A Scalable, Open-Source Architecture for Real-Time Monitoring of Adaptive Wiring Panels

Daniel Llamocca[1]
University of New Mexico, Albuquerque, NM, 87131, USA

Victor Murray[2]
Universidad de Ingeniería & Tecnología, Lima, Peru

Yuebing Jiang[3] and Marios Pattichis[4]
University of New Mexico, Albuquerque, New Mexico, 87131, USA

and
James Lyke[5] and Keith Avery[6]
Air Force Research Labs, Albuquerque, NM, 87112, USA

We recently introduced the first prototype of an Adaptive Wiring Panel (AWP) which implemented a reconfigurable switch fabric that allows dynamic routing of analog, digital, and power signals for space system applications. In this paper, we consider a complete redesign and re-implementation of the AWP system to address issues associated with scalability, reliability and real-time monitoring of the switching fabric. We demonstrate the new system using 48 cells as opposed to the 6 cells of the first AWP prototype. We make our hardware and software systems open source and provide recommendations to support further extensions to our system.

Nomenclature

AWP	=	adaptive wiring panel
CU	=	cell units
CMU	=	cell management unit
I^2C	=	inter-integrated circuit
AWM	=	Adaptive Wiring Manifold
FPGA	=	field-programmable gate array
AFRL	=	Air Force Research Laboratory
VHDL	=	very-high-speed integrated circuits hardware description language
NP-hard	=	non-deterministic polynomial-time hard
GUI	=	graphic unit interface

[1] Postdoctoral Fellow, Department of Electrical and Computer Engineering, MSC01-1100, 1 University of New Mexico, Albuquerque, NM, 87131.
[2] Professor, Department of Electrical Engineering and Automation, Universidad de Ingeniería y Technología, Av. Cascanueces 2281, Santa Anita, Lima 48, Lima, Perú.
[3] Doctorate Candidate, Department of Electrical and Computer Engineering, MSC01-1100, 1 University of New Mexico, Albuquerque, NM, 87131.
[4] Professor, Department of Electrical and Computer Engineering, MSC01-1100, 1 University of New Mexico, Albuquerque, NM, 87131
[5] Technical Advisor, Space Vehicles Directorate
[6] Senior Engineer, Space Vehicles Directorate.

I. Introduction

An Adaptive Wiring Manifold (AWM) uses dynamic routing of signals and power for implementing spacecraft wiring harnesses that support self-healing circuits and circuit customization. At a basic level, the AWM allows the users to interconnect components over an Adaptive Wiring Panel (AWP) whose configuration can be modified on demand. A first prototype of the AWP was presented in [1]. This first prototype presented a proof of several of the basic concepts associated with the AWM. In this paper, we seek to address several significant issues associated with scalability, real-time monitoring, and extendibility of the original prototype.

The motivation for the AWM is to provide an effective method for implementing circuits that can recover from component and connection failures during real-time operation as well as provide support for implementing new circuits after launch. To recognize the need for the AWM, we note that recovery during real-time operation is not possible with standard wiring harness or the use of standard PCBs. The wiring harness will provide fixed connections between components that cannot be altered during spaceflight. If one of the connections fails during spaceflight, there is no recovery for a traditional wiring harness. Similarly, the use of quick-turnaround PCBs can provide the necessary circuit components needed for a faster time to launch. However, standard PCBs cannot recover from connection failures. Additionally, the use of triple redundancy can support recovery from single component failure but not from connection failure. To support recovery from connection failure, we require dynamic rerouting.

The concept of the AWM is related to the use of dynamic partial reconfiguration (DPR) with Field Programmable Gate Arrays (FPGAs). FPGAs allow the users to employ DPR technology to implement dynamic re-routing and re-implement digital signal components after launch (e.g., see [2]). Unfortunately, it is not possible to extend the functionality of the FPGA by adding new hardware components to the FPGA fabric. Furthermore, internally, the FPGA fabric does not support routing of analog signals or power.

The use of plug-and-play technologies does support some of the concepts associated with the AWM (e.g., see [3]. After connecting a plug-and-play device, it is automatically detected and made available to the host computer system. Furthermore, plug-and-play can provide power to the connected device. The AWM also supports automatic component identification and power connections. AWM extends the plug-and-play technologies by allowing arbitrary connections between components. More fundamentally, AWM provides a method for implementing circuits while plug-and-play provides a standard method for data communications between devices.

Overall, the purpose of the adaptive wiring manifold is to manage connections between components in real-time. To implement a circuit, the users only have to place components on an adaptive wiring panel and provide a description of the requested connections. The AWM will then implement the circuit by routing all necessary connections (e.g., signals, power) between components. Connection failures can be handled through dynamic re-routing. Simply put, the AWM uses the circuit description to reconfigure the AWP to avoid faulty connections. Assuming that we have redundant components, component failure can be addressed by removing all of the connections to the faulty component and dynamic re-routing to re-implement the circuit from its original description.

In what follows, we provide some more details on the implementation of the AWM. First, we require that the AWP should continually sense the components that are placed on it. By sensing the components, the user will not need to re-program the AWP for each component. Furthermore, component failure can be detected through a failure to sense the component presence. A simple graphical user interface is used to describe the circuit to be implemented. Internally, the circuit is described by a graph data structure. To implement the circuit, a connection is mapped to a physical route on the AWP. For each type of connection (e.g., power, digital, and analog signals), a different graph is constructed to keep track of the different possible routes that can be used to implement the connections. The routing is implemented sequentially by implementing one connection after the other. If the AWM cannot implement a connection, the ordering of the connection is modified and the procedure is repeated. Continuous sensing of the components placed on the AWP is used to detect component failures. When a failure to connect to a component is detected, the AWM looks for the component sensed at a different physical location on the AWP. Once such a component is found, the circuit is re-implemented using the new component.

A first hardware implementation of the AWP discussed in [1] is shown in Fig. 1(a). To implement a circuit, circuit components are placed on *modules*. Modules include SPICE descriptions of their components. The SPICE (Simulation Program with Integrated Circuit Emphasis) descriptions are read by the AWP once the modules are placed on the *cells* that make up the AWP. The cells are programmed to handle the routing and the connections by a *cell management unit*. Thus, a circuit is described to the cell management unit by specifying connections between the components.

In practice, users of the AWP will only need to specify circuits in terms of connections between components. This ideal scenario assumes that modules will be pre-built with standard components. Then, the AWP implements the circuits by routing each one of the specified connections. *Dynamic reconfigurability* of the circuits was achieved in [1] by continuously sensing the modules and re-implementing the connections. Thus, in the first prototype of the AWP described in [1], modules can be re-arranged while maintaining the pre-specified circuit through dynamic re-routing.

Unfortunately, there are several fundamental limitations of the original prototype that led to its re-design whose concept is shown in Fig. 1(b). We summarize the basic issues that motivated the complete re-design below:

- ***Scalability using an AWP substrate and redesigned cells:*** The original cell-unit arrangement depicted in Fig. 1(a) cannot be effectively serviced. To see this, note that failure of one of the central cells will require the disassembly of a large number of cells starting from the cells connected to the faulty cell. In the redesigned system of Fig. 1(b), all cells are connected to an *AWP substrate.* The cells were also redesigned so that they can be connected to any part of the AWP substrate and be independently removed without the need to remove any other cells. Furthermore, the cell interconnection topology can be varied without the need to connect directly to the nearest neighbors as in Fig. 1(a).
- ***Scalable module placement and sizes:*** In the first AWP prototype, module placement proved to be very challenging due to the need to fully align the pre-soldered pins with the cells. The pins have been replaced by miniature banana plugs that allow for much easier placement (including rotations) on the AWP substrate. Furthermore, the use of the AWP substrate supports modules of different sizes (see Fig. 2). We demonstrate this flexibility in module size using a module that is twice the size of a basic block size.
- ***I^2C signal reliability issues:*** The original I^2C signals could not be reliably read in the largest AWP designs considered in this paper. This issue was addressed through the redesign of the memory reading protocol to access a single byte at a time and the use of triple modular redundancy
- ***Real-time monitoring of dynamic routing at the switch level:*** In the first AWP prototype, it was not possible to trace the routing to its constituent switches. We provide a GUI interface that provides real-time hardware monitoring of the system that is accurate down to the single switching unit (e.g., relay-level).
- ***Open-source software and hardware system:*** We provide a hierarchical description of the source code (C-code and VHDL code) for implementing the AWP software and hardware to support further research in this area.

The rest of the manuscript is organized as follows. In Section II, we present related work that was done prior to the AWP. In Section III, we provide a description of the redesigned hardware components. Section IV presents the revised software interface (both command-line and GUI). Recommendations for future research are given in Section V. Concluding remarks are given in Section VI.

II. Related Work

From the lowest to the highest circuit levels, wiring remains an open problem. A number of Printed Circuit Boards (PCBs) are placed into electronic boxes (usually involving a wiring backplane) that are connected through a wiring harness. The construction of conventional wiring harness requires painstaking planning to identify the physical location, quantity, and quality of the wiring network. The many individual connections of a wiring assembly are often created manually, resulting in considerable time and expense. Only limited changes can be made once a wiring assembly is produced. Defects in either the wiring harness or the components in a platform can be difficult to isolate, and repairs are cumbersome undertakings. For a detailed discussion about representation, different wiring domains, addressing wiring harness complexity, and cabling/connections options of wiring harnesses, we refer to [1].

Wiring length distributions obey a power-law relationship as established in the empirical work of Christie [4] and Donath [5] for PCBs and ICs. A survey of wire lengths on the TacSat-2 spacecraft[*] suggests a wiring distribution characteristic similar to those found on ICs. We know from this glimpse of wiring utilization that most wiring runs are short, which makes intuitive sense, and the number of longer runs falls sharply.

Ultimately, we would like to reduce the mass of a spacecraft's wiring harness without compromising reliability [1]. To address these issues, AFRL introduced the Space Plug-and-Play (SPA) in [6] that is both scalable and topology-agnostic. The size of the network can be expanded using more hubs without affecting the system function. SPA-based systems have been developed in the laboratory, and simple SPA networks have been flown in experimental suborbital and orbital space missions (e.g., PnPSat-1) [3]. Similar to the AWP prototype, the PnPSat-1 used a 5x5 cm pegboard-like grid and short cables are used to connect the module to the panel. The spacecraft was then formed by assembling the panels together.

Throughout our work, we have focused on embodiments of the Plug-and-Play wiring harnesses, also called Adaptive Wiring Manifolds. The idea of the AWM has been previously described in [7][8][9][10] and demonstrated in [10]. Algorithms for implementing self-healing dynamically reconfigurable AWMs were reported by Thompson and Mycroft in [11]. In this manuscript, we present the final implementation of a novel strategy for implementing scalable AWMs through a cellular approach. Our approach helps to reduce the wiring bundle within systems. Previous approaches were simpler than our scheme [7][8][9][10] but lacked scalability and robustness. The work by Dehon et al. [12] allows for extensible and dynamically reconfigurable topology. Early issues associated with a Cell implementation of the AWP were discussed in [13]. The first design that demonstrates the concept on an array of Cell Units was presented in[14]. A first prototype using custom-made Cell Units is described in [15][1]. The present work details the final prototype of the Adaptive Wiring Panel using 6×8=48 Cell Units.

III. The Redesign of the Adaptive Wiring Panel

In this section, we describe the redesign of the AWP in terms of the AWP substrate, the Cell Unit, and Modules. We begin with a summary of the AWP concept.

Each Cell Unit (5x5cm) contains an array of solid-state relays controlled by an FPGA. The FPGA also handles communication with the master unit (PC side) and neighboring Cell Units. Cell units are interconnect through terminals located on their four sides. Terminals on the top side of each cell are used for connecting to the planar substrate.

Modules are attached on the AWP substrate. The modules connect to the planar substrate through a number of panel pins. The geometric configuration of the CUs, the modules' configuration (position, orientation), and the netlist information specifying the characteristics and pins' location of the electrical components within the modules are relayed to a computer that implements the Cell Management Unit (CMU), which manages global communications and routing configuration of all CUs. Each Module communicates with a Cell Unit via I^2C [16] in order to provide information about its components and placement. The CMU communicates via I^2C with each of the Cell Units to manage the interconnections.

As shown in Fig. 1(a), the original prototype requires interconnection of the cells in a fixed grid architecture. This requirements was removed in the new prototype depicted in Fig. 1(b) since the cells can be interconnected by using ribbon cables that run between the cells. The new prototype supports the possibility of experimenting with new interconnection topologies such as the omega network, delta network, and the extra stage cube described in [17]. The implementation of other interconnection topologies will come at a relatively minimal cost since it involves the use of ribbon cables of different lengths and adaptations of the software to support the visualization and validation of different topologies. In other words, the current software that supports the visualization of the cells arranged on a grid will have to be modified to support the standard conventions for interconnecting and visualizing alternative topologies that were described in [17]. However, overall, it is important to note that this will not require a structural re-designed of the new prototype. We mention the software modification in future work.

The AWP prototype has the ability of routing both digital and analog signals. This was accomplished by the AWP switching fabric (array of solid-state relays). This array is mapped to an undirected graph over which we apply routing algorithms in order to wire components.

The solid-state relays currently being used implement the exclusive pathway for analog signals. The relay characteristics fulfill most analog application requirements (relay ON-resistance = 0.8Ω).

The previous prototype[1] implemented the AWP with six (6) Cell Units. We have made several important modifications to the AWP. We modified the physical design of each cell unit, made changes to the way the CUs interconnect with each other, and also added a planar substrate. The physical design of the Modules has also been modified. We provide further details in the following subsections.

We note that the new prototype has been used to demonstrate dynamic re-routing and plug-and-play placement of components. The platform allows for the testing of routing algorithms and other interconnection topologies. But as it will be clear in this section, the prototype cannot feasibly fly without addressing power consumption issues and mechanical connection requirements. In terms of satisfying mechanical requirements for flight, we note that the current design can follow the same approach as the one taken for PnPSat-1 described in [1]. In the PnPSat-1, the modules were fastened to panels using bolts in standard mounting holes (a 5x5 cm pegboard-like grid is used on PnPSat-1) with short cables connecting the module to the panel, and the panels assembled together to form a spacecraft. Similarly, for this prototype to be flown, we can replace the ribbons by cables, and fasten the modules and cells to the panel using bolts in standard mounting holes.

A. Adaptive Wiring Panel Substrate

In the previous prototype [1], the AWP planar substrate was formed by tiling together the top portions of a number of Cell Units. The connection of a module to the AWP planar substrate was physically implemented by pins on the module that connect to the sockets in the AWP planar substrate. Each Cell Unit included 15 pins and each Module included 15 pins (5x5) or 30 pins (5x10). Due to fabrication and soldering issues, it was very difficult for the Module to have all the pins aligned with each other: in [1], one can see that a module consisted of two Printed Circuit Boards (PCBs) so as to improve on the alignment of the module's pins. Even so, the connection between the module and the AWP substrate was difficult to carry out. When we required translating or rotating a module, the task was so cumbersome that it would take some minutes to complete.

So, our former selection of simple pins and sockets presented us with a mechanical problem. We addressed this problem by utilizing more adept pins and sockets, the so-called miniature banana plugs and sockets. This connection is much more robust and the plugs are not required to be soldered, which allows us to only utilize plugs when we need to, facilitating the connection between the modules and the AWP substrate.

The second issue was related to the scalability and servicing of the AWP. The original AWP cell units could not be easily aligned to form a substrate that would work well with the modules. These problems were strongly exacerbated when we tried to connect modules that required connections to two or more Cell Units (e.g., for a 5x10 module, or rotated module).

In order to overcome this difficulty, we needed to discard the idea of tiling together the top portions of the Cell Units. Instead, the AWP substrate was redesigned to be made out of only one piece (see Fig. 1(b)). In this way, we can guarantee a seamless plugging/unplugging of modules at different rotations and re-arrangements.

The redesigned AWP substrate consists of 6x8 (30cmx40cm) Cell Units. Figure 2 depicts the new AWP substrate with two modules (a 5x5 and a 5x10) plugged into the substrate. In Figure 2(a), each Cell Unit is shown with three types of connectors on it: i) signal connectors (depicted as '×'), ii) power connectors (depicted as '☐'), and iii) a mechanical connector (depicted as '○'). We also include the ground connector depicted as '■'. Figure 2(b) shows a photo of the AWP substrate.

B. Cell Unit

The Cell Unit (5x5cm), also known as the AWP Cell, is the minimum independent unit of the AWP. Fig. 3(a) depicts the conceptual implementation of a Cell Unit with its ports for wiring resources, communication links, and its Logic Processing Unit (LPU). The surface routing terminal is a collection of pins that connects to the planar substrate. Fig. 3(b) depicts the I^2C communication links between the CU and the CMU, between the CU and its neighbors, and between the CU and the module attached to it. The I^2C link between the CU and the CMU is shared with all the other CUs. In what follows, we detail the important modifications to the physical design of the Cell Unit as well as the minor modification to the LPU.

1. Modification to the Physical Design of the Cell Unit

In the previous prototype [1], the Cell Unit was physically realized with five Printed Circuit Boards (PCBs): 4 lateral boards and 1 top board. As the new planar substrate consists of only one piece, the top board is no longer needed.

As mentioned earlier, in the original design, removing a CU from the AWP would have required removing the surrounding CUs (as the CUs are interconnected by D15 connectors). This process can become extremely cumbersome in larger designs since removing the surrounding CUs would usually require removing the ones surrounding those CUs. In the new design, each cell is attached to the AWP substrate directly. Just like SPA, connections with neighboring cells are made using ribbon cables. This allows for easy servicing of the AWP should one or more CUs fail. Figure 4 shows a physical depiction of the Cell Unit as well as an actual photo of the Cell Unit. Note that the 90° connector contains: i) the pins of the surface routing terminal that connect to the AWP substrate, and ii) pins for I^2C communication between the Cell Unit and a module. Similarly, the rectangular female headers, to which the ribbon cables are connected, contain the routable wiring resources of the Cell Unit as well as the I^2C pins for communication with the neighbors.

As in the original prototype, a female D15 connector is used to implement the I^2C port that allows the CU to connect to the CMU. This I^2C link is repeated throughout all CUs (each CU repeats it to all its neighbors), so that we can make the physical connection between all the CUs and the CMU by plugging a Video Graphics Array (VGA) cable coming from the computer to any Cell Unit. Usually, the I^2C signals from the VGA port of the computer does not provide enough power for the I^2C signals to reach a CU far from the CU to which the VGA cable is connected. We avoid this problem by including an I^2C bus extender (Philips 82B715) in each I^2C port inside the CU that has a direct physical connection to the computer (CMU). The computer side (CMU) also includes this I^2C bus extender.

An actual physical arrangement of these new Cell Units is displayed in Fig. 5. The Adaptive Wiring Panel is shown in a 6x8 array configuration. Fig. 5(a) displays the AWP in operation with two modules on top of the AWP planar substrate. Fig. 5(b) shows the corresponding GUI snapshot where all the Cell Units and modules have been detected. Fig. 5(c) displays an upside-down view of the AWP. Note that the underneath part of the planar substrate contains an array of female headers that allow the Cell Units to be connected to the AWP substrate. Fig. 5(d) shows a close view of the front side and back side of the Cell Units. The components of the Cell Units (e.g., VGA connector, FPGA, relays) can be observed.

2. Logical Processing Unit inside the Cell Unit

The Logical processing unit (LPU), that represents the configurable logic (implemented on an FPGA) inside the CU, has been modified to allow for a more robust communication with a Module. In the original prototype [1], when the CU received the command to read the module netlist (or datasheet), it would read the entire datasheet from the module, and then it would relay the information in the form of: first the number of bytes, and then the actual data. This approach, though effective, does not account for reliability issues. In the 6x8 array configuration, we found that in some cases, the data retrieved from the module were erroneous. We initially attributed this issue to the unreliability of the I^2C communication link between the CU and the I^2C memory (Atmel EEPROM AT24C08B) inside the Module. This problem did not occur as often in the 6 Cell Unit configuration (previous prototype [1]), but it happened frequently in the 6x8 array configuration. Thus, the problem seemed to stem from the fact that the shared I^2C bus links a large number of Cell Units (6×8 = 48) to the Cell Management Unit (CMU).

The communications problem was mitigated by modifying the Logical processing Unit so that it only reads one byte at a time from the I^2C memory inside the module. This modification allows us to develop robust software routines for data reliability (to be discussed in Section IV). Specifically, we defined a new command (`0x60`) that the CMU issues. The command is followed by the address of the I^2C memory (from 0 to 255) from which we want to read one byte. The CU then reads the byte at the specified address in the I^2C memory, and finally the CU relays the one data byte to the CMU. The Local Processing Unit is described in VHDL, and Figure 6 depicts a diagram showing the dependency of the corresponding VHDL files.

C. Modules

The previous prototype [1] supported modules of dimensions 5x10 cm. We performed the following modifications to the physical design of the modules:

1) To connect to the AWP substrate sockets, the modules were re-designed to use miniature banana plugs. This allows for a more robust and flexible plugging, unplugging, and rotation of modules to the AWP.
2) We investigated the use of modules of different sizes. For our experiments, we considered two different dimensions: 5x5 cm, and 5x10 cm. The 5x5 module contains 12 signal connectors, 3 power connections, and one mechanical connector. The 5x10 module has 24 signal connectors, 6 power connections, and one mechanical connector.

The I^2C memory (Atmel EEPROM AT24C08B) inside the module contains the electronic datasheet (in SPICE language) of the components that are connected on the module. The I^2C memory can store up to 256 bytes of data. To download (and read) the module's datasheet, a direct I^2C connection of the module's mechanical connector with the CMU is required.

Figure 7(a) depicts the modules (5x5 and 5x10 dimensions) when plugged into the AWP substrate. All the possible rotation configurations (0°, 90°, 180°, and 270°) are depicted. Note how one 5x5 module can span up to 4 Cell Units, and one 5x10 module can span up to 6 Cell Units. Fig. 7(b) shows the modules' depiction, with the naming conventions for the pins and the datasheets when some components are connected to the modules. Finally, Figure 8 shows actual pictures of the 5x5 and 5x10 modules. Notice the miniature banana connectors being used.

Note that the components can be very varied: they can have more than two pins. The naming convention for a component is `<Letter><Unique number>`. If there are two or more similar components (e.g., three resistors), each one must have a unique number within all the Modules. For example, in Fig. 7(b), the two modules contain 3 resistors (R1, R2, R3), 3 LEDs (D1, D2, D3), and one battery (V1).

We note that the paper uses simple circuits consisting of batteries, resistors, and LEDs to demonstrate the basic concepts. While this is an effective way to test the AWP hardware, it is clear that practical implementations would require support of subsystems such as a reaction wheel, star tracker with a camera, etc. Figure 9 shows two examples of what connections would be needed for implementing the following subsystems:

- A Star Tracker (e.g., Vixen Polarie Star Tracker) that requires 5 VDC. A camera is mounted on top of the Star Tracker. The Star Tracker is the sole component in a module. The 5VDC source is connected to another module.

- A Reaction Wheel (e.g., MSCI MicroWheel 200). It requires 24 VDC and a RS-485 port for three control lines. The 24 VDC source belongs to another module. The three control lines could be connected to another module containing a microcontroller.

Note how in both cases we can easily connect/disconnect the power source from the subsystems by activating /de-activating the solid-state relays via software.

D. AWP Power consumption

To understand power consumption issues associated with the current AWP design, we also provide an analysis of power consumption of the AWP. First, we measured (using the ES-687 clamp meter) the idle power consumption of the Cell Unit (when no relay is activated). Second, we activated the relays one after the other and measured the power consumption as the number of activated relays increased. Figure 10 shows the power consumption (in mW) of a Cell Unit as the number of activated relays increases. The idle power consumption of a Cell Unit was about 0.42 W, while the power consumption per relay was on average 85mW. This information allows the software interface to provide real-time power estimation based on the number of active cells and number of closed relays.

The estimated power consumption of the current AWP prototype with 6×8 = 48 Cell Units is about 20.16 W, while the maximum possible power consumption (considering all relays within the 48 Cell Units are activated) is about 305.5W. In practice, however, we expect that the average number of relays activated per cell not to exceed 10, resulting in the AWP drawing 60.48W in the worst case.

From these measurements, it is clear that a practical implementation would require significantly lower power consumption. Currently, the AWP has an idle power consumption of 20.16W. Most of this power is drawn by the FPGAs. Also, note that the miniature relays used here (Panasonic AQY221R2M1Y, size: 2.95x2.2 mm^2) draw 85 mW when activated (i.e.,,they are in the ON state). Latching relays (such as the Omron G6KU-2F-Y, size: 6.5x10mm^2) would reduce power consumption, as it requires 21mA@5V for 10 ms. The main problem with latching relays however is their size. The Omron latching relay was the smallest one available and it is physically about 10 times larger as the ones we are using. This poses a problem for PCB design, as the PCBs would also have to be much larger. By far the best solution is the use of ASICs that integrate the control logic inside the FPGA, the processing elements, and the array of solid state relays so as to dramatically reduce the size and power consumption.

IV. Real-time Monitoring of the AWP

Real-time monitoring is handled by the Cell Management Unit (CMU). The CMU manages global communications and routing configurations of all Cell Units on the AWP. The CMU is implemented in software on a Linux machine.

The CMU handles: i) reading the configuration of the Cell Units and represent them as an undirected graph, ii) reading the configuration of the Modules and their netlist specification, iii) providing the user with an interface to specify a circuit (or circuits), iv) routing all required connections for the specified circuit(s), and v) keep the connections that make the circuit(s) in response to a change in the configuration of the Modules and/or Cell Units (the user can also modify or add connections). Each time the configuration of the Modules and/or Cell Units changes, a new graph is generated, over which the routing algorithm is run again. As for the routing algorithms, we are using the same algorithm as the one mentioned in [1].

The software routines are divided into two layers: i) a Command Line Interface that provides a complete control of the AWP, used mainly for debugging purposes, and ii) a Graphical Unit Interface (GUI), which goes on top of the Command Line interface, that provides a much smoother and intuitive user experience.

In this section, we describe the updates to the software routines as well as improvements made to the Graphical User Interface (GUI).

A. Reading the datasheet from Modules

The I^2C communication between the Cell Unit and the CMU was summarized using four commands [1] (Type I, Type II, Type III, and Type IV). Specifically, the Type III command was used to read data from a Module: the CMU issued a command to read the module datasheet (or netlist), the CU immediately read the entire datasheet from the I^2C memory of the module, and finally the CU would relay the information (first the number of bytes, then the actual data) to the CMU. This simple approach did not account for reliability issues, as explained in Section III.B.2.

The new approach requires reading only one byte at a time from the I^2C memory inside the module. This is implemented by including a new `0x60` command within the Type IV commands (the Type III command is no longer used). Figure 11 depicts the timing diagram along with the characteristics of this command: the CMU issues the command word `0x60`, followed by the address of the I^2C memory (0 to 255) from which we want to read data (one byte). The CU then reads the byte from the specified I^2C memory address, and finally the CU relays the data byte to the CMU.

This command `0x60` allows us to develop robust software routines for data reliability. We have modified the software routines that manage the communication between the CU and the CMU. When writing/reading on the I^2C bus, the I^2C communication link can fail and we might think that an otherwise existing CU is failing or not present. As a result, when writing/reading on the I^2C bus, we perform up to ten (10) attempts before branding that CU as nonexistent or failing. Thus, we perform up to ten (10) attempts for each of the following operations: sending the `0x60` command, sending the address of the I^2C memory, and receiving one byte of data. In addition, we issue the `0x60` command three times (i.e. we read the same byte thrice) and make sure that at least two times the same byte was read. The datasheet in the I^2C memory contains an end-of-string character (`0x00`) that indicates that we have reached the last byte of information.

B. Graphical User Interface (GUI): Real-time monitoring

The GUI was developed to provide a smoother user experience when managing the Adaptive Wiring Panel (AWP). It was developed based on the GTK+library [18].

As in [1], the GUI automatically updates what it displays when: i) the relative position of the Cell Units change, ii) when Modules are replaced, removed, or rotated, and iii) when the components in the modules change (assuming the datasheet inside the module has been updated).

We updated the GUI so that it now provides more information about the AWP. In addition to the Cell Array Layer, Module Layer, and Layer, we have added the ‘Link Layer’. This layer displays the relays that are currently in operation (i.e., closed) and the Cell Units that contain activated relays. If there are closed relays inside a Cell Unit, it turns green. By double-clicking on the green-colored Cell Unit, the user can see the cell in detail with the status of the relays (closed/open). The user can also see the geometric distribution of the 71 relays with respect to the Cell Unit pins (y1-y12, z1-z3) and the pins that are distributed to the neighbors (X1-X16, U1-U3, GND). See [1] for more information about these pins.

The GUI also provides real-time power estimation. The power estimation is based on the number of active cells, and the number of activated relays (if any). This information is available in any of the layers.

GUI snapshots of the available layers are shown in Figure 2. A snapshot of the Cell Array Layer (6x8) is shown in Figure 5(b). Figure 12 shows a snapshot of the Module Array layer where four (4) Modules with different orientations/sizes are plugged into the AWP substrate. We indicate the mechanical connector of each Module. Power estimation is also displayed for 48 Cell Units with no closed relays.

Figure 13 shows a snapshot of the Circuit Layer, where we have specified and created a circuit that consists of one battery, two LEDs, and two resistors (the battery powers two LEDs). Note that the circuit, as depicted in the Circuit Layer, is a bit difficult to read. For clarity purposes, we have also added a view of the circuit on the left side of Figure 13. The circuit requires 16 closed relays, and as such the estimated power has increased.

As for the Link Layer, Figure 14 displays the AWP Cell Array where Cell Units with activated relays are green-colored. By double-clicking on a Cell Unit, we can see the geometric distribution of the 71 relays within the Cell Units, and more importantly the relays that are activated. Figure 15 shows a snapshot of the internals of a Cell Unit with the 71 relays and the Cell Unit pins. The activated relays are depicted as solid red.

C. Open-source Interface

To support further research in this area, we provide an open-source implementation of the interface (GPL license). For the Command-line Interface , we provide a summary of the code structure and main functions in the Appendix. The password-protected code (available upon request) can be found at: www.ivpcl.org/AWPcode.zip.

V. Recommendations for Future Work

In this section, we provide a list of recommendations for future work on the AWP. We believe that this list will contribute to the wider adoption of the AWP:

- ***Database of typical circuits:*** The overhead associated with the AWP can only be justified for large-scale circuits. We used very simple circuits to demonstrate the concepts here. Ideally, we would to test the system on a library of typical circuits (e.g., navigation, communication circuitry, payload, see [1]).
- ***Dynamic routing algorithms:*** The current algorithm is based on a heuristic that uses the shortest path algorithm to route each connection. The search start with an initial ordering of the connections. If a solution

is not found, the search restarts from a different initial ordering. The process is repeated until a solution is found. With the 6x8 AWP prototype, we found that the routing algorithms do not perform well when the number of connections is greater than 10. This is to be expected since the routing problem is known to be nondeterministic polynomial-time hard (NP-hard). Dynamic routing would be best performed using a Steiner forest algorithm as described in [19].

- ***Distributed management approach:*** For the next generation of the AWP, it will be a good idea not to require the current external Cell Management Unit (CMU) for routing purposes. In a distributed management approach, routing decisions could be made locally. This would also make the routing computation to be fault-tolerant.
- ***Exclusive connectivity pathway for digital signals:*** the AWP solid-state relays can switch at 5 KHz at most, rendering the AWP unsuitable for a large variety of digital applications. Thus, for digital signals, instead of using a relay array, we can simply use an FPGA fabric for routing the connections.
- ***Incorporation of signals of different nature:*** This idea extends the original AWP concept to a more universal solution, where the switch fabric includes power, optical, and RF signals. This will allow the network to perform in different scenarios and might potentially provide a universal solution for interconnection in aerospace systems.
- ***Miniaturization:*** It is important to investigate the feasibility of the integration of the control logic inside the FPGA, the processing element, and the array of solid-state relays (we will be looking for latching relays) in a single die. This miniaturization effort can dramatically reduce the size and power consumption of a Cell Unit. It will provide more space within the Cell Unit to include various types of connections, such as optical and RF.
- ***Mechanical connections:*** The current use of a motherboard-daughterboard approach will have to be modified for flight. We recommend the use of an approach similar to the one taken for PnPSat-1 described in [1] and summarized in Section III.
- ***Testing of other interconnection topologies***: The redesigned AWP prototype allows the consideration of different interconnection topologies (e.g., omega network, delta network). In terms of hardware, no further modifications are needed for supporting new topologies. However, the software will need to be modified to support visualization and validation of the new topologies.
- ***Reliability of electronics under Single Event Effects (SEEs)***: To prepare the system for flight, we will need to address the possibility of SEEs on the FPGAs and the solid-state relays. Here, we note that a common technique to mitigate the effects of SEEs on FPGA is the use of Triple-Modular Redundancy (TMR), but this incurs in increased area and power requirements. This problem is addressed in [2], where the authors presented a framework (for FPGAs) for reconfigurable fault tolerance that allows for dynamic adjustment of a system's level of redundancy and fault mitigation based on the varying radiation incurred at different orbital positions. As for the solid-state relays, the use of radiation-hardened relays can be of great help here (e.g., IR's RDHA701CD10A2N), although they come at a steep price. Finally, as the Cells of the AWP are made of many other components beside the FPGA and the relays, it is recommended to have some kind of shielding.
- ***Robustness:*** Prior to implementing connections in hardware, it is important to perform basic testing that avoids damaging the circuits. For instance, user-specified connections that can potentially short battery terminals should be detected and not allowed. The SPICE format of the circuit allows for SPICE simulation that detects damaging connections. In addition, components can notify the system of possible failures detected through self-testing.

VI. Conclusions

We have described the final prototype of the Adaptive Wiring Panel (6x8 = 48 Cells) that can interconnect (in principle) arbitrary electronic components together using a large reconfigurable mesh. The adaptive wiring concept can be thought as an extension of the ideas on FPGA routing. The power and utility of the AWP will likely become evident with even larger scale systems. The technology implications for fully adaptive wiring systems are potentially profound. Systems can be constructed more quickly, they can be more resilient, and can have more flexibility. The design challenges include making the pieces (Modules, Cell Units) 'smart', including synthesis tools to manage the complexity of the dynamic wiring.

A complete redesign of the original AWP prototype (6 cells) had to be made in order to develop an effective system with 48 Cell Units. The redesigned system included the use of I^2C power extenders, robust software routines for data reliability, the use of new miniature banana plugs and sockets, the introduction of a single AWP substrate, physical re-design of the Cell Unit, and re-design of the physical connections among Cell neighbors. We also introduce real-time monitoring tools for measuring power consumption and visualizing dynamic routing down the single switch (relay) level. The redesigned prototype of the Adaptive Wiring Panel can act as a test-bed for improving the routing algorithms, communication reliability, mechanical issues. In addition, it opens the door to new ideas about scaling the architecture, power management, miniaturization, autonomous reconfigurable interconnect cells, incorporation of more computing power within the Cell Units, etc.

As mentioned in [1], we believe that there is benefit in extending our implementation concepts to three dimensions. Overall, we hope that our research will contribute to concept of a universal reconfigurable switching network that supports dynamic routing of different types of signals.

Appendix

In this Appendix, we provide a succinct description of all the software routine and function that make up the Command Line Interface of the Adaptive Wiring Panel. We begin with the basic concepts (cell array, graph, module array) that will be used for describing the AWP software routine:

- Cell Unit (CU): Minimum independent unit of the AWP that contains wiring resources (relays, connections to neighbors) and an FPGA. The FPGA controls: i) the I^2C communications of the CU with the CU's neighbors and the CMU, ii) the relays.
- Grid: Array of Cell Units whose arrangement is mechanically modifiable by the user. In the software routine, it is represented by the structure `M`.
- Graph: It is the representation of the wiring mesh that includes the wiring resources within each active Cell Unit. Each CU contains the vertices `z1-z3,y1-y12,x1-x16,` and `u1-u4` (`u4=GND`). The 71 relays (considered edges) connect the vertices (look the arrangement in Fig. 15). The combination of all the vertices and edges for all Cell Units makes up the graph. In the software routine, it is represented by the structure `G`.
- Module Array: Array of Modules whose arrangement can be modified by the user. The modules are attached on top of Cell Units. In software, the module array is represented by `Module_List`.
- Cell-level circuit: Set of connected pairs (represented by structure `Circ`) or to be connected (represented by structure `C`) at the Cell-level, i.e. each node (or pin) is represented by `<node_name>-<cell ID>`. The available node names are: `z1-z3,y1-y12` (pins at the surface routing terminal of the CU).
- Module-level circuit: Set of connected pairs (represented by the structure `Module_List->circuitry`) at the Module-level, i.e., each node (or pin) is represented by its name and the Cell Unit to which the Module is connected to: `<node_name>-<cell ID>`. The available node names are: `Z1-Z3`, `Y1-Y12` for 5x5 modules, and `Z1-Z3`, `Y1-Y12`, `V1-V12`, `W1-W3` for 5x10 modules. Note that a node like `Y1` on the module might not be the same as the node `y1` in the Cell Unit, as the Module might be rotated.

The Command-line Interface manages the AWP Prototype shown in Figure 5(a). We now provide a description of the available commands. Table 1 lists the commands available when we type `mytest_i2c -interface`.

Table 1. List of commands of the Command Line Interface accessible when typing mytest_i2c -interface

`$ mytest_i2c -interface`	
`AWP>> generate_grid`	Creates a connected grid of Cell Units (based on the detected CUs). Fig. 5(b) shows the GUI visualization of a grid with 6x8 Cell Units.
`AWP>> connect A B`	Connects two nodes at the Cell Level. Node format: `<node_name>-<cell ID>`. Example: `AWP>> connect y1-03 y2-1E`. Connects node `y1` of Cell `03` with node `y2` of Cell `1E`.
`AWP>> unconnect circuit`	Opens the currently closed relays and regenerated the Grid.
`AWP>> unconnect_all`	Opens all relays and regenerates the Grid. Useful after the program execution stops unexpectedly, and the currently closed relays are unknown.
`AWP>> print circuit`	Prints the set of connected pairs
`AWP>> print grid`	Prints the current Grid in text-format (unlike that of Fig. 5(b))
`AWP>> print graph`	Prints the list of nodes with their respective neighbors
`AWP>> exit`	Exits interface.
`AWP>> modules`	Enters a sub-interface that allows the issuing of commands at the Module Level, i.e., connections are set by using the Modules' name pins.
`AWP(module level)>> read modules`	For each module: It reads the module size, orientation, components, and the cell to which the module is attached to. Then, it maps the module pins to the corresponding cell pins.

`AWP(module level)>> set_circuit`	Specification of the Module-Level circuit (set of connection pairs). Format of a connection pair:
	`>> ComponentA pinx ComponentB piny`
	A pin of Component A connects to a pin of Component B (in some cases B can be A). Example:
	`Enter pair >> R1 y1 R2 z3`
	(Pin y1 of component R1 connects to Pin z3 of component R2)
	`Enter pair >> C1 y1 R1 y2`
	`Enter pair >> exit`

`AWP(module level)>> connect once`	This instruction attempts to connect the circuit specified in the 'set_circuit' instruction. After the connections are made, the user returns control of the interface.
`AWP(module level)>> connect loop`	The instruction attempts to connect the circuit specified in the 'set_circuit' instruction. After the connections are made, the interface enters an infinite loop, in which the program routinely reads the Cell Grid and Module Array looking for changes. If there are changes and if the circuit's components still exist, the software re-routes the paths to make the circuit connections for new Cell and Module configuration. By pressing a key, the user can always exit the loop.
`AWP(module level)>> print circuit`	Prints Module-Level circuit
`AWP(module level)>> unconnect_circuit`	Disconnects current circuit
`AWP(module level)>> exit`	Exits Module Level

Table 2 lists the two commands available when we type `mytest_i2c -interface_2`. This interface is a little different from the one of Table 1. It immediately enters an infinite loop in which the user can always set, modify, or add circuit connections.

Table 2. List of commands of the Command Line Interface accessible when typing mytest_i2c -interface_2

`AWP>> go`	It enters a loop that repeatedly reads the status and updates the displaying of the Cell Grid and Module Array. At every iteration, the user can press a key and then typing: `AWP>> set` → sets a new circuit, `AWP>> add` → adds connections to the current circuit, and `AWP>> mod` → deletes connections of the current circuit.
`AWP>> exit`	Exits interface

Table 3 lists a series of commands that allow the user to write/read information to/from the I2C memory (Atmel EEPROM AT24C08B) inside a Module. The information consists of: module type (5x5 or 5x10), and a Spice-formatted list of components inside a Module. In addition, the use of these commands require a direct connection of the I^2C pins of the Module Board to the VGA cable that connects to the D15 connector in the computer (CMU).

Table 3 Command-Line Interface commands for writing/reading the I^2C memory inside a Module

`mytest_i2c -write_AT {Spice Datasheet #}`	It writes a Module datasheet to a I^2C memory. No more than 256 characters can be written. The spice datasheets are provided in a text file named: `I2Cprom-{number}.txt.`
	Example of text file: 5x10 modules, 4 components
	`5x10`
	`R3 V1 V2 330`
	`D3 V8 W3 HLMP-C100`
	`D4 Y11 Y8 HLMP-C100`
	`V2 W1 W2 6v`

`mytest_i2c -read_AT`	It reads the Module datasheet from a I^2C memory. This is useful to verify the state of the I^2C memory inside a Module.

Table 4 lists a series of commands that can be used for debugging purposes. Note that the parameter `cell ID` is the identification number (hexadecimal) of a Cell Unit. They go from 2 to 127 (`0x02-0x7F`).

Table 4. Command-Line Interface commands for debugging purposes

`my_test_i2c <cell ID> -relays -{off,on}`	Opens (off) or closes (on) the 71 relays of the given Cell Unit (Cell ID)
`my_test_i2c -relays {off,on}`	Opens (off) or closes (on) all relays of every Cell Unit.
`my_test_i2c -scan`	It scan all Cell Units (`0x02-0x7F`) and lists whether each Cell Unit exists.
`my_test_i2c <cell ID> -neighbors`	Get the neighbors' IDs (if any) of the given Cell Unit (Cell ID). When a neighbor is not present, its ID is 0x01.

`my_test-i2c -grid`	It creates a Cell Grid
`my_test_i2c -m <cell ID>`	It reads and displays the Module information of the Module attached (if any) to the given Cell Unit (Cell ID)
`my_test_i2c <cell ID> -relay <relay #> -{open,closed,status>`	Open, close, or get status of a specific relay in a Cell Unit `<relay #>`: decimal number from 1 to 71. Examples: `mytest_i2c AB -relay 64 –open` % Opens relay 64 from cell 0xAB `mytest_i2c 0C -relay 12 –closed` % Closes relay 12 from cell 0x0C `mytest_i2c 10 -relay 32 –status` % Returns status of relay 32 of cell 0x10

Table 5 lists three commands for low-level debugging purposes. These commands handle the basic I^{2}C communication between a computer and a Cell Unit (I^{2}C peripheral). The user can write/read a byte to/from a given I^{2}C peripheral. The peripheral can be either the FPGA inside the Cell Unit or the I^{2}C memory inside the Module (provided there is direct connection between the Module Board and the Computer). The parameter `data` is an hexadecimal number (`0x00-0xFF`). The parameter `cellID` is the identification number (hexadecimal) of a Cell Unit (`0x00-0x7F`). The value `cellID=0x00` is special: when writing, the I^{2}C command is broadcast to all Cell Units; when reading, the routine grabs data from the first Cell Unit that responds. The value `CellID=0x01` is not allowed as it means inexistent Cell Unit (only used when reading neighbors of a Cell Unit)

Table 5. Commands for low-level debug

`mydebug_i2c -w <cell ID> <data>`	It writes a byte of data on a Cell Unit (Cell ID). Example: mydebug_i2c -w 02 FA % Writes 0xFA on Cell Unit 0x02
`mydebug_i2c -r <cell ID>`	It reads a byte of data from a Cell Unit (Cell ID) Example: mydebug_i2c -r 03 % Reads a byte of data from Cell Unit 0x03
`mydebug_i2c -s`	It runs a scan of all Cell Units (0x02-0x7F) and lists whether each Cell Unit exists.

Table 6 shows the file structure of the AWP Command-Line Interface. A Makefile file takes care of the configuration for the gcc compiler. Two executables are obtained: '`mydebug_i2c`', and '`mytest_i2c`'.

Table 6. File structure of the AWP Command-Line Interface.

Files	**Description of files**
`mytest_i2c.c`	Main file that provides access to all features of the Command-Line Interface
`i2c.h`	Header file that lists the low-level functions for I^{2}C communication with the CMU (Computer)
`i2c-linux.c`	Low-level functions for I^{2}C communication with a Computer.
`my_i2c_commands.h`	Basic routines for I^{2}C communication of the AWP.
`user_interface_functions.h`	High-level routines that implement common user commands (generate grid, disconnect, connect circuit)
`module_level_functions.h`	High-level routines for the management of the Module Array.
`grid_creation_functions.h`	High-level routines for the management of the Cell Grid (or Cell Array)
`shortest_path_functions.h`	Routines that deal with an undirected graph: add/delete vertices from a graph, get vertex value, implement the shortest path algorithm..
`priority_queue_functions.h`	Routines that implement a minimum priority queue. These functions are required for the implementation of the shortest path algorithm.
`my_i2c_macros.h`	Advanced routines for AWP management (scan neighbors, scan cells, connect graph, interface enabling)
`mydebug_i2c.c`	Main file that provides access to low-level functions for debugging.

`i2c.h`
`i2c-linux.c`
`my_i2c_commands.h`	

References

[1] V. Murray, D. Llamocca, J. Lyke, K. Avery, Y. Jiang and M. Pattichis, "Cell-Based Architecture for Adaptive Wiring Panels: A First Prototype," *AIAA Journal of Aerospace Information Systems,* vol. 10, no. 4, pp. 187-208, 2013.

[2] A. Jacobs, G. Cieslewski, A. D. George, A. Gordon-Ros and H. Lam, “Reconfigurable Fault Tolerance: A Comprehensive Framework for Reliable and Adaptive FPGA-Based Space Computing,” *ACM Transactions on Reconfigurable Technology and Systems,* vol. 5, no. 4, p. 30, 2012.

[3] D. Fronterhouse and J. Lyke, “Plug-and-Play Satellite,” in *International SpaceWire Conference*, Dundee, Scotland, UK, Sept. 2007.

[4] P. Christie and D. Stroobandt, “The Interpretation and Application of Rent's Rule,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 8, no. 6, pp. 639-648, December 2000.

[5] W. Donath, “Placement and average interconnection lengths of computer logic,” *IEEE Transactions on Circuits and Systems,* vol. 26, no. 4, pp. 272-277, April 1979.

[6] J. Lyke, “Plug-and-Play as an enabler for Future Systems,” in *AIAA Space Conference*, Anaheim, CA, Sept. 2010.

[7] J. Lyke, “Bringing the Vision of Plug-and-Play to High-Performance Computing on Orbit,” in *Thirteenth Annual Workshop on High Performance Embedded Computing*, Lexington, MA, Sept. 2009.

[8] J. Lyke, “Space Plug-and-Play Avionics (SPA): A Three-Year Progress Report,” in *AIAA Infotech Conference*, Rornert Park, CA, May 2007.

[9] J. Lyke, D. Fronterhouse, D. Lanza and T. Byers, “A Plug-and-Play concept for spacecraft,” in *Military and Aerospace Programmable Logic Devices (MAPLD)*, Washington, D.C., Sept. 2005.

[10] W. Wilson, J. Lyke and G. Forman, “MEMS-Based Reconfigurable Manifold,” in *Military and Aerospace Programmable Logic Devices (MAPLD)*, Washington, D.C., Sept. 2005.

[11] S. Thompson and A. Mycroft, “Self-Healing Reconfigurable Manifolds,” in *Designing Correct Circuits (DCC'06)*, Vienna, Austria, March 2006.

[12] A. DeHon, R. Huang and J. Wawrzynek, “Hardware-assisted Fast Routing,” in *IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, April 2002.

[13] V. Murray, G. Feucht, J. Lyke, M. Pattichis and J. Plusquellic, “Cell-Based Architecture for Reconfigurable Wiring Manifolds,” in *Infotech@Aerospace Conference*, Atlanta, GA, April 2010.

[14] V. Murray, D. Llamocca, Y. Jiang, J. Lyke, M. Pattichis, S. Achramowicz and K. Avery, “Adaptive Wiring Panels using Cell-Based Architectures: A First Approach,” in *Reconfigurable Systems DCT Workshop*, Albuquerque, Nov. 2010.

[15] V. Murray, D. Llamocca, Y. Jiang, M. Pattichis, J. Lyke, S. Achramowicz and K. Avery, “Cell-Based architecture for Adaptive Wiring Panels: A First Approach,” in *AIAA Reinventing Space Conference*, Los Angeles, CA, May 2011.

[16] Philips, “The I2C Bus Specification,” Philips Semiconductors, 2000.

[17] G. I. Adams, D. Agrawal and H. Siegel, “A Survey and Comparison of Fault-Tolerant Multistage Interconnection Networks,” *Computer,* vol. 20, no. 6, pp. 14-27, June 1987.

[18] A. Krause, Foundations of GTK+ Development, Berkeley, CA: Apress, 2007.

[19] M. Feldman, G. Kortsarz and Z. Nutov, “Improved Approximating Algorithms for Directed Steiner Forest,” *Journal of Computer and System Sciences,* vol. 78, no. 1, pp. 279-292, January 2012.

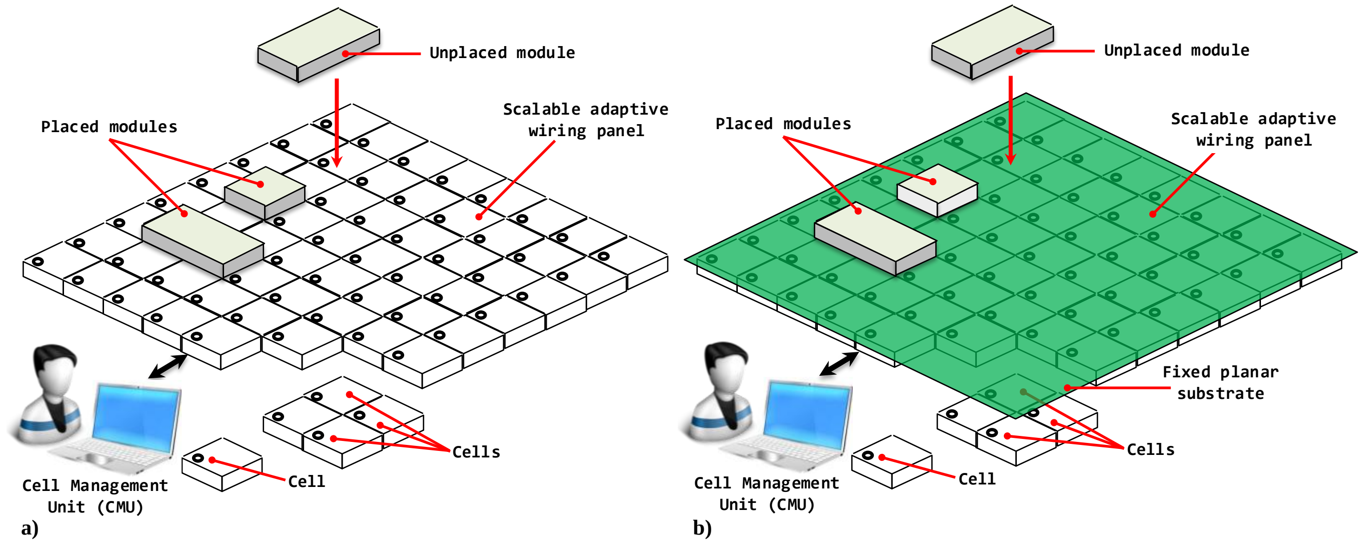
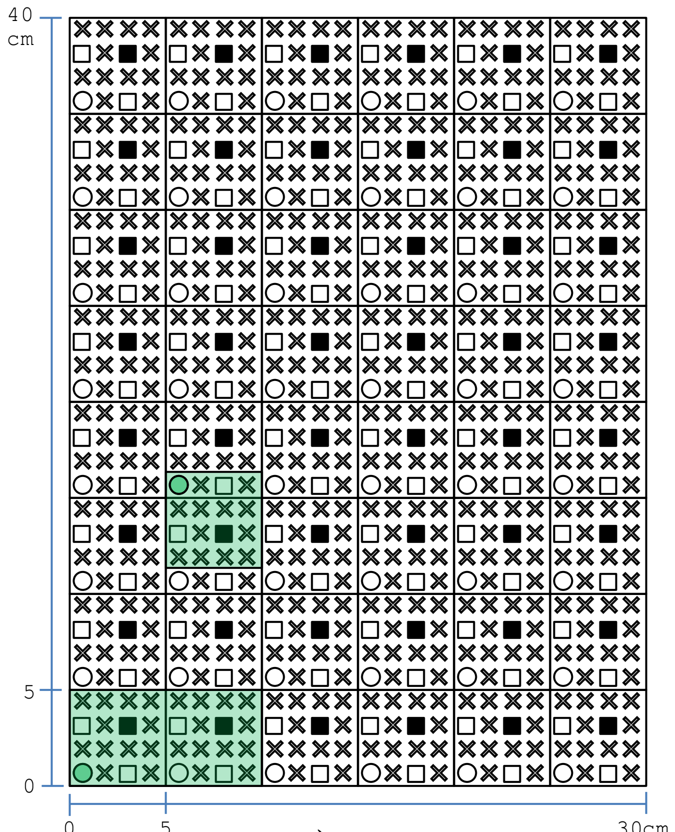
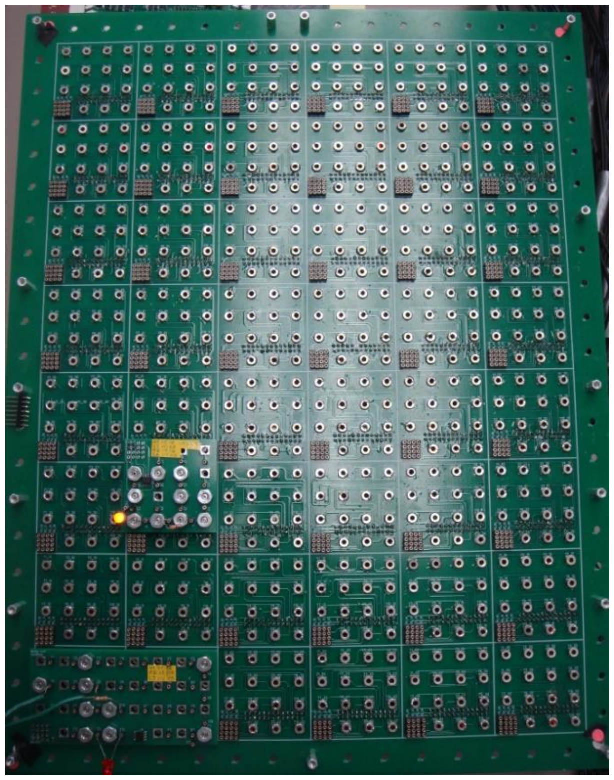

Unplaced module
Placed modules
Scalable adaptive wiring panel
Cell Management Unit (CMU)
Cell
Cells
a)
Unplaced module
Placed modules
Scalable adaptive wiring panel
Fixed planar substrate
Cell Management Unit (CMU)
Cell
Cells
b)

Figure 1. (a) Original Adaptive Wiring Panel (AWP) concept [1]. (b) Adaptive Wiring Panel (AWP) concept with a fixed planar substrate on top of the Cells. Each Cell contains adaptive wiring resources for the modules. Each Cell also contains interconnection resources for the neighboring Cells.

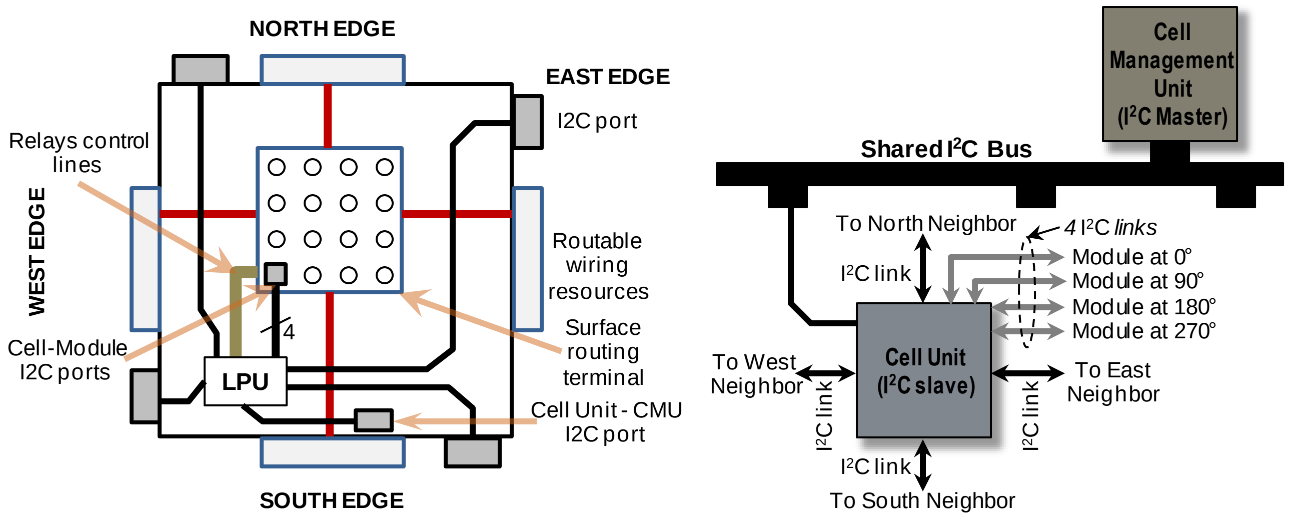


a)



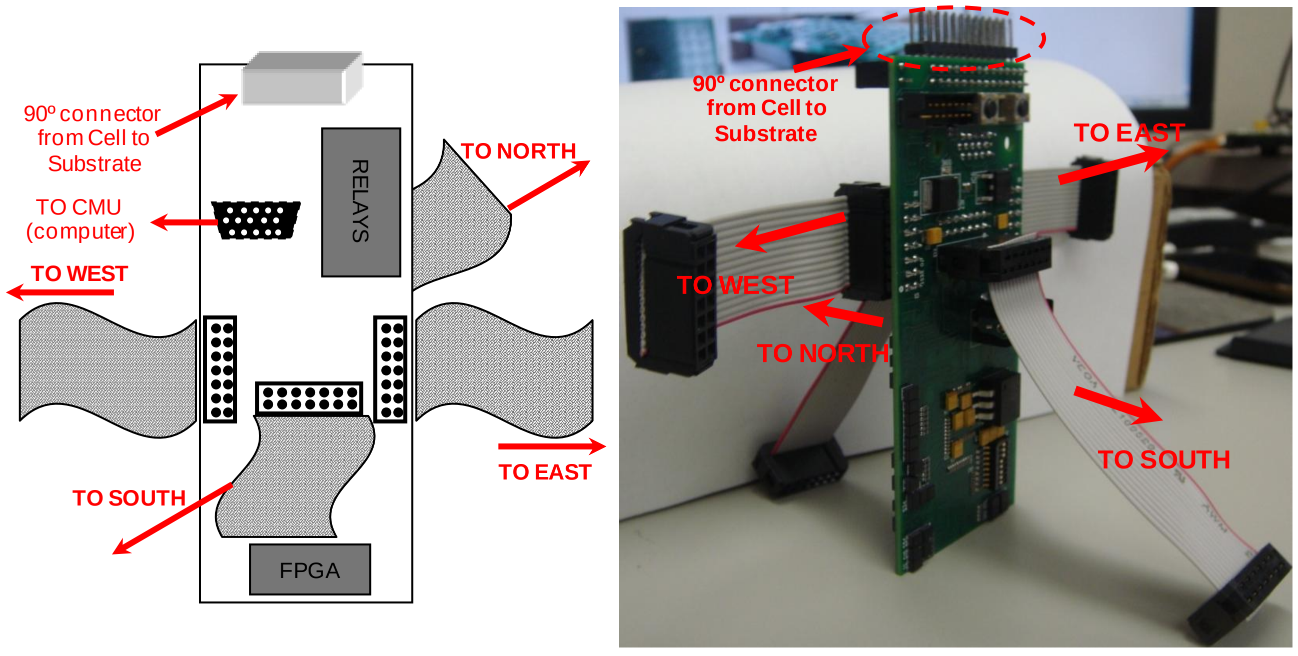
b)

Figure 2. AWP planar substrate. a) Concept using 48 cells in a 6x8 configuration. Two modules are shown in green. b) Picture of the actual AWP substrate. Note how the modules are connected to the AWP planar substrate, and how one LED is lit.



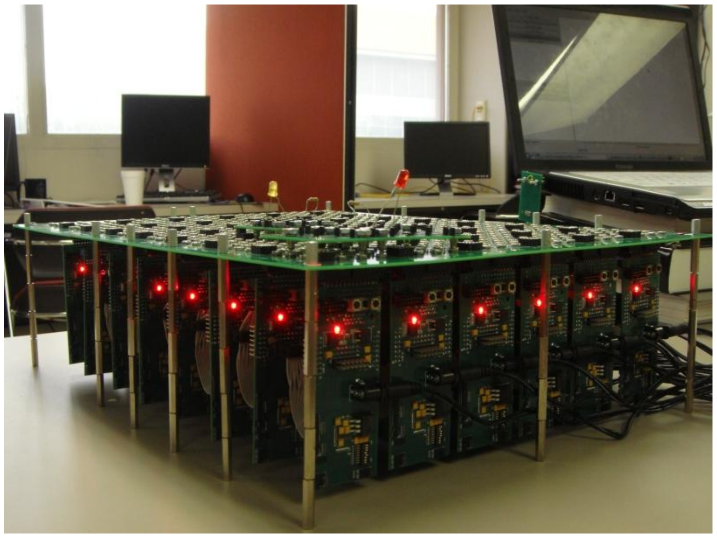
a) b)

Figure 3. Architecture of the Cell Unit. a) Conceptual implementation. LPU stands for Local Processing Unit within the Cell Unit. b) I^{2}C links for the Cell Unit and CMU.

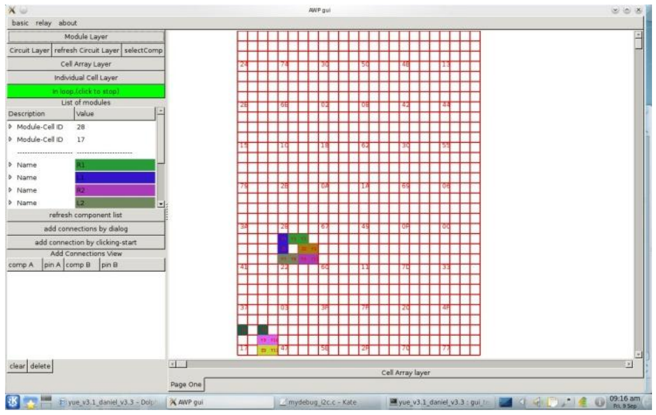


a) b)

Figure 4. Cell Unit. a) Conceptual Implementation. b) Photo that shows the connections (ribbon cables) to the neighboring Cell Units and the connection (90° header) to the AWP Planar Substrate.

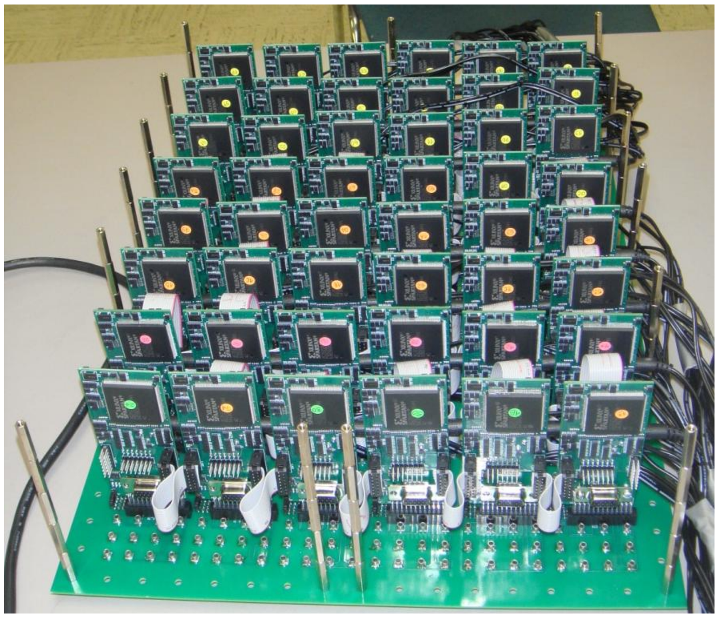


a)

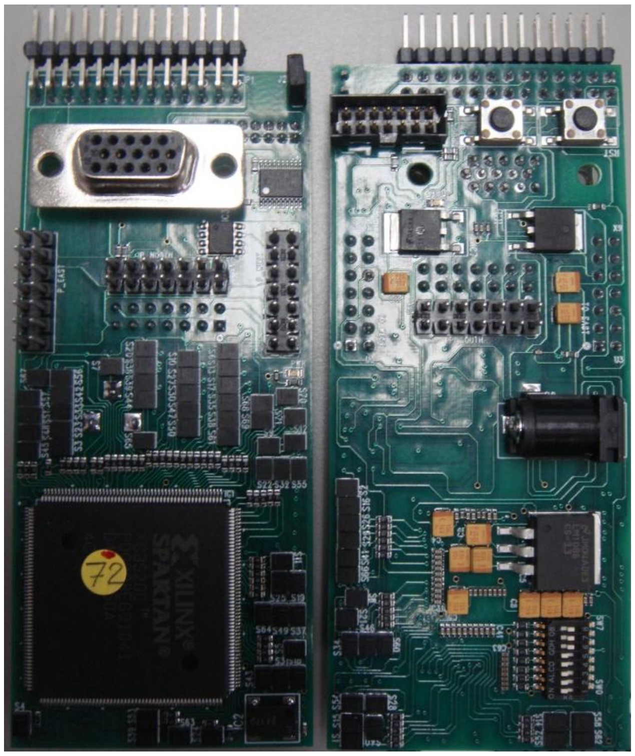


Description	Value
Module-Cell ID	28
Module-Cell ID	17
Name	R1
Name	L1
Name	R2
Name	L2

b)



c)



d)

Figure 5. AWP Final Prototype. a) Prototype in operation: view of the panel with 6x8 Cell Units beneath and 2 modules on top. b) GUI snapshot showing the 6x8 AWP with two modules connected to it. c) Upside-down view of the AWP. d) Close view of a Cell Unit: Back side and Front side. Relay: Panasonic AQY221R2M1Y, FPGA: Xilinx Spartan-3 XC3S200.

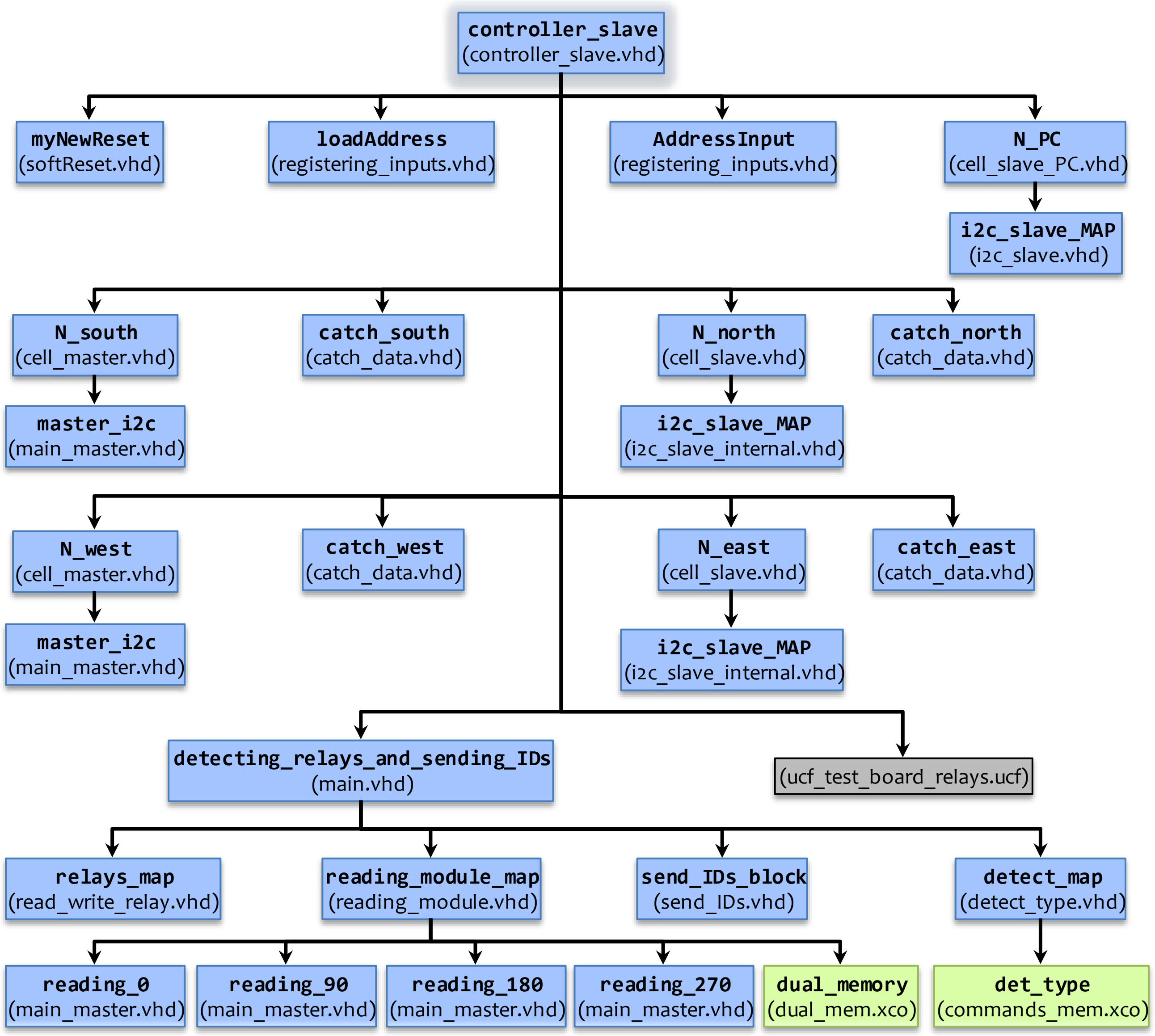
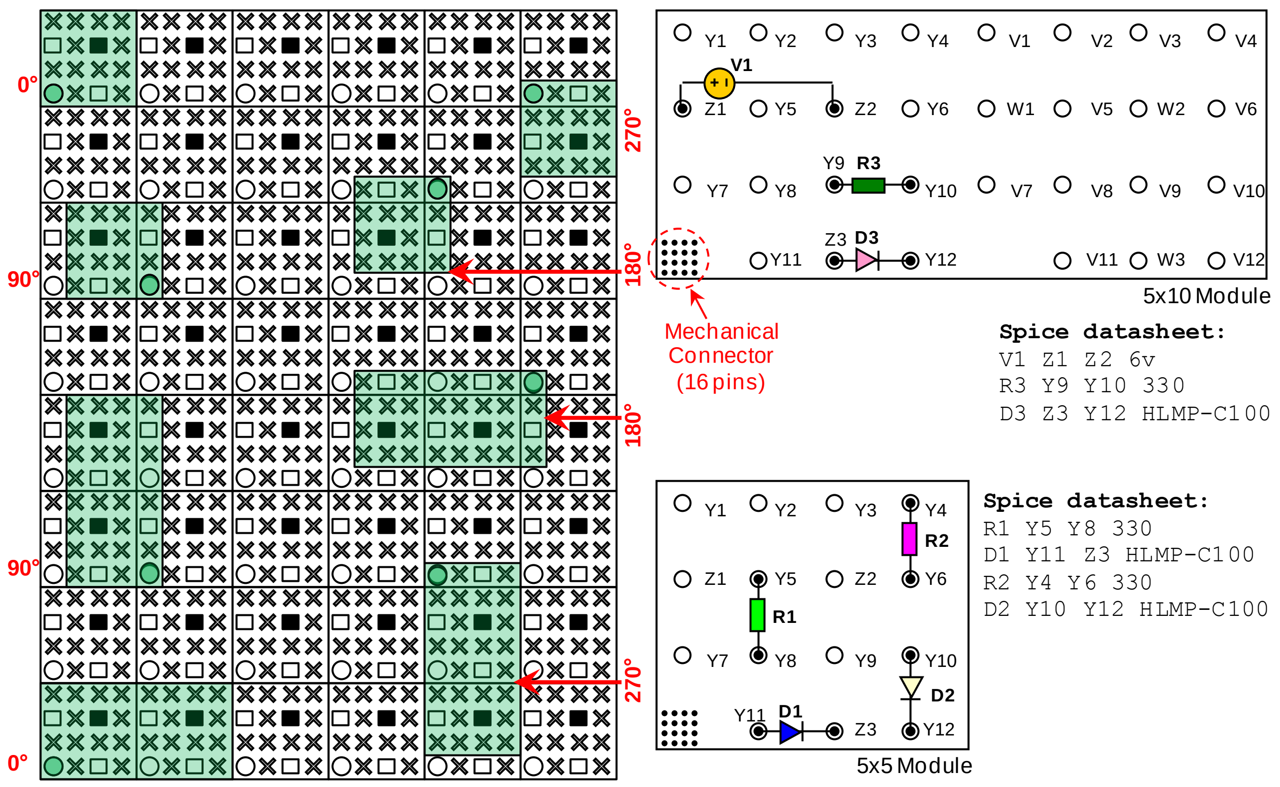


Figure 6. Dependency diagram of the hardware blocks (described in VHDL) that make up the Local Processing Unit within each Cell Unit.



```
Spice datasheet:
V1 Z1 Z2 6v
R3 Y9 Y10 330
D3 Z3 Y12 HLMP-C100
```

```
Spice datasheet:
R1 Y5 Y8 330
D1 Y11 Z3 HLMP-C100
R2 Y4 Y6 330
D2 Y10 Y12 HLMP-C100
```

Figure 7. AWP modules. a) 5x5 and 5x10 modules in all orientations (0°, 90°, 180°, and 270°). b) 5x5 and 5x10 modules with components and their respective datasheets. Note the naming convention for the pins (Y1-Y12, Z1-Z3, V1-V12, W1-W3).

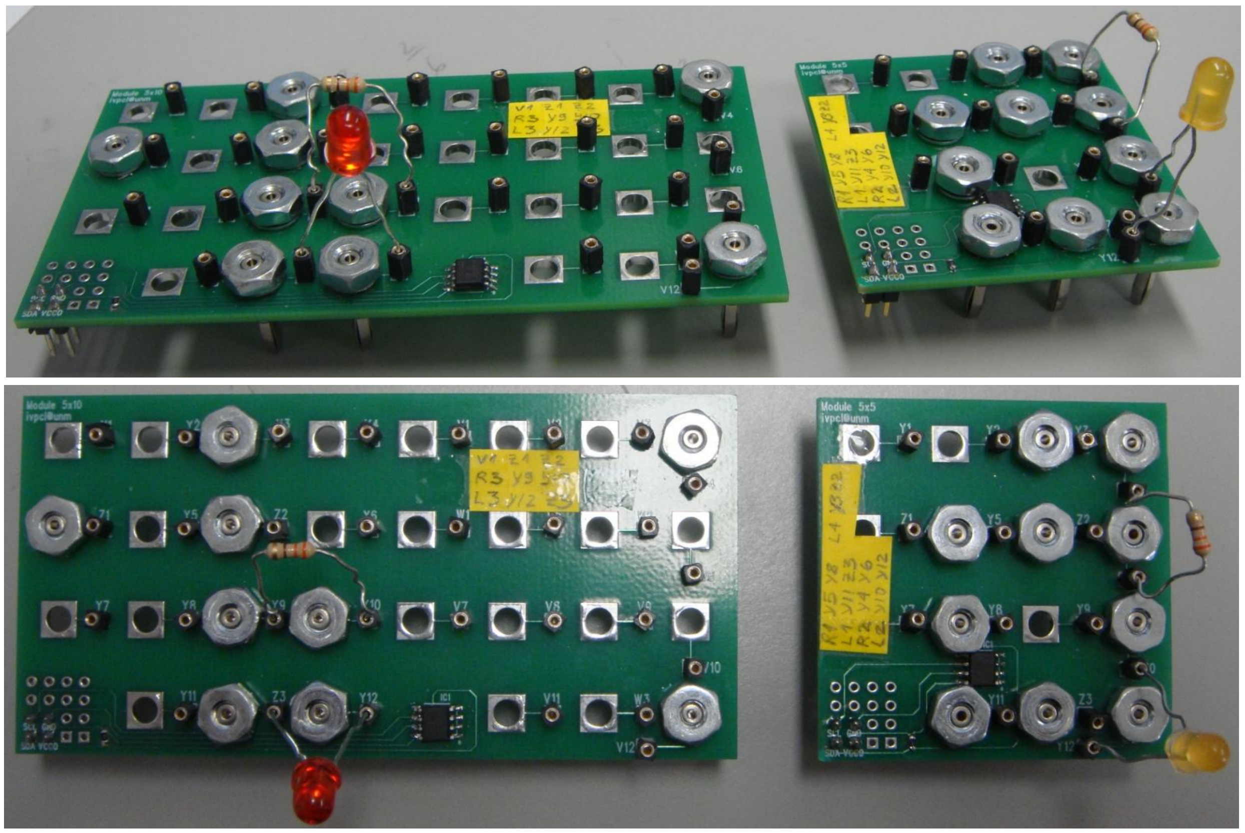


Figure 8. Modules of size 5x5 and 5x10. Top photo: View of the modules at an angle. Bottom photo: Top view of the modules

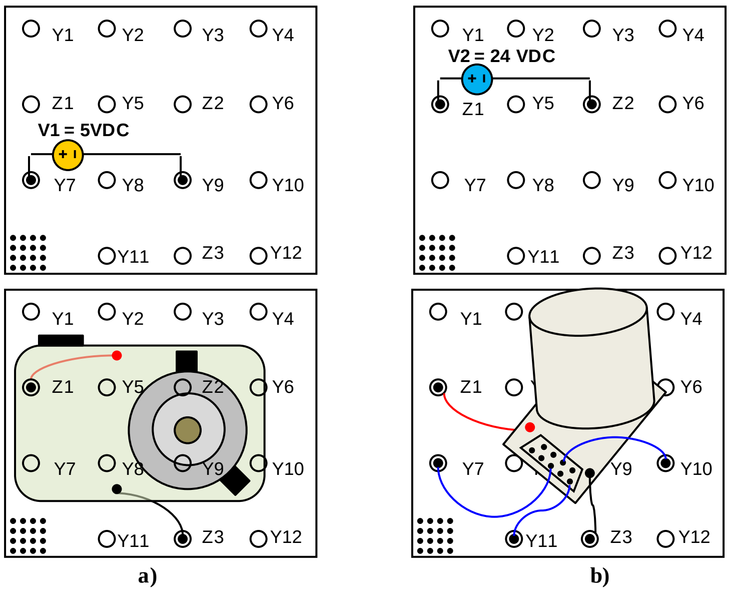


Figure 9. Examples of practical sub-systems that can be supported as independent Modules. a) Star Tracker (Vixen Polarie Star Tracker). b) A Reaction Wheel (MSCI MicroWheel 200).

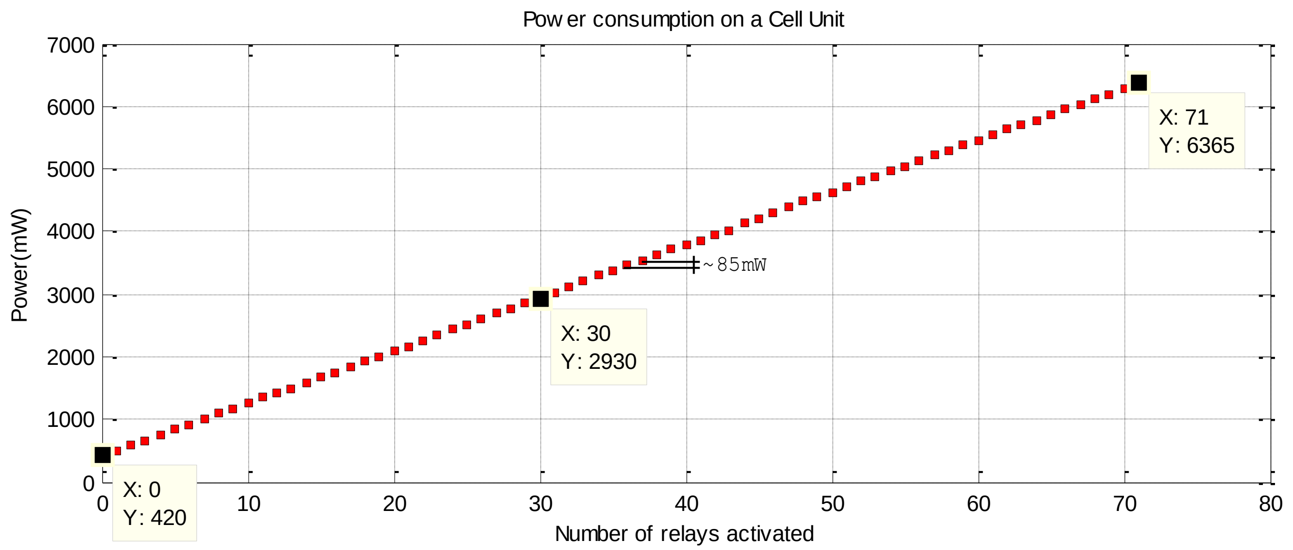
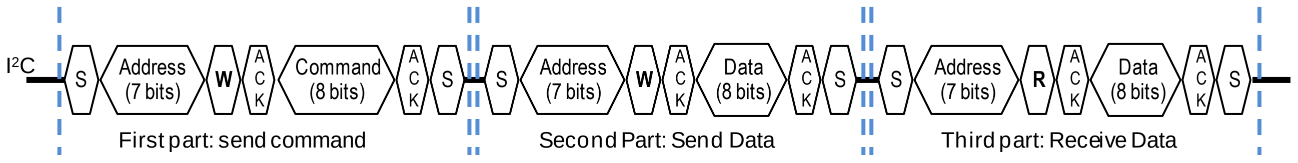


Figure 10. Power consumption of a Cell Unit of the AWP. As expected, the power increase is quasi linear. The idle power consumption is about 0.42 W. Each activated relay adds about 85mW.



a)

	Data								
Command/Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Explanation**
`0x5B` Read Relay Status	The number of the relay whose status will be read							0	One bit suffices to represent the relay status Thus, 0xFF means 'closed', and '0x00' means 'open'.
`0x60` Read datasheet word	The address of the datasheet memory (0-255) is specified								The specified address contains one byte that is relayed to the CMU. Max. memory size: 256

b)

Figure 11. Update to the Type IV command. The `0x60` command is added. a) Timing Diagram. b) Characteristics of data to be transferred for the `0x60` command as well as for the `0x5B` command.

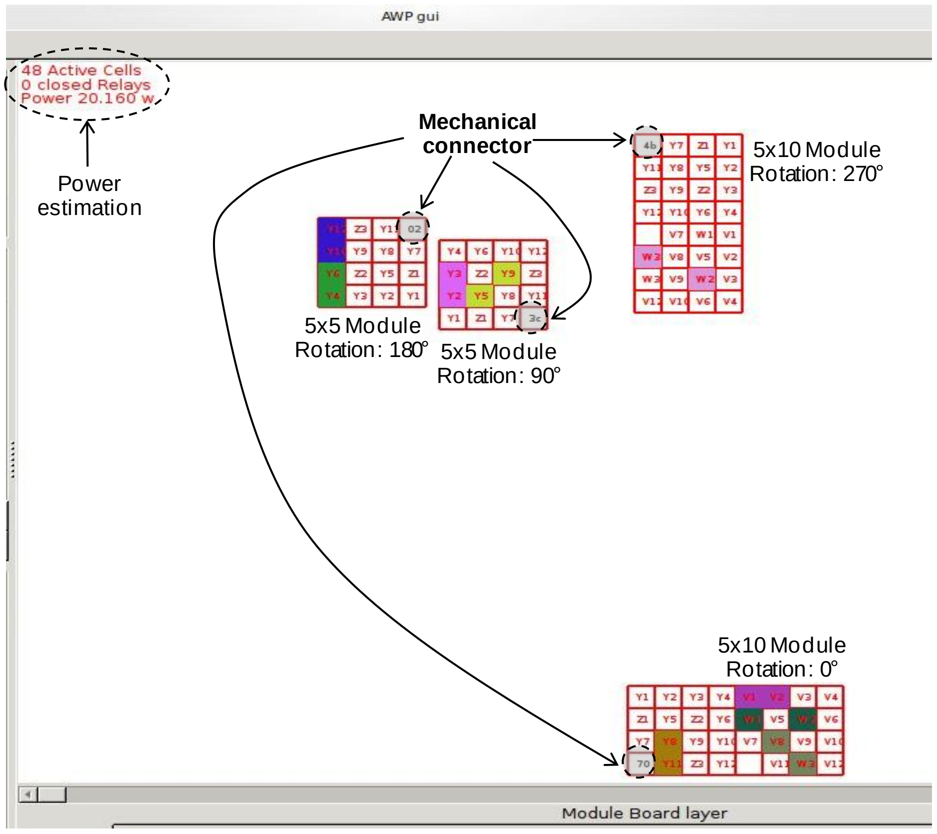


Figure 12. GUI Module Layer displaying the status of the AWP. Two modules of 5x5cm and two modules of 5x10cm are shown with different orientations.

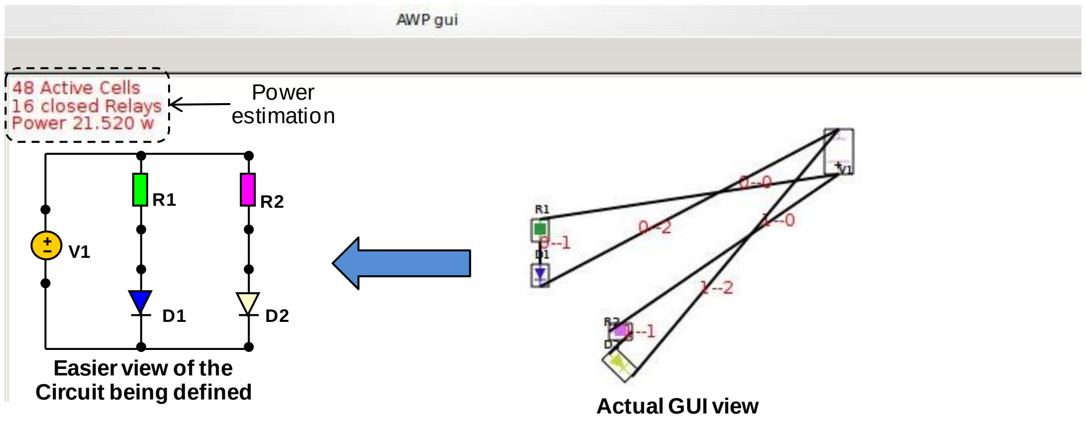
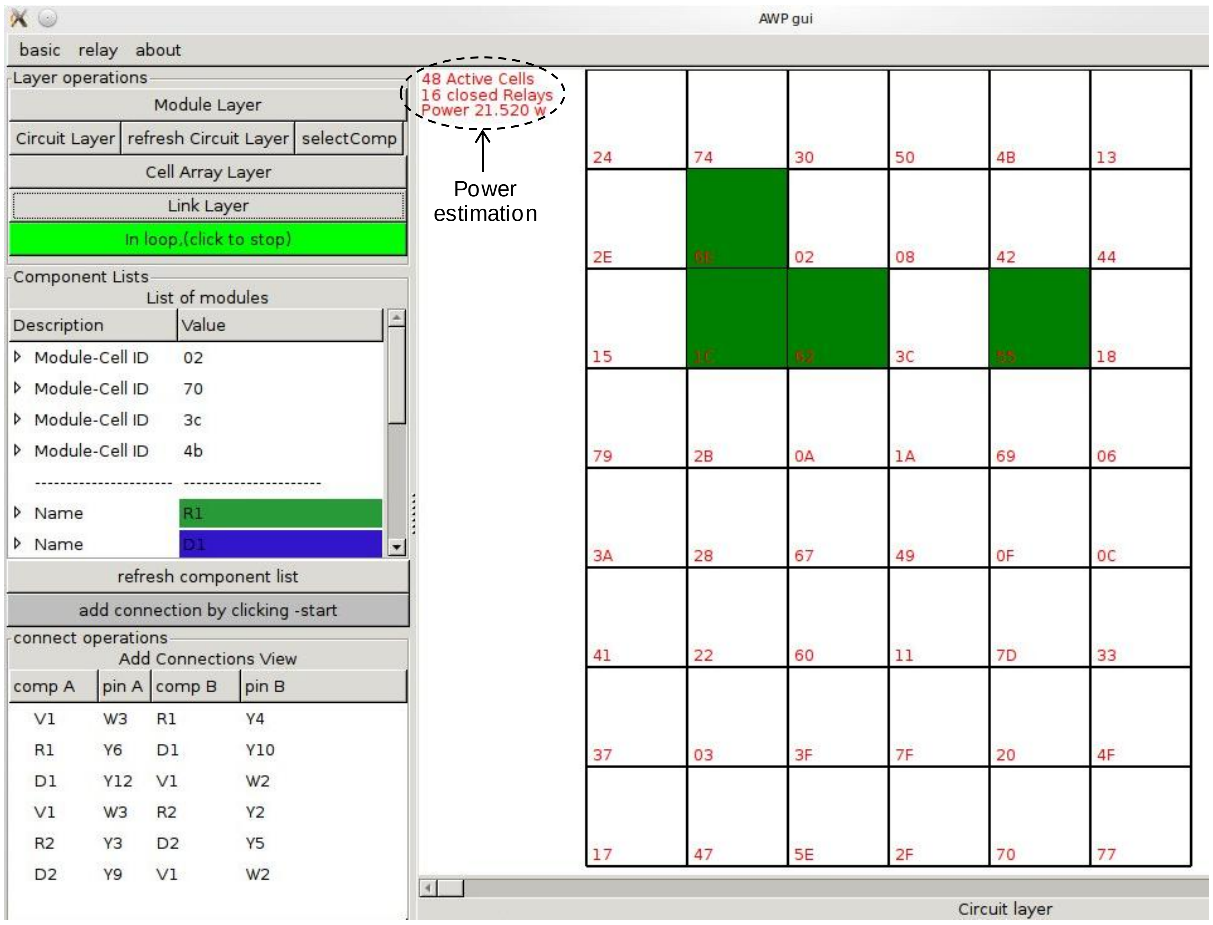


Figure 13. Circuit Layer of the GUI. The circuit on the right is what the GUI actually displays. The circuit on the left is shown for clarity purposes. The circuit requires 16 relays to be activated (as reflected in the top left corner)



48 Active Cells
16 closed Relays
Power 21.520 w

Component Lists

List of modules

Description	Value
Module-Cell ID	02
Module-Cell ID	70
Module-Cell ID	3c
Module-Cell ID	4b
---------------------	---------------------
Name	R1
Name	D1

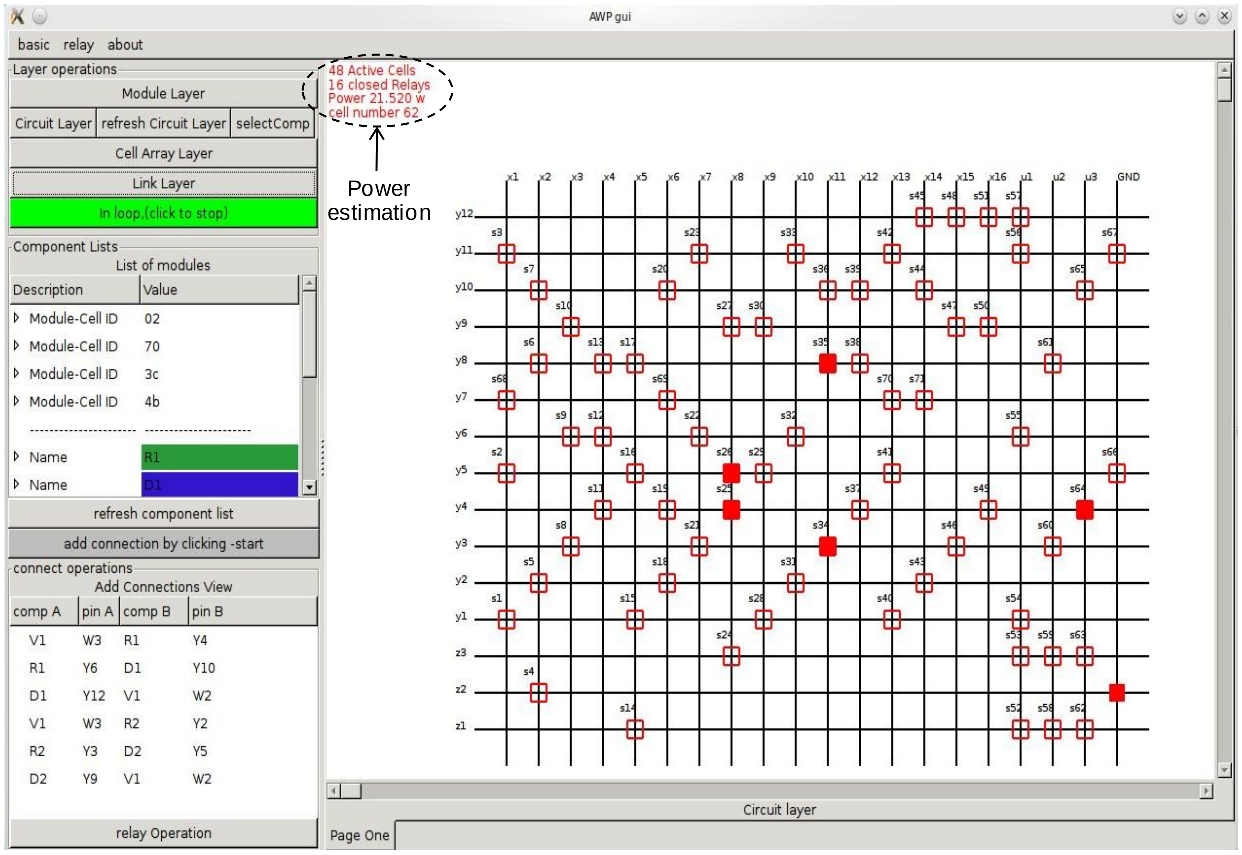
connect operations

Add Connections View

comp A	pin A	comp B	pin B
V1	W3	R1	Y4
R1	Y6	D1	Y10
D1	Y12	V1	W2
V1	W3	R2	Y2
R2	Y3	D2	Y5
D2	Y9	V1	W2

24	74	30	50	4B	13
2E	6E	02	08	42	44
15	1C	62	3C	55	18
79	2B	0A	1A	69	06
3A	28	67	49	0F	0C
41	22	60	11	7D	33
37	03	3F	7F	20	4F
17	47	5E	2F	70	77

Figure 14. Link Layer of the AWP. The 4 Cell Units in green indicate that they contain activated relays that implement the circuit of Fig. 13.



48 Active Cells
16 closed Relays
Power 21.520 w
cell number 62

Description	Value
Module-Cell ID	02
Module-Cell ID	70
Module-Cell ID	3c
Module-Cell ID	4b
---------------------	---------------------
Name	R1
Name	D1

comp A	pin A	comp B	pin B
V1	W3	R1	Y4
R1	Y6	D1	Y10
D1	Y12	V1	W2
V1	W3	R2	Y2
R2	Y3	D2	Y5
D2	Y9	V1	W2

Figure 15. Internal view of Cell Unit 0x62. The highlighted rectangles in red indicate activated relays (closed). Note that for a Cell Unit, the 71 relays are used to make connections among the pins that go into the planar substrate (y1-y12, z1-z3). The pins x1-x16, u1-u3, and GND are pins that go to the Cell Unit neighbors (North, South, East, West), these pins are useful because they can find more routing resources in other Cell Units.