C-031

A Novel Detection and Recovery Techniques for Hard Errors in FPGAs

Motoki Amagasaki† **Masaki Inoue**† **Yuki Nishitani**† **Masahiro Iida**†
Morihiro Kuga† **Toshinori Sueyoshi**†

1.Introduction

A fault tolerance is crucial for dependable systems, such as airplanes and medical equipment. FPGAs(Field Programmable Gate Arrays) can more easily realize fault tolerant techniques owing to their programmability. However, although traditional systems are practically attained by hardware redundancy techniques, these are very expensive in terms of area, power, and delay.

If faulty points are detected, FPGAs can recover from hard error unlike ASICs(Application Specific Integrated Circuits) using its programmability. Hence, it is necessary to detect and avoid faulty points to realize fault tolerance of target systems. However, FPGA fault detection consumes a great deal of test time compared to ASICs because FPGAs have more complex structures. Moreover, once faulty points are detected, placement and routing must be performed again to avoid these points. Because the relationship of detection, avoidance and performance have a trade-off based on the detection granularity, we must consider recovery time and performance degradation.

In order to explain these trade-off, Fig. 1 shows three detection level: (A) tile level, (B)switch block(SB) level, and (C) multiplexer level. We explain these differences based on the assumption that there is one faulty in SB. The coarsest granularity is tile level detection. In this case, we need to re-place and re-route, because the tile includes logic block(LB), SBs and connection blocks(CBs). Although this case is the simplest, many hardware resources are wasted in spite of faulty multiplexer is only one. Consequently, performance degrade significantly. On the other hand, The finest granularity is multiplexer level detection. In this case, we only have to execute re-routing to avoid the faulty multiplexer. Hence, we can mitigate performance degradation. However, multiplexer level detection consumes longer time than tile level detection.

In the present paper, we propose stuck-at error detection method for global interconnects and investigates fault avoidance technique using CAD tools according to three detection levels. First, we propose the fault detection method for a faulty multiplexer in SBs. In previous studies[1][2], an easily testable routing architecture and an efficient method are introduced to reduce testing time for manufacturing test. In this study, we introduce two detection levels, which

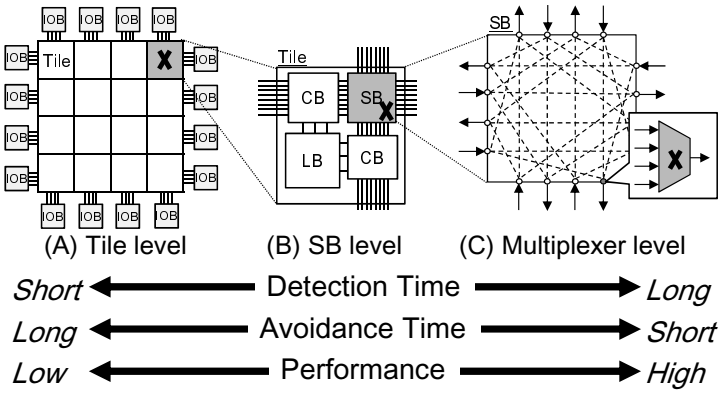

Figure 1: Relationship of three detection levels.

are tile level and multiplexer level, and basically use the previous configuration patterns for fault detection. Moreover, additional configurations are added to identify the genuine faulty point. Only six test configurations can detect candidate of faulty points by using our FPGA architecture. Next, in order to avoid faulty points, we modify VPR5.0[3], which is used for following re-placement and re-routing:

- *Re-placement*: avoidance of LB in the faulty tile.
- *Re-routing*: prevention of the signals from both SBs and CBs in the faulty tile.

Finally, we discuss the detection granularity in terms of detection time, avoidance time, and circuit performance.

We evaluated the novel detection and avoidance technique. Our method achieved efficient detection of faulty points in a short time. Moreover, we confirmed that multiplexer level detection is superior to tile level detection in terms of recovery time and circuit performance.

The remainder of the present paper is organized as follows. Related research is discussed in Section 2. Section 3 describes our previous research on testing. In Section 4, we describe the proposed fault detection method. Section 5 explains the placement and routing algorithms used to avoid multiplexer defects. In Section 6, we evaluate the proposed detection and avoidance methods and discuss their granularities. Finally, conclusions are presented in Section 7.

2.Related work

Fault avoidance for FPGA can be roughly classified into two types[4]. The first approach attempts

†Graduate School of Science and Technology, Kumamoto University

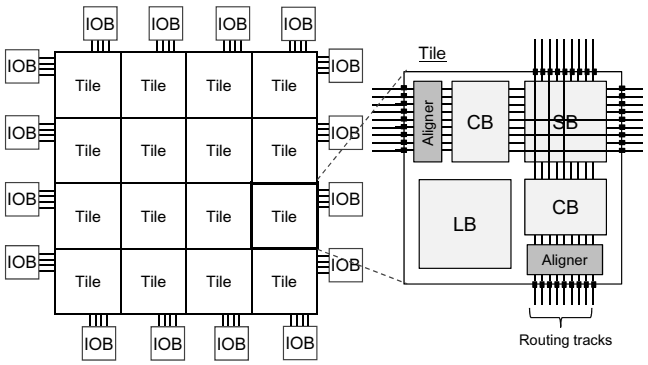

Figure 2: Completely homogeneous tile architecture.

to provide spare regions of the tile array. When faults are diagnosed during manufacturing tests, in order to avoid fault points, user circuits are re-mapped onto the spare region. In most studies involving this technique[5][6], fault avoidance is executed in tile level or block level fault detection due to the area overhead of the extra hardware resources. In this case, tile level detection is preferable to multiplexer level detection in terms of the detection time. The second approach executes re-mapping, re-placement, and re-routing the circuits[7]. In this technique, multiplexer level detection is adopted because most of the hardware resources can be used except at fault points. Thus, in order to realize an appropriate fault tolerant technique, detection and avoidance must be considered simultaneously.

3.Previous Research

This section describes our previous research[1][2] on FPGA testing. First, we explain the easily testable routing architecture, and the test configurations for SB level detection is then introduced.

3.1.Easily testable routing architecture

Although most FPGAs have complex structures to achieve high programmability, this structure makes testing difficult. Thus, we proposed a simple and genuine regular FPGA structure, as shown in Fig. 2. In this architecture, all tiles have the same structure, unlike the traditional island-style FPGA architecture, which is composed of several types of different tiles. We also implement aligners to simplify the connections of wire segments. Moreover, we prepared test pattern generators (TPGs) and output response analyzers (ORAs) in IOBs and LBs to perform testing effectively, as shown in Fig. 3. As such, only five test configurations can completely test the routing resources for manufacturing test.

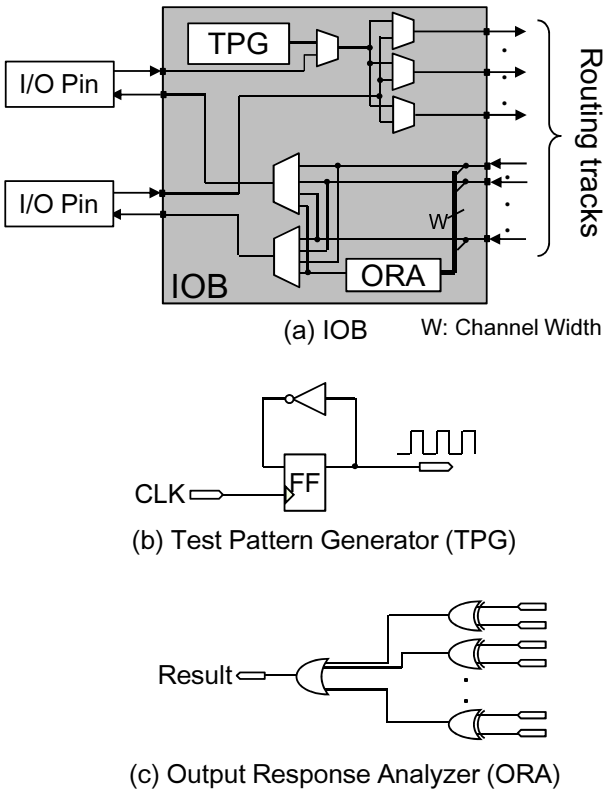

Figure 3: IOB structure for testing.

3.2.Configurations for global track testing

The configuration patterns are generated using the regularity of the Wilton-type SB[8]. The Wilton-type SB can be divided into three types of paths: (a) orthogonal, (b) clockwise, and (c) counterclockwise, as shown in Fig. 4. The feature is that each path is tested individually. For example, when all of the SBs are configured as clockwise paths, the propagation paths are formed as a single stroke path (see Fig. 5). In this case, test signals enter from LBs or IOBs, and all clockwise paths are tested completely. The other two path types can also be tested in the same manner. These configurations achieved 100% of fault coverage for global interconnections.

4.Fault Detection

In this section, we define the fault model, and propose a fault detection technique and additional configurations for test.

4.1.Fault model

The stuck-at fault[9] is a logical fault model that has been used successfully for decades. A stuck-at fault affects the state of logic signals on lines in a logic circuit, including primary inputs, primary outputs, internal gate inputs and outputs. A stuck-at fault transforms the correct value on the faulty signal line

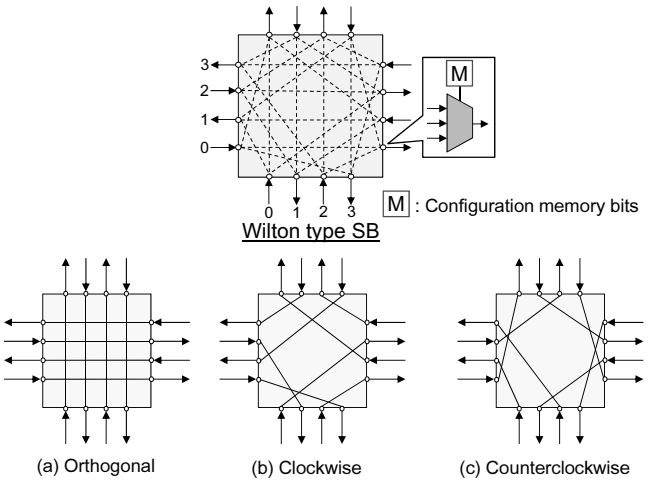

Figure 4: Three types of SB configurations.

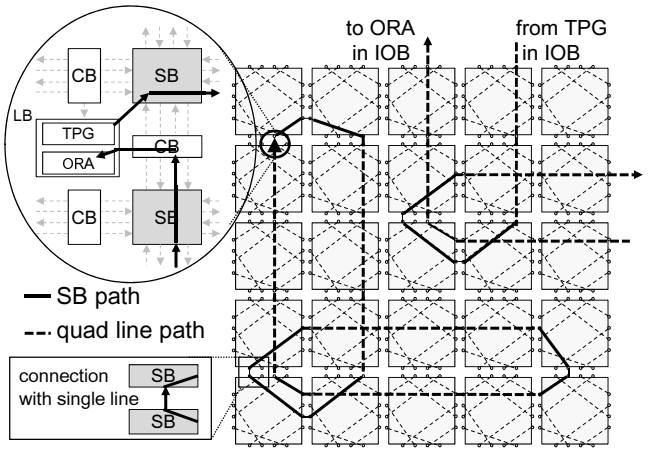

Figure 5: Configuration for clockwise paths.

to appear to be stuck at a constant logic value, either a logic 0 or logic 1, referred to as stuck-at-0(SA0) or stuck-at-1(SA1), respectively, as shown in Fig. 6. In this paper, we employ stuck-at model as a fault model.

4.2.Novel detection technique

The configurations of our previous research attempt to detect SB level defection. However, these configurations cannot specify the genuine defect points. For example, even if a fault exists in only one SB, these configurations cannot specify the fault, as shown in Fig. 7(a). Moreover, because all of the candidates are judged as faults, and we cannot use seven SBs in the results. Therefore, we prepare additional test configurations for different test paths, as shown in Fig. 7(b), and identify the location of the SB level or multiplexer level defect by margining both of (a) and (b), as shown in Fig. 7(c) or 7(d).

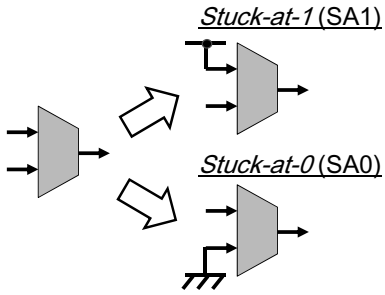

Figure 6: Stuck-at fault model.

4.3.Additional configurations

We propose two additional test paths, as shown in Fig. 8. The first is the combination of the right-up path and the orthogonal path. Second combines the left-up path and the orthogonal path. Fig. 9 shows all of the SBs configured to the right-up and orthogonal paths. The SBs configured to the right-up path and the orthogonal path are aligned alternately. In this configuration, we enter the toggle signals from the TPGs in IOBs to cover the entire device. Finally, these signals arrive at the ORAs in the IOBs. All of the paths can be tested simultaneously because they do not interfere with each other.

Moreover, we can shift one row of the configuration bits using a shift-configuration technique[2], as shown in Fig. 10. By shifting the configuration bits, the SBs configured to the right-up path are re-positioned on the orthogonal path, and the SBs configured to the orthogonal path are re-positioned to the right-up path. Although these paths change slightly compared to Fig. 9 patterns, we can obtain accuracy enhancement for multiplexer level detection. The shift-configuration time is very short compared to the entire configuration time. Therefore, this method improves the detect precision and the test time. The merged left-up and orthogonal test path is tested in the same manner as the merged right-up and orthogonal path.

5.Fault Avoidance

In order to avoid the defected point in tile level or multiplexer level, we incorporate the fault avoidance function into the placement and routing tools based on VPR5.0[3].

5.1.Placement

In the case of tile level avoidance, we cannot use the LB in the faulty tile. Thus, we need to avoid such LBs in the placement process. Fig. 11 shows the pseudo-code of the placement having the fault avoidance function. This placement algorithm is changed on VPlacer[10], main differences are lines 5 and 14. In

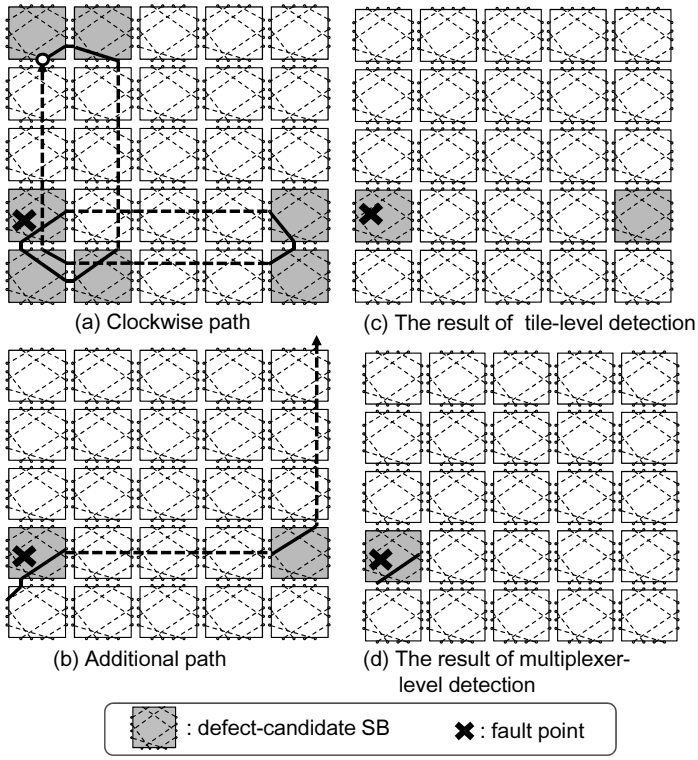

Figure 7: Combining the paths to specify the faulty point.

these steps, we check whether selected LBs are defect blocks. If defect blocks are used, placement will be re-attempted until no defect block is selected. Note that lines 5 and 14 are not executed in multiplexer level avoidance.

5.2.Routing

In the cases of tile level and multiplexer level avoidance, we need to avoid defect multiplexers and connected wires. Therefore, we modify the pathfinder routing algorithm[10], as shown in Fig. 12. In line 21, we determine whether the selected node, such as a multiplexer or a wire, is defective.

6.Evaluation

First, we evaluate the novel detection technique in terms of test time and detection precision compared to the previous test technique[1]. Next, we avoid the faulty points using the modified placement and routing tools. In order to clear affection of fault avoidance, circuit performance and recovery time are evaluated. Finally, we discuss the granularity of detection by considering the trade-off between test time, performance, and recovery time.

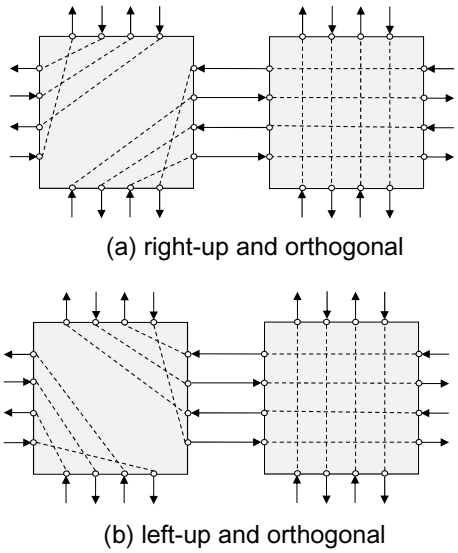

Figure 8: Additional test paths.

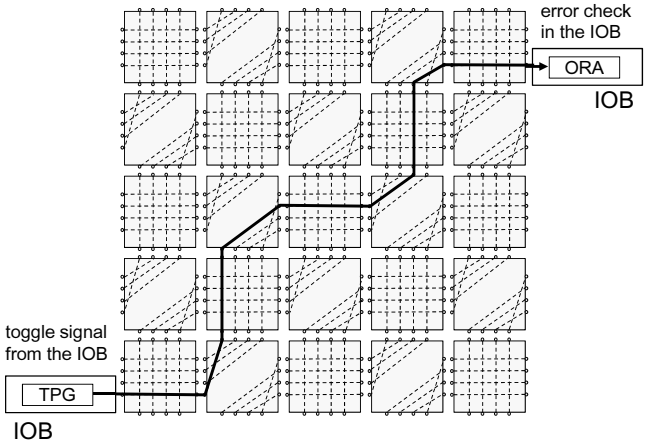

Figure 9: Test configuration for right-up and orthogonal paths.

6.1.Evaluation condition

Target device for this evaluation is shown table 1. The device is a homogeneous FPGA[1] (see Fig. 2), which consists of 16×16 LBs, each of which has four 6-LUTs. We design this FPGA architecture using Verilog-HDL and synthesize a gate level netlist with Synopsys Design Compiler Y-2006.06-SP6-2 by using a 65-nm CMOS standard cell library. Next, to perform fault detection, we execute a logic simulation using Cadence NC-Verilog 06.20-s004. In this study, toggle coverage is introduced as fault coverage for detecting stuck-at faults. Moreover, we implement MCNC benchmark circuits using ABC mapper[11] and T-VPack[10] CAD tools and modified placement and routing tools to clarify the performance. The specification of imulation machine is composed of an

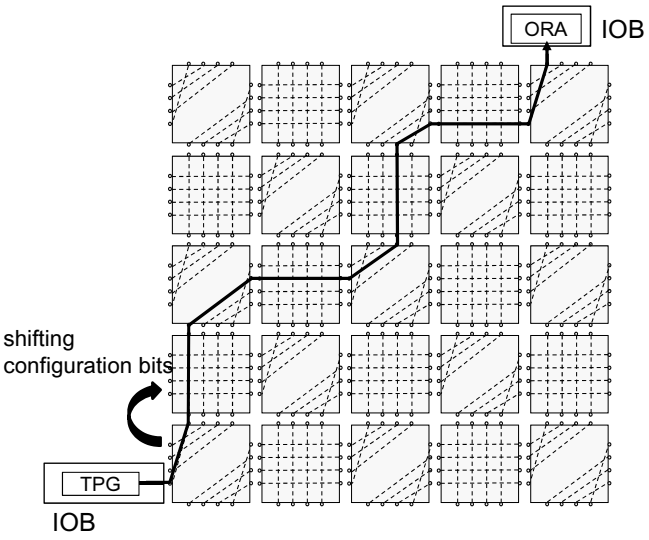

Figure 10: Shifted test configuration.

```
1: program fault avoidable routing algorithm
2:  /*RT: Routing tree, Crit: Criticality of the net,
3:   p: present congestion cost
4:  /*h: historical congestion */
5:    Crit(i, j) = 1; /* for all nets i and sinks j */
6:    while(overused resources exist) { /* Illegal routing? */
7:       for(each net i) {
8:         rip-up routing tree RT(i) and update affected p(n) values;
9:         RT(i) = NetSource(i);
10:        for(each sink j of net(i) in decreasing Crit(i, j) order) {
11:           PathCost(n) = Crit(i,j)·delay(n) for n in RT(i);
12:           PriorityQueue = Addtree(RT(i), PathCost(n));
13:           while(sink(i, j) not found) {
14:             m = LowestCost(PriorityQueue);
15:              for(all fanout nodes n of node m) {
16:               PathCost(n) = Cost(n) + PathCost(m);
17:               PriorityQueue = Addnode(PathCost(n));
18:              }
19:           }
20:            for(all nodes n in path from RT(i) to sink(i, j)) {
21:              if(defect_node(n) == False)
22:              { /* check whether fault node */
23:                 Update(p(n));
24:                 RT(i) = Addnode(n);
25:              }
26:           }
27:        }
28:       }
29:       for(all nodes n)
30:         Update(h(n));
31:       for(all nets i and sinks j)
32:         timing_analysis_and_update(Crit(i, j));
33:  } /* End of one routing iteration */
```

Figure 12: Pseudo-code in fault avoidable routing

```
1: program fault avoidable placement algorithm
2:  /*S: Solution, T: Temperature, C: Cost, R_limit: Range limiter */
3:
4:    /*Initial placement*/
5:    while(UsingDefectBlock(S) == True)
6:    {/* Fault check */
7:      S = RandomPlacement();
8:     }
9:    T = InitialTemperature();
10:   R_limit = InitialR_limit();
11:
12:    while(ExitCriterion() == False) { /* Outer loop */
13:       while(InnerLoopCriterion() == False) { /* Inner loop */
14:          while(UsingDefectBlock(S_new) == True) {
15:            /* Fault check */
16:            S_new = GenerateViaMove(S, R_limit);
17:            }
18:            ΔC = Cost(S_new)·Cost(S);
19:            r = random(0, 1);
20:            if(r < e^(−δC/T)) {
21:              S = S_new;
22:            }
23:        } /* End of Inner loop */
24:       T = UpdateTmp();
25:       R_limit = UpdateR_limit();
26:    } /* End of Outer loop */
```

Figure 11: Pseudo-code in fault avoidable placement

Table 1: Target device for evaluation.

Item	Value
array size	16 × 16
logic element	6-LUT× 4
# of LB inputs	12
SB	Wilton (Fs = 3)
CB	normal (Fc = 0.5)
# of single lines	8/channel
# of quad lines	40/channel
# of I/O pins	128
# of configuration bits	136,896

Intel Xeon X5680 3.30-GHz processor and 48 GBytes of memory. In this evaluation, we assume that only one multiplexer in global interconnects has fault.

6.2.Fault detection

The results of fault detection are shown in Table 2. Note that we show both the worst and best cases for the number of detected fault tiles and/or multiplexers because they depend on the route of the test path and the ability of ORAs. For example, when test path is long, the test signal passed through more tiles and/or multiplexers than the case of shorter test paths. As a result, the previous test technique[1] required five configurations and 798,992 clock cycles to achieve 100% fault coverage of global interconnects. However, the precision of the fault detection is low: 44 tiles at worst and 4 tiles at best. In contrast, the proposed technique can reduce the number of fault candidate tiles to 7 at worst and 1 at best, but the cost of test time increased by 40.5%. The number of fault candidate multiplexers decreased to approximately 10% by using additional test configurations.

6.3.Fault avoidance

In this evaluation, we injected a single stuck-at fault into the multiplexer in the SB. Note that we prepared two samples faulty FPGAs(samples 1 and 2), which have single faults in different locations. The faulty points are determined randomly to fair comparison. The fault information for each FPGA is shown in Table 3. For example, for a single stuck-at fault, 23 tiles and 78 multiplexers were regarded as fault candidate resources by the previous test method in sample1. On

Table 2: Results of detection.

	Previous	Proposed
# of normal configurations	5	6
# of shift-configurations	–	2
test time (clock cycle)	798,992	1,122,970
time increase	–	+40.5%
# of fault tiles (worst)	44	7
# of fault tiles (best)	4	1
# of fault multiplexers (worst)	144	14
# of fault multiplexers (best)	12	1

Table 3: Candidates of deteced faulty point.

	Previous		Proposed	
Level	Tile	MUX	Tile	MUX
sample1	23	78	2	6
sample2	25	48	2	2

MUX: multiplexer

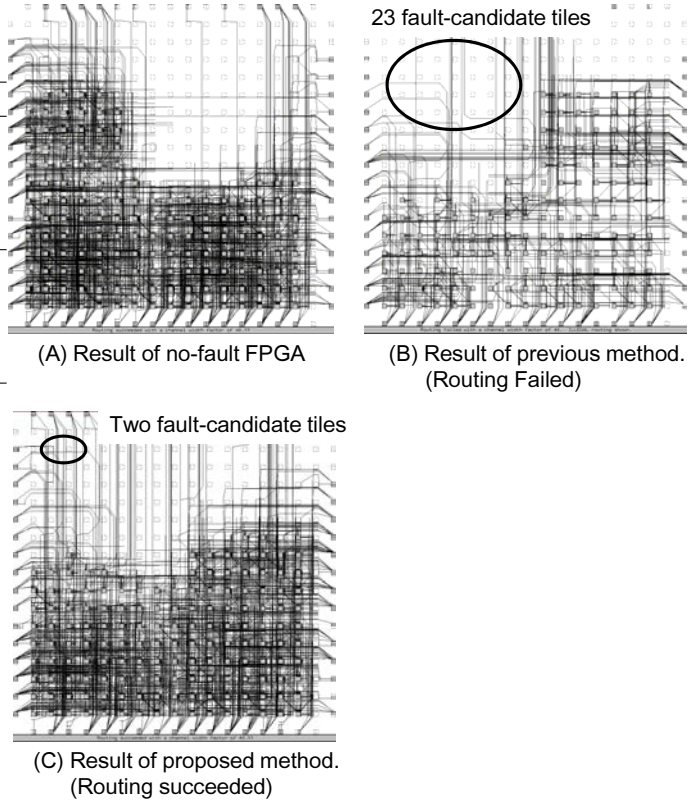

Figure 13: Result of ex4p benchmark circuit.

the other hand, the fault resources were two tiles and six multiplexers in the proposed method.

In order to discuss the effect of precision of fault detection, we implemented benchmark circuits in which (1) no fault occurs, (2) the faulty tiles are detected by the previous test method[1], and (3) the faulty tiles are detected by the proposed method. Table 4 shows the critical path delays in three cases. As a result, the delay of the proposed method was increased by 2% compared to the no-fault FPGA, but improved 4% compared to the previous method. Further, the previous method failed to implement all benchmarks in the sample-2 because many candidates of faulty tiles exist. On the other hand, implementation was successful for all benchmarks using the proposed method. Fig. 13 shows example of implementation with ex4p benchmark: (A) is the result of no fault, (B) previous method and (C) proposed method. In this example, there are two faults at upper left area of the tile array. Hence, the circuit was avoided this area in Figs. 13(B) and(C). Note that, Fig. 13(B) failed to routing because of the large number of faulty candidate tiles. Similarly, Table 5 shows the delays of multiplexer level avoidance. Using additional configurations, the delays of the proposed method were approximately the same as the no-fault FPGA.

The recovery times are shown in Table 6. The time of multiplexer level avoidance was far shorter than that of tile level one. The recovery time of tile level is 82% shorter than one of multiplexer level on average.

6.4.Discussion

From Tables 4 and 5, the success ratio of recovery is quite different between the tile level and the multiplexer level. Furthermore, previous method[1] failed to implement all of the circuits for the sample-2 fault, even if the number of faulty tiles is slightly different from sample-1. Most of the failed cases in sample-2 are placement failures. In contrast, although sample-1 has 30 more defect multiplexers than sample-2, all of the benchmarks were succeeded. These results indicate that the main cause of failure is excess avoidance of LBs. Hence, global interconnect faults should be avoided at the multiplexer level.

In terms of avoidance time, multiplexer level avoidance provides better results than tile level avoidance. Moreover, the proposed detection technique can identify the defect multiplexers with high precision in a short time. Consequently, multiplexer level fault tolerance is preferable to tile level fault tolerance in terms of fault detection and avoidance time, success ratio of recovery, and circuit performance for the faults of global interconnect.

7.Conclusion

In this study, we proposed a fault detection technique for the global interconnect and developed placement and routing tools to avoid faulty points in vari-

Table 4: Delays of tile avoidance [*nsec*]

circuit	fault type	no fault	Previous	Proposed
C6288	sample-1	96.7	93.2	93.5
	sample-2	96.7	N/A	100.3
cordic	sample-1	103.4	97.9	91.7
	sample-2	103.4	N/A	111.4
ex4p	sample-1	46.4	N/A	45.7
	sample-2	46.4	N/A	49.1
vg2	sample-1	39.3	42.2	41.5
	sample-2	39.3	N/A	40.2

N/A: Implementation failed.

Table 5: Delays of multiplexer avoidance [*nsec*]

circuit	fault type	no fault	Previous	Proposed
C6288	sample-1	96.7	97.0	96.7
	sample-2	96.7	9.66	96.2
cordic	sample-1	103.4	98.4	103.4
	sample-2	103.4	102.2	93.9
ex4p	sample-1	46.4	46.8	46.3
	sample-2	46.4	46.7	47.4
vg2	sample-1	39.3	38.7	39.3
	sample-2	39.3	42.4	41.1

ous granularities. In the proposed technique, we realized high-precision fault detection with only six test configurations. As a result, the success ratio of fault avoidance is improved compared to previous study[1]. Moreover, the performance degradation caused by faults is very slight in multiplexer level avoidance. At present, we are in the process of developing a detection and avoidance method for other circuit blocks, including LBs, CBs, and IOBs.

Table 6: Avoidance time[*sec*]

circuit	Tile (re-place&routing)	MUX (re-routing only)
C6288	4.25	0.94
cordic	3.60	0.87
ex4p	3.28	0.38
vg2	1.12	0.15

参考文献

[1] K. Inoue, H. Yosho, M. Amagasaki, M. Iida, and T. Sueyoshi, “An Easily Testable Routing Architecture and Efficient Test Technique ,” in *Proceedings of the 21th International Conference on Field Programmable Logic and Applications*, Sep. 2011, pp. 291–294.

[2] K. Inoue, M. Koga, M. Iida, M. Amagasaki, Y. Ichida, M. Saji, J. Iida, and T. Sueyoshi, “An easily testable routing architecture and prototype chip,” *IEICE Transactions on Information and Systems*, vol. E95-D, pp. 303–313, Feb. 2012.

[3] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. Fang, and J. Rose, “Vpr 5.0: Fpga cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling,” in *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, Feb. 2009, pp. 133–142.

[4] J. Cheatham, J. Emmert, and S. Baumgart, “A survey of fault tolerant methodologies for fpgas,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, no. 2, pp. 501–533, 2006.

[5] S. Durand and C. Piguet, “Fpga with self-repair capabilities,” in *Proceedings of the ACM International Workshop on FPGAs*, 1994.

[6] N. Howard, A. Tyrrell, and N. Allinson, “The yield enhancement of field-programmable gate arrays,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 2, no. 1, pp. 115–123, 1994.

[7] V. Lakamraju and R. Tessier, “Tolerating operational faults in cluster-based fpgas,” in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, 2000, pp. 187–194.

[8] G. G. Lemieux and D. M. Lewis., “Analytical framework for switch block design,” in *Proceedings of International Conference on Field-Programmable Logic and Applications(FPL)*, 2002, pp. 122–131.

[9] M. A. B. M. Abramovici and A. D. Friedman, *Digital Systems Testing and Testable Design*. Wiley Interscience, 1993.

[10] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for deep-submicron FPGAs*. Kluwer Academic, 1999.

[11] “ABC: A System for Sequential Synthesis and Verification.” [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc/