Run-Time Defect Tolerance using JBits

Prasanna Sundararajan
Xilinx Inc.
2100 Logic Drive
San Jose, California 95124
Prasanna.Sundararajan@xilinx.com

Steven A. Guccione
Xilinx Inc.
2100 Logic Drive
San Jose, California 95124
Steven.Guccione@xilinx.com

ABSTRACT

The ability to tolerate defects in semiconductor devices has the potential for both increasing yields of devices being manufactured and making it economically feasible to manufacture even larger devices. While FPGA devices appear to be well suited to providing defect tolerance, practical application of existing research and techniques has been somewhat elusive. One barrier to acceptance is that existing defect tolerance techniques for FPGAs have tended to rely on either modifications to device architectures or modifications to design tools. We describe a software-based technique for providing defect tolerance which requires neither changes to device hardware or software tools. This approach uses the Xilinx *JBits*$^{(tm)}$ toolkit and operates at the core library level. Addressing defect tolerance locally using core library elements rather than taking a global approach helps provide direct support for run-time reconfiguration. Circuits may be configured and reconfigured rapidly in the presence of these defects. This rapid configuration also provides a path for practical use in more traditional manufacturing environments.

Keywords

Defect Tolerance, Run-Time Reconfiguration, FPGA, Java, Cores

1. INTRODUCTION

As with most semiconductor devices, the size of FPGAs are limited by the quality of their manufacturing process. Because there are typically a relatively fixed number of flaws distributed across a silicon wafer, the larger the device, the fewer the number of working parts per wafer. The largest devices typically have very small manufacturing yields and are the most expensive. While defect density is a limiting factor in the size of semiconductor devices, improvements in manufacturing processes have led to a steady increase in device sizes. Table 1 shows the recent growth in density of the largest available FPGA devices from Xilinx [15].

FPGA 2001, February 11-13, 2001, Monterey, CA, USA

Table 1: FPGA device density.

Device Family	System Gates	Date Introduced
XC4000XLA $^{(tm)}$	500K	Sept. 1998
Virtex$^{(tm)}$	1,124K	Oct. 1998
Virtex-E	4,047K	Sept. 1999
Virtex2	10,000K (approx)	May 2000

While advances in manufacturing technology have made impressive gains, other techniques to improve the number of working devices, or *yield* of a manufacturing process are possible. In this paper, we describe a software-based technique for providing defect tolerance. This is done by implementing defect tolerance at the *Run-Time Parameterizable (RTP) Core* [6] level. These cores designed using *JBits* software tool kit [7] selectively skip the defective circuit elements and functions correctly even in the presence of defective configurable logic blocks and interconnect wires. This also provides the core designer control over the layout of the design in the presence of defects. A defect tolerant constant multiplier RTP core was designed using this technique and was implemented on a Xilinx Virtex device and the results were verified.

The paper is outlined as follows: Section 2 discusses the related work. Section 3 provides an overview of *JBits* interface and RTPCores. Section 4 describes the RTPCore based defect tolerance approach. In Section 5 design of a defect tolerant constant multiplier and the implementation results are provided. Section 6 discusses the technique to tolerate defective interconnect wires. In Section 7 conclusions and the future work are discussed.

2. RELATED WORK

One popular technique for *Defect Tolerance* in semiconductor devices relies on the use of redundant circuitry [9]. These redundant circuitry techniques are generally based on making available "spare parts" on the device die which may be swapped in to replace other defective elements. This permits otherwise faulty devices to function correctly. These defect tolerance techniques not only have the ability to increase yields, but also to make larger devices economically feasible.

While these defect tolerance techniques have long been of theoretical interest, they have only found practical application in a small range of device types. The limiting factor for these redundant logic techniques is providing the redundant components efficiently. In the worst case, providing redun-

dant circuitry for the entire device could easily double the die size. This would in all likelihood eliminate any gains made by increasing the yield of functional devices.

One area where defect tolerance has been used successfully is in memory arrays. Memory devices are somewhat unique in that they contain a large, regular array of relatively independent cells. Providing a high level of defect tolerance typically requires only a spare column of memory cells which can be used to replace a column containing a defective cell. This spare column is swapped in using a variety of techniques in the final stage of manufacturing. In the case of memories, the relatively low overhead of providing a spare column of memory cell makes this technique economically feasible. FPGAs represent a similar, but more complex cellular array than memories. In simple programmable logic devices, it may be possible to employ similar redundancy techniques to improve manufacturing yields.

Using programmable cellular arrays for defect tolerance has been explored as far back as Minnick [14]. Redundant cells or columns of cells can be made available and swapped in much in the same way spare columns of memory cells are reassigned. Unlike simple memory arrays, the high level of interconnect between cells and the timing issues involved in programmable logic devices make hardware approaches such as this infeasible except for the simplest architectures.

Research that suggests special architectures and modifications to hardware that would result in defect tolerance is also being pursued [3] [10]. While special purpose architectures and manufacturing process enhancements may be used to provide defect tolerance in FPGA devices, another approach is to use design software to map circuits around defective portions of the device. Such software-based defect tolerance does not require any redundant circuitry or any changes to the existing device architecture or manufacturing processes.

Much of the work in defect tolerance for FPGAs has revolved around the idea of mapping circuits around known faults at the device, wafer and even system level [1] [2] [4] [8] [13]. The general approach is to perform off-line testing of the device and mark defective portions of the device, which can later be bypassed by the design mapping tools. The drawback of this approach is the necessity to re-run the design mapping tools to produce a unique circuit configuration for each device. Such tools are typically compute intensive and require time on the order of hours on a relatively powerful system to produce a result. This is usually not acceptable in a large-scale manufacturing environment. In addition, because these techniques typically require either special tools or modifications to existing tools, very little of this work has been demonstrated on commercially available devices.

By contrast, the work of Emmert et al. [5] operates on a commercially available FPGA device and operates *on-line*, that is, at run-time. This approach combines a technique for automatically detecting faults in an operating FPGA circuit and reconfiguring the circuit around any detected defects. This approach solves the problem of long tool runtimes by bypassing the design tools altogether. Circuits are re-mapped incrementally at the device level.

The system we have implemented combines the techniques of on-line and off-line approaches to defect tolerance. The assumption is that devices are tested and defects detected off-line during the manufacturing process are logged in a defect database. This database contains the location of defective Configurable Logic Blocks (CLBs) and interconnect wires. Using this defect database circuits may then be configured and reconfigured on-line at run-time, continuing to operate in the presence of these defects. The use of *JBits* run-time reconfiguration tool suite [7] provides the speed and flexibility necessary to produce new configurations in the presence of defects.

3. RUN-TIME RECONFIGURATION USING JBITS

The *JBits* tool suite is a collection of Java *Application Program Interfaces (APIs)* and associated tools used to build, test and debug *Run-Time Reconfigurable (RTR)* systems. At the lowest level, *JBits* gives direct read and write access to all configurable elements of the FPGA device. This permits both the logic in Look-Up Tables (LUTs) and the routing to be quickly and directly modified in-system at run-time.

Currently *JBits* is being used primarily as a development tool for run-time reconfigurable systems. Circuits are specified in terms of logic and routing and grouped into Java objects in the form of cores. The *JBits* model views an FPGA device as a two dimensional array of CLBs. These CLBs are indexed with a row and column parameter. Partitioning the device this way permits small sections of the device to be programmed. Loops or conditional statements can be used to replicate the programming. This forms the basis of designing RTP Cores. Along with this, *JRoute*, [11] a router based on the *JBits* API is used to route the interconnects.

Unlike the traditional static circuits of existing schematic capture or HDL Cores, *JBits* Cores are not fixed data objects. *JBits* Cores are instead code sequences describing how to construct circuits. This permits a high degree of flexibility in how circuits are instantiated. In particular, circuit parameters such as bit width can be specified as late as at run-time. These *Run-Time Parameterizable (RTP) Cores* not only simplify library design and provide a wider range of choices for designers, but allow circuits to be modified and configured in-system at run-time, perhaps based on user input or sensor data.

RTP Cores are the basic building block for *JBits* designs. By defining the functionality of groups of CLBs, they provide a useful framework not only for hierarchical design, but also for run-time reconfiguration. It is within this framework that support for defect tolerance is provided.

4. RTP CORE-BASED DEFECT TOLERANCE

One concern with the existing approaches to software-based FPGA defect tolerance is that they tend to operate at the device level. The software to support defect tolerance in this manner also takes a global view of resources and operates more or less autonomously. This necessarily inserts a layer between the application software and the implementation which not only interferes with direct control of resources, but complicates attempts to do run-time reconfiguration.

Because the RTP Core model is run-time reconfiguration friendly, it is desirable to extend this model in some way to provide defect tolerance. The most direct way to accomplish

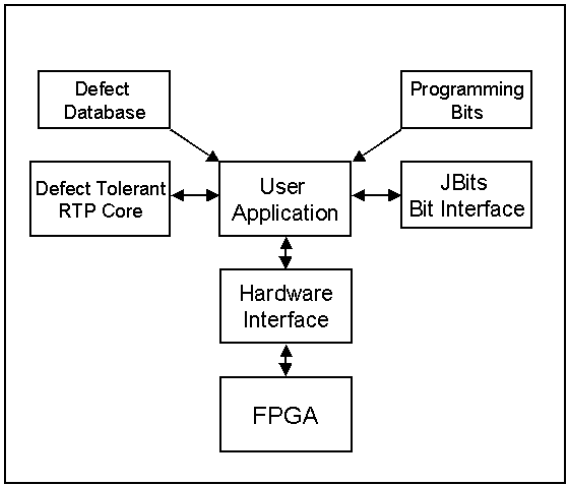

Figure 1: Defect Tolerant RTP Core Scheme

this is to require the RTP Core to produce a correctly operating circuit, even in the presence of defects. RTP Cores would be specifically constructed to bypass faulty logic and interconnect dynamically at run-time. This technique does not attempt to correct the faults but rather avoids the usage of defective CLBs and interconnects in the circuit being implemented. Thus irrespective of the nature of defect present in CLBs and interconnects this technique would produce defect tolerant RTP Cores. This approach may require that RTP Cores use more resources in the presence of defects, but that correctly functioning circuits are produced. The elements involved in the implementation of a Defect Tolerant RTPCore is shown in Figure 1.

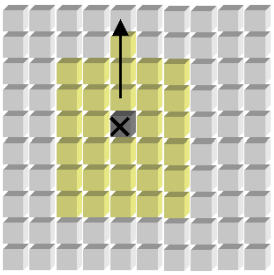

Figure 2: Skip CLB Mode

Because RTP Cores operate at the CLB level, defect tolerance using RTP Cores also necessarily operates at the CLB level. When a defective component (LUT, Mux etc) is detected in a CLB, the entire CLB is marked as unusable. While this approach makes less efficient use of resources than identifying individual defective components of the CLB, other smaller grained approaches could have been pursued. It was decided, however, that CLB level granularity was appropriate, given that it resulted in simplified software. The list to track defective components would only be a list of defective CLBs, rather than a more complicated list of heterogeneous resources.

One advantage of the RTP Core approach is that a single

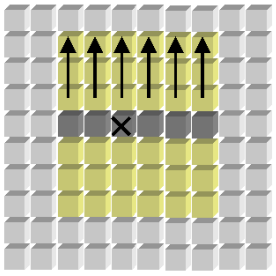

Figure 3: Skip Row Mode

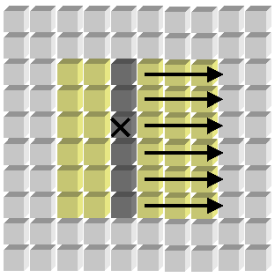

Figure 4: Skip Column Mode

overall scheme to provide defect tolerance is not required. Each RTP Core can use the most appropriate technique for its circuit or mode of operation. Three basic modes (*Skip CLB, Skip Row, Skip Column*) of constructing an RTP Core in the presence of a defective CLB are shown in Figures 2, 3 and 4. The mode is usually specified by an input parameter to the RTP core. In case of a defect in the *Skip CLB* mode, an entire CLB can be ignored and marked off as defective, with logic and routing flowing into the next CLB. Additionally, an entire row or column of CLBs may be marked off and treated as defective, in the *Skip Row* and *Skip Column* modes. These modes are used only to avoid defective CLBs and are not applicable to interconnect defects. The reasons for inapplicability of the mode while tolerating the defective wires is discussed in Section 6.

Each of these modes have different advantages which are typically dependent on the type of circuit being implemented. For example, if the circuit's output needs to be aligned in contiguous CLB locations in the output column, then *Skip Column* mode would be effective. On the other hand, if the circuit's CLB usage needs to be as minimum as possible, then *Skip CLB* mode would be useful. In general, these modes provides the desginer with various options to layout the design in the presence of defects and its up to designer's discretion to choose the mode of implementation based on the design constraints.

While RTP Cores may be constructed of other RTP Cores in a heirarchical fashion, these higher level cores do not have to be concerned with defects. Only the lowest level RTP Cores which are built from arrays of CLBs must be made to produce defect-tolerant circuits. While this approach may at first seem ad-hoc -with different, perhaps complex, schemes used to produce each defect tolerant RTP Core- this was not the case in practice. Nearly all RTP Cores are con-

structed by iteratively programming logic and interconnections in some parameter-based loop. For instance, a simple shift register is built by programming a chain of some number of linearly interconnected flip-flops. The number of stages in the shift register is typically a run-time parameter to the core, enabling shift registers of arbitrary sizes to be constructed on the fly.

If a faulty CLB is encountered while constructing such a shift register, for instance, all that is required is that the faulty cell be skipped and interconnection be performed to the next CLB. In this case, the additional software overhead of providing defect tolerance is a test for a defect and conditional statement for modifying a CLB index. However, the routing defects are tolerated by marking off the defective interconnects even before performing the routes for a RTPCore. In general, alongwith the marking of bad interconnects, the overhead of making an RTP Core defect tolerant is typically a test of the list of marked CLBs and a conditional incrementing of an index. The flow diagram in Figure 5 describes how the *Skip CLB* mode is implemented; *Skip Column* and *Skip Row* modes can also be implemented similarly.

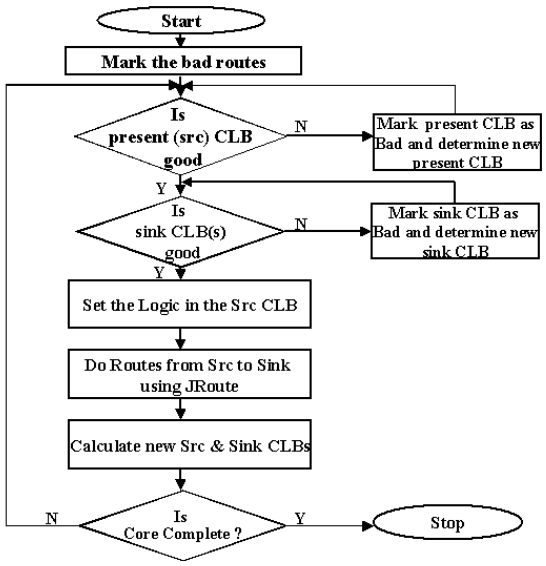

Figure 5: The Defect Tolerant RTP Core design flow.

Logic for a particular CLB is specified using *JBits* API while routing the interconnects is performed using *JRoute*. *JRoute*, a run-time routing API, is available in the *JBits* toolkit. Built on *JBits*, the *JRoute* API provides access to routing resources in a FPGA device. *JRoute* provides point to point routing at run time between RTP Cores and has various levels of routing control. This ranges from turning on or off a single connection to auto-routing between various sources and sinks. In this work, auto-routing is extensively used to route the defect tolerant RTP Core. The technique outlined in Figure 5 is based on assuring that both source and destination of all routes is established before interconnections are routed using the *JRoute* API.

5. A DEFECT TOLERANT CONSTANT MULTIPLIER RTP CORE

One RTP core that is especially common in RTR systems is a constant coefficient multiplier. Unlike a standard two input multiplier, one input is tied to a constant value in a constant coefficient multiplier. The multiplier folds the constant value into the circuit, producing a smaller, faster and more power efficient implementation. As a trivial example, a constant multiplication by a factor of two can be reduced to a simple shift operation. Other constant values can be reduced to similar combinations of shifts and additions.

A defect tolerant constant multiplier was designed adopting the flow described in Figure 5 for all the three modes of defect tolerance. The multiplier in this example is an eight bit input, sixteen bit output constant coefficient multiplier with a constant value of 75. This RTP Core was verified with other test inputs, but the diagrams in the remainder of this section uses a test input of 255 (0xFF), producing a final result of 19,125 (0x4AB5).

Although the Virtex device that was used in these experiment to verify the functionality of the multiplier did not contain any defective CLBs and wires, certain CLBs and wires were marked as bad in the defect database. Thus these defective CLBs were ignored in the configuration data and the multiplier was placed and routed around these defects. The functionality of the multiplier was verfied using the *Boardscope* debug tool which is available in the *JBits* tool kit.

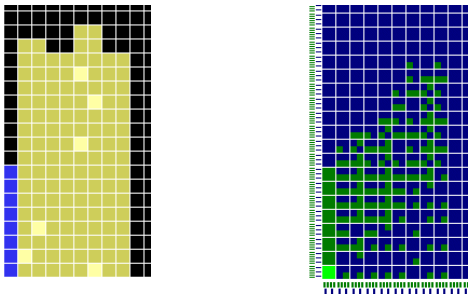

Figure 6: Boardscope core and state views for the SKIP CLB mode.

Figure 6 shows an implementation using the *Skip CLB* method of avoiding defects. This figure is derived from actual screen output from the *BoardScope* debug tool operating on a Virtex XCV800 device [12]. In *BoardScope* a FPGA device is represented as an array of CLBs with index (0,0) at the bottom left corner of the array. *BoardScope* has both a *Core* as well as a *State View*. While the *Core View* shows cores present in a design, the *State View* shows the state of all the flip-flops in a device. Each of the four flip-flops present in one Virtex CLB is represented by a quadrant of a CLB space in the *BoardScope State View* with state zero and one represented in blue and green color respectively.

In this example, a constant value RTP Core with a value of 255 is placed in the lower left corner of the CLB array at CLB(0,0). This value drives the Defect Tolerant Constant Coefficient Multiplier RTP Core placed just to the right at location CLB(0,1). Defective CLBs are marked at six locations, including CLB(0,6), CLB(1,1) and CLB(3,2). The *Core View* to the left in Figure 6 clearly shows the defective

CLBs. On the right, perhaps less clear than the *Core View*, the *BoardScope State View* shows the state of all flip flops in the device, with the unused "defective" CLBs containing flip-flops with all states set to zero. In addition, the output can be determined to be correct value of 19,125(0x4AB5). The output value of the multiplier can be interpreted from the state of lower right quadrant flip-flop value of each CLB present in the last column of the multiplier design.

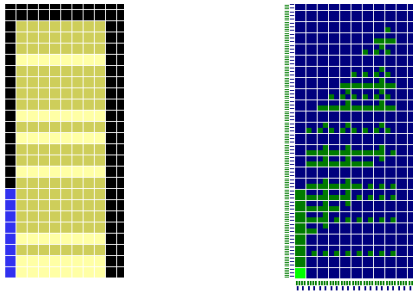

Figure 7: Boardscope core and state views for the skip row mode.

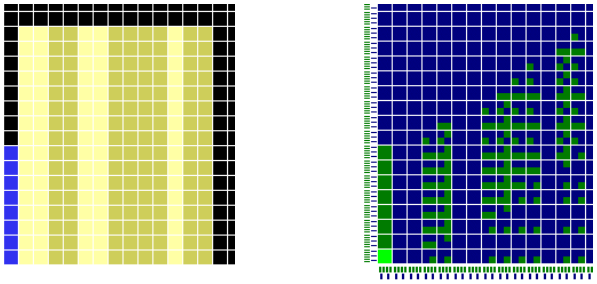

Figure 8: Boardscope core and state views for the skip column mode.

Similar to the Skip CLB mode, Figure 7 demonstrates the *Skip Row* mode while Figure 8 demonstrates the *Skip Column* mode. Here, defective CLBs result in an entire row or column of CLBs being ignored. These modes are useful in producing cores that may have more desirable geometries. Aligning all RTP Core outputs in a single column, even in the presence of faults may be desirable for layout purposes.

6. MARKING DEFECTIVE WIRES

While the model based on marking off defective CLBs is sufficient, there are often cases where ownership of a resource to a particular CLB is not obvious. For instance, single length lines that connect adjacent CLBs are not clearly part of either CLB. By convention, all wires can be assigned to a particular CLB, but this is not necessary in *JBits*.

Because a *JBits* RTP core uses the *JRoute* run-time routing API, a simple marking scheme for defective wires is already available. The *JRoute* API used to arbitrarily interconnect inputs and outputs in the device must keep track of the interconnect resources that are in use. In the existing implementation, *JRoute* begins by assuming that all routng resources in the device are free. As connections are made, *JRoute* updates the interconnection database. To mark off individual wires as defective in *JRoute* is fairly straight forward. The *JRoute* resource database is pre-loaded to make the defective wires appear to be already in use before routing the defect tolerant RTPCore. Thus these wires will not be included in the routing produced for a RTP core by the *JRoute* API.

This technique of marking off the defective wires require the availability of sufficient defect-free wires that can be used to route the design. Present FPGA architectures provide abundance of routing resources. For example, Virtex routing architecture has 24 single-length and 72 buffered Hex lines in each of the four directions per CLB. Direct and Long lines are also available to route the signals[15]. Thus even in the presence of few defective wires in a CLB, the router would have sufficient wires to route the signals.

Because a very large percentage of the device and a large percentage of the resources in present commercial FPGAs such as Virtex are interconnect resources, this pre-loading of the *JRoute* resource usage database is expected to address a large number of defects. This eliminates the need for marking off entire CLBs and large blocks of routing when only a single wire may be defective. Because this should greatly decrease the number of wires marked as faulty, routing around defects should be simplified. In the presence of interconnect defects only, the logic and other good routing resources can still be used without marking off the entire CLB as defective. It is for these reasons that the modes of skipping a defective CLB are irrelevant when tolerating the defective interconnect resources.

7. CONCLUSIONS AND FUTURE WORK

A technique has been demonstrated for providing defect tolerance in FPGA devices at the RTP Core level. This requires that the defect tolerance scheme used is explicitly specified by the RTP Core implementation. This approach has the advantage that no other additional software or modifications to device architectures are required. In addition, the software overhead of providing such defect tolerance is minimal and need only be provided for the lowest level cores in a heirarchical design. This technique is also unique in that it provides direct support for run-time reconfiguration. Circuits may be configured and re-configured transparently in the presence of defects.

While this approach is currently viable, there are still many new directions to be explored. We are currently actively exploring on-line run-time defect detection and isolation. This will permit defects to be found and logged at run-time. The combination of these techniques will permit systems to continue to operate in the presence of device faults, even very late in the lifetime of the system. This has advantages for various remote applications where physical repair or replacement of faulty components may be difficult or impossible.

Presently *JRoute* does not provide support for timing constraints but work is also being done on a static timing analyzer. This can be used to verify that timing constraints have not been violated by the presence of defects in a circuit. This tool will also work on-line and should complement the on-line techniques being explored. Finally, new Defect Tolerant RTP Cores are being built to further explore application level defect tolerance.

The techniques described in the paper permit FPGA devices to be configured and reconfigured in the presence of de-

fects. In addition to supporting run-time reconfiguration in the presence of defects, this *JBits*-based approach may also be used to configure devices in a more traditional static environment. Without re-compilation or any interaction with design tools, large numbers of defective FPGA devices can be rapidly configured. This provides a practical path for making use of defective devices in a traditional manufacturing environment. In addition to raising device manufacturing yields, this approach should also permit much larger FPGA devices to be produced and programmed. The largest practical device which could be produced and programmed by these techniques may in fact be limited only by the size of the silicon wafers used in the manufacturing process.

8. ACKNOWLEDGEMENTS

This work was supported by the U. S. Defense Advanced Research Projects Agency, under contract DABT63-99-3-0004. Authors are thankful for the advice and assistance from other members of the *JBits* project.

9. REFERENCES

[1] R. Cuddapah and M. Corba. Reconfigurable logic for fault-tolerance. In W. Moore and W. Luk, editors, *Field-Programmable Logic and Applications*, pages 380–388. Springer-Verlag, Berlin, August/September 1995. Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications, FPL 1995. Lecture Notes in Computer Science 975.

[2] W. B. Culbertson, R. Amerson, R. J. Carter, P. Kuekes, and G. Snider. Defect tolerance on the Teramac custom computer. In K. L. Pocek and J. Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 116–123, Los Alamitos, CA, April 1997. IEEE Computer Society Press.

[3] A. Doumar, S. Kaneko, and H. Ito. Defect and fault tolerance FPGAs by shifting the configuration data. In N. Harrison and C. Metra, editors, *Proceedings of the 1999 International Symposium on Defect and Fault Tolerance in VLSI Systems*, November 1999.

[4] J. M. Emmert and D. Bhatia. Partial reconfiguration of FPGA mapped designs with applications to fault tolerance and yield enhancement. In W. Luk, P. Y. Cheung, and M. Glesner, editors, *Field-Programmable Logic and Applications*, pages 141–150. Springer-Verlag, Berlin, September 1997. Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications, FPL 1997. Lecture Notes in Computer Science 1304.

[5] J. M. Emmert, C. E. Stroud, B. Skaggs, and M. Abramovici. Dynamic fault tolerance in FPGAs via partial reconfiguration. In K. L. Pocek and J. Arnold, editors, *IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2000.

[6] S. A. Guccione and D. Levi. Run-time parameterizable cores. In P. Lysaght, J. Irvine, and R. W. Hartenstein, editors, *Field-Programmable Logic and Applications*, pages 215–222. Springer-Verlag, Berlin, August/September 1999. Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications, FPL 1999. Lecture Notes in Computer Science 1673.

[7] S. A. Guccione, D. Levi, and P. Sundararajan. JBits: A java-based interface for reconfigurable computing. In R. Katz, editor, *Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, September 1999.

[8] F. Hanchek and S. Dutt. Methodologies for tolerating logic and interconnect faults in FPGAs. *IEEE Transaction on Computers*, pages 15–33, January 1998.

[9] F. Hatori and et al. Introducing redundancy in field programmable gate arrays. In *IEEE Custom Integrated Circuit Conference*, pages 7.1.1–7.1.4, 1993.

[10] N. J. Howard, A. M. Tyrrell, and N. M. Allison. The yield enhancement of field-programmable gate arrays. *IEEE Transactions on VLSI Vol 2 No 1*, pages 115–123, March 1994.

[11] E. Keller. JRoute: A run-time routing API for FPGA hardware. In J. Romlin et al., editor, *Parallel and Distributed Processing*, pages 874–881. Springer-Verlag, Berlin, May 2000. 7th Reconfigurable Architectures Workshop (RAW), part of the Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS) Workshops 2000 held in Cancun, Mexico May 1-5, 2000. Published in the series Lecture Notes in Computer Science 1800.

[12] D. Levi and S. A. Guccione. BoardScope: A debug tool for reconfigurable systems. In J. Schewel, editor, *Configurable Computing Technology and its use in High Performance Computing, DSP and Systems Engineering, Proc. SPIE Photonics East*, pages 239–246, Bellingham, WA, November 1998. SPIE – The International Society for Optical Engineering.

[13] J. F. McDonald, B. Philhower, and H. J. Greub. A fine grained, highly fault tolerant system based on WSI and FPGA technology. In W. Moore and W. Luk, editors, *FPGAs*, pages 114–126. Abingdon EE&CS Books, Abingdon, England, 1991.

[14] R. C. Minnick. A survey of microcellular research. *J. ACM*, 14(2):203–241, 1967.

[15] Xilinx, Inc. *Xilinx Data Book*, 2000.