Node-Covering Based Defect and Fault Tolerance Methods for Increased Yield in FPGAs*

Fran Hanchek and Shantanu Dutt

Department of Electrical Engineering
University of Minnesota
Minneapolis, MN 55455

Abstract

Fault tolerant techniques are proposed which make use of the reconfigurability of SRAM-based field programmable gate arrays (FPGAs). Based on the principle of node-covering, a routing discipline is developed that reserves unused wiring in the routing channels to allow each cell to cover (to be able to replace) its neighbor in a row. If testing identifies a faulty cell, switches are set to reconfigure the faulty cell out of the array. Not only can reconfiguration of the FPGA be performed by the user, but it can also be done at the factory in such a way as to be transparent to a user programming the array. This can result in substantial yield improvement.

1 Introduction

Our model characterizes a Field Programmable Gate Array (FPGA) as being composed of an array of programmable logic cells surrounded by channels of segmented programmable interconnection wiring. The channel wiring architecture is shown in Fig. 1, and a logic cell consists of programmable combinational functions with optional output registers. This model is similar to the Xilinx FPGA architecture [7]. We are interested in reprogrammable FPGAs, in which data in a configuration memory overlaid on the array defines the interconnection of the channel wiring and the functions performed by the logic cells. Signals from the memory control pass transistors to make or break programmable connections and define the functions realized by the array. Reprogrammability allows a test configuration to be programmed into an FPGA at the factory, or at power-up in a user's system, before reprogramming to the desired configuration.

Fault tolerance (FT) in FPGAs is desirable from two points of view. For the manufacturer, it can increase the yield of usable chips. If faults can be detected and reconfigured around at the factory in a manner such that they are transparent to the user, then these chips do not have to be discarded. For the user, fault tolerance means reduced downtime and lower maintenance costs. We propose a technique whereby FPGAs may be reconfigured around faults detected at the factory in ways transparent to the user, thus increasing the overall yield of usable chips. We also show how the technique can be modified to allow the user to automatically reconfigure an FPGA around faulty cells.

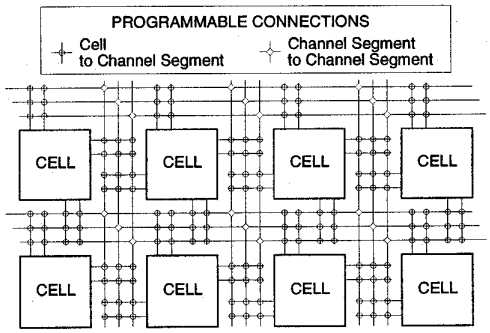

Figure 1: FPGA generic channel wiring architecture with segment length spanning one cell.

1.1 Existing Fault Tolerance Methods

Since FPGAs are composed of a large number of identical cells in a highly regular array, two methods of providing fault tolerance are readily apparent. One is simply to note the locations of faulty cells and reroute the user's circuit to avoid them using spares or other unused cells instead [3, 6]. However, requiring the layout tools to perform a new routing of a circuit for each new faulty cell location encountered puts a heavy burden on the user, who must also keep track of all of the different routings for a given circuit design. A variation on this rerouting technique makes fault tolerance transparent to the user by using extra wiring and factory-configured switches on the tracks to physically "warp" the channel routing segments around a faulty cell while maintaining the same logical routing configuration. The extra switch delay overhead is too high for use in reconfiguring around individual FPGA cells, but can be employed more efficiently using blocks of cells [5]. However, the area overhead of spare blocks of cells and extra wiring is also very high.

The other fault tolerant technique, adding spare rows and/or columns of cells, is intended for recon-

*This research was funded in part by NSF grant MIP-9210049 and in part by funds from the graduate school at the University of Minnesota. The authors can be reached at {hanchek, dutt}@ee.umn.edu.

figuration at the factory, making the technique transparent to the user. To reconfigure around a faulty row, fuses are burned at the factory such that non-faulty rows are remapped to include the spare row. For the faulty row to be transparent, it is necessary to maintain the original connectivity between the rows on either side of the faulty one. One method of doing this is to employ longer wiring segments in the vertical channels, but then extra *tracks* (segmented channel wires) must be added to these channels to retain the original routing flexibility [4].

1.2 Proposed Fault Tolerant Method

Variations on our proposed method of fault tolerance allow reconfiguration around faults to be performed by the user, by the factory, or by both. We require neither the factory nor the user to generate new routing maps to reconfigure around faulty cells. Instead, the original configuration data can be reused. No extra switches need to be added in the channel wiring, as is done in [5], but the method involves a routing strategy that requires the use of additional wiring segments. On the other hand, no additional tracks are needed in the channels in order to avoid the loss of connection flexibility seen in [4]. Our method has the low overhead of spare rows or columns, but promises greater yield improvement because it is a finer-grained FT method and is able to tolerate more fault patterns.

2 Node Covering Method

Under the principle of *node-covering* [2], each primary node, or cell, u in the FPGA is assigned a *cover cell* which can be reconfigured to replace it in the event that cell u becomes faulty. Primary cells are assigned to cover other primary cells in a chain-like manner, with a spare cell covering the last primary cell in the chain. If a faulty cell is identified through testing, the FPGA can be reconfigured such that the faulty cell is replaced by its cover, which in turn is replaced by its own cover, and so on until a spare cell in the chain is reached.

Our FT FPGA designs will be able to tolerate one fault in each row (or column), and this will subsequently be called the *node-covering* method. In order for a cell to cover another cell (the *dependent cell*), two conditions must be met. First, the cover cell must be able to duplicate the functionality of the dependent cell. This is easy in an FPGA, since all cells are identical. Configuration data for the dependent cell itself is simply transposed to the cover cell. Second, the cover cell must be able to duplicate the connectivity of the dependent cell with respect to the rest of the array. Our method of ensuring connectivity is described next.

2.1 Cover Segments

Consider a row of the FPGA to be a fault tolerant group, with the rightmost cell being a spare. We will assume the generic channel wiring of Fig. 1, though the technique also applies to segments of longer lengths. The channel segment interconnections will be considered to be separate from the cell-to-channel-segment connections, which will be associated with the cell configuration. Therefore, when cell configuration data is transposed to a cover cell, all of the cell-to-channel connection data is transposed as well. Figure 2 shows some representative nets routed to allow the replacement of each cell by its cover cell. These nets are shown as (a) initially configured, and (b) reconfigured around faulty cells.

To meet the connectivity requirement, each net connected to a cell through a channel segment must also include the corresponding channel segment—a *cover segment*—bordering the cover cell. Cover segments are included in a net in one of two ways. First, segments in the net may already be in positions to act as covers. For example, in Fig. 2a, the channel segment connecting to Cell A is covered by the net channel segment connecting to Cell B. In case the above condition does not hold, additional segments, termed *reserved segments*, should be attached to the net to provide covers, and these constitute overhead since they are only in use when the circuit is reconfigured around a faulty cell. For example, the channel segment connecting to Cell B must be covered by a reserved segment that can be connected to Cell C, which covers Cell B. With fault tolerant groups defined along rows, covering a cell connection to a horizontal channel segment requires, at most, one reserved segment. Covering a cell connection to a vertical channel segment, however, can require up to two reserved segments if the cover is not already provided in the net. This is illustrated in

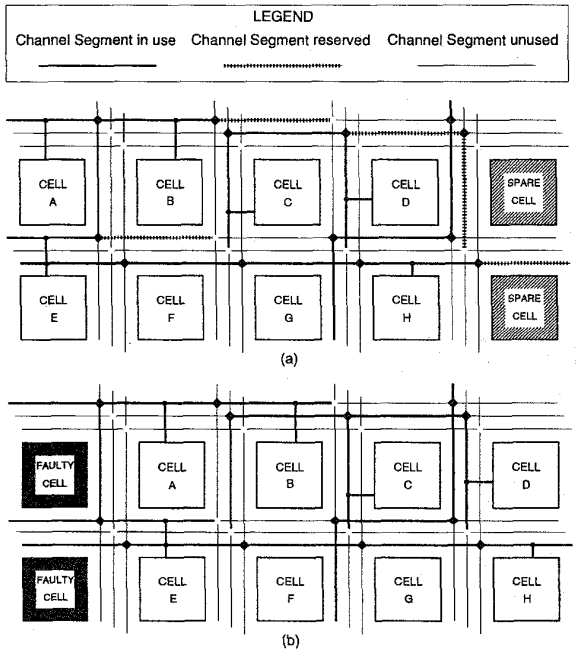

Figure 2: (a) Fault tolerant routing of nets using cover segments. (b) Reconfiguration around faulty cells.

Fig. 2a, where it can be seen that the vertical channel segment connecting to Cell C is already covered by channel segments of the same net connecting to Cell D. However, for the net connection to Cell D two reserved segments are required for the connection to Cell D's cover, the spare cell at the end of that row. One is the vertical cover segment that can be connected to the spare cell if and when the spare replaces Cell D; the other is a horizontal segment to connect the vertical segment to the net.

A *point-to-point path* is defined as a connection either between a pair of cell terminals or between an I/O pad and a cell terminal. A net consists of such intersecting point-to-point paths. When two paths of a net connect at some point, the unused track segments available might not allow them to share the same segment at the connection point, and thus be connected along a single track. In such a situation, the router may perform a "track-hop," in which the two point-to-point paths are routed along different tracks (in the same channel), using the cell terminal to connect the tracks. This is illustrated in Fig. 2a with a track-hop at the connection to Cell E. In such a situation, there are two segments of the same net connected to Cell E, and both must be covered. The cover segment for the lower one is already part of the net, and a reserved segment over cover Cell F must be added to cover the other, as seen in Fig. 2a. A reconfiguration is shown in Fig. 2b, where the track-hop is now performed at the new Cell E (previously Cell F).

2.2 Reconfiguration

The reconfiguration procedure described here assumes that cover segments necessary for reconfiguration around a faulty cell are provided by the routing tools and included in the initial configuration data. Our model of the configuration memory assumes that its data is loaded serially.

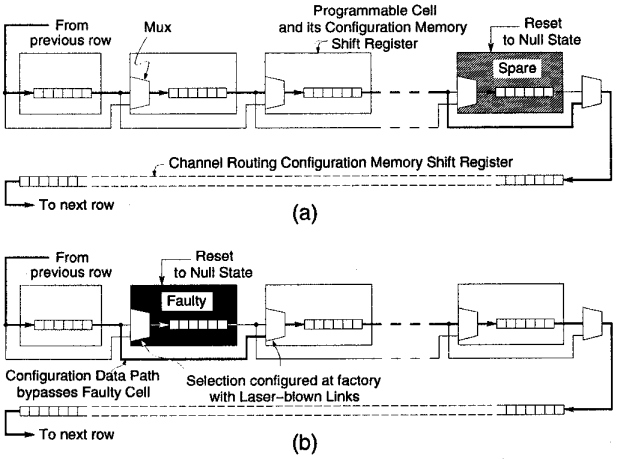

Figure 3: (a) Row of factory reconfigurable FPGA cells. (b) Reconfigured around faulty cell with configuration data for the faulty cell reset to the null state. Flow path of configuration data is highlighted.

Figure 3a shows the configuration memory for a fault tolerant row of an FPGA, where the last cell in the row is a spare. Configuration data for a cell and its connections to the channel wiring is grouped contiguously in its shift register. Configuration data for the channel segment interconnections is kept separate from the cell data for a row, either in its own portion of the same shift register or else in its own shift register. The last (spare) cell in the row is initially reset to a null configuration to disconnect it from the array. Configuration data produced by the user's placement and routing tools assumes that the primary cells are non-faulty and maps them to the user's circuit accordingly. However, it also includes channel segment covers to allow for reconfiguration around one faulty cell per row. Both spares and faulty cells are transparent to the user. When configuration data is loaded in the absence of faults, it simply bypasses the spare cell enroute to the other rows, and a factory-enabled signal forces the configuration pass-transistors of the spare cell to the OFF state in order to disconnect that cell from the array, as shown in Fig. 3a. If a cell is faulty, the faulty cell's configuration pass-transistors are turned off instead. The user does not need to know the location of factory-detected faulty cells and does not need to modify the configuration data obtained from the router. Instead, laser-programmable links burned at the factory allow configuration data to bypass the detected faulty cells and travel to the cover cells when it is loaded into the FPGA. Configuration data originally intended for the faulty cell and the cells following it in that row is rerouted by a factory-programmed mux in the cell so that it automatically flows instead to the cover cells, including the spare cell, as shown in Fig. 3b. Since the FT routing rules ensure that cover segments are already programmed into the channel segment configuration data, a correct reconfiguration is automatically achieved when the configuration data is loaded, and any factory-detected faults are transparent to the user adhering to these rules.

If the laser-programmable links are replaced by switches controlled by non-volatile *microconfiguration memory* bits (e.g. EEPROM), reconfiguration can be performed by both the factory and the user. As with laser-programmable links, reconfiguration performed at the factory is transparent to the user. However, a user detecting any additional fault now has the capability of reading from and writing to the microconfiguration memory to reconfigure around the new fault.

3 Eliminating Reserved Segment Delay

Due to the additional parasitic capacitance, the propagation delay of a net will increase if reserved segments are left attached to the nets before any reconfiguration. This effect can be minimized and even eliminated if reserved segments are not actually connected to the nets until they are needed. We present two different methods to facilitate this. The first requires additional memory external to the FPGA, while the second requires additional circuitry inside the FPGA.

In the first method, the routing tools would produce two separate maps of track interconnections for each

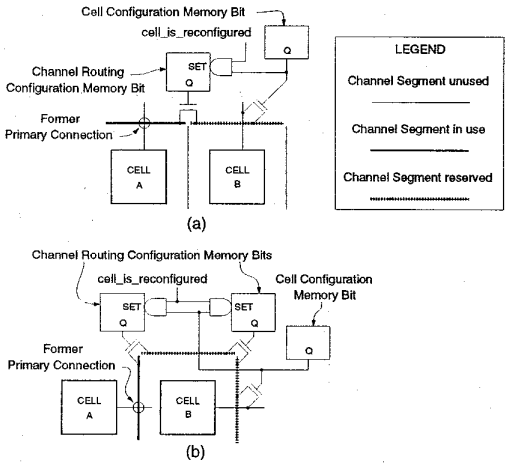

Figure 4: Automatic connection of reserved segments to a cover cell for reconfiguration. (a) Top cell-to-channel connection. (b) Right cell-to-channel connection.

channel row. The reconfiguration map would have all of the reserved segments connected, and the normal map would connect only those segments necessary for fault-free operation. Both maps would be loaded into the FPGA upon initialization, one after the other for each channel row. A faulty cell present in a row would cause the reconfiguration maps for the channels above and below that row to be selected for shifting into the channel configuration memory. Otherwise, the normal maps would be shifted in and the reconfiguration maps shunted aside. Only those nets affected by the reconfiguration would experience a propagation delay penalty, and it would be much less than if all of the reserved segments were connected. Only the track segment connection data needs to be duplicated, and is estimated to be about 30% of the total configuration data.

The second method makes use of the fact that there is a simple relationship between the locations of the cell-to-channel connections and the locations of the pass transistors in the segment-to-segment connection switches required to connect the reserved segments for a reconfigured cell. Circuitry associated with each cell would sense that the cell has been reconfigured as a cover and would force ON the appropriate pass transistors to connect cover segments as directed by the cell-to-channel connection data then in effect. This can be seen for a top cell-to-channel connection in Fig. 4a, where the output of the cell configuration memory bit controlling the cell-to-channel pass transistor also controls which channel routing configuration pass transistor to turn ON. Figure 4b applies to either a left or right cell connection, where it must be assured that two pass transistors are ON.

4 Yield Improvement

An important benefit of fault tolerance is an improvement in chip yield, where chip yield is defined as the percentage of usable chips. We present a comparison of our technique to the spare rows and/or columns techniques based on a set of common parameters. It is necessary in all of the techniques that channel wiring must remain fault-free in order to reconfigure around faulty cells. Wiring is also more robust than logic and so we will consider yield improvement based only on active cell area. A 16×16 array of cells is defined, to which will be added one spare row and/or column of cells or one spare cell per row. We use the Poisson yield model [1], in which defects are assumed to occur uniformly and independently. For each of the fault tolerant techniques, the yield of an array is computed as the sum of the probabilities of all usable configurations of defective and defect-free cells. These fault tolerant yields are plotted in Fig. 5 against the yield of the array without fault tolerance. It can be seen that the node-covering technique offers much greater yield improvement than using a spare column, although it has the same low area overhead in spare cells—about 6%. Also, it offers significantly more yield improvement than the technique of spare rows and columns, which has twice as much area overhead.

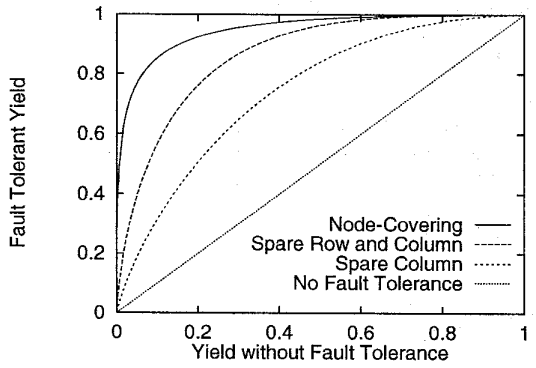

Figure 5: Fault tolerant yield comparison.

5 Overheads of Fault Tolerance

Although the potential of this FT technique for improving yield is significant, it can have an effect on the routability of a circuit. Some nets may have a high reserved channel segment overhead, and this may reduce the circuit's routability. In order to quantify this effect, we analyze the number of routing tracks required with and without our FT techniques.

5.1 Analysis Procedure

We have obtained FPGA routing software from the University of Toronto, and also several MCNC benchmark circuits whose netlists are in a format compatible with the routing tools. The software consists of a global router which assigns channels to the route of each point-to-point signal connection in a net, and a detailed router which chooses the actual track segments within the channels to be used by the nets. Thus, we are able to compare the number of tracks required in a non-FT circuit to the number after FT techniques have been applied.

We input the node covering specifications to the global router. However, since it is not known at that

MCNC benchmark	circuit nets	Without Track-hopping						With Uncovered Track-hopping						Segment Usage with Covered Track-hopping	
		segments used			tracks used			segments used			tracks used				
		non-FT	FT	ovhd. %	non-FT	FT	ovhd. %	non-FT	FT	ovhd. %	non-FT	FT	ovhd. %	FT	ovhd. %
too_large	186	1730	2332	35	14	24	71	1822	2604	43	12	17	42	2876	58
example2	205	1707	2190	28	21	24	14	1762	2379	35	19	19	0	2568	46
vda	225	2644	3408	29	17	26	53	2774	3802	37	16	20	25	4196	51
alu2	153	1453	2009	38	13	23	77	1542	2280	48	12	16	33	2551	65
alu4	256	3024	3944	30	17	28	65	3193	4424	39	16	18	12	4904	54
average				32			56			40			22		55

Table 1: Fault tolerance overheads for MCNC benchmark circuits.

time where the detailed router might perform a track-hop, it will be necessary to modify the detailed router to add the reserved segments necessary to cover the instances of track-hopping. Also, note that while the track-hopping technique results in efficient use of track resources, it inserts two additional pass transistors into the path of a signal traveling from one point-to-point path to the other, thus increasing the propagation delay. It is useful, therefore, to present results from our current version of the detailed router, which eliminates track-hopping and instead always requires contiguous track segments, although the requirement leads to greater track usage. We then estimate the track usage overhead expected when detailed router modifications to allow track-hop covering are completed.

5.2 Results

Table 1 compares non-FT circuits to the FT circuits routed to include cover segments. When track-hopping is not allowed, the average increase in tracks required for fault tolerance is 56%. However, we arrive at a reasonable estimate of the track overhead incurred for fault tolerance where track-hopping is allowed, as follows. Tests run with track-hopping allowed, but with no attempt to add the track-hopping reserved segments, show an average track usage increase of only 22% (due only to the non-track-hopping reserved segments). Segment usage in this case increases by an average of 40%, as seen in the table. If the additional number of segments needed to cover all of the track-hops observed in these tests were added, the total segment usage would be as shown in the final two columns of Table 1—a total increase of 55%. Extrapolating the track usage accordingly gives a track usage overhead of $(55/40) \times 22\% = 30\%$ that we should expect to see after completing modifications to the router to allow track-hop covering. This means that circuits mapped to FPGAs that use up to 77% ($1.3 \times 77\% = 100\%$) of the available tracks can be mapped to FPGAs with one cell defect per row (or be reconfigured around one faulty cell per row, in the case of operational faults). This should be quite tolerable, since most circuits do not use all of the tracks in an FPGA, anyway.

6 Conclusions

We have presented a fault tolerant technique to increase the reliability, availability, and yield of FPGAs. This technique makes use of a node-covering method which allows a cell in the array to be replaced by its neighbor if it becomes faulty. Reconfiguration is simplified by a routing discipline that provides the "hooks" for cell replacement by placing the necessary channel segments near the cover cells beforehand. Channel segments reserved for use in reconfiguration do not add extra parasitic delay to nets in a non-reconfigured array, since they are connected (automatically) only when needed. No rerouting is necessary in the event of a fault, and the configuration data for the faulty cell, including the connections to the channel wiring, is simply transposed to the cover cell. The simplicity of the reconfiguration method is an improvement over other techniques that require a new rerouting for each different faulty cell location. Our technique is also an improvement over row and/or column replacement done at the factory in that it can be done either at the factory or by the user. Additionally, it offers substantially greater yield improvement than either of those methods—for example, 10% greater than spare rows and columns and 50% greater than spare rows alone when non-FT yield is 30%.

References

[1] J. Cunningham, "The use and evaluation of yield models in integrated circuit manufacturing," *IEEE Trans. Semiconductor Mfg.*, Vol. 3, No. 2, pp. 60–71, May 1990.

[2] S. Dutt and J. P. Hayes, "Some practical issues in the design of fault-tolerant multiprocessors," *IEEE Trans. Comput.*, Vol. 41, pp. 588–598, May 1992.

[3] N. Hastie and R. Cliff, "The implementation of hardware subroutines on field programmable gate arrays," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 31.4.1–31.4.4, 1990.

[4] F. Hatori, *et al.*, "Introducing redundancy in field programmable gate arrays," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 7.1.1–7.1.4, 1993.

[5] N. Howard, A. Tyrrell, and N. Allinson, "The yield enhancement of field-programmable gate arrays," *IEEE Trans. VLSI Syst.*, Vol. 2, No. 1, pp. 115–123, March 1994.

[6] J. McDonald, B. Philhower, and H. Greub, "A fine grained, highly fault tolerant system based on WSI and FPGA technology," *Oxford 1991 Int. Workshop Field Programmable Logic Applicat.*, pp. 114–126 of *FPGAs*, W. Moore and W. Luk (eds.), Abingdon EE & CS Books, Abingdon, England, 1991.

[7] Xilinx, Inc., *The Programmable Logic Data Book*, 1994.