 20–30% of the logic slices become faulty.

Figure 8(c) 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 its 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 0.2ns 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 toolchain would be able to tolerate more faults.

From the figures it can be seen that the success rates are high until about 10% of the logic slices become faulty. Thus, even for those designs that cannot compromise on their frequency, it may 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 section examines the amount of frequency reduction necessary to successfully place and route the experimental runs that failed when no frequency degradation was allowed.

5.2. Logic Slice Fault Tolerance when Degraded Performance is Permitted

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 I summarizes the key observations obtained from this evaluation that give evidence that our proposed approach is quite tolerant of FPGA logic slice faults. It shows that designs using 25%, 50%, and 75% of the chip can tolerate 30%, 30%, and 20% logic slice faults, respectively.

The success rate of the proposed approach for a given percentage of errors can be seen in Figures 8(a), 8(b), and 8(c). Figures 9(a), 9(b) and 9(c) must be looked at in the context of Figures 8(a), 8(b), and 8(c) respectively. For example, if, for a utilization of 25% and 20% logic slice errors the success rate in Figure 8(a) is 70%, then 30% of the runs fail at the original frequency. These runs require the frequency degradation shown in Figure 9(a).

As can be seen from Figures 8(a) and 9(a), at a 25% utilization and up to 15% errors, the designs can be implemented in 67.5–82.5% of the cases. In those cases where this is not possible, they still can be implemented after about a 1% degradation in performance, until about 15% of the logic slices are faulty. From 15–30% 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,

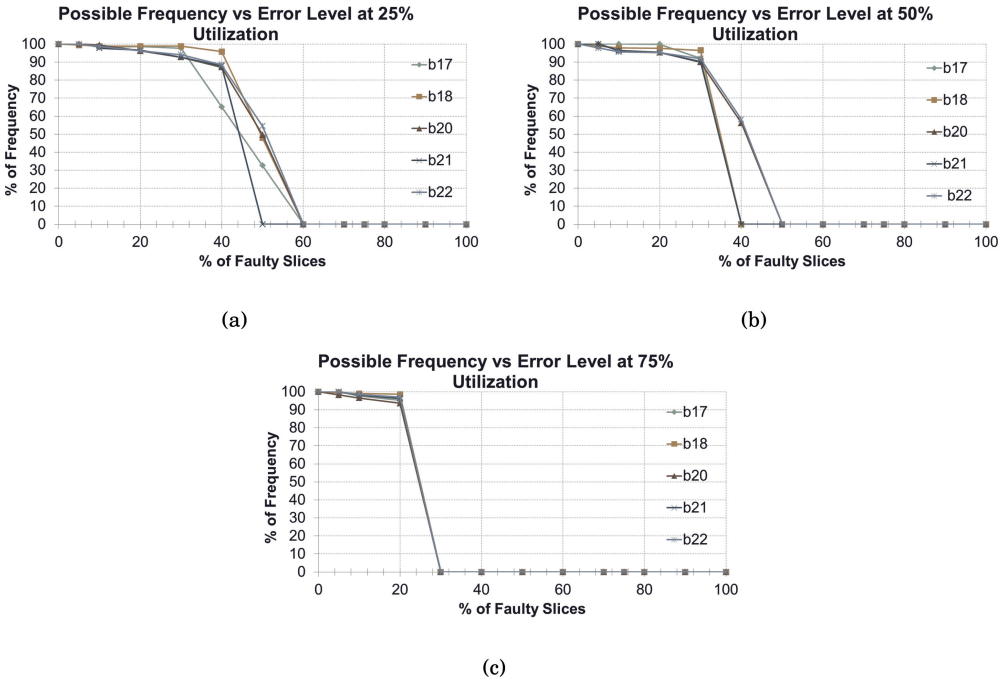


Fig. 9. Degradation in frequency required for test runs that failed for the circuit's original timing constraint at device utilizations of: (a) 25%; (b) 50%; (c) 75%. All data points are within a 7% confidence interval at a confidence level of 90%.

our fault-aware toolchain 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 an FPGA speed grade).

Similarly, from Figures 8(b) and 9(b), it can be seen that between 32–56% of the designs successfully meet timing until about 20% of the logic slices are faulty. Those that fail can meet timing with an 8–10% frequency degradation, until about 30% of the logic 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 Figures 8(c) and 9(c) 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 those cases where the designs fail timing, a 3–5% frequency degradation allows them to be implemented. This knee varies with utilization. In these graphs the knee 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 logic slices being faulty can be tolerated by the proposed approach for a maximum performance cost of 10% (see Table I).

5.3. Routing Faults Compared to Logic Slice Faults

In Rubin and DeHon [2009], the authors state that 80–90% of an FPGA's area is allocated to routing resources. Thus, it is important to evaluate our proposed approach in the presence of routing faults in addition to logic slice faults. Figures 10(a) and 10(b) show that routing faults can have a much larger impact on the toolschain's ability to tolerate faults as compared to logic slice faults.

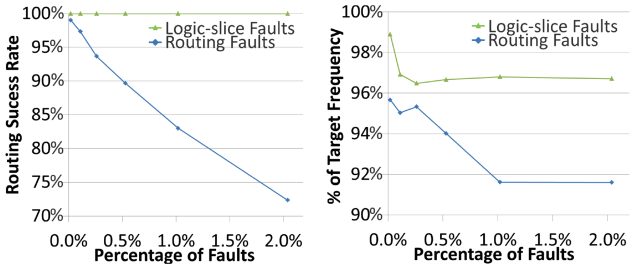


Fig. 10. Comparing logic slice and routing faults: (a) connectivity versus percentage of faults; (b) percentage degradation of circuit frequency for b17 using 25% of the FPGA.

Figure 10(a) quantifies at what fault level the toolchain can no longer route (i.e., fully connect) the design. This concept did not exist when only logic slice faults were emulated, since 100% of the routing resources were available for constructing any given signal's path. However, now that routing resources are being reduced (i.e., removing available switch matrices), the tools quickly run into issues fully connecting (i.e., routing) the design. In this case, even with only 25% of the FPGA being utilized and when just over 1% routing faults are present, approximately 15% of the designs are not connectable by the toolchain, whereas, for fault scenarios made up only of logic slice faults, the toolchain could route designs up until the point where there were not enough fault-free logic slices to implement the design.

In addition to the primary observation in Figure 10(a) that circuit connectivity is more sensitive to routing faults than when only logic slice faults are present, Figure 10(b) illustrates that, compared to logic slice faults, routing faults cause a greater degradation in frequency for those designs that can be routed.

Loosely Coupled vs. Tightly Coupled Cores. We investigate how the relative degree of coupling within a design impacts the toolchain's ability to successfully and efficiently place and route designs. Section 4.3 provides an overview of the benchmarks compared and explores why, relative to one another, they are considered tightly versus loosely coupled.

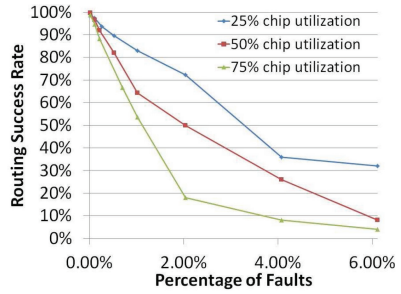
Figure 11(a) examines benchmark b17, which consists of relatively large densely routed cores. For most cases, before routing faults reach 2%, over 30% of the designs are not even connectable by the toolchain. Furthermore, Figure 11(c) shows that, even for just 1% routing faults, the designs for these same fault emulation experiments are already quickly degrading in operating frequency performance.

However, benchmark b20, which is composed of smaller loosely coupled cores, gives very different behavior. Figure 11(b) shows that the toolchain can fully connect the design, even as the number of routing faults reaches 30% or more. In addition, Figure 11(d) shows that for b20 the toolchain can tolerate up to 25% routing faults before degrading operating frequency by more than 10%.

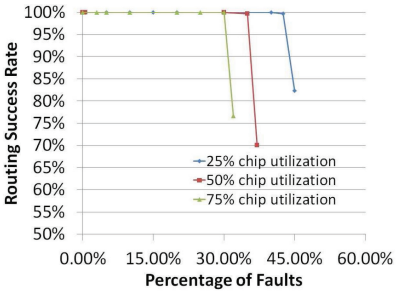
These results clearly show that the toolchain can better tolerate routing faults when implementing designs made up of small loosely coupled cores as opposed to those composed of large densely routed cores.

5.4. Comparison with Smaller, Fault-Free Chips

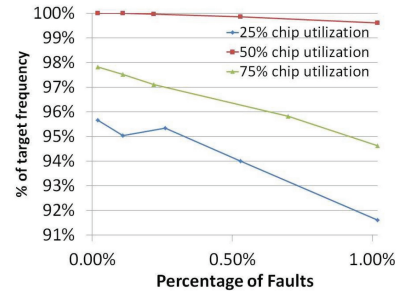
Figure 12(a) shows the ratio of the maximum frequency at each error level in the larger chip (LX110T) with respect to the frequency possible in smaller fault-free FPGAs. A ratio of 1 indicates the same performance. As can be seen in Figure 12(a), having up to 30% logic slice errors in the larger chip results in approximately the same performance



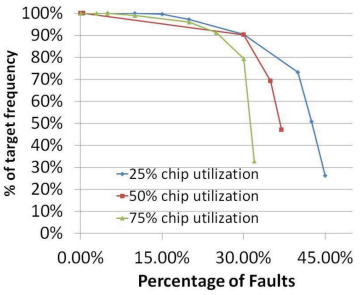
(a) routability for b17: dense 80386 cores



(b) routability for b20: small loosely coupled Viper cores

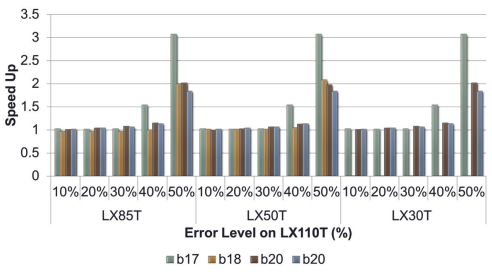


(c) performance degradation for b17: dense 80386 cores

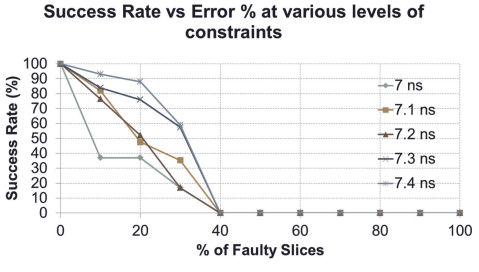


(d) performance degradation for b20: small loosely coupled Viper cores

Fig. 11. It can be seen that designs made up of small loosely coupled cores allow the toolchain to tolerate routing faults significantly better than designs composed of large densely routed cores.



(a)



(b)

Fig. 12. (a) Ratio of the frequency at which different benchmarks could have been implemented on smaller fault-free chips as compared to the maximum frequencies possible at various error levels on the larger LX110T FPGA. These tests were run for benchmarks that utilized 25% of the LX110T, (b) sensitivity analysis as success rate at various error percentages with different timing constraints in nanoseconds (benchmark b17, 25% utilization).

as a smaller fault-free chip. Thus, a design that fits a smaller fault-free chip can be efficiently implemented on a larger faulty chip having the same number of fault-free logic slices. This observation could be used by manufacturers to help market chips with faults at a reduced price.

If a manufacturer were to choose an error threshold below which any chip is deemed appropriate for market, then the manufacturer's effective yields would increase to those given in Eq. (1), where RBT (Rejected Below Threshold) is the percentage of originally

rejected chips that have an error level below a set threshold.

$$Effective\_Yield = Old\_Yield + \frac{(100 - Old\_Yield) * (RBT)}{100}. \quad (1)$$

5.5. Timing Constraint Sensitivity

All benchmark designs were constrained to within approximately 0.2ns of their minimum fault-free periods. In order to get an idea of how sensitive success rate was to how tightly the design was constrained, we ran tests to constrain benchmark b17 from 7ns (∼minimum fault-free period) to 7.4ns. Figure 12(b) shows the findings for this experiment. As can be seen in the figure, the tools are quite sensitive to how tightly the design is constrained, thus it seems reasonable that relaxing the timing constraints by just 10% (∼.7ns) allows the tools to tolerate a significant number of faults.

6. CONCLUSION

We have presented and evaluated an approach for developing an FPGA toolchain flow that is fault aware. Our results have shown this approach can tolerate a significant number of logic slice faults (up to 30%) with a modest decrease in circuit frequency performance (10% or less) for the Virtex-5 LX110T. With respect to routing faults, it was found that designs made up of small loosely coupled modules could tolerate up to approximately 30% faults as well. However, for designs composed of large and tightly coupled cores, only about 1% routing faults could be tolerated (a factor of 30 less).

Even for designs in which only 1% of routing faults could be tolerated (which is still a significant number of faults), it is worth recalling that in our experiments we defined a single routing fault to be an entire unusable switch matrix. This is an extremely pessimistic routing fault model. It is expected that the tools would perform much better if only a couple of routes within a switch matrix were marked faulty.

We have also proposed the idea of establishing an equivalence between larger, faulty FPGAs and smaller, nonfaulty ones. Our results show that for logic slice faults such an equivalence is feasible.

It has been clearly shown that, while FPGA toolchains were not designed to avoid faults, they show an impressive capability to do so. If fault information could be coupled more tightly with the tool's underlying routing algorithm, it is expected that even more impressive performance could be obtained. Our results provide strong motivation for further exploration of such research directions. A vehicle for future investigation is the open-source VPR place-and-route tool that is used widely by academia for advancing computer-aided design [Betz and Rose 1997; Rose et al. 2012; Hung et al. 2013].

REFERENCES

A. Agarwal, J. Cong, and B. Tagiku. 2013. The survivability of design-specific spare placement in FPGA architectures with high defect rates. *ACM Trans. Des. Autom. Electron. Syst.* 18, 2, 33:1–33:22.

H. Almurib, T., Nandha Kumar, and F. Lombardi. 2014. Scalable application-dependent diagnosis of interconnects of SRAM-based FPGAs. *IEEE Trans. Comput.* 63, 6, 1540–1550.

R. Amerson, R. Carter, W. Culbertson, P. Kuekes, and G. Snider. 1995. Teramac-configurable custom computing. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'95).* 32–38.

A. Amouri, S. Kiamehr, and M. Tahoori. 2012. Investigation of aging effects in different implementations and structures of programmable routing resources of fpgas. In *Proceedings of the International Conference on Field-Programmable Technology (FPT'12).* 215–219.

A. Amouri and M. Tahoori. 2012. High-level aging estimation for FPGA-mapped designs. In *Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL'12).* 284–291.

A. Amouri and M. Tahoori. 2013a. Degradation in FPGAs: Monitoring, modeling and mitigation (PHD-forum paper: Thesis broad overview). In *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications (FPL'13).* 1–2.

A. Amouri and M. Tahoori. 2013b. Lifetime reliability sensing in modern FPGAs. In *Embedded Systems Design with FPGAs*, P. Athanas, D. Pnevmatikatos, and N. Sklavos, Eds., Springer, 55–77.

V. Betz and J. Rose. 1997. VPR: A new packing, placement and routing tool for FPGA research. In *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications (FPL'97)*. Springer, 213–222.

X. Bing and C. Charoensak. 2002. 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 (ICARCV'02)*. Vol. 3. 1653–1657.

N. Campregher. 2005. FPGA interconnect fault tolerance. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'05)*. 725–726.

N. Campregher, P. Y. K., Cheung, G. Constantinides, and M. Vasilko. 2005a. Yield modelling and yield enhancement for FPGAs using fault tolerance schemes. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'05)*. 409–414.

N. Campregher, P. Y. K., Cheung, G. A., Constantinides, and M. Vasilko. 2005b. Analysis of yield loss due to random photolithographic defects in the interconnect structure of FPGAs. In *Proceedings of the 13th ACM/SIGDA International Symposium Field-Programmable Gate Arrays (FPGA'05)*. 138–148.

J. A., Cheatham, J. M., Emmert, and S. Baumgart. 2006. A survey of fault tolerant methodologies for FPGAs. *ACM Trans. Des. Autom. Electron. Syst.* 11, 501–533.

F. Corno, M. Reorda, and G. Squillero. 2000. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Des. Test Comput.* 17, 3, 44–53.

W. Culbertson, R. Amerson, R. Carter, P. Kuekes, and G. Snider. 1996. Exploring architectures for volume visualization on the teramac custom computer. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'96)*. 80–88.

W. Culbertson, R. Amerson, R. Carter, P. Kuekes, and G. Snider. 1997. Defect tolerance on the Teramac custom computer. In *Proceedings of the 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'97)*. 116–123.

D. Das and N. Touba. 1999. A low cost approach for detecting, locating, and avoiding interconnect faults in FPGA-based reconfigurable systems. In *Proceedings of the 12th International Conference on VLSI Design-VLSI for the Information Appliance (VLSID'99)*. 266–269.

S. Durand and C. Piguet. 1994. FPGA with self-repair capabilities. In *Proceedings of the ACM International Workshop on Field Programmable Gate Arrays (ICVD'94)*. 266–269.

S. Dutt. 2006. Mixed PLB and interconnect BIST for FPGAs without fault-free assumptions. In *Proceedings of the 24th IEEE VLSI Test Symposium (VTS'06)*. 36–43.

B. Dutton and C. Stroud. 2009a. Built-in self-test of configurable logic blocks in Virtex-5 FPGAs. In *Proceedings of the 41st Southeastern Symposium on System Theory (SSST'09)*. 230–234.

B. Dutton and C. Stroud. 2009b. Built-in self-test of programmable input/output tiles in Virtex-5 FPGAs. In *Proceedings of the 41st Southeastern Symposium on System Theory (SSST'09)*. 235–239.

J. Emmert, C. Stroud, and M. Abramovici. 2007. Online fault tolerance for FPGA logic blocks. *IEEE Trans. VLSI Syst.* 15, 2, 216–226.

J. M. Emmert and D. Bhatia. 1997. 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 (FPL'97)*. 141–150.

A. Gupta and J. Lathrop. 1972. Yield analysis of large integrated-circuit chips. *IEEE J. Solid-State Circ.* 7, 5, 389–395.

A. Gupte and P. Jones. 2010. An evaluation of a slice fault aware tool chain. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'10)*. 1803–1808.

F. Hanchek and S. Dutt. 1996. 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 (ICCD'96)*. 326–331.

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. 1993. Introducing redundancy in field programmable gate arrays. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC'93)*. 7.1.1–7.1.4.

N. Howard, A. Tyrrell, and N. Allinson. 1994. The yield enhancement of field programmable gate arrays. *IEEE Trans. VLSI Syst.* 2, 1, 115–123.

C.-L. Hsu and T.-H. Chen. 2009. Built-in self-test design for fault detection and fault diagnosis in SRAM-based FPGA. *IEEE Trans. Instrument. Measur.* 58, 7, 2300–2315.

J. Huang, M. Tahoori, and F. Lombardi. 2005. Fault tolerance of switch blocks and switch block arrays in FPGA. *IEEE Trans. VLSI Syst.* 3, 7, 794–807.

E. Hung, F. Eslami, and S. Wilton. 2013. Escaping the academic sandbox: Realizing VPR circuits on Xilinx devices. In *Proceedings of the 21st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'13).* 45–52.

T. Inoue, S. Miyazaki, and H. Fujiwara. 1998. Universal fault diagnosis for lookup table FPGAs. *IEEE Des. Test Comput.* 15, 1, 39–44.

N. Itazaki, F. Matsuki, Y. Matsumoto, and K. Kinoshita. 1998. Built-in self-test for multiple CLB faults of a LUT type FPGA. In *Proceedings of the 37th Asian Test Symposium (ATS'98).* 272–277.

ITRS. 2013a. The International Technology Roadmap for Semiconductors (ITRS), Interconnect. http://www.itrs.net/.

ITRS. 2013b. The International Technology Roadmap for Semiconductors (ITRS), Metrology. http://www.itrs.net/.

ITRS. 2013c. The International Technology Roadmap for Semiconductors (ITRS), Yield Enhancement. http://www.itrs.net/.

J. Keane, X. Wang, D. Persaud, and C. Kim. 2010. An all-in-one silicon odometer for separately monitoring HCI, BTI, and TDDB. *IEEE J. Solid-State Circ.* 45, 4, 817–829.

J. Kelly and P. Ivey. 1994. Defect tolerant SRAM based FPGAs. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'94).* 479–482.

H. Khan, D. Mamaluy, and D. Vasileska. 2008. Simulation of the impact of process variation on the optimized 10-nm FINFET. *IEEE Trans. Electron. Devices* 55, 8, 2134–2141.

K. Kim, M. Zuo, and W. Kuo. 2005. On the relationship of semiconductor yield and reliability. *IEEE Trans. Semiconductor Manufact.* 18, 3, 422–429.

G. Klutke, P. Kiessler, and M. Wortman. 2003. A critical look at the bathtub curve. *IEEE Trans. Reliab.* 52, 1, 125–129.

T. Kumar and F. Lombardi. 2013. A novel heuristic method for application-dependent testing of a SRAM-based FPGA interconnect. *IEEE Trans. Comput.* 62, 1, 163–172.

P. Lala and A. Burress. 2003. Self-checking logic design for FPGA implementation. *IEEE Trans. Instrument. Measur.* 52, 5, 1391–1398.

B. Landman and R. L. Russo. 1971. On a pin versus block relationship for partitions of logic graphs. *IEEE Trans. Comput. C20,* 12, 1469–1479.

P. Maidee and K. Bazargan. 2006. Defect-tolerant FPGA architecture exploration. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'06).* 1–6.

G. Moore. 1998. Cramming more components onto integrated circuits. *Proc. IEEE* 86, 1, 82–85.

G. E. Moore. 2006. Cramming more components onto integrated circuits. *IEEE Solid-State Circ. Newslett.* 20, 3, 33–35.

J. Narasimham, K. Nakajima, C. Rim, and A. Dahbura. 1994. Yield enhancement of programmable ASIC arrays by reconfiguration of circuit placements. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* 13, 8, 976–986.

Y.-L. Peng, D.-M. Kwai, Y.-F. Chou, and C.-W. Wu. 2014. Application-independent testing of 3-D field programmable gate array interconnect faults. *IEEE Trans. VLSI Syst.* 22, 2, 207–219.

F. P. Preparata, G. Metze, and R. T. Chien. 1967. On the connection assignment problem of diagnosable systems. *IEEE Trans. Electron. Comput. EC16,* 6, 848–854.

P. Rao, A. Amouri, S. Kiamehr, and M. Tahoori. 2013. Altering lut configuration for wear-out mitigation of FPGA-mapped designs. In *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications (FPL'13).* 1–8.

M. Renovell, J. M. Portal, J. Figueras, and Y. Zorian. 1999. Testing the configurable interconnect/logic interface of SRAM-based FPGA's. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'99).* 618–622.

J. Robertson. 2007. Growth of nanotubes for electronics. *Mater. Today* 10, 12, 36–43.

J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson. 2012. The VTR project: Architecture and CAD for FPGAs from verilog to routing. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'12).* ACM Press, New York, 77–86.

K. Roy and S. Nag. 1995. On routability for FPGAs under faulty conditions. *IEEE Trans. Comput.* 44, 11, 1296–1305.

A. Ruan, J. Yang, L. Wan, B. Jie, and Z. Tian. 2013. Insight into a generic interconnect resource model for Xilinx Virtex and Spartan series FPGAs. *IEEE Trans. Circ. Syst. II: Express Briefs* 60, 11, 801–805.

R. Rubin and A. Dehon. 2009. Choose-your-own-adventure routing: Lightweight load-time defect avoidance. In *Proceeding of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'09)*. ACM Press, New York, 23–32.

C. Stapper. 1983. Modeling of integrated circuit defect sensitivities. *IBM J. Res. Devel.* 27, 6, 549–557.

E. Stott, P. Sedcole, and P. Cheung. 2008. Fault tolerant methods for reliability in FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'08)*. 415–420.

E. Stott, P. Sedcole, and P. Cheung. 2010a. Fault tolerance and reliability in field-programmable gate arrays. *IET Comput. Digital Techniques* 4, 3, 196–210.

E. Stott, J. Wong, and P. Cheung. 2010b. Degradation analysis and mitigation in FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'10)*. 428–433.

E. A. Stott, J. S. Wong, P. Sedcole, and P. Y. Cheung. 2010c. Degradation in FPGAs: Measurement and modelling. In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'10)*. ACM Press, New York, 229–238.

C. Stroud, J. Nall, M. Lashinsky, and M. Abramovici. 2002. BIST-based diagnosis of FPGA interconnect. In *Proceedings of the International Test Conference (TEST'02)*. 618–627.

X. Sun, S. Xu, B. Chan, and P. Trouborst. 2000. Novel technique for built-in self-test of FPGA interconnects. In *Proceedings of the International Test Conference (TEST'00)*. 795–803.

M. Tahoori. 2011. High resolution application specific fault diagnosis of FPGAs. *IEEE Trans. VLSI Syst.* 19, 10, 1775–1786.

M. Tahoori and S. Mitra. 2005. Application-independent testing of FPGA interconnects. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* 24, 11, 1774–1783.

S. Tzilis, I. Sourdis, and G. Gaydadjiev. 2010. Fine-grain fault diagnosis for FPGA logic blocks. In *Proceedings of the International Conference on Field-Programmable Technology (FPT'10)*. 154–161.

J. Vial, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, and A. Virazel. 2008. Using TMR architectures for yield improvement. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems (DFTVS'08)*. 7–15.

S.-J. Wang and T.-M. Tsai. 1999. Test and diagnosis of faulty logic blocks in FPGAs. *IEE Proc. Comput. Digital Techniques* 146, 2, 100–106.

C.-F. Wu and C.-W. Wu. 1999. Fault detection and location of dynamic reconfigurable FPGAs. In *Proceedings of the International Symposium on VLSI Technology, Systems and Applications (VTSA'99)*. 215–218.

XILINX. 2011. Easypath-6 FPGA product brief. http://www.xilinx.com/publications/prod_mktg/EasyPath6_Product_Brief.pdf.

XILINX. 2104. Virtex-5 FPGA datasheet. http://www.xilinx.com/support/documentation/data_sheets/ds202.pdf.

Received February 2012; revised October 2014; accepted November 2014