# Architecturally-Enforced InfoSec in a General-Purpose Self-Configurable System

Nicholas J. Macias Peter M. Athanas
Virginia Polytechnic Institute and State University
Bradley Department of Electrical and Computer Engineering
Blacksburg, Virginia, USA
nmacias@vt.edu athanas@vt.edu

***Abstract*—Unique characteristics of biological systems are described, and similarities are made to certain computing architectures. The security challenges posed by these characteristics are discussed. A method of securely isolating portions of a design using introspective capabilities of a fine-grain self-configurable device is presented. Experimental results are discussed, and plans for future work are given.**

## I. Introduction

Biological systems provide inspiration for a number of artificial processing models such as artificial neural networks [1], genetic programming [2] and adaptive design [3]. More-generally, biological systems of different types exhibit similar characteristics in their basic architecture: they are composed of simple, semi-homogeneous cells; the cells are interconnected in a mainly-local scheme; increased complexity is associated with a larger number of cells, vs. an increase in the complexity of each cell; operations are mainly localized, with many parts of the system acting independent of any centralized control; and very few if any cells are absolutely critical to the system's overall viability (in other words, the system tends to be fairly defect-tolerant). It is thus natural to explore artificial computing architectures which exhibit similar system features. One such architecture is the Cell Matrix [4].

The Cell Matrix architecture has been applied to research in such bio-inspired areas as embryonics [5], fault handling [6] and evolvable hardware [7]. In these areas, it is often desirable/necessary for circuitry to be able to both analyze and modify itself. For example, in responding to a defect in a triple-redundancy system, repair circuitry may analyze the remaining non-defective circuits, and use them to re-configure or re-locate the failed elements [8]. Given the capability of circuits to analyze and modify themselves, more-sophisticated approaches to fault-tolerance can be developed, including the use of *artificial embryological techniques* for re-growing a circuit following large-scale failure [9]. For such applications, the ability of circuitry to analyze other circuitry is vital. From an information-security (InfoSec) standpoint though, this introspection capability in fact poses a considerable challenge.

Traditional infosec *within* a computing system may be focused primarily on data and software: preventing its unauthorized access (confidentiality) or modification (integrity), detecting when such breaches occur, and so on. This is different from the case of a reconfigurable [10], or self-configurable computing system, where the hardware is actually modifiable, and thus subject to the same security concerns as the system's software or data.

In this paper, circuitry that can be implemented on the Cell Matrix, and that takes advantage of certain architectural features of the Cell Matrix to implement low-level infosec will be described. Section 2 provides background on the Cell Matrix architecture, and discusses infosec strategies on externally-configured devices such as FPGAs [10]. Section 3 describes circuitry which can be implemented inside a trusted region of the Cell Matrix, and which will automatically detect and prevent any attempt by non-trusted regions to access the configuration information from within the trusted region. Section 4 will describe experimental results, and Section 5 will discuss future work.

## II. Background

Reconfigurable devices such as FPGAs need to be configured before they can be used. This configuration involves loading a particular bitstream into the device, so as to cause the device to act a desired way. Since the bitstream specifies everything about the final circuit's design, protection of the implemented circuit begins with protection of the bitstream itself. This is one level of security in protecting a circuit's design. In modern devices, such security may be achieved via encryption of the bitstream [11], which affords good security as long as the encryption key remains secret. A perhaps simpler approach is to simply not reveal details of how the device interprets the bitstream. This form of security has several disadvantages however: it is only as strong as the difficulty of reverse-engineering the FPGA's configuration mechanism; it may frustrate clients who wish to interact with the device in non-standard ways; and it leaves the device manufacturer solely responsible for tool development and support.

Nonetheless, bitstream security can be achieved to a degree often acceptable to the user. In extreme cases, FPGA-based devices can be configured in-house, and released already powered-up, with no need to release the actual configuration bitstream itself. What remains is a secondary security challenge: preventing someone in possession of the configured device from learning its exact configuration. This could be achieved by simply not having a readout mechanism. This again, however, may frustrate users, as a general readout capability may be vital for debugging designs[12]. Moreover, even if not used, such portals to the configuration store will likely remain on these devices for testability reason. Because of this, they remain susceptible to malicious activity or viruses.

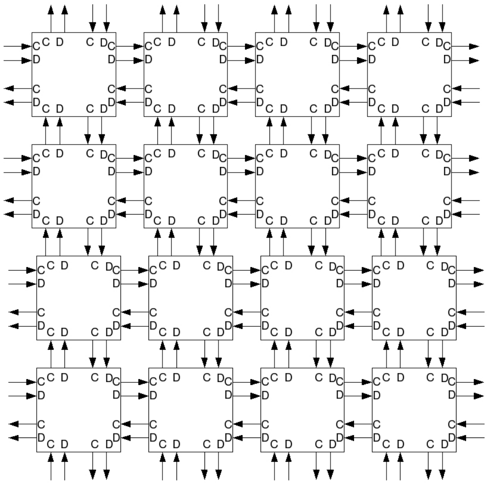

**Figure 1. A 4x4 Cell Matrix**

A self-configurable architecture, such as the Cell Matrix, further frustrates these security efforts: since the system is designed to be self-analyzing and self-modifying, the architecture itself supports the very actions we wish to control and, in some cases, prevent. Rather than disable those capabilities, we will use them, as described in Section 3, to monitor the actions of untrusted circuits and respond accordingly.

The Cell Matrix is composed of a regular collection of simply, homogeneous processors called *cells*, interconnected in a fixed topology, with each cell directly connected to only a small number of immediate *neighbors*. Figure 1 shows a small collection of 4-sided cells in a 2-D von Neumann interconnection topology. Each cell exchanges two bits – C and D - with each of its neighbors. 4-sided cells contain a 128-bit *truth table* which specifies, for each of the 16 possible combinations of incoming D values, the 4 D and 4 C outputs which the cell should produce.

Each cell acts in one of two different *modes*: D-mode and C-mode. In D (Data processing)-mode, D and C outputs are produced using the cell's truth table. In C (Configuration)-mode, the cell's truth table is being read and written by one or more neighboring cells. The mode of a cell is determined by the value of its C inputs: if any C inputs are 1, then the cell is in C-mode. Otherwise, the cell is in D-mode. Thus, any cell can place any neighboring cell into C-mode by simply asserting its own C output to that neighbor.

For the circuits discussed in the next section, it will be important to understand more-fully the details of how a cell behaves in C-mode. When one more more of a cell's C inputs are asserted, the cell enters C-mode. Any side on which the C input is asserted is called an “active side.” When a cell is in C-mode:

- all C outputs are forced to 0;
- all D outputs on non-active sides are forced to 0;
- the D output on each active side is set to the bit value currently being read from the cell's truth table; and
- the D inputs from each active side are ORed together, and the resulting value will be used to re-load one bit of the cell's truth table.

The reading and writing of bits in the cell's truth table occur in sync with a system-wide clock:

- an internal counter (specific to each cell) points to a single bit of the cell's truth table;
- on the clock's rising edge, the active sides' D inputs are latched; and
- on the clock's falling edge, the latched values are ORed and loaded into the truth table at the position indicated by the cell's internal counter; the internal counter is incremented; and the truth table bit pointed to by the counter is sent to each active sides' D output.

The circuitry for realizing these behaviors, i.e., for implementing single cell, is extremely simple, comprised of a single truth table memory, a 7-bit counter, and a few dozen gates of glue logic. It is thus an easy build target for contemporary CMOS, but is also likely well-suited for newer, emerging technologies such as DNA scaffolding [13], Quantum CA [14] and Crossbar Nanocomputing [15], where huge numbers of cells could potentially be created, on 2-D or 3-D substrates..

With this relatively simple configuration scheme, cells within the Cell Matrix can read, modify and write neighboring cells' configurations. Collections of cells can work together to access non-adjacent cells, reading and writing their configurations, thus analyzing and changing the behavior of remote regions of the Matrix.

This capacity for self-analysis and self-configuration can be used to perform many non-standard operations, including self-replication of circuits [9], modification of circuits in response to physical defects of the underlying hardware [16], and embryological growth of circuitry [17]. More generally, when large numbers of cells work together, higher-order behaviors such as adaption and self-organization may begin to emerge. It is for this reason that architectures that support large numbers of components (cells) are so interesting from a bio-inspired point of view. And, because of this interest in high-component-count systems, the need for distributed, localized control becomes essential. This forces us to explore systems with such capabilities as internal analysis and modification.

## III. InfoSec Circuitry

The same characteristics that make the Cell Matrix well-suited to implementation of bio-inspired circuitry also pose a considerable challenge from a security point of view. Given that a small set of cells can, theoretically, read and modify the configuration of any other cells within the Matrix, there is potential cause for great concern when multiple circuits, some of which may be untrusted, are placed on the same substrate as circuits we wish to secure. One example of this would be in a security device whose physical security cannot be assured (e.g., a smart ID card containing proprietary encryption circuitry). Another example would be a weapons control system aboard an aircraft, which could potentially fall into enemy hands. In these cases, regions of the Matrix could be reconfigured and used to gain access to the secure circuitry, analyzing its configuration and thus revealing sensitive information.

However, by using certain characteristics of a cell's C-mode behavior, it is possible to detect and prevent unauthorized access to secure regions of the Matrix. C-mode access can be completely blocked, or can be granted based on receipt of an authorization key, or on completion of an appropriate handshake. A self-destruct can also be implemented, effectively walling-off secure regions as soon as an attempt at C-mode access is made. Despite these safeguards, D-mode exchanges can still be made between secure- and untrusted regions of the Matrix.

Figure 2 shows a two-cell circuit for partially controlling the actions of an untrusted cell. The cell labeled “GC” is called a “Guard Cell.” It is placed into C-mode by the cell on its left. Since a C-mode cell can never assert its own C outputs, GC will never attempt to configure any of its neighboring cells. This means that an untrusted cell (such as the cell labeled “?” to the right of the dashed line) is unable to modify GC's behavior in general, and, in particular, **cannot coerce GC into reading or modifying the configuration of any cell within the trusted region**. This two-cell circuit thus blocks the untrusted cell's only *direct* access to trusted cells.

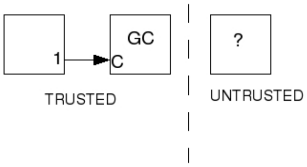

**Figure 2. A simple two-cell guard circuit.**

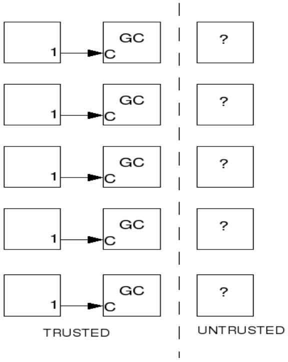

**Figure 3. A guard wall.**

Of course, the untrusted cell in Figure 2 could use cells above or below it to attempt to access trusted cells above or below GC. This can be handled as shown in Figure 3, by simply building a wall of guard cells (called a guard wall”). In this case, none of the cells in the untrusted region have direct access to cells in the trusted region. Similarly, a closed ring of guard walls (a “guard ring”) can completely wall off a trusted region of cells from all untrusted access.

Of course, the problem with this simple scheme is that *all* access has been blocked to the trusted cells: both C- and D-mode access is prevented, since the guard cells (“GC”) cannot be used to pass data, but instead output either 0 or their own truth table bits. The trusted circuitry is so tightly secured that it cannot be used!

One option to loosen the security is, in Figure 2, to have the leftmost cell assert a 1 to its C output only when unauthorized access is detected. The guard ring would effectively be disabled, allowing data to flow between the trusted and untrusted regions, but then, following any break-in attempt, the ring would become active and prevent unauthorized access. The difficulty with this scheme is detecting when unauthorized access is attempted.

Figure 4 shows a single-cell *break-in detector* (labeled “BID”). This cell is configured to pass *data* between its

Western edge (inside the trusted region) and its Eastern edge (at the untrusted boundary). The cell is also outputting a 1 to its Northern edge (“BREAKIN-BAR”). This simple configuration is sufficient to detect any attempted C-mode access by the untrusted cell (labeled “?”): if the untrusted cell places BID into C-mode, it's Northern output will drop from 1 to 0. This is the break-in indicator: when BREAKIN-BAR drops, internal circuitry can detect this, and activate the LOCK signal, which then forces BID into C-mode. Thus, even if the untrusted cell “?” reconfigured BID, it can't actually *use* BID, since BID is held in C-mode by the LOCK signal.

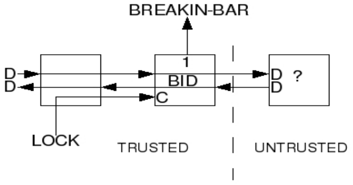

**Figure 4. Automatic break-in detector.**

Figure 5 shows how an entire region of trusted cells (surrounded by the dashed rectangle) can be protected in this way. Any cell outside the dashed rectangle is considered untrusted. Nonetheless, each trusted/untrusted cell pair straddling this rectangle may freely exchange data across the trust boundary: bits are passed through the boundary cells in simple D-mode transfers.

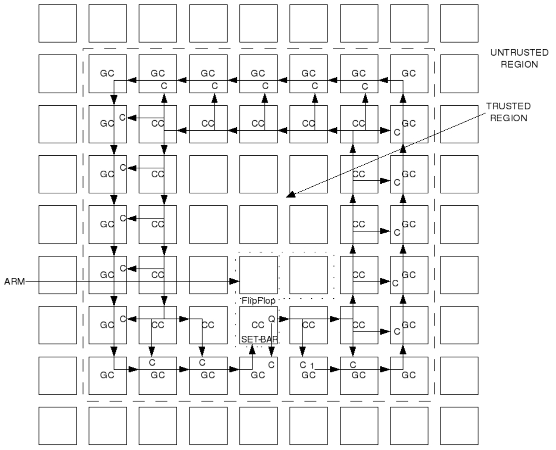

**Figure 5. Fully-protected region with automatic break-in detection and response.**

Under normal conditions (no break-in attempt), the perimeter's guard cells (labeled “GC”) are collectively passing a “1” bit around the entire perimeter. This bit is then passed to the SET-BAR input of a flip-flop. The flip flop is initially cleared, i.e., its Q output is 0. This Q output is sent through an inner ring of *control cells* (labeled “CC”), each of which copies this incoming bit to its corresponding GC's C input. Thus, under normal conditions, each GC is in D-mode (since all C inputs are 0), and data transfer can occur across each GC, allowing free data transfer across the trust boundary.

If, however, any untrusted cell places any GC into C-mode, each of that GC's D outputs within the trusted region will immediately be forced to 0. In particular, the GC ring will begin transmitting a 0 from that cell forward, which eventually will hit the flip flop's SET-BAR input. This will set the flip flop, driving its Q output to 1, which will thus cause each CC cell to assert its own C output to its corresponding GC cell. Once this occurs, the trusted region is effectively isolated, forever, from the untrusted region: without any way to access the flip flop's cells, Q can never be reset to 0, and thus each GC will forever remain in C-mode. No data can pass through any GC, nor can any GC configure any adjacent cells. Since the GCs completely surround the trusted region, the trusted cells are thus completely isolated from the rest of the Matrix.

One potential complication with the circuit in Figure 5 is how to initially build the circuit. As it stands, the flip flop would need to be configured *after* the CC and GC rings are in place, to avoid falsely triggering a break-in detection. In fact, since D access is available between the trusted and untrusted regions, an ARM signal can be sent into the completed circuit. This signal sets another internal flip flop, the output of which is used to gate the main flip flop's Q output onto the CC ring. Thus, the CC/GC circuitry is disabled until the ARM signal is asserted. Once that occurs, the CC/GC circuit is armed, and can not be disabled from outside the trusted region. Note that such an arming technique could be useful for changing the size of the trusted region. Internal circuitry could build a larger trust ring around itself, verify the integrity of all contained cells, arm the larger ring, then disarm and disassemble the smaller ring. This would allow the trusted region to expand (as well as potentially shrink) as system requirements change.

## IV. Experiments

Figure 6 shows the layout of the trusted region outlined in Figure 5. The ARM input is located on the left (row 5, column 1). The layout was converted into a binary file, and loaded into the Cell Matrix simulator. This allowed testing of the break-in detection and response circuit.

Upon initial loading of the circuit, it was still possible to place the perimeter cells into C-mode, i.e., one could configure perimeter cells so as to gain access to cells within the trusted region. This was the expected behavior.

When the ARM input was raised and then subsequently reset (the reset is not actually necessary), the entire circuit entered an armed state, ready to detect the change of any perimeter cell's state from D-mode to C-mode. Once armed, placing any of these outside cells into C-mode (by simply asserting any such cell's exposed C inputs) immediately activated the internal circuitry, causing each of the CC cells to force its corresponding GC cell(s) into C-mode, thus preventing use of these cells for any purpose (including gaining access to trusted-region cells). Thus, the circuit works exactly as desired.

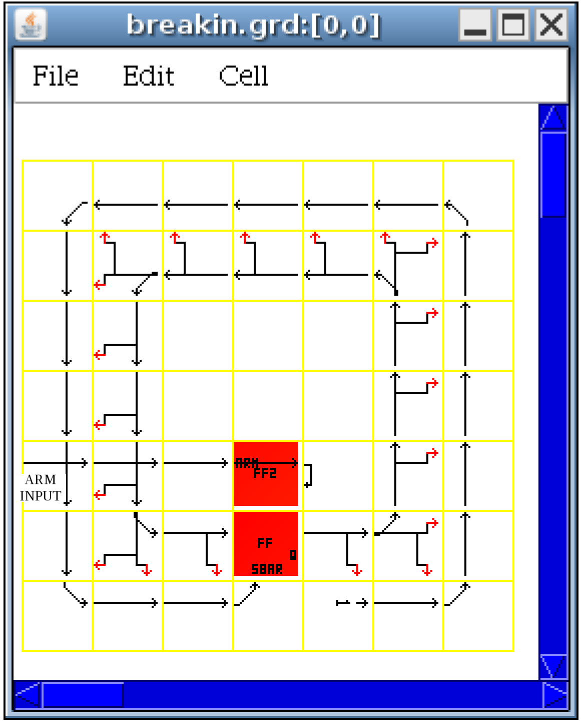

**Figure 6. Layout of break-in detector and response system, used to generate a binary file for subsequent simulation.**

The circuit in Figures 5 and 6 is laid out as a *7x7* circuit. This leaves only a few cells inside for the actual protected circuitry. In practice, much larger GC and CC rings would be used, allowing an arbitrarily-large region to be protected. For an *NxM* region, it would take approximately *2N+2M* propagation delays for the entire region to become protected (i.e., for each GC to be placed into C-mode from within). Since actual C-mode configuration occurs in sync with the (relatively slow-moving) system clock, it is highly unlikely an outside circuit could configure even a single GC bit, and probably impossible any CC could be compromised.

## V. Conclusions and Future Work

The Cell Matrix has previously been demonstrated as a useful architecture for implementing bio-inspired systems. The architecture's self-configurability, which makes possible autonomous, adaptive behavior, also poses potential security risks should untrusted circuitry be placed on the same substrate as trusted circuitry. This paper has presented simple, effective circuitry for isolating regions of the Matrix, as well as circuitry for detecting attempts at reading or changing a cell's configuration. By combining these circuits, an entire 2-D region of cells (a “trusted region”) can be protected from unauthorized access by untrusted circuitry. The protection scheme allows *data* to pass between trusted and untrusted regions, while preventing untrusted cells from reading or modifying the configuration of trusted cells. If such access is attempted, the circuitry effectively shuts off C-mode access to all the cells surrounding the trusted region.

The circuitry presented in this paper is somewhat Draconian: any attempted C-mode access permanently shuts off all access to the entire trusted region. In terms of confidentiality, this may be appropriate, but in terms of availability, it is a severe overreaction. Future work will focus on a more-limited, better-controlled response to such access attempts. For example, the CC ring could be moved further into the interior of the circuit shown in Figure 5. This would effectively set up a guard ring, without actually modifying the GC cells or their state. In this way, once untrusted C-mode access ceases, the circuitry can detect this state, and then re-build the GCs in case they have been modified. The potential show-stopper in this scheme is, if external circuitry sufficiently perturbs the GC configurations, it may not be possible to detect when C-mode access has ceased, i.e., the circuitry may remain locked. Repeated attempted rebuilding of the GC cells, while not an efficient solution, could nonetheless solve this problem.

Another area for future work is to allow *controlled* C-mode access to trusted cells, even after the break-in detector is configured and armed. Logically, allowing C-mode access simply requires clearing the flip flop which is set by the ARM input in Figure 5, i.e., returning the circuit to its initial unarmed state. This is useful assuming access to any of the GC cells is trusted. In practice, we may only wish to grant C-mode access to a small number of GC cells. In this case, the circuity of Figure 5 can be modified to divert the GC datapath (which is conveying the “1” bit counterclockwise around the GC ring) around certain GC cells (perhaps into the CC ring). Those bypassed GC cells can then be placed into C-mode, without affecting the state of the trusted region's circuitry.

Determining when to allow partial- or full-access to GC cells can be negotiated through a handshake between cells on each side of the trust boundary. Since D-mode access is supported across that boundary (such as the ARM input, for example), an untrusted cell can transmit a pre-determined code (password) into the trusted circuitry, which validates the code and then grants C-mode access accordingly. Or, trusted circuitry can issue a challenge, which requires a particular corresponding response [18]. Other validation schemes can similarly be employed.

And finally, this methodology might be extended to other architectures, such as the Virtex-4, which supports Partial Dynamic Reconfiguration [19]. In devices such as this, part of the reconfigurable substrate (say a single column of elements) can be modified while the remainder continues to run. By explicitly building and setting a D-flip flop using gates from two columns, a shutdown of one of the columns can be detected by a failure of the flip flop to retain its SET value. This creates a break-in detector, which internal circuitry can use to identify attempts to partially-reconfigure the device.

## Acknowledgment

The authors wish to thank the reviewers, whose comments helped extend the scope of this work, as well as increased the readability of this paper. Thanks also to karla, who provided the main motivation for my research and submission.

## References

[1] Patterson, D., “Artificial Neural Networks,” Singapore: Prentice Hall, 1996.

[2] Koza, J.R., “Genetic Programming: On the Programming of Computers by Means of Natural Selection,” MIT Press, 1992.

[3] Negoita, M., Torrenson, J. and Hintea, S., “Bio-Inspired Technologies for the Hardware of Adaptive Systems,” Studies in Computational Intelligence, Vol. 179, Springer Berlin, 2009.

[4] Durbeck, L. and Macias, N., “The Cell Matrix: an architecture for nanocomputing,” Nanotechnology vol 12 pp 217-30 (Bristol, Philadelphia: Institute of Physics Publishing), 2001.

[5] Petraglio, E., “Fault Tolerant Self-Replicating Systems,” Thesis No. 2973, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 2004.

[6] Durbeck, L. and Macias, N., “Obtaining quadrillion-transistor logic systems despite imperfect manufacture, hardware failure, and incomplete system specification,” Nano, Quantum and Molecular Computing ed S.K. Shukla and R.I. Bahar Boston: Kluwer Academic Publishers pp 109-132, 2004.

[7] Macias, N., “The PIG Paradigm: The Design and Use of a Massively Parallel Fine Grained Self-Reconfigurable Infinitely Scalable Architecture,” Proceedings of The First NASA/DOD Workshop on Evolvable Hardware (EH’99), A. Stoica, D. Keymeulen and J. Lohn, eds., pgs. 175-180, 1999.

[8] Macias, N. and Durbeck, L., "Autonomous Reversal of Multiple Configuration Upsets in Cell matrix Circuitry," white Paper delivered to Los Alamos National Laboratory under subcontract #90843-001-04 4x, June 2005.

[9] Macias, N. and Athanas, P., "Application of Self-Configurability for Autonomous, Highly-Localized Self-Regulation," AHS, pp. 397-404, Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007), 2007.

[10], Maxfield, C., “The Design Warrior's Guide to FPGAs,” Elsevier, 2004.

[11] Hori, Y., Satoh, A., Sakane, H. and Toda, K., “Bitstream encryption and authentication with AES-GCM in dynamically reconfigurable systems,” IEICE Tech. Rep., vol. 108, no. 48, RECONF2008-3, pp. 13-18, May 2008.

[12] Paulsson, K., Viereck, U., Hübner, M. and Becker, J., “Exploitation of the External JTAG Interface for Internally Controlled Configuration Readback and Self-Reconfiguration of Spartan 3 FPGAs,” IEEE Computer Society Annual Symposium on VLSI, pp 304-309, 2008.

[13] Dwyer, C. et al., “Design tools for a DNA-guided self-assembling carbon nanotube technology,” Nanotechnology 15, pp 1240-1245, 2004.

[14] Tougaw, P. and Lent, C., "Logical devices implemented using quantum cellular automata," J. Appl. Phys. 75, pp 1818–1825, 1994.

[15] Rose, G. S. et al, “Designing CMOS/molecular memories while considering device parameter variations,” ACM Journal on Emerging Technologies in Computing Systems (JETC), Volume 3, Issue 1, Article No. 3, April 2007.

[16] Durbeck, L. and Macias, N., “Defect-tolerant, fine-grained parallel testing of a Cell Matrix,” Proc. SPIE ITCom 2002 Series 4867 ed J Schewel, P James-Roxby, H Schmit and J McHenry pp 71-85, 2002.

[17] Macias, N. and Durbeck, L., "Self-Assembling Circuits with Autonomous Fault Handling," Proc. The 2002 NASA/DOD Conference on Evolvable Hardware, eds. A Stoica, J Lohn, R Katz, D Keymeulen and R Salem Zebulum, pp 46-55, 2002.

[18] Batina, L., Ors, S., Preneel, B. and Vandewalle, J., “Hardware architectures for public key cryptography,” VLSI Journal, Integration 34(1-2) 1–64, 2003.

[19] Taghipour, H., Frounchi, J. and Zarifi, M.H., "Design and implementation of MP3 decoder using partial dynamic reconfiguration on Virtex-4 FPGAs," International Conference on Computer and Communication Engineering, 2008.