國立交通大學

電子工程學系 電子研究所

碩 士 論 文

應用於三維可程式邏輯閘陣列之

容錯架構探索暨快速重組態演算法

Fault Tolerant Architectural Exploration and Fast Reconfiguration Algorithm for 3D FPGAs

研 究 生：彭晧凌

指導教授：黃俊達 博士

中 華 民 國 一００ 年 十一 月

應用於三維可程式邏輯閘陣列之
容錯架構探索暨快速重組態演算法

Fault Tolerant Architectural Exploration and Fast Reconfiguration Algorithm for 3D FPGAs

研究生：彭晧凌　　Student: Hao-Lin Peng

指導教授：黃俊達 博士　　Advisor: Dr. Juinn-Dar Huang

國立交通大學

電子工程學系 電子研究所

碩士論文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical & Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Electronics Engineering & Institute of Electronics

September 2011

Hsinchu, Taiwan, Republic of China

中華民國一００年十一月

應用於三維可程式邏輯閘陣列之
容錯架構探索暨快速重組態演算法

研究生：彭晧凌　　指導教授：黃俊達 博士

國立交通大學
電子工程學系 電子研究所碩士班

摘　　要

當二維整合電路遇到瓶頸時，三維的製造技術被視為一個很好的解決方案，它是藉由堆疊多個晶粒(die)至單一晶片(chip)並利用直通矽穿孔(through-silicon vias, TSVs)做為垂直方向的連接所完成的。三維的可程式邏輯閘陣列(3D FPGAs)可以被藉由將二維的可程式路由切換器擴展為三維來製作。規律的可程式邏輯閘陣列架構提供了固有不被使用的冗餘元件，這些元件可以被使用於容錯(fault tolerance)的需求，將其視為備用的元件，一旦有任何元件發生故障，則可以利用可程式邏輯閘陣列重新配置(reconfiguration)的功能將故障的元件置換到無故障的備用元件上以達到修復的效果。首先我們針對三維可程式邏輯閘陣列提出了一個考慮到故障元件對於固有冗餘資源需求的重新配置演算法，藉由部分重新配置故障的可程式邏輯單元(CLBs)到無故障的可程式邏輯單元之功能來避免發生故障。實驗結果顯示相較於之前的容錯方法，我們可以提高容錯的修復成功率而且不會有明顯的電路速度衰減。另外針對三維的可程式邏輯閘陣列提出預先佈置冗餘元件的概念進行架構探索，在權衡預先配置與其產生的時間延遲兩項因素下找出具容錯能力較佳的架構。

Fault Tolerant Architectural Exploration and Fast Reconfiguration Algorithm for 3D FPGAs

Student: Hao-Lin Peng Advisor: Dr. Juinn-Dar Huang

Department of Electronics Engineering & Institute of Electronics
National Chiao Tung University

Abstract

Three-dimensional (3D) manufacturing technologies, which stack multiple dies within a single chip and utilize through-silicon vias (TSVs) as vertical connections, are considered as promising solutions to the bottlenecks in 2D integrated circuits. The 3D field programmable gate array (FPGA) can be realized by extending the 2D programmable routing switches to the 3D one. The architectural regularity of FPGAs provides an easy way to allocate inherent redundancy resources (spares), which can be used for fault tolerance. Faulty FPGAs can be repaired by replacing the faulty resources with redundancies. At first, we propose a demand-aware fault-tolerant reconfiguration algorithm for 3D FPGAs that partially remaps the functionality of faulty CLBs to fault-free CLBs to avoid functional faults. Experimental results show that our method can increase the success rate of fault repair without significant timing degradation compared to previous works. Furthermore, we also propose a fault-tolerant pre-allocation concept that performs an architectural exploration for 3D FPGAs, and picks out the most appropriate architectures with a better balance between the success rate of repair and the timing degradation.

Acknowledgment

終於走到這一步了，這次論文能夠完成，得力於很多人的幫助，如果沒有他們的支持與幫忙，我想致謝也只能繼續放在心中了。首先我要感謝我的指導教授，黃俊達老師，總是以最嚴謹的觀點和最大的耐心來教導我，讓我學習到面對問題時能夠以更多的角度去思考它，此外，老師提供了最舒適的實驗環境和最自由的學習風氣，使的我們在做研究的過程中更為順利。

接著我要感謝我爸媽，總是支持我，做我的後盾，不斷的給予安慰讓我能夠撐過這段日子。

最感謝的是陳嘉怡學姊(形姊)和黃雅詩學姊(莎姊)，用無限的愛與包容在指導我，在我遇到瓶頸時給予最大的支持與鼓勵，一路的陪伴著我不離不棄，真的是頗為感動，感激之情，我想並非以三言兩語可以表達。劉揚翔學長(牛哥)、劉家宏學長(宏哥)，時常給與關心並且抽空陪我到處走走散散心，感謝你們。

實驗室的同學張瀚元、王峻澤是患難見真情的最佳典範，在做研究的過程中一起討論，一起甘苦，分擔了我不少的壓力。隔壁實驗室的張舜婷同學，因為修課有幸認識你，常常聽我抱怨真是辛苦你了。也感謝學弟劉廣正、謝明廷，在口試的時候幫忙打理一些事務，最後感謝實驗室的其他學長學弟妹，能夠聚在一起就是緣分。

Content

List of Tables

List of Figure

Chapter 1
Introduction

1.1 3D Integrated Circuits

Smaller feature size and increasing transistor counts allow the implementation of more complex and larger designs. However, as process technology scaling continues, manufacturing yield becomes increasingly low. And the global interconnect has become domination the delay of circuits, as shown in Figure 1. The delay of global interconnects is much larger than that of gates at 32nm process.

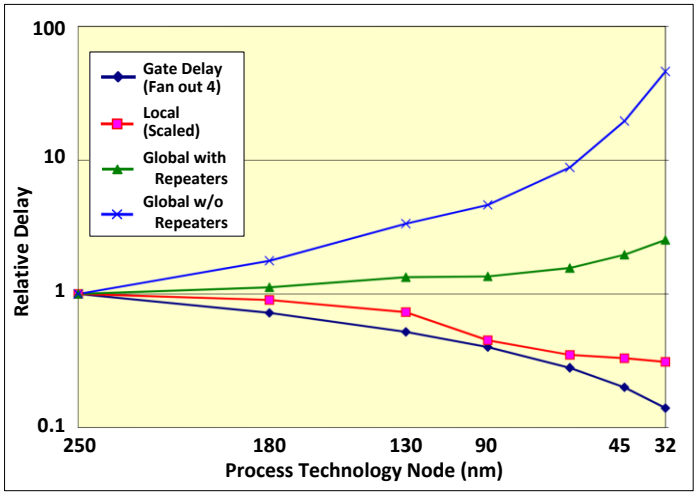


Figure 1. Relative delay vs. feature size [1].

Three-dimensional (3D) manufacturing technologies are viewed as promising solutions to the bottlenecks in 2D integrated circuits. It has many advantages such as high density, low power and high performance. It is realized by wafer/die bonding techniques [2][3]; the communication between different layers is accomplished through vertical signaling, so the global interconnect is significantly reduced, as shown in Figure 2.

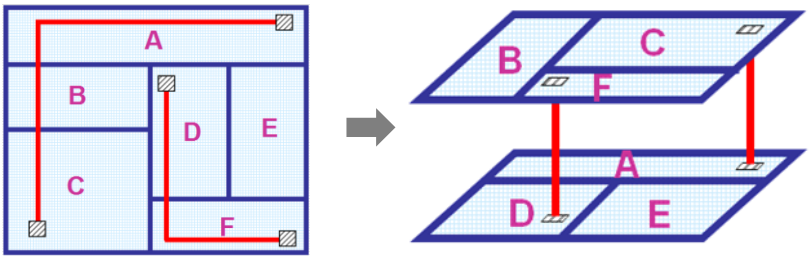


Figure 2. Global interconnects before and after 3D integration.

There are two methods of stacking chips SiPs (system-in-package) and PoPs (package-on-package) which are used for several years [4]–[6]. Chips are stacked and use wire-bonding for vertical signal links while packaging as shown in Figure 3.

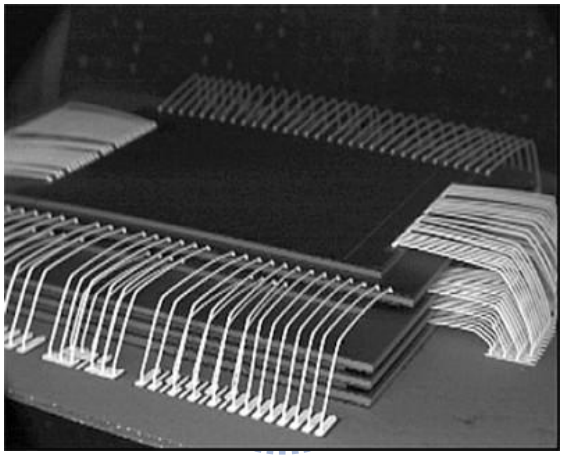


Figure 3. Wire bonding technology [7].

However, it has some drawbacks. Inter-layer connections are restricted on the periphery of the chip; it takes longer communication path between devices. This problem can be resolved by through-silicon via (TSV) technology as shown in Figure 4. The TSV-based 3D ICs stack multiple dies on a single chip and use inter-die vias for vertical connections. These vias can be located almost everywhere within a chip. Though the benefits offered by TSVs are extremely attractive, such as shorter global interconnects [8]–[11], lower interconnect power [12], smaller footprint [13] and better heterogeneous integration [2]. There are still many challenges of TSV-based 3D integration, including reliability, yield [13], power density, and above all, the huge area cost.

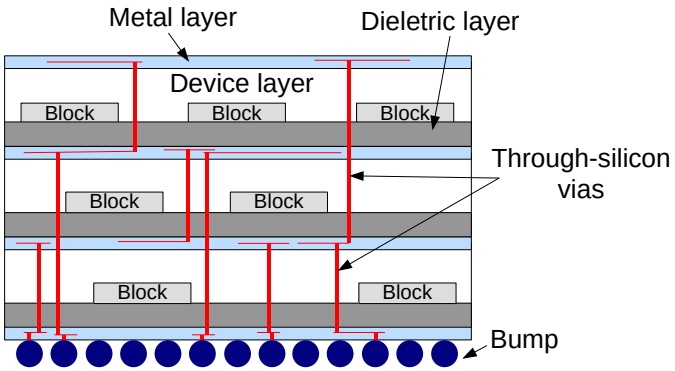


Figure 4. TSVs technology.

Field Programmable Gate Arrays (FPGAs) is an integrated circuit designed to be programmed by the customer or designer after manufacturing. There are many applications of it such as data processing, networks, and other industrial fields. The reconfigurability of FPGAs makes faster time-to-market and mitigates unforeseen design errors. Since 3D integration technology provides several unique advantages compared with the conventional 2D one, so the FPGAs are also extended from 2D to 3D.

1.2 Field Programmable Gate Array

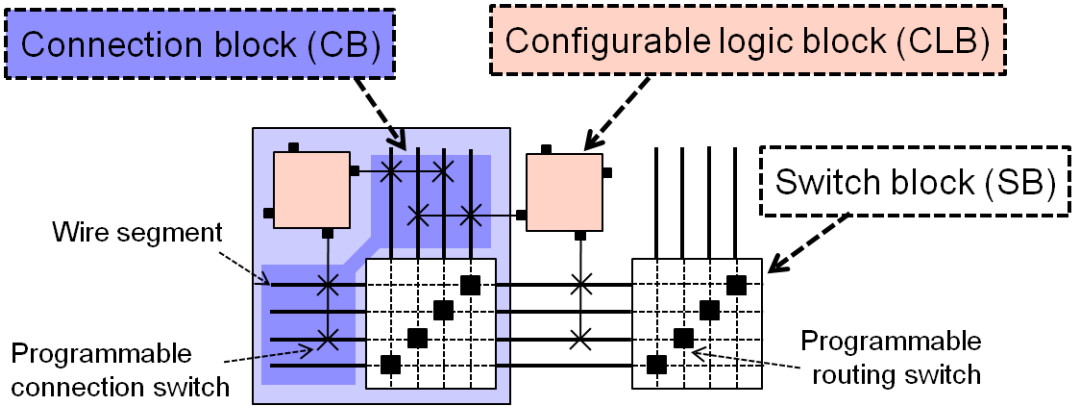


Figure 5. Tile structure.

An FPGA is a regular tile array, as shown in Figure 5. The major component is configurable logic blocks (CLBs). Each CLB contains multiple basic logic elements (BLEs) and can be programmed to implement any boolean function of up to *J* inputs. To avoid confusion between the CLBs of a netlist and the physical CLBs on FPGAs,

the CLBs of the netlist will be called *blocks*. Another major component of FPGA is the interconnecting resource such as wire segments, connection boxes, and switch boxes which connect complex routing.

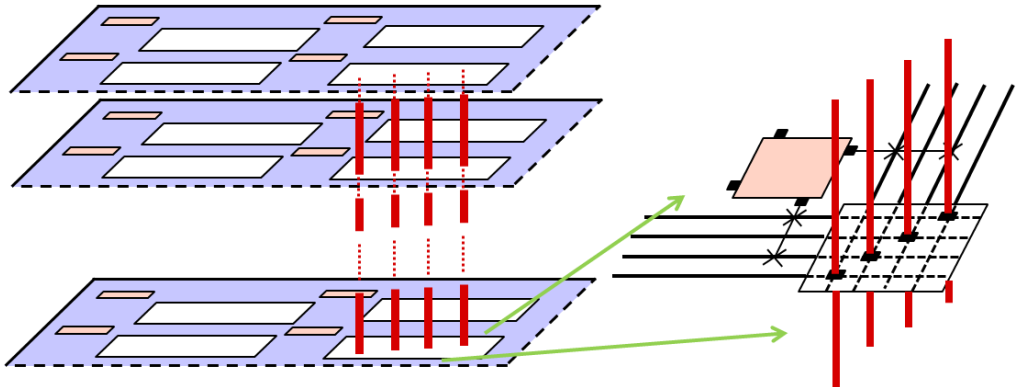


Figure 6. 3D FPGA architecture.

The 3D FPGA architecture is shown in Figure 6. It consists of several identical 2D FPGA layers and the vertical inter-layer communication is accomplished through extending the signal switching scheme of SBs from 2D to 3D while TSVs are used as the vertical links between different layers [14].

1.3 Yield Issues

As process technology scaling continues, integrated circuits face greater challenges from faults, process variability and reliability; therefore, manufacturing large fault-free integrated circuits becomes increasingly difficult.

The failure rate may be observed over the entire life of the circuits, as shown in Figure 7 [15]. The curve can be divided into three stages:

i) Infant – faults cause of material or manufacturing and can be exhibited as circuit signals which are stuck-at 0 or 1 or switch too slowly to meet the timing specification.

ii) Random – faults cause of improper operation or external factor such as SEUs (single event upsets) and SETs (single event transients), caused by certain types of

radiation.

iii) Wearout – faults cause of losses or degradation, the permanent deterioration of a circuit over time, resulting in a negative impact on performance such as electromigration and hot-carrier effects.

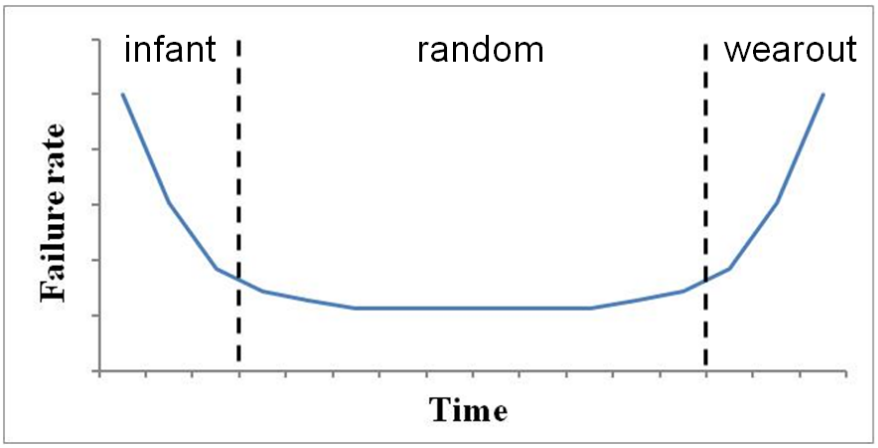


Figure 7. The failure rate of electronic devices varies over time.

The exact level of fault densities is unknown, but it is usually assumed that 1-15% [16] of resources on a chip may become fault and the estimated future effective yields of non-fault-tolerant FPGAs are shown in Table 1 [17]. Regardless of reference yields, future yields always decrease and the reduction is more noticeable for low reference yields. For example, if the yield of current FPGAs is 75%, the yield will be only 21% at 2021. As a result, fault tolerant methods are truly needed.

Table 1. Estimated future effective yields of FPGAs.

	Years				
Ref. yield	**2009**	**2012**	**2015**	**2018**	**2021**
70%	70%	60%	46%	28%	12%
75%	75%	67%	55%	38%	21%
80%	80%	74%	64%	52%	37%
83%	83%	77%	69%	59%	47%

1.4 Previous Work and Motivation

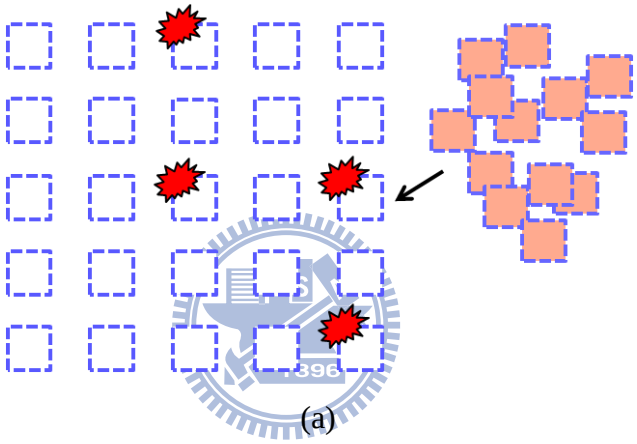
The reconfugurability of FPGAs decreases the time-to-market and mitigates unforeseen design errors and the architectural regularity of FPGAs provides inherent redundancy resources (CLB utilizations are between 70–80% [18][19]) which can be exploited for fault tolerance and yield enhancement. Fault tolerance of FPGAs is discussed for several years; the fault tolerance methods can be categorized into two different levels:

i) *Hardware level* – the hardware-level repair is to reserves redundant resources [20]–[22], which are used to replace faulty resources by re-routing their connections. The redundancy introduced has area overhead, and these methods are limited in the number of faults that can be tolerated, as shown in Figure 8. For ease of exposition, we will refer to a block mapped on a faulty CLB as a faulty block.

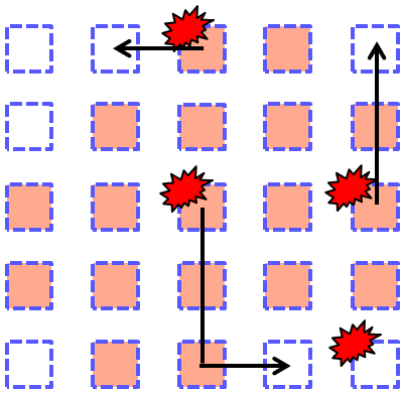
The faulty block at the first row can be repaired by re-routing its connections to the rightmost redundant CLB, and the faulty CLB at the last row is a unmapped, so we do not need to repair it; however, the number of faulty blocks at the third row is more than the number of spare CLB, i.e., two faulty blocks versus one spare CLB, so that this circuit can not be completely repaired.

Figure 8. Reserve redundant resources for fault tolerance.

ii) *Configuration level* – the configuration-level repair is to map the netlist onto a set of fault-free resources by utilizing the spares of the FPGA and no need to have additional redundancy [19][23]–[25]. This kind of repair method can be divided into i) *resynthesis* – redo circuit placement and routing regarding the faulty resources unable to be mapped and ii) *incremental mapping* – partially reconfigure the design to avoid the faulty resources, as shown in Figure 9. These techniques provide a tradeoff between configuration time and timing degradation. An elaborate configuration procedure results in low timing degradation at the cost of high runtime and vice versa.



(a)



(b)

Figure 9. (a) Resynthesis. (b) Incremental mapping.

Here we introduce one of the configuration-level methods named ripple move reconfiguration algorithm [23]. The faulty block is reconfigured by moving it to adjacent CLBs along a path from faulty and mapped CLB to non-faulty spare CLB.

The faulty block is iteratively moved to spare CLB in order of timing criticality. For each block, a directed acyclic graph (DAG) is constructed. Nodes represent CLBs; edges are weighted with the delay required upon moving the node from existing location to the adjacent CLB. In the DAG, the faulty and mapped CLB is the source and the *K* nearest non-faulty spare CLBs are modeled as the destinations where *K* is a parameter determining the quality of reconfiguration. A greater value of *K* represents a DAG with more destinations, which also implies that this DAG has higher probability to find the solution but also increases the problem size. Finally, the shortest path algorithm is applied to find the reconfiguration path, and then the faulty block is reconfigured by moving it to adjacent CLBs along this path. The procedure is iteratively invoked for each faulty block until all faults are successfully reconfigured or no path is found resulting in reconfiguration fail, as shown in Figure 10.

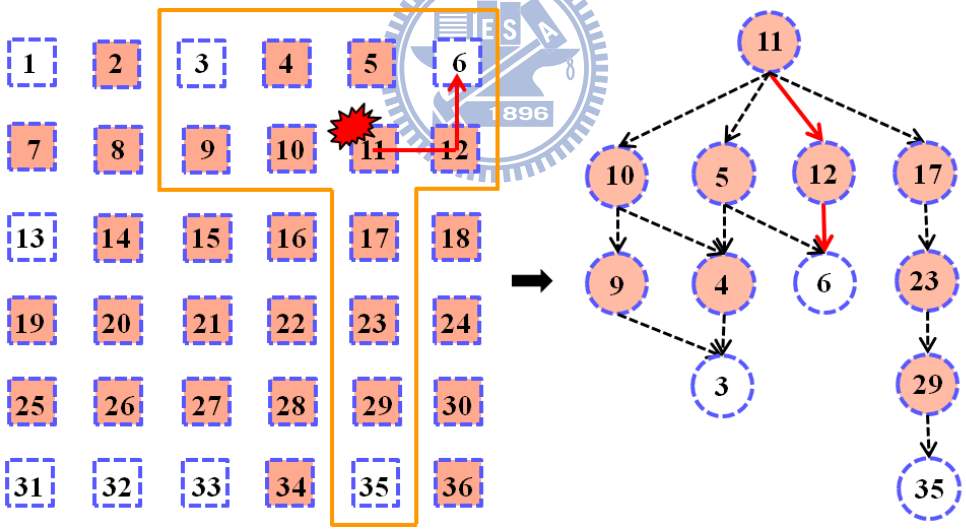


Figure 10. The ripple move reconfiguration algorithm.

However, the algorithm only finds a locally optimal solution at each iteration. An example shown in Figure 11, assume the criticality of the three faulty and mapped CLBs is in order of $\{b_{F8}, b_{F10}, b_{F15}\}$, the value of *K* is 3 and the edge cost is the Manhattan distance between two CLBs. At the first iteration, a DAG is constructed with b_{F8} as source and $\{b_{S1}, b_{S3}, b_{S13}\}$ as destinations and the shortest path is found

{b_{F8}, b_7, b_{S13}}. Second, another DAG is constructed with CLB 10 as source and {b_{S1}, b_{S3}, b_{S6}} as destinations with the shortest path {b_{F10}, b_9, b_{S3}}. In the last DAG, b_{F15} is set as the source and {b_{S1}, b_{S25}, b_{S32}} are the destinations with the shortest path {b_{F15}, b_{14}, b_{20}, b_{26}, b_{S32}}. The total cost of the example is 2+2+4=8.

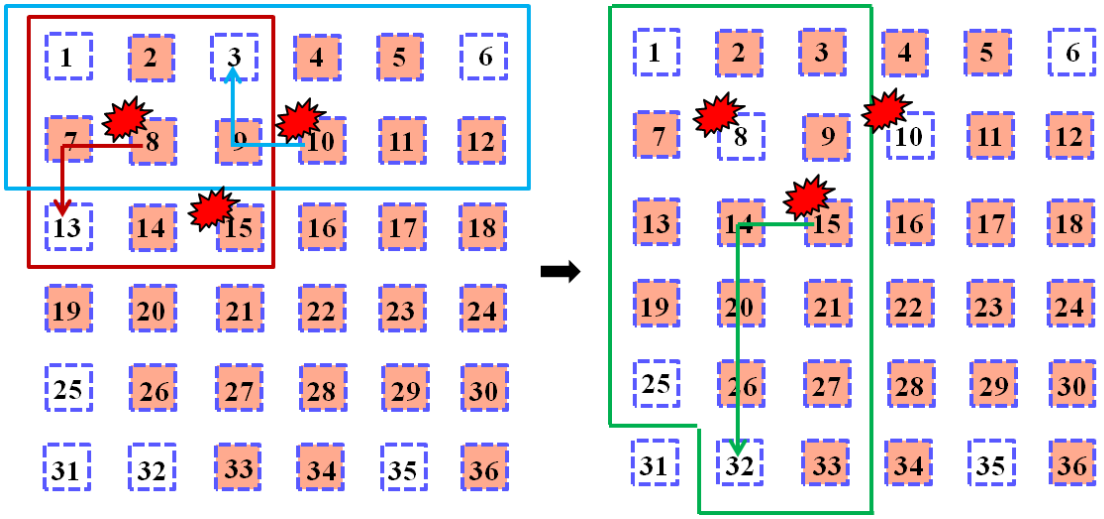


Figure 11. The reconfiguration algorithm without considering the demand issue.

From the example mentioned above, the distance between b_{F8} and any one of {b_{S1}, b_{S3}, b_{S13}} is 2, and b_{S13} is chosen as a destination. It can be observed that b_{F15} is more demands for b_{S13} because the number of non-faulty CLBs in its neighborhood is less than b_{F8}'s. If b_{S13} is mapped, b_{F15} has to choose a farther non-faulty CLB as destination. Therefore, it results in higher timing degradation.

With the same example, a different reconfiguration result is shown in Figure 12. Now we consider that if the shortest path is found {b_{F8}, b_7, b_{S1}} then the shortest path for b_{F15} is {b_{F15}, b_{14}, b_{S13}}, the total cost of the example becomes 6. The result showed that previous work can not handle this problem very well and therefore we want to develop a reconfiguration algorithm from more global point of view.

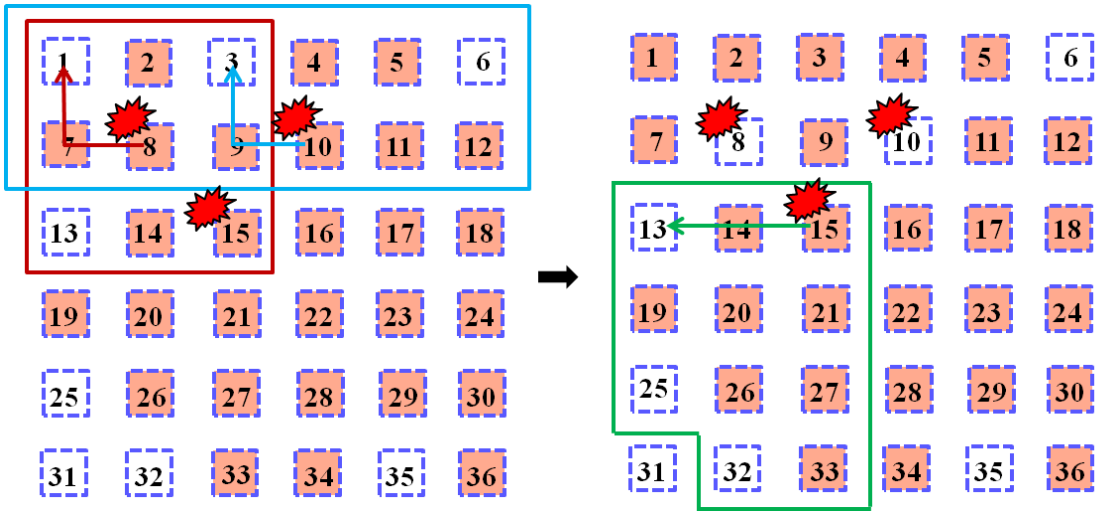


Figure 12. The reconfiguration algorithm consider the demand issue.

1.5 Contribution

In this thesis, we propose a configuration-level algorithm that reconfigures faults through replacement with spares followed by routing adjustment and yields higher reconfiguration success rate compared to the previous work. In addition, we also propose a generic fault tolerant architecture for 3D FPGAs that distributes spares evenly across the 3D FPGA in order to provide a reconfiguration friendly architecture to improve the success rate.

1.6 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, timing model, fault model, definitions and problem formulation are presented first. Then we propose our reconfiguration algorithm in Chapter 3. In Chapter 4, two fault tolerant architectures are proposed. Experimental results are presented in Chapter 5 and some contributions are concluded in Chapter 6.

Chapter 2
Preliminaries

In this chapter we first introduce the tool which used in the thesis and then describes two models used to timing analysis and fault location assumption. Finally, we describe the definitions and problem formulation.

2.1 3D P&R Tool

Three dimensional place and route (TPR) [14][26] is the first complete CAD flow in academia from layering process to routing process for 3D FPGAs. The main flow of them is shown in Figure 13.

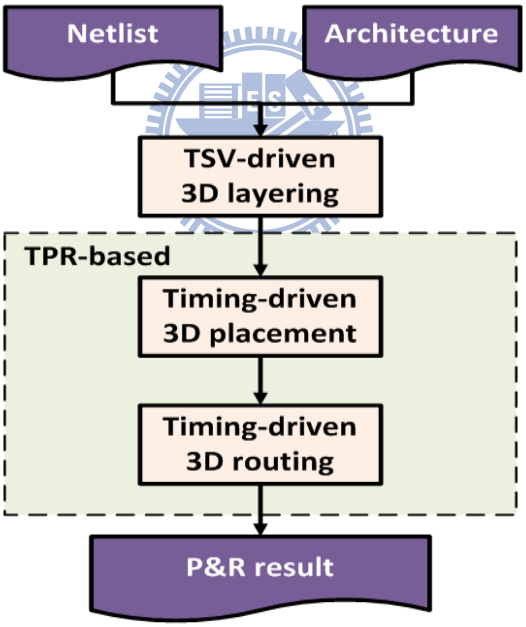


Figure 13. A 3D FPGA CAD flow.

The flow starts with a technology-mapped netlist in .blif format, which describes the circuit. To map a circuit into FPGA, T-VPack [14] converts the .blif netlist into a .net netlist of FPGA logic blocks. Then, the .net netlist and the architecture description file are input to the placement algorithm. At first, the placement algorithm partitions the circuit into *n* balanced partitions, where *n* equals to the number of layers

in a 3D FPGA design. Second, all layers are placed by an SA-based placement algorithm; CLBs are selected and swapped or moved randomly during the placement until maximum number of iterations is reached. Finally, global and detailed routing is performed using the adapted 3D version of the TPR routing algorithm.

2.2 Timing Model

In a "tile-based" FPGA, the FPGA structure is homogeneous, i.e. every location (x, y, z) in the FPGA is constructed from identical tiles. Exploiting such architectures, a delay lookup matrix indexed by (Δx, Δy, Δz) is constructed. Each (Δx, Δy, Δz) entry in the matrix is computed by TPR's timing-driven router, that performs a routing between the two blocks and the delay is recorded in the delay lookup matrix at location (Δx, Δy, Δz), so the matrix performs as a function that return estimated delay between two blocks given the delta location (Δx, Δy, Δz) of them.

The circuit is represented as a directed acyclic graph. Nodes represent the input and output pins of circuit elements such as registers and LUTs. Connections between nodes are modeled with edges in the graph which are annotated with the delay required to pass through the circuit element or routing. To determine the delay of the circuit, a breadth-first traversal is applied to the timing graph. Each node with incident edges is labeled with its arrival time as shown in Equation (1):

$$T_{arrival}(i) = Max_{\forall j \in fanin(i)}\{T_{arrival}(j) + delay(j,i)\} \quad (1)$$

Node i is the node currently being computed, and delay (j, i) is the delay value marked on the edge. To compute the slack, we perform a second breadth-first traversal of the timing graph for required time $T_{required}$. $T_{required}$ at all sinks is set to the maximum arrival time and then propagated backwards starting from the sinks with the following Equation (2):

$$T_{required}(i) = Min_{\forall j \in fanout(i)}\{T_{required}(j) + delay(i,j)\} \quad (2)$$

Finally, the slack of a connection (j, i) as shown in Equation (3):

$$Slack(i, j) = T_{required}(j) - T_{arrival}(i) - delay(i, j) \quad (3)$$

2.3 Fault Model

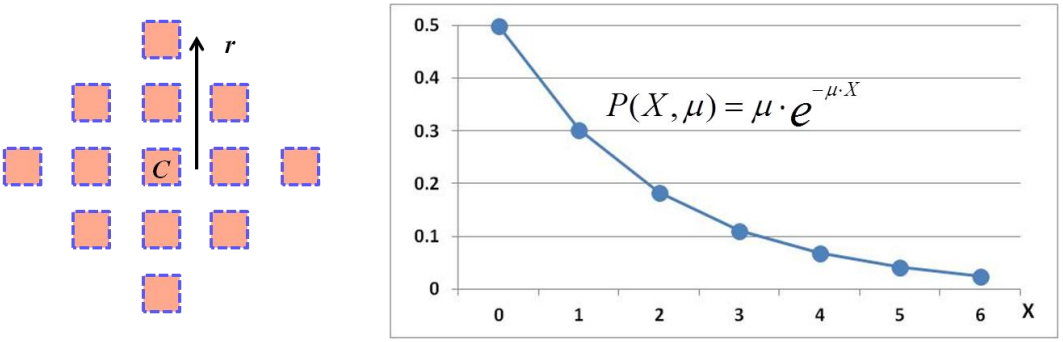
In this paper we only use a CLB-level fault model, which assumes that any fault in a CLB then the CLB is disabled; faults in other part are not considered. In our experiments, we use two different fault models described below.

i) *Uniform fault model* – Faults uniformly distribute across the FPGA. In the other words, the probability of a CLB being fault is independent of the state of the neighboring CLBs. The model is implemented by randomly assuming a CLB of coordinate (x, y, z) to be faulty.

ii) *Clustered fault model* – Faults distribute in clusters. In the other words, if a CLB is faulty then its neighboring CLBs have a higher probability of being faulty, as shown in Figure 14. This model is implemented by randomly assuming a CLB of coordinate (x, y, z) to be the center C of a fault cluster of radius r. On the layer z, the CLBs within distance r from the center are faulty with an exponentially decreasing probability function as shown in Equation (4):

$$P(X, \mu) = \mu e^{-\mu X} \quad (4)$$

where μ is failure rate and X is a positive value range between 1 and r.



(a) (b)

Figure 14. (a) A clustered fault of r = 2. (b) The exponential probability function.

2.4 Definitions

In this section we define some sets and function that will be used in the following chapters. We define a set called *B*, which includes all CLBs on the FPGA is shown in Equation (5) and B_F is a set of all faulty and mapped CLBs, as shown in Equation (6). Similarly, B_S is a set of all non-faulty spare CLBs (as shown in Equation (7)). b_{FT} is a faulty and mapped CLB, which is the most critical one and $d_M(b_i, b_j)$ is a function which returns the Manhattan distance between two CLBs, as shown in Figure 15.

$$B = \{b \mid b \text{ is a CLB}\} \tag{5}$$

$$B_F = \{b_F \mid b_F \text{ is a faulty and mapped CLB}\},\ B_F \subseteq B \tag{6}$$

$$B_S = \{b_S \mid b_S \text{ is a non-faulty spare CLB}\},\ B_S \subseteq B \tag{7}$$

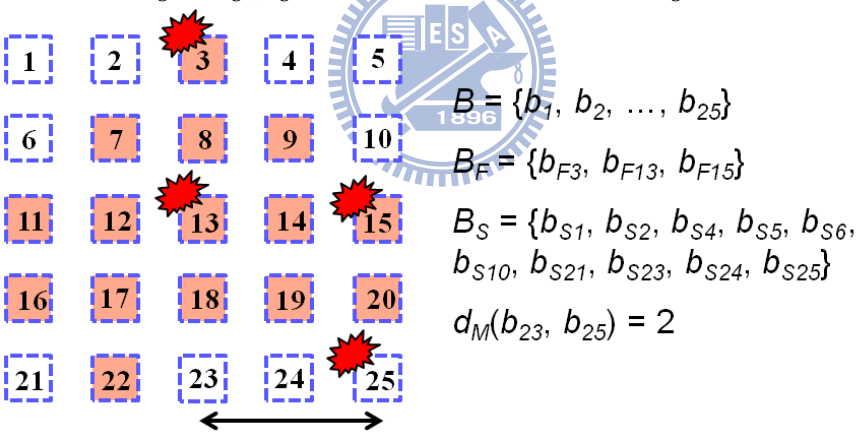


Figure 15. The definitions of CLB array.

2.5 Problem Formulation

Given a netlist of CLBs, architecture specification, existing placement and routing result and locations of faulty CLBs, our objective is to partially reconfigure the design avoiding the faulty CLBs and the circuit delay meets the timing requirement.

Chapter 3

Proposed Demand-Aware Reconfiguration Algorithm

As mentioned in Section 1.4, we propose a demand-aware reconfiguration algorithm considering the demand issue during fault reconfiguration. According to the cost function of ripple move reconfiguration algorithm (as shown in Equation (8)), we modify the cost function to the demand-aware version consisting of the delay and the demand cost as shown in Equation (9).

$$Cost_{path} = Cost_{delay} \tag{8}$$

$$Cost_{path} = \alpha \times Cost_{delay} + \beta \times Cost_{demand} \tag{9}$$

For quickly estimating the delay between two blocks, an approximation can be computed by using Manhattan distance. The estimated delay is recorded in the delay lookup matrix, so when we construct a DAG for a fault CLB, edges are weighted with the delay required upon moving the block from existing location to the adjacent CLB by looking up the delay matrix and $Cost_{dealy}$ is calculated by finding the shortest path (least delay cost) on the DAG.

The objective of the demand cost $Cost_{demand}$ is to achieve a solution with more aspect of consideration. α and β are the adjustable parameters. In demand-unaware method, α is set to 1 and β is set to 0. In proposed demand-aware method, α is set to 0.2 and β is set to 0.8.

In the rest of this chapter, the concept of our algorithm is described in Section 3.1.1. Section 3.1.2 explains how to gradually reconfigure the circuit placement by our iterative reconfiguration algorithm and in the last section, the routing algorithm and re-routing algorithm are described.

3.1 Reconfiguration Algorithm

3.1.1 The Concept of Our Algorithm

In the ripple move reconfiguration method, a DAG is constructed for each faulty block. The weight of an edge e=<u, v> is set to the difference between the required delay of the block residing in the CLB u and in the adjacent CLB v; then the shortest path (least cost) between the source CLB and destination CLB is found. According to the shortest path, the faulty block is reconfigured by ripple moving CLBs to their adjacent ones along this path. However, the result of this greedy method may be too local because for each faults. It constructs a DAG only from the faulty block point of view; and non-faulty spare CLBs are greedily chosen as destinations in order of the locally delay cost is minimized in each iteration. In fact, the demand for non-faulty spare CLB of every faulty block may be different, thus these two factors have to be considered when constructing DAG. Therefore, our algorithm is based on such a concept to resolve this problem.

3.1.2 Demand-Aware Reconfiguration Algorithm

For ease of explanation, the following is an example to detail how we use this concept of demand. At the begin of each iteration, we determine a range with the distance d_{SER} by expanding the from b_{FT}, the target faulty and mapped CLB which has the most critical, until this range contains at least K (K=3 in this example) spare CLBs. Notice that it is a three-dimensional range, as shown in Figure 16 and the definition is shown in Equation (10).

$$B_S' = \{b_S \mid d_M(b_{FT}, b_S) \le d_{SER}\} \text{ and } |B_S'| \ge k,\ B_S' \subseteq B_S \qquad (10)$$

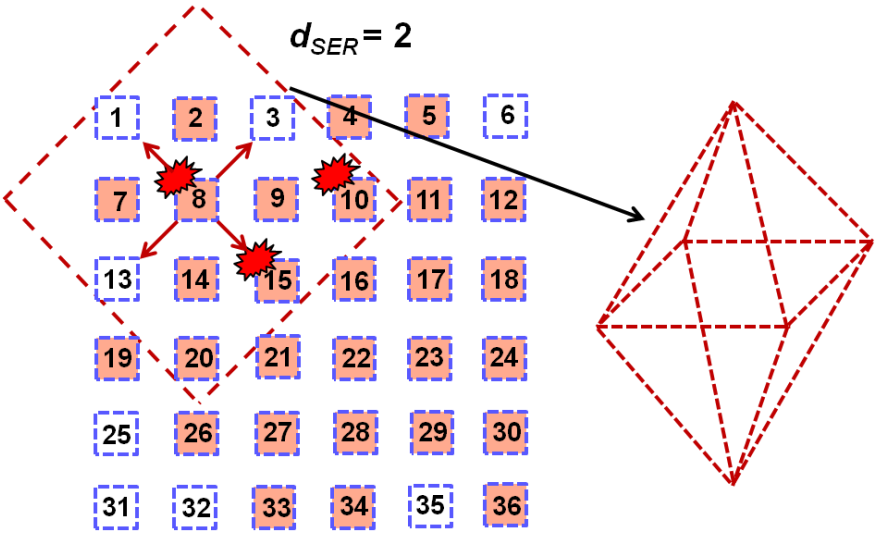


Figure 16. Find the searching distance.

Next, we find the neighbor faulty and mapped CLBs of each b_S. Consider b_{S3} at first, we utilize the d_{SER} to find b_{FN}, as shown in Figure 17. We define a B_{FN} as a set of faulty and mapped CLBs in the neighborhood of b_S, as shown in Equation (11).

$$B_{FN}(b_S) = \{b_{FN} \mid d_M(b_S, b_{FN}) \le d_{SER}\}, B_{FN} \subseteq B_F \quad (11)$$

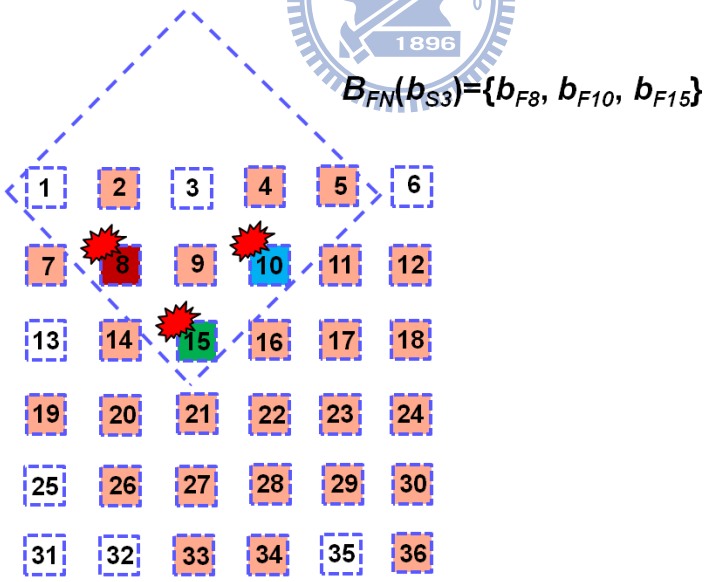
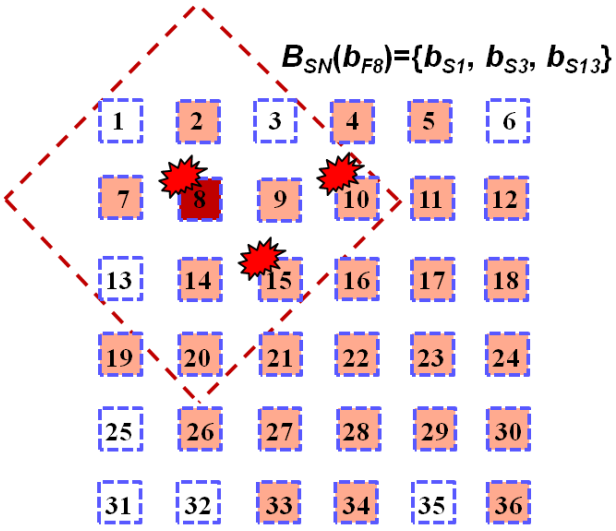


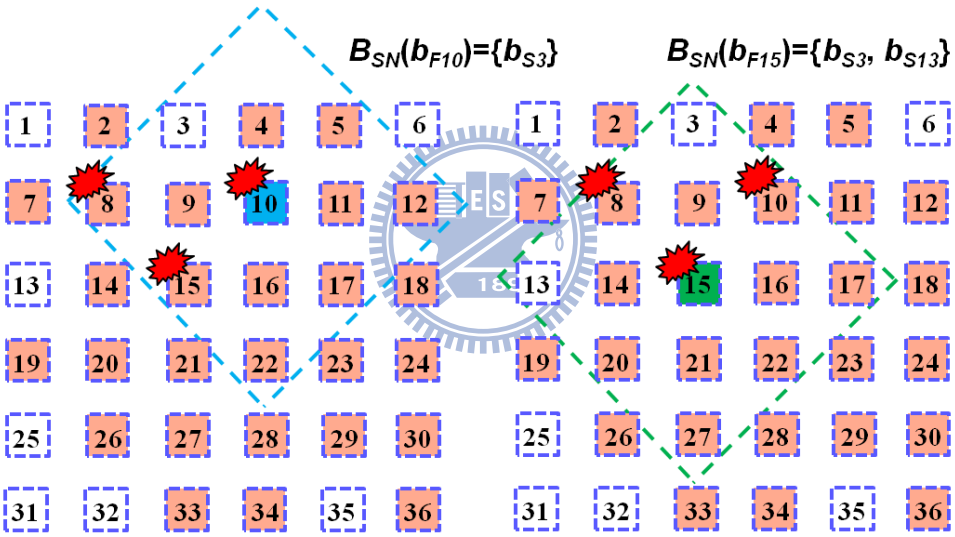
Figure 17. Neighbor faulty and mapped CLBs of b_{S3}.

And we utilize the d_{SER} to find the neighbor non-faulty spare CLBs B_{SN} of these faulty and mapped CLBs, as shown in Figure 18. B_{SN} is defined as spare CLBs residing in the neighborhood of b_F with the distance smaller or equal to d_{SER}, as shown in Equation (12).

$$B_{SN}(b_F) = \{b_{SN} \mid d_M(b_F, b_{SN}) \le d_{SER}\}, B_{SN} \subseteq B_S \quad (12)$$



(a)



(b) (c)

Figure 18. Neighbor spare CLBs of (a) b_{F8}, (b) b_{F10}, and (c) b_{F15}.

Here we introduce Equation (13), which represents the demands for b_S of b_F and is in inverse proportion to the number of spare CLBs in the neighborhood and the Manhattan distance between them. If there is a spare CLB very close to a b_F and only few spare CLBs is in the b_F's neighborhood, then the demands for the spare CLB of the b_F is very high. The maximum demand $D_{md\text{-}max}$ for a spare CLB is defined as Equation (14).

$$D_{md}(b_F, b_S) = \frac{1}{d_M(b_F, b_S)} + \frac{1}{|B_{SN}(b_F)|}$$

(13)

$$D_{md-\max}(b_S) = \underset{b_F \in B_{NF}(b_S)}{Max}\{D_{md}(b_F, b_S)\} \quad (14)$$

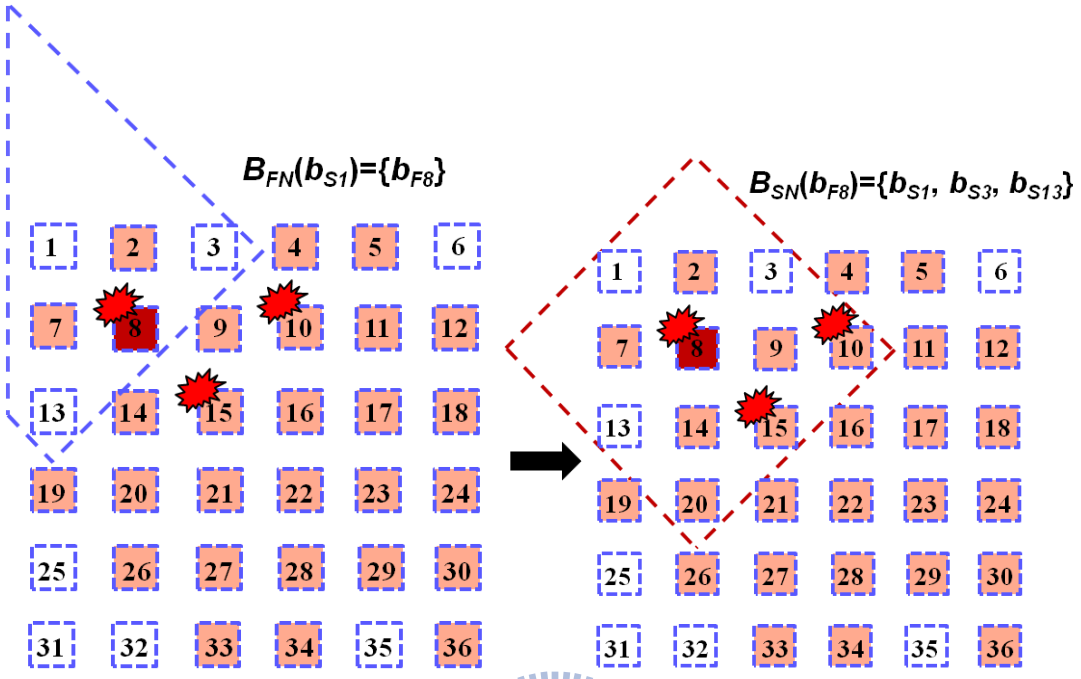


Figure 19. The reconfiguration procedure when consider b_{S1}.

According to the definition, the demands for b_{S3} of b_{F10} and b_{F15} are:

$$D_{md}(b_{F8}, b_{S3}) = \frac{1}{d_M(b_8, b_3)} + \frac{1}{|B_{SN}(b_{F8})|} = \frac{1}{2} + \frac{1}{3} = 0.83$$

$$D_{md}(b_{F10}, b_{S3}) = \frac{1}{d_M(b_{10}, b_3)} + \frac{1}{|B_{SN}(b_{F10})|} = \frac{1}{2} + \frac{1}{1} = 1.5$$

$$D_{md}(b_{F15}, b_{S3}) = \frac{1}{d_M(b_{15}, b_3)} + \frac{1}{|B_{SN}(b_{F15})|} = \frac{1}{2} + \frac{1}{2} = 1$$

For the reconfiguration, the demand cost of b_{FT} for b_S is defined in the Equation (15). The small demand cost means the demand of b_{FT} for b_S is high.

$$C_D(b_{FT}, b_S) = 1 - \frac{D_{md}(b_{FT}, b_S)}{D_{md-\max}(b_S)} \quad (15)$$

So in the example, the demand cost for b_{S3} of b_{F8} is:

$$C_D(b_{F8}, b_{S3}) = 1 - \frac{D_{md}(b_{F8}, b_{S3})}{D_{md-\max}(b_{S3})} = \frac{0.83}{1.5} = 0.55$$

Similarly, for b_{S1}, the procedure is shown in Figure 19. Notice that only b_{F8} in the neighborhood of b_{S1}, so the demand cost is 0.

$$D_{md}(b_{F8}, b_{S1}) = \frac{1}{d_M(b_8, b_1)} + \frac{1}{|B_{SN}(b_{F8})|} = \frac{1}{2} + \frac{1}{3} = 0.83$$

$$C_D(b_{F8}, b_{S3}) = 1 - \frac{D_{md}(b_{F8}, b_{S1})}{D_{md-\max}(b_{S1})} = 1 - \frac{0.83}{0.83} = 0$$

For b_{S13} at the last, the procedure is shown in Figure 20.

$$D_{md}(b_{F8}, b_{S13}) = \frac{1}{d_M(b_8, b1_3)} + \frac{1}{|B_{SN}(b_{F8})|} = \frac{1}{2} + \frac{1}{3} = 0.83$$

$$D_{md}(b_{F15}, b_{S13}) = \frac{1}{d_M(b_{15}, b_3)} + \frac{1}{|B_{SN}(b_{F15})|} = \frac{1}{2} + \frac{1}{2} = 1$$

$$C_D(b_{F8}, b_{S3}) = 1 - \frac{D_{md}(b_{F8}, b_{S13})}{D_{md-\max}(b_{S13})} = 1 - \frac{0.83}{1} = 0.17$$

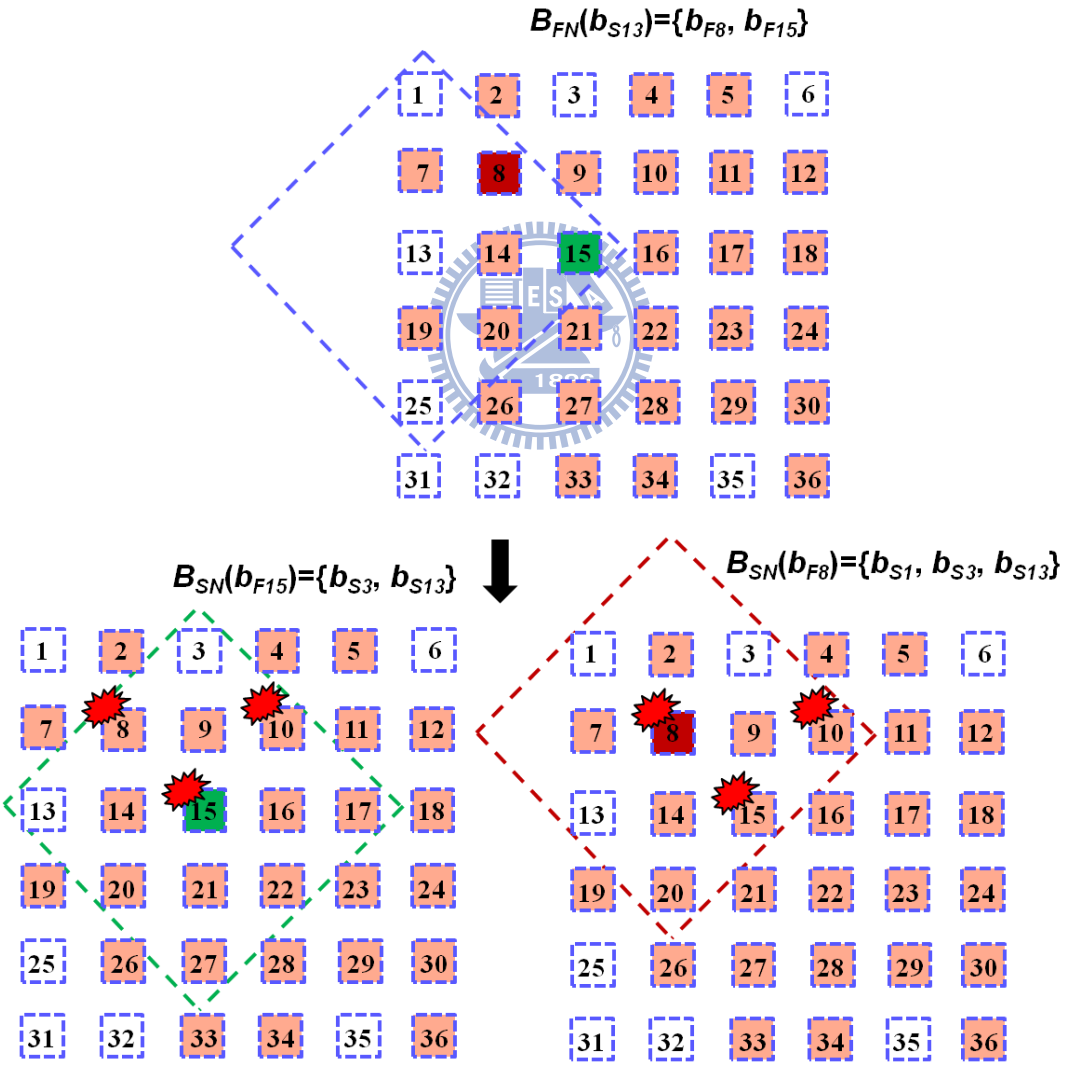


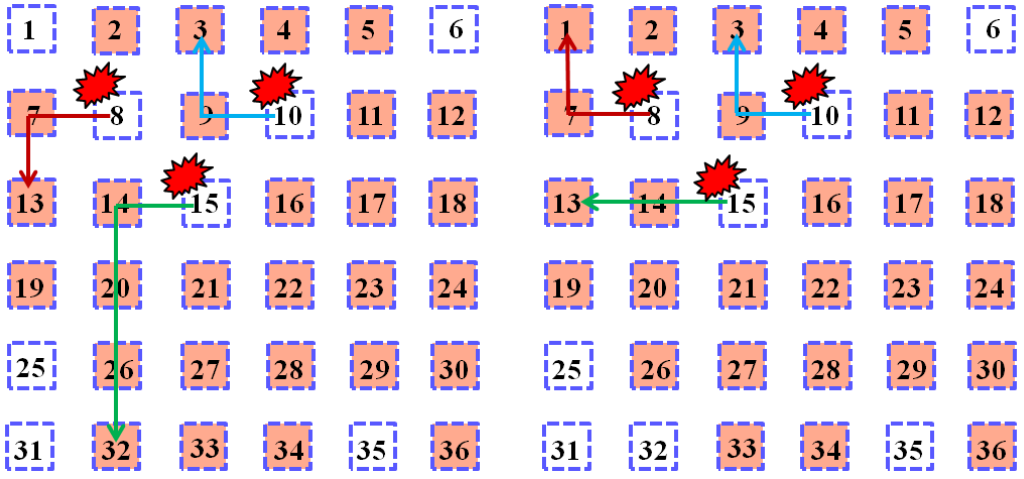
Figure 20. The reconfiguration procedure when consider b_{S13}..

In the example, we assume the delay cost equals to the Manhattan distance between two CLBs, so the delay cost of all shortest path in this example equal to 2. The costs of each candidate destinations in the first iteration are shown in Table 2, where α is 0.2 and β is 0.8.

Table 2. The cost of first iteration in the example.

Destination	Cost		
	Delay	Demand	Total
b_{S1}	α x 2	β x 0	0.4
b_{S3}	α x 2	β x 0.45	0.76
b_{S13}	α x 2	β x 0.17	0.536

Finally, we take b_{S1} as the destination and the b_{F8} is reconfigured by moving it to adjacent CLBs along the shortest path $\{b_{F8}, b_7, b_{S1}\}$. This procedure is iteratively performed for each faulty block until all faults are successfully reconfigured or no path is found resulting in reconfiguration failure. The final result of ripple move reconfiguration algorithm is shown in Figure 21-(a). This method randomly chooses one of $\{b_{S1}, b_{S3}, b_{S13}\}$ as destination because it does not consider the demand cost. Another result is shown in Figure 21-(b), it is demonstrated that our demand-aware reconfiguration algorithm can find the better solution compared to the ripple move reconfiguration algorithm.



(a) (b)

Figure 21. The result of (a) ripple move algorithm and (b) our algorithm.

The algorithm flow is shown in Figure 22; after constructing a DAG first, the reconfiguration path is determined by finding the shortest path between the source and one of the *k* destinations; finally, the fault is reconfigured by moving blocks to adjacent CLBs along this path. The *reconfiguration iteration* is performed until every block is located on non-faulty CLBs.

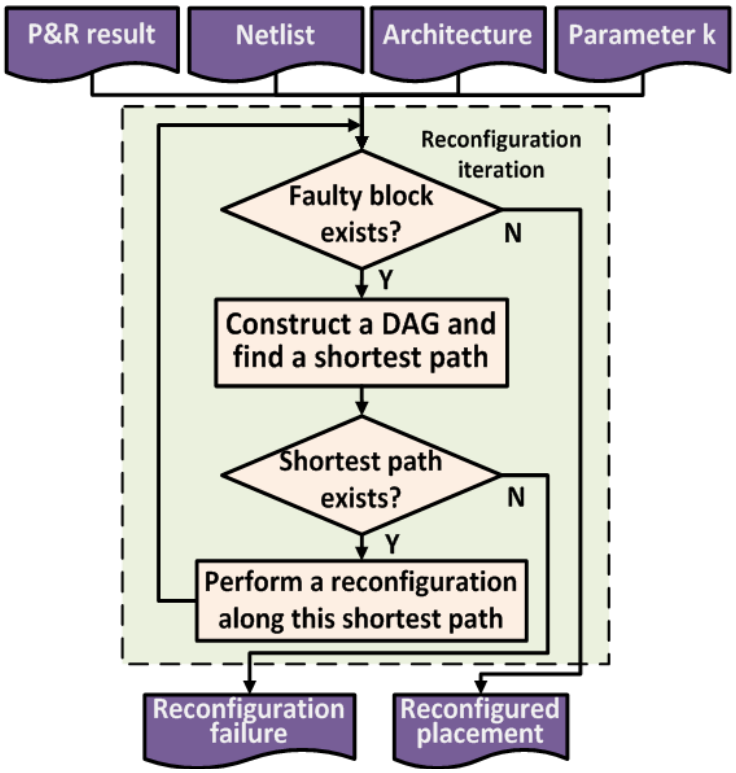


Figure 22. Our algorithm flow.

3.2 Re-routing Algorithm

3.2.1 Concept of Routing Algorithm

After placement, the locations of all CLBs have been determined, and then a timing driven router connects all connections between CLBs. In routing stage, the FPGA architecture is represented as a routing resource graph. It represents wire segments, TSVs and input or output pins of logic blocks, as shown in Figure 23.

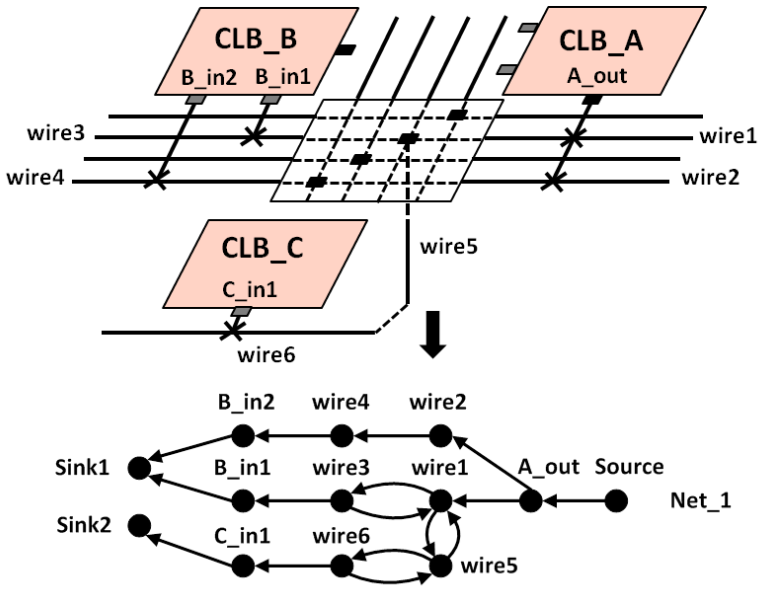


Figure 23. FPGA routing architecture and routing resource graph.

The routing algorithm in TPR is based on Pathfinder negotiated congestion algorithm [26]. It iteratively rips-up and re-routes every net until the result meets the congestion constraint. At the first iteration, all nets are routed for minimizing delay without congestion constraint; that is, the routing resources are allowed overuse. When overuse exists at end of a routing iteration, the cost of overusing a routing resource is increased, so congestion will be resolved at another routing iteration. This process is repeated until all routing resources only are used once.

3.2.2 Re-routing Algorithm

During fault-tolerant reconfiguration, the blocks on the shortest path are moved for one grid in each iteration and a set of blocks are moved due to the 10% faulty CLBs generally. When the block is moved, its connections are also affected, thus we have to re-route these connections, as shown in Figure 24.

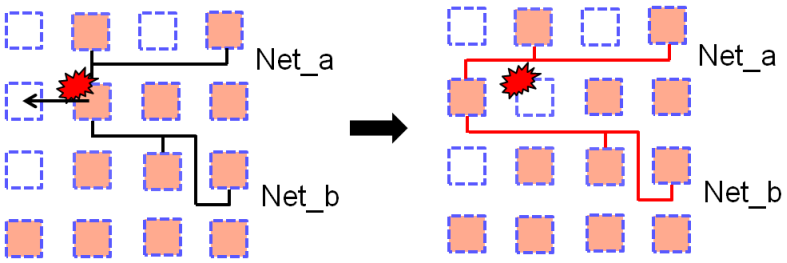


Figure 24. The concept of re-route.

We record the blocks which are moved during the placement stage of fault-tolerant replacement stage. When in the routing stage, we rip-up all the affected nets and fix the exist routing, and then re-route them. Figure 25 shows an example that endpoints of Net_1 connect to CLB_A, CLB_B and CLB_C, respectively. If the block originally residing in CLB_A is moved to a new CLB, we rip-up Net_1 and the routing of Net_1 is started from the output pin of the new CLB and terminated at the original sink1 and the original sink2. If the block residing in CLB_B or CLB_C is moved to a new CLB, we rip-up Net_1 and the new routing of Net_1 is started from the original source and terminated at the input pin of the new CLB.

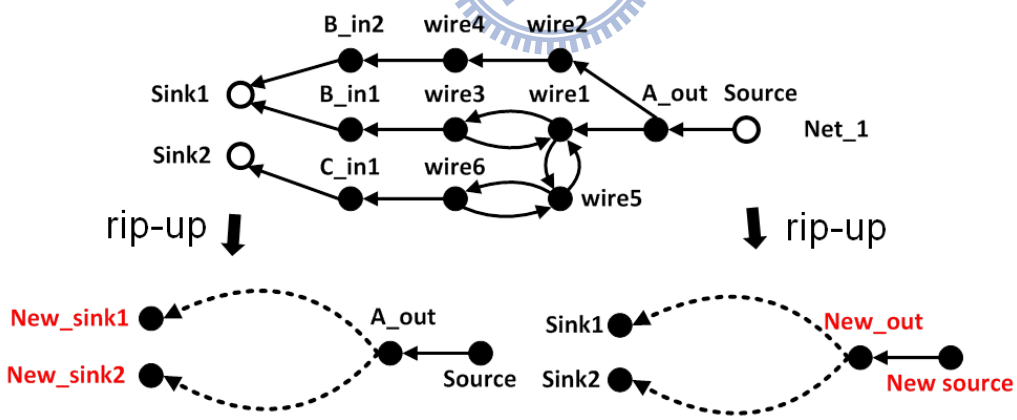
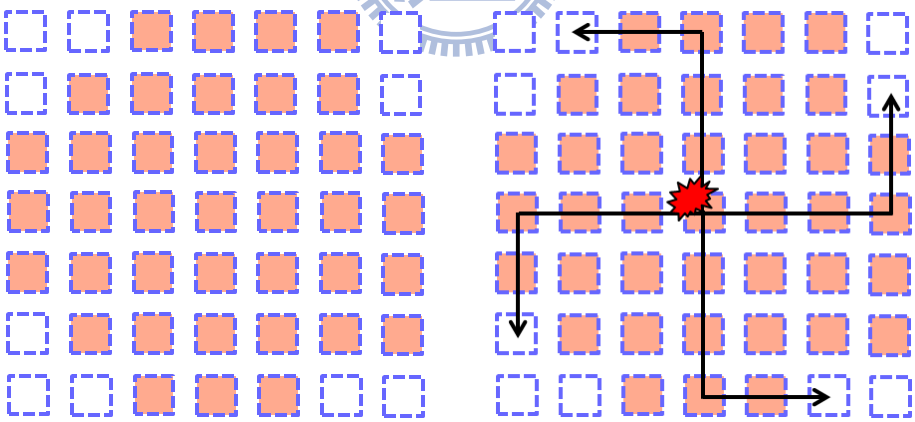


Figure 25. Rip-up and re-route the affected net.

Chapter 4
Fault Tolerant Architecture

4.1 Non-Reserved (NR)

Locations of blocks are determined using an SA-based placement algorithm with the objective of minimizing wirelength and circuit delay. Thus, spare CLBs are pushed to the edge of FPGA, such a distribution of spare CLBs is called *non-reserved* (NR) architecture, as shown in Figure 26-(a). As the result, this placement is not suitable to fault reconfiguration through replacement with spare CLB because most spares located along the edge, which may cause a large amount of CLBs moved by ripple-move fault reconfiguration, as shown in Figure 26-(b). Therefore, even if we have a better reconfiguration algorithm, results will be limited because the restrictions of architecture.



(a) (b)

Figure 26. (a) The timing-driven placement. (b) The drawback of timing-driven placement for fault tolerance.

4.2 Evenly-Distributed (ED)

As mentioned above, traditional architecture is not suitable to fault tolerance, which inspires us to discover new architectures that take fault tolerance into consideration. We address this problem by evenly distributing spare CLBs across the FPGA and force them to pre-allocate spare resources before the SA-based placement algorithm. These pre-allocated spare CLBs are not allowed being used during SA-based placement, so we can get a placement result with spares evenly distributed in the 3D FPGA design. Such a distribution of spare CLBs is called *even-distributed* (ED) architecture. When faults occur, spares are very likely close to the faulty CLBs and benefit replacement without severely timing degradation.

We propose five optional ED architecture ED3, ED4, ED5, ED6 and ED7. *ED#* represents a spare pattern that the postfix # specifies the maximum distance between two adjacent spare CLBs in either X or Y or XY direction, as shown in Figure 27. The estimated percentage of reserved spare CLBs of each ED architecture is shown in Table 3.

Table 3. The estimated percentage of reserved spare CLBs.

	Spare pattern				
	ED7	ED6	ED5	ED4	ED3
%Reserved	2	2.7	4	6.25	11

It should be noticed that the CLB utilization of most FPGA is only 70–80% in order to enhance the routability. As we use spare CLBs, the total number of signal nets does not increase. Thus, routing complexity does not significantly increase, however, a price to be paid for using the fault tolerant architecture is an additional delay increasing because we change the original timing driven placement, detail

discussions are concluded in Chapter 5.

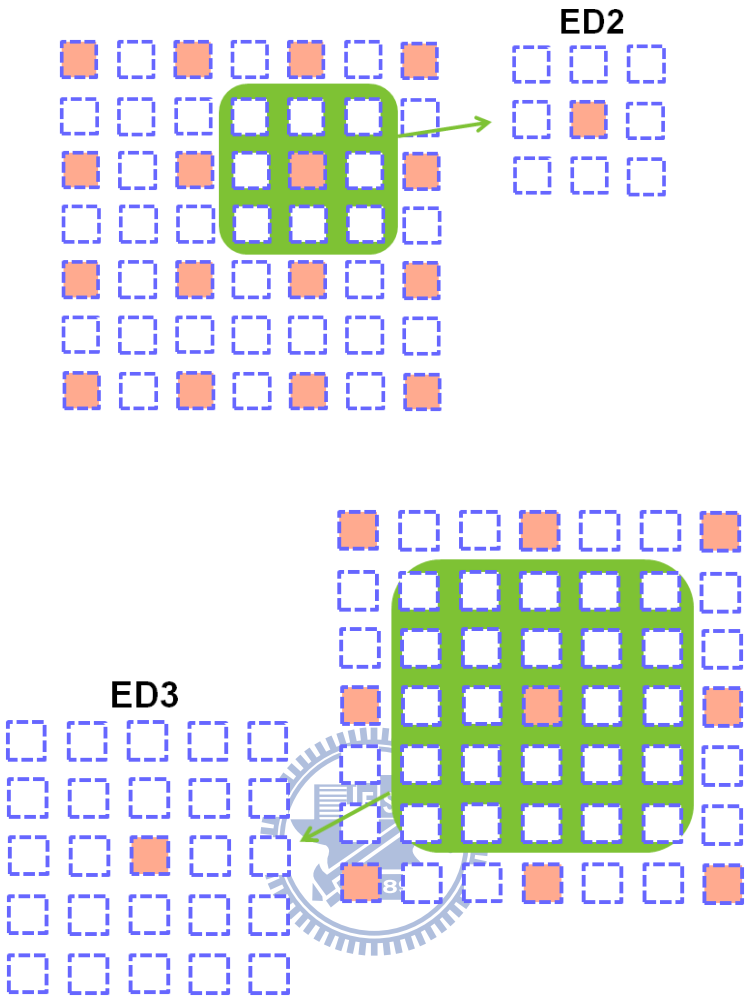


Figure 27. Evenly-distributed architecture.

Chapter 5
Experimental Results

5.1 Experimental Environment

The architectural setting in our experiments are shown in Table 4. The settings of CLBs and channel width are based on Altera Stratix IV [27], Xilinx FPGAs [28] and related work [29]. There are 4 wire segments with different lengths in these 32 wires, L1, L2, L4 and L8. The length of a wire segment is the number of CLBs it spans. There are 12 L1/L2 and 4 L4/L8 wires. In Z direction, each TSV spans one layer only for routability.

Table 4. The architecture setting.

Architectural Settings			Value
LB (Altera Stratix IV)	**# of inputs of an LB (*I*)**		8
	# of LUTs in an LB (*N*)		2
	# of inputs of a LUT (*J*)		6
Channel width (Xilinx)	**Both W_X and W_Y**		32
# of wire segments	**X-Y directions**	**(*L1*, *L2*, *L4*, *L8*)**	(12, 12, 4, 4)
	Z direction	**L1 only (Routability-driven)**	16
I/Os	**Location**		Bottom-most layer
TSV	**Pitch**		6um
Process technology node			45nm

Table 5 shows the 16 test cases in our benchmark set – 15 are from MCNC [30] and 1 is from IWLS2005 [31], which are sorted by number of CLBs. Each test case perform 25 experimental runs with different random seeds (5 fault seeds and 5 placement seeds) and find the average as the result. In addition, the number of layers (nz) is set to 4. The CLB utilization is set to 70% and the fault rate is set to 10%.

Table 5. The benchmark circuits.

Design	#CLBs	#Nets	#I/Os
tseng	524	885	174
ex5p	532	937	71
apex4	631	1086	28
dsip	685	1374	426
misex3	699	1159	28
alu4	761	1257	22
seq	875	1458	76
apex2	939	1572	41
s298	966	1361	10
frisc	1778	2823	136
elliptic	1802	3039	245
spla	1845	2977	62
pdc	2288	3671	56
ex1010	2299	3932	20
clma	4192	6871	144
usb_funct_0mv_b	7440	11374	234

5.2 Results and Analysis

5.2.1 Experimental Flow

In our experiment, three types of configuration-level repair methods are implemented: i) resynthesis ii) Cong's reconfiguration algorithm and iii) our reconfiguration algorithm. Figure 28 shows the experimental flow of resynthesis, the faulty CLB are marked before layer assignment and regarding them unable to be mapped. Figure 29 shows the experimental flow of two reconfiguration algorithms. Taking the initial placement and routing as an existing result, faults are repaired by partially reconfiguring blocks avoiding faulty CLB.

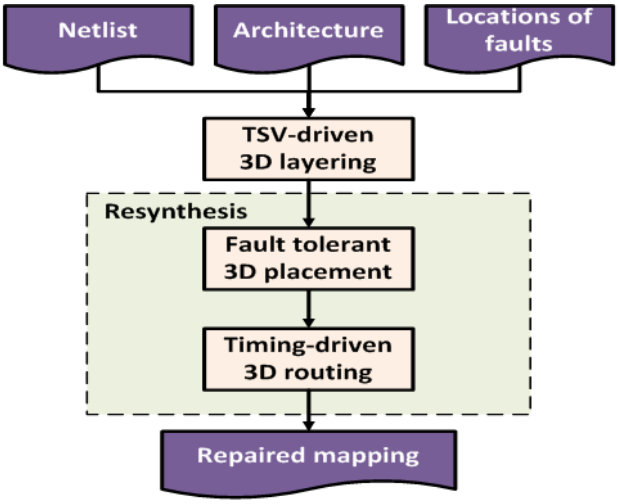


Figure 28. The experimental flow of resynthesis.

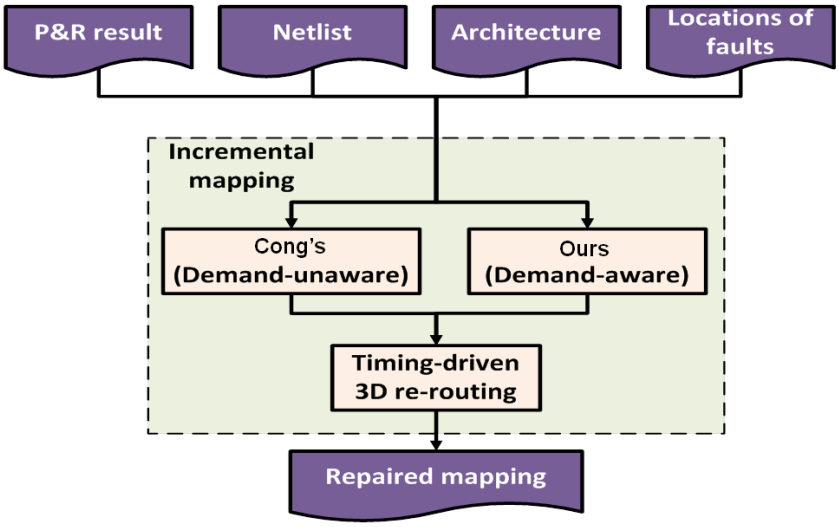


Figure 29. The experimental flow of reconfiguration.

5.2.2 Analysis of Timing Penalty

Following are two reasons cause of timing degradation:

i) Initial architecture – there are six architectures used in our experiment, NR, ED3, ED4, ED5, ED6, ED7 with different percentages of reserved spare CLBs for each pattern, i.e., different spare densities; the higher spare density results in more blocks spread to the edge of the FPGA and thus the more delay increases. Figure 30 shows the delay increase of each architecture compared to NR. ED7 has the minimal

impact to timing because it has the minimal spare density, otherwise, ED3 has the maximal timing overhead. For ease of exposition, we refer to the result of the NR architecture as IA-NR.

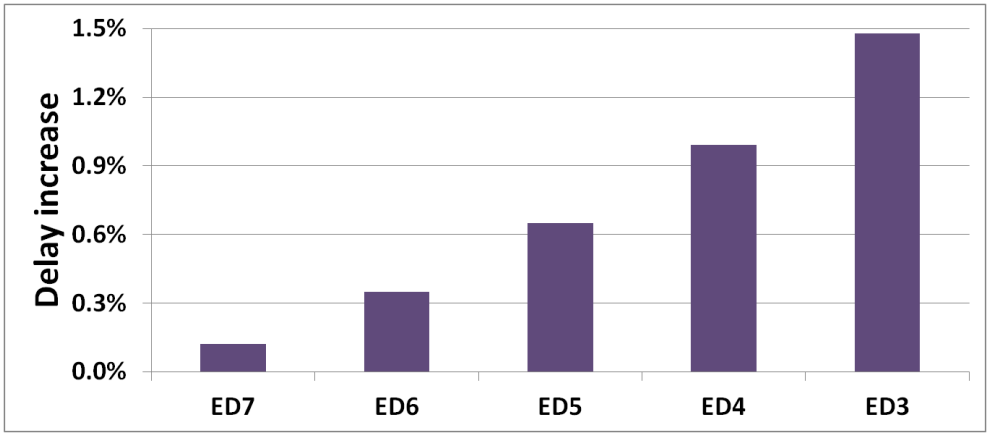
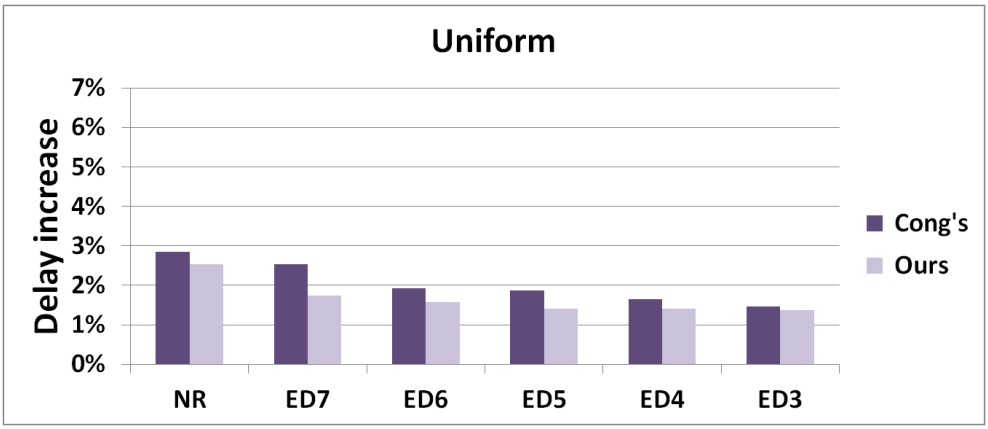


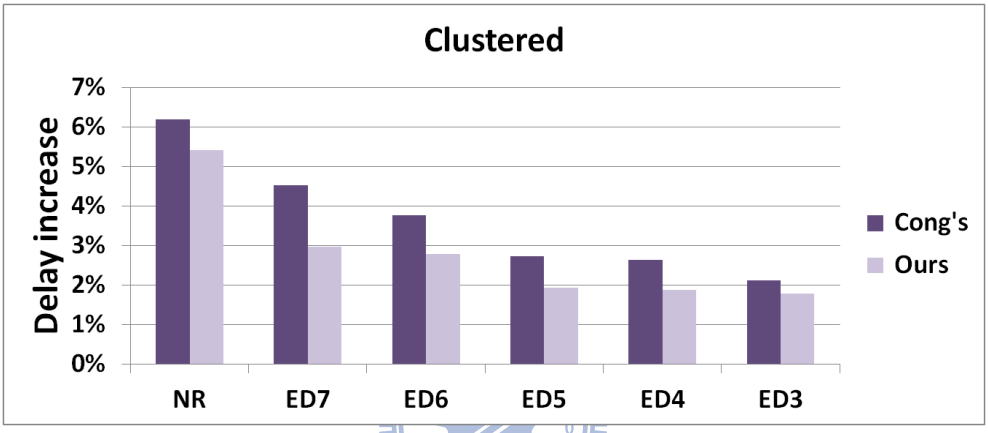
Figure 30. Timing penalty caused by fault tolerant architecture.

ii) Reconfiguration – the delay is increased as the circuit placement being reconfigured. Because the ED architecture provides a fault tolerant friendly architecture. The higher spare density is, the more spare CLBs close to faulty blocks, which causes the timing degradation is lower during reconfiguration. Figure 31-(a) shows the delay increase caused by reconfiguration for uniform fault model based on their IA results. The delay overhead is gradually reduced as spare density grows, and the increased delay of our method is always lower than Cong's.

Similarly, Figure 31-(b) illustrates the delay increase for clustered fault model. The delay increases is significantly higher compared to uniform fault model because of a number of faults being localized within a region. it is represents clustered fault distribution is more difficult to be reconfigured.



(a)

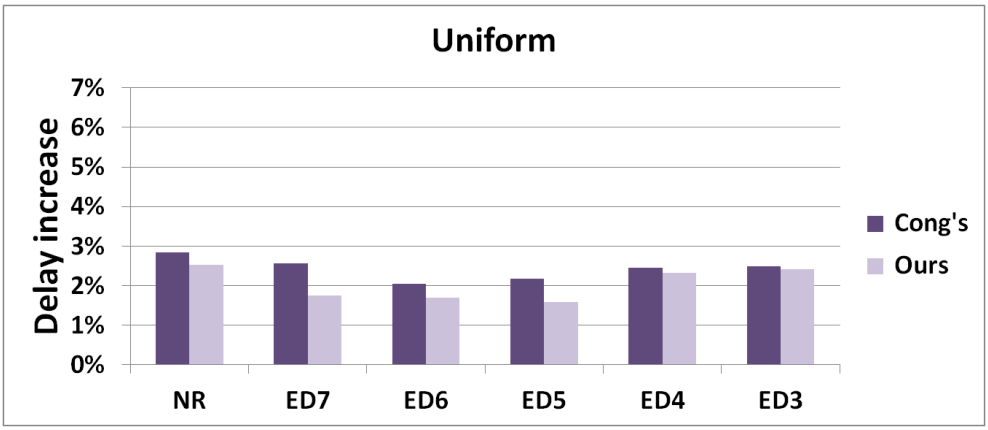


(b)

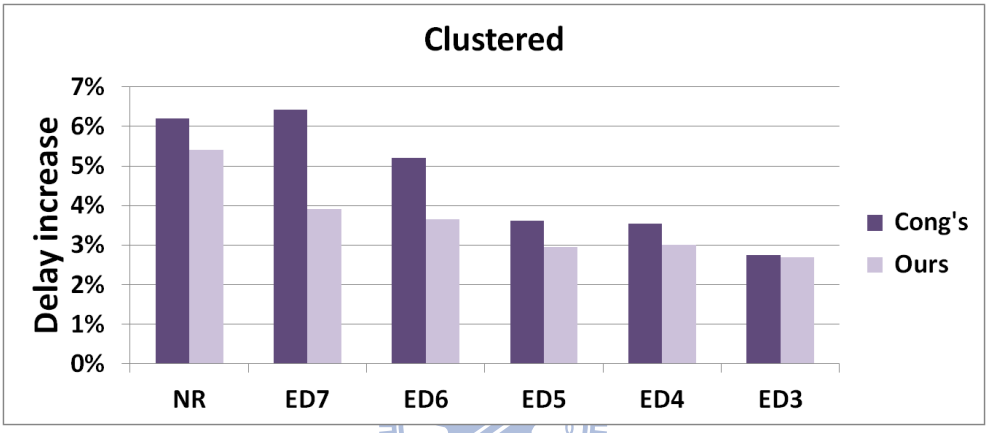
Figure 30. Timing penalty caused by reconfiguration.

Figure 32-(a) shows the delay increase caused by reconfiguration for uniform fault model with the IA-NR as the baseline. It is observed that our delay increases are lower than Cong's. The delay increase is gradually reduces at the beginning as the spare density grows; however, if we continue increase the spare density, the timing degradation caused by initial architecture will dominate the FPGA, so the delay increase is gradually increased.

The total delay increase caused by reconfiguration for clustered fault model are much higher compared to the pattern degradation, so the delay increase is decreased as grows spare density, and our delay increase are lower than Cong's, as shown in Figure 32-(b)



(a)



(b)

Figure 31. Combined effect on timing penalty.

5.2.3 Success Rate

A *successful result* is defined as a result with the all faults successfully reconfigured and the critical delay is within timing constraint; otherwise, the case is called *failure case*. Then the success rate is the percentage of the successful results. Resynthesis has the highest success rate as well as minimal timing degradation. We take the result as the baseline of reconfiguration. Therefore, we set the timing constraint to the delay that every case has 96% success rate in resynthesis flow. Figure 33 shows the results of success rate for uniform fault model. Our algorithm improves up to 13% success rate. If we relax 1% of the timing constraint, (i.e., 101% of the delay of the resynthesis flow with 96% success rate) the overall success rate is

increased by 5~10% and our algorithm has up to 9% improvement.

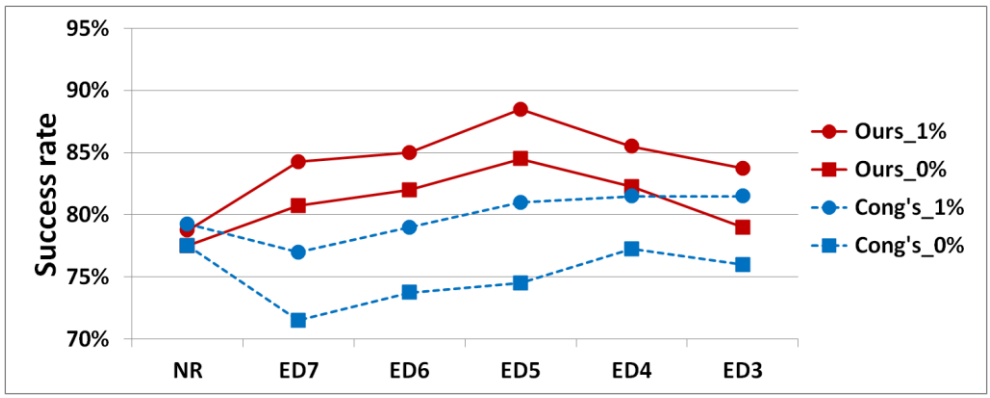
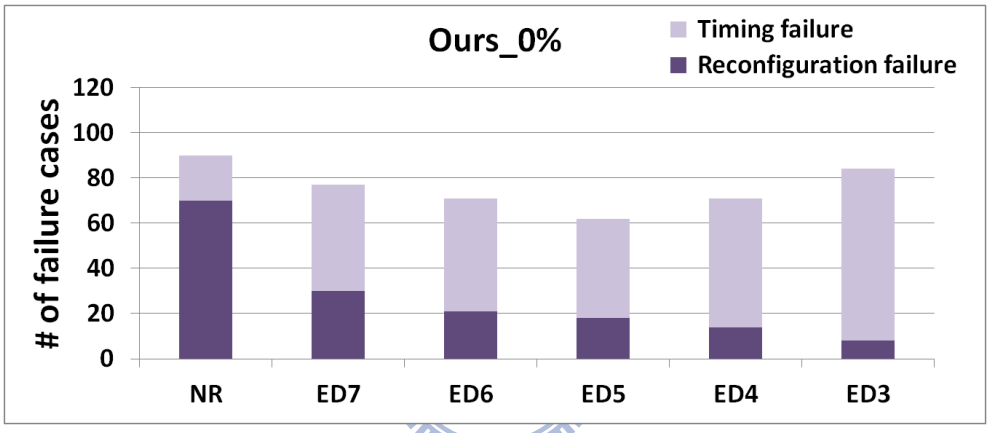
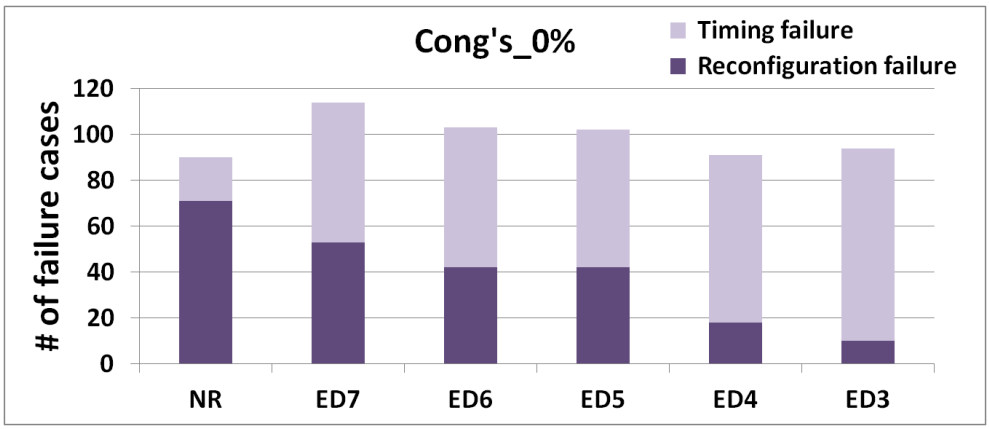


Figure 32. The success rate for uniform fault model.



(a)



(b)

Figure 33. The number of failure cases for uniform fault model.

Figure 34 shows the number of failure cases for uniform fault model, which is separated into i) reconfiguration failure – not all faults can find the corresponding reconfiguration paths; ii) timing failure – all faults can be reconfigured but the

resultant timing cannot meet the timing requirement. In the Figure 34-(a), it meets our expectation that the number of reconfiguration failure cases are decreased as the spare density grows. However, the initial architecture with high spare density dominates the timing degradation. It makes the number of timing failure cases more than one of the architectures with lower spare densities. Therefore, the total number of failure cases is increased. In the Figure 34-(b), it is also meets our expectation that the number of reconfiguration failure cases is decreased as the spare density grows. However, the number of timing failure cases is unstable because this algorithm just makes the locally optimal choice at each iteration. Take the example of NR and ED7, the low timing degradation can be obtained in the initial iterations for ED7 because the faulty blocks are close to spare CLBs; however, there are more results violating timing constraint in the last iterations of ED7 compared to NR.

Figure 35 shows the results of success rate for clustered fault model. Our algorithm improves up to 25% success rate. If we relax 1% of the target timing constraint, the overall success rate is increased 3~5% and our algorithm improves up to 25% success rate. The number of failure cases is far more than uniform fault model because concentrated faulty and mapped CLBs are difficult to be reconfigured, as shown in Figure 36. It is observed that the number of failure cases are decreased as the spare density grows; however, the results of two algorithm is not much difference in high spare density because the number reserved non-faulty spare CLBs is too much.

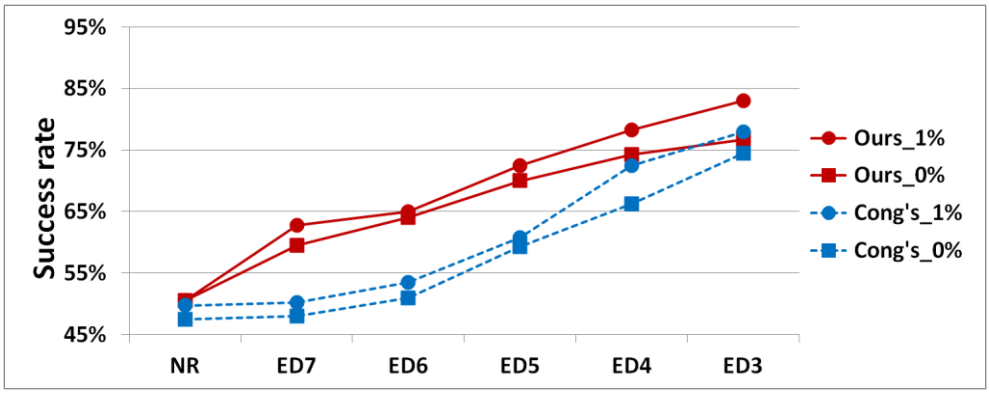
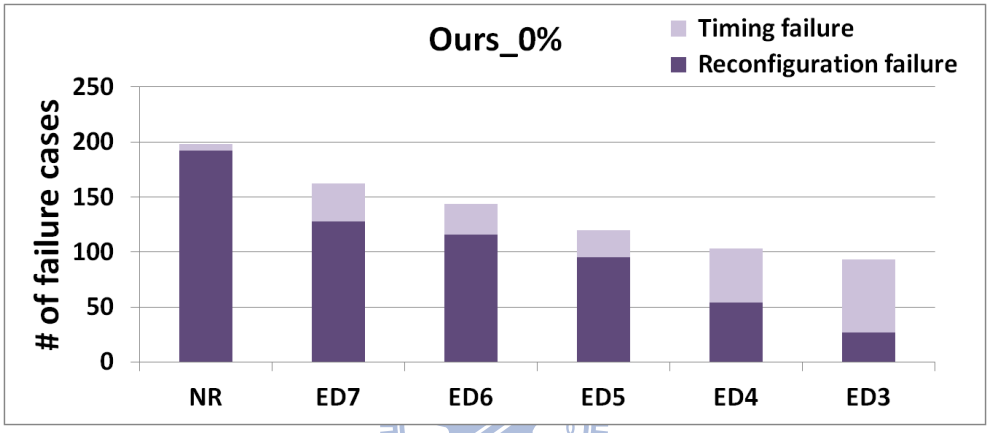
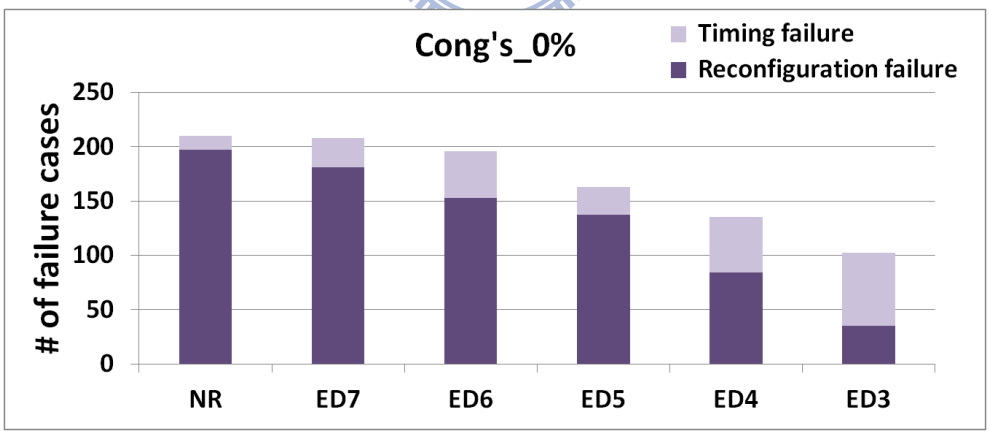


Figure 34. The success rate for clustered fault model.



(a)

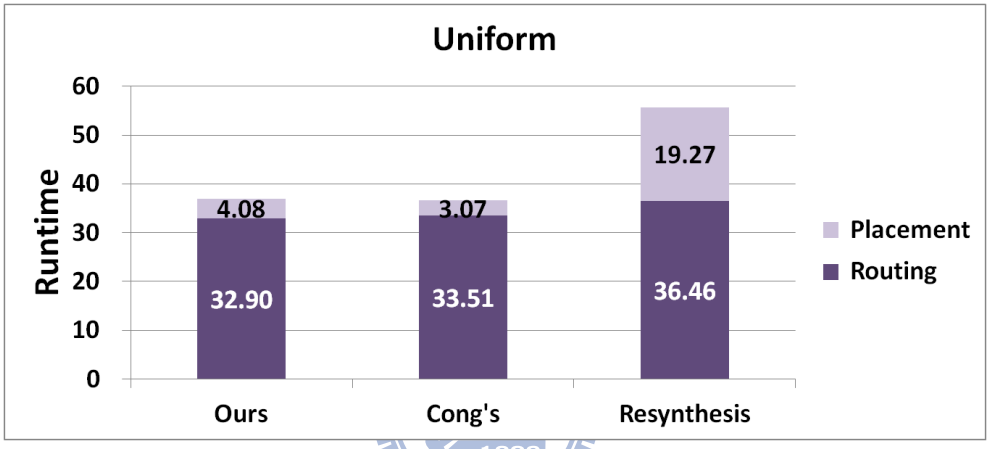


(b)

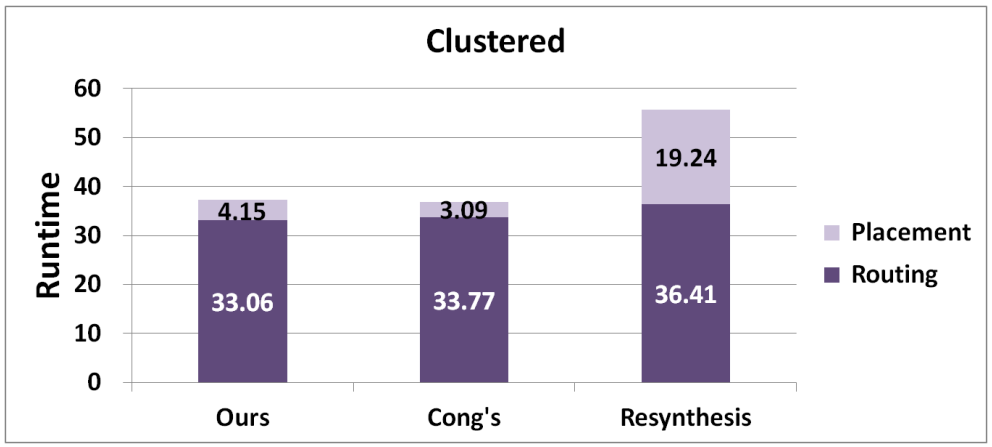
Figure 35. The number of failure cases for clustered fault model.

5.2.4 Runtime

The average runtime is shown in Figure 37. From the three configuration-level repair method, the runtime of reconfiguration methods (i.e., ours and Cong's) is roughly half of the resynthesis method. Moreover, the improvement is dominated by the placement stage since in the reconfiguration methods, constructing the DAGs and then finding the shortest paths are more efficient than SA-based method.



(a)

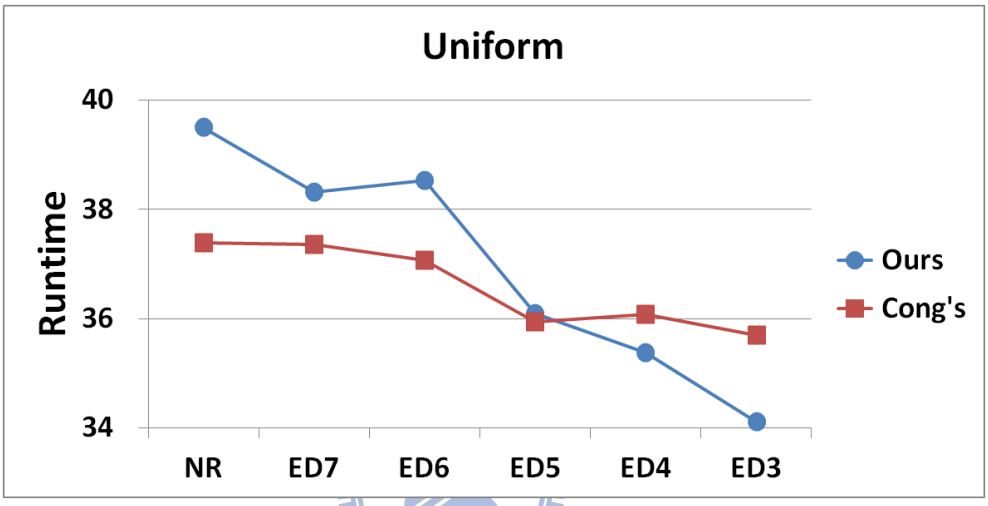


(b)

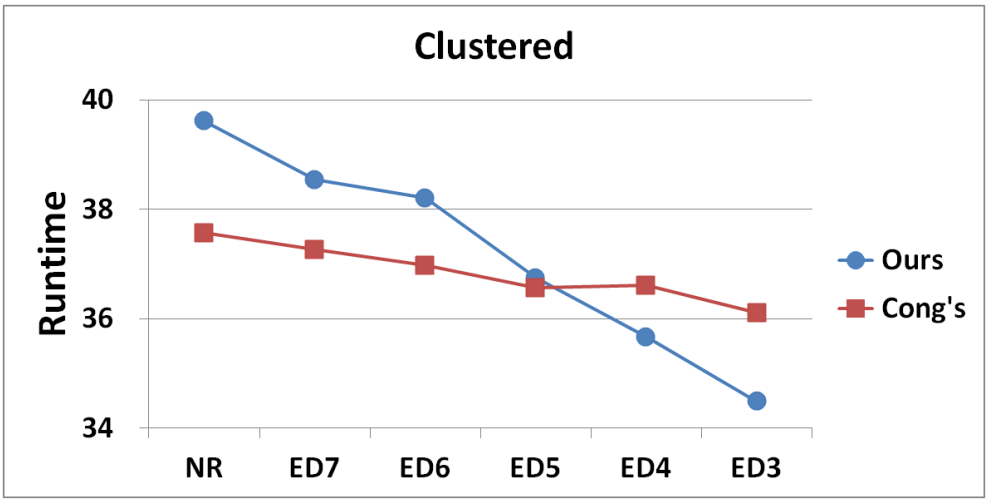
Figure 36. The runtime for uniform fault model.

In Figure 37, the runtime is separated into placement runtime and routing runtime. The runtime overhead of placement in our methods is slightly more than

Cong's because our placer considers more factors when calculating costs in the reconfiguration iterations. With more global point of view, the affected nets of our placer is less than those of Cong's, which implies less number of routing iterations will be taken. Therefore, our router runs faster than Cong's. Figure 38 shows the average runtime for each architecture, the runtime is decreased as the spare density grows because the faulty blocks are closer to spare CLBs.



(a)



(b)

Figure 37. The runtime for clustered fault model.

Chapter 6
Conclusion

As process technology scaling continues, manufacturing large fault-free integrated circuits become increasingly difficult. The architectural regularity of FPGAs provides inherent redundancy resources which can be exploited for fault tolerance and yield enhancement. In this thesis, we propose a fault tolerant reconfiguration algorithm for CLBs. A faulty block is relocated to its adjacent CLBs along a reconfiguration path from faulty and mapped CLB to non-faulty spare CLB. After all faulty CLBs are successfully reconfigured, we rip-up the affected nets and then re-route them. We also propose a generic fault tolerant architecture for 3D FPGAs that distributes spare CLBs evenly across the 3D FPGA, which provides a reconfiguration friendly architecture to improves the success rate. The experimental results show that more faults can be repaired when the fault patterns are generated using the uniform fault model than for the clustered fault model. As well, our algorithm improves up to 13% success rate for the uniform fault model and 25% success rate for the clustered fault model compared to the previous work. The runtime overhead of our method is only slightly more than the prior art.

Reference

[1] International Technology Roadmap for Semiconductor. Semiconductor Industry Association 2005–2010.

[2] A. W. Topol, D. C. La Tulipe, L. Shi, D. J. Frank, K. Bernstein, S. E. Steen, A. Kumar, G. U. Singco, A. M. Young, K. W. Guarini, and M. Ieong, "Three-dimensional integrated circuits," IBM J. of Res. and Develop., vol. 50, no. 4/5, pp. 491–506, Jul.–Sep. 2006.

[3] K. Banerjee, S. J. Souri, P. Kapur, and K. C. Saraswat, "3-D ICs: a novel chip design for improving deep submicron interconnect performance and systems-on-chip integration," Proc. IEEE, vol. 89, no. 5, pp. 602–633, May. 2001.

[4] R. R. Tummala and V. K. Madisetti, "System on chip or system on package?" Design & Test Computers, vol. 16, no. 2, pp. 48–56, Apr.–Jun. 1999.

[5] P. H. Shiu, R. Ravichandran, S. Easwar, and S. K. Lim, "Multi-layer floorplanning for reliable system-on-package," Int'l Symp. Circuits and System, pp. 23–26, 2004.

[6] K. L. Tai, "System-In-Package (SIP): challenges and opportunities," Asia South Pacific Design Automation Conf., pp. 191–196, 2000.

[7] SOCcentral. [Online]. Available: http://www.soccentral.com

[8] S. Das, A. P. Chandrakasan, and R. Reif, "Calibration of rent's rule models for three-dimensional integrated circuits," IEEE Trans. Very Large Scale Integration Systems, vol. 12, no. 4, pp. 359–366, Apr. 2004.

[9] A. Rahman and R. Reif, "System-level performance evaluation of three-dimensional integrated circuits," IEEE Trans. Very Large Scale Integration Systems, vol.8, no.6, pp. 671–678, Dec. 2000.

[10] S. Das, A. Fan, K. Chen, C. S. Tan, N. Checka, and R. Reif, "Technology, performance, and computer-aided design of three-dimensional integrated circuits," Proc. Int'l Symp. Physical Design, pp. 108–115, 2004.

[11] I. Kaya, S. Salewski, M. Olbrich, and E. Barke, "Wirelength reduction using 3D physical design," Int'l Workshop Integrated Circuit System Design, pp. 453–462, 2004.

[12] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A.M. Sule, M. Steer, and P. D. Franzon, "Demystifying 3D ICs: the pros and cons of going vertical," IEEE Design & Test of Computers, vol. 22, no. 6, pp. 498–510, Nov.–Dec. 2005.

[13] I. Loi, S. Mitra, T. H. Lee, S. Fujita, and L. Benini, "A low-overhead fault tolerance scheme for TSV-based 3D network on chip links," Proc. Int'l Conf.

Computer-Aided Design, pp. 598–602, 2008.

[14] C. Ababei, H. Mogal, and K. Bazargan, "Three-dimensional place and route for FPGAs," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 6, pp. 1132–1140, Jun. 2006.

[15] E. Stott, P. Sedcole and P. Cheung, "Fault Tolerance and Reliability in Field Programmable Gate Arrays," Computers & Digital Techniques, vol. 4, No. 3, pp. 196–210, 2010.

[16] M. Mishra, S. Goldstein, "Defect tolerance at the end of the roadmap," Proc. Int'l Test Conf. Vol. 1, pp. 1201–1210, Sep. 2003.

[17] P. Maidee, "Methodologies and Tolls for Yield Improvement of Field-programmable Logic Architectures," PhD thesis, 2009.

[18] J. Emmert, C. Stroud, and M. Abramovici, "Online Fault Tolerance for FPGA Logic Blocks," IEEE Trans on Very Large Scale Integration (VLSI) Systems, Vol. 15, No. 2, pp. 216-226, Feb. 2007.

[19] A. Mathur and C. L. Liu, "Timing-driven placement reconfiguration for fault tolerance and yield enhancementin FPGAs," Proc. European conference on Design and Test, pp. 165–169, 1996.

[20] F. Hatori, T. Sakurai, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, H. Muroga, A. Tanaka, and K. Kanzaki, "Introducing redundancy in field programmable gate arrays," Proc. CIC Conf., vol. 7, pp. 1–4, Aug. 2002.

[21] F. Hanchek and S. Dutt, "Design methodologies for tolerating cell and interconnect faults in FPGAs," Conf. Computer Design. pp. 326–331, 1996.

[22] A. Doumar and H. Ito. "Defect and fault tolerance SRAM-based FPGAs by shifting the configuration data," IEICE Trans. Inf. Syst. pp. 1104–1115, 2000.

[23] A.K. Agarwal, J. Cong, and B. Tagiku. "Fault tolerant placement and defect reconfiguration for nano-FPGAs," In Proc. Int. Conf on Computer Aided Design, 2008.

[24] J. Narasimhan, K. Nakajima, C. S. Rim, A. T. Dahbura, "Yield enhancement of programmable ASIC arrays by reconfiguration of circuit placements," IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, pp. 976–986, Aug. 1994.

[25] F. Hanchek, S. Dutt, "Node-covering based defect and fault tolerance methods for increased yield in FPGAs," Proc. 9th Int. Conf on VLSI Design. pp. 225–229, 1996.

[26] C. Ababei, Y. Feng, B. Goplen, H. Mogal, T. Zhang, K. Bazargan, and S. Sapatnekar, "Placement and routing in 3D integrated circuits," IEEE Design Test Computers, vol. 22, no. 6, pp. 520–531, Nov. 2005.

[27] Altera. [Online]. Available: http://www.altera.com/

[28] Xilinx. [Online]. Available: http://www.xilinx.com/

[29] C.-I Chen, B.-C. Lee, and J.-D. Huang, “Architectural exploration of 3D FPGAs towards a better balance between area and delay,” Proc. Design, Automation & Test in Europe Conf. and Exhibit., pp. 587–590, 2011.

[30] S. Yang, “Logic synthesis and optimization benchmarks user guide,” Technical Report 1991-IWLS-UG-Saeyang, Microelectronics Center of North Carolina, 1991.

[31] [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/benchmarks/