6-24-2010

Design and control of a cellular architecture-based adaptive wiring manifold
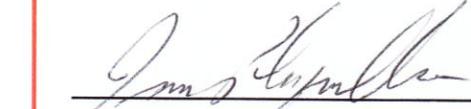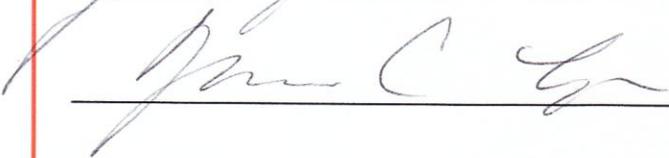
Gregory Feucht

Gregory Alan Feucht
Candidate

Electrical & Computer Engineering
Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dr. Jim Plusquellic,
Chairperson

Dr. Jim Lyke
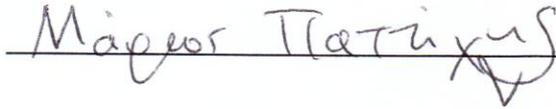
Dr. Marios Pattichis

**DESIGN AND CONTROL OF A CELLULAR ARCHTECTURE-BASED
ADAPTIVE WIRING MANIFOLD**

BY

GREGORY ALAN FEUCHT

**B.S.
APPLIED MATH, ENGINEERING, & PHYSICS
UNIVERSITY OF WISCONSIN - MADISON**

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

Computer Engineering

The University of New Mexico
Albuquerque, New Mexico

May, 2010

DEDICATION

To my wife, Jamesina,

for your continual love and encouragement,

and for which my dream of graduate school

would not have been possible without...

ACKNOWLEDGMENTS

DESIGN AND CONTROL OF A CELLULAR ARCHTECTURE-BASED ADAPTIVE WIRING MANIFOLD

BY

GREGORY ALAN FEUCHT

ABSTRACT OF THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

Computer Engineering

The University of New Mexico
Albuquerque, New Mexico

May, 2010

DESIGN AND CONTROL OF A CELLULAR ARCHTECTURE-BASED

ADAPTIVE WIRING MANIFOLD

by

Gregory Alan Feucht

B.S., Applied Math, Engineering, & Physics, University of Wisconsin - Madison, 2001

ABSTRACT

Existing spacecraft wiring harnesses utilize fixed wiring harness architectures, consisting of either bundles of physical wires, electrical components with backplanes, or motherboard/card arrangements. All such designs are generally configured at manufacture and require significant rework when mission requirements change. A programmable wiring harness is proposed and implemented that, like a field programmable gate array (FPGA), is a pre-built switch fabric that is soft-configured at the time of use, and adapted in real time as components to be wired are added. By providing reversible and dynamically programmable software wires, when embedded in a wiring system, these can be used to build a programmable wiring manifold. The useful properties of this adaptive wiring system include design time reduction by orders of magnitude over traditional wiring harness implementations, the potential of self-healing/diagnostics, and soft-definable probe signals to aid in discovery of component faults. Algorithms used in FPGA routing are exploited to guide the formation of switchable wire paths in the adaptive wiring manifold. A physical system is implemented

in this thesis that demonstrates the concepts of substrate/cell creation, master routing control and graph creation, and wiring commands generated and transmitted to the cell substrate in order to route electrical connections based on gathered netlists of detected components.

TABLE OF CONTENTS

Contents

CHAPTER 1 - INTRODUCTION

Conventional spacecraft wiring harnesses are built with architectures that are fixed at manufacture, though evolving mission requirements often tend to favor systems that can adapt to support the concept of Responsive Space missions. With launch costs exceeding $30,000 per kg, reducing the mass of a spacecrafts wiring harness without compromising reliability is highly desirable [1].

A programmable wiring harness is being built which exhibits qualities similar to a field programmable gate array (FPGA), in that it is a pre-wired architecture that is soft-configured as needed, and ideally within minutes of a defined mission. By providing reversible and dynamically programmable software wires, when embedded in a wiring system, these can be used to build a programmable wiring manifold. Adaptive wiring systems have many useful properties, including the potential of self-healing/diagnostics and soft-definable probe signals. Algorithms used in FPGA and switch routing are exploited to guide the formation of switchable wire paths in the adaptive wiring manifold.

A reconfigurable switch fabric enables dynamic routing of signals in many different applications. Power routing, digital and analog signals, and high frequency transmissions can be routed for space systems, but the adaptive wiring concept can also be applied to terrestrial applications such as aircraft wiring and ground-based systems where the need arises for a solution that can be "wired" within minutes of a defined mission requirement.

The inspiration for a cell-like structure for adaptive wiring comes from the cellular self- assembly found in all living organisms from nature. The parallels to this work seeks to answer the question as to whether a self-organizing system based on cellular elements can be created for the purposes of local communications and global command execution in order to solve a global-oriented task. The cellularity idea applied to the concept of adaptive wiring initially targets the macroscopic scale, sufficiently sized to meet the functional requirements of self-awareness and limited computational ability at the cellular level. Once the cellular building blocks have demonstrated their ability to "self assemble", i.e., once the cells are physically put in place, they can recognize neighbors and perform basic commands relative to this awareness. At this time, the scale of each cellular element can be reduced in order to increase functionality and collective computational ability with the goal of applying the concept to a real-world system.

CHAPTER 2 - CONCEPTUAL ARCHITECTURE

The concept of adaptive wiring is straightforward, and this section describes a number of basic principles for adaptive wiring systems [8]. To introduce the basic idea, an adaptive wiring structure, which we refer to as a *substrate,* contains a number of input/output (I/O) terminals, as depicted in Fig. 1. In the case of adaptive wiring, the termini can be thought of as representing a wiring "problem" (Fig. 1a), in which it is desired to connect termini together bearing the same label (i.e., "A" connecting to "A", etc.). In general, the wiring problem can be referred to as a *netlist specification*. In this depiction, the actual adaptive wiring system is represented as a "cloud" within the substrate. A number of possibly non-unique solutions might exist to solve a particular netlist specification, as depicted in Fig. 2. The "cloud" of course notionally suggests that the problem is solved without detailing either how the interior fabric within the substrate might be defined or how the desired configuration is gleaned.
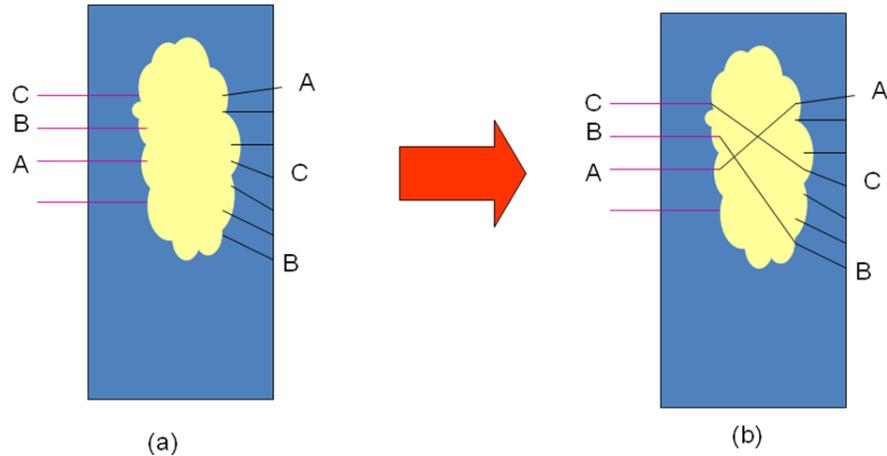
Fig. 1. Basic concept of adaptive wiring. (a) Substrate having a number of exterior termini defining a wiring "problem". (b) Depiction of wiring solution (divorced from implementation details).

Fig. 2 depicts a notional implementation concept to provide some intuition about how an adaptive wiring system might actually be implemented. In this case, the substrate takes on the aspect of a physical panel featuring four sockets where components or "modules" can be mounted (Fig. 2a). To implement the amorphous "cloud" of wiring resources in Fig. 1, we depict here a deliberate configuration consisting of a matrix of wires in rows and columns, with circles shown at the intersection points. The circles represent electrical switches that, when closed, short together the associated row and column. At this moment, we are not concerned over the specific medium for switches. They could be, for example, metallic relays, solid state switches, microelectromechanical systems (MEMS) devices, or combinations of these and other switch types [2]. Using such fabrics, implementing a solution to a particular wiring netlist amounts to closing a number of switches, as shown in Fig. 2b, which shows how a two-netlist problem (involving two placed modules) might be solved (through a total of six switch closures).

4
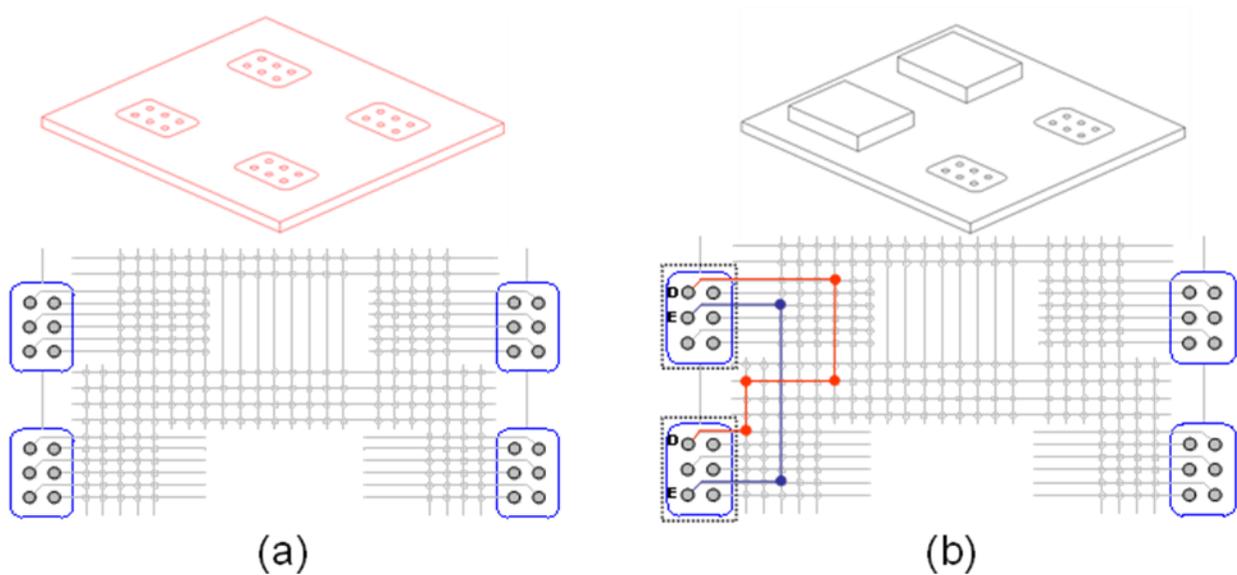
Fig. 2. A notional physical embodiment. (a) Unprogrammed substrate, containing a number of sockets for components (modules). (b) Example placement of two modules and wiring of a two-net netlist.

To permit scaling these implementations, we could consider a number of substrates that could themselves be tiled or otherwise connected together, as depicted in Fig. 3. In this case, two adaptive wiring substrates or "cells" form an extended system. Clearly, a netlist solution involves the definition of two compatible sub-solutions, one implemented in each cell, to generate the necessary overall connected paths between the termini of the source wiring problem (netlist specification).
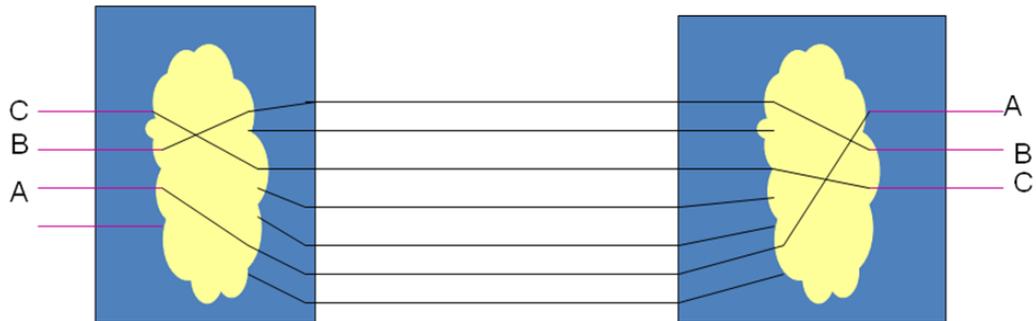
Fig. 3. Extended adaptive wiring system contains two "cells".

As previously discussed, adaptive wiring manifolds offer a number of benefits in developing new systems. Since the adaptive wiring substrates (or panels) may be pre-built and inventoried until use, it is possible to retrieve them as needed and configure them on demand. Rather than wait for custom-defined wiring harnesses to be developed and delivered, a process that could take weeks or months, the adaptive versions can be configured very quickly. Unlike custom wiring harnesses, whose wiring pattern is permanently locked in, adaptive panels can be altered as needed to accommodate late-point changes. Adaptive wiring systems furthermore, have two powerful benefits that are impossible in any other wiring technology.

One advantage of this architecture is the ability to adapt to faults that occur after a system is placed in the field. Since wiring patterns can be software-definable, defects can conceivably be circumlocuted by computing an alternate configuration. This concept is illustrated in Fig. 4. In this case, a faulty connection between B-B can be rectified by configuring other wiring resources that can achieve an equivalent connection without

6

removing a system from the field (which is often impractical, as is the case for space

systems).
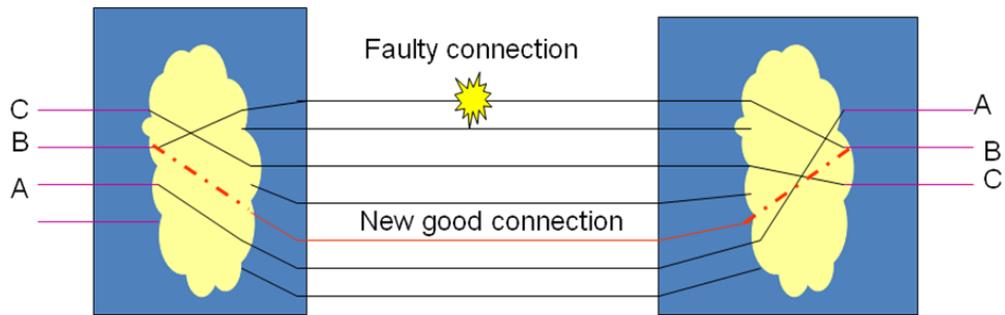
Fig. 4. Demonstration of fault/defect tolerance.

The second unique advantage of adaptive wiring systems is the ability to form

probe connections for diagnostic and maintenance purposes. Temporary probes can be

inserted at normally inaccessible buried nodes within a wiring system and "dissolved"

when no longer needed. This concept is depicted in Fig. 5. In this case, we use the

adaptive wiring system to set up a temporary connection to check a possible problem with

terminal C on the right panel.
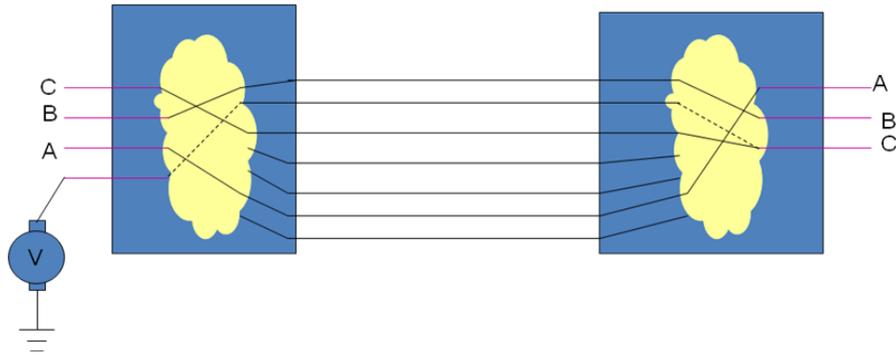
Fig. 5. Demonstration of temporary probe formation.

The techniques demonstrated in Fig. 4 and Fig. 5 can be combined with algorithms to form a self-healing system. Self-healing, as an "active" concept, can be viewed as having two phases, the first being diagnostic, the second being restorative. Clearly, the use of temporary probes can serve to probe an adaptive wiring system, even in situ, to explore the continuity of wiring resources. Upon discovery of defects, an algorithm in the "live" system can compute an alternative wiring path. In earlier AFRL-sponsored work [1], we learned it was possible to achieve self-healing as a linear-time process in an active system. To see this, consider that diagnostics can be implemented on live networks by forming first a redundant path connection to a given node pair, then disconnecting the original path for the purpose of continuity probing. If the original path is found defective, the defect is noted and the defective resources are not re-used in the future. The originally redundant connection remains in place as the new primary connection. Clearly, it is never necessary to restore the primary connections after probing, so this approach implements self-healing as a linear time procedure (proportion

8

to the number of switch resources) that is based simply on refreshing all system wiring

configurations for the purposes of probing them at a repetition interval chosen under

system-wide control.

CHAPTER 3 – EXISTING TECHNOLOGIES, AND PREVIOUS EFFORTS
TOWARDS RECONFIGURABLE WIRING

Traditional Spacecraft Wiring Harness Implementations

Standard wiring harnesses for both terrestrial and non-terrestrial applications typically rely on a fixed wiring architecture. Wiring harnesses typically found in both automotive, industrial, and terrestrial aerospace disciplines rely on bundles of physical cables. These arrangements are typically avoided when possible in space because of both weight and reliability. [10] Aside from physical cable bundles, two approaches are commonly used for satellite wiring architectures: 1) Motherboard/daughterboard configurations, and 2) Card frame with passive backplane. In the first instance, one or several daughter boards plug directly into a motherboard, and active electronics would possibly be employed on the motherboard itself. In the second instance, each card slides into a backplane where typically only power would be routed to the cards. In both instances, the need for physical wiring is eliminated, thereby increasing reliability of the system. The layout for interconnects in this arrangement is primarily fixed at manufacture. Redundancy in connections for reliability concerns must therefore be engineered into the system from the beginning of the design effort, and any changes to wire routing if system requirements change can be time consuming and costly. Changes to wire routing once the system is deployed are near impossible, due to the fixed architecture.

Review of previous work in reconfigurable wiring

Several groups have built grids of FPGAs, but work in designing or implementing an electrical wiring manifold with full reconfigurable capabilities has not been shown to have successful implementations as of this writing. Large collections of work exist in routing algorithms specific to within FPGAs.

A study performed in 1999 [5] by S. Hauk introduces a mesh topology grouping of FPGAs. The systems studied in the work consider the effects of different "cell" connection schemes in a 2-D grid with delay of digital signals and internal routing resources utilized. Variations on the traditional mesh interconnects are specifically studied (e.g., variations on cell connection schemes where simply N, E, S, and W neighbors are connected.) For example, effects of delay are studied on also connecting NE, SE, SW, and NW "kitty corner" neighbors, and also connecting two nearest neighbors. Significant speedup is realized in processing capability in both of these studied cases, and as such should be considerations for future interconnect study within the AWM concept. It should be noted, however, that the interconnect structure for routing of digital only signals.

The substantial work of A. DeHon in exploring the trade-offs between computational elements and interconnect count and bandwidth underscores the important design task required in implementing the concept of adaptive wiring efficiently in terms of numbers of nodes/wires to share between cells, determining the size and number of cells to implement in a given area, relative to the performance capability of each cell.

DeHon shows in [3] that in an FPGA network, that wires and interconnect area dominate the overall area of a computational design. It should be noted here that this initial design study of adaptive wiring does not necessarily seek to maximize the computational capacity of a cellular grid of either microcontrollers or FPGAs, but rather seeks to first prove the concept that a wiring structure can first be built that is inherently plug and play, gives the capability for fault tolerance, and that each cellular element can perform a level of independent computations. The detailed study of interconnect size and quantity as it pertains to determining the interconnect allocation to various subnets is indeed an important area of future study for this concept.

The work of Thompson and Mycroft ([9], [10], [11]) has yielded significant progress in the area of both the design of reconfigurable wiring and inherent fault-tolerance methods. Advances in the concept of adaptive wiring, including routing different signal types to corresponding subnets, fault tolerance studies, and the general crossbar switch arrangement were all studied in these works. It is in fact part of the goal of this work to create a physical implementation based on the crossbar concept presented in these works.

CHAPTER 4 – SCALABLE CELLULAR IMPLEMENTATION OF ADAPTIVE
WIRING MANIFOLDS

In this Chapter, we discuss a particular wiring manifold approach that supports a

number of useful features, as well as implementing the necessary infrastructure for

controlling both local (cellular) and global configurations, the automatic detection of

cellular boundaries, and the discovery of placed components.

The Adaptive Wiring Panel (AWP) is a manifold of adaptive wiring cells cast as a

single overall panel. The panel is a pegboard-like structure, which (unlike Fig. 2), do not

articulate specific sockets, but rather provides a continuous grid of contact pads and

mechanical mounting holes. The AWP implementation is based on three basic elements:

(i) cell units, (ii) a cell management unit and (iii) modules. Cell units are defined as the

minimum independent unit of the AWP substrate, all with interconnections and

communications with other cells, and form the switch fabric by which we wire

components to each other. The Cell Management Unit communicates independently with

all cells and manages the wiring path and switch connections of the panel. Finally,

modules are the "widgets" that make up components to be wired. Examples include

power supplies, gyros, thermisters, electrical components such as capacitors, resistors,

etc.

Fig. 6 depicts the three layers of the adaptive wiring implementation. The top

layer, or substrate, provides an interface plane upon which modules are connected to. The

middle layer contains a PCB dedicated for relays and routing of signals, while the bottom

layer provides logic, a communication hub for global and neighbor interactions, and

facilitates switch commands to be sent to the relay board.

Description of connections:

Between external pins and relays.

Between local signals and neighbor signals.

Between relays and the FPGA.

Between cells (I2C).

Between the cell and the master.

Between the probe pin and the FPGA.

Top layer: Pins

Middle layer: Relays

Bottom layer: FPGA

FPGA
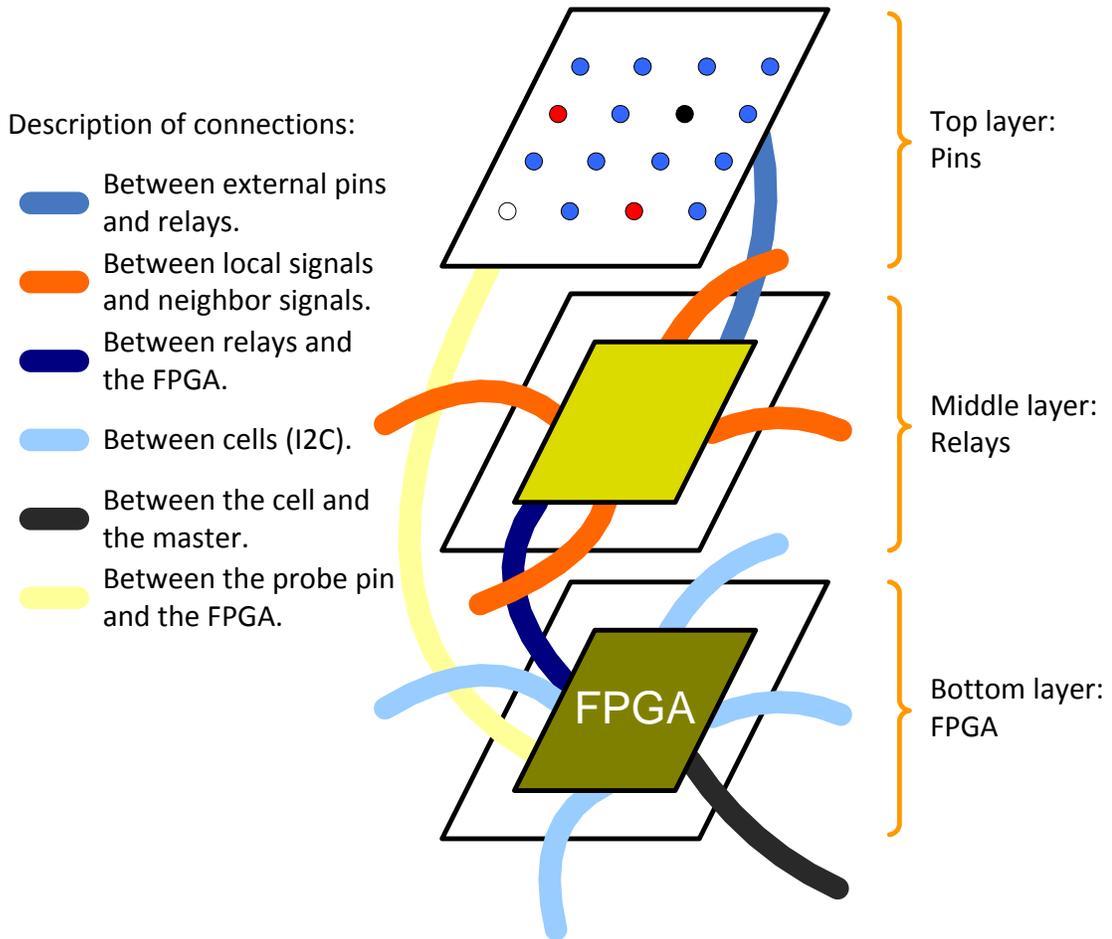
Fig. 6.   Adaptive Wiring Manifold Layers.

In the next sub-sections, we describe the high-level functionality of these

components.

Cell Unit

Each cell unit (CU) is the minimum independent unit to create the AWP. For the primary design study, each CU measures 5x5cm. However, for initial testing purposes, the cell enclosure has dimensions of approximately 5 by 7 inches. In Fig. 7 we show the planned top view of a CU. This block has: (i) 12 signal connectors, (ii) 3 power connections (one is a fixed connection to GND), and (iii) 1 mechanical connector.
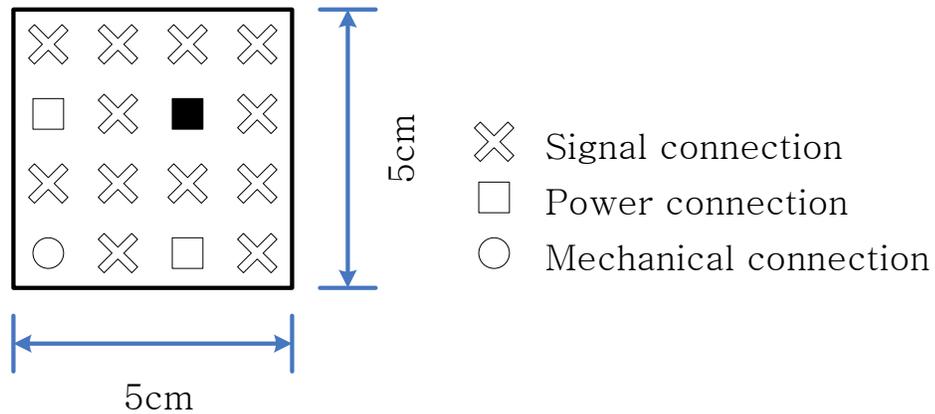
Fig. 7.   Cell block. Each cell block has 16 connectors: 12 for signals, 3 for power and 1 for mechanical connections.

For the initial version of the AWP, each CU is implemented using a Xilinx Spartan 3 FPGA board. (The embedded Microblaze microprocessor is not used for the present design, but could be utilized for additional functionality at a future time.) The functions of each CU are:

1) Control the programmable connections of the AWP.

2) Communication with (up to) four neighbors: each CU needs to communicate with its physical neighbors to recognize spatial orientations. This is performed by implementing independent state machines and I^2C master/slave arrangements within each cell that correspond to managing a communication bus respective to each neighbor. Though the I^2C protocol has the ability to arbitrate in a multi-master arrangement, it was decided (for ease of initial implementation) to assign "master" status to communications between cells that originate to the North and East directions, and slave status for South and West directions. In this manner, when cells are connected to each other, communication between cells will only act between a master and slave. Fig. 8 shows the continuous contact substrate created by joining multiple cells together, and their index identification. The intent of this arrangement is to allow a module to be plugged into a cell in any 90 degree orientation (for purposes of space/packaging) and allow the module's pins to always align with contact pads on the cell substrate, even for the case where a single module's pins traverse several cells.

3) Read Electronic Data Sheets from modules via probe pins: each module has a probe pin (to be described in the next sub-section) that transmits its module-ID and the information about required connections to other modules on the AWP.

4) A low current power supply is used to power the modules to enable transmission of electronic data sheet information through predefined probe pins.

5) Communication with the cell management unit: each cell unit transmits and receives information to/from the cell management unit (i.e., cell units aside from neighbor recognition cannot communicate directly with each other.) Each cell block, upon system power-up, will send identification information such as ID of the CU, the IDs of its neighbors and relative orientations, and module Electronic Data Sheets, if connected to that CU.
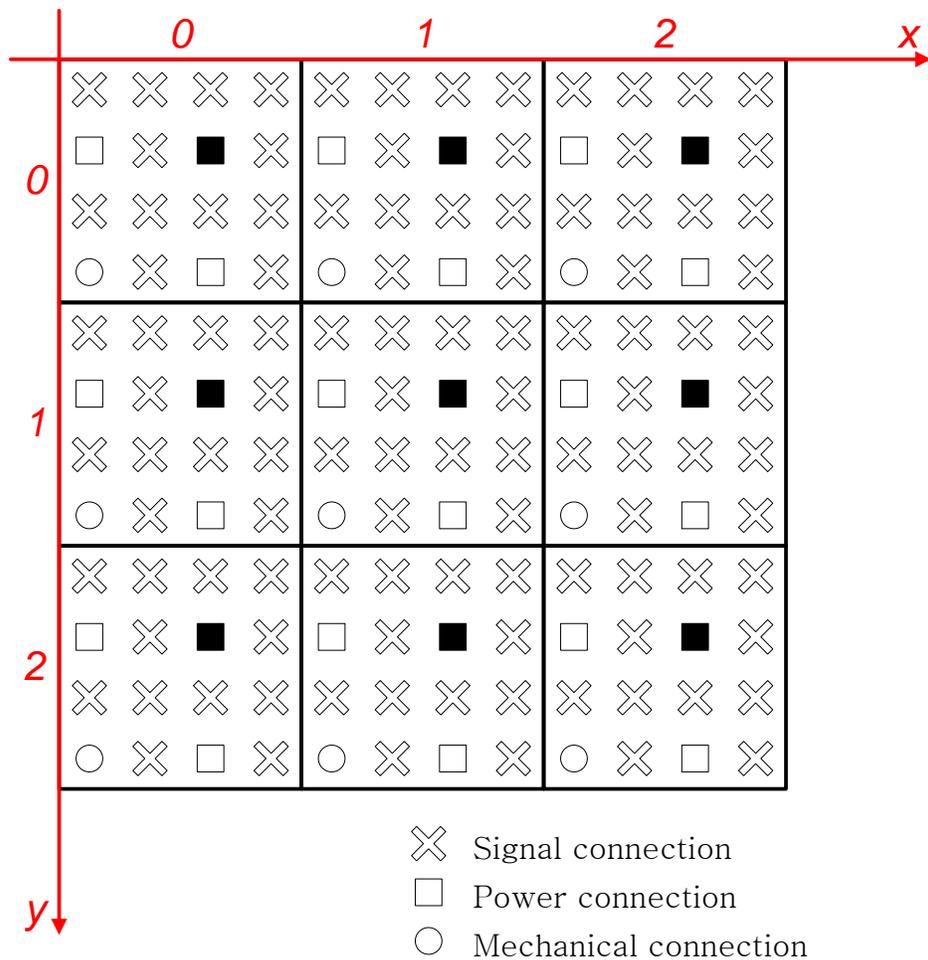

Figure 8. Adaptive wiring panel built by connecting independent CUs. Scale: 3:10. The labeling of each CU is performed according to the top-left CU labeled as CU(0,0).

Each CU will have three layers, as described above: (i) top layer for pins to connect the CU with the modules, (ii) middle layer with the array of relays and (iii) bottom layer with the FPGA to control the system.

The node/switch connections are graphically represented via an architecture that closely resembles a crossbar. In order to route a particular input to a given output, all that is necessary is for the switch corresponding that input and output to be closed. In order to reduce the number of switches used (compared to the "brute force" approach where a switch exists at every wire crossing), a heuristically-chosen subset of switches are used. This is shown in Fig. 9.

In terms of optimal use of hardware, it should be acknowledged that utilizing a crossbar arrangement is relative inefficient. For example, it is possible to route an input to any number of outputs, even though this is almost never necessary. However, the ability to route alternate paths in the event that a switch or node/wire is determined to be damaged is straightforward. In this manner, single point failures can be remedied as soon as the fault is detected. Redundant connections for a given input/output pair are therefore built into the architecture, and additional wiring or hardware dedicated to an I/O pair is not necessary.

Cell Management Unit

The cell management unit (CMU) manages communications and performs routing configurations of all cells on the AWP. An architectural diagram is shown in Fig. 7.

The functions of the CMU are:

18

1. Manage the communication bus: I^2C is the chosen communication bus between the CUs and the CMU, and the CMU serves in single master mode. Additionally, I^2C is also the communication protocol between adjacent cells, and also between cells and connected modules to the grid, details of which are discussed in the sections respective to those components.

2. Read and organize initial location/orientation information from the CUs.

3. On a periodic basis, scan for IDs of any modules connected to a particular CU, and read module Electronic Data Sheets.

4. Compute the shortest path and routing of required signal connections: the CMU has a priori knowledge of the layout of relays in each cell block. This is achieved by reading in a pre-established switch combination spreadsheet, with the necessary information for mapping graph nodes (wires within a subgraph/cell) with the location of paths (i.e., switches) that allow the connection from one node to another. When the geometry of the AWP is created, a complete graph related to the AWP and the connections are built by merging the subgraphs of each individual cell. (The details of this procedure are described in the following chapter.) After reading the module Electronic Data Sheets (with information of the netlists required to generate a functional circuit) containing required switch connections, the CMU computes the shortest path to connect corresponding terminals on the modules.

5. Command cell blocks to open or close specific relays.
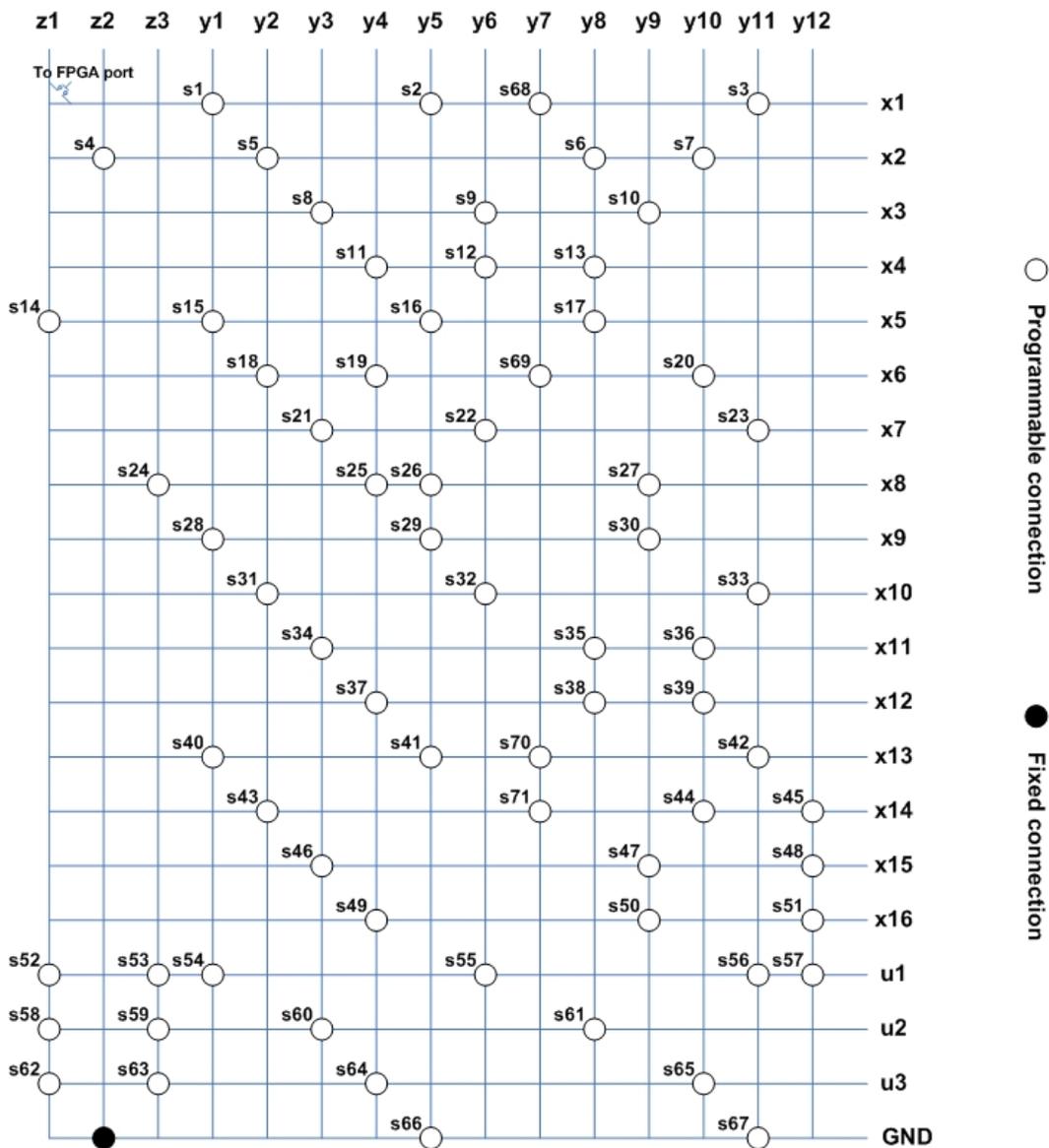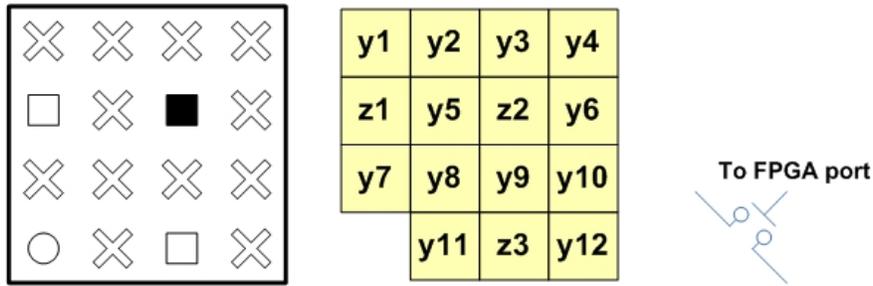
Fig. 9 Single cell crossbar-like arrangement of switch locations. Y and Z signals (vertical wires shown here) represent paths to module pins; X, U, and GND signals (horizontal wires) represent connections to adjacent neighbor cells.

20
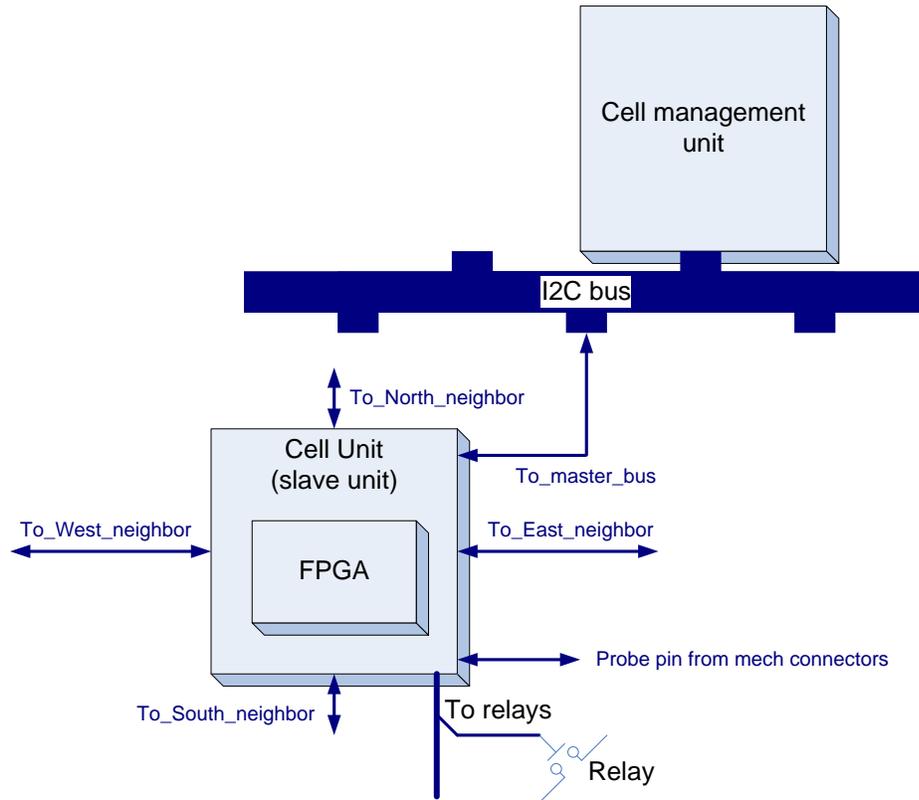
Fig. 10   Cell Management Unit (CMU) with a typical connection to a Cell Unit.

Dijkstra's Algorithm

Dijkstra's algorithm was chosen to compute the minimum shortest path between nodes that require connections. It solves the single-source shortest-path problem on a weighted, directed graph G = (V,E) for the case in which all edge weights are nonnegative. [11] The actual speed of the algorithm can vary from $O(n*lg(lg(n)))$ to $O(n^2)$. The basic algorithm is relatively simple:

1. Take the length of the shortest path to every node (whether directly connected or not) be infinity. Mark the length of the shortest path to the source node as "0". Since we know that no edges, or weights, can be negative, we know that the path to the source is "0", therefore it's trivially optimal.

2. The algorithm works by making sure that the path to one more node is optimal at each step of the algorithm. Therefore, at the nth step, we know that we have n paths to n nodes that we can guarantee are optimal.

3. At each step, seek to optimize the path to every node which connected to the present node. If a shorter path is found, replace the value at that node with the new (shorter) value.

4. Repeat each step until all nodes are visited.

This algorithm, therefore, produces a list of minimum path lengths from a designated source node to all other nodes in the graph, and is a good starting point for purposes of mapping routes from a designated node, or in the case of the AWM, pin corresponding to a module to be connected, to a destination pin likely at another module. The next section describes the modifications required to the algorithm to support both 1) path storage, and 2) enabling of multiple routes on the same graph without using overlapping resources.

Modifications to Dijkstra's

We require storage of the shortest path between a source and destination node in order to send commands to the cells of which switches to close for purposes of recreating this path in the wiring substrate. To achieve this, know that for every node n, we need to

track the distance from source to every node. As noted above, this is achieved by

calculating the new distance and comparing with the previously shortest distance, and

replacing if the new path is shorter. When this is achieved we track the previous node in

the path to *n* in a separate array *prev[n]*. This array is updated every time a new shortest

path is found. Finally, in order to recall the entire path returning to the source, a recursive

function is employed.

The other critical modification required is to allow the ability of routing multiple

signals on the same graph without overlapping node/wires and switches. To perform this,

node weights are subsequently assigned to routes that are already have paths designated to

a previous wiring requirement. By this method, when an additional route is added and the

algorithm seeks to use some of the same node/wire resources to route a new signal,

Dijkstra's algorithm simply sees that this node has a new (arbitrarily high) weight

corresponding to a node that is now no longer optimal in terms of a shortest path, and the

next shortest path is then chosen.

Relevant source code for Dijkstra's algorithm and other critical functions are

listed in Appendix A of this document.

Modules

Modules (the components to be wired) have an interface to the cells with a minimum of 5x10 cm with 24 signals connectors , 6 power connectors and 2 mechanical connectors (not all of which need be connected). The system is designed with the capability to route signals corresponding to separate subnets such as power, ground, analog transmission, and high frequency signals independently. Fig. 11 shows the contact orientation of a planned module. The contacts for a module will traverse at least two cells – modules with higher I/O requirements may span more than two cells.
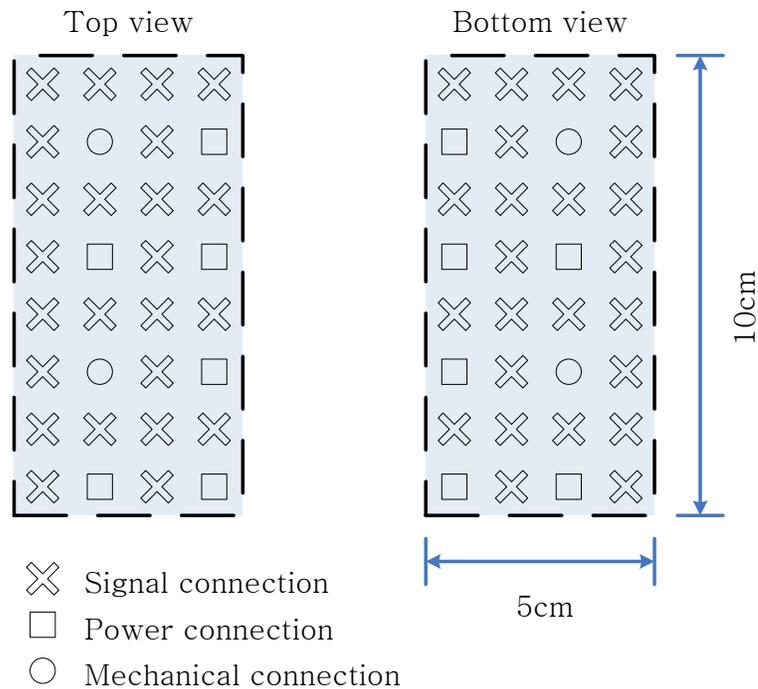
Figure 11. Module pinout interface. Top view and bottom view is shown above of a module of minimum size of 2 cells. It is important to note that not all signal connections are necessarily populated – simply the minimum number required to support component circuitry.

Modules can be arranged on the grid in any orientation. An example of three

simple circuit elements is shown in Fig. 12. The inherent design is such that tight spacing

constraints can be met, so long as module communication pins are aligned on the grid.

Details of this pin arrangement are described in the next section.
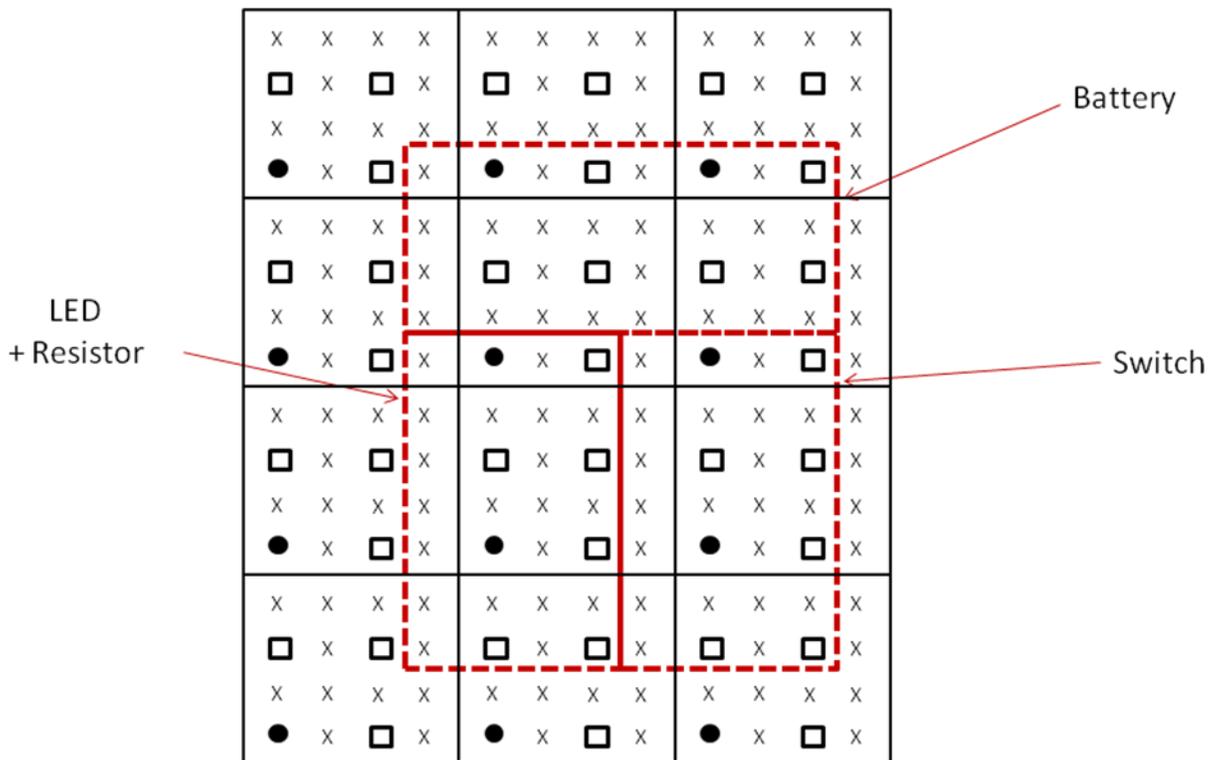
Figure 13. Example of three circuit elements placed on the cell substrate. Note that each module traverses a minimum of two probe pins (depicted above by circles), which helps to form better mechanical connections between module and substrate material.

Module Probe Connector

The requirements for connecting to a module are the following:

1. Four connections are required for initial module communications: (V_power, GND, I2C_SCL, and I2C_SDA).

2. Orientation of Module must be determined for purposes of pin location determination.

As a result, it was determined that the design represented in Fig. 13 satisfies both requirements. The pin connector design from the module side is such that a .100" pitch 4x4 header pin is inserted into the cell socket. From the module side, only four out of the 16 pins are populated. However, the cell VHDL logic is constantly polling for responses from all four of the SDA pins on the connector. When a module connects via this connector arrangement, only one SDA/SCL signal combination will provide a response (i.e., the combination corresponding to a populated pin on the module side, which depends on module orientation). Module orientation information is then transmitted to the master controller when requested.

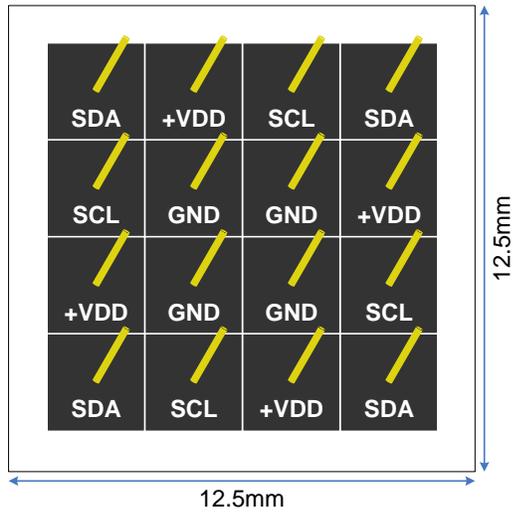**Figure 13. Mechanical and I²C connector for each CU. The connector has 16 pins to be plugged in with the probe pin of the module to be connected. Depending on the orientation of the module, the information is received through a specific set of I²C pins.**

CHAPTER 5 – DEMONSTRATION OF INITIAL ADAPTIVE WIRING MANIFOLD

PROTOTYPE

An initial implementation of the cell management unit was achieved with a
software application written in C using Microsoft Visual Studio. The C programming
language was chosen for its easy interfacing ability with the I^2C protocol, and ease of
implementation of Dijkstra's shortest path algorithm. The application runs on a laptop
running the Windows 7 operating system. Pictures of the initial prototype are shown in
Fig. 14, 15, and 16.

Cell hardware layout

To implement the cell architecture depicted in Fig. 6 shown earlier in this paper,
we choose to separate the FPGA board layer from the PCB containing all switches. A
total of 72 General Purpose I/O pins are routed via ribbon cable to a custom assembled
relay board; each pin corresponding to one switch on the relay board. Picture representing
the physical layout are shown below.

I^2C Interface

I^2C messages are transmitted from the program to the wiring panel via a custom
library and device handle to a Devasys I2C-USB interface board [4]. Several I^2C
interface options were evaluated during the course of development, including build-in I^2C
drivers inherent in an Intel-based Macintosh laptop, and also external hardware
interfacing with both a Diolan I2C-USB, Arduino microprocessor board (with built-in I^2C

28

capability), and I^{2}C signals built into a Apple Macintosh VGA out port . The Devasys
board has the necessary drivers and hardware necessary to serve both read and write
requests as required by the program, and was shown to be easy to implement read and
write commands as requested by the program. Fig. 14 shows a picture of the board
including the required interface connections, and housed in an enclosure which also
supplies power to all individual cells.
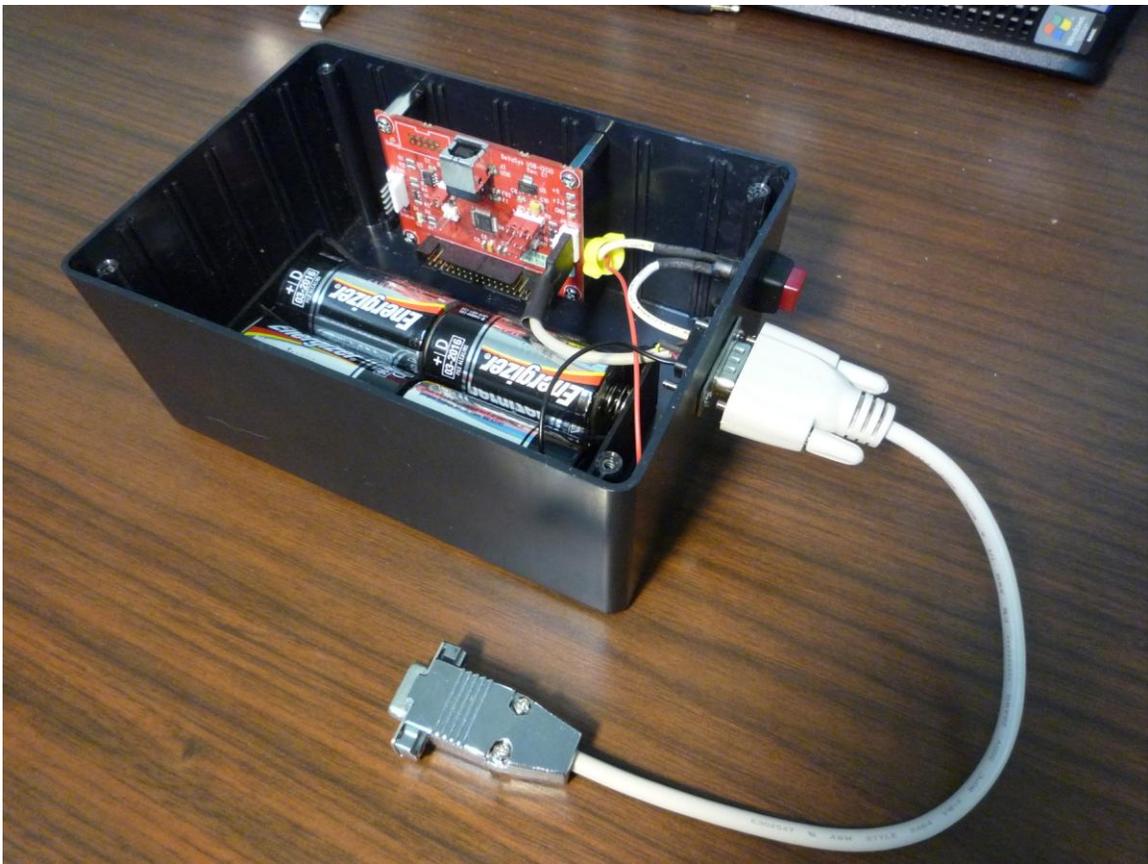
**Fig. 14 Power and communications box. Included is a power supply for all connected cells, and the
Devasys I2C-USB board.**

Once initial communication was established between master controller and slave device, debugging of the I^{2}C message handling for both master and slave devices was achieved with an I^{2}C packet analyzer, manufactured by Saleae Logic.

A first manifestation of cells was created and successfully tested. Switch command execution, I^{2}C communications between cells and to attached modules, and cell management unit communication was achieved, albeit in a larger form factor for both cells and modules than was originally anticipated, as described earlier. In this version, specific sockets were created for interfacing with modules, as opposed to the continuous grid of contacts that was envisaged and described in previous sections. Fig. 15 shows the bottom two layers of a single cell assembly.

Fig. 15 Initial prototype of one Cell. 25 pin D-sub connectors are used for cell to cell connections. The bottom board layer is a Spartan 3 Board, and middle layer the custom designed relays board.

Cell Grid Assembly

Communication generated from the master controller is via a sequence of encoded 7 bit (with one bit reserved for read/write) I^{2}C commands, as detailed later in this section. Upon initial power-up of the system, the initial objective is to build a grid of connected cells. The system needs to identify all cells on the grid, and all immediate neighbor information for each cell in order to construct the 2-D map. The master code initially polls for a response of a cell corresponding to the lowest addressed cell. For example, if ten cells are connected to the grid, and the cell with the lowest address reads 0x09, then the master will first recognize this cell as being the initial cell added to the master graph. Proceeding this, the master submits another read request to this cell for its neighboring information (stored in a separate register in the cell memory), and if a neighbor exists, the master adds cells to the graph one by one, starting with a North neighbor (then East, South, and West). The master maintains a list of both cells corresponding to a global cell identifier being added to the list, and also an array of cells yet to visit in order to poll for neighbor information. In this fashion, a breadth first search of adjacent cells commences until:

1) all neighbors of identified cells have been checked, and

2) all identified neighbors have been added to the master graph.

Implementation of cell connections is achieved through wires connected from the Spartan 3 board to D-sub connectors located on each side of the cell enclosures. Fig. 16

shows two cells connected together, and to a power and communication enclosure for

master I^{2}C commands.
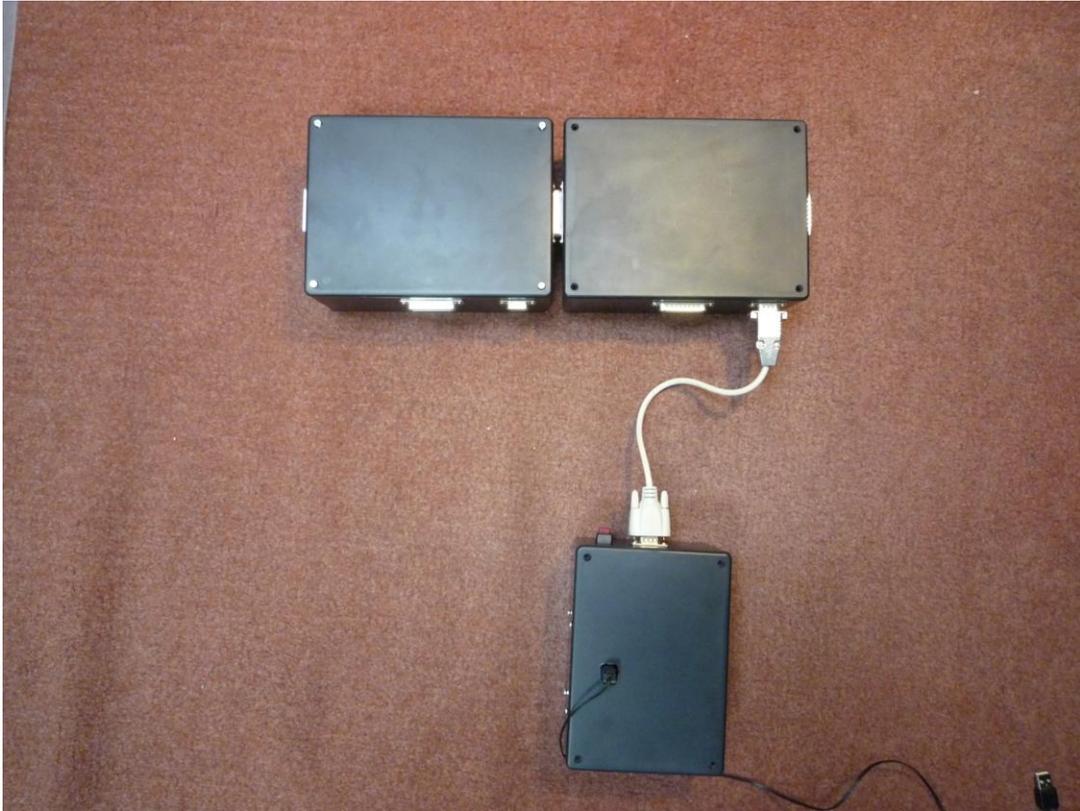
Fig. 16 Example of multiple cells joined together. The bottom enclosure is the communication and power enclosure. The USB connection leads to a laptop which runs the cell management unit.

Joining of Subgraphs

When two cells are joined to a graph, the selection of nodes to join between

subgraphs corresponding to two cells will depend on the two cells relative direction as

determined by neighbor reporting. For example, in the case that we have just one cell

added to the master graph, and we wish to join another cell to the East, the nodes X9,

X10, X11 and X12 is joined to East cell nodes X1, X2, X3 and X4. Fig. 17 shows an example subgraph connection scheme for node joining patterns in a 2 cell arrangement.

Module Signal Routing Commands

When a module is detected on the cell grid, its datasheet and netlist is transmitted and stored in the master controller. The netlist information is stored in a struct identified by the location of the cell that the module attaches to, and on a periodic basis, the program checks for the ability to connect all components together described by this netlist. For example, in the instance that an LED module is attached to the grid, the netlist information will contain information that one lead should attach to a 1K resistor, and the other lead to the negative terminal of a 5V DC power supply. The system scans all other identified modules for the existence of these nets, and if they exist, determines the shortest route between those two nodes. Once the route has been determined, the system sends out individual commands to each cell dictating what switches to close in order to route the appropriate signal.
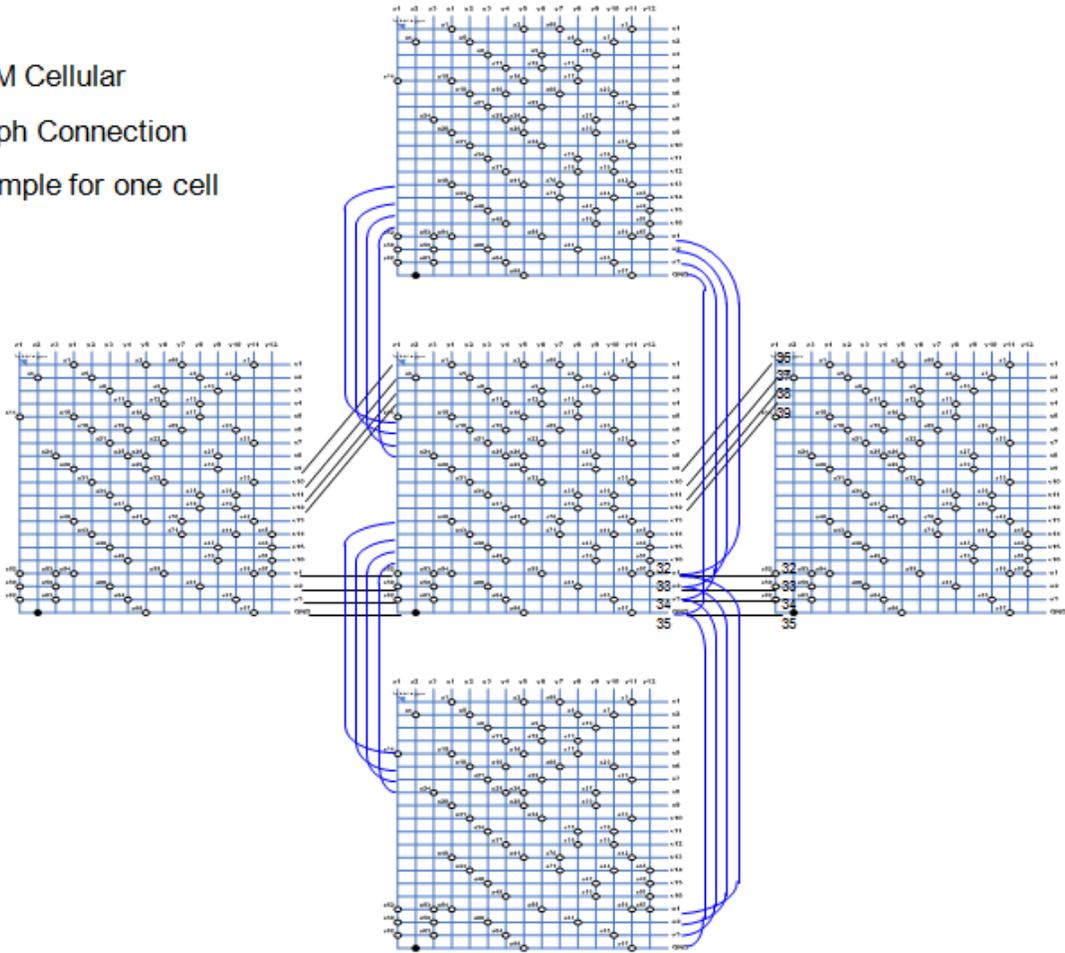
Fig. 17   Required node/wire connections for center cell to neighbors.

Commands for Master Communication

Below is a starting list of communication commands required by the master controller:

- Master addresses all slaves/cells on network via individual global ID calls (0x00 to 0x7f) with 2s period. Read request and address request of 0x44, which is

34

arbitrary address corresponding to global address and neighbor request. Slave

responds with Global ID of cell.

- MASTER COMMUNICATION STRUCTURE
  - Master addresses all slaves/cells on network via individual address calls

    with 2s periodicity, with read request and address defined in Fig. 18.

Master Command	Slave Response
0x44, read request	Send Cell ID
0x45, read request	Send N Neighbor Cell ID. 0x01 if not existing
0x46, read request	Send E Neighbor Cell ID. 0x01 if not existing
0x47, read request	Send S Neighbor Cell ID. 0x01 if not existing
0x48, read request	Send W Neighbor Cell ID. 0x01 if not existing
0x49, read request	Send Module ID byte. 0x01 if not existing
0x50, read request	Send # Bytes Netlist Info for Module
0x51, read request	Send Module Config Byte 1
0x52, read request	Send Module Config Byte 2
0x53, read request	Send Module Config Byte 3
0x54, read request	Send Module Config Byte 4
0x55, read request	Send Module Config Byte 5
0x56, read request	Send Module Config Byte 6
0x57, read request	Send Module Config Byte 7
0x58, read request	Send Module Config Byte 8
0x59, read request	Send Module Config Byte 9

Fig. 18   Partial Master read command list.

Master to Slave Commands

- Message structure is:

    o Byte 1: Global address identifier of cell.

    o Byte 2: Cell Command

    o Byte 3: Cell Operand (if defined)

- Enable Switch

    o Cell command = 0x33

    o Then number if switches to close = 0x**

    o Then index of switch identifiers = 0x**, 0x**, 0x**, 0x**

From Slave to Master

- Who I am (ID)

- Who my 4 neighbors are (4 IDs)

- Module information:

    o Orientation

    o Characteristics (type and number of internal components, e.g., LED)

      o   Location of internal components

From Slave to Slave

- Periodic Request for Adjacent Cell ID via I^{2}C

Module Database

- Send one byte corresponding to a module ID. The master controller stores the

  netlist with required connections via a struct, which it looks up when a cell reports

  an attached module ID, as shown by example in Fig. 19.

Module ID	Component
0x02	LED
0x03	1k resistor
0x04	5V DC Power Supply

.
.
.

Fig. 19   Required node/wire connections for center cell to neighbors.

CHAPTER 6 – CONCLUSIONS, AND SIGNIFICANT FUTURE WORK TO COME

Summary of Completed Work

This paper has presented an introduction, motivation, and a high-level architecture of the Adaptive Wiring Panel, and described the interactions required between modules, cells, and the Cell Management Unit. *Moreover, the work of this thesis has culminated in a working prototype of an adaptive wiring manifold implementation.* This requires graphing algorithms that intelligently program switches to wire the required module connections. Dijkstra's shortest path algorithm provides a clearly workable initial solution for connecting modules in the sparse environment of the adaptive wiring panel. Throughout this writing, reference is made to a specific size of unit cell dimensions, and also interface dimensions for a module. Though a specific implementation of these dimensions was not performed as of this writing, a working prototype with the precise conceptual framework was designed, built, and successfully tested.

Immediate future work

Five major objectives can best summarize where work on a high-impact prototype can be focused:

1. Reduce scale of cell down to described 5 x 5 cm size. These dimensions were originally chosen as they are the defined hole pattern spacing for the AFRL Responsive Space Testbed Plug and Play Sat. As such, this design should provide a relevant example of the scale needed to exploit adaptive wiring in a real-world system.

2. Combine FPGA and switch PCB onto one board.

3. Fully implement Self-Healing Algorithm. The work of S. Thompson has shown that an algorithm for self-healing with this kind of architecture can be performed in $O(n)$ time, rather than $O(n^2)$, or on the order of the number of switches as might be thought, as the processing power in terms of adding more and more cells is on the order of *n*. [9]

4. Exploit sub-graph capabilities with Dijkstra's Algorithm by assigning weights to defined node subnets.

5. Allow greater decision making at the cellular level. Presently, the master controller dictates all cell assembly, graph creation, and assignment of routes. In a future version of the AWM, more local autonomy will be assigned to individual cells for routing of signals and detection of modules, though in the immediate future, a master controller in some capacity will still be required.

Towards Programmable Matter

Once these objectives are met, the future direction of the work seeks at least two major long-term objectives:

1. Reduce scale of cells further. The present implementation is what can be described as macroscopic in its cellular nature. This may well be appropriate for implementation for a PnPSat, or even CubeSat application. Milliscopic cellularity would imply an order of magnitude size reduction, and also reduction in the

number of nodes mapped per cell, but effective *increase* in actual node count per unit area. Moreover, the goal of scale reduction is to allow individual computing elements, or "lumped" analog electrical components to be represented by a cell. For example, a cell could contain several resistors, capacitors, and inductors, and when requested, could route wires to specific components as needed. In this fashion, complete analog circuitry could be "programmed" on demand as mission requirements change.

2. Branching out to differing cell types. Different families of cells could be envisioned, defined by their inherent functionality. As described above, cells of lumped electrical components are an example, and cells dedicated to computation and memory (analogous to a preset-day FPGA) are other possible applications. Additionally, other potential cell types are photonic (light-routing) cells, thermal cooling cells (i.e., routing fluids), energy storage, and cells dedicated to photovoltaics.

REFERENCES

[1] Cormen, T., Leiserson, C., Rivest, R., Stein, C., Introduction to Algorithms, "Dijkstra's Algorithm", pp. 595-601, 2001, MIT Press.

[2] DeHon, A. Interconnect – A Fundamental Constraint. Progress Reports, Engenious, Fall 2001.

[3] DeHon, A. Balancing Interconnect and Computation in a Reconfigurable Computing Array (or, why you don't really want 100% LUT utilization), Seventh International Symposium on Field-Programmable Gate Arrays, Feb. 21-23, 1999, Monterey, CA.

[4] Devasys I2C-USB-I/O API Users Guide, Version 0.1.04, Released 9/28/2002, available at http://www.devasys.com/download/UsbI2cIo/API_Users_Guide.pdf

[5] Hauk, S., Borriello, G., Ebeling, C., Mesh Routing Topologies for Multi-FPGA Systems, IEEE Transactions on VLSI Systems, Vol. 6, No. 3, pp. 400-408, Sept. 1998.

[6] Lyke, James C., R., Wilson, Warren G., and Broyls, Ren H., Albuquerque, NM, U.S. Patent Application for a Adaptive Manifold, Pub No. US 2002/0141130 A1, filed 3 Oct. 2002.

[7] Lyke, James, Wilson, W., Contino, P. MEMS-based reconfigurable manifold. In Proc. MAPLD (2005), NASA, available at http://klabs.org/mapld/index.htm.

[8] Murray, V., Feucht, G., Lyke, J., Pattichis, M., Plusquellic, J., Cell-based Architecture for Reconfigurable Wiring Manifolds, to be presented at AIAA@Infotech, April 22, 2010.

[9] Thompson, S., <u>On the Application of Program Analysis and Transformation to High Reliability Hardware</u>, Ph.D. Dissertation / Technical Report, Cambridge University, 2006. Available at http://www.cl.cam.ac.uk/TechReports/.

[10] Thompson, Sarah, Mycroft, Ian, <u>Self-Healing Reconfigurable Manifolds</u>, *Proc. DCC'06: Designing Correct Circuits,* Vienna, Austria, Mar. 25-26, 2006.

[11] Thomson, Sarah, Mycroft, Alan, <u>Dynamic Testing and Automatic Repair of Reconfigurable Wiring Harnesses</u> Defense Technical Information Center Accession Number ADA463036, Final Report. 29, Jan. 29, 2007.

APPENDIX A – SELECTED SOURCE CODE

Section 1: Dijkstra's Implementation

```c
void dijkstra(int s) {
    int i, k, mini;
    int visited[GRAPHSIZE];

    for (i = 1; i <= nodes_total; ++i) {
        l_shortest_path[i] = INFINITY;
        prev[i] = -1; /* no path has yet been found to i */
        visited[i] = 0; /* the i-th element has not yet been visited */
    }

    l_shortest_path[s] = 0;

    for (k = 1; k <= nodes_total; ++k) {
        mini = -1;
        for (i = 1; i <= nodes_total; ++i)
            if (!visited[i] && ((mini == -1) || (l_shortest_path[i] <
l_shortest_path[mini])))
//for each node,
                mini = i;

        visited[mini] = 1;

        for (i = 1; i <= nodes_total; ++i)
            if (dist[mini][i])
                if (l_shortest_path[mini] + dist[mini][i] <
l_shortest_path[i]) {
                    l_shortest_path[i] = l_shortest_path[mini] +
dist[mini][i];
                    prev[i] = mini;
                }
    }
}
```

Section 2: Recursive Path Implementation

```c
void storePath(int dest) {
    if (prev[dest] != -1)
        storePath(prev[dest]);
    printf("%d ", dest);
      path[path_index] = dest;
      path_index++;
```