Timing Driven Placement Reconfiguration for Fault Tolerance and Yield Enhancement in FPGAs [*]

Anmol Mathur [†]   C. L. Liu
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Abstract

The architectural regularity of FPGAs provides an inherent redundancy which can be exploited for fault tolerance and yield enhancement. In this paper we examine the problem of reconfiguring the placement of a circuit on an FPGA to tolerate a given fault pattern in the array of CLBs. The primary objective of the placement reconfiguration is to minimize timing degradation. The concept of a slack neighborhood graph *is used as a general tool for timing driven reconfiguration with a low increase in critical path delay. Our algorithm simultaneously achieves both provably low timing degradation and low re-programming cost. For a wide range of fault probabilities and circuits our algorithm successfully reconfigures the placement with less than 1% degradation in the circuit delay.*

1 Introduction

An FPGA is a regular array of programmable logic blocks, also referred to as *configurable logic blocks* (CLBs) which can be programmed to implement any boolean function of up to K inputs. In this paper, we will concentrate on Xilinx-like architectures, where the CLBs are arranged in a 2-dimensional grid with programmable routing resources in the routing channels. The programmability of these architectures makes reconfiguration a powerful technique to tolerate faults in logic blocks. In this paper we examine the problem of reconfiguring the *placement* of a combinational circuit on an FPGA to tolerate a given fault pattern in the array of CLBs. The primary objective of the reconfiguration is to minimize the timing degradation, measured as the increase in the delay in the critical paths in the implementation.

Introduction of fault tolerance in VLSI devices has been advocated as a means of achieving both longer lifetime (through on line reconfiguration to avoid faults) and higher production yield (through production time reconfiguration of partially faulty devices). Most FPGA-based designs have a lot of unused (spare) CLBs (typical CLB utilizations are between 70-80%). This makes reconfiguration feasible and effective for FPGAs.

FPGAs are being used more and more often in the final marketed designs of digital systems. This emphasizes the need for on-line algorithms for reconfiguration in FPGAs to tolerate faults in the logic blocks, in order to increase the lifetime of the product. This is even more important in real-time and embedded applications of FPGAs.

Reconfiguration is also a useful technique for *yield enhancement* by tolerating faults in the FPGAs being used to implement a design. In this scenario, a design has been placed and routed to meet timing requirements, but some faulty CLBs are discovered prior to programming the FPGA. The placement and routing needs to be changed in such a way that the performance of the implementation does not degrade significantly.

The results of Howard et al. [1] also provide compelling evidence for the use of reconfiguration for yield enhancement in FPGAs. They argue that to overcome the relatively low logic density of FPGAs many large designs need to be split across multiple FPGAs resulting in large circuit delay. To overcome this problem, the level of integration of FPGAs needs to be increased and this leads to FPGAs with larger chip area, and consequently greater probability of faults. In order to get reasonable yield for such ultra large scale FPGAs it is imperative that some reconfiguration based fault tolerance scheme be employed.

Reconfiguration for fault tolerance in FPGAs differs from traditional processor array reconfiguration in the following ways:

- In traditional reconfiguration of faulty processor arrays, the objective is to extract the largest possible sub-array consisting only of non-faulty processors. For FPGAs, the topology to be extracted from the faulty array of CLBs depends on the circuit for which the reconfiguration is being done. So, in the context of FPGAs, it is the placement of a particular circuit on the FPGA that is reconfigured.
- The reconfiguration of circuit placement alters the critical path delay of the circuit. Consequently, it becomes necessary to develop reconfiguration algorithms that guarantee low timing degradation.

Howard et al. [1] explore various fault-tolerant FPGA architectures. Narasimhan et al. [4] investigate algorithms for reconfiguring FPGAs using minimization of the total wire length of the reconfigured design as the primary objective. The main contributions of this paper are:

- We present an algorithm for placement reconfiguration to minimize the timing degradation that also reduces the amount of re-programming that needs to be done.

[*] Work partially supported by NSF under grant MIP 92-22408.
[†] Author's current address: Silicon Graphics Inc., 2011 N. Shoreline Blvd., Mountain View, CA 94043.

- Our algorithm for timing driven reconfiguration is significantly faster than doing placement and routing again to avoid the faulty CLBs.
- We present another application of the concept of slack neighborhood graph introduced in [2] for timing driven placement in regular architectures. This shows that this technique is robust and can be applied to diverse domains.

2 The Fault Model

In this paper we consider only faults in CLBs. Faults in the routing blocks are not considered. This is realistic because in the current FPGA designs the CLBs are much more complex than the routing blocks. We use a block level fault model, which assumes that any fault in a CLB completely disables the CLB. This is a much stronger fault model than the "stuck at" fault model traditionally used for modeling faults in combinational logic. Thus, when a CLB becomes faulty, it will be discarded and an attempt will be made to map its functionality on a different CLB. A more refined fault model would take into account the fact that a faulty CLB may still be able to implement some (but not all) combinational functions.

We used two different models for the distribution of faults in our experiments:

- **Independent Fault Model:** This model assumes that the faults in CLBs are independent – that is the probability that a CLB is faulty is not dependent on the state (faulty/non-faulty) of the neighboring CLBs. The probability of a CLB being faulty is referred to as the *fault probability.*

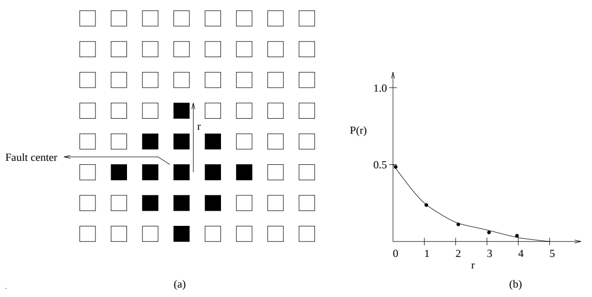

Fig. 1: (a) A clustered fault of radius 2. (b) The exponential probability distribution for fault radius.

- **Clustered Fault Model:** This model assumes that faults occur in clusters. Thus, if a CLB is faulty then its neighbors have a higher probability of being faulty. We simulate this model by assuming that a CLB has a fixed probability (termed *cluster fault probability*) of being the *center* of a fault cluster of radius r. All the CLBs within distance r from the center are assumed faulty. The radius r is assumed to have an exponentially decreasing probability distribution (see Fig. 1).

3 Problem Formulation

The input to the timing driven reconfiguration problem is a combinational circuit, its implementation on an FPGA specified by the placement of the modules and the routing of the interconnections, and the set of CLBs identified to be faulty (the *fault pattern*). The circuit is specified by its interconnection topology, module delays and the signal arrival times and required times at the primary inputs and primary outputs respectively. The output is a new placement that avoids the faulty CLBs obtained by reconfiguring the original circuit placement.

The main objectives of the reconfiguration algorithm are:

- Minimize the increase in the critical path delay as a result of reconfiguration.
- Minimize the amount of re-programming to be done to achieve the reconfiguration.

The first objective is a natural one for high performance ASICs designed using FPGAs. Low re-programming overhead may be important when FPGAs are used in a system that requires high availability. However, this objective is relevant only when the FPGA architecture allows for selective re-programming of CLBs and routing blocks. Some of the more recent FPGA architectures, such as the Concurrent Logic CLi6000 series and Algotronix FPGAs, have this feature.

4 Timing Driven Reconfiguration

In this section, we give an overview of our algorithm for timing driven placement reconfiguration. Many of the details are omitted due to lack of space. The flowchart in Fig. 2 gives an overview of the main steps involved in our timing driven reconfiguration algorithm. Our reconfiguration algorithm attempts to relocate all the modules currently mapped to faulty CLBs in such a way that increases in the delays will not be excessive. The main steps involved in this process are:

1. **Computing Distributed Edge Slacks**: The *slack of an edge* measures the amount by which the delay of the edge can be increased without violating any timing constraints. Edge slacks can be computed efficiently by finding the arrival and required times at the destination of the edge, and taking the difference of the required time and the arrival time. The slacks of the edges incident to a module determine the neighborhood within which the module can be moved without violating the timing requirements. Since several circuit modules may move simultaneously during reconfiguration, we use a variant of the zero-slack algorithm [3] to compute *distributed edge slacks*. The delays of *all* the edges can be *simultaneously* be increased by an amount equal to their distributed slacks without any timing violations. Often, a certain increase in critical path delay is acceptable, and it is not necessary to have the mobility of the modules completely restricted by the distributed slacks. We introduce a *relaxation parameter*(λ) that allows the reconfiguration algorithm to increase the values of distributed edge slacks. The distributed edge slacks augmented by the relaxation parameter are referred to as *relaxed slacks.*
2. **Constructing the Slack Neighborhood Graph (SNG)**: Increase in the delay of an edge can be translated into an increase in the length of the interconnection corresponding to the edge. Hence, the relaxed slack of an edge can be interpreted as an upper bound on the amount by which the length of the

interconnection can be increased. These bounds on the distance that a module can move without significant timing degradation, imply that a module can only be moved to certain CLBs in the neighborhood of the CLB it currently occupies. Informally, the neighboring CLBs to which a module can be moved without significant timing degradation are referred to as its *slack neighborhood*. The graph in which the adjacency relation reflects these slack neighborhoods is referred to as the *slack neighborhood graph* (see Fig. 3 (b)). The slack neighborhood graph is computed by searching the neighborhood of each module for CLBs to which it can be moved without increasing the delay of any edge incident to the module by an amount more than its relaxed slack. Notice that slack neighborhoods of IO modules are limited to the peripheral CLBs, and those of logic modules to the CLBs in the interior of the FPGA. This prevents IO modules from moving into the interior of the CLB array (and vice versa) during reconfiguration. Also, faulty CLBs are not included in the slack neighborhood of any module.

3. **Finding Reconfiguration Paths**: In order to reconfigure the circuit placement, we need to move the modules occupying faulty CLBs to non-faulty spare CLBs. We accomplish this by finding vertex disjoint paths in the slack neighborhood graph from the faulty, occupied CLBs to non-faulty spare CLBs, and then moving modules along these paths. Since these reconfiguration paths are paths in the SNG, the resulting reconfiguration has low timing degradation. In order to find vertex disjoint reconfiguration paths efficiently, and incorporate low re-programming cost as an objective in reconfiguration, we use min-cost flow in a flow network derived from the SNG.

   The sources in this flow network are the faulty, occupied CLBs and the sinks are the non-faulty, spare CLBs. The edges are the same as in the SNG and have capacity of 1 unit of flow. In addition , each vertex has a flow capacity of 1 unit. Each edge, (u, v), of the flow network is associated with a cost that depends on the following factors:

   - The cost is low if the CLB corresponding to v is spare (this encourages shorter reconfiguration paths, and lower timing degradation).
   - The cost is low if moving a module occupying CLB u has fewer nets incident to it (this encourages the min-cost flow to find reconfiguration paths with low re-programming cost).

   Finding the min-cost flow in the flow network described above yields paths from the faulty, occupied CLBs to non-faulty, spare ones. These paths are vertex disjoint due to the node capacity constraints, and they result in low re-programming cost because of the way we define the edge costs. If the SNG is not dense enough to allow the relocation of all the modules mapped to faulty CLBs, min-cost flow still yields a partial reconfiguration. In such a scenario, the relaxation parameter is increased to allow more edges in the SNG, leading to a successful reconfiguration. Unlike several other reconfiguration algorithms, our algorithm does not simply compute a matching from the faulty CLBs to the spare CLBs.

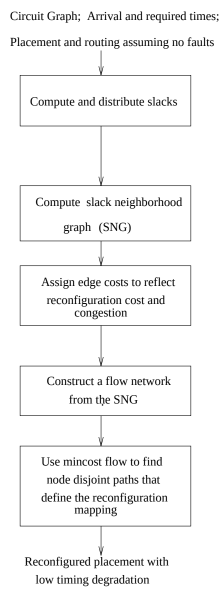
Circuit Graph; Arrival and required times;
Placement and routing assuming no faults
Compute and distribute slacks
Compute slack neighborhood graph (SNG)
Assign edge costs to reflect reconfiguration cost and congestion
Construct a flow network from the SNG
Use mincost flow to find node disjoint paths that define the reconfiguration mapping
Reconfigured placement with low timing degradation

Fig. 2: Flowchart for our timing driven reconfiguration algorithm.

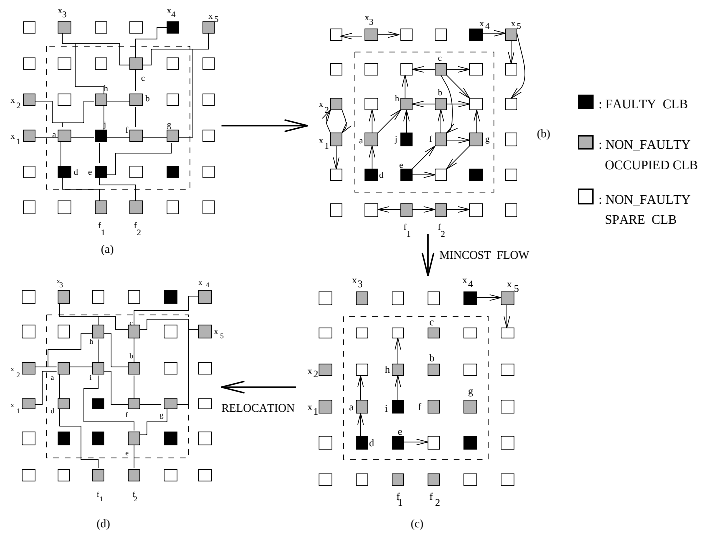

Fig. 3: An example illustrating the timing driven reconfiguration algorithm. (a) A placed and routed circuit and a fault pattern. (b) The slack neighborhood graph (SNG) for the circuit in (a). (c) The vertex disjoint paths in the SNG computed using min-cost flow. (d) The reconfigured circuit.

Fig. 3 illustrates the steps involved in our timing driven reconfiguration algorithm. An example of the vertex disjoint paths defining the reconfiguration mapping for the slack neighborhood graph in Fig. 3 (b) is shown in Fig. 3 (c). Notice that each path with non-zero flow starts at a faulty occupied CLB and ends at a non-faulty spare CLB. Moving the modules along these reconfiguration paths results in the configuration shown in Fig. 3 (d) in which all the modules are mapped to non-faulty CLBs. We will show in the next section that since the reconfiguration is done in the confines of the SNG, the timing degradation is low.

4.1 Analysis of Timing Degradation

Since the paths along which modules are moved to reconfigure the placement are found in the SNG, we can prove that the worst case increase in the critical path delay is bounded. The following theorems showing that our timing driven reconfiguration algorithm is good are stated without proof.

The following theorem gives a naïve bound on the worst case increase in the delay of the critical path in the circuit. **Theorem 1.** *In the timing driven reconfiguration, the worst case increase in the delay along any path π in the circuit is $2l(\pi)\lambda$, where $l(\pi)$ is the number of edges in the path and λ is the relaxation parameter.*

Corollary 1 *No delay increase is incurred in the timing driven reconfiguration if $\lambda = 0$. Thus, a reconfiguration with no additional slack does not cause any increase in the critical path delays of the circuit.*

The bound on the increase in delay during the timing driven reconfiguration given above is a worst case bound and the performance degradation is expected to be much lower on the average. We can prove much tighter bounds on the worst case timing degradation that we omit due to lack of space. From the above discussion we conclude that the min-cost flow based reconfiguration can be accomplished in a manner that guarantees low performance degradation.

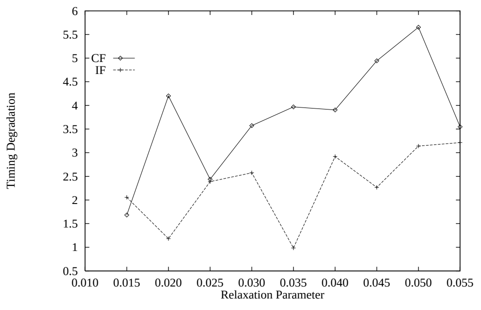

Fig. 4: Results for circuit C432 with RP = 30. IF = Independent Fault Model; CF = Clustered Fault Model.

5 Experimental Results

We implemented the slack neighborhood graph based algorithm for timing driven reconfiguration in C on a sparc10.

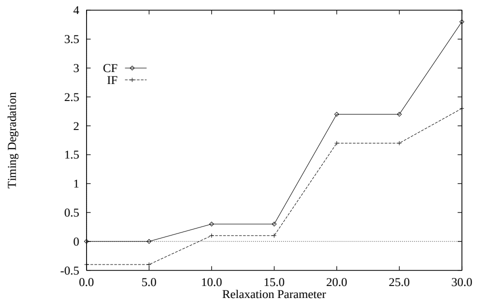

Fig. 5: Variation of timing degradation with relaxation parameter for circuit C432.

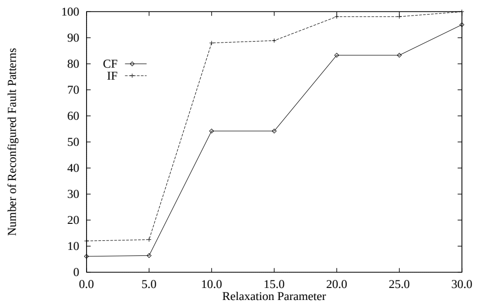

Fig. 6: Number of successfully reconfigured fault patterns (measured as percent of the total number of fault patterns) vs relaxation parameter.

	RP = 0		RP = 5		RP = 10		RP = 15		RP = 20		RP = 25		RP = 30		Average	
Circuit	IF	CF	IF	CF	IF	CF	IF	CF	IF	CF	IF	CF	IF	CF	IF	CF
C499	-0.1	-1.5	-0.1	-1.5	-0.4	-0.8	-0.4	-0.8	0.1	0.7	0.1	0.7	1.1	1 .6	0.1	-0.2
C432	-0.4	0.0	-0.4	0.0	0.1	0.3	0.1	0.3	1.7	2.2	1.7	2.2	2.3	3.8	0. 7	1.3
c8bl	-0.1	0.0	-0.1	0.0	0.1	-0.1	0.1	-0.1	0.2	-0.1	0.3	-0.1	0.2	0.1	0.1	-0.0
b9	-0.1	-0.8	-0.1	-0.8	-0.5	-1.0	-0.7	-1.0	-0.2	-0.3	-0.2	-0.3	0.3	0.4	-0.2	-0.6
ttt2	-0.0	-0.1	-0.0	-0.1	0.1	-0.0	0.1	0.0	0.2	-0.0	0.2	-0.1	0.2	0. 1	0.1	-0.0

Table 1: Average percent timing degradation as a function of the relaxation parameter (RP). IF = Independent Fault Model; CF = Clustered Fault Model.

In this section we discuss the results obtained by running our algorithm on benchmark circuits assuming random fault patterns with different fault probabilities.

We performed two sets of experiments. In the first set of experiments fault patterns were generated using the independent fault model, while in the second set of experiments fault patterns were generated using the clustered fault model described in Section 2. The input for each run of the algorithm consisted of an initial placement that satisfies all the timing constraints and a fault pattern determining the faulty CLBs. The initial placements were generated using the *sysdias* package [2]. Ten different fault patterns were generated for each value of fault probability ranging from 0.015 to 0.055 (in steps of 0.005). The size of the CLB array was 20 by 20 for all the experiments. For each circuit, reconfiguration was done using values of the relaxation parameter ranging from 0 to 30 (where a value of 10 corresponds to a distance of 1 in the CLB grid array) to study the dependence of the timing degradation on the relaxation parameter. Timing degradation for a circuit as a result of reconfiguration was measured as the percentage increase in the average of the arrival times at all the primary outputs of the circuit. The variation in the *maximum* of the arrival times at the primary outputs was also observed to be similar to the average arrival time(in fact for all our experiments the degradation in the maximum arrival time was less than in the average arrival time). Table 4.1 summarizes the average timing degradation for the 5 circuits used in our experiments for various values of the relaxation parameter. Notice that for circuits C432 and C499 the timing degradation is consistently higher for the clustered fault model than for the independent fault model. However, for circuits c8, b9 and ttt2 the degradation is almost identical for the two models. This is probably because these circuits have fewer modules and hence the number of spares available is higher making reconfiguration easier. The CPU time for one reconfiguration run was around 15-20 seconds.

Figure 4 shows the timing degradation as a function of fault probability for circuit C432 when the relaxation parameter is 30. The worst timing degradation was observed for C432, and it was around 6%. The average timing degradation for C432 was 0.7% for the independent fault model and 1.3% for the clustered fault model. For all the other circuits that we used, the average timing degradation was less than 1%. For many of the reconfigurations (specially with low fault probability) there was actually a decrease in the average arrival time. This demonstrates that our timing driven reconfiguration algorithm is able to successfully reconfigure faulty FPGAs with very low timing penalty. In general, with increase in fault probability there is an increase in the timing degradation since the presence of more faults restricts the slack neighborhoods and increases the value of flow required in the SNG for successful reconfiguration.

It is interesting to study the variation in timing degradation with changes in the relaxation parameter. As predicted by Corollary 1, there is no timing degradation when the reconfiguration is done with relaxation parameter set to zero. However, several fault patterns cannot be reconfigured when the relaxation parameter is zero. Increasing the relaxation parameter enables us to reconfigure more fault patterns. However, the timing degradation increases. This behavior is observed in the Figures 5 and 6. Also, more faults can be reconfigured when the fault patterns are generated using the independent fault model than for the clustered fault model.

References

[1] N. J. Howard, A. M. Tyrrell, N. M. Allinson, The Yield Enhancement of Field-Programmable Gate Arrays, *IEEE Trans. on VLSI Systems, Vol. 2, 04 1994, pp. 115–123.*

[2] A. Mathur, C. L. Liu, Compression-Relaxation: A New Approach to Performance Driven Placement for Regular Architectures, *Proc. Intl. Conf. on Computer-Aided Design, 1994.*

[3] R. Nair, C. L. Berman, P. S. Hauge, E. J. Yoffa, Generation of Performance Constraints for Layout, *IEEE Trans. on Computer-Aided Design, Vol. 8, Aug. 1989, pp. 860–874.*

[4] 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, Vol. 13, Aug. 1994, pp. 976–986.*