Defect Tolerance for Yield Enhancement of FPGA Interconnect Using Fine-grain and Coarse-grain Redundancy

by

Anthony J. Yu

B.A.Sc., The University of British Columbia, 2001

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

THE FACULTY OF GRADUATE STUDIES

Electrical and Computer Engineering

The University of British Columbia

August 2005

Abstract

Field programmable gate arrays (FPGAs) are integrated circuits (ICs) designed to implement, or be programmed with, any user circuit. This unique ability makes FPGA extremely popular; however, it also introduces a significant amount of area and delay overhead to the circuit. Fortunately, FPGA are typically manufactured in a process that is two to three generations ahead of the one used by application specific ICs. This allows some reclaiming of area and delay lost due to the programmability. However, the problem with being this far ahead is manufacturing defects appearing in immature technologies. The aggressive scaling of feature sizes and the migration to new technologies makes the manufacturing of perfect FPGAs increasingly unlikely. Utilization of defect-tolerant techniques is one method of alleviating this growing problem. Defect-tolerance enable defective FPGAs to appear as "perfect." This thesis presents and compares two new approaches to FPGA defect-tolerance: fine-grain redundancy (FGR) and coarse-grain redundancy (CGR). FGR has an array-size-independent overhead of up 50%, and is capable of tolerating an increasing number of defects as array size grows. In constast, CGR, at low defect levels, demonstrates a diminishing amount of area overhead as array size increases. At low defect levels, CGR requires less area overhead than FGR; however, in situations where more than 2–3 defects are expected, FGR requires less overhead.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Field-programmable gate arrays (FPGAs) are large integrated circuits comprised of programmable logic blocks and programmable routing. Their size, density requirements and regular layout makes them attractive for aggressive tuning in the latest technology processes. As such, they are also prone to manufacturing defects [5, 32, 33, 34].

The number of manufacturing defects is expected to increase as the density of FPGAs increases, or as the programmable logic paradigm migrates to new technologies such as nano-technology [12]. This increase in defect rates severely impacts the viability of programmable devices. It also highlights the importance of incorporating defect tolerance strategies into FPGAs.

Modern FPGAs are predominantly programmable routing. Defects are thus more likely to appear in the routing resources as opposed to the programmable logic blocks. This makes the ability to tolerate defects in the interconnect extremely important. In this thesis, two different approaches to interconnect defect-tolerance are presented and compared. The interconnect encompasses the physical wiring, switch elements, configuration bits in both the switch block and connection block.

Traditional methods of defect tolerance involve the use of a spare row and column in the FPGA architecture [18]. As shown in Figure 1.1, defects are tolerated by bypassing the defective row/column, and shifting part of the design to the spare row/column. This coarse-

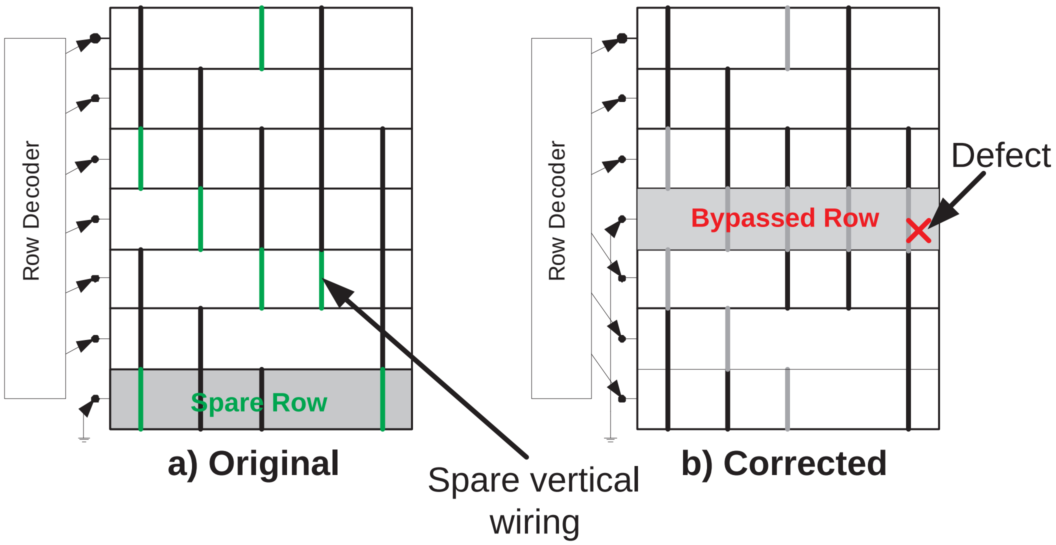


Figure 1.1: Overview of Coarse-Grain Hardware Redundancy

grain approach is capable of tolerating defects in routing and logic blocks. However, the consolidation of spare resources into a single row or column limits its ability to efficiently tolerate multiple *distributed* defects.

This thesis presents a new architecture that embodies a fine-grain approach to defect tolerance. Spare resources are distributed across the FPGA fabric. This allows the architecture to tolerate multiple distributed defects in the interconnection of an FPGA. In the proposed architecture, defect tolerance is based on shifting individual connections. As shown in Figure 1.2, shifting allows signals to route around a defect. This is called fine-grain redundancy.

A comparison between traditional coarse-grain redundancy (CGR) and fine-grain redundancy (FGR) is also presented. The comparison will show that FGR provides a scalable solution that is better at tolerating more defects as larger FPGAs are made. CGR, with its smaller area overhead at current FPGA dimensions, is adequate for now.

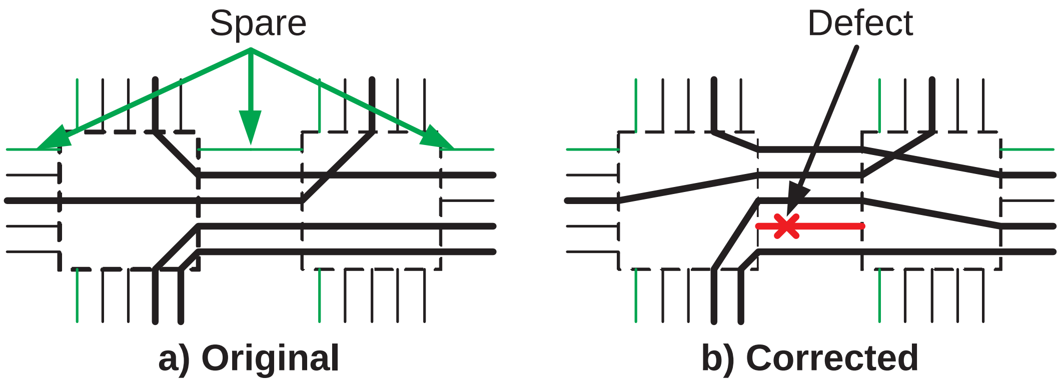


Figure 1.2: Overview of Fine-Grain Hardware Redundancy

1.1 Motivation and Objectives

A marketable FPGA is one where it can be programmed with any user design. Defects inhibit this ability and cannot typically be sold. Thus, yield lost due to defects represents a lost of revenue for FPGA vendors. The desire to maximize revenue suggest the need to minimize yield lost due to defects. This thesis explores one such way to improve yield; it investigates the use of defect-tolerant architectures for yield enhancement.

A viable defect-tolerant architecture should be capable of a) efficiently tolerating multiple random distributed defects in the interconnection, b) preserving the signal timing characteristics of the original circuit, and c) computing defect corrections quickly. These attributes are desirable for the following reasons. The first is important because random distributed defects are expected to contribute to the highest yield loss [33], and that the number of defects is expected to increase with the scaling of technology [5]. Next, to guarantee the correctness of a defect-corrected circuit, drastic changes to the routing solution cannot be allowed as the changes can lead to unanticipated timing violations, race conditions and skew. Lastly, the ability to apply defect correction in a timely manner is essential in a manufacturing environment since multiple circuit boards must be quickly programmed with the FPGA bitstream.

Unfortunately, a survey of current defect-tolerant approaches revealed that there is no such architecture capable of handling all of the abovementioned features. This then sets the stage for this thesis. The objective of this research is development and comparison of defect-tolerant architectures that are capable of tolerating multiple defects, capable of rapid defect correction and capable of not altering signal timing.

1.2 Contributions

This thesis proposes two defect-tolerant architectures that are capable of tolerating multiple random distributed defects. The first architecture, fine-grain redundancy (FGR), is scalable and can tolerate an increasing number of defects as chip density or area increases. Additionally, defect correction can be applied quickly and does not affect signal timing. Although not shown in this thesis, it is recognized that the architecture can also be used to repair crosstalk type faults.

In contrast, the second proposed architecture embodies coarse-grain redundancy (CGR) and utilizes multiple spare rows and columns to attain defect-tolerance. Although CGR has been published before [18], the framework for handling multiple spares is new. The CGR redundancy scheme does not scale as well as FGR. However, it will be shown later that the use of multiple spare rows and columns is better suited for current FPGA dimensions.

FGR and the multiple spare rows and columns schemes are the first architectures to address all of the aforementioned desired features of defect-tolerance. Namely, both are capable of a) efficiently tolerating multiple random distributed defects in the interconnection, b) preserving the signal timing characteristics of the original circuit, and c) computing defect corrections quickly.

Yield models for FGR, multiple spare rows and columns CGR and traditional CGR are presented and compared. The comparison is based on 4 factors that influence yield: switch implementation, switch flexibility, array size and wire length.

In summary, the major contributions of this paper include the following:

- Presentation of a new fine-grain defect-tolerant architecture and its yield model,
- Presentation of a new coarse-grain multiple spare row and column architecture and its yield model,
- A detailed study of the area and delay overhead required for hardware-based defect-tolerant interconnect scheme in FPGAs, and
- A comparison between fine-grain and coarse-grain redundancy.

This work has been published in two FPGA conferences [36, 37] and is currently being evaluated by the University-Industry Liaison Office of the University of British Columbia for patent opportunities.

1.3 Outline

This thesis is organized as follows. Chapter 2 presents an overview of modern FPGA architectures and describes previous approaches to FPGA defect redundancy. Chapter 3 describes the new the fine-grain redundancy architecture (FGR). The traditional coarse-grain redundancy architecture (CGR) and a new multiple-spare rows and columns architecture are presented in Chapter 4. A comparison between the yield of the FGR and CGR is shown in Chapter 5. Finally, conclusions are given in Chapter 6.

Chapter 2

Background

This chapter presents the architectural detail and terminology needed in the discussion of FPGA defect tolerance. First, a brief overview of modern FPGA architecture is presented. This is followed with the defining of defect, fault, test and diagnosis. Finally, a summary of past FPGA defect-tolerance approaches are presented.

2.1 FPGA Architecture

An FPGA is an integrated circuit composed of programmable routing resources and programmable logic resources. The programmable logic, called configurable logic blocks (CLBs), are composed of k-input lookup tables and flip-flops. Each lookup table can implement any k-input logic function, and connects with a designated flip-flop. Together, the lookup table and flip-flop pair form a basic logic element (BLE). Figure 2.1 shows a CLB with I inputs and N BLEs.

Programmable routing can be further divided into three parts: the wires, the switch blocks (S blocks) and the connection blocks (C blocks). Together, the logic blocks, wires, S blocks and C blocks for modern commercial FPGAs are organized into an island-style architecture as shown in Figure 2.2. This architecture has proven to be very successful in modern FPGA architectures [10, 22]. As such, the island-style architecture was also used as the foundation for the new defect-tolerant architecture.

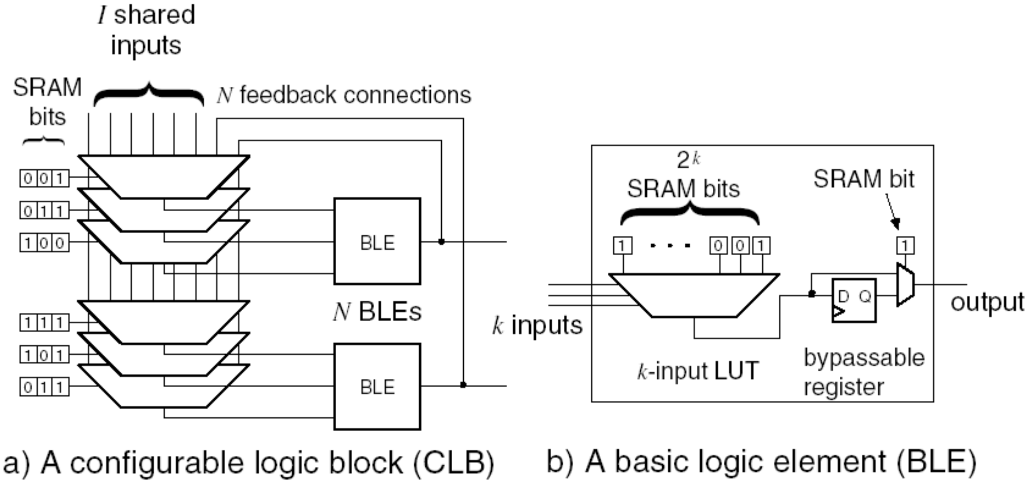


Figure 2.1: Overview of Fine-Grain Hardware Redundancy

Wires within an FPGA reside in routing channels and are indexed by track numbers. A *channel*, as shown in Figure 2.2, spans the width or height of the FPGA and has its boundaries defined by the CLBs. A wire's track number is based on its position relative to the width of the channel. The convention of this paper is that wires at the bottom-most/left-most position in each channel is assigned a track number of 0. Similarly, wires at the top-most/right-most position are assigned a track number of $channel\_width - 1$.

Modern FPGAs utilize single-driver wiring [25], where each wire is driven by a single tapered buffer connected to an input multiplexer. Figure 2.3 presents a wire that spans 3 switch blocks and uses single-driver wiring. Note that this wire can only be driven at its start point, hence a "directional" attribute is instilled upon the wire. To minimize the effects of the single-driver wiring, wires are added in pairs, one for either direction. The adopted convention in this thesis and [25] is that even tracks contain signals that move left/down, and odd tracks contain signals that move right/up.

S blocks are formed at the intersection of horizontal and vertical channels. An S block allows nets to turn corners or extend further down the channel. They also allow for net fanouts. Figure 2.4 shows both a detailed and high-level representation of the directional

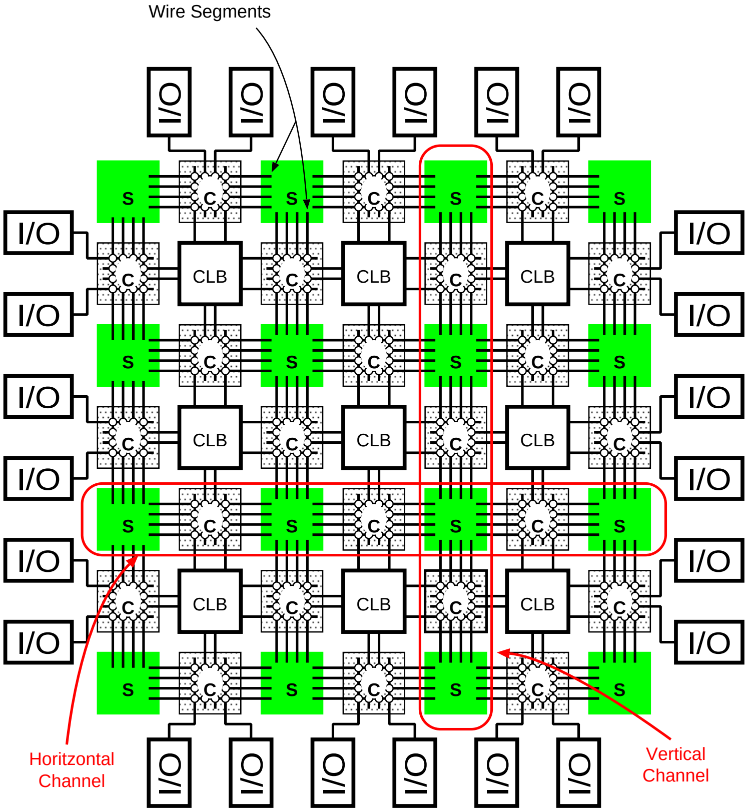


Figure 2.2: Island-style Architecture

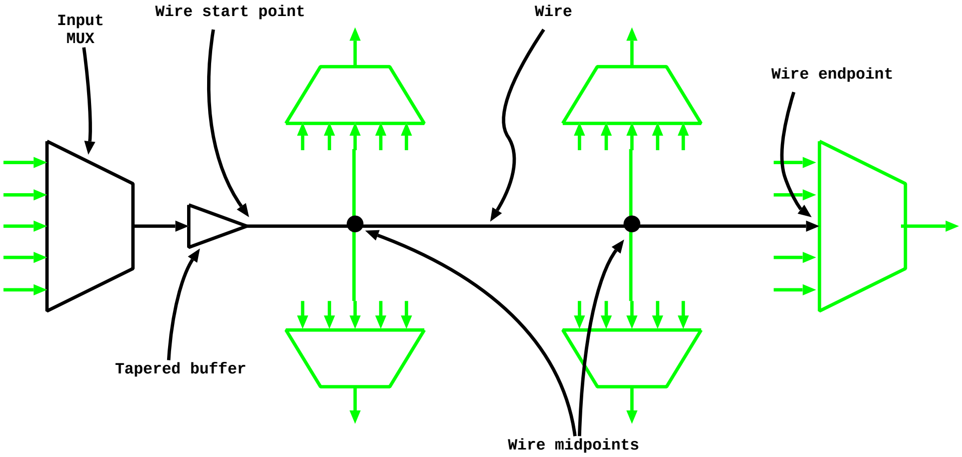


Figure 2.3: Directional wire (L = 3)

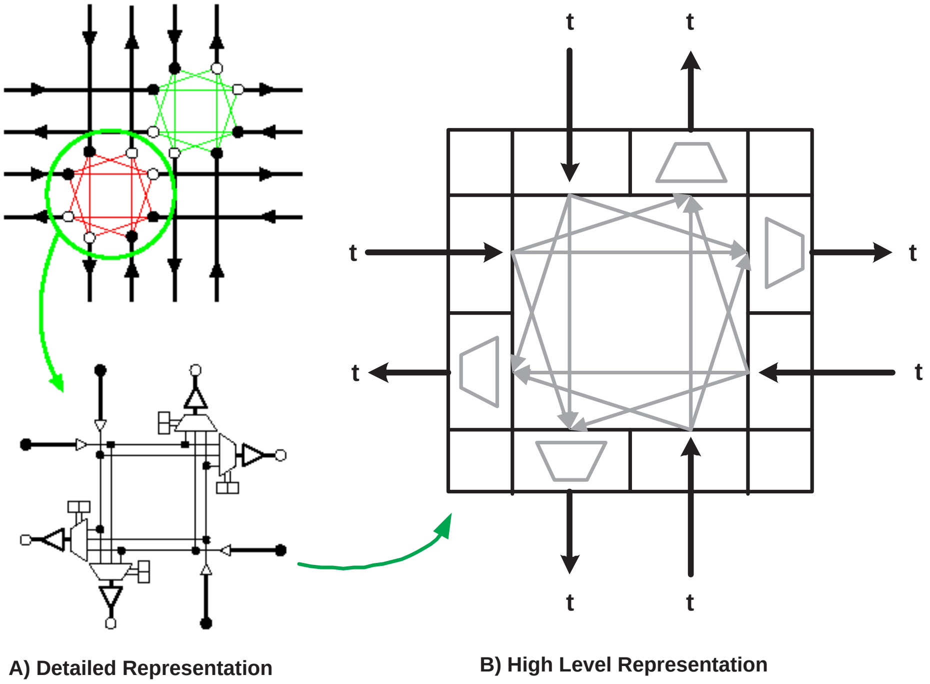


Figure 2.4: Directional switch block

switch for length 1 wires. In the high-level representation, a group of wires and buffers are replaced by single arrows.

The C block provides the interface between the CLB and the wires. Since wires can only be driven from its start point, the outputs of the CLB must connect directly to the input multiplexers of the wires which start nearby. The inputs of the CLB are also selected from wires in the routing channel at designated taps. These taps connect to input multiplexers of the CLB. Figure 2.5 shows an example of a C block for length 1 wiring. For clarity, only a few S block connections are shown in the figure.

2.2 Definitions

This section introduces a few definitions of terms used throughout this thesis.

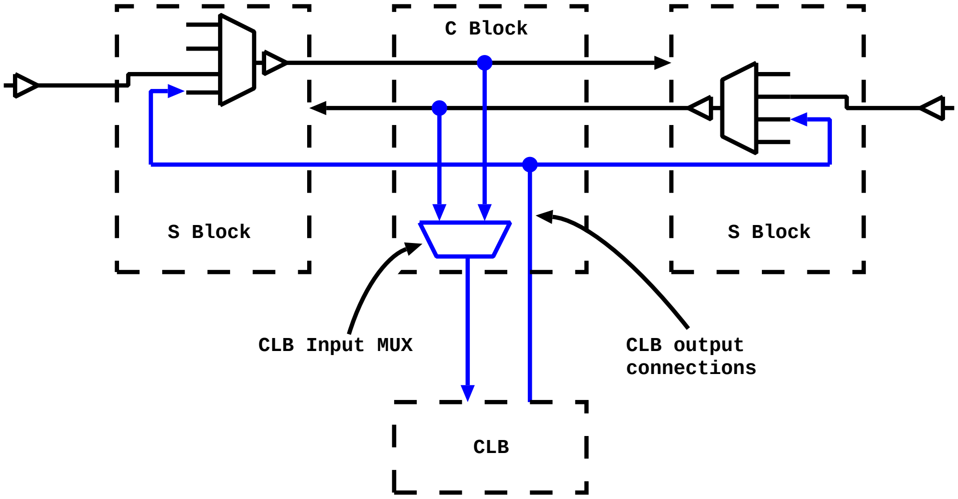


Figure 2.5: CLB input connection

2.2.1 Defects vs. Faults

Processing and mask imperfections can create physical imperfections in silicon. These imperfections can lead to variations in the electrical behaviour of wires and transistors [29]. In an extreme case, the imperfections can result in functional failure and unexpected circuit behaviour. This thesis defines a *defect* as an imperfection that causes a functional failure.

Faults are not physical imperfections, but rather models that encapsulate the behaviour of defects. They model defects at the highest level of abstraction [16]. Several different types of defects can be modeled by a single type of fault, such as stuck-at-1 faults. Abstraction simplifies test design by reducing the number of individual defects that must be considered.

Unlike defects, faults can also be transient in nature. That is, faults can be used to describe errors that occur during the operation of the FPGA. This includes errors caused by single-event upsets (SEUs) [17] and crosstalk [15]. This thesis is not concerned with transient faults, although FGR can be used to avoid some crosstalk problems.

2.2.2 Testing vs. Diagnosis

Testing is the process of determining if a circuit is operating as designed. This is typically accomplished by applying some stimuli to the circuit input and observing its output. A properly tested chip will produces a clear “yes, the circuit is working” and “no, the circuit is not working” answer.

Testing requires time and money. To facilitate testing and reduce costs, specialized test structures can be incorporated into the circuit. An example of this is built-in self test (BIST) structures.

Diagnosis involves the identification (i.e., determining the type) and location of defects [8]. When compared to test, diagnosis is a more complex and time-consuming process. This is because diagnosis demands the identification and location of defects in addition to a pass/fail result. Diagnosis for FPGAs is typically performed by exhaustively exercising all resources, and observing its output [28, 30]. This thesis will assume that some type of diagnosis can be done in a cost-efficient manner to identify all defects in a device. Hence, each device is assumed to have a defect map. This map can be rather simple.

2.3 Previous Redundancy Approaches

Defect tolerant approaches for FPGAs can be loosely classified into 3 groups. These are software redundancy, hardware redundancy and run-time redundancy. Each approach has its advantages and typically trade off between time (critical path delay and processing/application time) and resources (silicon area, external storage, etc.). In the following subsections, examples from each redundancy approach will be presented, and its advantages and disadvantages outlined.

2.3.1 Software Redundancy

In the software redundancy approach, CAD tools are used to map around defective resources. This method typically has no hardware overhead; however, the application of defect correction may take a long time. Furthermore, the effectiveness and efficiency of defect correction is heavily dependent on the abilities of the CAD tools.

In general, software redundancy is impractical in a production environment for two reasons. First, generating a unique placement and routing solution for each FPGA is a time-consuming process. Second, it is impractical to fully verify timing for each solution because the timing characteristics of the new placement and routing solutions will be different. For some high-performance circuits, it may be impossible to meet very stringent timing requirements with every new placement and routing solution. Despite these disadvantages, software redundancy does have some advantages, notably minimal hardware overhead and the ability to efficiently tolerate more defects than most other approaches.

Swapping and Incremental Rerouting

The approach proposed by [24] addresses defects in two ways. For defects in CLBs, defective resources are swapped with unused resources. This approach is based on the premise that the resources within an a CLB are never fully utilized, and that the logic within the CLB can be permuted such that the defective resource is avoided. For defects in routing wires, an incremental congestion-aware router is used to reroute the signals affected by the defects. By limiting the number of nets that are rerouted, the impact on signal timing is minimized.

For high-performance designs, it may not always be possible to find a new routing solution that meets the stringent timing requirements of the design. This is also the case for defects in logic blocks. For dense designs, it may not be possible to find an unused resource necessary for swapping. Despite these shortcomings, the approach is attractive because it requires no area overhead and can potentially tolerate a large number of defects.

Design Shifting

Another method for defect avoidance requires the reservation of spare resources. By carefully avoiding the use of certain resources, it is possible to avoid defects by “shifting” the

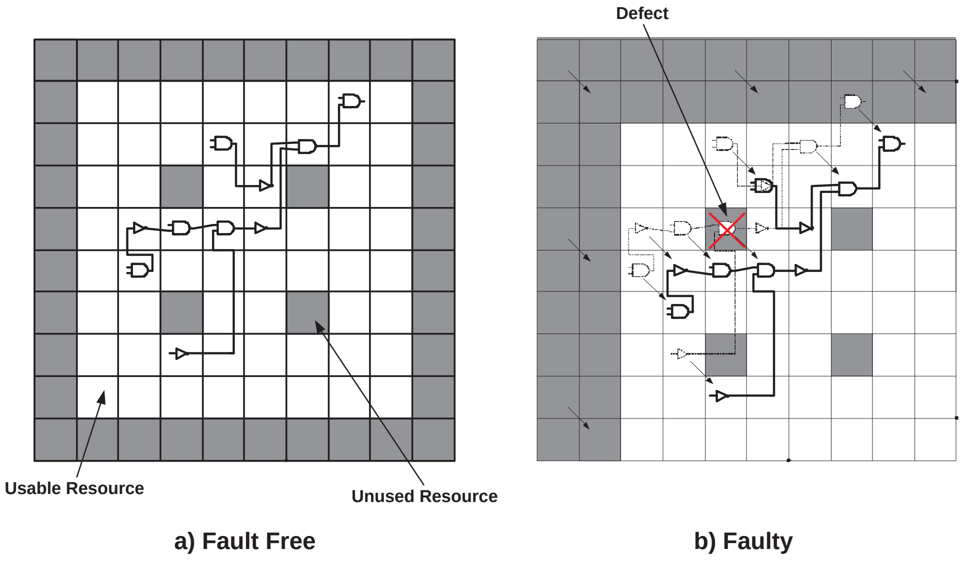


Figure 2.6: Design shifting for defect correction

entire design by one row or column in the array [14]. Figure 2.6 shows an example of defect correction using this approach.

Design shifting can be applied in a short amount of time since it only requires bitstream manipulation. However, without hardware support, the shifting results in a slight variance in I/O timing. It can also be complicated by heterogeneous (memory or DSP) blocks in the array. Furthermore, to support multiple defects, the defects must be perfectly aligned with the spare locations.

Precomputing Designs

To reduce the time needed to correct defects, a number of placement and routing solutions for the same design can be precomputed. Each solution differs in resource usage. When programming a defective device, defect correction involves selecting the appropriate solution - one that does not use the defective resources(s) [20, 23].

The clear advantage of this approach is that it does not have any on-chip hardware overhead and requires a relatively short defect correction time. The disadvantage is that there are many possible design permutations. This approach requires a lot of computing time (even if it is done beforehand) and a lot of storage space for the different design permutations.

EasyPath

Xilinx, a major vendor of FPGA integrated circuits, has a unique approach for dealing with defective parts. Customers are asked for their bitstream. In return, Xilinx provides the customer with a set of FPGAs that are guaranteed to work with the provided bitstream. The vendor can make this guarantee because they ensured that the customer design is not using any defective resources on the device sold. In essence, rather than forcing the configuration bitstream to avoid the defective resources, defects are forced to avoid the bitstream [35].

This approach is advantageous in a sense that customers can purchase discounted FPGAs that correctly implements their design. For mature designs, this translates to a significant cost reduction. The same cannot be said about new designs. Since new designs are subject to design revisions, when the place and route solutions is changed, it is unlikely to work on the faulty chip.

2.3.2 Hardware Redundancy

Hardware redundancy requires the addition of extra or spare resources. These spare resources facilitate defect correction by allowing the use of a defective resource to be shifted over to a spare one. This shifting reduces correction time since the time needed for shifting is typically less than the time needed to generate a new placement and routing solution.

The disadvantage of this approach is the need to incorporate redundancy at an early design stage of the FPGA itself. Hardware redundancy also costs area overhead and typically tolerates fewer defects than the software counterpart. Despite this, the approach is effectively used in industry [2, 3, 11]. This suggests the advantages outweigh the cost in area overhead. It also suggests that tolerating a large number of defects is not yet required in today’s technology!

Spare Rows and Columns

The spare row and column architecture is one of the first published FPGA defect-tolerant architectures [18]. One spare row and one spare column are incorporated into the FPGA. If a defect exists in one row or column, the defective row/column is bypassed, and the following row/column are shifted until they utilize the spare. This architecture can naturally tolerate clusters of defects in the same row/column, but not if the cluster spans two rows/columns. It has also been successfully applied in industry [2, 3, 11].

The weakness of this architecture lies in its *coarse-grain* nature. In the event of any defect, an entire spare row/column is utilized for correction. Thus, this architecture cannot tolerate multiple distributed defects. The advantage of this architecture is that it requires very little correction time. Industrial designs typically implement the row/column decoder such that rows/columns can be permenantly enabled and disabled (e.g., fuses can be blown). This reconfiguration can be made transparent to the user so the original bitstream can still be used on this defective FPGA.

A more in-depth analysis of this architecture will be presented in Chapter 4. In particular, an extension will be shown to generalize this approach for multiple spares.

Spare Connections

The architecture proposed in [13] incorporates spare wires and switches into the switch block design. As shown in Figure 2.7, the spare resources allow any defective transistor to be bypassed. This architecture can tolerate one defective pass-transistor per switch block.

One problem with this architecture is its impact on signal timing. Because of its length and heavy loading, the spare connections are slower than regular connections. As a result, defect correction introduces a significant timing variance in the routing solution. Another problem is the inability to tolerate bridging type faults between wires. The advantage of this architecture is its *fine-grain* approach to defect correction. By repairing defects at the transistor level, this architecture can tolerate multiple defects in the routing network.

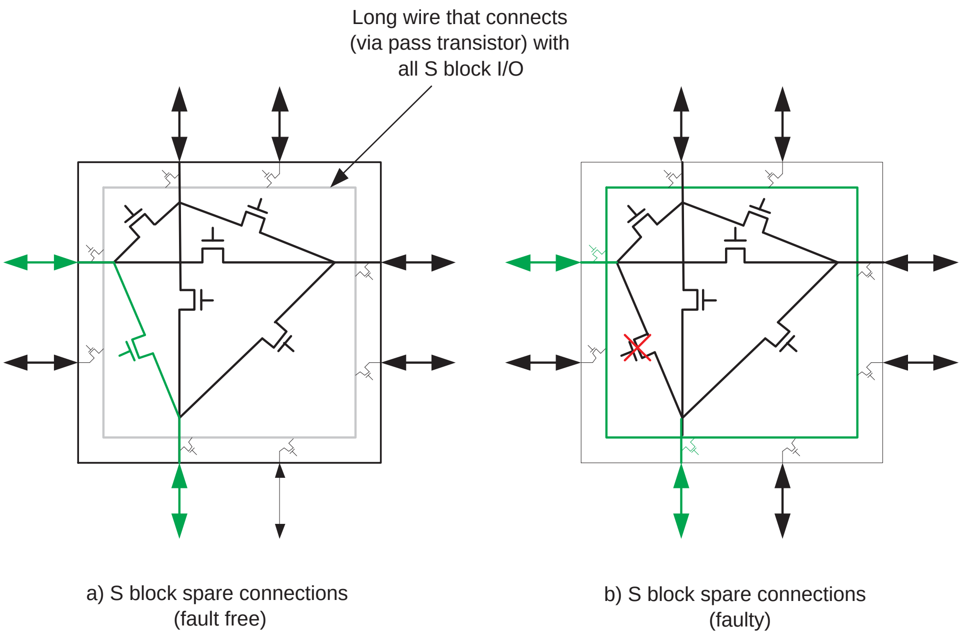

Long wire that connects
(via pass transistor) with
all S block I/O
a) S block spare connections
(fault free)
b) S block spare connections
(faulty)

Figure 2.7: Switch block with spare connections

2.3.3 Run-time Redundancy

Run-time redundancy deals with errors that occur when the FPGA is in operation. These errors include transient errors as well as permanent errors such as the ones resulting from electromigration. Transient errors, also called SEUs, are not permanent defects. They result when a radiation particle strike a memory element with sufficient energy to cause a change in the memory's stored value [17]. SEUs can be corrected by reprogramming the memory element. In contrast, the errors cause by electromigration cannot be corrected through reprogramming alone. It was observed in [19] that the stresses of carrying electrical signals can cause breakages in interconnect. This phenomenon is called electromigration and results in a permanent open circuits in the interconnect.

Run-time redundancy has the disadvantage of needing both hardware and software overhead. Additional diagnosis circuitry is incorporated into the design to detect inconsistencies arising from the errors. If an error is detected, CAD tools are used to correct the

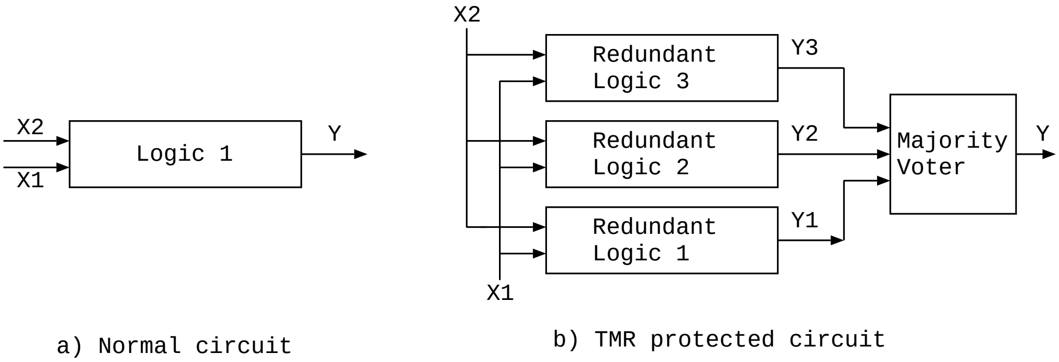


Figure 2.8: Triple-modular Redundancy

circuit (either by mapping around the defect or reprogramming the defective component). The advantage of this approach is that defect diagnosis occurs during the operation of the design. Defect correction can also occur during run-time as long as the underlying FPGA architecture supports partial reconfiguration. This reduces the impact of defect correction and eliminates “down time”.

n-modular Redundancy

In n-modular redundancy, the circuit that desires defect protection is replicated n times[1] and their outputs redirected to a voter circuit [6, 21]. The voter circuit performs a bit-wise majority vote on the outputs of the replicated circuits. In a defect-free environment, the outputs of the n circuits are identical, thus the output of the voter is identical to its input. However, if the voter circuit detects an inconsistency in the replicated output values, for example if the circuit experienced an transient error, the output will be that of the majority input value. The use of odd values of n ensures than ties are not possible. The replication(s) that exhibited the inconsistency may be taken offline, reprogrammed, and then reenabled [7]. Figure 2.8 shows an example of triple-modular redundancy.

The advantage of this approach is that defect detection and correction can occur while the circuit is still in operation. This approach can also be scaled. A higher level of

[1] n is usually odd, ie. $n = 3, 5, 7 \ldots$

protection can be attained simply by increasing the value on n. The clear disadvantage of this architecture is its high area overhead. Defect correction also requires that the FPGA being used has the ability to perform partial reconfiguration. If not, defect correction requires the reprogramming of the entire FPGA. Furthermore, it is unclear whether this approach can be used to tolerate multiple permanent defects.

Roving STAR

Roving STAR addresses defects that occur during the lifetime of the FPGA [1]. This approach divides the FPGA into two areas, a spare area (STAR) and the working area. The STAR is used for tests and diagnostics, while the working area contains the design. Each area contains a number of spare unused resources. These spare resources are used for defect correction.

During the operation of the FPGA, the STAR shifts across the FPGA by copying the configuration and state of the perspective STAR location (pSTAR) into the current STAR location (cSTAR), and enabling and disabling cSTAR and pSTAR respectively. The new STAR location is then reprogrammed to run tests and diagnosis on itself. Defects are catalogued and saved in memory. When the next shift is about to takes place, the configuration of the next perspective STAR location is manipulated so that spare unused resources are used in place of the defective ones. The shifting process continues until the entire FPGA is tested.

Like n-modular redundancy, Roving STAR requires that the underlying FPGA supports partial reconfiguration. It also requires the reservation of spare resources, and the incorporation of a reconfiguration manager. The shifting of design blocks can also affect signal timing for inter-block communication. Nevertheless, this approach is advantageous because testing and diagnosis is performed during run-time, and that the reconfiguration manager can dynamically change the configuration to avoid defects. The latter allows the design to continue operating in the presence of unanticipated permanent defects.

2.4 Summary

FPGAs are composed of programmable logic and programmable routing. These structures are arranged in a regular pattern and are susceptible to defects. FPGA manufacturers must perform tests on FPGAs to ensure that the chips they sell are “defect-free.” Additionally, diagnosis is sometimes required to apply defect correction (i.e., spare row and column).

Defect tolerant techniques have been developed to increase the number of usable FPGAs. Each technique varies in both its ability to tolerate defects and the amount of overhead needed. In one extreme, the software redundancy approach can tolerate a large number of defects with little or no area overhead, but costs a significant amount of computing time or storage space. In the other extreme, the hardware redundancy approach can perform defect correction quickly, but costs area overhead. Further, hardware redundancy approaches typically tolerate fewer defects than software redundancy approaches. The existence of the two extremes suggest the presence of an architecture that can both tolerate a number of defects in addition to the ability to preform defect correction quickly. The subsequent chapter presents one such example.

Chapter 3

Fine-grain Redundancy (FGR)

Rather than consolidating the spare resources into rows and columns, the proposed fine-grain redundant architecture (FGR) contains spare resources that are distributed evenly across the FPGA. This approach to defect tolerance allows the architecture to tolerate multiple randomly distributed defects.

3.1 Architectural and Implementation Details

FGR builds upon the island-style directional wiring architecture described in [26]. The original architecture is not defect-tolerant. This section will present the changes needed to make it defect-tolerant.

3.1.1 Switch Block Changes

To add defect-tolerance to the original directional switch block, two layers of multiplexers are wrapped around the switch block. This is shown as the two outer layers in Figure 3.1. The outer-most layer represents the shift-avoid layer of multiplexers (*omux*), and the middle layer represents the shift-restore layer of multiplexers (*imux*).

The *omux* allows signals to “steer” away from a downstream defect. By means of these multiplexers, signals routed on track t can be shifted up to tracks t+1 or t+2. When there is a defect on track t, the defect is avoided by shifting up all signals routed on tracks

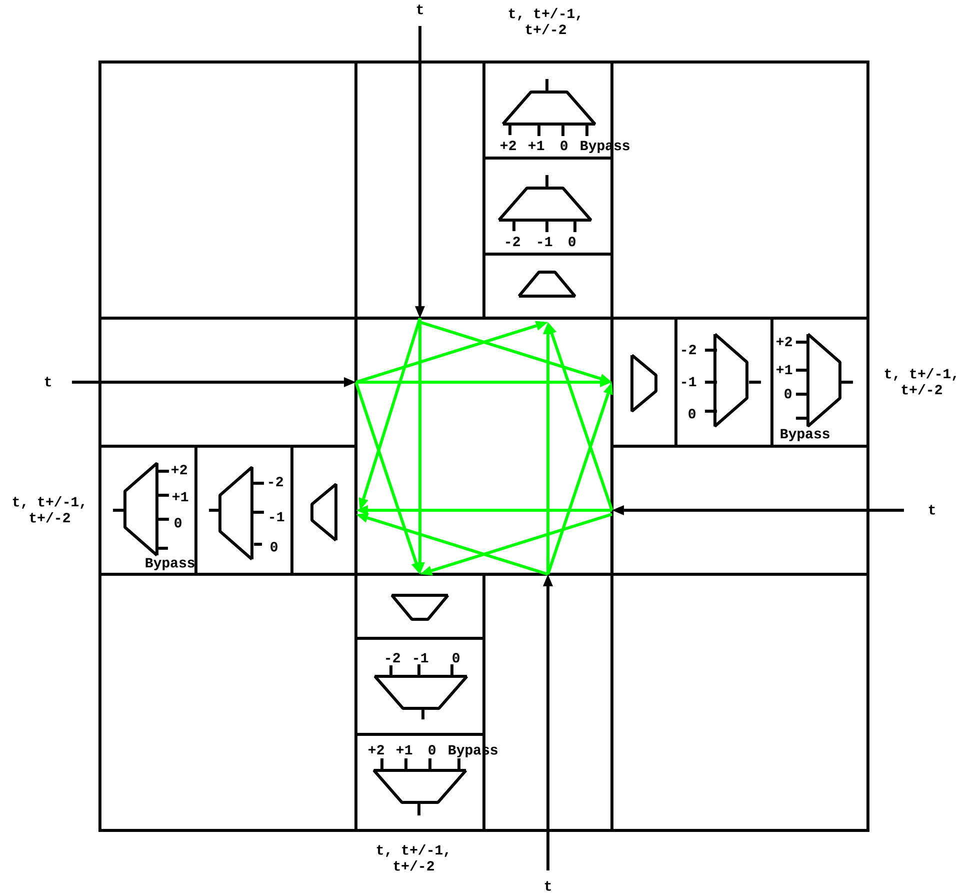


Figure 3.1: High-level defect-tolerant switch block

$\geq$ t. Signals on tracks $<$ t remain in place. Clearly, the shifting requires that there be spare routing tracks. As will be shown later, these spares incur about 10% area overhead for each spare set. Because this is a fixed overhead amount, the percentage tends to decrease as FPGA dimensions (i.e., channel widths) are scaled up.

The *imux* is used to reverse or restore the shift-avoid action taken by an upstream switch block. These multiplexers allow a signal on track t+1 or t+2 to shift down to track t, thereby nullifying any upstream shifting action. To keep the effects of track shifting localized, the switch block was designed such that any signal leaving a perturbed neighbourhood can be restored to the original track number. This localization allows our architecture to tolerate multiple distributed defects.

To reduce the delay of long nets, a bypass path similar to [27] is introduced into the switch block. This bypass path connects a straight-through wire endpoint directly with the corresponding *omux* on the opposite side of the switch block. This reduces the multiplexer depth per wire from 3 down to 1. Note that the bypass path is optional: it creates an alternate path for signals travelling across the channel by skipping the imux and normal directional switch.

In an attempt to reduce area and delay overhead, a reduction in switch block flexibility, F_s, was considered. F_s is the number of other wires connected to a wire at a given switch block [31]. By decreasing F_s, the number of potential connections available to a signal is reduced. This in turn reduces the number of inputs on the input multiplexer and thus improves both area and delay. With long wires, the flexibility at the end switch blocks or *endpoints* can be different than at the middle switch blocks or *midpoints*. We considered the following switch flexibilities:

1. The *E3M2* switch is the directional switch described in [26]. It has $F_s = 3$ for endpoints and $F_s = 2$ for midpoints. This allows endpoints to form connections with interconnects on the left (left turn), right (right turn) and opposite side (straight-through) of the switch block, and midpoints to form connections with interconnects on the left and right side of the switch block respectively.

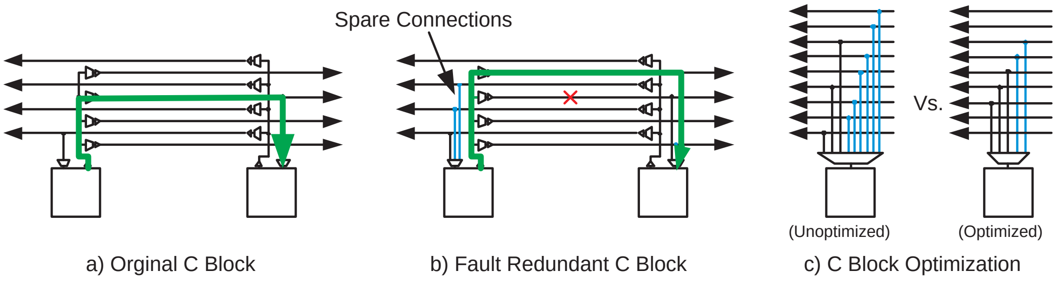


Figure 3.2: Connection block design

2. The *E3M1* switch also uses $F_s = 3$ at endpoints. However, midpoints are reduced to $F_s = 1$, meaning they can only turn either left or right (not both). The turn direction alternates along the length of a wire.

3. The *E2M1* switch has $F_s = 2$ for endpoints and $F_s = 1$ for midpoints. Endpoints include the straight-through connections and a left or right turn, while midpoints can only turn left or right. Turns are handled in the same manner as *E3M1*.

3.1.2 Connection Block Changes

As a consequence of track shifting, signals that were once routed on track t can now reside on tracks t+1 or t+2. To accommodate for this variability, the connection block must also be modified. In FGR, the CLB outputs do not need to be modified because they are already fully connected to all of the tracks. However, the CLB input connectivity must be increased by adding the connections required to the shifted tracks. This modification is shown in Figure 3.2ab.

Initially, the CLB inputs are connected to half of the routing tracks. This amount of connectivity was shown to provide a suitable trade-off between routability, area and delay [4, 26]. To adjust for the track shifting, for every track t that is originally connected to a CLB input, tracks t+1 and t+2 must also be connected to the input (assuming these connections don't exist already). Thus, if a signal gets shifted up by 1 or 2, the CLB can still extract the correct signal. Clearly, area overhead can be reduced by maximizing the number of consecutive tracks that are connected to a particular CLB input, as shown in Figure 3.2c. However, this optimization is left for future work. Ultimately, in FGR, the CLB inputs are connected to slightly more that half of the routing tracks.

3.1.3 Supported Defects

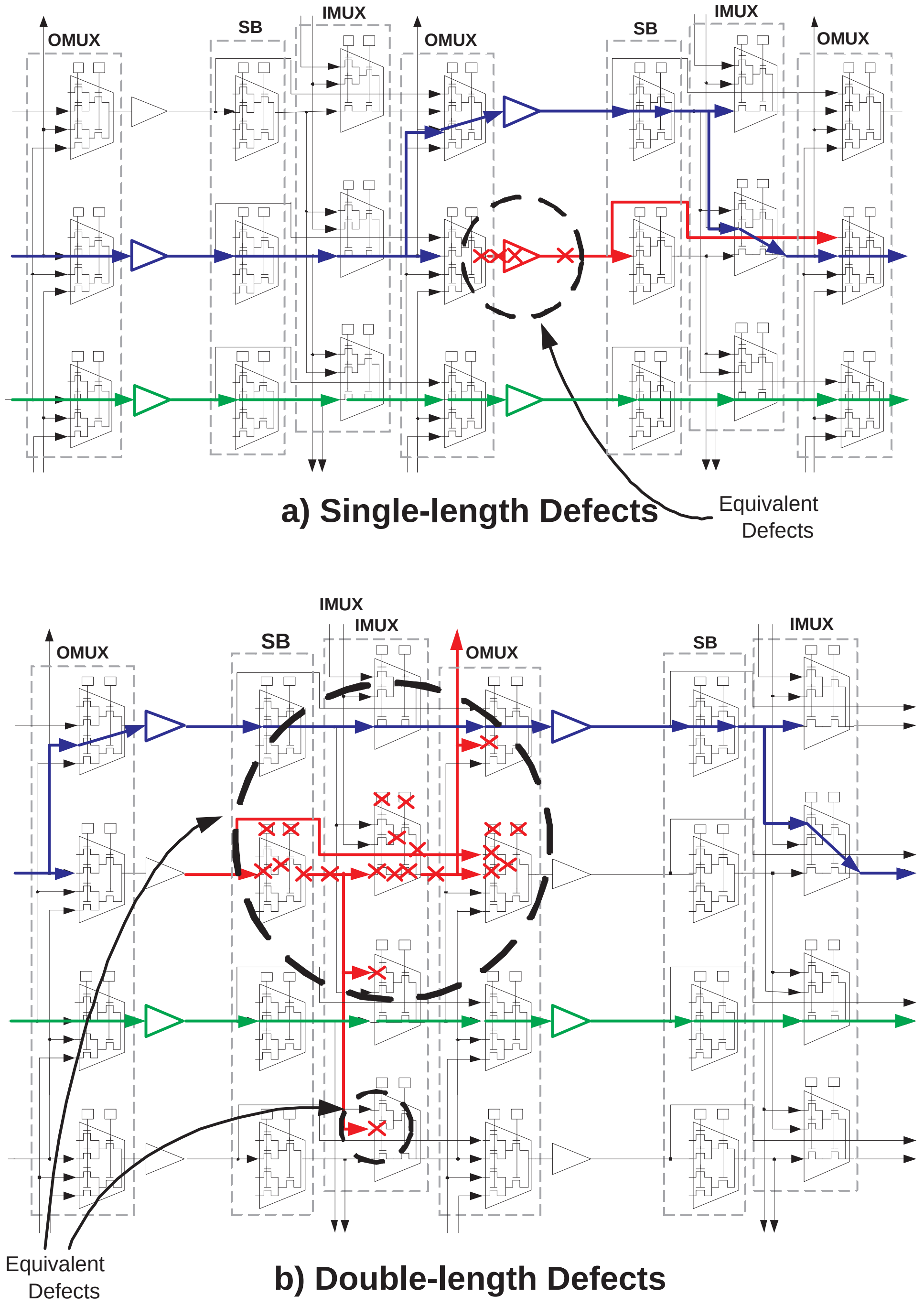
The proposed scheme categorizes non-bridging interconnect defects into three disjoint classes: single-length, double-length and intolerable defects. Depending on the underlying FGR architecture, the number of defect classes for bridging defects can potentially increase to five: single-length, double-length, triple-length, quadruple-length and intolerable defects.

Non-bridging Defects

If an open or stuck-at fault occurs on the wire, or there is a stuck-at fault in the wire driver or the output of the *omux*, the defect is a *single-length defect*. In this case, one switch block avoids the defect and all adjacent "downstream" switch blocks do the restore. This kind of defect is isolated to one wire length. Figure 3.3a illustrates how a single-length defect is corrected. With single-length defects, the change is purely localized in the channel to a group of wires with common start and ending points in the array. Such a group of wires is called a ***trackgroup***. To accommodate shift, each trackgroup has one spare wire for each direction.

If a defect is found in any of the multiplexers (aside from the output of the *omux*), the defect is categorized as a *double-length defect*. Due to their location, these defects actually impair the defect-correcting ability of the current switch block. To fix this, the switch block of the adjacent "upstream" trackgroup is used to avoid the defect, and the downstream switch blocks to the restore. Hence, this kind of defect requires two wire lengths to correct.

Figure 3.3b indicates how a double-length defect spans two adjacent trackgroups: the upstream trackgroup on the left, and the defective one on the right. In fact, for this example there are additional upstream switch blocks (above and below) that reside in the



a) Single-length Defects

b) Double-length Defects

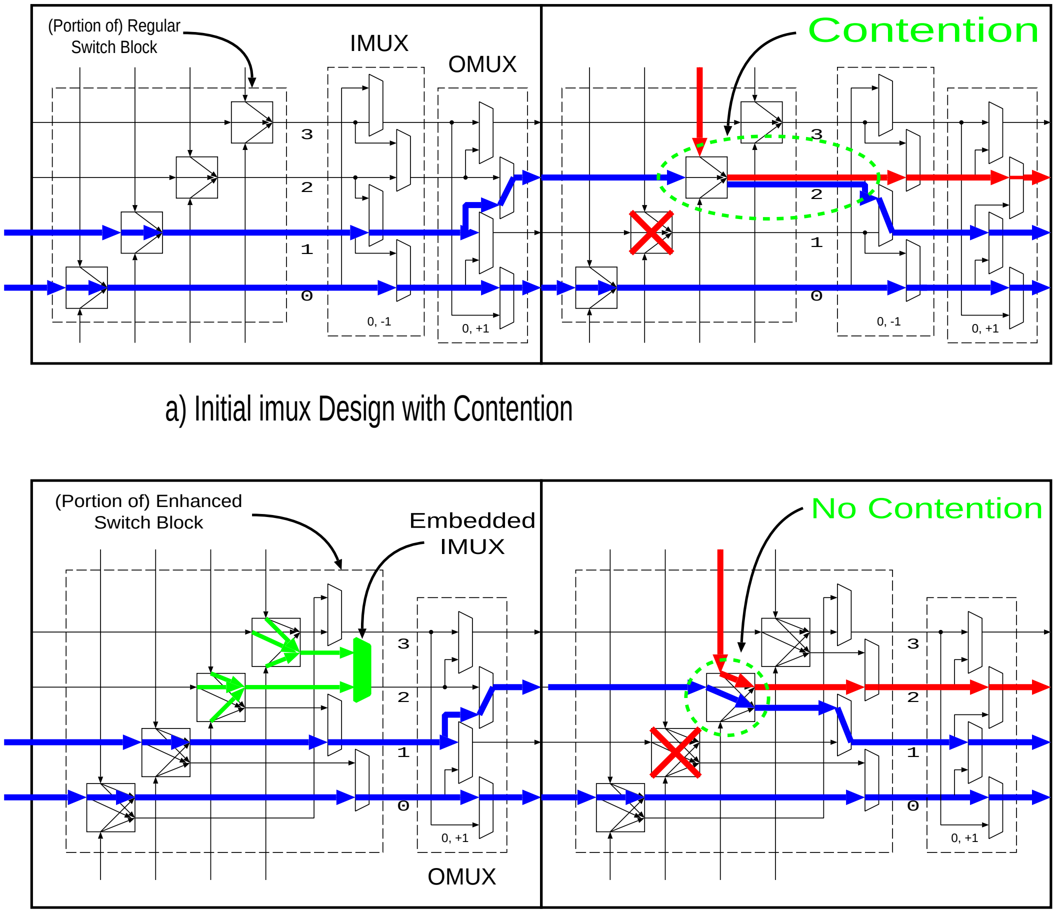
Figure 3.3: Single and Double-length defects

vertical channel. As shown in Figure 3.4a, this introduces contention when a straight-through signal is shifted up onto a track that is expected to be available for turning signals. To avoid contention, signals on tracks $\geq$t in the vertical channel must be shifted before arriving.

The upstream pre-shifting just described is only one way to solve the conflict problem with double-length defects. A more robust solution is shown in Figure 3.4b. Here, the *imux* is *embedded* within the switch block and the internal switch block multiplexers are duplicated. This shrinks the requisite defect-free area to just the two adjacent trackgroups.

Figure 3.5 highlights the neighbourhood that must be defect-free to correct for a double-length fault in an architecture with length 2 wires. The defect in question spans two trackgroups and affects the same track number. The defective track is avoided by up-shifting all the signals in the faulty trackgroup (highlighted in yellow). To localize the defect, all the signals leaving the faulty trackgroups are restored to the original track number immediately. The trackgroups containing these “fanout” signals include endpoint and midpoint connections, and are highlighted in blue. Highlighted in green are all the trackgroups with signals that “fanin” to the defective trackgroups. As mentioned before, signals in these trackgroups are pre-shifted by the same amount as the defective trackgroup to eliminate resource contention. Pre-shifting guarantees that the arriving signals are on the correct and unoccupied track.

The maximum number of trackgroups that can be affected by defect correction is called the ***minimum fault-free radius*** or ***MFFR***. The MFFR of a defect encompasses all the trackgroups that needs (or potentially needs) to be up-shifted and down-shifted. In general, trackgroups that contain shifted signals, or trackgroups that have had its signals recently restored are included in a defect’s MFFR. The MFFR of the previous example encompasses all the yellow, green and blue trackgroups.


(Portion of) Regular Switch Block
IMUX
OMUX
Contention
3
2
1
0
0, -1
0, +1
(Portion of) Enhanced Switch Block
Embedded IMUX
No Contention
OMUX

a) Initial imux Design with Contention

b) Embedded imux Design Avoids Contention

Figure 3.4: Embedding *imux* to avoid contention

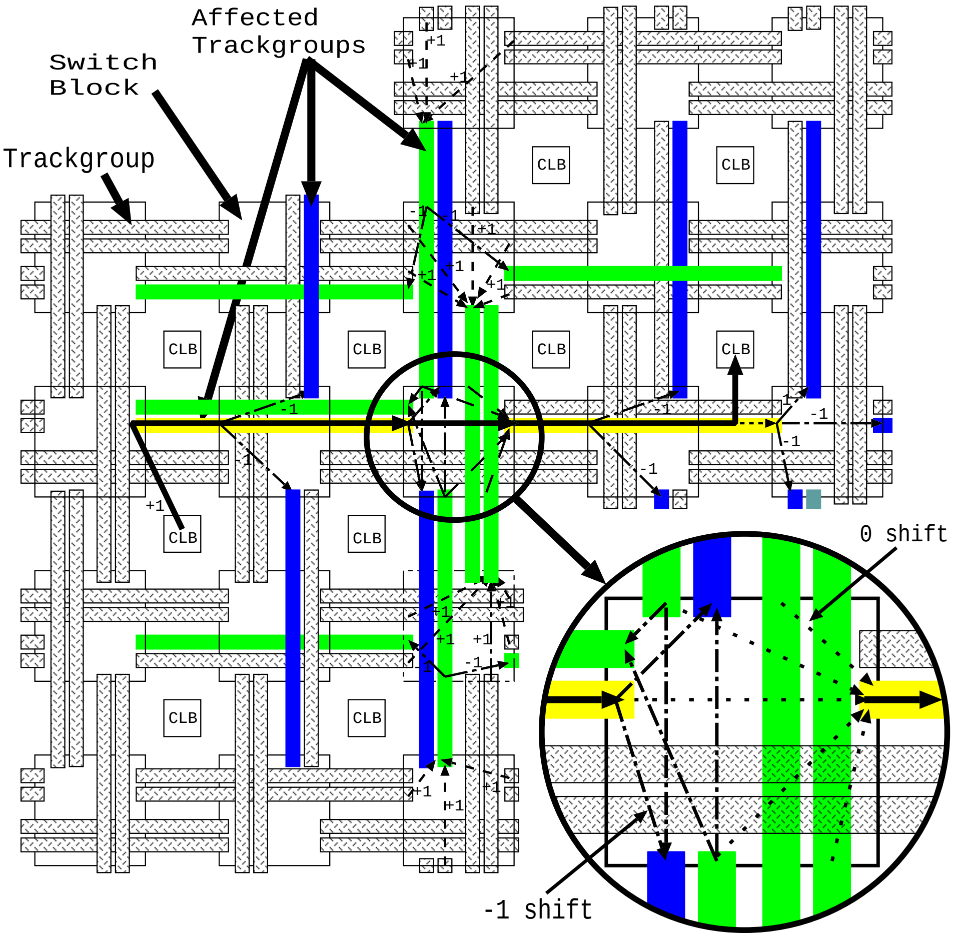


Figure 3.5: Defect correction example (L=2)

Bridging Defects

Wire bridges and certain source-drain shorts have the potential to render two adjacent tracks as unusable. To avoid such a defect, the upstream switch block(s) must shift tracks up by 2 and the downstream switch blocks must shift down by 2. If this is implemented using +/-2 shifts, the defect is classified as a double-length defect. If this is implemented with a combination of +/-2 and +/-1 shifts (i.e., one +2 shift followed by two -1 shifts), the surrounding neighbourhood that must be defect-free can potentially be larger than that of a double-length defect. An architecture that only supports shifts by +/-1 require two +1 shifts followed by two -1 shifts to avoid bridging defects or source-drain shorts. In this worst case scenario, the defect is considered a quadruple-length defect.

Intolerable Defects

There exists a class of intolerable defects that has not been considered which includes power/ground shorts and clusters of defects. The first type of defect cannot be tolerated. However, it may be possible to tolerate the latter by complementing our architecture with a spare row/column technique, i.e., [18].

3.1.4 Modes of Operation

FGR allows defect redundancy to be an option to the user. This means an FPGA can operate in two modes: *normal defect-tolerant mode* and *recovery mode*.

The *normal* mode assumes the customer will buy imperfect, low-cost devices, and utilize the underlying defect-tolerant architecture to avoid defects. In this mode, the routing software reserves a spare routing track in each trackgroup.[1] This reduces the number of routing tracks available to the application, but the spares are needed for defect correction. For many applications which do not stress the routing network but require inexpensive devices, this is an easy way to lower device cost.

[1] Two spares are needed for devices with bridging defects, which may be sold at even lower cost.

The *recovery* mode assumes the customer will buy perfect devices at a price premium. In this mode, the routing software uses the additional *imux* and *omux* routing multiplexers to increase the flexibility of the interconnect. In essence, the router is using the redundant resources to recover some area/delay efficiency that was sacrificed when they were added. This mode is used for those few applications that have high interconnect demands where the resulting increase in interconnect flexibility is even more helpful. However, in this mode, there is no natural ability to tolerate defects.

Recovery mode is the true overhead cost of FGR. It is the additional cost imposed on aggressive applications with high interconnect demands.

When normal mode is used, the overhead appears to be higher because spare tracks are also counted as overhead. This is misleading! Applications with low interconnect demands already have an abundance of unused routing tracks, so this extra capacity is already built-in. The customer has already paid for these unused routing tracks, so there is no real end-user cost to supplying them. The proposed redundancy scheme merely finds a use for these free tracks by calling them spare tracks.

3.1.5 Detailed Transistor-level Design

The transistor circuit schematic used for HSPICE simulations is shown in Figure 3.6. The components in the circuit (from left-to-right) are: input buffer, directional multiplexer, strengthening buffer, shift-restore multiplexer (*imux*), shift-avoid multiplexer (*omux*), tapered driver and the wire model with loads.

For area considerations, the directional multiplexer is implemented using a tree of minimum-sized transistors. This allows the use of encoded control lines and reduced SRAM usage. We also assumed both true and complemented outputs are available from a 6-transistor SRAM cell.

The *omux* is implemented using a *decoded* multiplexer with a single level of minimum-width pass transistors. Each pass transistor is controlled by an independent SRAM cell. The motivation for this is to reduce delay.

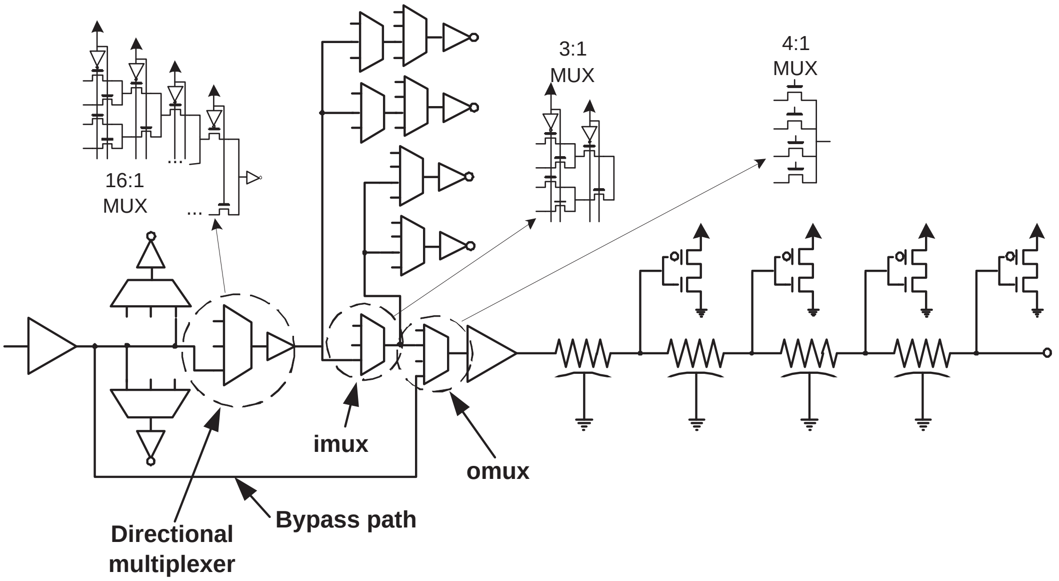


Figure 3.6: HSPICE schematic for delay characterization.

Three different implementations of the *imux* were considered: decoded, encoded and embedded. The decoded multiplexer is identical in implementation as the *omux*. This kind of multiplexer trades area for delay. The encoded *imux* is built like the directional multiplexer. It trades delay for area.

As mentioned earlier in Section 3.1.3, it is also possible to embed the *imux* into the directional multiplexer. This enhanced multiplexer is built by duplicating the inputs of the directional multiplexer for track t+1 and t+2, and connecting them to the directional multiplexer for track t. An embedded *imux* allows signals to turn and shift at the same time. As will be shown later, this improves yield with double-length defects at the expense of some area.

Seven different defect-tolerant implementations were considered in this study. These implementations vary in the implementation of the *imux* and shifting ability of the multiplexers. The +/-2 shifts use additional area to improve yield of bridging defects. The attributes and differences between the switch implementations are summarized in Table 3.1.

Arch.	imux impl.	imux “-2” shift	omux “+2” shift	imux/omux ±1 shift
EM22	embedded	Y	Y	Y
EM12	embedded	N	Y	Y
EM11	embedded	N	N	Y
FL22	decoded	Y	Y	Y
EN22	encoded	Y	Y	Y
EN12	encoded	N	Y	Y
EN11	encoded	N	N	Y

Table 3.1: Defect-tolerant switch implementations.

The area and delay performance of the implementations are also sensitive to the precise transistor-level circuit design of the multiplexers and buffers. The procedure used for transistor sizing is the same as the one described in [25, 26], and is as follows:

1. Select a parameter to be optimized
2. Sweep the selected parameter across a range of values while holding all other circuit parameters fixed
3. Determine the best parameter value based of the minimization of area-delay product for the entire circuit
4. Reiterate steps 1 to 3 for remaining circuit parameters
5. Repeat steps 1 to 4 until the area-delay product stabilizes

The stabilization of circuit parameters and delay profiles was observed to take approximately 3 complete iterations. Delay results are computed from HSPICE simulations of TSMC’s 180nm technology.

3.1.6 Software Aspect of FGR Defect Avoidance

To successfully correct a defect, the location of the defect must be known. One such way to provide this information is through the use of a relatively unique list of defective resources or *defect map*. These can be stored on-chip in non-volatile storage, or in an off-chip database indexed using a unique on-chip serial number.

The aforementioned hardware changes provide an infrastructure for defect correction, but not the means to apply it. Defect correction for FGR is applied through bitstream manipulation. When an FPGA is being programmed, the defect map specific to that FPGA is called up and the appropriate modifications are made. The correction can be applied during programming or bitstream generation. It can even be applied by means of an embedded configuration processor within the FPGA or configuration memory or subsystem. With this latter method, defect correction can be completely hidden from the user.

3.2 Limitations

The proposed architecture has a number of limitations. First, FPGA and VLSI testing strategies capable of locating defects were assumed to be available. The defect information, in a form of a defect map, may be provided by the vendor or generated by the user. The defect map does not need to be overly detailed. For each defect, it must identify the wire segment location in the array (x, y and track numbers) and type (single- or double-length). Bridges are identified as adjacent defect pairs.

The method of dealing with bridging defects assumes that routing tracks within the same channel are laid out beside one another. This may not be a realistic assumption since there are many factors that influence the layout of an FPGA. This solution should only be viewed as a general approach. To fully protect an FPGA from bridging defects, the final FPGA layout must be considered.

As described earlier, defects must be surrounded by some defect-free resources for successful repair. As a result, this approach cannot tolerate clusters or closely-spaced defects. To reconcile this shortcoming, it is possible to complement FGR with a spare row/column technique [18].

Finally, defects in the logic block have been ignored in this architecture. This issue has been addressed in the past [20, 23, 24]. In these techniques, logic blocks are intentionally left under-utilized (thereby creating the impression of having spare resources). In the event of a defect, resource assignment within the logic block is manipulated so that the logic on a defect resource is shifted to an unused resource. To achieve logic block defect tolerance, these techniques, or a spare row/column technique, can be used to complement FGR.

3.3 Area and Delay Results

The new architectural features were incorporated into an enhanced version of the VPR place and route tool, VPRx [26], which now supports directional wires [25]. VPRx was then used to map the 20 largest MCNC benchmark circuits [9] into an island-style FPGA consisting of directional length 4 wires and CLBs containing eight 4-input LUTs. The area and critical path delay results reported by VPRx for each circuit is normalized; then a geometric average across the circuits is computed.

3.3.1 Area

Routing experiments with non-defect-tolerant switch blocks indicated that the directional switch *E3M1* used 2% less area than *E3M2* and *E3M1*. The average critical path delay for *E3M1* was also 3% lower than the other two architectures. In comparison, the average channel width increased by 2% and 8% for *E3M1* and *E2M1*, respectively. These results are shown in Figure 3.7. Hence, the non-defect-tolerant *E3M1* was selected to be the basis for all area and delay normalization (=1.0).

Figure 3.8 presents the average area overhead for the defect-tolerant switch blocks. The results have been normalized to the non-defect-tolerant *E3M1*, the best alternative without defect tolerance. When routing the design in *normal* mode, two spare sets of wires were added in the channel for the 7 architectures that tolerate bridging defects. Only one spare set of wires is inserted for the architectures that do not tolerate bridging defects (the 2 architectures with -NB). These spare wires were not used during routing. The EN11-E3M1 architecture was the most area-efficient, having an area overhead of 24% for non-bridging defects and 34% for bridging defects. The difference in area cost (10%) is one set of spare

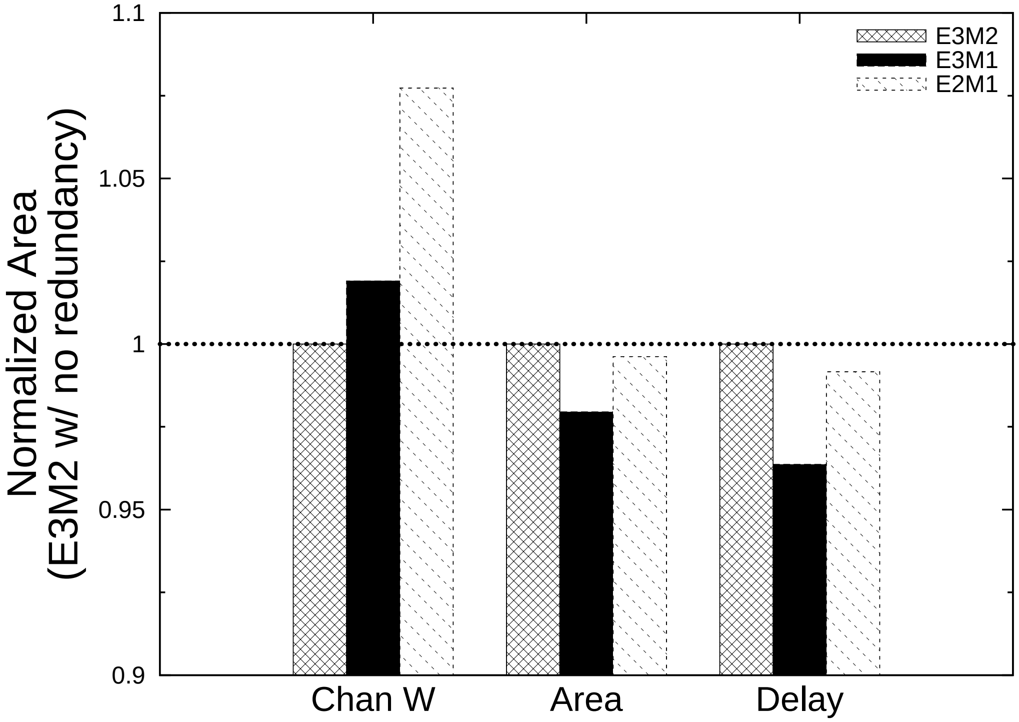


Figure 3.7: Flexibility exploration for non-fault tolerant architectures

wires. Notice that the second-best area architecture, EM11-E3M1, needs +4% to embed the *imux,* but it will be shown later that it tolerates more defects.

3.3.2 Delay

The average critical path delay for each architecture is shown in Figure 3.9. These numbers were obtained by rerouting the 20 benchmark circuits using a channel width equal to the minimum channel width obtained from the defect-tolerant area investigation plus one additional set of wires. Unlike the spare wires that are held in reserve, the router was allowed to use this new set of wires to relieve delay increases caused by congestion. The experiment indicated that the EM11-E3M2 architecture gave the lowest average critical path delay overhead of 15%. Overhead for the non-embedded version, EN11-E3M1 was 24%.

Figure 3.9 also shows the importance of the bypass path for delay reduction. The

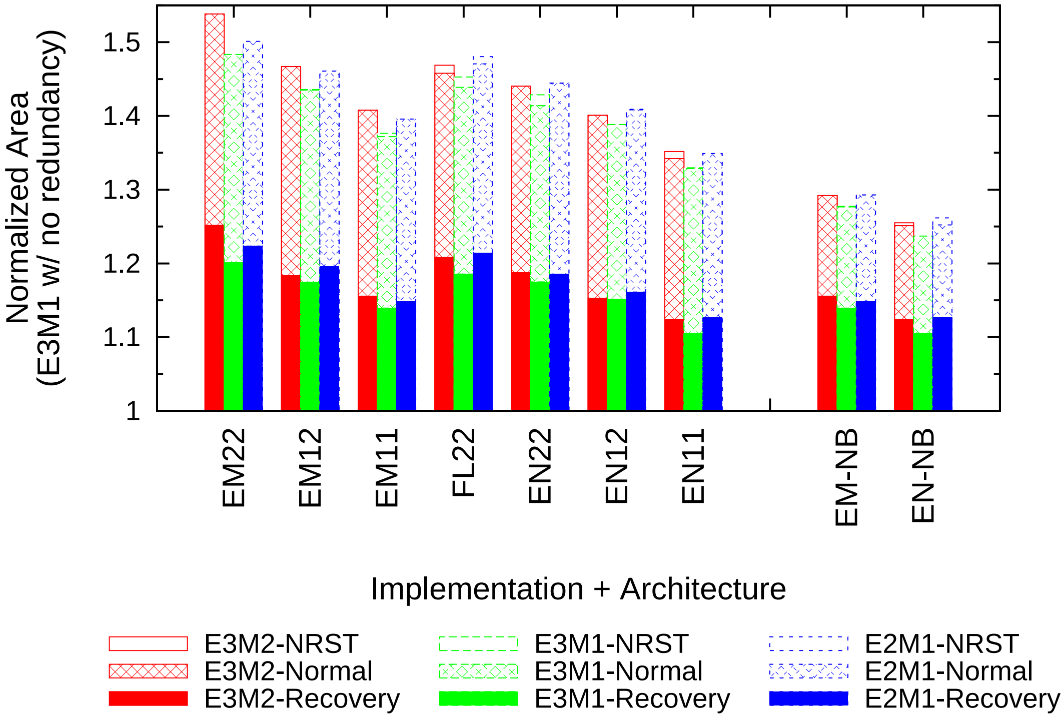


Figure 3.8: Area of defect-tolerant implementations.

-NRST results (no route on straight through) show significantly higher delay when the router is forced to avoid using the bypass path.

3.3.3 Area and Delay Recovery

To explore the true area and delay overhead, the routing tool was put into recovery mode. In general, it was observed that the router needs lower channel widths for defect-tolerant architectures in recovery mode than non-defect-tolerant architectures. Hence, the additional multiplexers do help improve interconnect flexibility.

Figure 3.8 shows the true area overhead for the defect-tolerant implementations in recovery mode. The EN11-E3M1 architecture demonstrated the lowest area overhead of 11%. Overall, recovery model saves a significant amount of area.

The critical path delay overhead in recovery mode is shown in Figure 3.9. Here, the

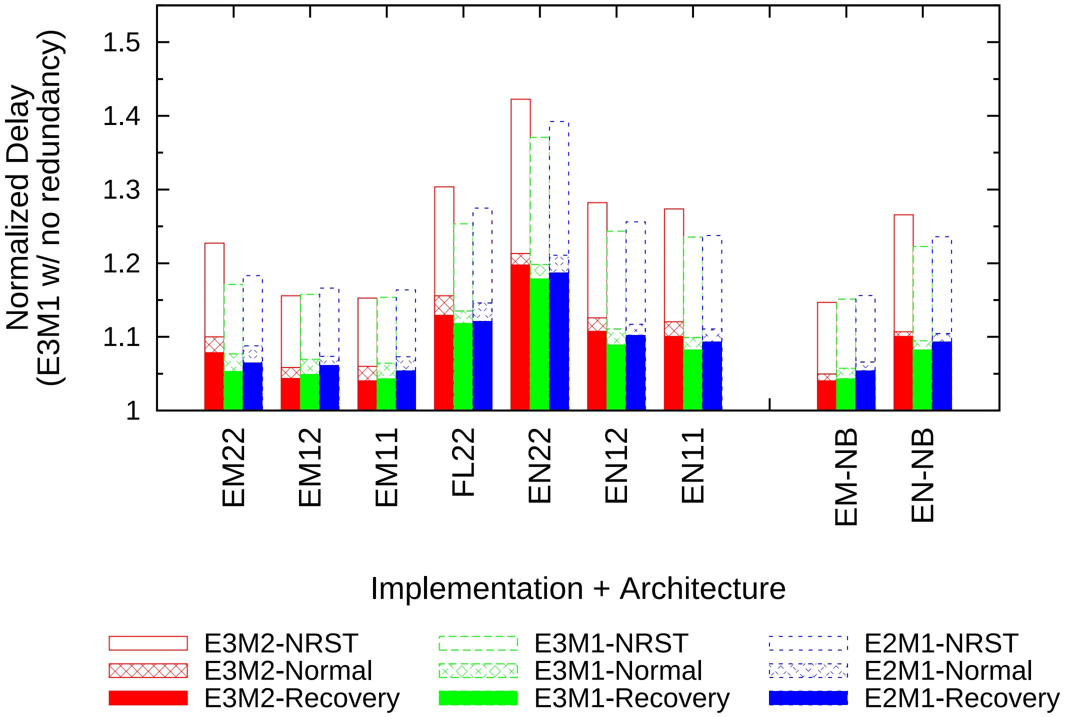


Figure 3.9: Delay of defect-tolerant implementations

EM11-E3M2 architecture demonstrated the lowest delay overhead of 5%.

3.3.4 Area and Delay Product

Using the delay and area results obtained from the previous two experiments, the area-delay product for each architecture in recovery mode was computed. Figure 3.10 shows that the EM11-E3M1 architecture produced the lowest area-delay product.

When comparing yield later in this thesis, EM22-E3M1 will be selected as the best FGR variation. Although it has the highest area overhead, it tolerates the largest number of defects and has among the lowest delay.

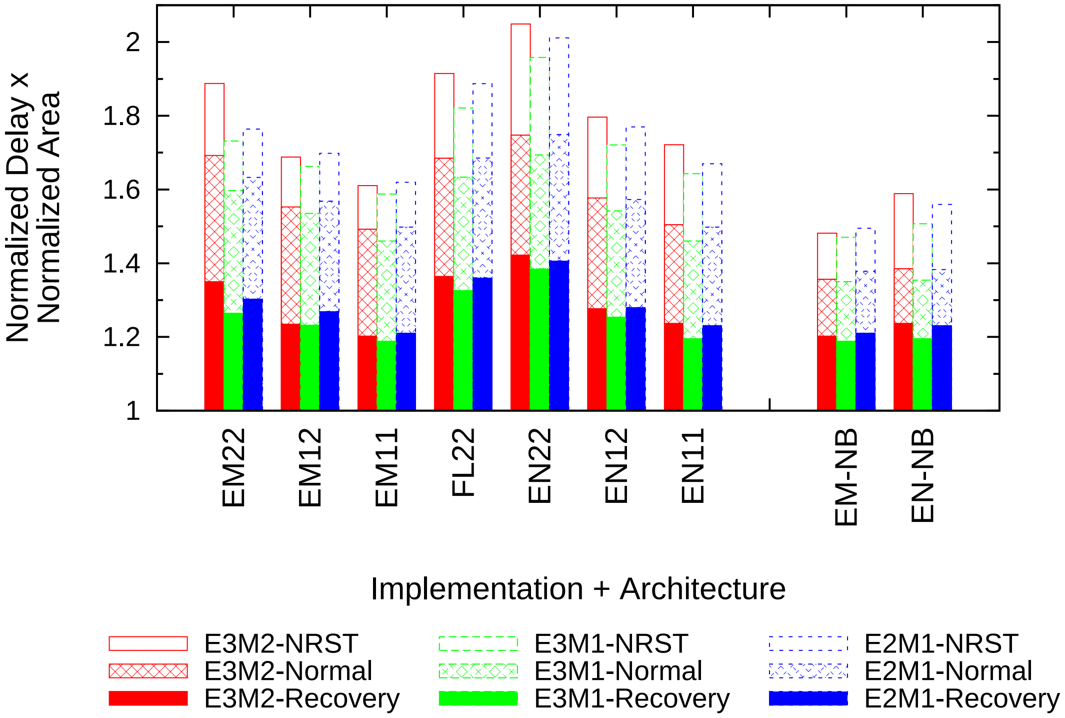


Figure 3.10: Area-delay product comparison.

Chapter 4

Coarse-grain Redundancy (CGR)

In a traditional coarse-grain redundancy (CGR) scheme, one spare row and one spare column is used to correct defects. This approach can tolerate clusters of defects within the same channel. However, the consolidation of spare resources into a single spare row/column severely restricts this architecture's ability to tolerate randomly distributed defects. In this chapter, two schemes for adding multiple spare rows and columns to tolerate these types of defects are considered.

4.1 Architectural and Implementation Details

Traditional CGR adds one spare row and one spare column to the existing FPGA layout. This architecture is limited to defect correction for one row and one column, but it can naturally tolerate clusters of defects within the same channel. In the event of such defect clusters, the row or column containing the defects is bypassed and the spare row or column is used. This architecture can in fact tolerate multiple defects within the same channel. However, as array size grows, it becomes increasingly unlikely that multiple defects will lie in the same row/column. To increase yield, a scheme is needed to add additional spare rows and columns to the architecture. However, traditional CGR does not clearly indicate how multiple spare rows/columns can be added. How to support multiple spare rows/columns in an island-style FPGA architecture is considered below.

4.1.1 Switch Block Changes

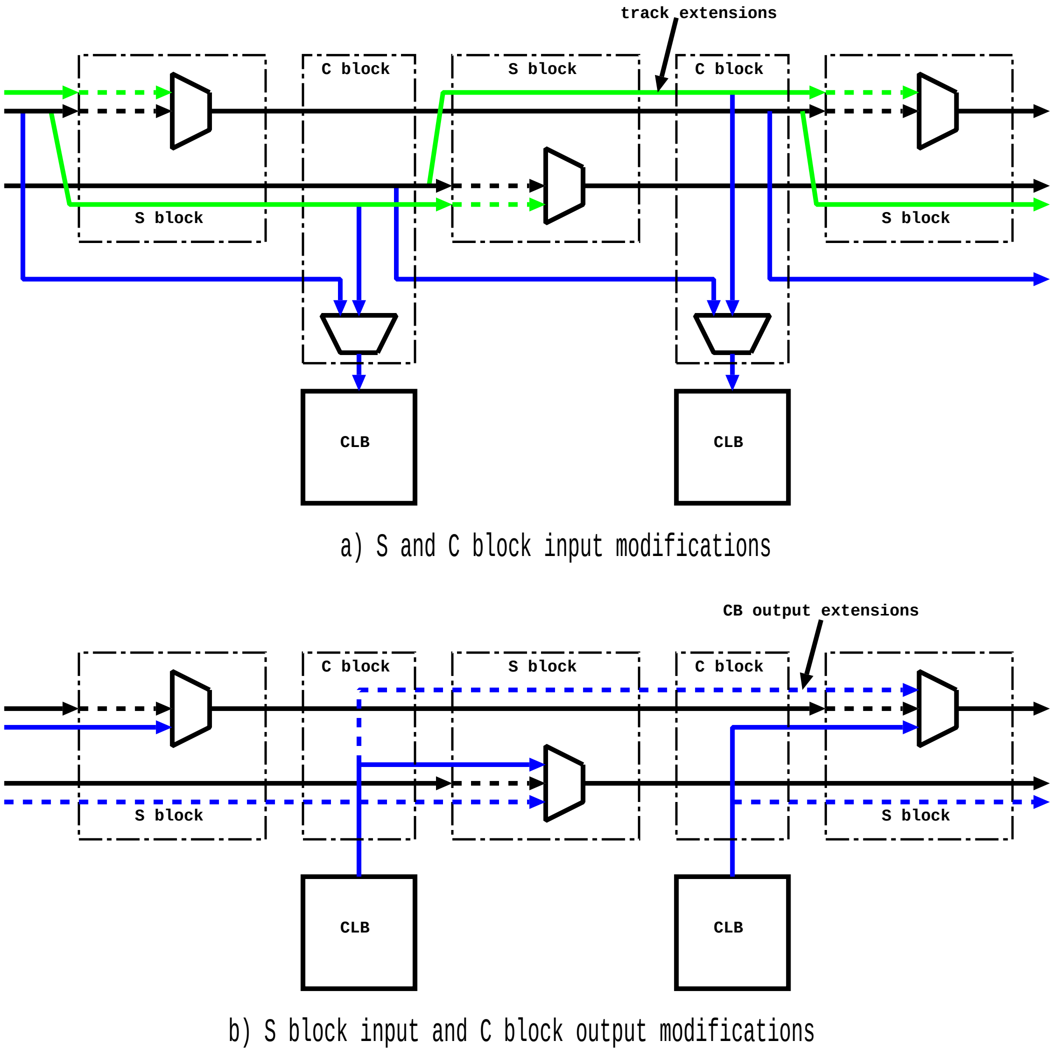
CGR requires modification to the existing detailed switch block design. To allow one row (and column) to be bypassed, all interconnect wires are extended in length by one row/column. Figure 1.1a highlights these track extensions in green. These track extensions are not used or needed in a defect-free FPGA. In the presence of a defect(s), the extensions allow signals that would have terminated at the defective row/column to reach the subsequent defect-free row/column. An example of row bypassing is shown in Figure 1.1b. Notice that the track extensions are enabled only for the wires that traverse the defective row/column or for wires that start at a shifted row/column. The impact of these extensions on the S block is shown in Figure 4.1. These detailed changes have been documented by previous papers in the area.

4.1.2 Connection Block Changes

The bypassing of a row/column also necessitates changes in the connection block design. These changes are highlighted in Figure 4.1. In addition to the original connections, notice that the CLB inputs now accept connections from the wire extensions, and that the CLB outputs connect to wire starts in the original switch block and the switch blocks one channel over. The duplication of input and output connections are required to make the row/column bypassing transparent to the CAD tools.

4.1.3 Multiple Spare Rows and Columns

There are two ways to construct a multiple spare rows and columns architecture. The first method adds *global* spare rows and columns to the array. These *global* spares can be used to correct defects anywhere in the FPGA. This architecture can tolerate as many defective rows/columns as there are spares. The added cost of this approach is the increased length of the routing wires, the spare rows/columns, plus the extra multiplexing needed within the S and C block to accomplish the bypass. The additional switching adds significant area overhead, and the wire extensions add significant capacitance which increases both delay



a) S and C block input modifications

b) S block input and C block output modifications

Figure 4.1: Connection block changes for CGR

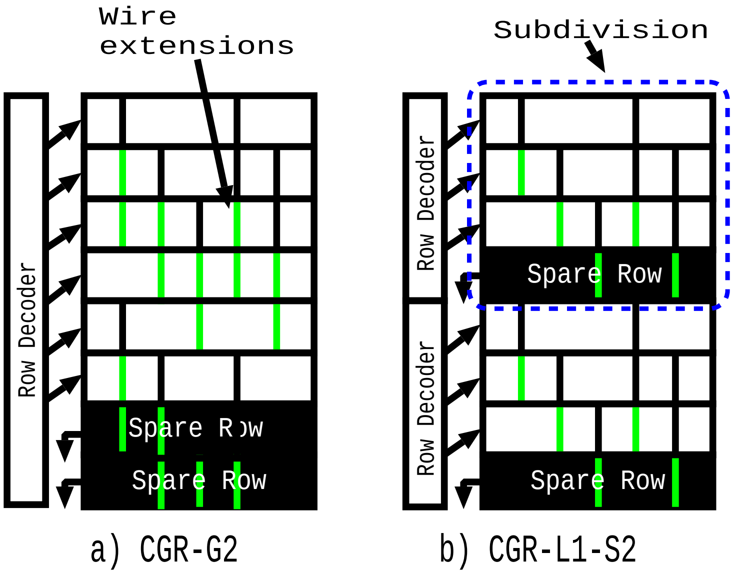


Figure 4.2: Multiple spare row and column architectures

and power.

The second method of implementing a multiple spare rows and columns architecture is by distributing spare rows/columns evenly among subdivisions of the chip. The FPGA is divided into smaller subdivisions. Each subdivisions has dedicated *local* spare resources. Defect correction is handled locally within each subdivision. This approach has a smaller area overhead for a given total number of spares because the spare wire extensions are shorter and switch overhead is reduced.

For conciseness, the multiple *global* spare rows and columns architecture and the multiple *local* spare rows and columns architecture will be denoted as CGR-Gn and CGR-Ln-Sp, respectively. CGR-Gn has exactly n global spare rows/columns that can be used to repair any defective row/column. CGR-Ln-Sp is divided into $2p$ subdivisions, p subdivisions for rows and p subdivisions for columns. Each subdivision has n spares. Overall, this architecture has $2pn$ spare rows/columns. Figure 4.2 presents CGR-G2 and CGR-L1-S2.

4.1.4 Supported Defects

In CGR, all defects are essentially treated equal. A channel containing defects is always replaced with an entire spare row or column. This simplifies the correction process, but has the drawback of being inefficient with resource usage.

Since there is only one spare row and one spare column, traditional CGR can only tolerate defects in one row and one column. Within that row/column, multiple defects can be tolerated. This gives the architecture the natural ability to tolerate clusters of defects within the same channel. CGR can also tolerate defects in both routing resources and logic blocks; but like FGR, it cannot tolerate power/ground shorts. Also, some types of defects may render **both** row and column unusable because it happens at a turning switch (i.e., a horizontal wire is shorted with a vertical wire).

4.1.5 Detailed Transistor-level Design

Unfortunately, published research does not present the delicate circuit details needed to perform the bypass. Altera patents provide some insight [3] and indicate that additional circuitry is required for bypassing. Without a detailed design, it will be demonstrated in Section 4.3 that the switch overhead alone for CGR-G1 and CGR-L1-Sp is similar to FGR.

4.2 Limitations

Although the area overhead of spare rows/columns appears to be very clear, the additional circuitry required within each S block and C block to bypass a faulty row/column is non-trivial and is not reported in previous work. To account for this additional overhead, estimations are use in place of detailed transistor models.

An underlying assumption of traditional CGR is that defects are isolated to one row/column. This assumption allows defect-avoidance through row/column bypasses. However, some types of defects (i.e., shorts between wires in two different rows/columns) may render two rows or two columns unusable. Such defects cannot be tolerated in traditional

CGR since there is only one spare row/column.

4.3 Estimated Results

The following area and delay estimates are based upon the CGR model shown in Figure 4.3 in comparison with the FGR model in Figure 4.4. This figure highlights the architectural changes needed to convert a non-defect tolerant switch block into a CGR-G1 and CGR-G2 switch block. The architectural changes needed for a CGR-Ln-Sp architecture is comparible to that of a CGR-Gn architecture for the same n, but more spares are needed.

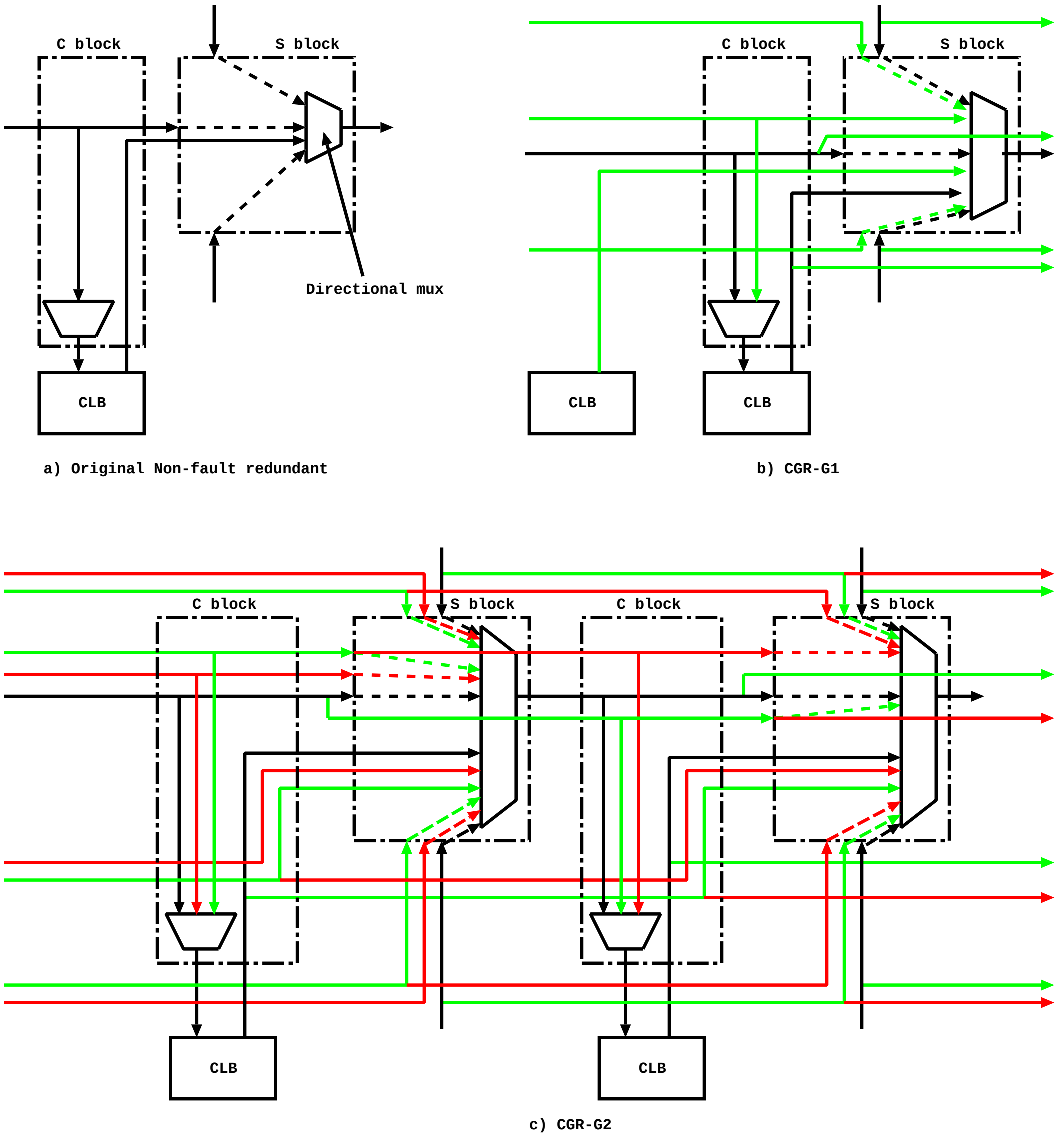
4.3.1 Area

Figure 4.5 shows the spare row/column area increases needed for CGR-G1. These results do not include the additional switch area, only the overhead resulting from the addition of one spare row and column. Notice that the area overhead decreases as the FPGA area size grows. In a 32x32 FPGA, the area overhead for one spare row and column is approximately 6% (2 $\times$ 1/32). The area overhead reduces to approximately 1% for a 256x256 FPGA.

Figure 4.3a presents a non-defect tolerant switch block. To make this a CGR-G1 switch block, wire extensions are needed for CLB inputs and outputs and routing tracks. Figure 4.3b highlights the necessary extensions in green. Notice that the number of inputs for the CGR-G1 directional multiplexer roughly doubles. This actually makes it similar in size to the EM11 embedded imux shown in Figure 4.4a. In addition, EM11 requires an additional 2:1 multiplexer (omux).

The necessary changes for CGR-G2 are shown in Figure 4.3c. This architecture has two spare rows/columns, thus wire extensions span two switch blocks. The first extension is highlighted in green, while the second in red. This switch block is comparable to the EM22 switch block shown in Figure 4.4b.

The similarity between the switch blocks allows for a rough approximation of area overhead. For CGR-G1, the area overhead is approximately that of EM11 which is about



a) Original Non-fault redundant

b) CGR-G1

c) CGR-G2

Figure 4.3: Comparison between CGR implementations

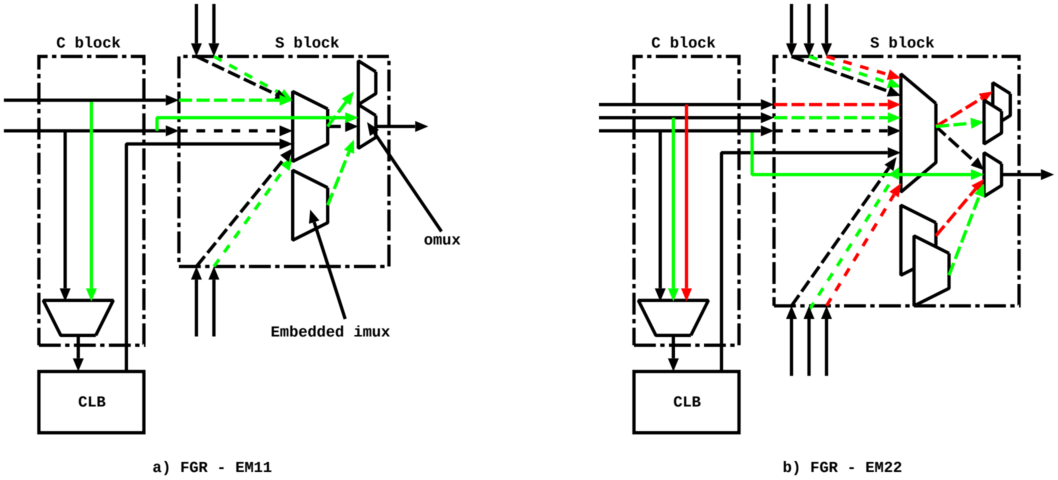


Figure 4.4: Comparison between FGR implementations

40%. CGR-G2, which is similar to EM22, has an area overhead of approximately 50%. The switch block area overhead for CGR-G1 and CGR-G2 will be slightly less than that of EM11 and EM22 respectively since the FGR approach requires the addition of omux multiplexers. However, for architectures where there are more than 2 global spares, the CGR area overhead will be significantly greater than that of FGR.

4.3.2 Delay

Again, the similarity between switch block implementations allows for a gross approximation of delay overhead. EM11 and EM22 have delay overheads of approximately 20%. The delay overhead for CGR-G1 and CGR-G2 architectures are likely to be slightly less than the FGR archtectures since CGR has one fewer multiplexer level. For architectures with more than 3 spare rows/columns, the delay overhead will increase beyond FGR as there are significantly more inputs (and thus levels) to the directional multiplexer itself.

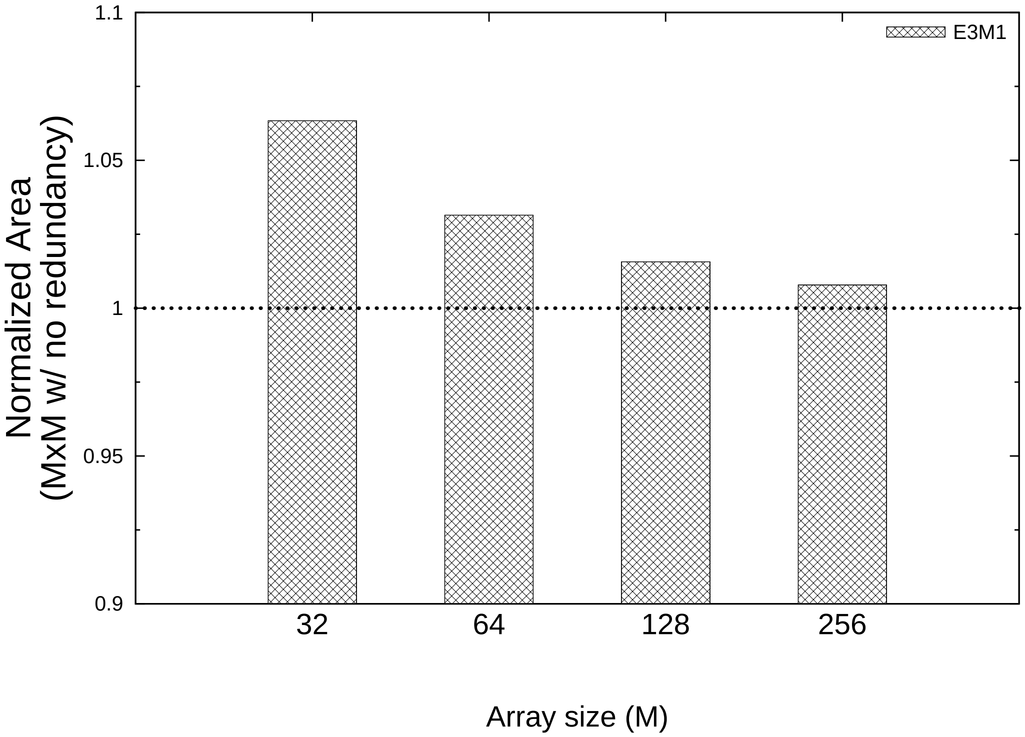


Figure 4.5: CGR-G1 for increasing array sizes (spare row/column overhead only)

4.3.3 Scaling Factors

To approximate area for CGR, the area of an NxN[1] FPGA is multiplied by a constant scaling factor of 1.5. The value of 1.5 is used because the S block for CGR-G2 is similar to EM22, which has an area overhead of 50%. Similarly, switch area overhead for CGR-G1 and CGR-L1-Sp is less than this, being similar to EM11. In this case, a scaling factor of 1.3 is used. Since architectures with more than 2 spare rows/columns will likely be significantly larger than EM22, this approximated area represents the lower bound area overhead for CGR with more than 2 spare rows/columns.

Similarly, a scaling factor of 1.2 can be used for the approximation of delay overhead. However, delay results for CGR is not presented in this thesis.

[1] N = M + n, where M is the base array size and n is the number of required spare rows/columns.

Chapter 5

Yield Comparison

This chapter presents a yield comparison between FGR and CGR. The comparison is based on four factors that influence yield: switch implementation, switch flexibility, array size and wire length. First, however, a yield model is presented for both CGR and FGR.

5.1 Yield Model

For the subsequent yield analysis, all faults are assumed to be bridging defects (worst case). Logic faults, which are intolerable in the present FGR architecture, are not considered.

5.1.1 Coarse-grain Model

A number of simplifications and assumptions are made for the CGR yield model. The model assumes the following:

- All channels have identical routing resources and thus have an equal probability of being defective;
- The vertical and horizontal channels are disjoint routing networks (defects are assumed to be isolated to just a row or just a column);
- MxM FPGAs are perfectly symmetrical; and

- A spare row and column are added in tandem to retain the square shape.

To inject a random defect, a random row or column is selected, and the defect count for that row/column is incremented. For CGR-G1, a failure occurs when there are defects in two different rows or two different columns. A failure is represented as a non-zero defect count for two different rows or two different columns. Architectures with multiple *global* spare rows and columns are evaluated in a similar manner. A failure occurs when there are more defective rows or columns than there are spare rows/columns.

In CGR-Ln-Sp, each subdivision has exactly n designated spare rows/columns. This architecture can tolerate at most n defective rows/columns per subdivision. Figure 5.1 highlights the differences in terms of defect correction between CGR-G2 and CGR-L1-S2.

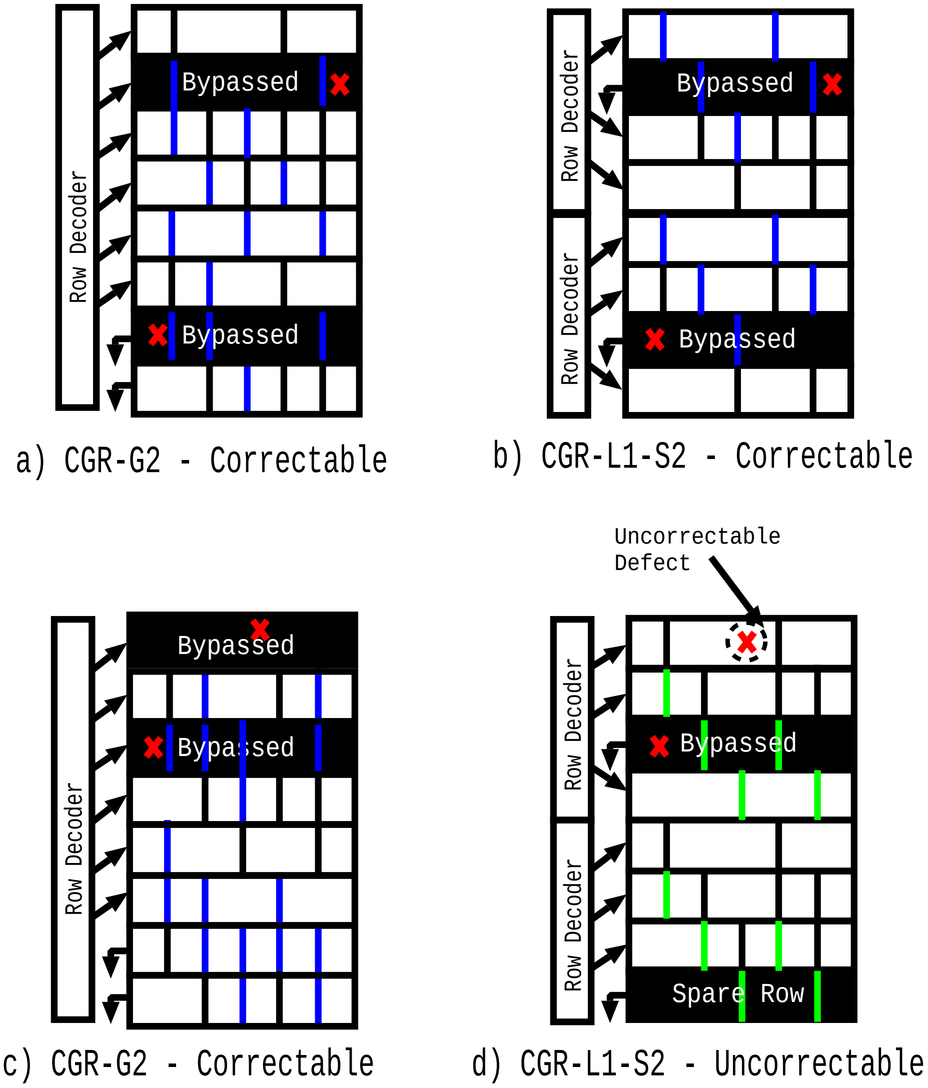
5.1.2 Fine-grain Model

To model the behaviour of defect correction for FGR, state variables are assigned to every trackgroup within the FPGA. A trackgroup can have one of three states: *perfect*, *faulty*, or *must be perfect*. The *faulty* state indicates the presence of a defect in that particular trackgroup. The *must be perfect* state is used to mark the MFFR of a defect. As mentioned before, the MFFR of a defect defines the region needed for shifting to avoid and restore around the defect. To guarantee that a defect can be correctable, the MFFR of a defect must be defect-free. The *must be perfect* state facilitates the enforcement of this requirement.

Defects are injected into the model by randomly selecting a trackgroup and setting its state to *faulty*. The neighbouring trackgroups as defined by the MFFR are marked as *must be perfect*. The MFFR will vary depending on the defect type and the underlying routing architecture. Chip failure occurs when any of the following conditions are violated:

- A *faulty* trackgroup overlaps with another *faulty* trackgroup – only one defect per trackgroup can be tolerated; or

- A *faulty* trackgroup overlaps with a *must be perfect* trackgroup – the defect may inhibit the ability to correct the other defect; or



a) CGR-G2 - Correctable

b) CGR-L1-S2 - Correctable

c) CGR-G2 - Correctable

d) CGR-L1-S2 - Uncorrectable

Figure 5.1: Switch block with spare connections

- A *must be perfect* trackgroup overlaps with another *must be perfect* trackgroup – the defect corrections may interfere with one another.

The yield approximation for FGR is pessimistic in two ways. First, the approximation only considers the injection of bridging defects. Second, MFFR overlap is not allowed. In reality, not all faults are of the bridging category, and thus have a significantly smaller MFFR. Next, the avoidance and correction of certain defects can in fact be overlapped. However, without real manufacturing defect information, the worst case position is assumed. It is suspected that the accounting of these two factors would appreciably improve the yield for FGR.

5.2 Architectural Considerations

Several factors can affect yield. This section discusses a few important ones. Results will follow in the next section.

5.2.1 Switch Implementation Impact on Yield

As mentioned in Chapter 3, the switch block in fine-grain redundancy can either have an embedded or extracted imux. Of the two, the embedded imux has the highest area overhead but also the greatest connectivity. The additional connections in an embedded imux allows signals to turn and shift at the same time. As will be shown in the next section, the ability to turn and shift improves yield by reducing the number of trackgroups that must be pre-shifted.

It was also noted that the shifting ability of the switch block can be varied. If the ability to shift by two is eliminated, the size of the switch will decrease. Bridging defects and source-drain shorts can still be tolerated, but avoiding them will require two "+1" shifts followed by two "-1" shifts. In fact, any combination of shifts, a "+2" followed by two "-1", two "+1" followed by a "-2" is acceptable. As noted in Section 3.1.3, changing the shifting ability of the multiplexers can potentially increase both the number of defect categories and the MFFR.

5.2.2 Flexibility Impact on Yield

The number of wires connected to a given switch block wire is defined as its flexibility, F_s [31]. F_s can be used to describe both end switch blocks or *endpoints* and middle switch block or *midpoints*. With long wires, the flexibility at the endpoints and midpoints can differ. Lower F_s values equate to fewer connections, and thus smaller MFFRs. Thus, lower F_s values actually help improve yield.

5.2.3 Array Size Impact on Yield

It was shown in Chapter 4 that CGR-G1 demonstrates a decreasing amount of area overhead as array size increases. Unfortunately, as array size grows, it also becomes increasingly unlikely that multiple defects will lie in the same row/column. Thus, to maintain a fixed yield for growing array sizes, it is necessary to increase the number of spare rows and columns.

In FGR, spare resources are distributed across the FPGA. Increasing the array size increases the number of trackgroups in the FPGA, and thus the amount of available spare resources. Since the amount of spare resources naturally grows with size, the architecture tolerates more and more defects.

5.2.4 Wire Length Impact on Yield

In CGR, the routing network within the rows and columns are assumed to be identical and disjoint. Since all rows and columns contain the same routing resources, the ability to replace a defective row/column containing wires of any wire length with a spare row/column is guaranteed. This eliminates the dependency of wire length on defect correction for the spare row and column architecture.

For FGR, increasing wire length increases both the fanout and fanin of all routing wires. These increase because long wires naturally have a greater number of midpoint locations. This increases the MFFR and negatively impacts yield.

Current FPGA routing architectures utilize multiple wire lengths within their routing architecture. In FGR, wires of different lengths are modelled as disjoint routing networks[1]. Each routing network will have its own unique set of spare resources. Defect correction is restricted to the individual routing networks. Of course, these disjoint networks "join together" at the connection blocks which have been modified to correct for defects in any of them.

5.3 Limitations

The yield model does not account for switch area or total die area changes. The area of the switch is an important consideration because larger circuits have a greater probability of being defective than smaller ones. For example, CGR-Gn and CGR-Ln-Sp have significantly larger switch areas because of the necessary bypass circuitry. This is not modelled, so the presented yield for CGR is over-estimated. This is also true for the various FGR implementations.

Power/ground shorts have been ignored in the fault simulation. These defects cannot be tolerated in either architectures. Defects in the logic blocks have also been ignored because of lack of real-life manufacturing data and because this thesis is only concerned with interconnect faults, which are more difficult to tolerate.

Routing tracks within the same channel are assumed to be laid out beside one another. When bridging defects are injected into the model, two adjacent tracks are made unusable. This assumption allows FGR to bypass bridging defects by performing shifts by 2. Larger faults (i.e., 3 wires) would not be tolerable in FGR.

The FGR model assumes that there can be at most one defect per trackgroup. However, certain types of single and double-length faults can in fact be overlapped with one another. For example, two defects can be overlapped if the underlying fault redundant ar-

[1] This is mostly a limitation of academic routing architectures which treat the routing networks of different wire lengths as disjoint networks.

chitectures supports bridging defects, the faults themselves are not bridging defects, and the defects do not reside on adjacent tracks. Provided that these conditions are met, the defect on the lower track can be avoided using a “+1” shift while the one on the higher track can be avoided with a “+2” shift. Also, defects in the same tracks should be tolerable.

When computing yield, the delay bypass path of the omux is assumed to be not used. If it is utilized, either signal timing must be perturbed when correcting a defect or device defects must be limited to one-per-channel.

Defects in CGR are assumed to require either a spare row or a spare column to be tolerated. Some defects require both, hence the yield for this architecture is over estimated.

Lastly, when computing the MFFR for FGR, defects are assumed to be injected into the middle of an infinitely sized FPGA[2]. This results in worst-case MFFRs for the defects. Since the trackgroups near the edge of the FPGA have lower connectivity than those in the center, the use of the worst-case MFFR value for all defects is overly pessimistic.

5.4 Results

The yield estimates for CGR and FGR were obtained through Monte Carlo simulations. For a given number of defects, randomly located faults were injected into the interconnect for 100,000 different FPGA dies. The presented results do not account for intolerable defects such as power/ground shorts.

5.4.1 Switch Implementation

An embedded imux allows certain signals to shift and turn at the same time. This attribute relieves the need to pre-shift certain signals. Reducing the number of wires affected by defect correction results in a lower MFFR and improved yield.

Figure 5.2 shows the comparison between the EM11 and the EN11 architecture for both single-length and bridging fault correction. These architectures differ by imux

[2] Actually, it assumes a torus where the edges wrap around.

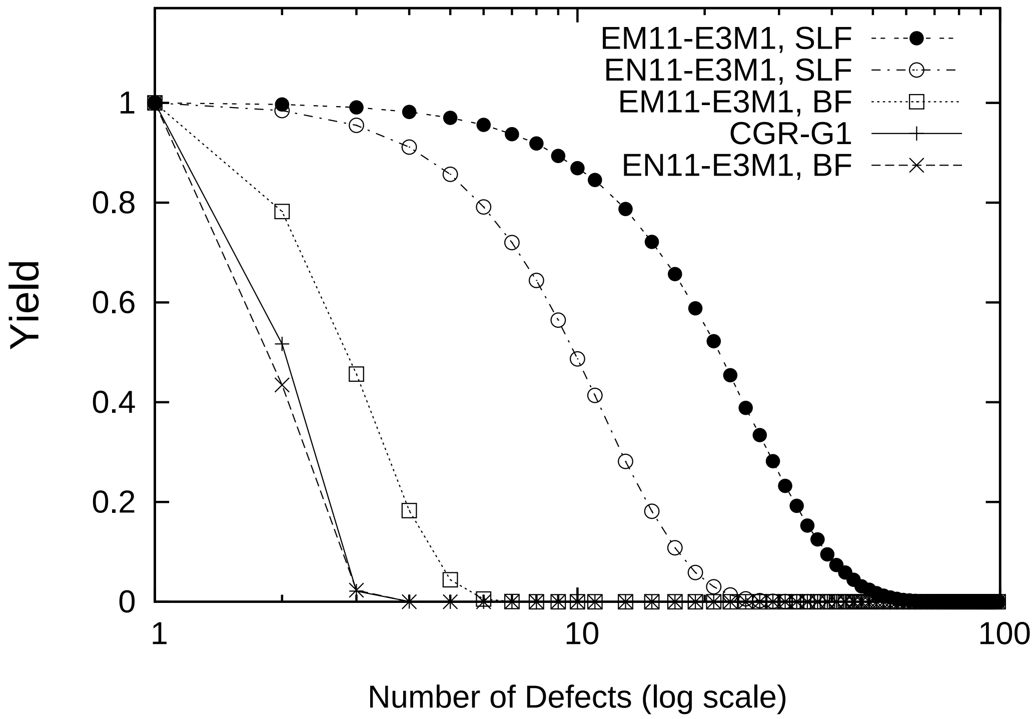


Figure 5.2: Imux implementation (L=4, M = 32)

implementation. The EM11 architecture uses an embedded imux, while the EN11 uses an extracted imux. The embedded imux demonstrates better yield for multiple defects.

The shifting ability of the switches also affects the length of the repair region. With a shorter the repair length, fewer trackgroups need to be pre-shifted and restored. This reduces the MFFR and improves yield. Figure 5.3 shows that the architectures with the greatest degree of shifting ability (EM22, FL22 and EN22) have the highest yield. Note also that architectures EN11-E3M1 has slightly lower yield than the spare row and column technique. The reason for this is that bridging faults require an increased repair length due to its restrictive shifting abilities, producing a large MFFR. The resulting MFFR is sufficiently large that it makes tolerating more than 2 defects within a 32x32 FPGA (with L=4) very difficult.

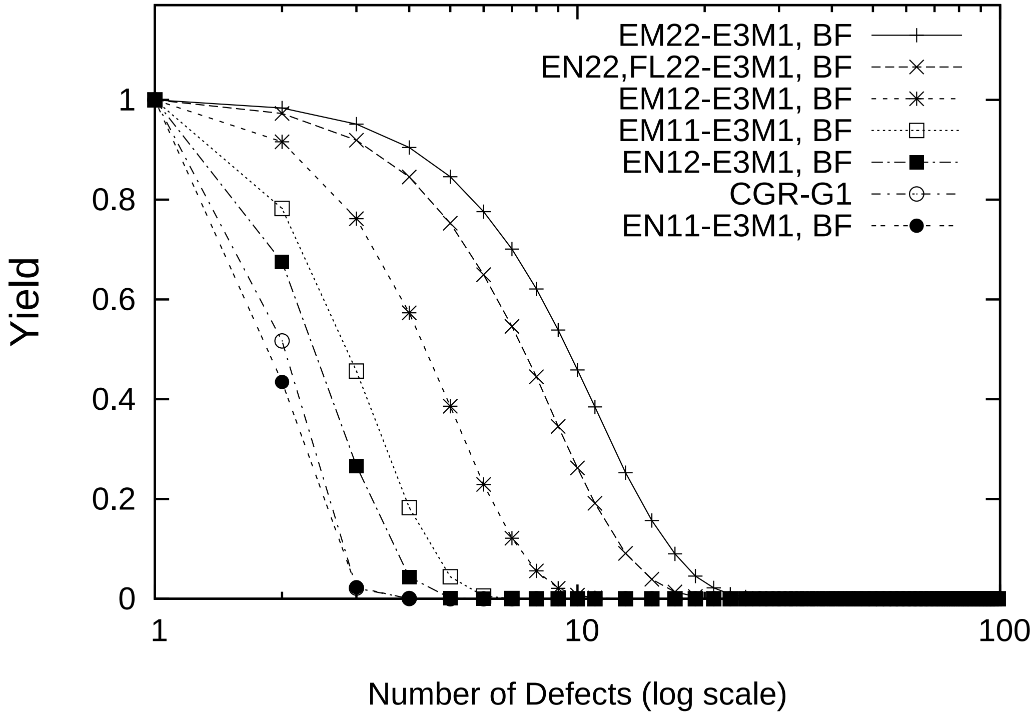


Figure 5.3: Shifting abilities (L=4, M = 32)

5.4.2 Switch Flexibility

The flexibility of the switches also has a significant impact on yield. When the flexibility is increased, the number of fanin and fanouts increased. As mentioned before, this increases the MFFR and reduces yield. Figure 5.4 shows that the architecture with the lowest F_s, E2M1, demonstrates the best yield for both single length faults (SLF) and bridging faults (BF). The reduction in midpoint flexibility improves yield more than the reduction in endpoint flexibility because there are more midpoint than endpoint connections for length 4 wires.

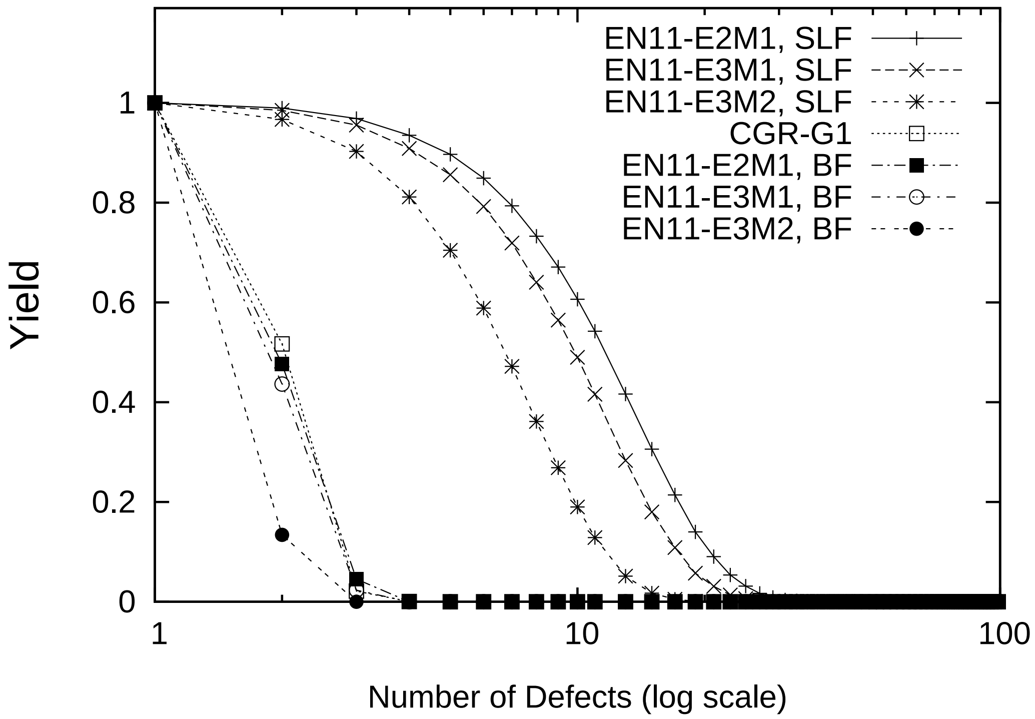


Figure 5.4: Flexibility (L=4, M = 32)

Baseline Architecture

Taking the above results and the area/delay results from Chapter 3, the EM22-E3M1 architecture was selected to be the baseline fine-grain redundancy architecture. This architecture is capable of shifts by +/-2 and +/-1, and has endpoint and midpoint flexibilities of 3 and 1 respectively. In all subsequent results , EM22-E3M1 will simply be denoted as *FGR*.

5.4.3 Fixed Array Size

FGR versus Global Spares

Figure 5.5 presents the yield for an 32x32 FPGA with a number of *additional* global rows/columns. The yield for CGR-Gn remained at 100% until the defect count became

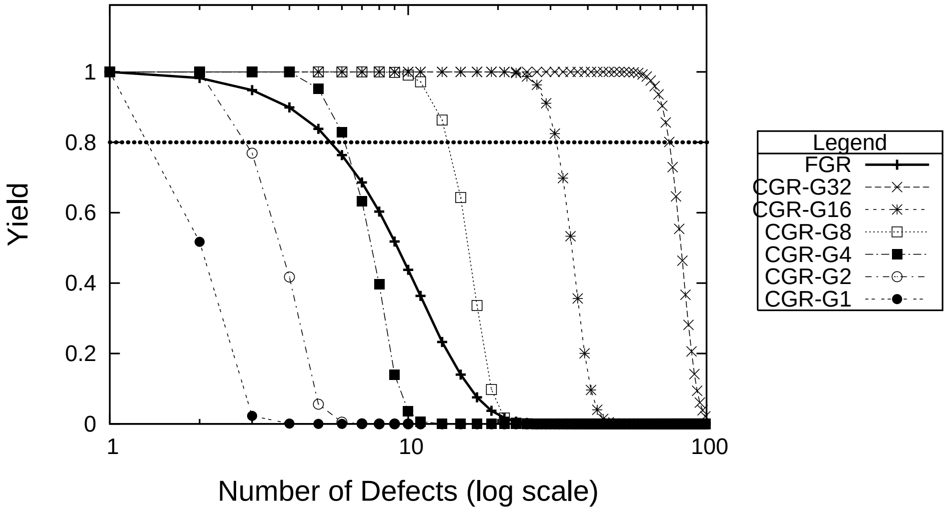


Figure 5.5: Increasing number of global spares (M = 32)

greater than n, the number of the number of spare rows/columns in the architecture[3]. After this threshold, the yield decreases dramatically. Yield for this particular architecture is especially sensitive to the number of spare rows/columns in the system. The figure also shows that there is a significant yield improvement when the number of *global* spares increase from one to two, and that FGR produces similar yields as CGR-G4. This can be observed by noting that both architectures fall below the 80% yield threshold at approximately the same number of defects.

FGR versus Local Spares

The CGR-Ln-Sp architecture demands that defects are spaced far apart from one another. If too many defects reside in the same subdivision, chip failure occurs. The impact of this restriction is shown in Figure 5.6. The figure shows the yield of a 32x32 FPGA with the indicated number of local subdivisions. Each subdivision contains one local spare row/column. Notice that the yield decreases almost immediately and is significantly less than the global

[3] In the worst case, all defects are located in rows/columns. Since there are n spares rows/columns, CGR-Gn can always tolerate n defects.

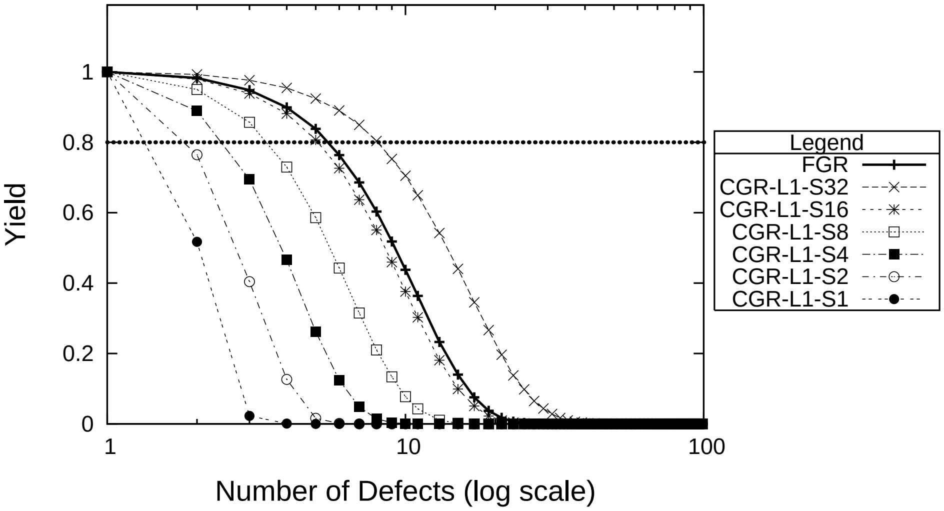


Figure 5.6: Increasing number of local spares (M = 32)

approach. The CGR-L1-S16 produces similar yields to FGR and CGR-G4. It should be noted that CGR-L1-S16 with 1 spare in each subdivision (16 spare rows and 16 spare columns total) is more practical to implement than the amount of additional multiplexing needed by CGR-G4 with 4 global spare rows and columns. Although easier to implement, CGR-L1-S16 more than doubles the area of then FPGA (from 32x32 to 32+16x32+16 or 48x48), making it significantly larger than FGR.

Figures 5.7 and 5.8 presents the yield curves for CGR-Gn and CGR-Ln-Sp, but a larger array size of 256x256 is used. At this array size, the number of defects tolerated by FGR increases and CGR slightly decreases (both local and global). FGR is now approximately equivalent to the yield of CGR-G16 and has higher yield than all implementations of CGR-L1-Sp. Note that more than two global spare rows/columns is impractical (and potentially infeasible) because the necessary wire extensions significantly increase switch area and signal timing. Although the local spare approach avoids this by having only one spare per subdivision, the CGR-L1-S256 architecture has 300% overhead in spare rows and columns and cannot tolerate as many defects as FGR. In comparison, the FGR area overhead is only 50%.

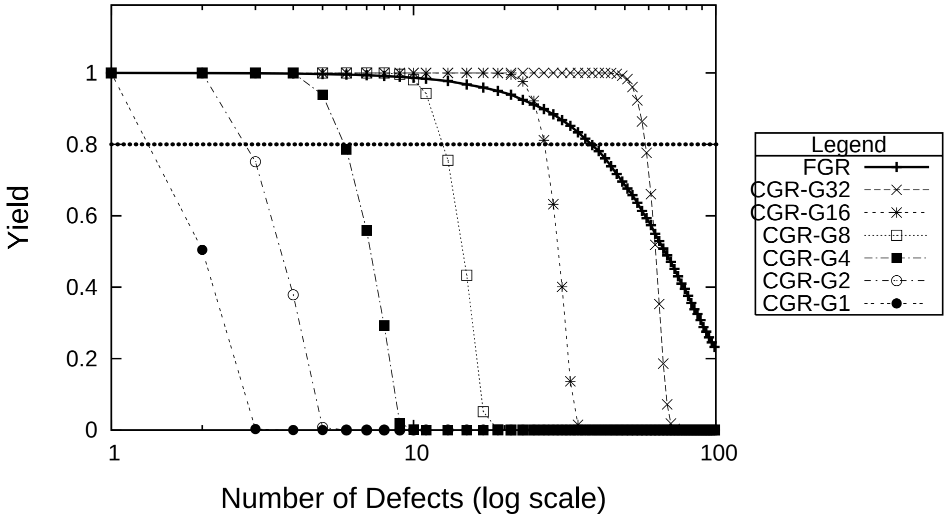


Figure 5.7: Increasing number of global spares (M = 256)

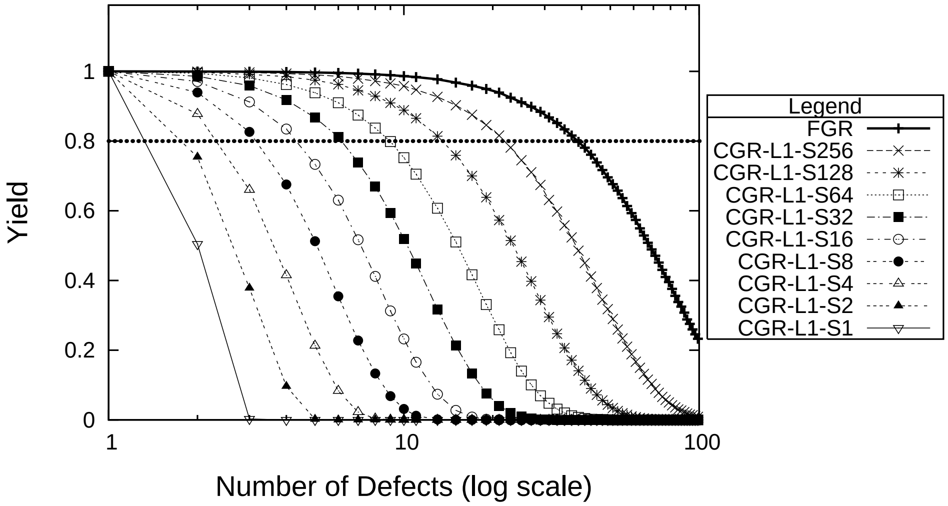


Figure 5.8: Increasing number of local spares (M = 256)

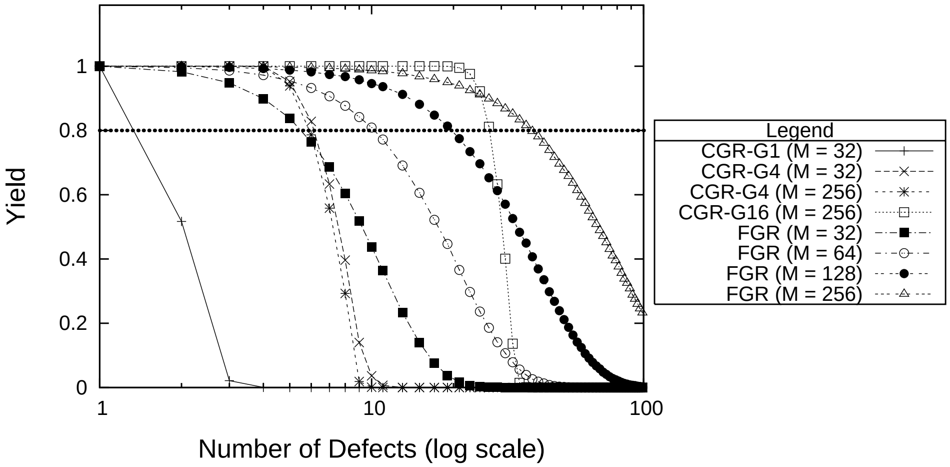


Figure 5.9: Increasing array size for FGR (L = 4)

5.4.4 Increasing Array Size

CGR can tolerate multiple defects in the channel. However, as array size grows, it becomings increasingly unlikely that randomly occurring defects will lie in the same channel. Hence, the yield for CGR with a fixed number of spare rows and columns was observed to be largely independent of array size. The only way to increase yield is through the addition of spare resources. This can be observed in Figure 5.9 where the yield is shown to be similar for CGR-G4 at M=32 and M=256.

For the FGR architecture, the amount of spare resources increases naturally as array size grows. Since the amount of resources needed for defect correction is constant, increasing the number of spare resources translates into the ability to tolerate more defects. This is demonstrated in Figure 5.9 where FGR is shown to tolerate an increasing number of defects as array size grows. To reach a similar level of defect tolerance as FGR at M=256, CGR requires 16 global spares, which is completely infeasible!

Figure 5.10 presents a rough area comparison between FGR and CGR for different values of M. For FGR, the reported area includes **all** necessary shifting multiplexers and spare wires for the given value of M. For CGR-Gn and CGR-Ln-Sp, the “n” and “p” values

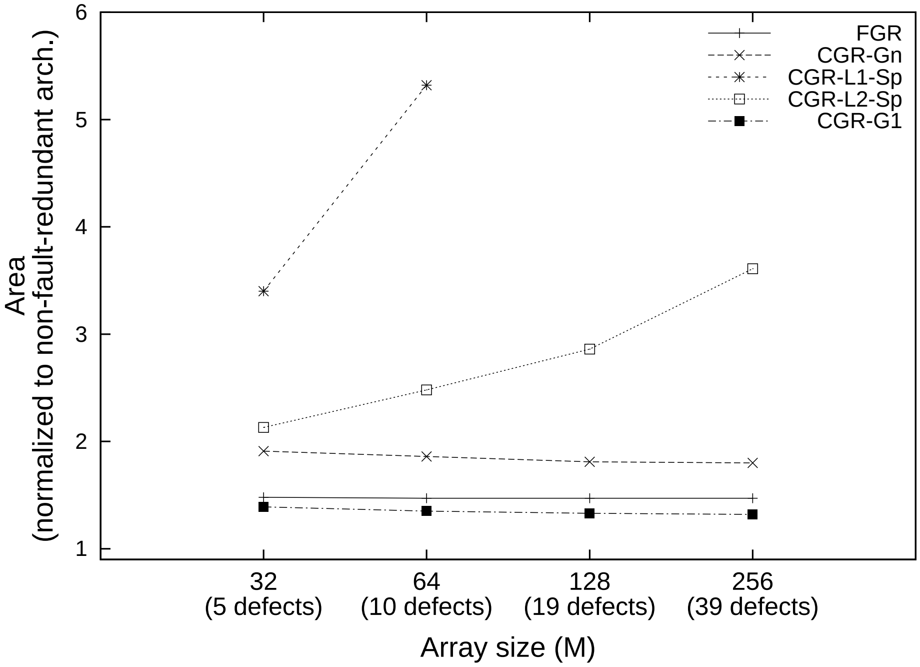


Figure 5.10: Area comparison between FGR and CGR at equal number of defects(L = 4)

were chosen to tolerate the same number of defects as FGR at the 80% yield. The reported CGR-Ln-Sp area includes the base MxM FPGA plus the additional spare rows/columns. To account for the additional bypass circuitry, the scaling factor noted in Section 4.3.3 is used to adjust the area results for both CGR approaches. The figure shows that CGR-Gn, CGR-L1-Sp and CGR-L2-Sp all requires more area overhead to tolerate the same number of defects as FGR can tolerate at 80% yield.

The figure also shows the area overhead for CGR-G1. However, CGR-G1 can only tolerate 1 defect for the different array sizes. A scaling factor of 1.3 is used to approximate CGR-G1 area. Note that the area overhead between CGR-G1 and FGR are similar, and that at large values of M, CGR-G1 has a lower overhead.

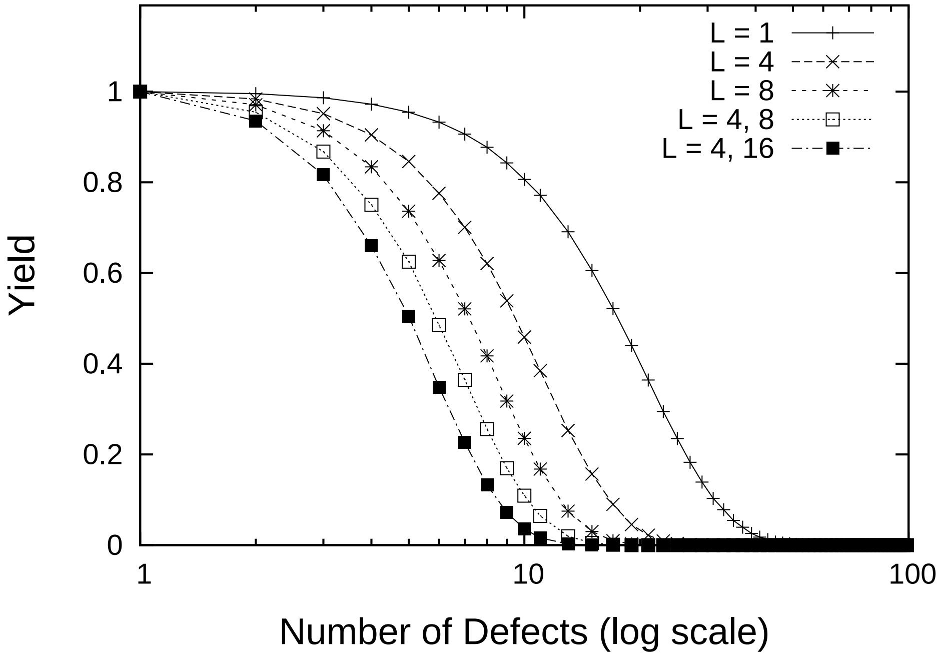


Figure 5.11: FGR yield for different wire lengths (M=32)

5.4.5 Wire Length

Long wires have a greater number of fanins and fanouts. This results in a larger MFFR and, consequently, a yield reduction. Figure 5.11 shows how much the yield for FGR decreases as wire length increases. Also note that the yield for mixed wire length is lower than the yield of the individual wire lengths it is composed of. This is largely the consequence of how mixed wires are implemented and modelled. In the fine-grain architecture, the routing network for different length wires are disjoint.

To put this yield into perspective, area and delay results were computed for the largest MCNC benchmark circuit, *clma*. The results for separate FGR architecture of length 4, 8 and 16 wires are presented in Figure 5.12. The reported numbers have been normalized to an architecture without redundancy at L=4. The channel width was fixed at 224 for all wire lengths because 224 is the minimum channel width needed to route *clma* using length 16 wires. The figure shows that delay increases and area decreases as wire length increases.

The use of the E3M1 architecture can be the potential caused of the alarmingly large delay overhead for length 16 wiring. It is possible that the reduced midpoint flexibility forces nets to take longer (thus slower) and less direct paths. Also, the wires are much longer than needed, slowing nets down. Lastly, routing with L=16 wires at 224 tracks is highly congested, and this increases delay.

In Figure 5.13, the area overhead for *clma* is broken down into the CLB (logic) area, the C block area, the S block area, the spare resources area (includes the spare wires and associated imux and omux) and the shifting multiplexers area (imux and omux for non-spare wires). The CLB and C block have fixed area overhead because array size and channel width are constant respectively. C block represents a large part of circuit area because a wide channel width is used. The S block and shifting multiplexers area shrink as the length of wire increases because longer wires have fewer switching elements at fixed channel widths. Lastly, the spare resources area grows because longer wires have fewer wires per trackgroups, hence the addition of a spare wire is an increasing function of the overall routing area.

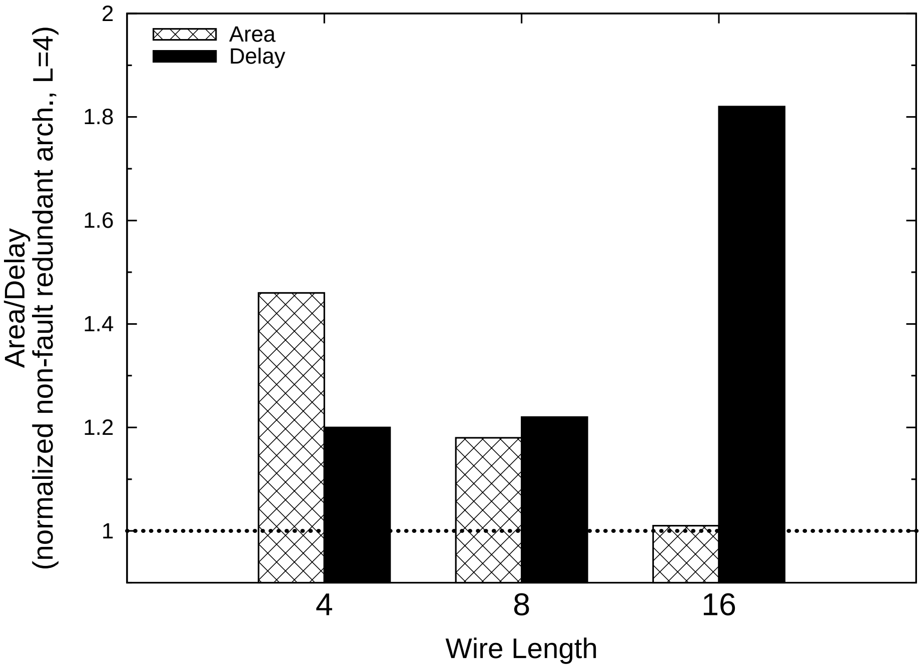


Figure 5.12: Area/delay overhead for *clma*

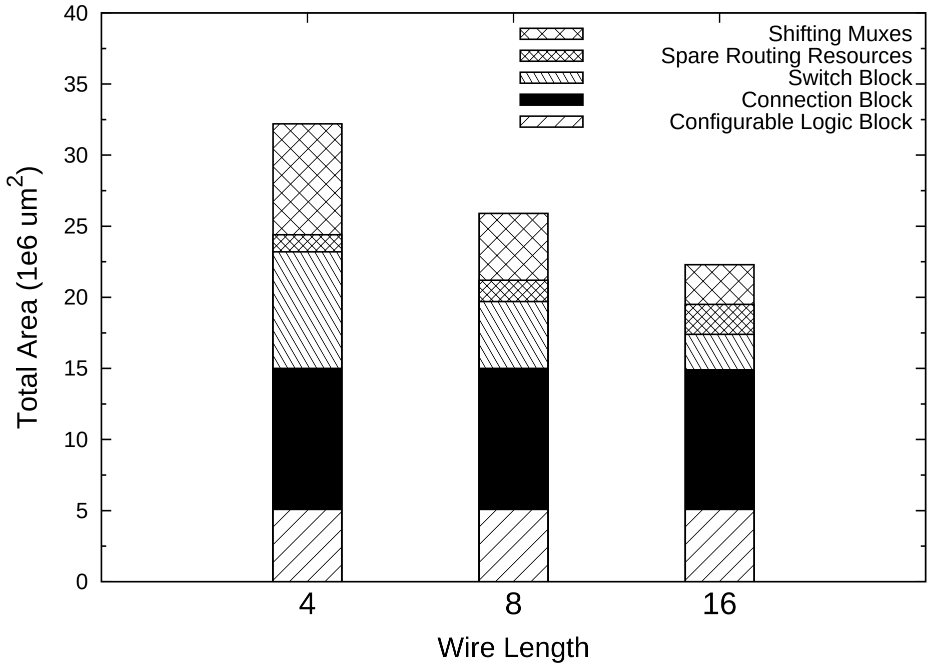


Figure 5.13: Area breakdown of *clma* for different wire lengths at a very wide channel width of 224 tracks

Chapter 6

Conclusion

This thesis presents a new defect-tolerant switch block and connection block architecture, called Fine-grain Redundancy (FGR), that can tolerate an increasing number of permanent manufacturing defects as the FPGA array size scales up. FGR is capable of handling tens of distributed random defects. FGR is compared to a more traditional approach employing coarse-grain redundancy (CGR). The results indicate that CGR does not scale well beyond 2 distributed defects unless significant area overhead is employed.

6.1 Area and Delay

The proposed FGR approach has a true area overhead of approximately 11% and delay overhead of 4% on aggressive applications that do not wish to be defect-tolerant.

When defect-tolerance is desired, it is conventional to include the cost of reserving a spare track. This increases area overhead to 35–50% and delay overhead to 5–20%. However, it should be noted that less aggressive applications will already have these spare (unused) routing tracks available for free, so the actual cost is much closer to the true area overhead.

A range of FGR implementation options that have a range of area and delay costs was also presented. Of these options, EN11-E3M1 has the lowest area, EM11-E3M2 has the lowest delay, and EM22-E3M1 has the highest yield. More detailed rankings of E3M1, the best flexibility option, are shown in Table 6.1 and presented visually in Figure 6.1.

6.2 Yield

This thesis also presents a comparison between CGR and FGR. Both approaches embody the idea of replacing a defective resource with a spare unused one; however the investigation indicates that the choice of defect tolerant architecture has a significant impact on yield and area overhead.

At low defect levels, CGR has a lower area overhead than FGR. Further, for sufficiently low defect levels, the area overhead for CGR diminishes as array size increases. This is not the case for FGR, where the area overhead for this approach is fixed at up to 50% for all array sizes.

Despite the fixed cost of redundancy, FGR demonstrated the ability to tolerate an increasing number of defects as array size grows. This is extremely important as the expected number of defects increase as die size grows and technology feature size shrinks. When comparing CGR and FGR at equal defect levels, CGR actually requires more area overhead to tolerate the same number of defects as FGR.

Other factors that influenced yield are wire length and switch implementation. This study showed that the yield for the FGR approach decreases as wire length or flexibility increases, and if the switch's shifting ability is reduced. These factors were found to increase the MFFR of defects, and thus reduced the number of tolerable defects. This is not so for CGR. Using spare rows and columns for defect correction is wire length and switch implementation independent.

6.3 Future Work

In terms of architectural improvements, future work includes the optimization of of C block design for FGR and the incorporation of the extra circuitry needed for row/column bypasses for CGR. The first is important as careful design of the C block can lead to a significant

Arch	Area	Delay	Area Recovery	Delay Recovery	Yield
EM22	7	3	7	3	1
EN22	4	7	4	7	2
FL22	6	6	5	6	2
EM12	5	2	6	1	4
EM11	2	1	2	2	5
EN12	3	5	3	5	6
EN11	1	4	1	4	7

Table 6.1: Summary ranking of FGR defect-tolerant schemes w/ E3M1

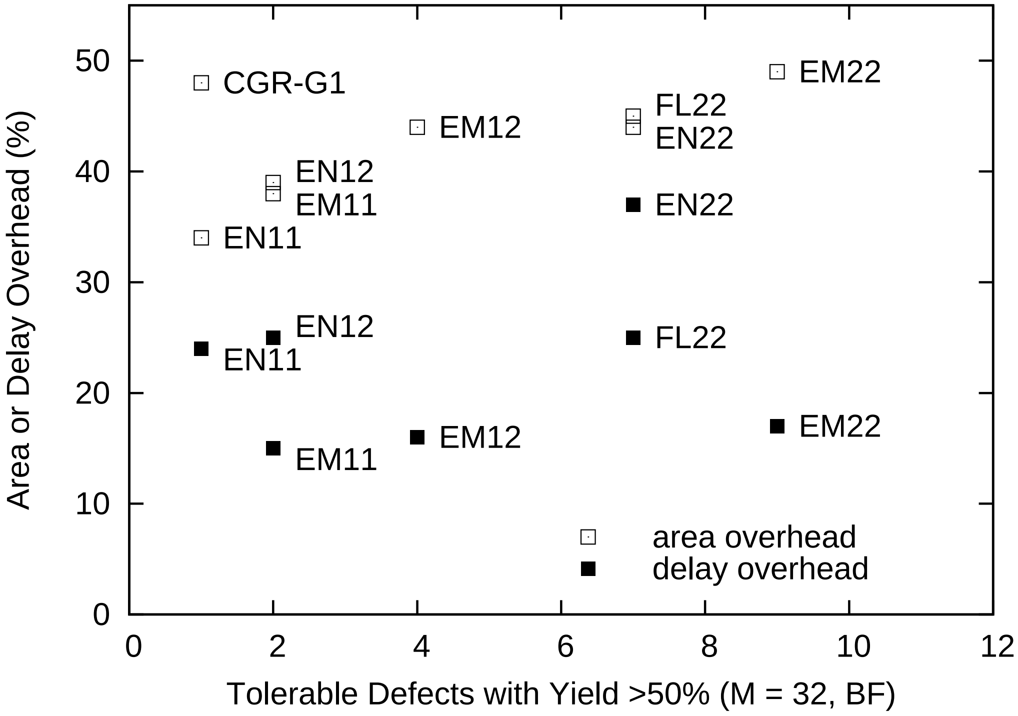


Figure 6.1: Summary of area/delay overhead vs defect tolerance of FGR

reduction in area overhead. The accounting of the extra bypass circuitry is useful as it provides a more accurate area and delay comparison between FGR and CGR.

Future work in the software domain can be further divided into two categories: support software and yield model improvements. To provide a complete defect tolerant solution, it is necessary to develop a suite of support tools. The functionality of these tools includes strategies for defect diagnosis, defect map generation, defect maps management and the application of defect correction. To better estimate yield, chip area should be incorporated into the yield model. Additionally, different kinds of defects should be injected into the model (as oppose to the current implementation where only worst-case bridging defects are used). Ideally, the model should be supplemented with manufacturing data to produce the most accurate estimates.

Bibliography

[1] Miron Abramovici, John M. Emmert, and Charles E. Stroud. Roving STARs: An integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems. In *Proc. of the The 3rd NASA/DoD Workshop on Evolvable Hardware*, pages 73–92. IEEE Computer Society, 2001.

[2] Altera Corp. Altera's patented redundancy technology dramatically increases yields on high-density APEX 20KE devices. In *Press Release*, Nov. 27, 2000.

[3] Altera Corp. In *United states patents #6,034,536, #6,166,559, #6,337,578, #6,344,755, #6,600,337 and #6,759,871*, 2000–2004.

[4] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Boston, 1999.

[5] Nicola Campregher, Peter Y.K. Cheung, George A. Constantinides, and Milan Vasilko. Analysis of yield loss due to random photolithographic defects in the interconnect structure of FPGAs. In *Int'l. Symp. FPGA*, pages 138–148, February 2005.

[6] C. Carmichael. Triple module redundancy design techniques for Virtex FPGAs. In *Xilinx Application Notes, XAPP197 (v1.0)*, 2001.

[7] C. Carmichael, M. Caffrey, and A. Salazar. Correcting single-event upsets through Virtex partial configuration. In *Xilinx Application Notes, XAPP216 (v1.0)*, 2000.

[8] Wu-Tung Cheng. Silicon diagnosis. In *International Test Conference*, 2003.

[9] Collaborative Benchmarking Laboratory. Lgsynth93 benchmark set: Version 4.0. Technical report, North Carolina State University, 1993.

[10] Altera Corp. Stratix II device handbook, vol. 1. 2005.

[11] Crosspoint Solutions Inc. FPGA redundancy. In *United states patents #5,777,887*, 1998.

[12] Andre DeHon and Michael J. Wilson. Nanowire-based sublithographic programmable logic arrays. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th International symposium on Field Programmable Gate Arrays*, pages 123–132, New York, NY, USA, 2004. ACM Press.

[13] Abderrahim Doumar and Hideo Ito. Design of switching blocks tolerating defects/faults in FPGA interconnection resources. In *Proc. 15th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, pages 134–142. IEEE Computer Society, 2000.

[14] Abderrahim Doumar, Satoshi Kaneko, and Hideo Ito. Defect and fault tolerance FPGAs by shifting the configuration data. In *Int'l Symp. on Defect and Fault-Tolerance*, pages 377–385. IEEE Computer Society, 1999.

[15] Mohamed A. Elgamel, Kannan S. Tharmalingam, and Magdy A. Bayoumi. Crosstalk noise analysis in ultra deep submicrometer technologies. In *ISVLSI '03: Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'03)*, page 189, Washington, DC, USA, 2003. IEEE Computer Society.

[16] Rudy Garcia. Rethink fault models for submicron-IC test. *Test & Measurement World*, October 2001.

[17] Scott Hareland, Jose Maiz, Mohsen Alavi, Kaizad Mistry, Steve Walstra, and Changhong Dai. Impact of CMOS process scaling and SOI on the soft error rates of logic processes. In *Proc. of the IEEE Nuclear and Space Radiation Effects Conference*, pages 73–74, 2001.

[18] F. Hatori, T. Sakurai, K. Nogami, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, A. Muroga, 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.

[19] C.K. Hu and J.M.E. Harper. Copper interconnect: Fabrication and reliability. In *International Symposium on VLSI Technology, Systems, and Applications*, 1997.

[20] W.-J. Huang and E.J. McCluskey. Column-based precompiled configuration technique for FPGA fault tolerance. In *Proc. IEEE Symp. Field Programmable Custom Computing Machines*, 2001.

[21] Xilinx Inc. Quintuple modular redundancy for high reliability circuits implemented in programmable logic devices. In *United states patents #6812731*, 2004.

[22] Xilinx Inc. Virtex-II Pro and Virtex-II Pro X platform FPGAs: Complete data sheet, version 4.3. 2005.

[23] John Lach, William H. Mangione-Smith, and Miodrag Potkonjak. Efficiently supporting fault-tolerance in FPGAs. In *Int'l Symp. on Field Programmable Gate Arrays*, pages 105–115. ACM Press, 1998.

[24] Vijay Lakamraju and Russell Tessier. Tolerating operational faults in cluster-based FPGAs. In *Int'l Symp. on FPGAs*, pages 187–194, 2000.

[25] Guy Lemieux, Edmund Lee, Marvin Tom, and Anthony Yu. Directional and single-driver wires in FPGA interconnect. In *Int'l Conf on Field-Programmable Technology*, 2004.

[26] Guy Lemieux and David Lewis. *Design of Interconnection Networks for Programmable Logic*. Kluwer Academic Publishers, Boston, 2004.

[27] David Lewis, Elias Ahmed, Gregg Baeckler, Vaughn Betz, Mark Bourgeault, David Cashman, David Galloway, Mike Hutton, Chris Lane, Andy Lee, Paul Leventis, Sandy Marquardt, Cameron McClintock, Ketan Padalia, Bruce Pedersen, Giles Powell, Boris Ratchev, Srinivas Reddy, Jay Schleicher, Kevin Stevens, Richard Yuan, Richard Cliff, and Jonathan Rose. The Stratix II logic and routing architecture. In *Int'l. Symp. FPGA*, pages 14–20, February 2005.

[28] J. Liu and S.J. Simmons. BIST-diagnosis of interconnect fault locations in fpga's. In *Canadian Conference on Electrical and Computer Engineering*, 2003.

[29] Sani R. Nassif. Within-chip variability analysis. *IEEE Int. Electron Devices Meeting*, December 1998.

[30] M.Y. Niamat, R. Nambiar, and M.M. Jamali. A BIST scheme for testing the interconnects of SRAM-based FPGAs. In *Midwest Symposium of Circuits and Systems*, 2002.

[31] Jonathan Rose and Stephen Brown. Flexibility of interconnection structures in field--programmable gate arrays. *Journal of Solid State Circuits*, 26(3):277–282, 1991.

[32] R. Singh, V. Parihar, K.F. Poole, and K. Rajkanan. Semiconductor manufacturing in the 21st century. *Semiconductor Fabtech 9th Edition*, pages 223–232, 1999.

[33] C. H. Stapper. Modeling of integrated circuit defect sensitivities. *IBM Journal of Research and Development, vol. 27*, pages 549–557, 1983.

[34] J. H. Stathis. Physical and predictive models of ultrathin oxide reliability in CMOS devices and circuits. In *Proceedings of the 2001 IEEE International Reliability Physics Symposium*, pages 132–149, 2001.

[35] Xilinx, San Jose, CA. *EasyPath Solutions*, 2005. `http://www.xilinx.com/products/easypath/`.

[36] A.J. Yu and G.G.F. Lemieux. Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement. In *to appear in Int'l. Conf. on Field Programmable Logic and Applications*, 2005.

[37] A.J. Yu and G.G.F. Lemieux. FPGA defect tolerance: Impact of granularity. In *to appear in Int'l. Conf. on Field-Programmable Technology*, 2005.