Mapping Circuits to Self-Assembled Networks

Chen-Ling Cheney Tsai

under the supervision of Dr. Alvin R. Lebeck,

Duke Computer Science Department, Duke University

May, 2011

_____

Research Supervisor

_____

Faculty Reader

_____

Faculty Reader

Honors Thesis submitted in partial fulfillment of the requirements for graduation with Distinction in Computer Science in Trinity College of Duke University.

Abstract

Self-organizing nano-scale irregular networks present unique placement and routing challenges. With increasing capabilities in the scale and control of these technologies, we have the possibility of creating a high-performance defect-tolerant architecture that is comparable in performance to existing architectures. This research explores the possibilities of placing some desired logic circuit onto these irregular networks. We look at the issues of connectivity and how differing levels of randomness and control can increase or decrease the likelihood of a successful mapping. We also present the possibility of modifying current toolchains to map circuits onto our custom architecture.

1. Introduction

Recent research in novel nanotechnologies has led to the generation of resource-constrained self-organizing nano-scale networks. While this may allow for smaller scale integration, architectures at this level have to deal with higher defective rates and other irregularities.[1] Given certain parameters to generate these networks, we aim to explore the possibility of mapping circuit netlists onto these irregular, highly defective structures.

Irregular networks can be difficult to approach using current routing and placement algorithms designed for organized, minimally defective structures. Because map computation and data placement knowledge can be complex, it is presently difficult to grasp the parameters that need to be optimized for a successful mapping. These parameters include factors such as node number, density, fanouts, and ability to sever links between nodes. [1]

This research focuses on the effects of variability in these mentioned variables and more. Furthermore, progress was made to determine if current toolchains such as Quartus, the RASP package developed by the University of Berkeley, the open VPR tool developed by the University of Toronto can be modified to simulate the circuit mapping.

While previous work has shown that “It is possible to design a high-performance defect-tolerant architecture that can match or even outperform existing architectures …” this undergraduate thesis aims to explore the possibility of building on top of this architecture and mapping to some random topology some specific desired functionality.[2]

2. Background

2.1 DNA Self-Assembled Systems – The Physical Model

This research explores the possibilities based upon advancements in DNA self-assembled systems. We assume an assembly process to replicate a simple circuit cell consisting of a transistor in the cavity of a DNA-lattice. Self-assembly can scale to some 10^{12} node arbitrary network which are randomly linked through one bit channels. This system allows for communication with external CMOS circuitry through a metal junction. [2]

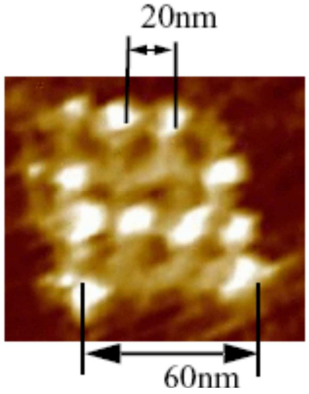

Figure 1 - DNA Self-Assembled Structure

No control mechanisms are used for the placement and orientation of these nodes and randomness is apparent on all levels of topology generation. This includes the possibility that a interconnect of 2 nodes may fuse with the interconnect of another 2, forming a local connectivity mesh of size 4. How this is dealt with is crucial in determining the possibility of success of a circuit mapping.[2]

The physical structure aims to provide the technology that a computer architecture such as SOSA can be developed on. Several factors can influence the overall possibility of successful computation on such a physical circuit. High defect rates will require placement and routing algorithms to identify and map around defective nodes and links. Unlike a structured grid laid out in a Field Programmable Gate-Array (FPGA), random interconnections between nodes result in the formation of a hypergraph, where interconnects/edges can fuse and link to any number of nodes. This greatly increases the computational cycles required to traverse such a graph. Nodes are only able to support

simple logic functions, thus requiring technology mapped net-lists be expanded. While the strength of the technology arises in the ability to link together billions of computational nodes to perform sophisticated computation, there are also challenges in determining how to utilize the sheer number of resources to output some desired function.

2.2 SOSA – Self Organizing SIMD Architecture

As described in the previous section, SOSA is a defect-tolerant self-organizing nano-scale SIMD architecture aiming to build a high performance defect-tolerant computing system using randomly assembled nodes. [2]

A node is completely asynchronous and contains a 1-bit ALU, a small 32-bit register file, a data buffer, and connects to other nodes with up to 4 other single wire links. For the purposes of this particular research, we use the given functionality as a upper limit and sometimes simply functionality to gate-level for ease of computation. We also explore the possibilities of having more or less interconnects with node disabling or interconnect severing assumed. [2]

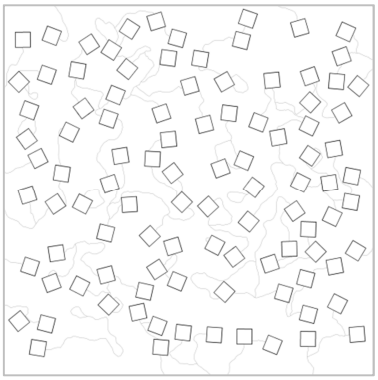

Figure 2 - Sample 100 node topology with 3 transceivers each

In dealing with irregularity, the architecture attempts to represent the arbitrary graph as a regular topology, by mapping out defective nodes and generating a broadcast tree. Using a depth-first traversal, the nodes self-organize into tree(s) that can be traversed in depth-first order and grouped into a Processing Element. Using this abstract structure, the architecture connects these configurable elements with a logical ring.[2]

SOSA is primarily involved in mapping large computational networks on these topologies, but for the purposes of my experimentation, I begin by looking at simpler nodes at the simple logic gate level with no abstraction level in between. However, SOSA does provide an example where is possible to map around defects, thus this will allow this research to assume such capability in our mappings.

2.3 ISCAS High-Level Models

The primary netlists used are from the ISCAS-85/89 and 74X series. The reverse engineering, generating high-level benchmarks from a logic circuit description, was carried out at the University of Michigan. These netlists required development of a custom parser to convert to a useable format for the research. As the raw files before technology-mapped netlists, these netlists contains specifications for gates of large (4+) fanouts and higher level functionality like a D-Flip-flops. These will require further gate decomposition to match, but in the initial stages of simulation, we assume a one to one, gate to node mapping, to temporarily simplify the technology-mapping step.[3]

2.4 Relation to FPGA Placement and Routing

Field Programmable Gate Arrays (FPGAs) allow for the quick implementation of any logic circuit. Like the approach we want to explore for our nano architecture, it begins with a description of a circuit and then a synthesis process into a gate-level network of primitive gates. Technology-mapping turns these gates into programmable logic blocks which are similar to the functionality of a single one of our nodes. Clustering of these blocks, similar to the processing element(PE) structure of SOSA, allow for placement and routing to physically arrange the logic blocks and define connections between them.

The placement goals involve finding maximal compacting of the logic blocks to minimize wire-length, critical path and increase circuit speed. The two terms often used to describe this are wire-length-drive placement and timing-driven placement. Sometimes balancing wire density is also considered. The three major algorithms for placement are min-cut, analytic and simulated annealing. Min-cut relies on recursive partitioning into smaller circuits to individually optimize. Analytic placement attempts to optimize the problem globally, and simulated annealing uses random logic block movement to converge towards an optimal graph. These approaches can all be applied to the placement and routing in our project. [4]

Routing focuses on creating the physical connections between logic blocks to minimize wire-length. This is often reduced to finding the minimal path between two nodes in a graph. We can use Dijkstra's algorithm to find minimum path in succession to determine the routing tree. We can allow certain nodes in our nano-architecture to function as simple pass-through nodes with no functionality to enable better routing. [2]

3. Related Technologies

3.1 Defective FPGAs

There was initial difficulty in finding existing algorithms useful to traverse a tree generated from a circuit netlists and map onto irregular self-assembled networks. Simplification of the problem allowed comparisons to be drawn to the sequence of portioning, placement and routing on a FPGA.

In a FPGA, the input is some netlist of logic blocks and interconnects, and the output is some set of (x,y) coordinates of the nodes after minimization. The portioning-based

algorithms regarding mapping were not relevant in the initial stages of the project, however, it was possible to draw a parallel between the networks of self-assembled, irregular nodes with a fault-tolerant FPGA. Specifically, a FPGA is designed to be a perfect grid in (x,y) space. If we then choose to cut certain interconnects, disable nodes, and short other interconnects, we have a similar makeup to the generated network. It's also important to note that this is presented as an organizational similarity, but functionally, because of the architecture difference, we won't be able to make direct conclusions from this relationship. [4]

This relationship between the two models brings light to HP research group's Teramac project (1990), which had a project vision that desired a programmable array system with capacity for a million gates running at a MHz. The Teramac project intentionally used defective hardware. The methods used to connect these defective pieces together hinted at the viability of parts of this project. [5]

3.2 Modifiable Toolchains

If a toolchain for FPGA routing, placement, portioning is open enough, we hypothesize that the process might be able to take on modifications for mapping onto self-assembled, irregular networks.

Rasp, an open FPGA/CPLD technology mapping and synthesis package is developed and maintained by the UCLA VLSI CAD LAB. Its project flow begins at the gate decomposition process and ends at the architecture specific mapping. Though not directly applicable, the package contains many algorithms that are pertinent to this thesis' specific

challenge. Particularly, the gate decomposition algorithm would be essential in reducing the ISCAS netlists to a dimension that can be mapped onto SOSA. [6]

Altera maintains the Quartus University Interface Program which provides university and other researcher's means to plug new CAD tools and ideas into the Altera Quartus CAD flow. Specifically, researchers are able to modify the intermediate files generated between each step of the Quartus project flow. This includes changing the technology mapping, placement, routing algorithms and other parameters in the process. Theoretically, we may be able to explore the possibility of using this CAD flow with our customized irregular architecture. This would allow us to take advantage of the optimized placement and routing algorithms in place. [7]

University of Toronto's FPGA CAD Tool, Versatile Place and Route, outperforms all published FPGA place and route tools at the time in terms of minimizing routing area. VPR modifies and improves upon previous simulated annealing tools. It was designed to be capable of targeting a broad range of FPGA architectures, and the specific algorithms in its netlist translation and clustering tool VPACK could be quite useful in minimizing critical path in our circuit mappings. I believe that given the architectural parameters that can be specified in VPACK's input architecture description file, it should be able to simulate the SOSA architecture along with the irregular topology of the network. [8]

4. Barebones Design

4.1 Scope and Methods

The barebones design attempts to design a mapping algorithm/tool from scratch in attempts to analyze the effects of different parameters on the likelihood of finding a solution.

Ideally, the algorithms described in the related technologies can be inserted into this design after customization for our architecture.

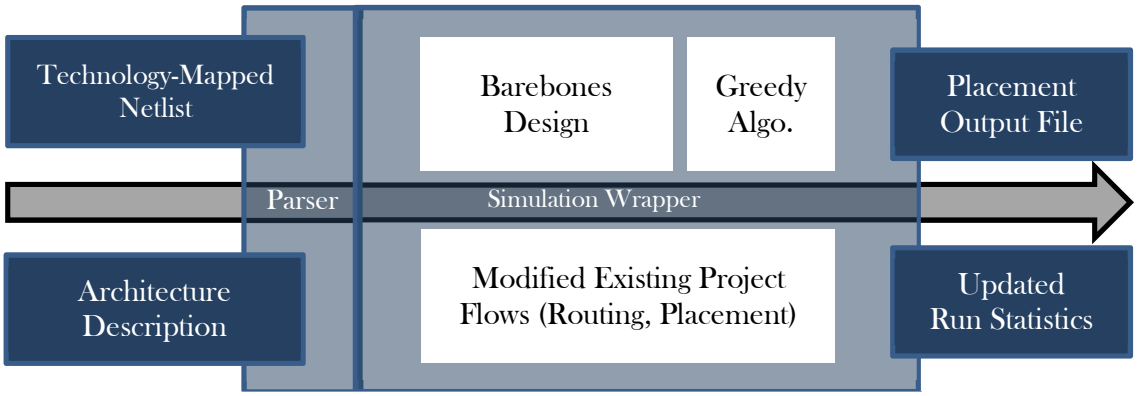

Figure 3 - Planned overall design

In figure 3, we can observe the intended design of a system that could possibly place a logical circuit on a randomly assembled network. I first worked on understanding the various technologies involved and the various file formats at each level. After parsing in the digital logic and topographic information, we can render the data into a technology-mapped set that can be used. I can then either use self-designed placement algorithm to map the circuit or attempt to cut into a current project flow and utilize preexisting resources. The success ratios of mappings given whatever constraints that we decide would dictate if the hypothesis is possible.

4.2 Greedy Approach

We define both the circuit netlist and the architecture topology as hypergraphs of vertices X, and edges E, where an edge can connect any number of vertices:

$$X = \{ x_m | m \in \mathrm{M} \} \qquad E = \{ e_i | i \in I, e_i \subseteq X \}$$

Minimizing critical path in a successful mapping is thus NP-complete in such a scenario. We begin with a greedy implementation using a Monte Carlo method to determine the initial placement. Traversing the netlist with a greedy breadth-first search yielded less backtracking for higher-fan out topologies. Also, pass-through nodes, high-fan out, logic decomposition, and more were necessary to handle edge cases.

Without backtracking, the software attempts to:

1. Select a NetNode (from the top)
2. Choose a random(or iterative) Physical Node
3. Determine if NetNode fanout <= PhysicalNode interconnects
4. If True, pair the two together.
5. Follow NetNode fanout to get next NetNode
6. Choose PhysicalNode randomly from what previous PhysicalNode was connected to

While optimizing at each node was simple to do. This greedy method required much backtracking in order to lessen the failure ratio. To work out edge cases that might come up, I first progressed doing a hand-trace mapping, which resulted high failure rates without the backtracking. It became apparent that prior knowledge of where large nets are located in both the logic circuit and topology could greatly lower the amount of backtracking required.

Backtracking resulted in more complicated issues that probably still remain to be debugged because of certain edge cases. There was a lot of difficulty in traversing a hypergraph, and the tool I made has difficulty with large node networks of greater than 300. This is perhaps due in part to inefficient backtracking.

4.3 Connectivity

Initial walk-throughs of the mapping resulted in nearly 100% failure rate. I noticed that this may be due in part to the connectivity of the network. We define connectivity to be

the number of nodes that can be reached from some fused or unfused set of links. The generated networks of 10,20,50, 100 nodes were set to have a maximum fanout of 4. At this relatively low number, fused links were minimal, but there is a possibility that this provides limited functionality. I began to look at how different parameters changed the connectivity of the network.

Note that in the topology generator, after the nodes are places, links are grown from each node with random angles and distance until it collides with another link or node or it reaches a preset length. The number of links per node is related directly to the number of transceivers on a node.[9]

At a higher fused link ratio, connectivity jumped significantly, 4-fanout resulted in minimal numbers of fused links, but 8-fanout has significantly larger nets. The numbers here have to be recalculated, but generally around 4-fanout, the mean net size is around 3 with a median of 2, but as your approach 8-fanout, the mean jumps to around 20 with a median of like 15. 8-fanout is impractical because of need to fit 8 transceivers onto one lattice, and a network of large nets will only be useful given technology that is able to cut links cleanly. Otherwise, large nets will actually render many nodes/nets useless because of shared busing.

Another parameter related to connectivity is how to deal with fused links. If we assume that all nodes on a link can communicate with all other nodes on that link, then we should enjoy much higher success ratios in mapping a circuit. A fused-bus is probably unlikely due to the technology involved to control such a process. Two possible ways of dealing with this are disabling the collateral nodes that are on the same link and cutting specific links. The former is a less desirable method.

4.4 Simulation Results

Netlist	# of Nodes In Topology					
	10	30	120	200	250	300
Full Adder (5 gate elements)	>20%	>40%	>50%	>50%	>50%	>50%
S208.1isc (66 gate elements)	NA	NA	<10%	<10%	<20%	<20%
74L85.isc (188 gate elements)	NA	NA	NA	<10%	<10%	<10%

Figure 4 - Approximate successful-placements from hand tracing and computer-aided simulation. (Out of 10 trails using transceiver number of 5/6)

Simulation failed to complete without allowing the cutting of interconnects instead of disabling collateral nodes of a successful node mapping. There was not enough time to construct and customize the implementations of the algorithms developed for FPGA placement and routing. Overall, mappings by hand and initial greedy simulations return few successful mappings.

Figure 4 is generated from running the developed program to see if a successful mapping occurs. Each circuit-topology pair was ran 10 times to see the success rate. As can

node disabling, these results would be significantly better. However, these are ran under the assumption that the there are no defective nodes and links. Presently, it looks as if the Y-node-topology to contain some circuit of X gate elements requires Y to be significantly greater then X to enable high chance of successful mappings. From the hand-traces it is apparent that the limiting factor here is connectivity and how one chooses to deal with those

issues of a shared bus. These results do not include proper gate decomposition, which will starve out circuits quickly with node disabling because high-connectivity nets are fewer in number and without gate decomposition, circuits with multiple high fan-out gates will often run into a dead end.

5. Repurposing of Existing Tools

5.1 RASP

I looked at this technology mapping and synthesis package hoping to use their project flow as the backbone of a custom simulation. However, it was paired with the UCLA FPGA Mapping Package that was not completely open to architecture definition. The limited open-source package of RASP contains the source code for the following mapping algorithms: DAG_Map (depth minimization), FlowMap (depth optimal), FlowMap-r (area-delay tradeoff), FlowSYN(FPGA synthesis), CutMap (simultaneous area delay minimization), HeteroMap(delay optimal mapping for heterogeneous FPGAs), BinaryHM(delay-oriented mapping for heterogeneous FPGAs with bounded resources, and PLAmap (delay-oriented mapping for CPLDs).[6]

Most of these algorithms relied on improvements with specific architectural needs that might not apply to irregular topologies, but the initial gate-decomposition was useful to further technology-mapping for our netlists. While the system assumes general gate networks of K-input Lookup-Tables, we can begin by assuming related functionality. Because K in our system is set to 4 default and ranges up to 8 in our testing, depth-optimal technology mapping is NP-hard. Reducing circuit depth should aid in reducing overall critical path of the mapping. [10]

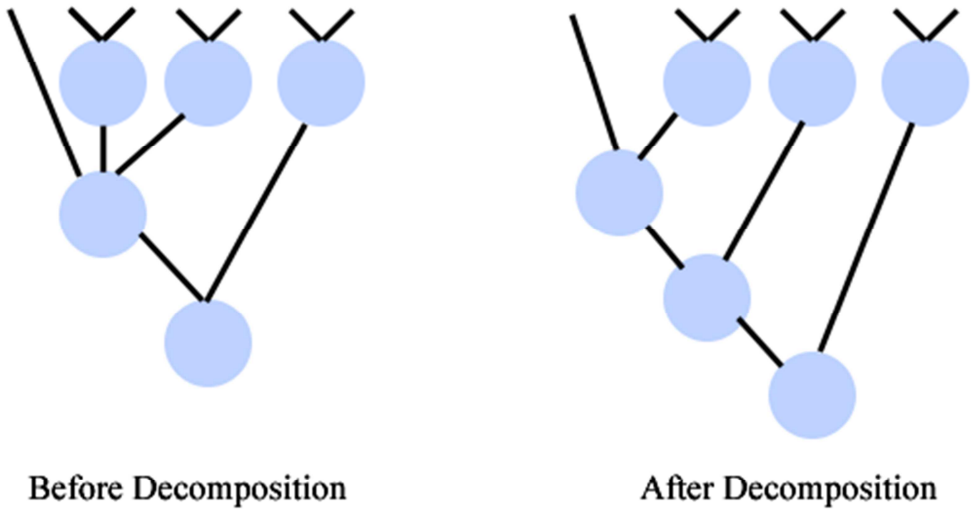

Figure 5 - Gate decomposition of a network of depth 3 to a decomposed network of depth 2. [10]

I attempted to implement basic gate decomposition in the barebones design with seemingly good results. However, my implementation only manages to expand flip-flop declarations into simple gates pre-determined depth, and then expand gates of large fanouts into smaller gates. This can be easily improved upon with the methods discussed in the papers, and hopefully can be added to the system at a later time.

The placement algorithms were generally all very involved and were not implemented since I opted for using pre-made project flows to be discussed in the next section.

5.2 Quartus Toolchain

I use the command line to access each step of the Quartus project flow. Since it takes in Verilog hardware descriptions, I first converted a test ISCAS netlist to Verilog format as input to the software. The resulting VQM (Verilog Quartus Mapping) format is the hardware description output from the synthesis tool. The function *quartus_map fnf –part=<Device*

Name> will perform HDL language elaboration, technology-independent logic optimization, technology mapping into lookup tables(LUT) and then packs flip-flops and lookup tables together to form logic cells.[7]

We want to match and modify these LUTs to map to a node on our topology. The *quartus_fit* command in Altera's FPGA design means to pack logic cells into logic array blocks, placing these blocks, and routing the connections between them. Hypothetically, this function can be used on our topology with certain constraints to take advantage of Quartus' algorithms. However, we have to assume look-up table functionality on each node and the ability to cut links without disabling node function. We can also use defined constrained routing in a RCF file to manually locate rigid elements such as the location of external output and perhaps other specialize nodes if required. The function generates an image that can viewed in Altera's chip viewer to show the resulting mapping and a QSF file with the exact pin location assignments of form:

#Pin & Location Assignments
#==========================
set_location_assignment LAB_X52_Y30 –to Node0
set_location_assignment LAB_X52_Y30 -to Node1
set_location_assignment PIN_D1 –to Input1

The described above process depends on the Quartus toolchain understanding the architecture we have defined. Within the files, I found architecture files defined in an undocumented XML format. It was difficult to ascertain where these architecture definitions were being used by Quartus or see if there was the possibility that there were internal compiled files already being used. Conversion of our architecture into the XML format was

also impractical because of the lack of documentation. Because of the tedious nature of these tasks, I didn't continue further on this portion of the research.

5.3 VPR and T-VPack

Since the discovery of the T-Vpack's applicability to our project occurred late into the research, not much headway was made in this direction. T-VPack doesn't actually require VPR and can be used as input into the Quartus flow described above, but VPR allows for easy definition of a architecture description file which was lacking in the Quartus flow. Using a *.blif* format to describe the circuit as a netlist of LUTs and Flip Flops provides the input into the T-VPack that packs the elements into logic blocks and outputs a *.net* netlist format for input into VPR along with some architecture file. I attempted to write a script to convert the ISCAS circuit descriptions to the *.blif* format but that remains to be debugged.[8] However, testing with some the benchmark circuits provided in the package showed that this should be quite simple to do. The FPGA architecture format is quite similar to the ones found in the QUIP package with some structural differences. The main difference arises in the ability of VPR to read in that architecture. However, I still ran into the problem of fully converting our architecture into that format. Judging from the quick modifications I made to the benchmark architectures and running through this CAD flow, with a correct custom architecture format, this is a completely viable option to implement the placement and routing of our circuits onto our topologies. It is important to note that we are assuming we have that ability to control certain elements in generating our node topology that might be a given in FPGA architectures, so this remains a special case to be explored.

6. Conclusions and Extensions

We managed to explore the difficulties involved in mapping circuits onto irregular networks. While altering parameters like maximum fanout, node function, and link disabling may improve the success ratio of mappings, it is difficult to fully control these parameters at the nano-scale. Despite these constraints, it still may be possible to modify current tools to simulate the mapping of some arbitrary logic circuit onto a irregular self-assembled topology.

We have shown through the greedy approach that connectivity is an important issue to be considered and successful results become more favorable if more control is assumed. Ideally, being able to cut links cleanly will probably give higher likelihood of successful mappings; however it probably more likely that certain mapped nets may render collateral nodes useless.

Current routing and placement technologies employ many sophisticated algorithms that, while applicable, are optimized to regular networks, and not completely suitable for irregular, highly defective graphs. However we have also explored how to utilize current technologies to help map the circuit, taking advantage of the algorithms developed for FPGA placement/routing.

This thesis could be improved upon if more emphasis on depth into a specific possibility rather than the breadth approach used. Significant research results and notes are also missing due to a uncontrollable notebook theft, but more quantitative data would help strengthen the qualitative analysis provided in this document.

If there were more time, I would have liked to complete and debug the barebones design to generate relevant statistics as it attempts to do placement using a variety of

algorithms to compare. The individual algorithms used for FPGA placement should be analyzed more meticulously to see where modifications can be made for usage with our system. Using VPR has possibilities in this realm that would require further effort to convert our architecture into a working input file for that system.

The next steps should involve more computationally rigorous simulations of circuit mappings to find better correlations between the success ratio and the network generation parameters. A project flow to compare this statistic among varying decomposition, routing, and placement algorithms would be essential to determine their viability.

We want to continue working towards developing a system and tool that has a high probability of mapping some arbitrary logic circuit onto a self-assembled network. We want to be able to do this while minimizing critical path and maintain high success ratios while maintain minimal levels of control that may or may not be possible.

7. References

[1] Y. Liu, C. Dwyer, and A. Lebeck. “Routing in self-organizing nano-scale irregular networks”. *J. Emerg. Technol. Comput. Syst.* 6, 1, Article 3 (March 2008), 21 pages. 2008.

[2] J. Patwardhan, V. Johri, C. Dwyer, A. Lebeck. “A Defect Tolerant Self-Organizing Nanoscale SIMD Architecture” *International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2006.

[3]M. C. Hansen, H.Yalcin, and J. P. Hayes. “Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering”. *IEEE Des. Test* 16, 3 (July 1999), 72-80. DOI=10.1109/54.785838 http://dx.doi.org/10.1109/54.785838

[4]Arvind Kumar and Sandip Tiwari. 2004. Defect tolerance for nanocomputer architecture. *Proceedings of the 2004 international workshop on System level interconnect prediction* (SLIP '04). ACM, New York, NY, USA, 89-96. DOI=10.1145/966747.966765 http://doi.acm.org/10.1145/966747.966765

[5] W. B. Culbertson, R. Amerson, R. J. Carter. “Defect Tolerance on the Teramac Custom Computer”. Proceedings of the 1997 IEEE Symposium on FPGA’s for Custom Computing Machines(FCCM ’97)

[6] Cong, Jason. "RASP - LUT Based FPGA Technology Mapping Package." *UCLA VLSI CAD Lab Home Page*. Web. 19 Apr. 2011. <http://cadlab.cs.ucla.edu/software_release/rasp/htdocs/>.

[7] Malhotra, S.; Borer, T.P.; Singh, D.P.; Brown, S.D.; , "The Quartus University Interface Program: enabling advanced FPGA research," *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on* , vol., no., pp. 225- 230, 6-8 Dec. 2004
[8] Xingzheng Li; Haigang Yang; Hua Zhong; , "Use of VPR in Design of FPGA Architecture," *Solid-State and Integrated Circuit Technology, 2006. ICSICT '06. 8th International Conference on* , vol., no., pp.1880-1882, 23-26 Oct. 2006

[6] C. Harting. “Compution on Self-Organized Networks”, 1999

[7] J. Patwardhn, C. Dwyer, A. Lebeck. “Self-Assembled Networks: Control vs. Complexity,” *International Conference on Nano-Networks*, 2006.

[8] C. Dwyer. A. Lebeck. “Self Assembled Architecture and the Temporal Aspects of Computing” IEEE Computer Society. 2005

[9] J. Patwardhan, C. Dwyer, A. Lebeck, D. Sorin. “NANA: A Nano-Scale Active Network Architecture,” *ACM Journal on Emerging Technologies in Computing Systems*, 2(1):1-30, 2006.

[10] J. Cong, and Y. Hwang, "Structural Gate Decomposition for Depth-Optimal Technology Mapping in LUT-based FPGA Designs," *Proc. ACM/IEEE 33rd Design Automation Conf.*, pp. 726-729, 1996.