Self-Assembling Circuits with Autonomous Fault Handling

Nicholas J. Macias
Cell Matrix Corporation
1004 Palmer Drive
Blacksburg, VA 24060
(877) 473-0882
nmacias@cellmatrix.com

Lisa J. K. Durbeck
Cell Matrix Corporation
1004 Palmer Drive
Blacksburg, VA 24060
(877) 473-0882
ld@cellmatrix.com

Abstract

This paper reports on the results of our recent NASA SBIR contract, "Autonomous Self-Repairing Circuits," in which we developed a novel approach to fault-tolerant circuit synthesis utilizing a self-configurable hardware platform. The approach was based on the use of atomic components called Supercells. These Supercells perform several functions in the building of a desired target circuit: fault detection, fault isolation, configuration of new Supercells, determination of inter-cell wiring paths, and implementation of the final target circuit. By placing these tasks under the control of the Supercells themselves, the resulting system requires minimal external intervention. In particular, for a given target circuit, a fixed configuration string can be used to configure the system, regardless of the location of faults in the underlying hardware. This is because the configuration string does not directly implement the final circuit. Rather, it implements a self-organizing system, and that system then dynamically implements the desired target circuit.

1. Introduction

Fault tolerance is an important goal for modern digital circuits, particularly those applied to critical, remote situations, such as satellite control. Various methods have been used to improve the ability of systems to operate in the presence of faults. One classic approach is to use redundancy, where multiple copies of critical system components are available, allowing failed components to be replaced in the event of failure. In [1], Miller takes a different approach to creating fault tolerant circuits, by evolving digital circuits from components with built-in stochastic behavior. These evolved circuits then behave more predictably in the presence of noise.

The present research is focused on developing a methodology for implementing fault tolerant circuits on top of a general purpose reconfigurable platform. Rather than taking the approach of directly replacing a component once it fails, the present work focuses on the design of a circuit that, once informed that a failure has occurred, can successfully rebuild a working version of itself with minimal external intervention. Such circuits may be called *self-repairing* or *self-assembling circuits*.

This approach to fault tolerance offers advantages over other approaches such as redundancy. In redundancy-based fault tolerance, there is a small number of copies of each system component, so that when a component fails, a replacement can take its place. Such a system is limited by the number of copies it has of each system component, which limits the number of times any given component can fail. FPGA-based redundancy schemes such as [2] use reconfigurability to effectively increase the number of replacement copies available.

In a self-assembling strategy, the system does not have a fixed number of copies of each system component. Instead, any component can be rebuilt from a large pool of identical, fine-grained building blocks, which are shared by all components of the system. Therefore, the ability of the system to self-assemble in the presence of faults is primarily a function of the amount of remaining undamaged hardware, rather than the physical location of the faults.

Another advantage of self-assembly lies in the pattern of faults that can be tolerated. Because failed components are rebuilt from a distributed set of small building blocks, the system can usually still recover from a failure event even if the system's hardware has been damaged in a large number of locations. This type of self-organizing system is similar in spirit to some of the embryological approaches taken by[4], [3] and [5]. Unlike the present work, which recovers from faults by performing a full system rebuild, these embryological approaches are able to recover from faults while continuing to operate. However, the present work may be more resilient than the row- or column- replacement strategy of, say, [4], which can not recover from a pattern of faults that affects every row or column.

Self-assembly with a*utonomous* fault handling refers to the ability of the system to perform its own fault analysis and repair operations. In contrast to systems that rely on external analysis and control, autonomous systems are extremely robust, because there is no additional vulnerability of an external control system.

The goals of this research are thus as follows:

- to develop a methodology for implementing a desired circuit on a general purpose reconfigurable platform;
- to endow these circuits with the ability to self-assemble, that is, to rebuild themselves in the event of a failure, such that full function is restored; and
- to create a system that operates with minimal external intervention.

2. Approach

To implement a particular target circuit, reconfigurable devices require some type of configuration information, which describes how their internal elements should be configured. However, if this configuration information rigidly maps particular device elements to particular pieces of the target circuit, then the configuration will fail if those device elements have been damaged. Therefore, in order to implement a self-assembling circuit, one must avoid overly explicit configuration information. This can be achieved by describing the target circuit more abstractly, in terms of circuit building blocks (such as gates) and connections among those blocks. This abstract description is then used as a guide to configure the device, in order to implement the target circuit.

More specifically, the strategy for creating self-assembling circuits comprises the following steps:

1. create an abstract description of the target circuit, without reference to particular elements within the reconfigurable device;
2. analyze the reconfigurable device's hardware, noting the locations of faulty areas; and
3. configure the device to implement the target circuit, based on the abstract circuit description, while avoiding the faulty device regions identified in step 2.

Step 1 only needs to be performed once per target circuit, and is similar to compilation of code or pre-processing of VHDL. Steps 2 and 3 must, however, be performed each time the device is to be configured, since the location of faults may change over time (for example, in a high radiation environment, more faults may accrue over the lifetime of the device). But if an external system is used to analyze or configure the reconfigurable device, then that external system represents an additional fault vulnerability. This is why one design goal is to minimize external intervention in steps 2 and 3.

One way to achieve this goal is to create circuitry *on the reconfigurable device itself* which performs steps 2 and 3. Such circuitry is called the *control circuit*. Note, however, that it is not sufficient to merely move the control circuit inside the reconfigurable device. To achieve true fault tolerance, the control circuit must *itself* be fault tolerant. But since the control circuit is responsible for creating fault-tolerant circuits, we appear to have a set of circular, self-referential requirements, i.e., a fault tolerant control circuit is necessary in order to implement a fault tolerant control circuit.

Typical reconfigurable devices (e.g. FPGAs) are externally controlled, and thus this type of self-referential behavior is difficult to achieve. However, by using a novel *self-configurable* device, called the Cell Matrix™, this self-referential system can be easily implemented.

3. Cell Matrix Background

The hardware substrate used for this work is the Cell Matrix [6,7]. The Cell Matrix is a fine-grained reconfigurable device, consisting of a homogeneous collection of cells, interconnected in a nearest-neighbor scheme. Cells perform the basic data processing functions of the system, by continuously mapping inputs to outputs. Each cell contains a small memory called a lookup table, whose contents specify the behavior (input-to-output mapping) of the cell. Cells exchange inputs and outputs with a fixed set of adjacent neighboring cells. By properly configuring cells, they can be made to cooperatively implement higher-order functions.

Unlike traditional reconfigurable devices that are controlled by external systems, the Cell Matrix is *self*-configurable. This means that the cells within the Cell Matrix are themselves capable of analyzing and reconfiguring other cells within the matrix. To accomplish this self-configurability, each cell

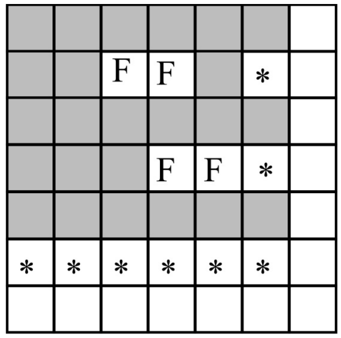

Figure 1 Tiling the Matrix with Supercells

Each square represents a potential Supercell, composed of many Cell Matrix cells. Gray regions are already configured as Supercells. Regions labeled "F" have been determined to contain faulty cells. In the next step, regions to the south of existing Supercells (labeled "*") will be tested and potentially configured as new Supereclls. All regions "*" are acted upon in parallel

operates independently in one of two modes: D mode or C mode. In D mode, incoming data is processed by the cell's lookup table, and used to generate outputs to the cell's neighbors. In C mode, incoming data is used to re-write the cell's lookup table. D mode is thus the normal "data processing" mode of a cell, while C mode is the configuration mode of a cell.

A cell's current mode is determined not by itself, but by its neighbors. Therefore, one cell can modify a neighboring cell's lookup table simply by placing that neighbor in C mode, and then writing to its own output lines to that neighbor. Similarly, a cell may read a neighboring cell's lookup table by placing that neighbor in C mode, and then reading its own input lines from that neighbor.

Finally, it is a cell's own lookup table that governs its ability to control a neighboring cell's mode. Therefore, one cell X may configure a neighboring cell Y, in such a way that Y will subsequently configure a third cell Z. In such a sequence, Y exhibits dual modes of operation: first as the target of a configuration operation, and then as the initiator of a configuration operation. The capability of cells to switch between these dual modes is one key to implementing the self-referential system described at the end of the previous section. Further background on the Cell Matrix architecture can be found at [8,9].

4. Supercell Overview

In the Supercell approach to self-assembling circuits, a small region of the Cell Matrix is first used to implement an initial building block, called a Supercell. Supposing that this single Supercell can be implemented without faults, it then performs a series of tests on nearby regions of the Cell Matrix, looking for defective areas. In regions that are found to be defect free, the initial Supercell then configures additional Supercells. Note that, because Supercells in this second generation are placed only on defect-free regions of the Cell Matrix, and because this is controlled by the original defect-free Supercell, these new Supercells can also be assumed to operate perfectly.

The second generation Supercells then repeat this process, first analyzing nearby regions for faults, and then implementing a third generation of Supercells in the nearby defect-free regions. This process continues for a fixed number of generations. The following should be noted about this strategy:

- The initial Supercell need not be located in a corner of the Cell Matrix; it can be located at any accessible point.
- All of the Supercells implemented are identical to each other.
- Assuming the initial Supercell works properly, and no new defects occur in the hardware, all subsequently configured Supercells can also be assumed to work properly.
- It is the Supercells themselves that are responsible for testing regions and creating new Supercells.
- Multiple regions of the matrix will be tested and configured in parallel, as shown in Figure 1.
- As the number of Supercells grows, so grows the degree of parallelism in each step. Thus there is actually an acceleration in the test/configuration rate of the system. On a two-dimensional Cell Matrix, approximately n^2 Supercells will have been created after 4n steps (new Supercells are first configured to the North, then the West, then the South and finally the East. Thus it takes 4 "steps" to extend the collection of Supercells on all 4 sides). Figures 2a-2i illustrate this O(n^2) growth.
- At all stages of this process, the set of Supercells is guaranteed to be connected, since Supercells

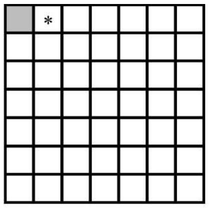

Figure 2a. Gray region indicates location of first Supercell (which could be located anywhere along any edge). "*" marks next region, to the East, to be tested and potentially configured.

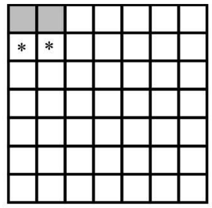

Figure 2b. Two Supercells have been configured, and will next operate on two regions to the South ("*") in parallel.

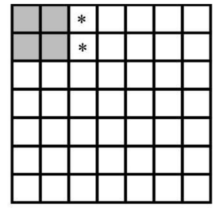

Figure 2c. Four Supercells have been configured, and will next operate on two regions to the East ("*") in parallel.

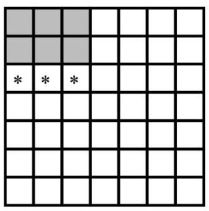

Figure 2d. 6 Superells acting on 3 regions to the South.

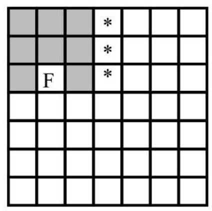

Figure 2e. One of the tested regions ("F") has failed a test. and thus was not configured. 3 new regions are being tested and configured.

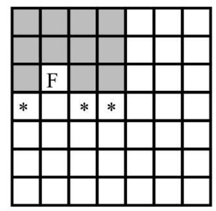

Figure 2f. 11 regions have been configured. Note that only 3 new regions are being operated on, due to faulty region.

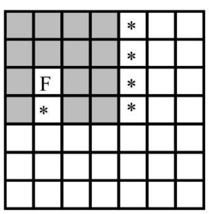

Figure 2g. 14 Supercells have been configured, and will next act on 5 new areas. The region below "F" is being configured by the Supercell to its left.

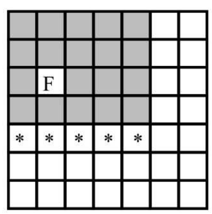

Figure 2h. 19 Supercells will next act upon 5 regions.

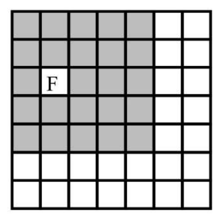

Figure 2i. 24 Supercells have been configured.

are only configured adjacent to pre-existing Supercells.
- The net effect of this process is to tile the defect-free regions of the Cell Matrix with a regular collection of Supercells, while leaving unconfigured any regions containing defects.

Once the Cell Matrix has been tiled with a collection of Supercells, it is this collection of Supercells that performs the task of converting the abstract target circuit description into a working circuit. However, rather than configuring additional regions of the Cell Matrix to implement the target circuit, the Supercells themselves are used as building blocks in the creation of the target circuit. Because the Supercell collection has been generated so as to occupy only defect-free regions of the hardware, the resulting target circuit will also be defect-free, as long there are no new defects. In the event that new defects appear, the entire process of Supercell analysis and synthesis, followed by creation of the target circuit, can simply be repeated.

In order for the Supercells to implement the target circuit, the following steps must be performed:
- particular Supercells must be assigned to particular elements in the target circuit’s abstract description;
- each Supercell must implement the element (i.e., gate) to which it has been assigned;

- each Supercell must determine which other Supercells it needs to exchange data with, based on the abstract target circuit description;
- each Supercell must locate those other Supercells with which it will exchange data; and
- each Supercell must synthesize the necessary pathways for exchanging data with other Supercells.

Supercell behavior can thus be broken down into two basic functions: tiling of the Cell Matrix with Supercells (which includes detection and avoidance of faults), and creation of the desired target circuit from those Supercells. The next two sections discuss the technical details of how Supercells perform these functions.

5. Supercell Details: Tiling of the Matrix

As with all circuits implemented on a Cell Matrix, a Supercell is composed of a collection of Cell Matrix cells, that have been configured in a particular way. Associated with a Supercell is a collection of bitstreams called a Short Configuration Sequence(SCS), whose primary function is to configure a region of the Cell Matrix to act as a Supercell. The SCS is supplied externally, but for a given target circuit, the SCS never changes. Therefore, the SCS may be supplied by an external memory such as an EPROM. Upon receiving the SCS, a Supercell performs three functions:

1. testing of unconfigured regions for faults;
2. configuration of non-faulty, unconfigured regions, to create new Supercells; and
3. transmission of the SCS to other Supercells.

The first step of the tiling process is testing of an unconfigured region, called the Region Under Test (RUT). The RUT is a region adjacent to the Supercell performing the tests, and is the same size as a Supercell. The SCS contains instructions that tell a Supercell how to build pathways throughout the RUT. These pathways give the Supercell access to each side of every individual cell within the RUT. The SCS also contains test patterns to be sent to each individual Cell Under Test (CUT), as well as the expected return pattern from the CUT. So, for example, the SCS may configure a cell as a simple feedback circuit, whose output is expected to be the same as its input. By sending a test pattern of 01010101… and looking for an expected return of 01010101… the Supercell would detect any stuck-at faults in the CUT's input or output. A second test pattern of 11111111… would check for stuck-at-0 faults inside the CUT's lookup table; likewise for a test pattern of all zeros. As a third test, the SCS may configure the CUT as an inverter. Sending a test pattern of 01010101… and comparing the output to an expected return of 10101010… would check for shorted memory bits inside the CUT's lookup table. In similar fashion, different tests may be applied to the CUT.

A Supercell contains a small comparison circuit, that compares the CUT's output to the expected output. This comparison is done only at certain quiescent times during the test cycle, to avoid false failures due to glitches or timing delays.

A Supercell also contains a circuit called a *guardwall*. This circuit runs the full length of each of the Supercell's own edges. Under normal circumstances, the guardwall acts simply as a bank of wires, which transmits data bi-directionally from one side of itself to the other. In this mode, the guardwall is effectively transparent. If, however, the Supercell detects a failure inside a RUT, it can *activate* the guardwall on the side adjacent to the RUT. Once activated, a guardwall blocks all data transmission from one side of itself to the other, thus preventing faulty cells within the RUT from sending bad data to the Supercell.

More importantly, **once activated, a guardwall can not be reconfigured**, that is, the cells from which it is composed can not be reconfigured until a system-wide reset is performed. This is an important feature in light of the Cell Matrix's ability to self-configure. Without it, a faulty cell inside the RUT could not only transmit bad data to the Supercell, but could actually reconfigure the Supercell's constituent cells. An activated guardwall prevents such reconfiguration information from affecting the Supercell.

Note that guardwalls are part of the non-faulty Supercell, **not** the RUT. Since Supercells are only placed in fault-free regions, the Supercell can be assumed to be functioning perfectly (barring the appearance of new faults), and thus it is safe to assume the Supercell's guardwalls are also functioning perfectly.

Note also that under this strategy, a single fault will cause an entire Supercell-sized region to be discarded as faulty. This behavior could be avoided by allowing slightly irregular tilings with shifted Supercells, combined with variable locations for inter-Supercell communication channels. However, it is not clear the benefit would be worth the added complexity.

Following testing of all cells within the RUT, the SCS contains commands that instruct each Supercell to create a new Supercell in adjacent, non-faulty regions. These commands are the usual bootstrap

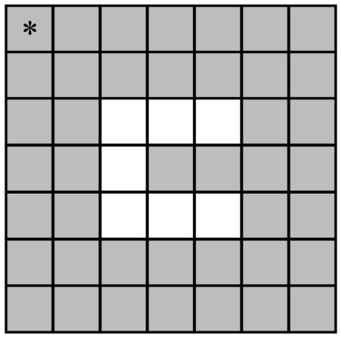

Figure 3a 7x7 Tiling of Supercells Gray regions contain Supercells. White regions are unconfigured due to presence of faults. "*" indicates initial Supercell.

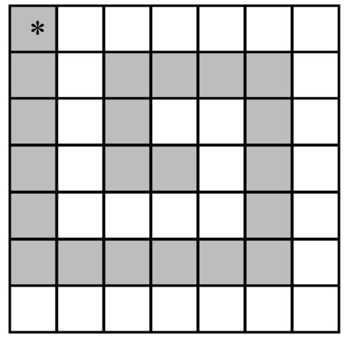

Figure 3b. More complex pattern of faulty regions (shown in white). All non-faulty regions are still succesfully configured.

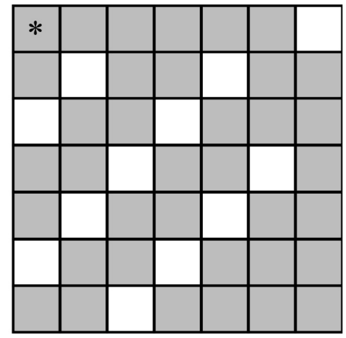

Figure 3c. Even more complex pattern of faulty regions (shown in white). Every row and column contains a fault. Gray regions are still succesfully configured as Supercells.

configuration commands that are used to configure any circuit in the Cell Matrix. Note that if a guardwall has been activated, the commands will not penetrate the guardwall, and thus no configuration will occur in a faulty RUT.

Note also that if the SCS is sent to a set of inputs along an edge of an empty (unconfigured) Cell Matrix, the result will be the configuration of a single Supercell along that edge. Thus, the system can be bootstrapped by simply sending the SCS into an initially empty Cell Matrix.

When a Supercell receives an SCS, it must respond differently depending on the presence or absence of an adjacent Supercell. As just described, if an adjacent region is empty, a new Supercell will be configured there. If, however, an adjacent region already contains a Supercell, then the SCS will be transmitted as simple data to that adjacent Supercell, which will subsequently either configure a new Supercell, or pass on the SCS the yet another Supercell.

In this fashion, the collection of Supercells forms a cooperative broadcast network for transmitting an incoming SCS to the perimeter of the collection. Each time the SCS is broadcast, the collection of Supercells grows, as shown in Figures 2a-2i. Therefore, a tiling of the Cell Matrix with Supercells can be achieved simply by repeatedly sending the SCS into the inputs along an edge of the Cell Matrix. This can be done by adding a counter to the external memory to cause repeated transmission of the SCS, or more simply by storing multiple copies of the SCS inside the external memory.

Note that the resulting tiling may include holes, where regions containing defective cells have been avoided. However, the entire connected set of defect-free regions which contains the initial Supercell will eventually be fully tiled with Supercells, if the SCS is sent enough times. Thus, complex tilings are possible, including concave ones, as shown in Figures 3a-3c.

6. Configuration of the Target Circuit

Once the Cell Matrix has been tiled with Supercells, a GO signal is delivered to the initial Supercell to initiate synthesis of the final target circuit. This GO signal is transmitted to all Supercells using the same broadcast mechanism used for SCS transmission. Inside each Supercell is a state machine that begins operation in response to this GO signal.

To implement the target circuit, two things must be done: first, individual Supercells must be assigned the role of each of the components (gates) of the target circuit (this step is called *node assignment*); and second, connections must be created among those components, based on the topology of the target circuit (this step is called *node wiring* or *path synthesis*).

Recall that in the Supercell approach to self-assembly, the target circuit is not described explicitly, but abstractly as a collection of nodes and connections. This description is referred to as the *genome* of the target circuit. It is a topological representation of the target circuit, independent of where particular nodes may be physically placed in the final configuration. Both node assignment and node wiring are accomplished by the collective action of each Supercell's internal state machine. The behavior of these state machines is governed closely by the target circuit's genome. Figure 4 shows a

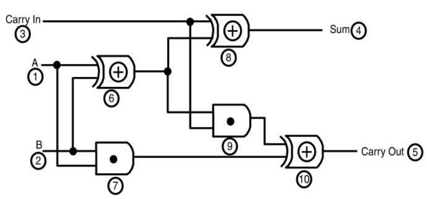

Figure 4a. Sample Target Circuit. This circuit represent a one-bit adder. Assigned node IDs are shown in circles

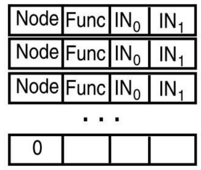

Figure 4b. Basic Genome Structure Node is an integer node ID, Func is an integer describing the node's function, IN0 and IN1 are node numbers of input sources. Node=0 indicates End Of Genome.

1 IN - -
2 IN - -
3 IN - -
4 Out 8 -
5 Out 10 -
6 XOR 1 2
7 AND 1 2
8 XOR 3 6
9 AND 3 6
10 XOR 7 9

Figure 4c. Genome for Sample Target Circuit. IN and OUT are special functions which allow the Supercell to communicate with the outside world.

sample circuit and the corresponding genome for that circuit.

Components within the genome are represented by integers. To perform node assignment, each Supercell must first be assigned a unique integer ID. These IDs will then be used to choose a component from the genome, and the Supercell will then assign itself to act as that component. Note that this requires that the IDs assigned to each Supercell be sequential, so that any ID contained in the genome will correspond to some Supercell. ID assignment occurs in two steps. First, each Supercell self-assigns itself a positional ID, based on its location within an imaginary grid of Supercells (imaginary because some of these Supercells may not exist, due to the presence of faults regions. This ID assignment is similar to the approach used in [10]. Note, however, that in the present work, every Supercell transmits both row and column information to every adjacent Supercell. Therefore, a Supercell which is otherwise isolated on a row can still receive column information from an adjacent row (instead of receiving it from a Supercell on the same row). Additionally, **t**he present work does not rely on failed Supercells **for any part** of the ID assignment process**.** Only non-faulty Supercells participate in the ID assignment. This means, for example, that a failed Supercell does not need to transmit ID information to its neighbors.

This approach to ID assignment is an easy operation to perform. Each Supercell simply consults a neighbor to learn that neighbor's row and column, and then adjusts for its own position relative to that neighbor. The initial Supercell (the first one configured) is marked with a flag when it is configured, indicating that it has a particular row and column position. All other positions are thus calculated relative to the initial Supercell.

While this positional ID is straightforward to compute, it is not directly usable for node assignment, since the assigned IDs depend rigidly on the position of the Supercell in the tiling of the Cell Matrix. Since that tiling may contain holes, it is impossible to guarantee that any particular positional ID will correspond to an existing Supercell. Therefore, there can be no direct correspondence between these positional IDs and the IDs stored in the genome.

However, positional IDs are still very useful, in that they assign an ordering to the Supercells. This ordering can be given, for example, by the formula ID = row + (2^32)*column. Using this ordering, the Supercells can then self-assign **sequential** integer IDs to themselves, by simply assigning consecutive integers to Supercells in the order given by their positional IDs. This self-assignment is performed by the collective action of the Supercells' state machines. In this way, the collection of Supercells are given sequential IDs, that can then be used to choose components from the target circuit's genome. At this point, the set of Supercells is mapped to a set of sequential integers, with no holes in the mapping.

Following assignment of sequential IDs to each Supercell, the Supercells begin the process of node wiring. The goal of this step is to create communication pathways among the components of the final target circuit. Node wiring is accomplished one Supercell at a time, in the order given by their sequential IDs. At each stage, the Supercell that is currently performing the node wiring is called the Master Supercell. Node wiring involves the following steps:

1. The Master Supercell creates a broadcast network among the Supercells. This is actually the same type of network that the initial Supercell built to broadcast the SCS to all other Supercells during the tiling phase.
2. The Master Supercell consults the genome, and determines which component it will implement in the final target circuit.
3. The Master Supercell determines the sequential ID of the Supercell (called the Source Supercell) that generates the first input to that component.
4. The Master Supercell broadcasts a request (using the network built in Step 1) for the Source Supercell's ID.

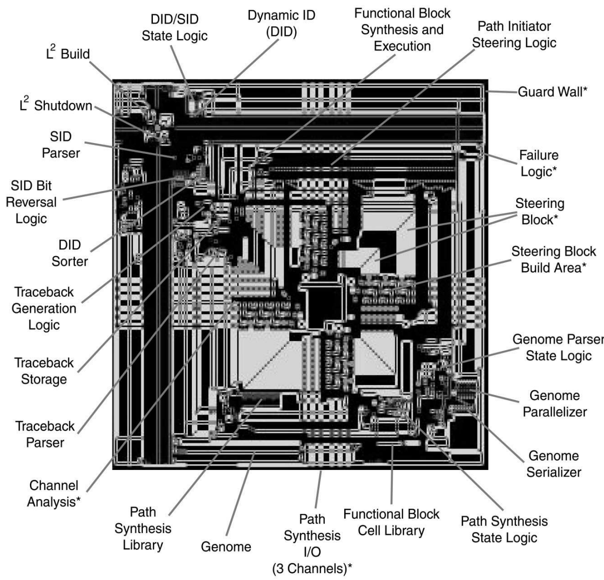

Figure 5. Supercell Block Diagram
Supercell is composed of a 270x270 array of Cell Matrix cells.
Components marked with "*" are present on all four sides of the supercell.

The Source Supercell will respond to that request, transmitting an acknowledgement back to the Master Supercell (again using the network built in Step 1).

5. As the acknowledgement signal is transmitted back to the Master Supercell, routing information is appended to it by the intervening Supercells involved in the transmission.
6. When the Master Supercell receives the acknowledgement, it decodes the appended routing information, thereby determining a path to the Source Supercell
7. The Master Supercell then generates a series of low-level cell configuration commands, that change the configuration of certain cells (in a region called the *Steering Block*) within the Supercells between the Master and Source. The result of this step is the creation of a communication channel from the Source to the Master. Note that this channel only uses cells within the intervening Supercells.

Steps 3-8 are repeated for each required input. Note that this routing algorithm is similar to a greedy algorithm, in that as soon as a path is found, it is used, without regard for future wiring requirements. This may cause the algorithm to fail to route a complete circuit, merely because of the ordering of the elements in the genome. Strategies such as [11], which have similar goals of autonomous routing, allow pieces of paths to be shared among multiple paths, which may relieve some routing congestion.

Following synthesis of all communication channels, the Master Supercell then performs a *differentiation* step. In this step, a small subcircuit (called the *functional block*) within the Supercell is configured to act as the designated component in the final target circuit. Following differentiation, the Master Supercell rescinds control, and the Supercell with the next sequential ID assumes the role of the Master. This process continues, until all wiring indicated by the genome has been achieved. At the conclusion of this process, a DONE signal is generated, indicating that the target circuit has been implemented and is ready for normal operation.

7. Results and Discussion

A prototype Supercell was successfully designed and implemented on the Cell Matrix. Figure 5 shows an annotated picture of this Supercell, taken from the Cell Matrix simulator. This Supercell has the following characteristics:

- The Supercell is composed of 72,900 Cell Matrix cells, in a 270x270 configuration.
- The final target circuit may contain up to 256 Supercells.
- Each Supercell implements a single two-input, one-output functional block.
- Each such functional block is comprised of three Cell Matrix cells, chosen from a library of 20 basic single-cell blocks.
- Two of these three cells may be chosen to receive input from either another Supercell's output, or from an external port.
- The third cell generates the Supercell's output, and is available as an input to other Supercells.
- For purposes of routing a Supercell's output to an input, each intervening Supercell contains three independent paths through its *steering block*. The steering block is a subset of the Supercell, available for building circuits to route data from one side of a Supercell to another side. Having three independent paths allows greater flexibility in routing than having a single path.

Note that none of the above specifications represent hard limits of the technique itself. Rather, the above limits (e.g. 256 Supercells, 20 single-cell blocks, etc.) were chosen for this particular implementation, based on the expected size and nature of the test problems in the current work.

Software tools were also developed, for taking a circuit's genome and creating the necessary Supercell configuration strings to implement that circuit. The design flow is thus fully automated, from genomic circuit description to final strings. Furthermore, the conversion form a HDL description of a circuit to a genome is easily automated, though this was not done in the present work.

This Supercell was thoroughly simulated with a cell-level Cell Matrix simulator, and its behavior was verified to be correct. The ability of the system to detect and isolate faults was tested, by allowing the simulator to treat certain cells as faulty. In all cases, Supercells were able to successfully detect simulated faults in adjacent regions, and guardwalls were successfully activated. These fault detection sequences have also been verified on physical (non-simulated) hardware.

Next, specific target circuits were chosen, and the ability of the system to self-configure under different fault conditions was tested. Test circuits ranged from simple combinatorial logic circuits to more complex sequential circuits, including a counter and a Linear Feedback Shift Register (LFSR).

In each test, a fixed genome was constructed for the target circuit. This genome was inserted into the fixed Supercell design, resulting in a fixed SCS. A set of cells on the Cell Matrix were then chosen to be faulty, and the ability of the system to properly self-configure into the test circuit was checked by supplying the SCS to a set of edge cells. Successful operation of the system was verified under a variety of fault conditions, including isolated failures, large sets of localized failures, concave failure regions, and distributed failures where every Supercell row and column contained a fault. Finally, the configured target circuits were themselves simulated, and shown to function perfectly.

While a Supercell is clearly a large, complex circuit, it ultimately performs a relatively simple function in the final target circuit. As such, it is perhaps more appropriately though of as a type of smart transistor. It can be used as a small-scale building block in a larger circuit, but it is much more powerful than a transistor switch or a simple gate. A Supercell represents a flexible, resilient, and, in some sense, self-aware building block. It is a block that can actively participate in the creation of a larger circuit, despite physical damage to the underlying substrate.

While the size and complexity of a Supercell may make it impractical for use with today's silicon technology, it could play an important role in future atomic-scale technologies, where the cost difference between 72,900 cells and one cell becomes negligible. Supposing that a single Cell Matrix cell contains 1,000 switches, a Supercell would require approximately 73 million switches. While this seems a huge number, consider a fabrication paradigm where a single switch can be created from 1,000 carbon atoms (e.g. Drexler's Rod Logic [12]). In this paradigm, a Supercell would contain 73 billion carbon atoms. Given the fact that 12 grams of carbon contains $6.022\text{x } 10^{23}$ atoms, this means a cube of carbon 1.7 cm on each side could contain over **8 trillion** supercells, and thus could implement circuits containing 100,000 times more switches than today's largest ICs.

On the other hand, it is not necessarily the case that today's ICs (including standard FPGAs) could necessarily scale up to trillion of switches, because of the inevitable presence of faults one would expect in such a large system. The architectural fault isolation of the Cell Matrix itself, coupled with the dynamic fault handling capabilities of Supercells, are both key to implementing working circuits in the presence of such faults.

8. Conclusions

The Supercell approach to self-assembling circuits appears to be a viable technique for implementing circuits in a fault-tolerant fashion with minimal external intervention. The system achieves a working version of a desired target circuit by first tiling the Cell Matrix substrate with a set of Supercells, and then implementing the target circuit using those Supercells. Faulty regions can be avoided at the granularity of a single Supercell, as opposed to entire row or column replacement. Complex tilings of Supercells are also feasible, thereby maximizing the usage of non-faulty hardware.

The present work utilizes the standard Cell Matrix architecture, without any modifications or additions to the architecture. This work uses the self-configurability of the Cell Matrix in a number of ways, including:

- development of test circuits for analysis of a region prior to Supercell synthesis;
- synthesis of new Supercells;
- synthesis of functional blocks within a Supercell to implement a component of the final target circuit; and
- synthesis of communication pathways among Supercells.

The fault tolerance of the system is based on the ability of the system to rebuild itself, by sending a fixed configuration string to an empty Cell Matrix. While, for a given target circuit, the configuration string remains the same, the resulting circuit implementation on the Cell Matrix will vary from run to run, depending on the location of faults in the hardware. Hence, external interaction is minimized, consisting only of transmission of a fixed bitstream to the Cell Matrix.

Finally, the ability of Supercells to autonomously self-configure without external intervention, despite the presence of hardware defects, may make the Supercell technique extremely useful for implementing digital circuits in future atomic-scale technologies, where the densities involved make manufacturing errors inevitable, and the scales involved make manual intervention impractical.

9. Future Work

The Supercells designed in this work were implemented on a two dimensional Cell Matrix. However, future nanotechnologies may be best suited to fabrication of three dimensional circuits. Therefore, it would be interesting to re-implement the Supercell on a three dimensional Cell Matrix. While the basic design would remain the same, the resulting circuit would likely be smaller. Additionally, by embedding the Supercells in three-dimensional space, the task of autonomous routing is simplified, thereby potentially leading to better fault handling.

Another interesting variation would be to eliminate the need for an externally-supplied configuration sequence. Instead, the Supercells could be given the ability to autonomously generate configuration sequences by themselves. This would require merging the Supercell design with previously completed work on self-replicating circuits. The resulting system would be extremely robust, since, in the event of severe damage, a single surviving Supercell would be sufficient to rebuild the entire target circuit.

Finally, adding the ability to Supercells to *detect* system failures would further increase the ability of the system to operate autonomously.

Acknowledgments

This work was funded in part by NASA SBIR Contract NAS2-01049. The authors gratefully acknowledge the support of NASA and Ames Research Center in this work. The authors also acknowledge the helpful comments of the reviewers.

References

[1] J. Miller, M. Hartmann, "Evolving Messy Gates for Fault-Tolerance: Some Preliminary Findings," *Proceedings of 3rd NASA/DoD Workshop on Evolvable Hardware*, D. Keymeulen, J. Lohn, A. Stoica and R. S. Zebulum, eds., July 2001.

[2] 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.

[3] A. Stauffer, D. Mange, G. Tempesti, C. Teuscher, "BioWatch: A Giant Electronic Bio-Inspired Watch," *Proceedings of 3rd NASA/DoD Workshop on Evolvable Hardware*, D. Keymeulen, J. Lohn, A. Stoica and R. S. Zebulum, eds., pgs. 185-192, July 2001.

[4] A. H. Jackson and A. M. Tyrell, "Asynchronous Embryonics," *Proceedings of 3rd NASA/DoD Workshop on Evolvable Hardware*, D. Keymeulen, J. Lohn, A. Stoica and R. S. Zebulum, eds., pgs. 201-210, July 2001.

[5] D. Mange, M. Sipper, A. Stauffer, G. Tempesti (invited paper), "Toward Self-Repairing and Self-Replicating Hardware: The Embryonics Approach," *Proceedings of 2nd NASA/DoD Workshop on Evolvable Hardware*, pgs. 205-214, IEEE Computer Society, Los Alamitos, 2000.

[6] N. Macias, "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)*, D. Keymeulen, J. Lohn and A. Stoica, eds., pgs. 175-180, July 1999.

[7] L. Durbeck and N. Macias, "The Cell Matrix: An Architecture for Nanocomputing," *Nanotechnology 12*, pgs. 217-230, Institute of Physics Publishing Ltd., 2001.

[8] US Patents #5,886,537; #6,222,381; and #6,297,667.

[9] Cell Matrix Corporation Website, http://www.cellmatrix.com

[10] A.H. Jackson, A.M. Tyrrell, "Asynchronous Embryonics with Reconfiguration," *Proceedings of 4th International Conference on Evolvable Systems*, pgs. 88-99, Tokyo, Japan, October 2001

[11] J.M. Moreno, E. Sanchez, J. Cabestany, "An In-System Routing Strategy for Evolvable Hardware Programmable Platforms", *Proceedings of 3rd NASA/DoD Workshop on Evolvable Hardware*, D. Keymeulen, J. Lohn, A. Stoica and R. S. Zebulum, eds., pgs. 157-166, July 2001.

[12] E. Drexler, "Engines of Creation," Anchor Press/ Doubleday, Garden City, NY, 1986.