HAFT: A Hybrid FPGA with Amorphous and Fault-Tolerant Architecture

Mingjie Lin, Steve Ferguson, Yaling Ma, and Timothy Greene

***Abstract*— We propose a hybrid FPGA architecture with a dense and defective nano-crossbar serving as its configuration memory. An *amorphous* routing architecture is adopted to optimally allocate logic and routing resource on per-mapping basis and to achieve high logic density. This hybrid FPGA is designed to be efficient in using nano-crosspoints, highly tolerant to memory defects, and versatile to provide features such as variable-granularity logic blocks and variable-length bypassing interconnects. A new placement algorithm and a modified delay-based routing procedure are designed to match with many unconventional architectural features of the proposed FPGA. Assuming zero defect-rate in the nano-crossbar, an FPGA with the proposed architecture can achieve a 30% improvement in logic density, 12% improvement in average net delay, and 8% improvement in the critical-path delay for the largest 20 MCNC benchmark circuits over an island-style baseline with the same nano-scale memory. As the rate of defects in the memory increases from 0% to 50%, this hybrid FPGA remains fully functional and its improvement in logic density and delay performance only drops by approximately 23%.**

I. Introduction

Aggressive scaling in modern CMOS technology has resulted in many challenges in IC design and fabrication. It is now widely believed that a paradigm shift from the planar lithographic process may be needed. Several alternatives to standard Si-based CMOS devices have been proposed, including single-electron transistors, quantum cellular automata, and molecular logic devices. A common theme that underlies many of these schemes is the drive to fabricate nano-scale logic devices. Unfortunately, due to the statistical yields of the chemical syntheses, a system built solely with nano-scale devices will inevitably suffer unreliable connectivity and nonoperational devices. To overcome these challenges and construct an efficient and error-free computational system, researchers have proposed many hybrid architectures that integrate nanowire crossbars with a CMOS chip [1], [2], [3]. Figure 1 depicts a typical nanowire crossbar, where each crosspoint is electrically configurable or reconfigurable. This nanowire crossbar can be electrically attached to a CMOS chip via some metallic pins on its surface as in Figure 2(b). Based on such nanowire crossbar, Dehon *et al* [1] proposed a nanowire-based programmable architecture that offers one to two orders of magnitude greater mapped-logic density than defect-free lithographic FPGAs at 22nm. More recently, Snider *et al* [3] proposed a field-programmable nanowire interconnect (FPNI) that enables a family of hybrid nano/CMOS circuit architectures, which generalizes the CMOL (CMOS/molecular hybrid) approach proposed by Strukov *et al* [2] and allows for simpler fabrication, more conservative process parameters, and greater flexibility in the choice of nanoscale devices.

The authors are with the Electrical Engineering Department at Stanford University. (Email: mingjie@stanford.edu).

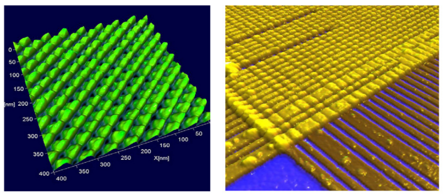


Fig. 1. Atomic force microscope topographs of a nearly defect-free region (left) [3] and highly defective region (right) in a 34 nm pitch nanowire crossbar [4].

Our proposed HAFT architecture differs from [1], [2], [3] in that nano-crossbar is only used as configuration memory and the routing architecture of HAFT is *amorphous* and *fault-tolerant*. Two major motivations are behind the HAFT architecture. The first one is to reduce the significant cost paid for routing in standard island-style FPGAs and translate the saving in hardware usage into performance gain. The second motivation is to explore how to fully utilize high-density memory cells provided by the nano-crossbar in an efficient and reliable way. The central idea of the amorphous routing architecture in HAFT is to make several architectural choices in an FPGA dynamically configurable on a per-mapping basis at configuration time. The concept of this architectural "shapelessness" is illustrated in Figure 2. While in the conventional island-style architecture, there is a strong separation between logic and routing resources, and this resource partition is fixed after the chip fabrication, the amorphous FPGA allows the dynamic resource partition at configuration time. As will be shown in Section II, the special design of ROLE blocks employs large number of configuration memory cells that would be prohibitively expensive should regular SRAM memory is adopted. More importantly, the design of ROLE blocks can tolerate large percentage of configuration memory cells to be defective and remains fully functional, which is critical in using nano-crossbar. Finally, the HAFT architecture can support (i) *dynamic resource allocation* between logic and routing, (ii) *variable-granularity logic blocks*, (iii) *variable-length interconnect overlay* without passing through switching points. All these features contribute to the significant performance gain of HAFT architecture reported in Section IV.

In the following section, we describe the HAFT FPGA architecture in detail. Before comparing performance and investigating the effectiveness of the proposed HAFT architecture to combat device defects in Section IV, we present our placement and routing algorithms in Section III. Finally in Section V, we summarize our findings and comment on some unsolved research problems.

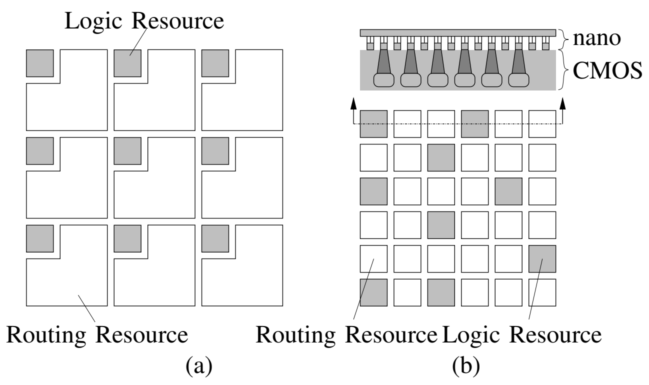


Fig. 2. Conceptual picture of FPGA architectures: (a) Conventional island-style FPGA, (b) A HAFT FPGA.

II. The HAFT Architecture

The top-level architecture of a HAFT FPGA consists of an array of Routing or Logic Element (ROLE) blocks with horizontal and vertical routing channel overlay on the top (Figure 3(a)). Different from the conventional island-style FPGA, the amorphous architecture in a HAFT FPGA replaces logic blocks with specially designed ROLE blocks that allow dynamically partitioning hardware resource between logic and routing on a per-mapping basis after chip-fabrication. Each ROLE block is capable of performing logic only, routing only, or the combination of both tasks.

Routing or Logic Element (ROLE)

As shown in Figure 3(b), a typical ROLE block is constructed with a number of MUTs (Multiplexer or look-Up Table), each of which contains three types of functional structures: a 4-input look-up table (4-LUT), a flip-flop register, and a MUX as depicted in Figure 3(c). A ROLE block can be configured into different types of functional blocks. If all 6-MUTs are used as 6-MUXes, then the whole ROLE will behave like a routing block. In contrast, if we use all 6-MUTs as combinations of 4-LUTs and their associated FFs, then the whole ROLE can be looked as a typical logic block with four 4-LUTs. Alternatively, we can partially use 6-MUTs and use the ROLE block as a hybrid of a routing block and a logic block with smaller number of 4-LUTs. All configuration memory bits contained in MUTs and MUXes in a ROLE block are implemented with nano-crosspoints on the top layer. Fault-tolerance comes from the fact that if some bits in a LUT are defective, the corresponded 6-MUT can be used as a 6-MUX. Since in a placed and routed design, only a few of 6-MUTs in each ROLE block will be used as logic blocks and the majority of ROLE blocks will only function as routing blocks, we anticipate this design can tolerate quite some defective nano-crosspoints.

Interconnect Overlay

In modern FPGAs, the routing fabric not only contributes the most to the system delay but also consumes most of the chip area [5]. To make the situation even worse, the fraction of total delay due to routing in an FPGA is increasing with each process generation [6]. Consequently, an FPGA architect must devise routing architectures that are both fast and area-efficient in order to fully exploit the performance and density potential of deep-submicron technologies. It is known that interconnect segment length can significantly impact the overall performance of an FPGA. Short connections, in general, are advantageous for routing but often result in very densely placed areas where clouds of highly connected logic are placed and cause regional routing problems. Additionally, using short interconnects tends to degrade the overall delay and power performance. On the contrary, long interconnect benefits power and delay but often hurts the routability. Previous studies have shown that optimizing segmentation is quite challenging. We took a different approach and developed a new *bypassing* interconnect for the HAFT FPGA. The key idea is to construct long interconnects without passing through heavy switching points and therefore improve delay and power performance without sacrificing routability. One additional benefit of more regular structure in bypassing interconnects is that the delay of long interconnects in the HAFT FPGA is much more predictable than that in a conventional segmented routing architecture, and therefore makes it easier for the CAD software to find the most optimal routes.

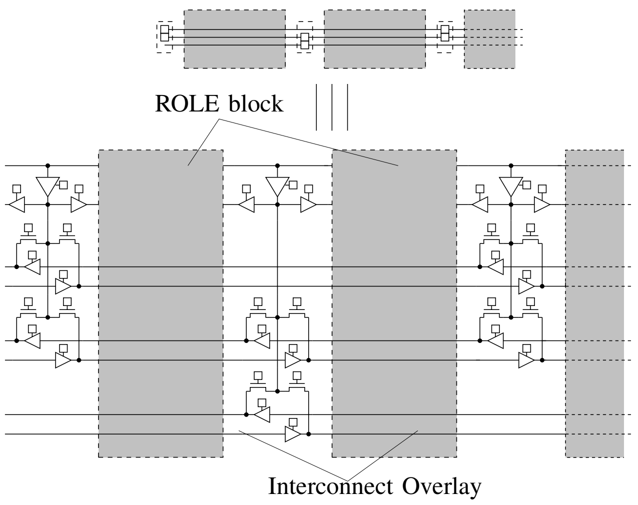


Fig. 4. Interconnect overlay and its logic design.

As shown in Figure 4, each routing channel comprises only Single and Double segments. Each Single or Double segment consists of two unidirectional wires controlled by two tri-state buffers. Segments can be connected directly to form longer interconnect segments by appropriately setting the states of the tri-state buffers without entering ROLE blocks. This helps reducing the parasitic loading along interconnects due to programming overhead. The segments can also be connected through routing blocks to make bends, fan-out, or connect to logic blocks.

III. CAD Software

In order to make meaningful FPGA architecture comparison, it is essential that the CAD tools used to place and map circuits into each architecture are of high quality. The routing phase of the HAFT FPGA are largely based on the previously published VPR [7]. The main difference between our router and VPR is that we keep track of visited nodes during the breadth-first-search to improve the run time. More details of this routing algorithm can be found in [6]. Because the logic and routing resource has to be partitioned before routing stage,

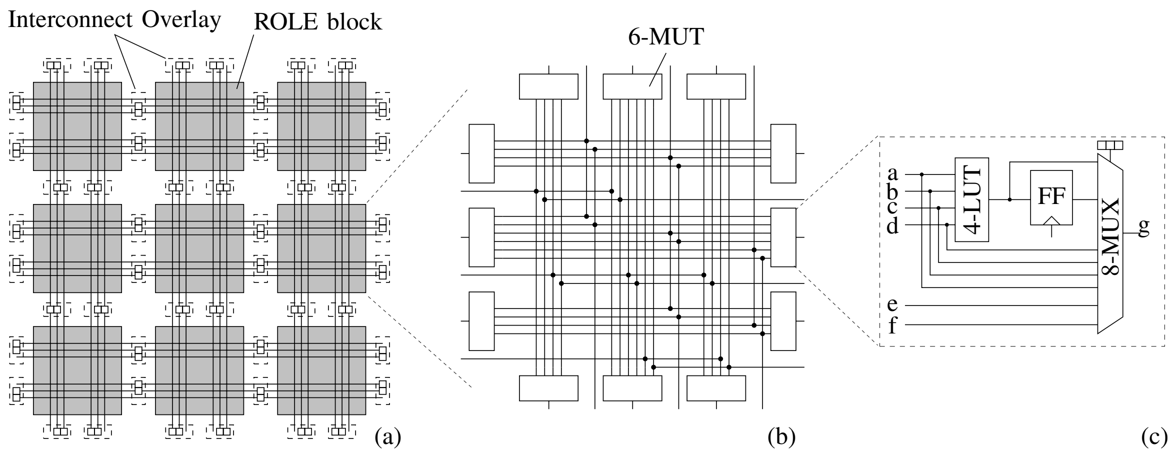


Fig. 3. (a)Top-level diagram of an amorphous FPGA architecture.(b) A small example of ROLE block. (c) Structure of a MUT.

we develop a new algorithm to solve the placement problem for the HAFT FPGA. Given a target design circuit, we start with logic packing assuming homogeneous logic blocks and the same technology packing procedure as in VPR.

Our placement software for the HAFT FPGA is based on a simulated annealing approach. The placement procedure of the HAFT architecture differs from the island-style FPGA in a fundamental way: its successful routing relies heavily on configuring some ROLE blocks as routing blocks and allocating them to match the routing demand of other RLB blocks configured as logic blocks in the array. To achieve high logic density, the routing resource in a HAFT FPGA should be distributed non-uniformly based on the routing demands between each pair of logic blocks, which is depicted in Figure 2(b). As a result, all ROLE blocks must be configured and positioned as part of the placement process. Given a $N \times N$ array of ROLE blocks, we start with a random placement of ROLE blocks configured as LBs. The corresponding routing graph is then generated and the target design is routed using the standard solution of the Euclidean Steiner tree problem. An initial temperature for simulated annealing is set. We then randomly pick a pair of ROLE blocks and swap them. The design is then rerouted using Steiner tree algorithm, and the cost metric is re-evaluated. If the metric value is reduced, the swapping is accepted, otherwise it is accepted with a probability that depends on the increase in the value of the cost metric and temperature. The process of ROLE swapping, computing its cost metric, and accepting or rejecting it is repeated until the inner-loop criterion becomes false. After exiting the inner loop, temperature is reduced and the process is repeated until the exit criterion becomes false.

The key ingredient of out placement algorithm is the cost function especially designed to match the routing fabric of the HAFT FPGA. Given a placement of ROLE blocks and the benchmark design, the first step of evaluating the cost is to perform the routing using the Steiner tree algorithm. It is well-known that most Steiner tree problems are NP-complete, i.e., thought to be computationally hard. To simplify our placement algorithm and to combat the intractability, we use a heuristic [8], in which we compute the Euclidean minimum spanning tree to approximate the Euclidean Steiner

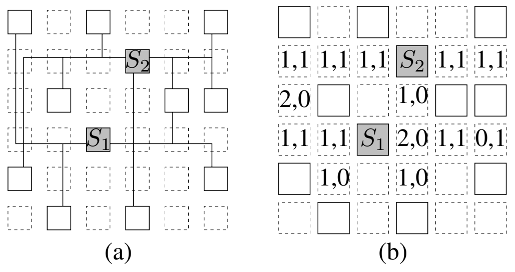


Fig. 5. (a) Two routed nets using the minimum-distance Steiner tree algorithm. (b) Computed $((x_{i,j}, y_{i,j}))$ at each block after two nets are routed.

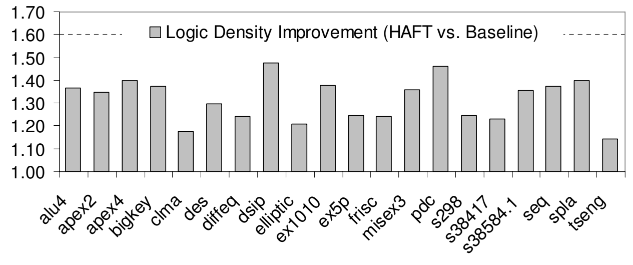
tree problem.

Figure 5(a) shows an example with two nets routed with the Steiner tree algorithm. Let $(x_{i,j}, y_{i,j})$ be the cost pair for a ROLE block located at (i, j), where $x_{i,j}$ is the total number of routed signals along both x- and y-directions above the ROLE block (i, j) and $y_{i,j}$ is the total number of routed signal turns above the ROLE (i, j) defined by $\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} x_{i,j}$ and $\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} y_{i,j}$ respectively. The cost of placement, c, is computed by $\sqrt{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} x_{i,j}^2 - \bar{x}^2} + \sqrt{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} y_{i,j}^2 - \bar{y}^2}$. The value of cost c roughly reflects the variance of the demand for routing resource from all ROLE blocks. Intuitively, small c means high placement quality in terms of the overall routability.

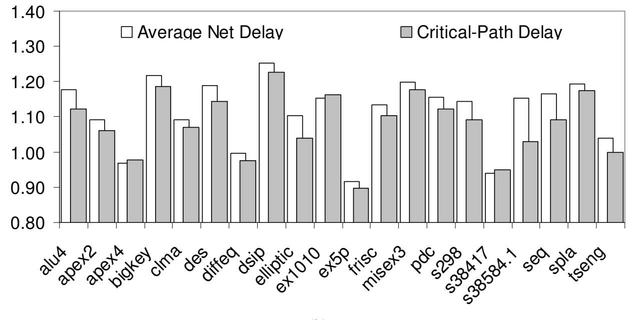
IV. Performance & Fault-Tolerance Study

In this section we compare an FPGA with HAFT architecture to the baseline FPGA described in [6] in terms of routability and delay. We assume a 65nm CMOS technology and the Berkeley Predictive Technology Model (BPTM) [9] for devices and interconnects. To make the comparison fair, we replace all configuration memory cells in the baseline FPGA with nano-crosspoints as in a HAFT FPGA.

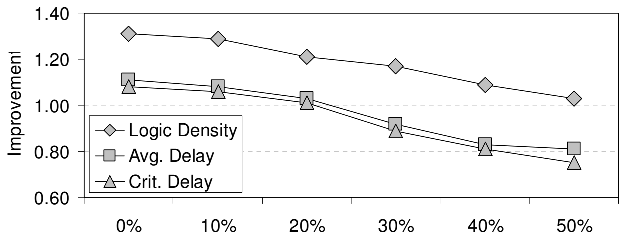
For routability comparison, we use the minimum required silicon area to successfully route a benchmark circuit for both architectures. For the baseline FPGA, we choose the number of LBs according to the size of benchmark circuit, vary the routing channel width, find the minimum track count for each design, and then apply the area model developed in [6]. For the HAFT architecture, we fix the routing channel width and find



(a)



(b)



(c)

Fig. 6. (a) Logic density improvement, (b) delay improvement, (c) performance improvement vs. the memory defect rate between HAFT and baseline FPGA for 20 MCNC benchmark circuits.

the minimum number of ROLE blocks in order to successfully place and route a target design. In Figure 6(a), we plot the logic density improvement of the HAFT FPGA over the baseline FPGA for 20 MCNC benchmark circuits. By logic density improvement we mean the ratio of the minimum required FPGA area between the baseline and the HAFT FPGA in order to successfully route each benchmark circuit. Note that on average, the new HAFT FPGA requires around 35% less silicon area than the baseline FPGA. The reduction of silicon area is due to several factors. First, the use of shorter segments improves the routability. Second, ROLE blocks are capable of being configured as either logic or routing block and therefore can dynamically adopt to the demand of routing resource between ROLE blocks configured as logic blocks.

For delay comparison, we use the geometric average of the pin-to-pin delays defined as the geometric average of all its pin-to-pin net delays and the critical-path delay computed with the same approach as in [6]. The improvement is defined as the ratio of the delay in the baseline FPGA to that in the HAFT FPGA. Results for the largest 20 MCNC benchmark circuits are plotted in Figure 6(b). Note that the improvements over the baseline FPGA range from 0.92 times to 1.25 times for the geometric average pin-to-pin delay and from 0.90 times to 1.23 times for the critical-path delay.

To study the effectiveness of fault-tolerant design in HAFT architecture, we investigate the relationship between the defect rate of nano-crosspoints and the performance degradation. We randomly set a percentage of nano-crosspoints to be nonfunctional and then compute the average performance improvement of the HAFT over the baseline FPGA for all 20 benchmark circuits. Figure 6(c) shows that as the defect rate increases from 10% to 50%, the logic density and delay improvements only decreases by only about 23% and in all cases, the HAFT FPGA is fully functional.

V. Conclusion

Integrating nano-crossbar with CMOS devices, hybrid chip can be a viable solution to several IC design and fabrication challenges due to technology scaling. The HAFT FPGA offers an interesting architecture to not only efficiently utilize high density nano-crosspoints but also address the issue of large defect rate in a nano-memory. The performance gain and fault-tolerant capability of a HAFT FPGA are significant when compared with the conventional island-style FPGA.

At least two open research problems remain. First, it is conceivable that better placement algorithms and cost metrics may exist. Secondly, although the HAFT architecture supports variable-granularity logic blocks, in this study, we have not unleash this potential. How to optimally technology-pack a design circuit into logic blocks with different size and further take advantage of the amorphous routing architecture is an open question.

References

[1] A. Dehon, "Nanowire-based programmable architectures," *J. Emerg. Technol. Comput. Syst.*, vol. 1, no. 2, pp. 109–162, 2005.

[2] D. B. Strukov and K. K. Likharev, "A reconfigurable architecture for hybrid CMOS/Nanodevice circuits," in *Proceedings of the 14th international symposium on Field programmable gate arrays*, pp. 131–140, 2006.

[3] G. S. Snider and R. S. Williams, "Nano/CMOS architectures using a field-programmable nanowire interconnect," *Nanotechnology*, vol. 18, pp. 1–11, 2007.

[4] G.-Y. Jung, E. Johnston-Halperin, W. Wu, Z. Yu, S.-Y. Wang, W. Tong, Z. Li, J. Green, B. Sheriff, A. Boukai, Y. Bunimovich, J. Heath, and R. Williams, "Circuit fabrication at 17 nm half-pitch by nanoimprint lithography," *Nano Letters*, vol. 6, no. 3, pp. 351–354, 2006.

[5] V. Betz and J. Rose, "FPGA routing architecture: segmentation and buffering to optimize speed and density," in *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field-Programmable Gate Arrays*, pp. 59 – 68, 1999.

[6] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-D FPGA," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, pp. 216–229, Feb. 2007.

[7] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, pp. 213 – 222, 1997.

[8] A. Kahng and G. Robins, "A new class of iterative steiner tree heuristics with good performance," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 11, pp. 893–902, July 1992.

[9] Y. Cao, T. Sato, M. Orshansky, D. Sylvester, and C. Hu, "New paradigm of predictive mosfet and interconnect modeling for early circuit simulation," in *Custom Integrated Circuits Conference, 2000. CICC. Proceedings of the IEEE 2000*, pp. 201–204, 2000.