Yield Enhancing Defect Tolerance Techniques for FPGAs

Asbjoern Djupdal
djupdal@idi.ntnu.no

Pauline C. Haddow
pauline@idi.ntnu.no

CRAB Lab (http://crab.idi.ntnu.no)
Department of Computer and Information Science
Norwegian University of Science and Technology

Abstract

As technology scales, the problem of production defects is expected to increase. This makes maintaining device yield a challenge. Also, it may be expected that more and more defect circuits will pass the production tests as the device testing challenge grows due to more and more transistors being compacted onto a single chip.

Reconfigurable technology has experienced an increasing popularity in recent years. Similar to ASIC design, reconfigurable technology suffers from production defects. However, unlike ASIC design, reconfigurable technology provides a bridge between production and the application designer. The inclusion of defect tolerance in the FPGA architecture could provide a functionally correct FPGA for the application designer, despite production defects. As such, the application designer is relieved of the extra complexity of designing for imperfect devices.

This paper presents a survey of known approaches to making defect tolerant FPGAs and discusses their advantages and disadvantages, especially in the context of maintaining FPGA yield and device correctness.

1 Introduction

Defect Tolerance for reconfigurable devices may be said to date back to the early 1990's. However, research focus moved more towards run-time fault tolerance in the late 1990s. Lately, renewed interest in defect tolerance has arisen due, in part, to increasing research into nano-computers where defect densities might be high.

Production of integrated circuits using optical lithography has always had problems with production defects, resulting in yield less than 100%. While these problems have been manageable, they are expected to increase. The ITRS roadmap states that yield enhancement will become a major challenge [12] and that "fabrication of chips with 100% working transistors and interconnects becomes prohibitively expensive."[11] They further state that *"relaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce costs of manufacturing, verification, and test."*[11]

There are several different causes of faulty operation in integrated circuits. Non permanent errors mostly occur from radiation. Permanent errors might be the result of radiation; wear-out phenomena such as electromigration and cracks from temperature fluctuations or from production defects. The latter is addressed in this paper with respect to yield. Contamination (dust) during the manufacturing process may result in production defects—typically shortened or broken wires. These defects are called random *particle defects*. Another kind of defect is *systematic defects* seen as malformed structures, often resulting from optical effects.

Reconfigurable technology, represented by field programmable gate arrays (FPGAs) has become more and more popular in recent years. Although originally a prototyping device, the FPGA today is, in addition, widely used as a component in the final product. Just like all other lithographically produced chips, FPGAs suffer from production defects. However, reconfigurable technology provides a bridge between chip production and the application designer. The inclusion of defect tolerance in the generic FPGA architecture would provide a functionally correct FPGA for the application designer, despite production defects. As such, the application designer is relieved of the extra complexity of designing for imperfect devices.

This paper presents a survey of known approaches for defect tolerant FPGAs and discusses their advantages and disadvantages, especially in the context of maintaining FPGA yield and device correctness. Only techniques targeting FPGA yield are discussed.

The rest of this paper is organised as follows: Section 2 gives the necessary background information on defect models, yield estimation and defect tolerance techniques in general. Section 3, 4, 5 and 6 present surveys of configuration approaches, architecture approaches using node redundancy, local redundancy and application specific FPGAs respectively. Finally, section 7 gives a summary and discussion.

2 Background

2.1 Defect Models

The common way to model production defects is to assume that each defect is like a disc with a certain diameter ranging from the minimum feature size up to some assumed maximum defect size. If a disc makes a short or an open on a die, it produces a fault on that die.

In order to estimate yield for a given chip design, several assumptions about the production process must be made. One assumption is the spatial distribution of defects on a wafer i.e. the degree of defect clustering. Another assumption is the defect size distribution, which is important as not all defects are large enough to make a fault in every chip design.

Based on a given chip design, the fault probability kernel — the probability that a defect of a given type and size results in a fault, may be found. Monte Carlo simulations on the chip layout, together with the defect size distribution, may be used to find the probability that a defect produces a fault. Knowing both the fault probability and having an assumption about the spatial distribution of defects, yield may be estimated. One, commonly applied yield equation is the negative binomial yield.

Estimating the yield of a given chip design is not very accurate. The problem lies in the strong dependency on both the design and the manufacturing process. Yield data from the manufacturing process is in general not available. The general spatial distribution of defects and the defect size distribution for a manufacturing process is very difficult to find, as it is typically only faults in a produced chip that is detected. Even when both design and manufacturing process are known, it is difficult to get accurate yield numbers without actually manufacturing the device. This is because of all the assumptions behind the calculations, hiding the complex physical circumstances that leads to defects. The best yield estimates are often found by having an older similar chip design with known yield numbers and then scaling the parameters for the yield equation.

While the most important work regarding production defects has to do with improving the manufacturing process itself, including defect tolerance in the specific chip design to be produced might allow the design to tolerate a certain amount of defect logic.

2.2 Defect Tolerance Techniques

Techniques for making defect tolerant designs involve some form of redundancy. Defect tolerance in hardware can be be achieved by either *static* or *dynamic* techniques.

Static redundancy is advantaged by the masking of faults without the need to detect them first. However, several equal modules implementing the same functionality are typically required, thus consuming a much larger die area than a non defect tolerant design. An example of such a technique is Triple Modular Redundancy (TMR) with tripled area and increased power consumption. Information redundancy is a static redundancy technique that involves adding redundant information to the original data. Information redundant techniques such as error correcting codes may be used efficiently with respect to die area, but only for parts of a typical hardware design.

To avoid reducing the total number of usable dies from a wafer (effective yield), area efficient defect tolerance techniques are needed. Dynamic redundancy has a mechanism for fault detection and actively recovers from the detected effect of a fault.

Dynamic defect tolerance techniques may be more area efficient than static techniques because there is no need to mask any possible faults. Only detected faults need special treatment.

When requiring area efficient defect tolerance, the typical approach is to exploit regularity in the design [15]. The problem with techniques like modular redundancy is that an enormous amount of redundancy must be introduced. A regular design may use the regularity to introduce only a small or moderate number of redundant elements and still give high defect tolerance. For example: The 16 bit Hyeti microprocessor [19] contains a bit sliced datapath with 17 slices, of which one is redundant. This datapath organisation has an area overhead of roughly $\frac{17}{16}$ and still makes the processor function correctly with defects in one of the slices. TMR would have the much larger overhead of roughly three times the original area. Two other better known examples are RAM ICs and hard drives where only a small amount of extra storage can be used to mask defects. The technique involves relocating data from the defective areas to the redundant ones.

Similar to RAM, the FPGA has a regular structure. This originally motivated the search for efficient defect tolerance techniques for FPGAs in the 1990's.

3 Configuration Approaches

This group of methods consists of the approaches that tolerate defects by introducing changes to the tool chain or chip configuration.

3.1 Chip-specific Bitfiles

Commercial tools for place-and-route have the option of specifying placement constraints. This means that the designer can specify which parts of the device are not to be used. This can be exploited for defect tolerance. If the device is first tested for defects, place-and-route can generate a bit file for the device that avoids the defective areas—see figure 1. Examples of this are Kumar et al. [16], the Teramac project [1], NanoFabric [22] and a somewhat different method using JBits [25].

Kumar et al. [16] described one of the first defect tolerance methods for FPGAs, with retesting of each chip to discover defects. Prior to configuration, changes were made to the layout so that defects were avoided without requiring a full place-and-route.

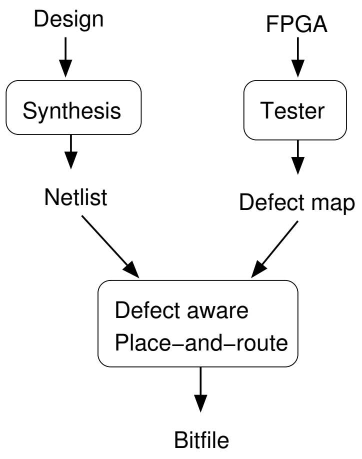


Figure 1: Chip specific bitfiles

The Teramac project [1] was a large custom computer with lots of partly defective FPGAs. One of their major contributions was to develop methods to precisely locate defects in generic FPGAs. In addition, they showed with real hardware that many partly defective FPGAs could be used successfully. The NanoFabric is a more recent project that resembles the Teramac approach but addressing reconfigurable nano-computers [22].

In Sundararajan and Guccione [25], JBits is used to generate run-time parametrisable cores for FPGAs. These cores are not fixed data objects, but code sequences describing how to construct circuits. Important design decisions like bus width can be decided at run-time enabling circuits to be modified while running. By including defect testing and avoidance in the core generating process, defect tolerance is achieved.

Chip-specific bit files provide a high degree of flexibility with respect to creating a new layout that avoids defects. As testing and reconfiguration are conducted offline (except for the JBits approach), little on-line resources is required.

However, chip specific bit files are not harmonious with mass-production. Each device produced must have its own bit file, making it resource demanding to create the bit files for high volume products. In addition, distributing firmware up-

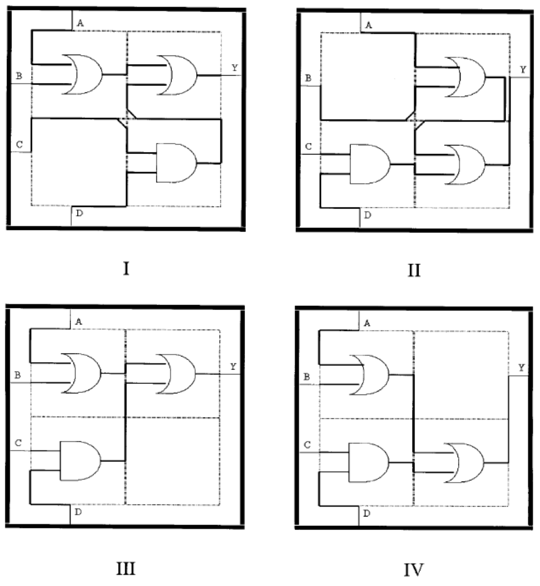


Figure 2: Precompiled configuration [17], showing one tile in the FPGA.

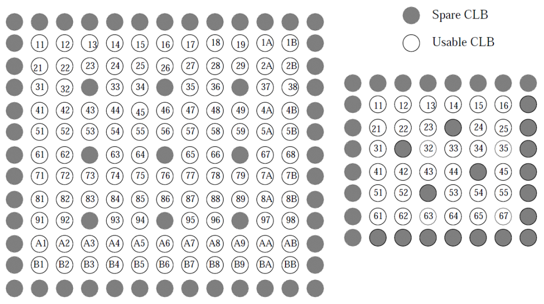


Figure 3: Shifting entire design [3]. Two examples of how spare nodes may distributed on the FPGA.

grades to end-users is difficult as each end-user needs a tailor made bit file. The Jbits approach does not suffer from the same problems because the method is designed for run-time parametrisable systems.

3.2 Precompiled Configuration

A set of different configurations, may be compiled. with the aim that at least one of these solutions will function correctly in the presence of a defect. In the example shown in figure 2, taken from Lach et al. [17], the FPGA is divided into *tiles* of $2 \cdot 2$ CLBs each. For each tile, one CLB is chosen as a spare and four configurations are made where the spare CLB has different positions. When a specific chip is to be configured with a defect in a tile, the tile configuration that does not use the defective CLB is chosen.

The advantage of this method compared to chip specific bit files is that there is no need to run place-and-route for each chip despite the fact that defects in the individual chips can still be avoided. One disadvantage is larger bit files i.e. an increased need for external storage. This disadvantage could be reduced with bit file compression, as described by Huang and McCluskey [10]. Another disadvantage is reduced flexibility in how many defects that can be covered. As an example, the method used in figure 2 can at most tolerate one defect in each tile.

3.3 Adaptive Configuration

3.3.1 Shifting Entire Design

Doumar et al. [3] has an interesting approach where defect tolerance is achieved by embedding spare nodes into the design as well as shifting the entire design vertically and/or horizontally. The chip is tested at power-on and if a defect is found, the entire design is shifted such that a spare node covers the defect.

There are several possible ways of embedding spare nodes into the design. Two possible examples are shown in figure 3. The left one shows how to embed spare nodes so that any single defect can be covered by shifting at most one step horizontally and/or one step vertically. The right example shows how to embed spare nodes so that at most one shift step, either horizontally or vertically, is needed to cover a defect.

The advantage of this approach is the simplicity of the relocation algorithm. No rerouting is required (except very simple rerouting at chip I/O pins). This method can however only cover one defect (unless very lucky in the location of defects) at the expense of a relatively large number of spares.

3.3.2 Dynamic Place-And-Route

A radical approach to defect tolerance is to have an adaptive way of "growing" circuits onto a defective medium e.g. the Cell Matrix reconfigurable

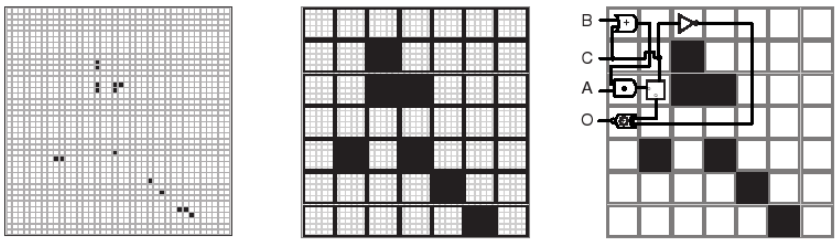


Figure 4: Cell Matrix [20]. Black areas are defective. Left: Initial Cell Matrix. Middle: After stage 1—Configuration of supercells. Right: After stage 2—supercells have found an implementation of target circuit.

device [5]. Cell Matrix is self-reconfigurable — each logic block has the possibility to reconfigure its neighbours. Macias and Durbeck [20] describes an adaptive configuration process for Cell Matrix that takes care of chip testing and placement and routing avoiding defective regions.

Figure 4 illustrates the configuration process starting with a defective chip (left). The first stage in the configuration process is to configure the Cell Matrix as a matrix of superblocks—defect free $n \cdot n$ logic blocks. Once a superblock is configured, it starts testing neighbouring $n \cdot n$ blocks. If found to be defect free, the block becomes a superblock (holding a netlist of the desired circuit), otherwise it is marked as defect.

In the second stage, a decentralised and distributed place-and-route algorithm takes place that results in each superblock being assigned a part of the target circuit and communication paths are set up between relevant superblocks (right part of figure 4).

The distributed wavefront method of doing testing and configuration of superblocks (stage 1) has the advantage of having some degree of parallelism. This could be important in a nano-computer context where the size of the reconfigurable array prevents a more conventional and slower sequential form of configuration. The distributed algorithm for doing place-and-route (stage 2) is currently sequential, removing much of the speed advantage of this method, but future work hopefully results in a distributed and parallel place-and-route algorithm.

A disadvantage of this method is the area requirement for a full copy of the netlist in each supercell and the inefficiency inherent in decentralised place-and-route.

4 Architecture Approaches through Node Redundancy

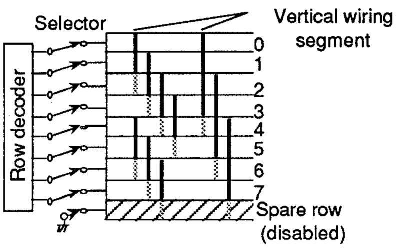
Nodes in the FPGA architecture may be reserved as spare nodes, together with the necessary resources for making the spare nodes take over for defective ones.

4.1 Redundant Row and/or Column

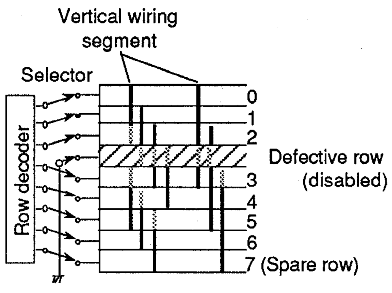
The FPGA may be made defect tolerant with the use of redundant rows and/or columns. One of the rows and/or columns are reserved for spare nodes and if a defective node is found in one of the normal rows or columns, that row or column is bypassed and the redundant row or column is put into use. Variants of the redundant row/column method have been investigated by Hatori et al. [8], Howard et al. [9], Durand and Piguet [4] and Shibayama et al. [24].

Hatori et al. [8] presented this method for the first time. Figure 5(a) shows an FPGA with a spare row. Horizontal wiring is unmodified, whilst vertical wiring segments span one extra row. In the case of a defect, shown in figure 5(b), the defective row is disconnected, vertical wiring is set up to bypass the disconnected row and all lower rows are shifted one row down.

Howard et al. [9] introduce a variant of this which they call a block structured defect tolerant FPGA. The idea is to divide the array of nodes into several large blocks of nodes and removing all global signals. This is similar to having several independent and interconnected FPGAs. Defect tolerance is achieved with redundant rows and columns of these large blocks. The rationale behind the block



(a) Without defective row



(b) With defective row

Figure 5: Row redundancy [8]

structuring is that many defects are unlikely to be confined to the node it occurs in (fault containment), instead they are likely to affect the whole array. The block structured FPGA will confine any defect within the block. In addition, timing within a block will be unaffected. They use Monte Carlo simulations to back up their claims about lack of fault containment in typical FPGAs.

Durand and Piguet [4] use a binary decision tree based FPGA with redundant columns. In a binary decision tree, only neighbouring test cells are connected together, which is reflected in their FPGA architecture where only neighbouring cells may communicate. This simplifies the logic needed to bypass defective columns. Each cell has two identical configuration registers with parity bits, holding the configuration data for that cell so as to tolerate defects in the configuration registers. When defects are detected at run-time, the configuration data of all columns to the right of the defect is shifted to the right, bypassing the defective column. To avoid shifting in the full column height, the reconfigurable array is divided horizontally into several subarrays with routers in between. Shifting to avoid a defect in a subarray can then be confined to within the subarray.

Shibayama et al. [24] have a physical implementation of a defect tolerant FPGA with both a spare row and a spare column. Run-time self checking triggers shifting and bypassing of rows and columns.

Advantages of the redundant row or column approach is the simplicity in defect avoidance by turning off and bypassing rows. This could be implemented using laser-blown fuses at the factory (as suggested by Hatori et al. [8]) or with run-time self reconfiguration logic, making the method completely transparent to the customer. The high overhead is a disadvantage, entire rows or columns are discarded for single defects. Extra switches and longer wires used for row or column bypassing lead to longer routing delays.

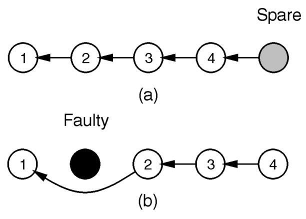


Figure 6: Node covering [7]

4.2 Redundant Single Nodes

Instead of invalidating entire rows or columns, schemes exist where a single redundant node can take over the functionality of a defective one. This can be implemented by introducing a number of spare nodes in each row and if there is one defective node in a row, nodes in that row are relocated so that the defective node is unused. With one spare node in each row, each row can tolerate one defective node.

Hanchek and Dutt [7] describes this method, which they call *node covering.* Relocating nodes in a row when a node is defect is achieved as in figure 6, where nodes to the right of the defective node are shifted one step further to the right. Spare wiring segments (cover segments) exist that allow signals to bypass defective nodes and to support correct routing between rows after a row has been restructured due to a defective node. The same technique is used for wiring segments between switch blocks or for entire "grids" (set of interconnectable tracks, one track from each horizontal and vertical channel). The node covering method was extended to a dynamic method with less overhead, where no extra cover segments are used. Instead, interconnect resources are incrementally rerouted and if necessary, layout is incrementally modified [6, 21].

A variant is described by Kelly and Ivey [13]. Off-chip testing generates a defect map for inclusion in the configuration bitstream. When configuration starts, on-chip modifications are performed on the array according to the defect map so that defective nodes are avoided in the same manner as in Hanchek and Dutt [7].

Compared to the redundant row or column approach, the general method of using redundant single nodes is less wasteful in that a single defective node does not discard the entire row or column. This method may thus be able to tolerate more defects in total than the redundant row or column approach. Some form of post-production configuration could be performed by the factory to make defect hiding transparent to the user. Just as for the redundant row or column method, there is extra delays due to switches and longer wires. A disadvantage is the extra complexity in the routing between rows which gives higher total overhead than the redundant row or column method.

5 Local Redundancy

5.1 Modifications to CLBs

Local modifications to a CLB can make the CLB more robust against production defects. The behaviour of a CLB is implemented using look-up tables (LUTs) and these can be made defect tolerant using error correcting codes. This has been suggested for use in the Cell Matrix [23] and in the NanoBox project [14]—a reconfigurable array for nanotechnology.

Another possibility is to allow inputs to a LUT to change place. If a LUT is only using three of its four inputs and there is one defect bit in the LUT SRAM memory, swapping inputs can be used to avoid activating the defective bit [18].

An advantage of error correcting codes in the LUTs is the ability to mask defects in the LUT using a static redundancy method. A separate testing phase is not necessary for this to work. Another advantage is the independence of LUTs — a defect in one LUT will not preclude the masking of defects in another. This is unlike the methods using a limited amount of spare nodes where there is a limited number of defects in total that can be tolerated. A disadvantage is that it requires extra logic in every LUT all over the FPGA.

The advantage of swapping LUT inputs is that resources unused in a specific design on the FPGA can be used for defect tolerance in a simple way. A disadvantage is the limited use of the method—this method can only be used for defects that occur in LUTs that happen to have unused inputs. This reduces the usefulness compared to error correcting codes.

5.2 Modifications to Switch Blocks and Local Routing

Local modifications to the routing system can be used to tolerate defects in switch blocks or wiring segments. Doumar and Ito [2] add an extra wire to the switch block, making it possible to connect any two wires, thus have the ability to bypass a faulty switch. Yu and Lemieux [28] have multiplexors on the inputs and outputs of a switch block, together with spare lines between switch blocks. With corresponding changes in connection blocks, this can then be used to bypass defective wires between switch blocks, which is quite similar to the switch blocks in Hanchek and Dutt [7].

Xu et al. [27] introduce extra wires and switches and a routing procedure to replace faulty CLBs using the spare routing capacity.

An advantage of these techniques is that interconnect defects can be tolerated without doing complex rerouting of the design on the FPGA. As a large part of the chip area of a modern FPGA is occupied by interconnect, many defects are likely to occur in the interconnect.

6 Application Specific FPGAs

One approach to using partly defective FPGAs is to make the defects fit the application. This is called *Application Specific FPGA* (ASFPGA) and is provided by Xilinx in their EasyPath program [26]. The idea is to that the designer creates a bitfile as usual and then Xilinx selects defective FPGAs that function with the given bitfile.

The advantage here is that there is no need for any change in the FPGA architecture, bitfile, tool chain or methods used when developing the design. The disadvantage is that the reconfigurability aspect is lost or reduced because the FPGA is not guaranteed to work with any bitfile.

7 Discussion

Table 1 gives an overview of all surveyed techniques for achieving defect tolerance in FPGAs with respect to different criteria., discussed below.

Defect coverage Defect coverage is the ability to tolerate defects. The ultimate goal of doing defect tolerance is to be able to tolerate all defects that may occur. This is not possible and only a limited number of defects and a limited set of defect types can be tolerated by any defect tolerance technique. As can be seen in table 1, defect coverage is highest for the techniques involving a full place-and-route after defects have been detected; chip specific bitfiles and the dynamic place-and-route method of the Cell Matrix. In addition, from a customer point of view, the application specific FPGAs have perfect defect coverage in that no FPGAs will have defects affecting the target application.

The redundant row/column technique has low defect coverage due to the inability to tolerate defects in more rows or columns than there are spare ones, typically only one. Similarly, the technique of shifting the entire design can in most situations tolerate only one defect.

Area overhead This criterion is represents the amount of extra chip area needed for redundancy. This is not applicable for chip specific bitfiles and application specific FPGAs because these techniques do not require allocation of spare resources.

The area overhead is high for the precompiled configuration technique because there is one spare node for every tile. The dynamic place-and-route of the Cell Matrix has also a very high area overhead due to the requirement of storing the entire netlist in every supercell.

The method of shifting the entire design is also quite expensive in terms of chip area due to the large number of spares needed.

Node redundancy techniques have low area overhead because the number of redundant nodes is small. The area overhead for the local redundancy techniques is also small — neither error correcting codes nor extra lines between switch blocks take a lot of space.

Timing overhead Timing overhead is low for most of the configuration techniques because the signals paths will only be marginally larger. An exception is the dynamic place-and-route method of the Cell Matrix that will have a large timing overhead simply because the supercells are so physically large, making signals travel far.

Timing overhead is medium for the node redundancy techniques due to the extra wire mengths and switches needed to bypass faulty nodes.

There might be some timing overhead for the local redundancy techniques, but it is likely to be small. Extra multiplexors on the switch blocks might, however, contribute significantly to the interconnect delay.

Bitfile size Most of the surveyed techniques have no effect on the bitfile size, compared to a typical non defect tolerant FPGA. The exceptions are the precompiled configuration technique that has several configurations for each tile and the dynamic place-and-route technique where the bitfile might be smaller as only the supercell configuration and a netlist of the target circuit is needed.

Extra hardware required Several of the surveyed techniques need extra chip area in the form of on-chip support of the technique itself. The techniques of chip specific bitfile and application specific FPGAs do not have any on-chip hardware support whereas all others have some. However, a large portion of each supercell, in the cellmatrix,

Table 1: Overview of surveyed techniques

(a) Configuration techniques

	Chip specific bitfile	*Precompiled*	*Adaptive configuration*	
			Shifting	*Dynamic PAR*
Defect coverage	High	Medium	Low	High
Area overhead	—	High	Medium/ high	High
Timing overhead	Low	Low	Low	High
Bitfile size	Medium	High	Medium	Low
Extra HW required	—	Low	Low	High
Maturity	High	Medium	Medium	Low
Mass production friendly	Low	High	High	High

(b) Other techniques

	Node redundancy		*Local redundancy*	*ASFPGA*
	Redundant row/col	*Single nodes*		
Defect coverage	Low	Medium	Medium	High
Area overhead	Low	Low	Low	—
Timing overhead	Medium	Medium	Low	—
Bitfile size	Medium	Medium	Medium	Medium
Extra HW required	Low	Medium	Low	—
Maturity	High	Medium	High	High
Mass production friendly	High	High	High	High

is dedicated to performing the configuration algorithm. The single redundant nodes have a moderate amount of support hardware, more than the redundant row/column approach, because of the more complex routing support between rows.

Maturity This criteria refers to how close this technique is to be put into use i.e. in a commerical setting. The dynamic place-and-route method is perhaps the least mature furthest from being finished and perhaps another technology base (huge reconfigurable nanoarrays) to be useful. The chip specific bitfile and the application specific FPGA techniques are the most mature and have been or are in use today. In addition, error correcting codes in LUTs and the node redundancy techniques have been researched by several research groups for several years and are relatively mature.

Mass production friendly This criteria reflects how well the technique suits FPGAs that are to be used in mass produced end-user products. The only technique that really does not suit mass production is the chip specific bitfile technique due to the problems associated with a tailor made bitfile for every end-user product.

All presented techniques rely on exploitation of the generality and regular structure of the FPGA, either at the top level or lower levels. At the top level, the generality of the FPGA components are exploited. Every CLB is equal and can, therefore, take over the functionality of a defective one. The regular structure of the CLBs makes it possible to use simple techniques such as redundant row for defect avoidance. At CLB level, the regularity of the look-up tables is exploited by introducing error correcting codes. Similarly, regularity in the interconnect makes it possible to introduce redundant wires.

Although one of the local methods (error correcting codes in LUT memory) does not require any post-production modifications. The other methods do to actively avoid a detected defect.

The effect of production defects on FPGAs affects the kind of defect tolerance techniques that can be used. Howard et al. [9] shows that fault containment should be studied for FPGAs designed for defect tolerance. In a typical FPGA, a significant amount of defects will have so large effect that they can not be tolerated. This indicates that approaches without any architectural changes to the FPGA might not be the best way to achieve good defect tolerance. Still, some real world examples

(Teramac and Xilinx EasyPath) demonstrate the usefulness of these methods.

The defect distribution is also importance with respect to tech suitability of different defect tolerant methods. If defects distribute in a highly clustered fashion, the defect tolerance method used should be able to tolerate more than one defect. If there is one defect on a die, there are probably other defects on the same die. Similarly, which parts of the FPGA consumes large areas on the chip. There is little point in concentrating efforts on making defect tolerant CLBs if the interconnect occupies most of the die area.

Bibliography

[1] W. B. Culbertson, R. Amerson, R. J. Carter, P. Kuekes, and G. Snider. Defect tolerance on the teramac custom computer. In *Proc. IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM)*, page 116, 1997.

[2] A. Doumar and H. Ito. Design of switching blocks tolerating defects/faults in FPGA interconnection resources. In *IEEE Symposium on Defect and Fault-Tolerance*, pages 134–142, 2000.

[3] Abderrahim Doumar, Satoshi Kaneko, and Hideo Ito. Defect and fault tolerance FPGAs by shifting the configuration data. In *Proc. International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 377–385, 1999.

[4] Serge Durand and Christian Piguet. FPGA with self-repair capabilities. In *Proc. International ACM/SIGDA Workshop on Field Programmable Gate Arrays*, 1994.

[5] Lisa J. K. Durbeck and Nicholas J. Macias. The cell matrix: an architecture for nanocomputing. *Nanotechnology, Institute of Physics*, 2001.

[6] Shantanu Dutt, Vimalvel Shanmugavel, and Steve Trimberger. Efficient incremental rerouting for fault reconfiguration in field programmable gate arrays. In *ICCAD*, pages 173–176, 1999.

[7] Fran Hanchek and Shantanu Dutt. Methodologies for tolerating cell and interconnect faults in FPGAs. *IEEE Transactions on Computers*, 47(1):15–33, 1998.

[8] F. Hatori, T. Sakurai, K. Nogami, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, H. Muraoga, A. Tanaka, and K. Kanzaki. Introducing redundancy in field programmable gate arrays. In *Proc. IEEE Custom Integrated Circuits Conference*, pages 7.1.1–7.1.4, 1993.

[9] Neil J. Howard, Andrew M. Tyrrell, and Nigel M. Allinson. The yield enhancement of field-programmable gate arrays. *IEEE Transactions on Very Large Scale Integration (VLSI)*, 2(1):115–123, mar 1994.

[10] Wei-Je Huang and Edward J. McCluskey. Column-based precompiled configuration techniques for FPGA fault tolerance. In *Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 137–146, 2001.

[11] ITRS. Design. Technical report, ITRS, 2005.

[12] ITRS. Lithography. Technical report, ITRS, 2005.

[13] Jason L. Kelly and Peter A. Ivey. Defect tolerant SRAM based FPGAs. In *Proc. IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD)*, pages 479–482, 1994.

[14] AJ KleinOsowski and David J. Lilja. The NanoBox project: Exploring fabrics of self-correcting logic blocks for high defect rate molecular device technologies. In *IEEE Symposium on VLSI*, pages 1–10, 2004.

[15] Israel Koren and Zahava Koren. Defect tolerance in VLSI circuits: Techniques and yield analysis. *Proceedings of the IEEE*, 86(9):1819–1837, sep 1998.

[16] Vijay Kumar, Anton Dahbura, Fred Fischer, and Patrick Juola. An approach for the yield enhancement of programmable gate arrays. In *International Conference on Computer-Aided Design*, pages 226–229, 1989.

[17] John Lach, William H. Mangione-Smith, and Miodrag Potkonjak. Low overhead fault-tolerant FPGA systems. *IEEE Trans. Very Large Scale Integr. Syst.*, 6(2):212–221, 1998. ISSN 1063-8210. doi: http://dx.doi.org/10.1109/92.678870.

[18] Vijay Lakamraju and Russell Tessier. Tolerating operational faults in cluster-based FPGAs. In *Proc. International Symposium on Field Programmable Gate Arrays*, pages 187–194, 2000.

[19] R. Leveugle, Z. Koren, I. Koren, G. Saucier, and N. Wehn. The Hyeti defect tolerant microprocessor: A practical experiment and its cost-effectiveness analysis. *IEEE Transactions on Computers*, 43(12):1398–1406, 1994.

[20] N. J. Macias and L. J. K. Durbeck. Adaptive methods for growing electronic circuits on an imperfect synthetic matrix. *Biosystems*, 73(3): 173–204, 2004.

[21] Nihar R. Mahapatra and Shantanu Dutt. Efficient network-flow based techniques for dynamic fault reconfiguration in FPGAs. In *International Symposium on Fault-Tolerant Computing*, pages 122–129, 1999.

[22] Maham Mishra and Seth C. Goldstein. *Nano, Quantum and Molecular Computing, Implications to High Level Design and Validation*, chapter 3: Defect Tolerance at the End of the Roadmap. Kluwer Academic Publishers, 2004.

[23] C. R. Saha, S. J. Bellis, A. Mathewson, and E. M. Popovici. Performance enhancement defect tolerance in the cell matrix architecture. In *Proc. International Conference on Microelectronics*, pages 777–780, 2004.

[24] Atsufumi Shibayama, Hiroyuki Igura, Masayuki Mizuno, and Masakazu Yamashina. An autonomous reconfigurable cell array for fault-tolerant LSIs. In *Proc. IEEE International Solid-State Circuits Conference*, pages 230–232, 1997.

[25] Prasanna Sundararajan and Steven A. Guccione. Run-time defect tolerance using JBits. In *Proc. FPGA*, pages 193–198, 2001.

[26] Xilinx. EasyPath FPGAs. `http://www.xilinx.com/products/easypath`.

[27] Jian Xu, Weikang Huang, and Babrizio Lombardi. A novel fault tolerant approach for SRAM-based FPGAs. In *Proc. Pacific Rim International Symposium on Dependable Computing*, pages 40–44, 1999.

[28] Anthony J. Yu and Guy G. F. Lemieux. Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement. In *Proc. Field Programmable Logic and Applications*, pages 255–252, 2005.