JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 11, No. 6, June 2014

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

Daniel Llamocca*
University of New Mexico, Albuquerque, New Mexico 87131
Victor Murray†
Universidad de Ingeniería y Tecnología, Lima, Lima 43, Peru
Yuebing Jiang‡ and Marios Pattichis§
University of New Mexico, Albuquerque, New Mexico 87131
and
James Lyke¶ and Keith Avery**
U.S. Air Force Research Laboratory, Albuquerque, New Mexico 87112

DOI: 10.2514/1.I010127

The first prototype of an adaptive wiring panel was recently introduced that implemented a reconfigurable switch fabric that allows dynamic routing of analog, digital, and power signals for space system applications. In this paper, a complete redesign and reimplementation of the adaptive wiring panel system is considered to address issues associated with scalability, reliability, and real-time monitoring of the switching fabric. The new system is demonstrated using 48 cells as opposed to the six cells of the first adaptive wiring panel prototype. The hardware and software systems are open source, and recommendations are provided to support further extensions to the system.

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) for which the 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 a standard wiring harness or the use of standard printed circuit boards (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 rerouting and reimplement 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, whereas 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 rerouting. 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 rerouting to reimplement 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 reprogram 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

Received 5 April 2013; revision received 6 August 2013; accepted for publication 13 December 2013; published online 6 June 2014.

*Research Assistant Professor, Department of Electrical and Computer Engineering, MSC01-1100.
†Professor, Department of Electrical Engineering, Av. Cascanueces 2221, Lima, Lima 43, Peru.
‡Doctorate Candidate, Department of Electrical and Computer Engineering, MSC01-1100.
§Professor, Department of Electrical and Computer Engineering, MSC01-1100.
¶Technical Advisor, Space Vehicles Directorate.
**Senior Engineer, Space Vehicles Directorate.

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 reimplemented using the new component.

Figure 1 shows two adaptive wiring panel concepts, where each cell contains adaptive wiring resources for the modules and interconnection resources for the neighboring cells. A first hardware implementation of the AWP discussed in [1] is shown in Fig. 1a. To implement a circuit, circuit components are placed on modules. Modules include SPICE (simulation program with integrated circuit emphasis) descriptions of their components. The SPICE 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 prebuilt 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 reimplementing the connections. Thus, in the first prototype of the AWP described in [1], modules can be rearranged while maintaining the prespecified circuit through dynamic rerouting.

Unfortunately, there are several fundamental limitations of the original prototype that led to its redesign, for which the concept is shown in Fig. 1b. We summarize the basic issues that motivated the complete redesign as follows:

1) For scalability using an AWP substrate and redesigned cells. the original cell-unit arrangement depicted in Fig. 1a 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. 1b, 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. 1a.

2) For 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 presoldered 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.

3) For interintegrated circuit (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

4) For 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 graphical user interface (GUI) that provides real-time hardware monitoring of the system that is accurate down to the single switching unit (e.g., relay level).

5) For an open-source software and hardware system, we provide a hierarchical description of the source code [C code and very high-speed integrated circuits hardware description language (VHDL) code] for implementing the AWP software and hardware to support further research in this area.

The rest of the paper is organized as follows. In Sec. II, we present related work that was done before the AWP. In Sec. 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 Sec. V. Concluding remarks are given in Sec. VI.

II. Related Work

From the lowest to the highest circuit levels, wiring remains an open problem. A number of printed circuit boards 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

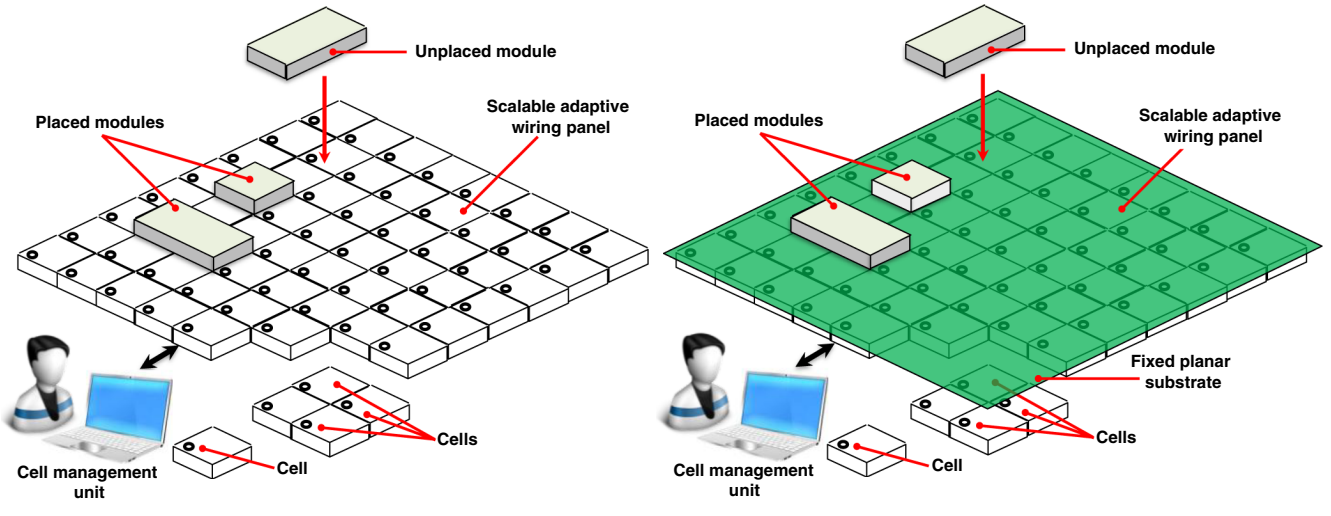


Fig. 1 Adaptive wiring panel concepts: a) original [1], and b) with a fixed planar substrate on top of the cells.

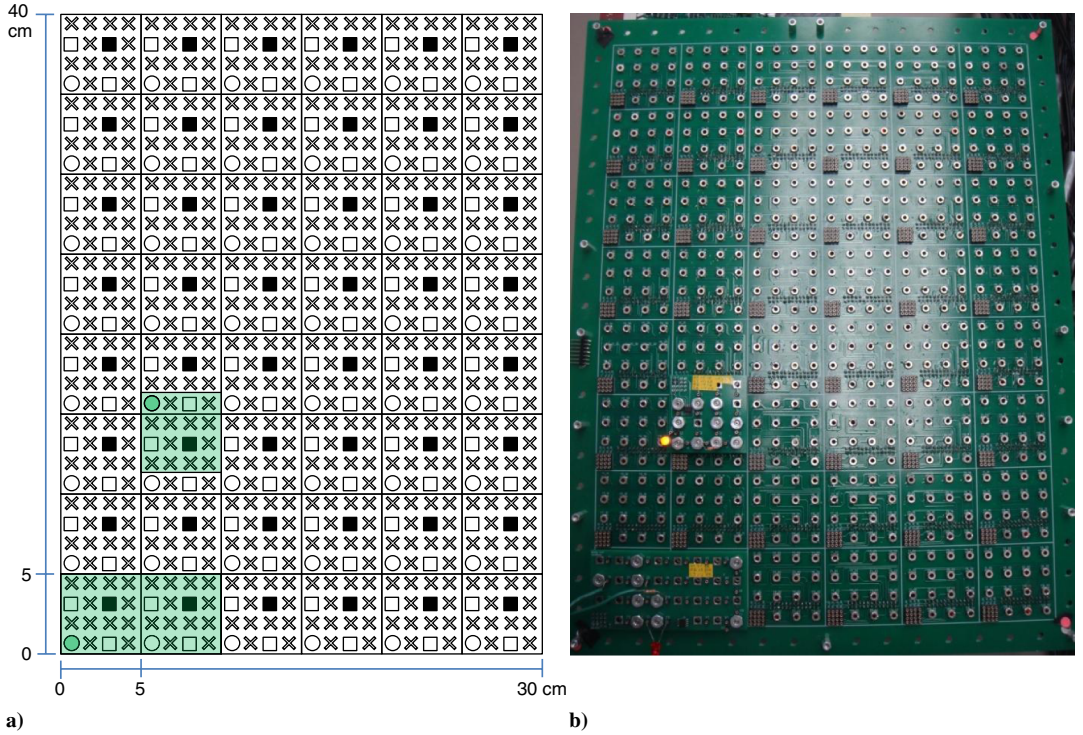


Fig. 2 AWP planar substrate: a) 48 cells in a 6×8 configuration, b) actual AWP substrate with connected modules [as indicated in (a)].

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 and Stroobandt [4] and Donath [5] for PCBs and integrated circuits (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, The U.S. Air Force Research Laboratory introduced the space plug-and-play avionics (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, also known as the plug-and-play satellite) [3]. Similar to the AWP prototype, the PnPSat-1 used a 5×5 cm pegboard-like grid, and short cables were 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 [7–10] and demonstrated in [10]. Algorithms for implementing self-healing dynamically reconfigurable AWMs were reported by Thompson and Mycroft in [11]. In this paper, 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 [7–10] but lacked scalability and robustness. The work by Dehon et al. [12] allowed 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 [1,15]. The present work details the final prototype of the adaptive wiring panel using $6 \times 8 = 48$ cell units.

III. 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 (5×5 cm) 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 (CUs) are interconnected 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 module configuration (position, orientation), and the netlist information specifying the characteristics and pin 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. 1a, the original prototype requires interconnection of the cells in a fixed grid architecture. This requirement was removed in the new prototype depicted in Fig. 1b because 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 redesigned 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 $= 0.8\Omega$).

The previous prototype [1] implemented the AWP with six 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 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 rerouting and plug-and-play placement of components. The platform allows for the testing of routing algorithms and other interconnection topologies. But, as 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 5×5 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 (5×5) or 30 pins (5×10). 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 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 using 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 use 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 5×10 module or rotated module).

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. 1b). In this way, we can guarantee a seamless plugging/unplugging of modules at different rotations and rearrangements.

The redesigned AWP substrate consists of 6×8 (30×40 cm) cell units. Figure 2 depicts the new AWP substrate with two modules (a 5×5 and a 5×10) plugged into the substrate. In Fig. 2a, each cell unit is shown with three types of connectors on it: 1) signal connectors (depicted with "×"), 2) power connectors (depicted with open boxes "□"), and 3) a mechanical connector (depicted with open circles "○:). We also included the ground connector, depicted with a solid box "■". Figure 2b shows a photograph of the AWP substrate.

B. Cell Unit

The cell unit (5×5 cm), also known as the AWP cell, is the minimum independent unit of the AWP. Figure 3a 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. Figure 3b 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: four lateral boards and one top board. As the new planar substrate consists of only one piece, the top board is no longer needed.

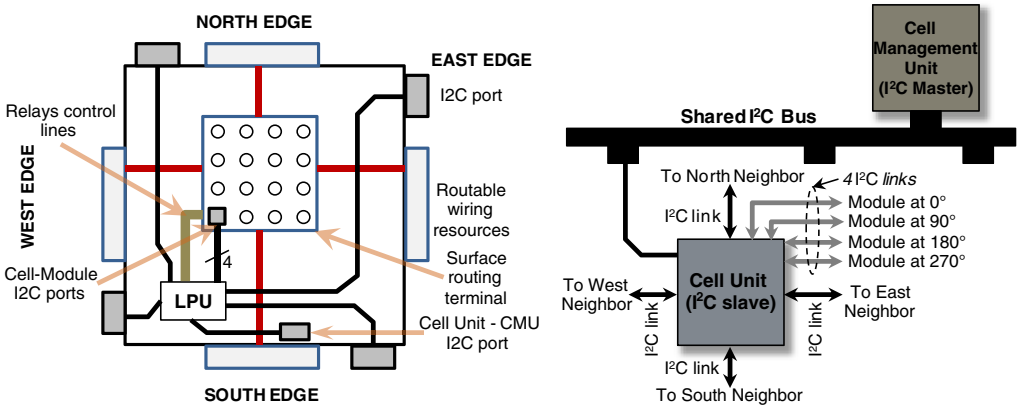


Fig. 3 Architecture of the cell unit: a) conceptual implementation, and b) I^2C links for the cell unit and CMU.

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 photograph of the cell unit. Note that the 90 deg connector contains 1) the pins of the surface routing terminal that connect to the AWP substrate, and 2) 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 6×8 array configuration. Figure 5a displays the AWP in operation, with two modules on top of the AWP planar substrate. Figure 5b shows the corresponding GUI snapshot where all the cell units and modules have been detected. Figure 5c 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. Figure 5d 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. Each relay is a Panasonic AQY221R2M1Y, and the FPGA is a Xilinx Spartan-3 XC3S200.

2. *Logical Processing Unit Inside the Cell Unit*

The logical processing unit, which 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 6×8 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 six cell unit configuration (previous prototype [1]), but it happened frequently in the 6×8-array configuration. Thus, the problem seemed to stem from the fact that the shared I^2C bus linked a large number of cell units ($6 \times 8 = 48$) to the cell management unit.

The communications problem was mitigated by modifying the logical processing unit so that it only read 1 B 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 Sec. IV). Specifically, we defined a new command (0×60) 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 1 B. 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 Fig. 6 depicts a diagram showing the dependency of the corresponding VHDL files.

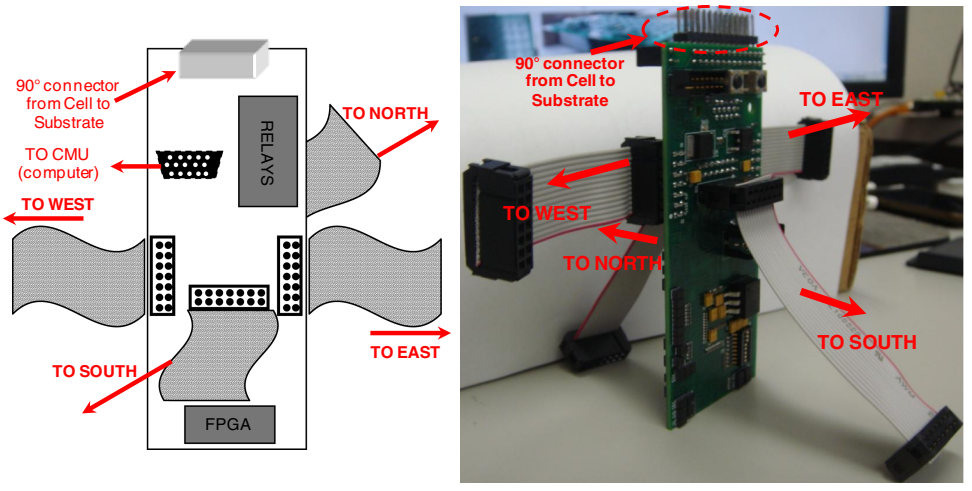
C. Modules

The previous prototype [1] supported modules of dimensions 5×10 cm. We performed the following modifications to the physical design of the modules:

1) To connect to the AWP substrate sockets, the modules were redesigned 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: 5×5 and 5×10 cm. The 5×5 module contains 12 signal connectors, 3 power connections, and 1 mechanical connector. The 5×10 module has 24 signal connectors, 6 power connections, and 1 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 B 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.



a) b)

Fig. 4 Cell unit: a) concept, and b) actual showing connections (ribbon cables) to the neighboring CUs and the AWP planar substrate.

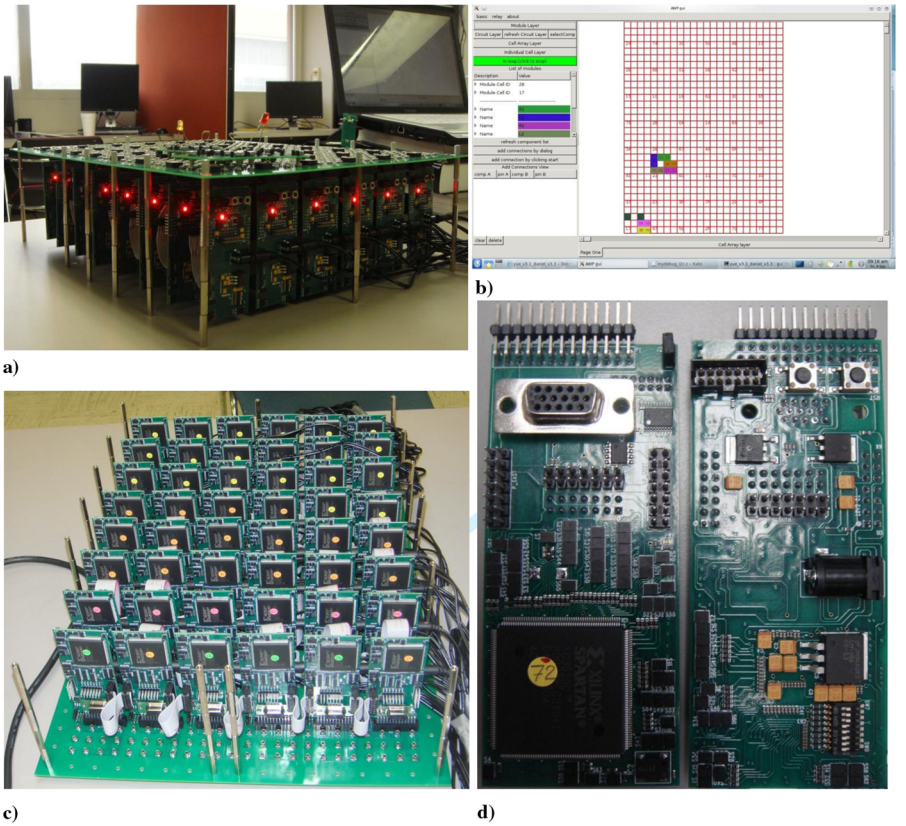


Fig. 5 AWP final prototype: a) prototype in operation; b) GUI snapshot; c) upside-down view; and d) close-up view of a cell unit.

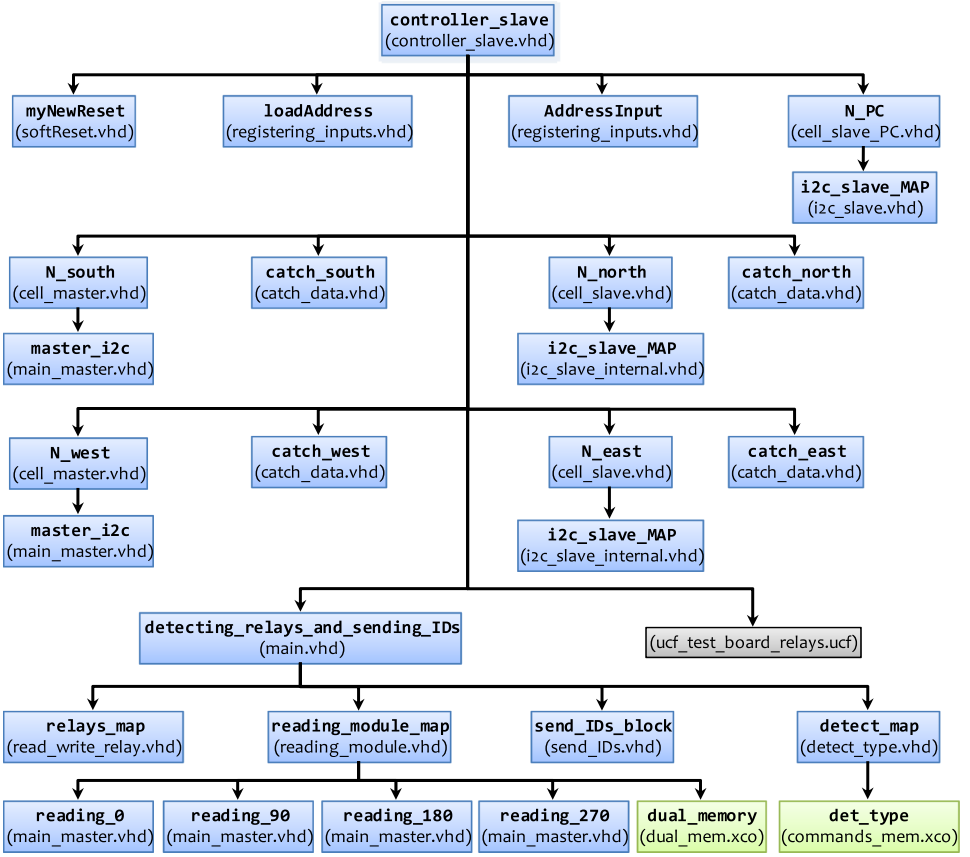


Fig. 6 Dependency diagram of the hardware blocks (described in VHDL) that make up the local processing unit within each cell unit.

Figure 7a depicts the modules (5 × 5 and 5 × 10 dimensions) when plugged into the AWP substrate. All the possible rotation configurations (0, 90, 180, and 270 deg) are depicted. Note how one 5 × 5 module can span up to four cell units, and one 5 × 10 module can span up to six cell units. Figure 7b shows the modules' depiction, with the naming conventions for the pins and the datasheets when some components are connected to the modules. Finally, Fig. 8 shows actual pictures of the 5 × 5 and 5 × 10 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. 7b, the two modules contain three resistors (R1, R2, and R3), three light-emitting diodes (LEDs) (D1, D2, and 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:

1) The first subsystem is a star tracker (e.g., Vixen Polarie Star Tracker) that requires dc 5 V. A camera is mounted on top of the star tracker. the star tracker is the sole component in a module. The dc 5 V source is connected to another module.

2) The second subsystem is a reaction wheel (e.g., MSCI MicroWheel 200). It requires dc 24 V and a RS-485 port for three control lines. The dc 24 V source belongs to another module. The three control lines could be connected to another module containing a microcontroller.

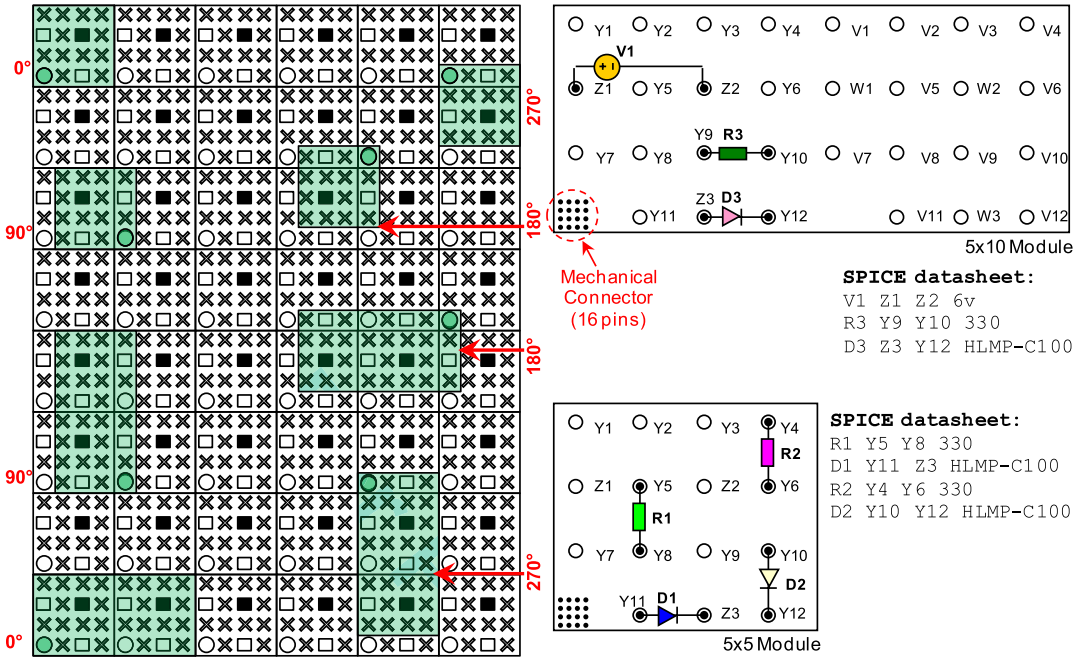


Fig. 7 AWP modules: a) 5 × 5 and 5 × 10 modules in all 4 orientations, and b) modules with components and their datasheets.

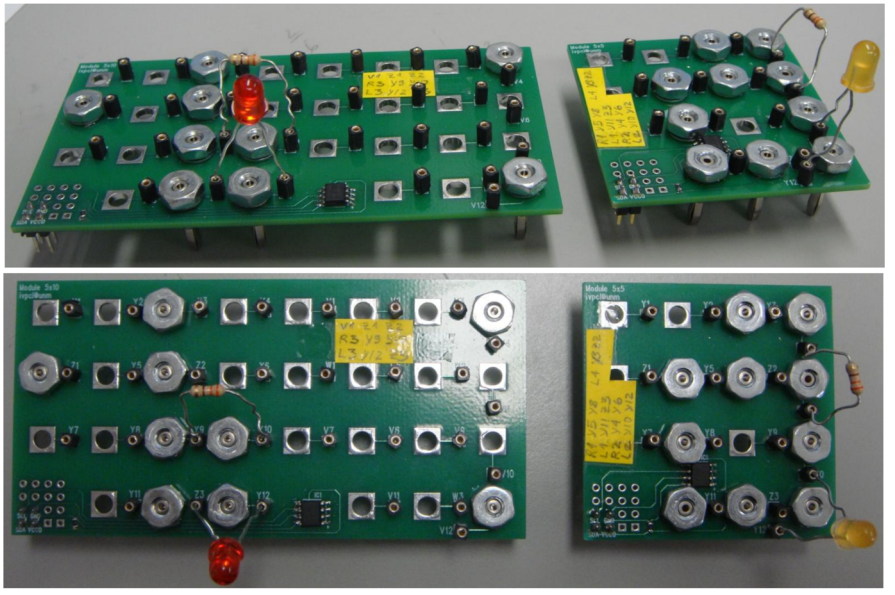


Fig. 8 Photographs of 5 × 5 and 5 × 10 modules.

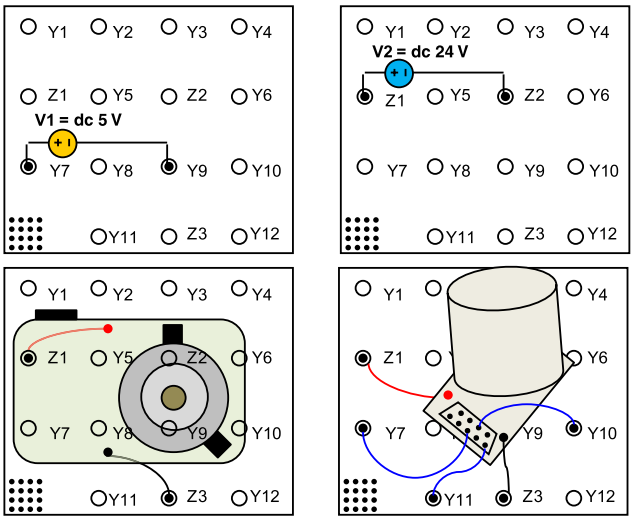


Fig. 9 Examples of subsystems that can be supported as independent modules: a) star tracker, and b) a reaction wheel.

Note how, in both cases, we can easily connect/disconnect the power source from the subsystems by activating /deactivating 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 milliwatts) 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 85 mW. 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 $6 \times 8 = 48$ cell units is about 20.16 W, whereas the maximum possible power consumption (considering all relays within the 48 cell units are activated) is about 305.5 W. In practice, however, we expect that the average number of relays activated per cell not to exceed 10, resulting in the AWP drawing 60.48 W 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 20.16 W. Most of this power is drawn by the FPGAs. Also, note that the miniature relays used here (Panasonic AQY221R2M1Y, size: 2.95×2.2 mm^2) draw 85 mW when activated (i.e., they are in the on state). Latching relays (such as the Omron G6KU-2 F-Y, size: 6.5×10 mm^2) would reduce power consumption, as it requires 21 mA at 5 V 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 application-specific ICs 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. 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 1) reading the configuration of the cell units and representing them as an undirected graph, 2) reading the configuration of the modules and their netlist specification, 3) providing the user with an interface to specify a circuit (or circuits), 4) routing all required connections for the specified circuit(s), and 5) 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

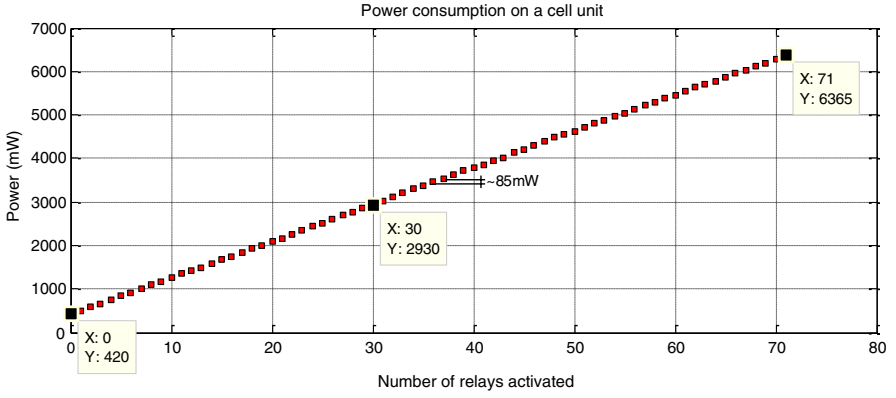


Fig. 10 Power consumption of a cell unit of the AWP. As expected, the power increase is linear.

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: 1) a command-line interface that provides a complete control of the AWP, used mainly for debugging purposes; and 2) a graphical unit interface, which goes on top of the command-line interface, which 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.

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, and then the actual data) to the CMU. This simple approach did not account for reliability issues, as explained in Sec. III.B.2.

The new approach requires reading only 1 B at a time from the I^2C memory inside the module. This is implemented by including a new 0×60 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 0×60, followed by the address of the I^2C memory (0 to 255) from which we want to read data (1 B). 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 0×60 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 10 attempts before branding that CU as nonexistent or failing. Thus, we perform up to 10 attempts for each of the following operations: sending the 0×60 command, sending the address of the I^2C memory, and receiving 1 B of data. In addition, we issue the 0×60 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 (0×00) that indicates that we have reached the last byte of information.

B. Graphical User Interface: Real-Time Monitoring

The GUI was developed to provide a smoother user experience when managing the adaptive wiring panel. It was developed based on the GTK + library [18].

As in [1], the GUI automatically updates what it displays when 1) the relative position of the cell units change; 2) when modules are replaced, removed, or rotated; and 3) 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 Fig. 2. A snapshot of the cell array layer (6×8) is shown in Fig. 5b. Figure 12 shows a snapshot of the module array layer where four 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 Fig. 13. The circuit requires 16 closed relays, and as such, the estimated power has increased.

As for the link layer, Fig. 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 boxes. Note that, for a cell unit, the 71 relays are used to make connections among the pins that go into the planar substrate ($y1$–$y12$ and $z1$–$z3$). The pins $x1$–$x16$, $u1$–$u3$, and GND are pins that go to the cell unit neighbors (north, south, east, and west); these pins are useful because they can find more routing resources in other cell units

C. Open-Source Interface

To support further research in this area, we provide an open-source implementation of the interface (general public license). For the command line interface, we provide a summary of the code structure and main functions in the Appendix.[††]

V. Recommendations for Future Work

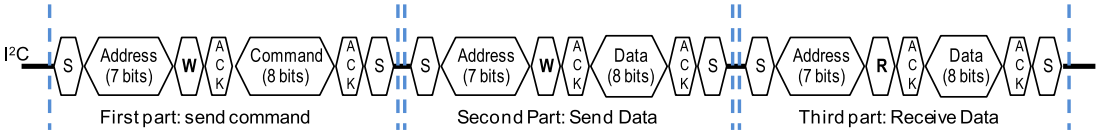
In this section, we provide a list of recommendations for future work on the AWP. We believe that this list will contribute to the wider adoption of the AWP:

1) For the 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]).

2) For dynamic routing algorithms, the current algorithm is based on a heuristic that uses the shortest-path algorithm to route each connection. The search starts 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 6×8 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 because the routing problem is known to be nondeterministic polynomial-time hard. Dynamic routing would be best performed using a Steiner forest algorithm, as described in [19].

3) For the 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 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.

[††]The password-protected code (available upon request) can be found at www.ivpcl.org/AWPcode.zip [retrieved 2014].



a)

Command/Bits	Data 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)

Fig. 11 Type IV command, example: a) timing diagram, and b) data for the 0 × 5B command and the 0 × 60 command.

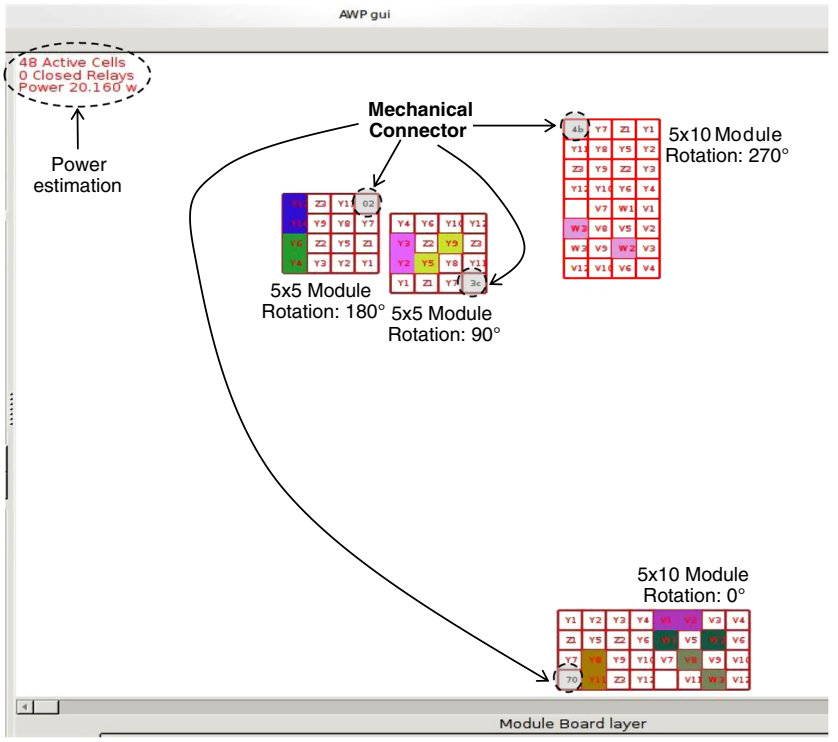


Fig. 12 GUI module layer displaying the AWP status. Two modules are shown.

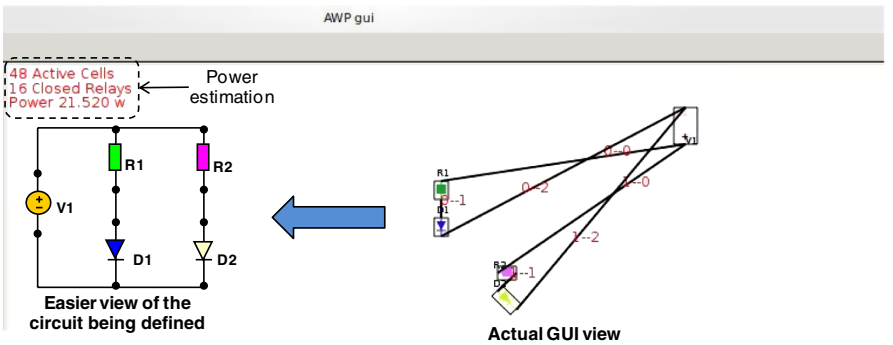


Fig. 13 GUI Circuit layer. The circuit on the right is what the GUI actually displays.

4) For 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.

5) For 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.

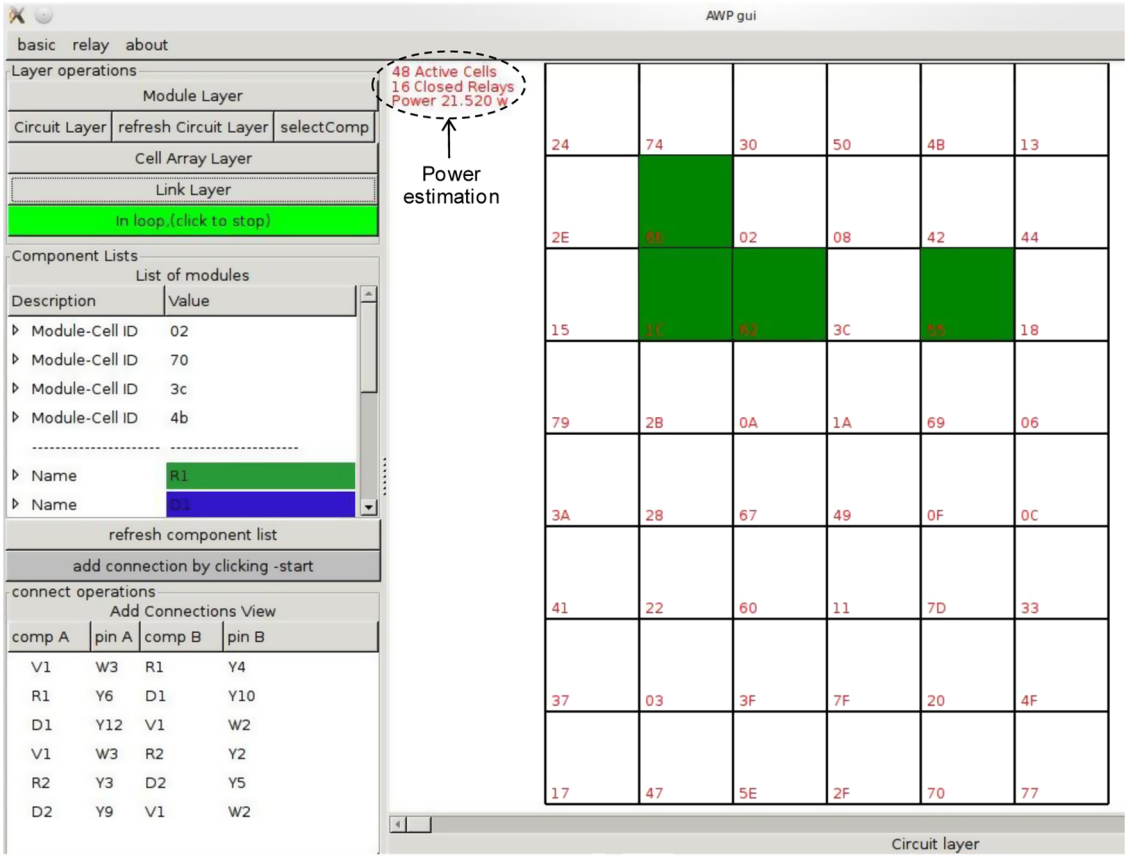


Fig. 14 GUI Link Layer. The four shaded cell units indicate that they contain activated relays that implement the circuit of Fig. 13.

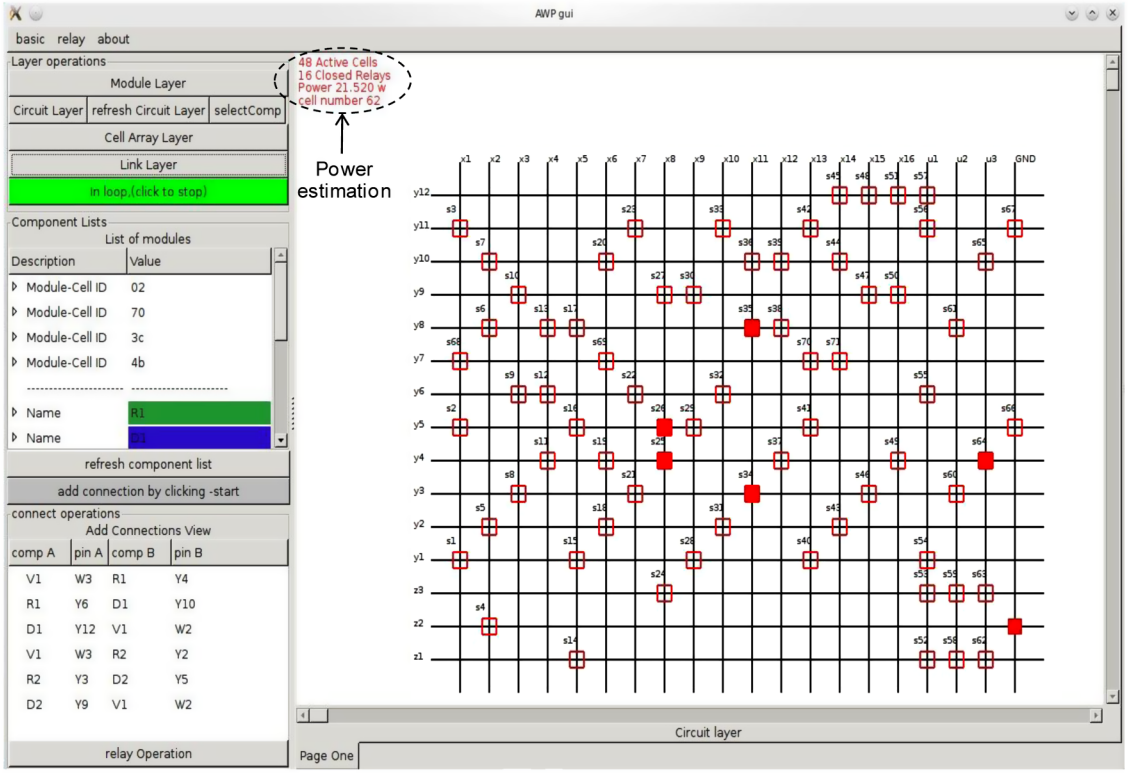


Fig. 15 Internal view of cell unit 0×62. The solid squares indicate activated relays (closed).

6) For 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.

7) For 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 Sec. III.

8) For 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.

9) For 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, 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., International Rectifier's RDHA701CD10A2N), although they come at a steep price. Finally, as the cells of the AWP are made of many other components besides the FPGA and the relays, it is recommended to have some kind of shielding.

10) For robustness, before 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

The final prototype of the adaptive wiring panel ($6 \times 8 = 48$ cells) has been described that can interconnect (in principle) arbitrary electronic components together using a large reconfigurable mesh. The adaptive wiring concept can be thought of 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 they 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 (six 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 redesign of the cell unit, and redesign of the physical connections among cell neighbors. Real-time monitoring tools were also introduced 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 testbed for improving the routing algorithms, communication reliability, and 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], it is believed that there is benefit in extending the aforementioned implementation concepts to three dimensions. Overall, it is hoped that the research will contribute to the concept of a universal reconfigurable switching network that supports dynamic routing of different types of signals.

Appendix: Command-Line Interface of the Adaptive Wiring Panel

In this Appendix, we provide a succinct description of all the software routines and functions 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:

1) The first concept is the cell unit. It is a minimum independent unit of the AWP that contains wiring resources (relays, connections to neighbors) and an FPGA. The FPGA controls a) the I^2C communications of the CU with the CU's neighbors and the CMU, and b) the relays.

2) The second concept is the grid. It is an array of cell units for which the arrangement is mechanically modifiable by the user. In the software routine, it is represented by the structure M.

3) The third concept is the 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 at 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.

4) The fourth concept is the module array. It is the array of modules for which the 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.

5) The fifth concept is the cell-level circuit. It is a set of connected pairs (represented by structure Circ) or a set of to-be-connected pairs (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).

6) The sixth concept is the module-level circuit. It is a 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: <node_name>-<cell ID>. The available node names are $Z1$–$Z3$ and $Y1$–$Y12$ for 5×5 modules, and $Z1$–$Z3$, $Y1$–$Y12$, $V1$–$V12$, and $W1$–$W3$ for 5×10 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 Fig. 5a. We now provide a description of the available commands. Table A1 lists the commands available when we type mytesti2c-interface. Table A2 lists the two commands available when we type mytest_i2c -interface_2. This interface is a little different from the one of Table A1. It immediately enters an infinite loop in which the user can always set, modify, or add circuit connections. Table A3 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 (5×5 or 5×10) and a SPICE-formatted list of components inside a module. In addition, the use of these commands requires 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 A4 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 A1 List of commands of the command-line interface accessible when typing mytest_i2c-interface

Command	Description
AWP>>generate_grid	Creates a connected grid of cell units (based on the detected CUs). Figure 5b shows the GUI visualization of a grid with 6×8 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 regenerates 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. 5b)
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 module 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 reroutes 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 A2 List of commands of the command-line interface accessible when typing mytest_i2c -interface_2

Command	Description
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 the following: 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 A3 Command line interface commands for writing/reading the I^2C memory inside a module

Command	Description
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: 5×10 modules, 4 components 5×10 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 A5 lists three commands for low-level debugging purposes. These commands handle the basic I^2C communication between a computer and a cell unit (I^2C peripheral). The user can write/read a byte to/from a given I^2C peripheral. The peripheral can be either the FPGA inside the cell unit or the I^2C memory inside the module (provided there is a 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^2C command is broadcast to all cell units; and when reading, the routine grabs data from the first

Table A4 Command line interface commands for debugging purposes

Command	Description
my_test_i2c <cell ID> -relays -{off,on}	Opens (off) or closes (on) the 71 relays of the given cell unit [cell identification (ID)].
my_test_i2c -relays {off,on}	Opens (off) or closes (on) all relays of every cell unit.
my_test_i2c -scan	It scans all cell units ($0 \times 02 - 0 \times 7F$) and lists whether each cell unit exists.
my_test_i2c <cell ID> -neighbors	It gets the neighbors' IDs (if any) of the given cell unit (cell ID). When a neighbor is not present, its ID is 0×1.
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}	Opens, closes, or gets 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 $0 \times$ AB mytest_i2c 0 C -relay 12 –closed % closes relay 12 from cell $0 \times$ 0C mytest_i2c 10 -relay 32 –status % returns status of relay 32 of cell 0×10

Table A5 Commands for low-level debug

Command	Description
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 0×03
mydebug_i2c -s	It runs a scan of all cell units (0x02–0x7F) and lists whether each cell unit exists.

Table A6 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^2C communication with the CMU (computer).
i2c-linux.c	Low-level functions for I^2C communication with a computer.
my_i2c_commands.h	Basic routines for I^2C 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	

cell unit that responds. The value `CellID = 0x01` is not allowed, as it means an inexistent cell unit (only used when reading neighbors of a cell unit)

Table A6 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.

References

[1] Murray, V., Llamocca, D., Lyke, J., Avery, K., Jiang, Y., and Pattichis, M., "Cell-Based Architecture for Adaptive Wiring Panels: A First Prototype," *Journal of Aerospace Information Systems*, Vol. 10, No. 4, 2013, pp. 187–208. doi:10.2514/1.I010024

[2] Jacobs, A., Cieslewski, G., George, A. D., Gordon-Ros, A., and Lam, H., "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, 2012. doi:10.1145/2392616

[3] Fronterhouse, D., and Lyke, J., "Plug-and-Play Satellite," *International SpaceWire Conference*, Dundee, Scotland, U.K., 2007.

[4] Christie, P., and Stroobandt, D., "The Interpretation and Application of Rent's Rule," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 8, No. 6, 2000, pp. 639–648. doi:10.1109/92.902258

[5] Donath, W., "Placement and Average Interconnection Lengths of Computer Logic," *IEEE Transactions on Circuits and Systems*, Vol. 26, No. 4, 1979, pp. 272–277. doi:10.1109/TCS.1979.1084635

[6] Lyke, J., "Plug-and-Play as an Enabler for Future Systems," *AIAA Space Conference*, AIAA Paper 2010-8660, Sept. 2010.

[7] Lyke, J., "Bringing the Vision of Plug-and-Play to High-Performance Computing on Orbit," *13th Annual Workshop on High Performance Embedded Computing*, Lexington, MA, Sept. 2009.

[8] Lyke, J., "Space Plug-and-Play Avionics (SPA): A Three-Year Progress Report," *AIAA Infotech Conference*, AIAA Paper 2007-2928, May 2007.

[9] Lyke, J., Fronterhouse, D., Lanza, D., and Byers, T., "A Plug-and-Play Concept for Spacecraft," *Military and Aerospace Programmable Logic Devices (MAPLD)*, NASA, Washington, D.C., Sept. 2005.
[10] Wilson, W., Lyke, J., and Forman, G., "MEMS-Based Reconfigurable Manifold," *Military and Aerospace Programmable Logic Devices (MAPLD)*, NASA, Washington, D.C., Sept. 2005.
[11] Thompson, S., and Mycroft, A., "Self-Healing Reconfigurable Manifolds," *Designing Correct Circuits (DCC'06)*, Vienna, Austria, March 2006.
[12] DeHon, A., Huang, R., and Wawrzynek, J., "Hardware-Assisted Fast Routing," *IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, April 2002, pp. 205–215.
[13] Murray, V., Feucht, G., Lyke, J., Pattichis, M., and Plusquellic, J., "Cell-Based Architecture for Reconfigurable Wiring Manifolds," *Infotech@Aerospace Conference*, AIAA Paper 2010-3497, April 2010.
[14] Murray, V., Llamocca, D., Jiang, Y., Lyke, J., Pattichis, M., Achramowicz, S., and Avery, K., "Adaptive Wiring Panels Using Cell-Based Architectures: A First Approach," *Reconfigurable Systems DCT Workshop*, Albuquerque, NM, Nov. 2010.
[15] Murray, V., , Llamocca, D., Jiang, Y., Pattichis, M., Lyke, J., Achramowicz, S., and Avery, K., "Cell-Based Architecture for Adaptive Wiring Panels: A First Approach," *AIAA Reinventing Space Conference*, Los Angeles, CA, May 2011.
[16] "The I2 C Bus Specification," *Philips Semiconductors*, Philips, Eindhoven, The Netherlands, 2000, pp. 1–46.
[17] Adams, G. I., Agrawal, D., and Siegel, H., "A Survey and Comparison of Fault-Tolerant Multistage Interconnection Networks," *Computer*, Vol. 20, No. 6, 1987, pp. 14–27.
doi:10.1109/MC.1987.1663586
[18] Krause, A., *Foundations of GTK+ Development*, Apress, Berkeley, CA, 2007, pp. 1–630.
[19] Feldman, M., Kortsarz, G., and Nutov, Z., "Improved Approximating Algorithms for Directed Steiner Forest," *Journal of Computer and System Sciences*, Vol. 78, No. 1, 2012, pp. 279–292.
doi:10.1016/j.jcss.2011.05.009

G. Brat
Associate Editor