Effects of Technology Mapping on Fault Detection Coverage in Reprogrammable FPGAs

Kevin Kwiat, Rome Laboratory
Warren Debany, Rome Laboratory
Salim Hariri, Syracuse University

ABSTRACT

Although Field-Programmable Gate Arrays (FPGAs) are tested by their manufacturers prior to shipment, they are still susceptible to failures in the field. In this paper, test vectors generated for the emulated (i.e., mission) circuit are fault simulated on two different models: the original view of the circuit, and the design as it is mapped to the FPGA's logic cells. Faults in the cells and in the programming logic are considered. Experiments show that this commonly-used approach fails to detect most of the faults in the FPGA.

1 Introduction

Field-Programmable Gate Arrays (FPGAs) resemble traditional mask-programmed gate arrays, but differ in that they are programmed but by the end user. A typical FPGA consists of a two-dimensional array of logic blocks (or cells) that can be connected through interconnection resources. In some FPGAs a cell may be as simple as a 2-input NAND gate, while the cells of other FPGAs may be as complex as an entire PAL-like structure. This paper deals with dynamically reconfigurable FPGAs [1].

The starting point for designing an FPGA is logic entry. A schematic capture or logic synthesis tool is used to create a description of the circuit to be implemented. Then, the circuit design is translated into a standard form consisting of basic logic gates. This process of converting the netlist of basic logic gates into a netlist of FPGA cells is referred to as *technology mapping* [2].

For this paper, we define the following terms:

unmapped logic circuit A circuit design described as a netlist of basic logic gates. This is the target circuit that is to be emulated by the FPGA.

mapped logic circuit The same logic design as an unmapped logic circuit, but implemented as FPGA cells through technology mapping.

mission vectors The vectors that would be applied to the unmapped logic circuit. These vectors are applied to the corresponding inputs of a mapped logic circuit after the FPGA has been programmed. In the case where both circuits are fault-free, their output responses for any sequence of mission vectors are the same.

Manufacturers' documentation of reprogrammable FPGAs state that the devices have been tested with 100% fault coverage prior to their shipment (e.g., [1] [3] [4]). However, faults induced thereafter (i.e., field failures), must be tested for by the user. The manufacturer's test algorithm may be unavailable, but even if it were available, it may be impossible to apply it during board level test due to the FPGA's pin assignments. Thus, the most common approach is to generate test vectors for the unmapped circuit that the FPGA will embody. This paper describes the adverse effects that technology mapping has on this approach.

2 Determining Fault Detection Coverage

The procedure we use to measure the effects of technology mapping on fault detection coverage in a reprogrammable FPGA is as follows:

- *Step 1:* Create a logic model of the unmapped circuit.
- *Step 2:* Perform the technology mapping.
- *Step 3:* Create a logic model of the unprogrammed FPGA consisting of cells, cell interconnections, cell program memory, and FPGA programming logic.
- *Step 4:* Create the programming vectors that embed the mapped circuit into the FPGA model.
- *Step 5:* Obtain mission vectors that achieve the maximum fault coverage for the unmapped circuit, and produce lists of detected and undetected faults for the unmapped circuit.
- *Step 6:* Using vectors obtained from *Step 4* and *Step 5*, obtain FPGA simulation vectors.
- *Step 7:* Fault simulate the vectors from *Step 6* on the mapped circuit, and obtain lists of detected and undetected faults in the FPGA.
- *Step 8:* From the lists of detected and undetected faults for both circuit types, determine the discrepancies in fault coverage.

Next, an illustrative example of a full adder mapped to an FPGA provides a demonstration of this procedure.

3 FPGA

3.1 Cell Model and Programming

The FPGA implementation considered in this paper is based loosely on the Atmel architecture [1]. In these FPGAs, each cell can perform a combinational function, a sequential function (a

d-type flip-flop), or both. These cells can also be used as simple "wires" to connect cells over short distances. For fast communication over longer distances, buses run horizontally between rows of cells and vertically between columns of cells. Figure 1 depicts the cell architecture.

Several programming methods may be available for a given FPGA. However, a method common to all reprogrammable FPGAs is serial loading of programming data into the device, and this is the only programming method considered in this paper. Figure 2 shows the FPGA programming logic and a cell's program memory. Associated with each cell are two registers: a shift register to accept the serial data and a parallel-load register to hold the program data for the cell. A feature of Atmel's dynamically reconfigurable FPGAs is the ability to reprogram any cell without disturbing the rest of the array. Our FPGA model exhibits this feature by addressing each individual cell for programming.

3.2 Circuit Mapping for Fault Simulation

The first step in our procedure involves determining the logic diagram of the circuit under study. Figure 3 shows a full adder composed of basic logic gates. This represents how the designer might describe the circuit to be implemented in the FPGA. From this schematic, the logic model of the unmapped full adder is created.

Technology mapping is the next step in measuring the fault coverage detection. Figure 4 shows a mapping of the full adder description to the FPGA where the set of logic gates specified in the original schematic have been transformed to those available through cell programming. While placing the cells that implement the full adder, additional cells are allocated for signal routing. Cells that are not involved in the full adder implementation have their bus drivers disabled. In Figure 4, individual cells are identified by their row and column coordinates, indexed from 1, with *cell (1,1)* being the cell in the top left corner.

4 Simulation

4.1 Simulation Setup

Logic models of the unmapped and mapped circuits are needed prior to fault simulation. A Rome Laboratory-developed language, called the Netlist Intermediate Form (NIF) [5], was used to model both circuit types. A computer program was written to generate automatically the NIF description of the unprogrammed FPGA. A gate-level NIF model of the unmapped full adder was created and then translated to the Navy's Hierarchical Integrated Test Simulator (HITS) [6] language for fault simulation. The simulation platform was a VAX 8650.

To reduce both the model complexity and the simulation times, the unused cells were modified.

An unused cell can be greatly simplified because the only path for its outputs to the adder output is through the bus interface, which is disabled. As a result, a cell that was not used by the technology mapper and router was replaced by a *stub* of a full cell. All programming logic was removed from a stub and its architecture was reduced to only disabled bus drivers and a single gate that sinks all the cell's inputs and sources a constant output of zero. For the full adder the elapsed time for fault simulation of the mapped circuit (2,312 gates) exceeded 18 hours.

An exhaustive set of mission vectors achieved 100% fault detection of all single stuck-at-0 and stuck-at-1 faults in the unmapped circuit; this set comprised the mission vectors for our experiment. A vector sequence that programs the FPGA to implement the full adder was created (882 vectors), and the eight mission vectors were appended to this sequence. This sequence of 900 vectors was then fault simulated, and lists of detected and undetected faults in the FPGA were obtained.

4.2 Simulation Results

Fault grading was done in accordance with the standard procedure for fault coverage reporting (MIL-STD-883 Procedure 5012) [7] [8]. This procedure provides a consistent means of reporting fault coverage, regardless of the logic and fault simulator used. Exceptions to the baseline procedure were as follows: the fault universe was based on all faults on the signal lines, instead of fault equivalence classes of those faults; undetectable faults were not dropped from the fault universe; and faults in the logic that fed only the bus enables (there were 95 such faults in each cell) which are detectable only as potential detects were dropped from the fault universe.

The final step of the procedure calls for determining the fault coverage discrepancies between the two circuit types. Table 1 shows the fault coverages on a cell-by-cell basis, and for the programming logic; these values are repeated in Figure 4 where the corresponding function of each cell is also shown.

Unfortunately, when the constant zeroes sourced by the stubs are propagated through the logic model, these constants cause the fault simulator to reduce the fault universe, and as a result the number of faults per cell is not constant. Likewise, cells on the FPGA's periphery have some of their inputs tied to a constant value, and this accounts for the differences in the total number of simulatable faults for these cells. After adjusting the fault universe to account for these artifacts of fault simulation, only 17.5% of the faults in the entire FPGA were detected.

No FPGA faults were detected until the first mission vector was applied. Most faults in the programming logic are detectable only as potential detects. Unprogrammed cells that are intended to be programmed produce indeterminate values in the simulated circuit, so the fault originating the error is only potentially detectable. However, if a programming logic fault results in unmapped and mapped circuits that are decisively not functionally equivalent, then the fault is still detectable as a solid detect.

The technology mapping illustrated by Figure 4 is not unique. It is well-known that layout can influence testability (e.g., [9]), and a different arrangement of the stubs would produce different fault coverages for their neighboring cells. The *technology mapping* determines what programming input a cell can receive in the presence of a fault, so another technology mapping alters the circumstances of detecting the fault. Furthermore, a change in the *order* in which cells are programmed can also produce differences in fault detection.

The fault simulation results show that, when a set of mission vectors that is a complete test for an unmapped logic circuit is applied to a mapped circuit, the fault coverage is greatly reduced. To determine if the low fault coverage for the mapped circuit was due to intrinsic lack of testability of the FPGA, an experiment was performed to determine the achievable level of fault coverage for an FPGA cell. An FPGA cell (without the bus drivers) was modelled and a test of size 12 was obtained that detects all detectable faults in the combinational part of the cell. Next, this test set was modified so that whenever a vector is applied to the cell, the cell's flip-flop is clocked, resulting in 24 vectors. A vector was then added to test the flip-flop reset, making the sequence length 25. In the worst case, each of these 25 tests would require a unique cell programming. A total of 25 clock cycles (24 to shift in the data and one additional cycle to load the hold register), multiplied by the number of vectors, means that at most 625 test vectors would be required to detect all detectable faults in a single FPGA cell. Using this test sequence, the maximum achievable fault coverage for a cell is 95.884%. This coverage represents detection of faults in both the cell architecture and the cell's program memory. A cell is therefore intrinsically highly testable; however, the low fault coverage achievable using a complete test vector set for an unmapped circuit demonstrates the scope of the adverse effect technology mapping has on fault coverage.

5 Conclusion

The effectiveness of user tests applied to an FPGA depends on the cell architecture, the cell program memory, and the FPGA's programming logic. The approach shown in this paper can be applied to study other reprogrammable FPGA architectures. We have demonstrated that, when a gate-level design is mapped to a network of FPGA cells and tested using mission vectors developed for the original, unmapped gate-level design, the reduction in fault detection coverage is enormous.

References

[1] *Configurable Logic Design and Application Book*, Atmel Inc., San Jose, CA 1993.

[2] Brown, S.D., Francis, R.J., Rose, J., and Vranesic, Z.G.: 'Field-Programmable Gate Arrays', Kluwer, 1992.

[3] *The Programmable Gate Array Data Book*, Xilinx Inc., San Jose, CA 1992.

[4] *Optimized Reconfigurable Cell Array (ORCA) Data Book*, AT&T Microelectronics, Allentown, PA 1993.

[5] Debany, W.H., Lui, W., Kwiat, K.A., Sherman, K.J., Hayes, J.M., and Carletta, J.E.: 'Intermediate Form for Digital Model Transformation', *Proceedings of the IEEE Reliability and Maintainability Symposium*, 1986, pp. 11-16.

[6] Hosley, L., and Modi, M.: 'HITS – The Navy's New DATPG System', *AUTOTESTCON Proceedings*, 1983, pp. 29-35.

[7] Debany, W.H., Kwiat, K.A., Dussault, H.B., Gorniak, M.J., Macera, A.R., and Daskiewich, D.E.: 'Fault Coverage Measurement for Digital Microcircuits', Mil-Std-883 Procedure 5012, Rome Laboratory (RL/ERDA), Griffiss AFB, NY 13441, Dec. 18, 1989 (Notice 11) and July 27, 1990 (Notice 12).

[8] Debany, W.H., Kwiat, K.A., and Al-Arian, S.A.: 'A Method for Consistent Fault Coverage Reporting', *IEEE Design & Test of Computers*, Sept. 1993, **10**, (3), pp. 68-79.

[9] Spencer T.H., and Savir, J.: 'Layout Influences Testability', *IEEE Transactions on Computers*, March 1985, **C-34**, pp. 287-290.

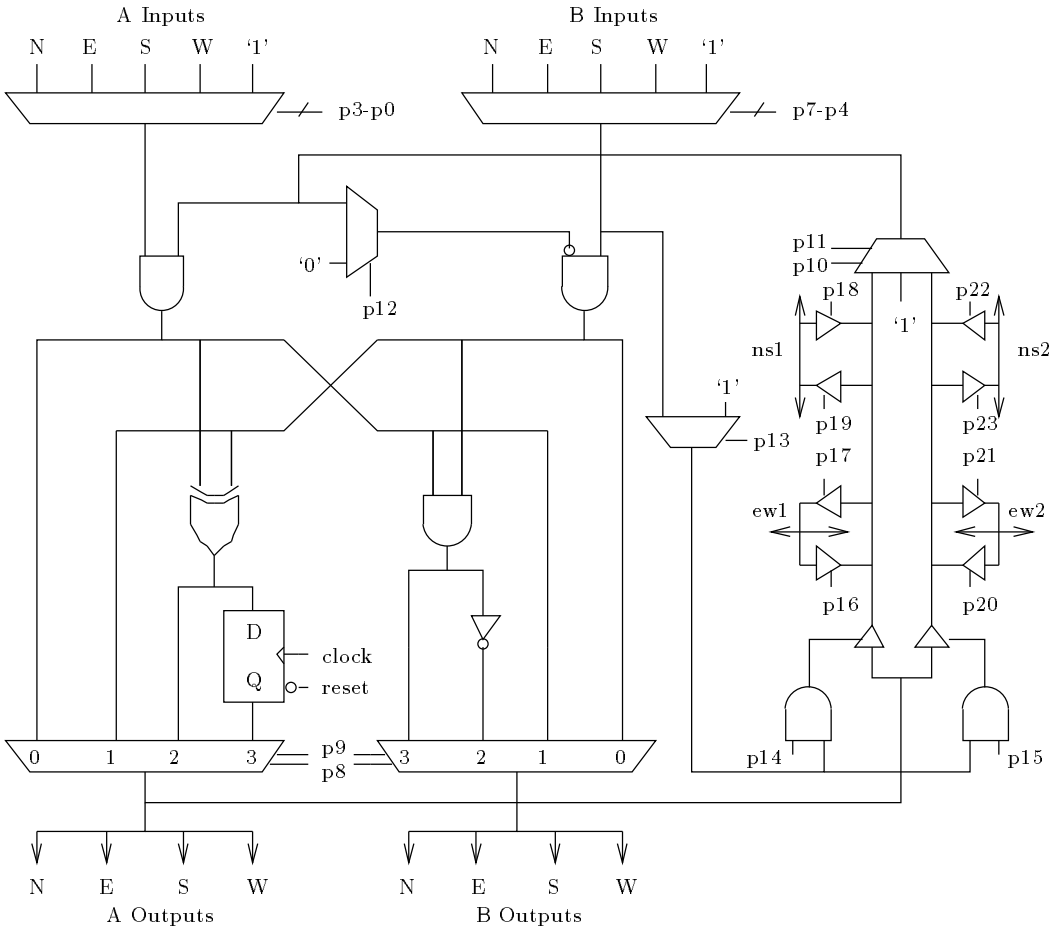

Figure 1: Cell Architecture

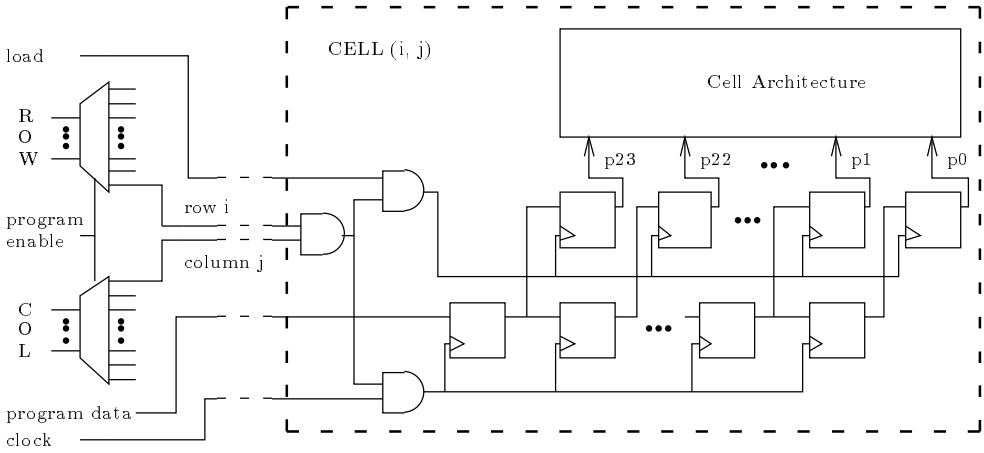

Figure 2: FPGA Programming Logic and Cell Program Memory

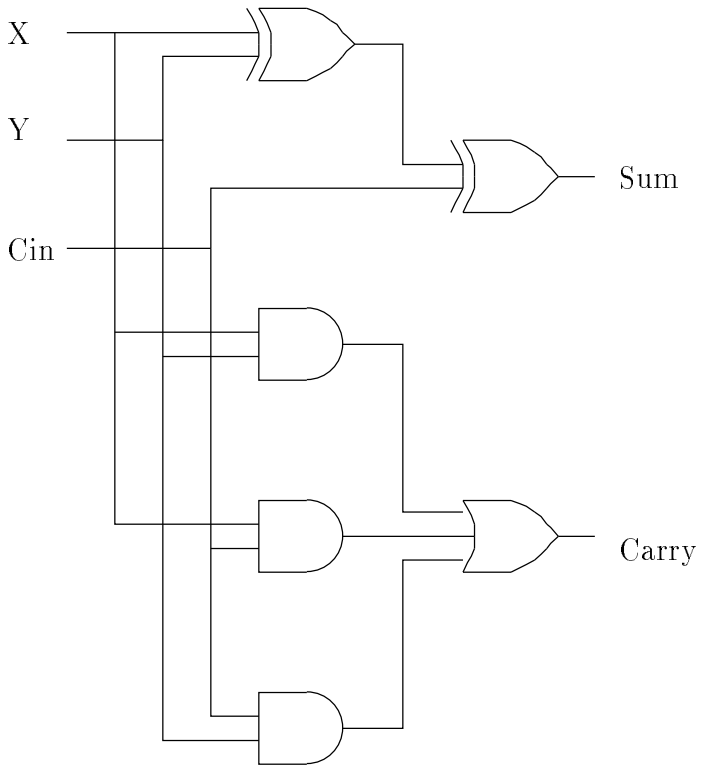

Figure 3: Unmapped Full Adder

FPGA Component	Faults Detected	Total Faults	Fault Coverage
cell (1,1)	152	711	24.7%
cell (1,2)	156	717	25.1%
cell (1,3)	144	703	23.7%
cell (1,4)	0	21	0.0%
cell (2,1)	110	709	17.9%
cell (2,2)	191	717	30.7%
cell (2,3)	0	21	0.0%
cell (2,4)	125	701	20.6%
cell (3,1)	0	21	0.0%
cell (3,2)	136	717	21.9%
cell (3,3)	221	717	35.5%
cell (3,4)	201	715	32.4%
cell (4,1)	115	709	18.7%
cell (4,2)	114	725	18.1%
cell (4,3)	222	717	35.7%
cell (4,4)	199	709	32.4%
cell (5,1)	161	709	26.2%
cell (5,2)	233	717	37.5%
cell (5,3)	0	21	0.0%
cell (5,4)	0	21	0.0%
cell (6,1)	0	21	0.0%
cell (6,2)	217	709	35.3%
cell (6,3)	232	707	37.9%
cell (6,4)	143	701	25.6%
prog logic	10	305	3.3%

Table 1: Fault Coverages

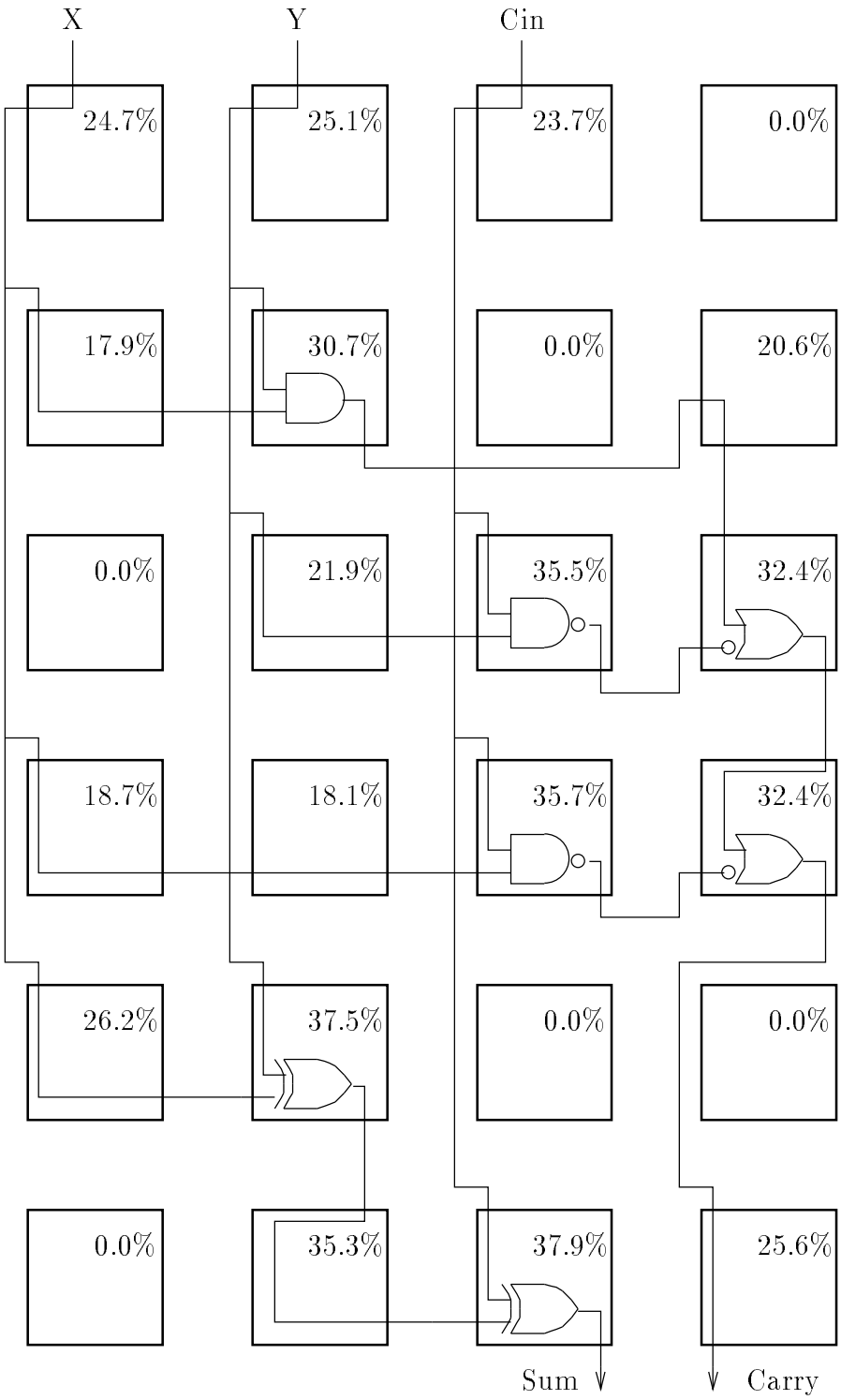

Figure 4: Mapped Full Adder. Percentages are individual cell fault coverages achieved by applying an exhaustive set of mission test vectors.