An Evaluation of a Slice Fault Aware Tool Chain

Adwait Gupte
Department of Electrical and
Computer Engineering
Iowa State University
Email: adwait@iastate.edu

Phillip Jones
Department of Electrical and
Computer Engineering
Iowa State University
Email: phjones@iastate.edu

***Abstract*—As FPGA sizes and densities grow, their manufacturing yields decrease. This work looks toward reclaiming some of this lost yield. Several previous works have suggested fault aware CAD tools for intelligently routing around faults. In this work we evaluate such an approach quantitatively with respect to some standard benchmarks. We also quantify the trade-offs between performance and fault tolerance in such a method. Leveraging existing CAD tools, we show up to 30% of slices being faulty can be tolerated. Such approaches could potentially allow manufacturers to sell larger chips with manufacturing faults as smaller chips using a nomenclature that appropriately captures the reduction in logic resources.**

I. Introduction

Field Programmable Gate Arrays (FPGAs) are a reconfigurable platform that can be used 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 volumes are not large enough 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 [1]. Technology advances continue to enable larger and denser FPGAs to host more complex applications. However as the density/size of chips grow, their yields tend to decrease [2]. Methods to make FPGAs usable despite the presence of some faults could improve the profitability of manufacturers, and extend device lifespans. In this work we evaluate a method that steps toward reclaiming some fraction of FPGAs that would currently be deemed as defective. Our method makes changes to the end user toolchain flow to tolerate both manufacturing defects as well as defects due to aging.

The two main contributions of this paper are 1) to quantitatively analyze a fault aware tool chain and 2) quantify the trade offs between performance and tolerance of faults in such an approach. Figure 1 gives an overview of the method evaluated in this work.

The rest of this paper is organized as follows: Section II gives related work from the fields of fault location and fault tolerance. Section III introduces our proposed approach. Section IV then describes the evaluation methodology used to quantify the effectiveness of this approach. Section V discusses the results of our evaluation experiments, and Section VI presents our conclusions and avenues for future work.

II. Related Work

This section is divided into two main parts, both closely related to our work. The first discusses previously proposed methods for fault location in FPGAs. Some of these methods are potential candidates for use in the "Test FPGA" phase of our modified tool flow shown in Figure 1(b). The second part of this section reviews fault tolerance techniques for FPGAs, and places our approach in context with respect to this existing body of work.

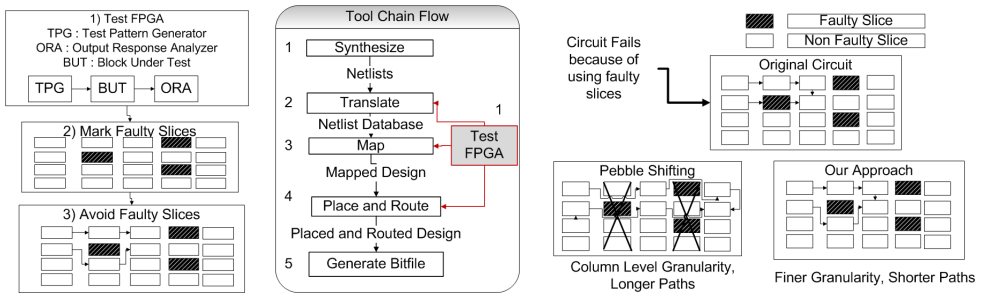
A. Fault Location Methods

The configurability and inherent parallelism in an FPGA allows for innovative methods for detecting and locating faults. [4] [5] propose different methods of fault detection on an FPGA by using the dynamic reconfiguration capability of the FPGA and chaining CLBs together respectively. In [6], Wang et al. suggest a method of fault location, and point out that such a method could help tolerate faults in a chip. Our approach can make use of their fault location method to this end. The method proposed by Wang is a BIST technique that uses regions of the FPGA to test other regions. The top portion of Figure 1(a) illustrates this idea. By configuring the FPGA multiple times, various parts are tested in turn. This method uses the classical Preparata, Metze, and Chien (PMC) model [7] to take into account that errors in the reconfigurable fabric may be present in the areas being used to test other areas.

Mishra et al [8] propose an approach very similar to what we suggest in this paper, but restrict their evaluation to only the testing of the FPGA. The use of the information gained from testing to avoid faults is only mentioned without evaluation. Rubin et al [9] propose a method for having alternative paths in the bitfile and choosing at configuration time the one that avoids the faulty areas. While this may be feasible for small circuits, as circuit size increases the sheer number of possibilities would make exhaustive enumeration infeasible.

B. Fault Tolerance Methods

In [10], Cheatham et al. present an excellent survey of different fault tolerance techniques that have been applied to FPGAs. They have classified these techniques into two broad types: Device Level and Configuration Level techniques.



(a) Conceptual Idea (b) Modified Tool Flow (c) Example comparison to an existing fault tolerant technique with respect to circuit critical path length

Fig. 1. (a) shows the high level conceptual view of our approach. (b) illustrates how our approach would be integrated into exiting tool flows. An additional testing stage is run in parallel with synthesis to locate faults. These fault locations are passed to the toolchain in order to leverage their implementation strategies to intelligently place the design while avoiding faults. (c) helps to 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 to the pebble shifting technique [3] that reconfigures at a column granularity.

1) Device Level: These techniques are incorporated into a device during manufacturing. Redundant resources such as routing paths, and programmable logic blocks are added to the device. In [11], Hatori proposes a technique that can tolerate faults that are discovered during manufacturing by adding redundant rows and selection logic to routing resources. In this method the faulty rows are made invisible to the user by routing around them. In [12], Kelly proposes a technique that can mask faults even when they occur in the field. An onboard 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 it remains mostly transparent to the end user software. In [13], Durand 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.

2) Configuration Level: These techniques incorporate run time fault tolerance into FPGAs. They first identify faulty chip areas. Then by performing simple shifts of configuration memory they use alternative resources to implement the design. "Pebble Shifting" by Narasimhan [3] and the work by Hanchek [14] are two examples of such techniques that shift configuration memory when faults occur to attempt to create a fault free implementation. Methods such as [15] by Emmert et al. use heuristics to try to choose the best direction for configuration memory shifts. In [16] FPGA slices are partitioned into groupings, then the redundancy within these groupings is used to maintain an error free operation. This method requires all possible configuration files be generated, which is infeasible for large circuits.

In summary device level methods find application in areas where no new faults are expected to occur. They mask errors at the manufacturing stage at the cost of additional chip-level or board-level resources. Configuration level techniques are useful in cases where new faults do occur on an FPGA, and the implementation tool chain cannot be rerun. This is often at the cost of a degraded circuit implementation as compared to what the tool chain can implement (see Figure 1(c)). Current FPGA tool chains do an excellent job of implementing circuits efficiently. We evaluate a method that incorporates error information into the toolchain to intelligently avoid faults.

III. A Fault Aware Toolchain

This section first gives an overview of our proposed method. Next the impact of our approach on the life cycle of an FPGA is given, and possible usage models are presented. This is followed by a discussion of an example nomenclature manufacturers could use to market FPGAs with faults. This section concludes with a discussion of some concerns associated with our approach.

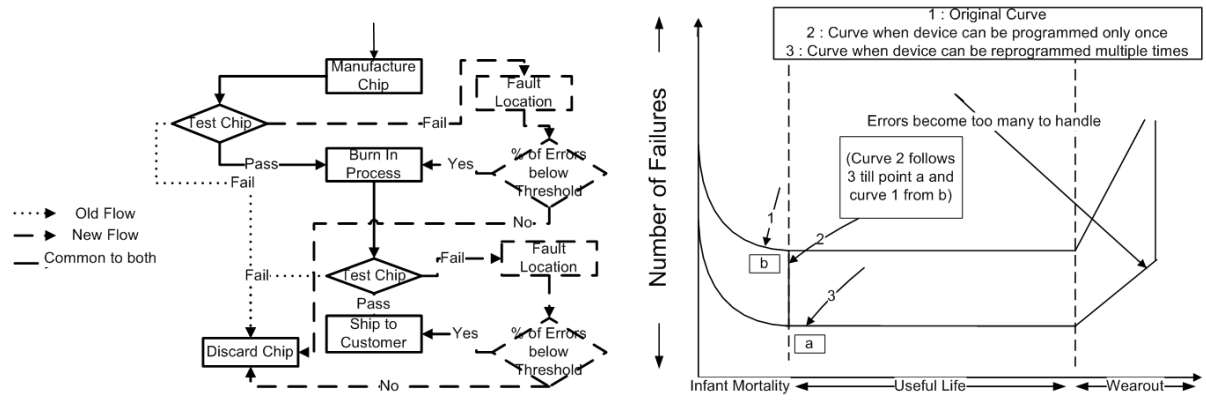
A. Overview

Figure 1(b) shows a high level overview of our proposed fault aware FPGA implementation toolchain flow. A design goes through several stages while being implemented on an FPGA. We propose an additional "Test FPGA" stage be included that uses one of the fault location techniques discussed in Section II. 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. Thus as compared to the previous methods discussed in Section II, our method leverages the intelligence of mature place and route tools to provide an efficient implementation of the design.

B. Applicability of Proposed Method

Two main usecases can be considered for the proposed method 1) using erroneous FPGAs for mass production and 2) rectifying FPGAs after a slice fault occurs. We discuss these usecases in turn in this subsection:

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 2(a). Curve 2 in figure 2(b) shows



(a) (b)

Fig. 2. (a) Changes in the manufacturing process needed to take advantage of a fault aware toolchain. These changes would increase effect chip yield according to the formula discussed in Section V-B. (b) The impact a fault aware toolchain has on the well known bathtub curve for device failure rate.

the change in the number of devices that are discarded by the manufacturer if such a method is used.

In the second use case, 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 on 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 which would avoid the faulty areas. Curve 3 in Figure 2(b) shows the change in the number of FPGAs discarded if this approach is used.

C. Error Grading

Due to variances in the manufacturing process, device frequencies are 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. In addition to lose of logic resources, as the number of faulty slices increases, the maximum frequencies of operation decreases. This is due to the place and route tools having fewer options from which to choose during implementation (see Section V-B). Hence either a separate error grade could be constituted or a composite grading system taking both the percentage of errors and speed into account could be created. Before shipping, the FPGAs could be run through a fault location phase. Depending on the number of faults detected, the chip could be discarded or branded with a 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.

D. Some Concerns

In this subsection we discuss two concerns with respect to our proposed approach.

1) Faulty Routing Resources: When a given area of the chip becomes faulty it is likely the routing resources in that area have become faulty as well. There are fault location methods that can identify faulty routing resources. Such information would need to be integrated into the tool flow in order to make our approach more practical.

2) 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 our approach or any other approach can do to allow the FPGA to be used. In the latter case, it may be possible to move the RLOC origin to another point to satisfy the constraints.

IV. 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 IV-B then presents our test flow, and the metrics that were used to evaluate our approach. Section IV-C describes the benchmark circuits that were used as a workload.

A. 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 by considering a slice as a whole to be either working correctly or faulty. Another common classification of faults is permanent or transient. This work assumes permanent faults.

The positioning and clustering of faults depend on the reason/source of those faults. In this work we have assumed faults have a uniformly random distribution.

B. Methodology

In this section we describe the criteria used to evaluated the fault aware toolchain. Figure 3 shows a flowchart for our testing procedure.

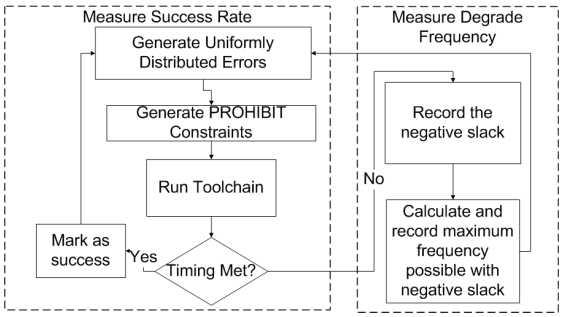


Fig. 3. The testing flow used to evaluate the proposed method.

1) 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 slices were marked as faulty. A normally distributed random variable was generated to represent the number of faulty slices. These faults were then uniformly distributed across the chip by appropriately marking an array that represented each slice location. PROHIBIT constraints were generated to restrict the tools from placing any components in the slices that were selected to be faulty. 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 it did meet timing, then the run was marked as a success for our proposed approach. If timing was not met, then the run was marked as a failure for our proposed approach. In order to obtain statistically significant results, 100 iterations of the entire toolchain flow were run for each fault level. A total of 75 computers were used to complete our runs in a reasonable amount of time. An automated test script ran these experiments for 2 full days. We discuss the results of these experiments, as well as how this metric relates to chip yields in Sections V-A and V-B.

2) Performance Degradation: We use this metric to investigate the relation between the percent of slices faulty on the FPGA, and the performance degradation needed by the tools to successfully implement a design. Section V-B evaluates our approach with respect to this metric.

3) 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 benchmarks circuits on smaller fault-free chips and measured the maximum frequency at which they could be implemented. These frequencies were then compared to the ones obtained for the larger faulty FPGA. Our findings are discussed in Section V-C.

C. Circuit/Device Description

A subset of the benchmark circuits proposed by F. Corno et al. in [17] were used to evaluate our approach. These benchmarks were originally created to evaluate test pattern generation methods for identifying stuck at faults. They 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 get utilizations of approximately 25%, 50% and 75%. The benchmarks chosen were:

1) b17: 3 subsets of the Intel 80386 processor.
2) b18: 3 Viper processors and 6 Intel 80386 processors.
3) b20: A Viper processor and a modified Viper processor.
4) b21: Two Viper processors.
5) b22: A Viper processor and 2 modified Viper processors.

The test circuits were deployed to a Xilinx Virtex-5 LXT 110[18]. This device has a total of 17280 slices and was chosen because there are 4 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 V-C).

V. Results & Analysis

The following section presents our findings for the experiments we performed using the metrics and benchmarks described in Section IV.

A. Error Tolerance

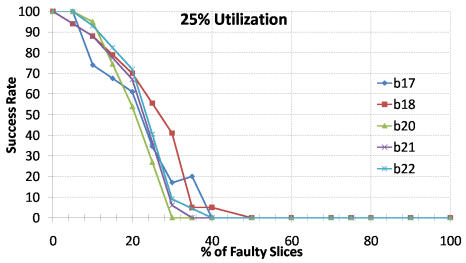
Figure 4(a) shows the success rate of the fault aware toolchain at 25% utilization of the LX110T FPGA. As seen in the figure, even with almost 75% of the chip empty, the designs have a significant chance of not meeting timing when around 10% of the slices become faulty. The success rate reaches 0% when between 30% to 60% of the slices become faulty. This can be explained with the very tightly constrained clock on the original design. The tools were able to only just 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 4(b) shows the success rate for 50% of the FPGA 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 between 20-30% of the slices become faulty.

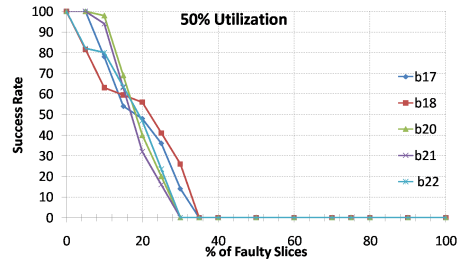
Figure 4(c) shows the success rate for 75% utilization of the FPGA. For the b17 benchmark, it was not possible to achieve an exact 75% utilization so it's utilization was kept at 70%. Hence the b17 benchmark success rate doesn't fall to 0% at 25% errors, instead it does so at 30% errors.

The designs we 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 our 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 the fault aware tool chain would be able to tolerate more faults.

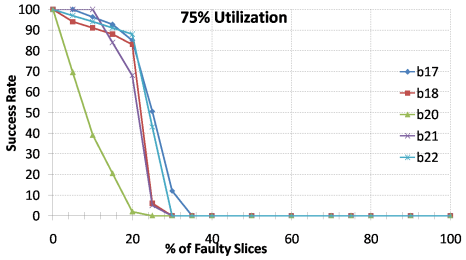
It can be seen that until about 10% of the slices become faulty, the success rates are high. Thus even for designs that cannot compromise on their frequency, it might still be cost effective to buy larger chips with errors present, and discard the small percentage of them that cannot meet timing. On the other hand, if a frequency degradation is acceptable, then



(a)

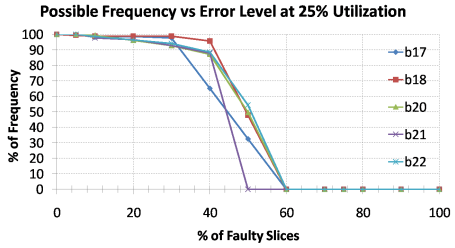


(b)

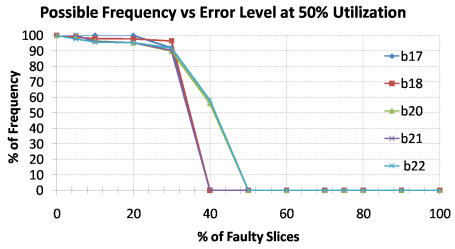


(c)

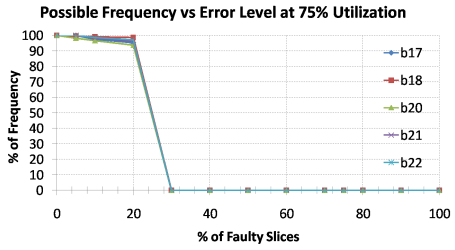
Fig. 4. The fault aware toolchain success rates for 25% - 75% device utilization. These success rates are within 1% of their actual values at a 95% level of significance.



(a)



(b)



(c)

Fig. 5. The design runs for which PAR fails for a given fault rate can still be successfully implemented by lowering its frequency requirement. These graphs show the amount of degradation required (as a % of the original frequency constraint) to obtain a successful PAR at different error levels for each design. The percentages are means that are within 4% of the actual value at a 95% level of significance.

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 implement the experimental runs that failed at various error levels.

B. Degraded Performance

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. We used the negative slack times reported by all the failed designs at various error levels, and calculated 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 a key observation obtained from this evaluation that gives evidence that our approach is quite tolerant of FPGA slice faults. It shows that 30%, 30% and 20% slice faults can be tolerated in designs using 25%, 50% and 75% of the chip respectively.

The success rate of our approach for a given percentage of errors can be seen from Figure 4. Figure 5 must be looked at in the context of Figure 4. For example, if for a utilization of 25% and 20% slice errors the success rate in Figure 4(a) is 70%, then 30% of the runs fail at the original frequency. It is these runs that require the frequency degradation shown in Figure 5(a).

As can be seen from Figure 5(a) and Figure 4(a), at a 25% utilization and up to 15% errors, the designs can be implemented in 67.5%-82.5% of the cases. When they cannot be, they can be implemented after about a 1% degradation in performance until about 15% errors. 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, our 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 4(b) and Figure 5(b) 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 our experimental results suggest for a design utilizing 50% of the chip, our approach can tolerate up to 30%

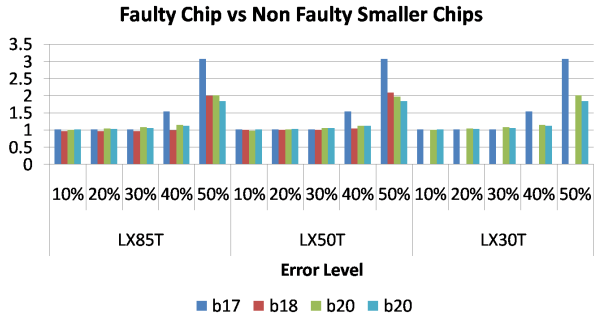


Fig. 6. This figure shows the ratio of the frequency at which the different benchmarks could have been implemented on smaller fault-free chips as compared to the maximum frequencies possible at various errors level on the larger LX110T FPGA. These tests were run for benchmarks that utilized 25% of the LX110T

of the chip being faulty at a reasonable performance cost.

From Figure 4(c) and 5(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 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 our approach for a maximum performance cost of 10%. (see Table I)

If a manufacture 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 1. Where RBT (Rejected Below Threshold) is the % of originally Rejected chips that have an error level Below a set Threshold.

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

TABLE I
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%

C. Comparison with smaller, fault-free chips

Figure 6 shows the ratio of the possible 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 6, having up to 30% slice errors in the larger chip results in approximately the same performance 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 slices. This observation could be used by manufacturers to help market chips with faults at a reduced price.

VI. CONCLUSION & FUTURE WORK

We have put forth and evaluated an approach for developing an FPGA implementation toolchain that is fault aware. We have shown that this approach can tolerate a significant number of errors (up to 30%) with a modest decrease in circuit performance (10% or less). Our empirical evaluation suggests that this approach is a step toward allowing manufacturers to reduce the number of devices discarded due to faults. Given our approach only considers a simplified fault model (restricted to faults being in slices and uniformly distributed), an important direction for this work is to extend the fault model to take into account other resources such as routing, DSP blocks, and hard-core processors. Another interesting avenue for future work is investigating the distribution in the number of errors that actually occur in FPGAs that are discarded by manufacturers.

REFERENCES

[1] X. Bing and C. Charoensak, "Rapid fpga prototyping of gabor- wavelet transform for applications in motion detection," in *International Conference on Control, Automation, Robotics And Vision*, Dec. 2002.
[2] A. Gupta and J. Lathrop, "Yield analysis of large integrated-circuit chips," in *IEEE Journal of Solid State Circuits*, vol. 7, no. 5, Oct. 1972.
[3] J. Narasimhan, K. Nakajima, C. S. Rim, and A. T. Dahabura, "Yield enhancement of programmable asic arrays by reconfiguration of circuit placements," in *IEEE Transactions on Computer Aided Design of Integrated Circuit Systems*, vol. 13, Aug. 1994.
[4] C.-F. Wu and C.-W. Wu, "Fault detection and location of dynamic reconfigurable fpgas," in *International Symposium on VLSI Technology, Systems and Applications*, 1999.
[5] T. Inoue, S. Miyazaki, and H. Fujiwara, "Universal fault diagnosis for lookup table fpgas," in *IEEE Design and Test of Computers*, May 1998.
[6] S.-J. Wang and T.-M. Tsai, "Test and diagnosis of fault logic blocks in fpgas," in *International Conference on Computer Aided Design*, 1997.
[7] F. Preparata, G. Metze, and R. Chien, "On the connection assignment problem of diagnosable systems," in *IEEE Trans. Electronic Comput.*, Dec. 1967.
[8] M. Mishra and S. Goldein, "Defect tolerance at the end of the roadmap," in *International Test Conference*, 2003.
[9] R. Rubin and A. DeHon, "Choose-your-own-adventure routing: lightweight load-time defect avoidance," in *FPGA '09: Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2009, pp. 23–32.
[10] J. Cheatham and J. Emmert, "A survey of fault tolerant methodologies for fpgas," in *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 2, Apr. 2006.
[11] F. Hatori, T. Sakurai, 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*, 1993.
[12] J. L. Kelly and P. A. Ivey, "A novel approach to defect tolerant design for sram based fpgas," in *Proceedings of the ACM International Workshop on Field Programmable Gate Arrays*, 1994.
[13] S. Durand and C. Piguet, "Fpga with selfrepair capabilities." in *Proceedings of the ACM International Workshop on Field Programmable Gate Arrays.*, 1994.
[14] F. Hanchek and S. Dutt, "Design methodologies for tolerating cell and interconnect faultsin fpgas," in *International Conference on Computer Design*, 1996.
[15] J. M. Emmert and D. K. Bhatia, "Partial reconfiguration of fpga mapped designs with applications for fault tolerance and yield enhancement," in *Proceedings of the 7th International Workshop on Field Programmable Logic and Applications*, 1997.
[16] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Efficiently supporting fault-tolerance in fpgas," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays. ACM*, 1998.
[17] F. Corno, M. S. Reorda, and G. Squillero, "Rt-level itc 99 benchmarks and first atpg results," in *IEEE Design & Test of Computer*, Aug. 2000.
[18] *Virtex-5 FPGA Datasheet*, Xilinx Inc.