Enhancing Reliability of Operational Interconnections in FPGAs

Alex Fit-Florea, Miroslav Halas, and Fatih Kocan
CSE, Southern Methodist University
{alex, mhalas, kocan}@engr.smu.edu

Abstract

SRAM-based Field-Programmable Gate Arrays (FPGAs) have fixed numbers of wires, switches and look-up tables. An application does not fully utilize all available components in a FPGA, e.g. wires. In this paper, we propose methods to improve reliability of less reliable operational interconnections by efficiently utilizing unused wires to mask errors dynamically. With these methods, we are able to improve the reliability of more than two-thirds of all interconnections in the studied MCNC benchmarks. As a result, the overall unreliability of operational interconnections decreases more than 20%.

1. Introduction

FPGAs are efficiently used in prototyping, logic emulation systems, and low and high volume applications. The critical systems on FPGAs require fault free devices and/or fault/error tolerable designs and implementations. Several fault tolerance methods are used: the physical design is partitioned into a set of tiles each with multiple configurations [3], complete module redundancy and majority voting (TMR) [2], alternate configurations are generated at design time and stored in memory [4], etc. Our approach employs a masking-based fault tolerance strategy based on static triple interconnection redundancy (TIR) that dynamically corrects errors generated on these interconnections. To implement the concept, we employ a new switch design with majority voter (MV) capabilities. As a result all single and some multiple stuck-at and signal integrity faults can be tolerated.

2. Graph model for FPGAs

A FPGA LUT-to-LUT connection (L-L path) is a succession of type: [LUT, Wire,] {Switch, Wire, ..., Switch, Wire,}[LUT]. For example, in Figure 1, a L-L path is $P = (L_4 - S_4 - S_3 - S_2 - S_7 - L_5)$, where L_4 and L_5 are LUTs and S_2, S_3, S_4, S_7 are switches. We use a graph model: each switch becomes a node and an unused wire between switches S_i and S_{i+1} becomes an edge between the same corresponding nodes. Figure 1 depicts the graph (b) associated with a FPGA implementation (a); thin lines represent

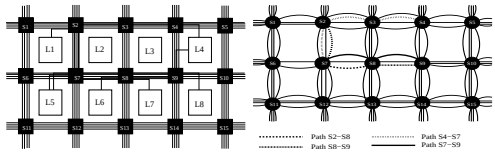

Figure 1. a. FPGA Implementation; b. Graph

available edges and existing L-L paths are included as emphasized lines. After assigning the circuit's L-L paths (for example using VPR [1]) only few wires will still be available for contributing toward improving the reliability via TIR. Finding these improvements is equivalent to finding edge independent paths in the corresponding graph based model.

3. Switch with MV logic

A new switch logic comprising logic splitter and majority voter capabilities is needed in order to achieve desired TIR functionality. A suggested solution is (Figure 2) an any-to-any switch with simple MV logic at each terminal. It con-

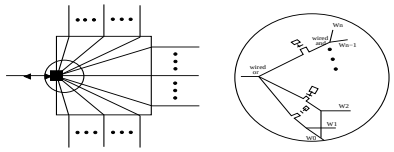

Figure 2. Switch with majority voter (MV)

sists of n+1 input wires $W_0, W_1, \cdots W_n$, a *wired-and* for any pair of wires, SRAM cell switch for each *wired-and* result, and a *wired-or* of all *anded* results. At any given time - through the use of SRAM cells - three *wired-and* results will be set as inputs to the *wired-or*. This way if 2 out of 3 inputs are 0(1), the result is 0(1).

4. Algorithms for Reliability Improvement

We adopt the combinatorial model in examining the basic definitions and mathematics of reliability evaluation. Figure 3 depicts the basic cases associated with TIR: a) path is unchanged, b) the whole path is tripled, or c) only a subpath is tripled. Corresponding reliability equations are: (1)

1530-1591/04 $20.00 (c) 2004 IEEE

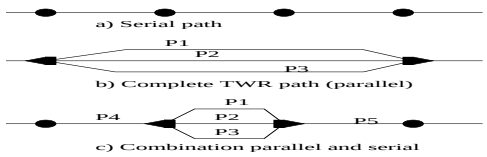

Figure 3. Basic TIR cases

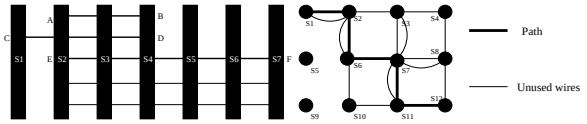

Figure 4. a) Overlapping Paths, b) Algorithm II Example

$R(P) = \rho^n \sigma_0^{n-1}$, (2) $R(P) = (R_1(R_2 + R_3 - R_2R_3) + (1 - R_1)R_2R_3)\sigma^2$, (3) $R(P) = R_4R_5(R_1(R_2 + R_3 - R_2R_3) + (1 - R_1)R_2R_3)\sigma^2$. Here ρ represents reliability of a S-S wire, σ_0 - original switch and its connections, σ - new switch, and R_i - reliability of sub-path P_i, respectively. Formulae (1), (2) and (3) could be used to recursively compute the reliability of any L-L path as a function of ρ, σ_0, and σ.

In respect with the original L-L path, the redundant wires used to improve reliability of the connections between switch boxes can be: (I) along the original path for the entire length of the path (on parallel tracks) (e.g. path $S_4 - S_3 - S_2 - S_7$ in Figure 1 b), or (II) on non-parallel tracks to the original path for a portion of the path. In case

Improve reliability of every connection between two switch-boxes which does not overlap with any other connection (e.g. $S_5 - S_6$ in Figure 4 a).
while there are improvable segments **do**
  Determine a improvable path P_w having worst reliability.
  Determine a path P_{bw} with best reliability among paths overlapping with P_w.
  Improve reliability of P_w using an improvable segment where P_w and P_{bw} overlap.
  Update reliabilities R_w, R_{bw}.
od

Figure 5. Algorithm I

(I) we use a set of greedy priority guidelines whose sketch is presented in algorithmic steps in Figure 5. Considering the more general case (II), in Figure 4 b), a path P exists between S_1 and S_{12}: $S_1 - S_2 - S_6 - S_7 - S_{11} - S_{12}$. Its sub-paths are $S_1 - S_2$, $S_1 - S_6$, ..., $S_1 - S_{12}$, $S_2 - S_6$, ..., $S_2 - S_{12}$, ..., $S_{11} - S_{12}$. Determining whether a sub-path SP with ends S_1 and S_2 is improvable or not can be done in two steps: find a shortest path SP_1 between S_1-S_2 and, if exists, remove it from the graph. This gives a 1^{st} edge-independent path with same ends as SP. If a 2^{nd} path $S_1 - S_2$ can be found (for example with a BFS-like algorithm), then P's reliability can be improved. The sketch of this algorithm is presented in Figure 6. To accommodate propagation delay

while there are possible improvements **do**
  Determine the set SI of sub-paths that can be improved
  for each element SP_i of SI **do**
    Compute score $\mathcal{S}(SP_i)$ of SP_i
  od
  Perform improvement for best computed score
  Remove from graph edges used to perform improvement.
od

Figure 6. Algorithm II

due to our changes, we can identify a maximum allowable number of tripled segments and/or use a slower clock cycle.

5. Experimental results

We assess the performance of our methods in terms of the improvement in the overall reliability measured as percentage of un-reliability decrease. As benchmark we used circuits from the Microelectronics Center of North Carolina (MCNC) circuit benchmark suite [5]. The input used by our algorithms is the one provided by running VPR. Table 1. presents the performance of algorithm corresponding to case (I).

Benchmark circuit	Percentage of improved paths	Decrease in unreliability	nr. new switches
alu4	64.3305%	21.8776%	615
apex2	74.8271%	19.4051%	1082
bigkey	61.7039%	35.6015%	746
clma	72.1532%	18.6278%	4353
des	69.4531%	25.0913%	889

Table 1. Benchmark Results

6. Conclusions

We proposed methods to improve reliability of less reliable operational interconnections by efficiently utilizing unused wires to mask errors dynamically. With these methods, we are able to improve the reliability of more than two-thirds of all interconnections in the studied MCNC benchmarks. As a result, the overall unreliability of operational interconnections decreases on the average more than 20%.

References

[1] V. Betz and J. Rose, *VPR: A New Packaging, Placement and Routing Tool for FPGA Research*, 7th International Workshop on Field-Programmable Logic, London, August 1997, pp. 213-222.

[2] C. Carmichael, *Triple Module Redundancy Design Techniques for Virtex Series FPGA*, Xilinx Application Notes 197, v1.0, Mar. 2001

[3] J. Lach, W. Mangione-Smith and M. Potkonjak, *Low Overhead Fault-Tolerant FPGA Systems*, IEEE Transactions on VLSI Systems, Vol. 6, No. 2, pp. 212-221, 1998.

[4] J. Lach, W.H. Mangione-Smith and M. Potkonjak, *Enhanced FPGA Reliability Through Efficient Runtime Fault Recovery*, IEEE Transactions on Reliability, 49(3):296-304, September 2000.

[5] S. Yang, *Logic Synthesis and Optimization Benchmarks, Version 3.0*, Tech. Report, Microelectronics Center of North Carolina, 1991.