The Satellite Stem Cell Architecture

A. O. Erlank, C. P. Bridges
Surrey Space Centre (SSC)
University of Surrey
Guildford, Surrey, United Kingdom
{a.erlank, c.p.bridges}@surrey.ac.uk

***Abstract*—Low-cost satellites continue to grow in popularity and capability, but have shown poor on-orbit performance to date. While traditional satellite missions have relied upon expensive fault prevention techniques, such as component screening, the use of radiation hardened components, and extensive test campaigns, low-cost missions must focus on fault tolerance, instead. This paper describes a novel, fault-tolerant system architecture, named Satellite Stem Cells. The Satellite Stem Cell Architecture, which is based on artificial cells, evolved from research into traditional reliability theory, bio-inspired engineering, and agent-based computing. Traditional reliability theory points towards k-out-of-n architectures for their superior reliability, while cell biology demonstrates how to build extremely multifunctional subsystems. Finally, agent computing provides a solution for facilitating the cooperation of a set of autonomous cells in a peer-to-peer environment. This paper describes the development of the architecture, details the artificial cell design, and gives preliminary implementation details.**

I. Introduction

The intrinsic physical and managerial limitations of low-cost satellite missions, such as Cubesats, have led to the popularity of commercial, off-the-shelf (COTS) components, single-string architectures, and reduced test campaigns. Unfortunately, these design decisions have also contributed to the poor on-orbit reliability of low-cost satellite missions seen to date [1], [2].

This research describes a novel, bio-inspired system architecture, named Satellite Stem Cells, which aims to increase system reliability, while minimising implementation overheads.

Three areas of research contributed to the development of the Satellite Stem Cell Architecture. Firstly, traditional reliability analysis points towards k-out-of-n system architectures for their increased reliability over other forms of redundancy [3].

Secondly, the study of biological life reveals nature's techniques for making highly multifunctional subsystems and highlights the benefits of a multicellular architecture. Bio-inspired computing is a large research field and many projects, including Embryonics [4], SABRE [5], eDNA, [6] and eTissue [7], have gained inspiration from multicellular organisms. However, this research aims to extend the multicellular concept beyond the cell-based computation aims of these projects, to a practical satellite avionics architecture.

Finally, the field of agent computing demonstrates how pieces of software can be viewed as living entities, granting a system autonomy, flexibility and fault tolerance. While agent-based systems have been implemented in a diverse range of applications, including traffic control, network management, scientific computing, and real-time control systems, applications on board satellites have been limited [8]. The first example of an agent running on board a satellite occurred in 1999 on the Deep Space One mission, where control of the satellite was temporarily handed to a single, experimental agent [9]. More recently, Princeton Satellite Systems developed the ObjectAgent environment for the Techsat 21 mission [10], and Bridges proposed using existing agent environments on board satellites using a hardware Java interpreter [11]. However, neither of these systems have been demonstrated on-orbit. This research aims to contribute to the state of the art by developing a real-time agent environment focused on reliability and low power consumption, making it particularly well suited for low-cost satellite applications.

II. Reliability Analysis

Techniques for increasing reliability can be divided into two broad categories, namely, fault intolerance and fault tolerance. Fault intolerance attempts to prevent failures from occurring in the first place. Techniques in this category include component screening, using radiation hardened components, and performing extensive test campaigns. These techniques are typically time consuming and expensive to implement, but have been a staple of the space industry for decades. Fault tolerance, on the other hand, accepts the fact that failures will occur. It focuses on ensuring that the system can survive or recover from foreseeable failures. Techniques in this category include most forms of redundancy and error detection and correction (EDAC) algorithms. Since low-cost satellite missions are typically forced into the extensive use of COTS components and reduced test campaigns, focus should be on fault tolerance.

The probabilistic time before failure of every component in a system can be described with a failure distribution. The failure distribution gives information about the expected lifetime of the component and about its instantaneous failure rate at any moment in time. Failure rates can be constant, indicative of charged particle damage, decreasing, indicating infant mortality, or increasing, due to component wear-out. Determining the failure distribution of an individual component, such as an integrated circuit (IC), can be done through accelerated life testing. Since accelerated life testing typically involves thousands of the component under test to gather statistically relevant results, it is impractical for determining the reliability

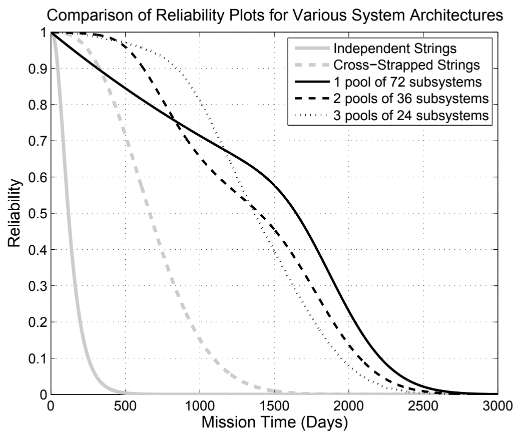


Fig. 1. Reliability plots of systems with comparable amounts of physical redundancy, but different system architectures. The 3 most reliable architectures are versions of k-out-of-n architectures

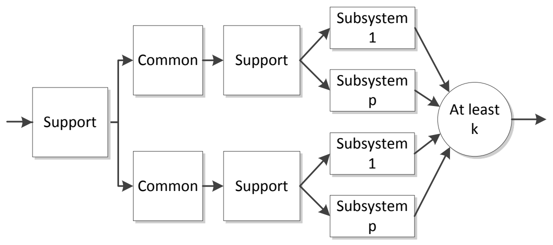


Fig. 2. An enhancement to the traditional k-out-of-n system architecture involves splitting the pool of components into a number of smaller pools, each with their own support and common components.

of a complex system such as a satellite. Instead, in the case complex systems, an analytical solution for reliability can be derived using knowledge about the individual component failure distributions and the system architecture.

The simplest system architecture is a series system, also known as a single-string architecture. In such a system, the failure of any component in the string causes system failure. To improve system reliability, without increasing component reliability, redundancy can be added. There are many ways in which redundancy can be added, but every form comes with overheads. For all satellite missions, but particularly for low-cost missions, it is vital to maximise reliability while minimising these overheads. Therefore, how redundancy is added is important.

Figure 1 shows reliability plots for five different system architectures, each containing a comparable amount of physical redundancy. In each case, the system is composed of 24 subsystems, with a level of redundancy equal to each subsystem being triplicated. Identical, continuous failure rates are assumed for all subsystems. The simplest forms of redundancy involve arranging the subsystems into three independent strings of 24 subsystems each, or cross strapping the subsystems across strings. Figure 1 shows that improved reliability can be achieved if each subsystem can be made multifunctional and placed into a global pool, which is known as an k-out-of-n architecture. In this case, as long as any 24 subsystems are still functional, the system remains functional. In practice, in addition to the pool of subsystems, any k-out-on-n architecture will require supporting and common components. Supporting components are interfaces between subsystems, such as communication and power buses. The reliability of supporting components decrease as more subsystems are added to the system. Common components represent infrastructure required to keep all subsystems operational, such as a power supply. The support and common components can present weak points in the system. Figure 2 depicts an enhancement to the traditional k-out-of-n architecture, where the subsystem pool is split into smaller sets, each with their own common and support components. The benefit, in terms of reliability, offered by such an architecture is evident in Figure 1, where the systems composed of multiple smaller pools of components show superior reliability during early life. A full reliability analysis and comparison to traditional architectures is given in [12].

III. Biological Inspiration

Biological life has achieved an impressive level of robustness, flourishing in even the harshest locations on Earth. Similarly, satellites are expected to display high robustness, having to operate in the extreme environment of outer space. In many ways, a satellite can be viewed as an artificial organism. Like its biological counterparts, a satellite requires a way of harvesting energy, hardware which continues to function without external maintenance, and the ability to make life-preserving decisions. In addition, just as biological organisms are under constant attack from environmental and biological hazards, so, too, a satellite must endure hazardous temperatures swings, a barrage of charged particles and micrometeorites. Therefore, it seems feasible that the reliability increasing techniques employed by biological organisms may be well suited for implementation on satellites, too.

The earliest forms of life were single celled organisms. Despite being comparable to single string systems, unicellular life has shown a remarkable level of robustness. This robustness is achieved through a variety of techniques, including genetic redundancy, physical adaption to the environment, and techniques for gene repair. While initially promising, these techniques turn out to be comparable to traditional reliability engineering techniques and therefore come with high implementation overheads. For example, genetic redundancy is comparable to discrete functional redundancy, physical adaptions to the environment are comparable to utilising specially radiation-hardened components, and techniques for gene repair are comparable to EDAC techniques commonly implemented on satellite memory.

In addition to these techniques, unicellular life relies on survival through sheer numbers. Interestingly, with the rise of small, low-cost satellites, this technique is becoming feasible for satellite missions, too. For example, the QB50 upper atmosphere studying mission [14] and the Planet Labs earth observation mission [15], are each composed of tens of

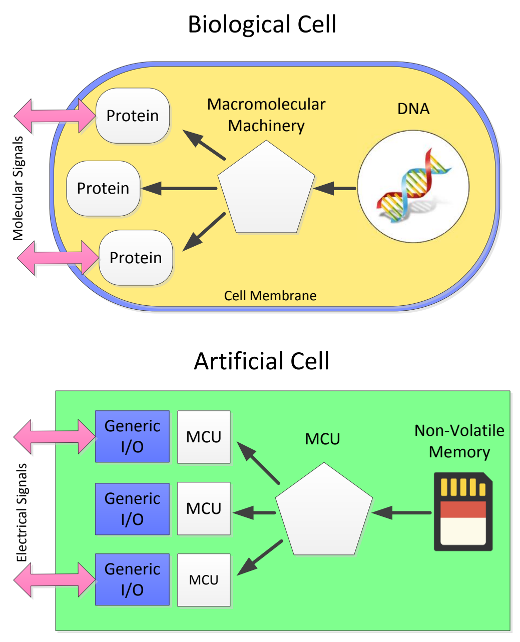


Fig. 3. A simplified depiction of the differentiation process in a biological cell and a proposed artificial implementation. Adapted from [13].

Cubesats. The loss of a few Cubesats would be acceptable, and is likely expected, for both missions. While conceptually interesting, it remains to be seen whether this strategy can be financially and operationally successful.

The discussion on redundancy in Section II highlighted the improved reliability offered by k-out-of-n architectures. Therefore, it is not surprising that a similar architecture emerged in nature in the form of multicellular organisms. Multicellular life has evolved from unicellular life multiple times during the Earth's past. In a simplified sense, multicellular organisms start out as a collection of identical cells, called stem cells. During the organism's development the stem cells adapt through a process known as differentiation to take on specific roles within the organism. In addition, some cells have the ability to continuously re-differentiate throughout their lifetimes. This change of specialisation, if possible, typically occurs in response to damage sustained by the organism. For example, a zebra fish can survive losing up to 20% of its heart, because fully differentiated cells in its heart will attempt to dedifferentiate before redifferentiating and proliferating to restore the missing tissue. Similarly, when the lens of a newt's eye sustains damage, nearby cells will redifferentiate to replace the damaged lens cells [16]. Thus, the cells in a multicellular organism are comparable to the multifunctional subsystems of a k-out-of-n architecture.

While both the analytical reliability analysis and investigation of biological life point towards k-out-of-n architectures for reliability, implementation of such an architecture faces three major engineering challenges.

Firstly, k-out-of-n architectures require highly multifunctional subsystems, which are traditionally complex to implement. Therefore, it is beneficial to understand how biological cells adapt to perform such a wide variety of tasks. Figure 3 shows a simplified, schematic representation of biological cell differentiation. Within biological cells, large molecules, known as proteins, are the workhorses. Proteins are essentially built-for-purpose pieces of machinery, meaning the set of proteins in a cell determine its capabilities. Proteins are manufactured inside cells from blueprints stored in the cell's DNA by a collection of large molecules, known collectively as macromolecular machinery (MM). The macromolecular machinery responds to internal and external conditions by reading different sets of blueprints and building the corresponding proteins. Therefore, the process of differentiation occurs when the macromolecular machinery of different cells respond to different local conditions by manufacturing different sets of proteins.

Learning from biology, the Satellite Stem Cell concept proposes building the subsystems of a k-out-of-n system out of artificial cells. Figure 3 depicts the differentiation process of an artificial cell. In this case, proteins are implemented using a set of discrete, reprogrammable processing elements, such as microcontrollers (MCUs) or field-programmable gate arrays (FPGAs). Each artificial protein is customised to perform a specific task by programming it with the appropriate firmware. A separate processing element acts as the macromolecular machinery. It responds to internal and external conditions, such as available power, by loading particular sets of firmware from non-volatile memory into the artificial proteins. The non-volatile memory, or DNA, on every cell is identical and contains programs for all system tasks, even though only a subset will be implemented on each cell at a time.

In the described artificial cell, the macromolecular machinery presents a potential single point of failure, while in a biological cell, the macromolecular machinery is highly redundant. To improve the reliability of the artificial macromolecular machinery, it is important to realise that the macromolecular machinery is, itself, largely composed of proteins. Therefore, it can be implemented in the same way as the artificial proteins. This results in the architecture shown in Figure 4. Every cell is composed of a set of artificial proteins, one of which is implementing the macromolecular machinery at all times. Thus, if the protein implementing the macromolecular machinery fails, another protein on the same cell can become the new macromolecular machinery. In this way, the potential single point of failure is eliminated.

The second engineering challenge involves interfacing to a variety of peripherals. While the described artificial protein design allows reprogramming to perform any required processing task, in practice, many tasks additionally require interfaces to the outside world. Peripherals such as reaction wheels and attitude sensors are not intrinsic to the cells and must therefore be interfaced to. Ideally, every protein should be interfaced to every peripheral, but this is impractical. Therefore, peripheral interfaces will place restrictions on cell differentiation. Such restrictions are not unfamiliar in biology, where localisation plays an important role in cell differentiation. For example,

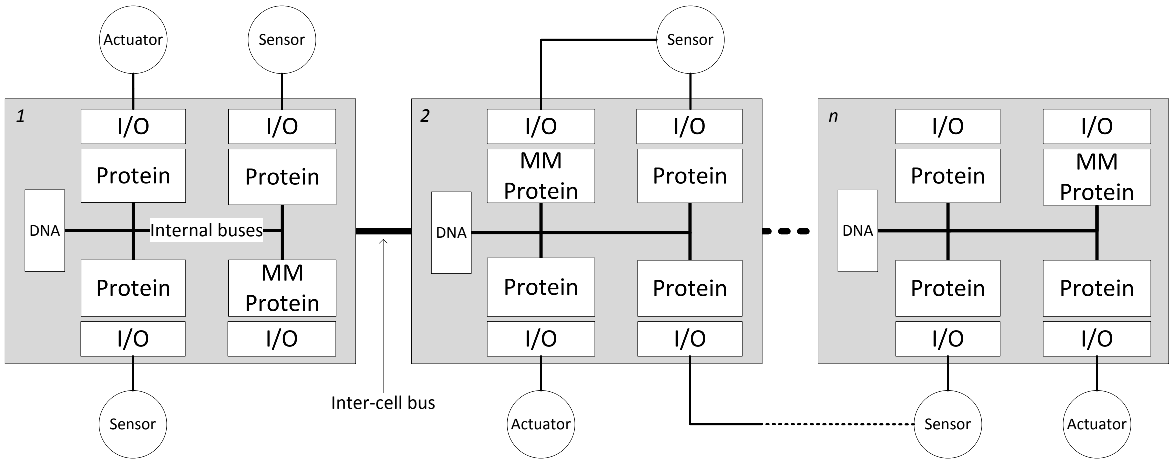


Fig. 4. A schematic representation of a system based on artificial cells. At any point in time, one protein per cell has the role of the cell's macromolecular machinery (MM). Peripherals interface to proteins via general purpose I/O circuitry and can be cross-strapped between proteins, or between cells.

there is little purpose in a heart cell redifferentiating into a retinal cell.

Any interface to the outside world additionally presents a potential hazard to the cell. In a biological cell, proteins send and receive chemical messages across a cell membrane, which protects the cell from the outside world. Therefore, a general purpose input/output (GPIO) interface is proposed which allows a protein to interface to a large variety of sensors and actuators, at high and low currents, while protecting the protein from unexpected behaviour from the attached peripheral. In addition, the GPIO interface allows peripherals to be cross-strapped between proteins. Implementation details of the GPIO circuitry are given in Section V.

Finally, there is the challenge of distributing tasks across the cells. In complex multicellular organisms the coordination of bodily functions is largely centralized through a central nervous system and brain. This is analogous to the design of most satellites in which the coordination of tasks is handled by a central onboard computer (OBC). However, the brain in an organism, and OBC in a satellite, present potential single points of failure. In contrast, simple multicellular organisms, like sea sponges, have no central nervous systems. In these organisms, all coordination is achieved through cellular, peer-to-peer communication.

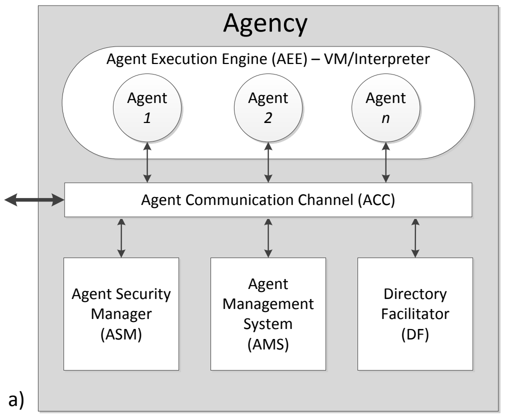
For reliability purposes, a distributed task management strategy, as seen in simple multicellular organisms, is preferred. In practical terms, this requires the implementation of middleware running on the artificial cells. A novel middleware, based on the active research area of Agent Computing, is described in the Section IV.

IV. Multi-agent System

Agent-based computing emerged in the 1990s as a powerful new programming paradigm [8]. The term 'agent' is loaded, having different definitions in different research areas. However, a broad description can be found in [17]: 'An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives'. Essentially, agents are convenient and powerful high level abstractions that allow complex pieces of hardware or software to be described in terms of behaviours. Agents work towards goals, have life cycles and may have a certain level of mobility. Similarly to proteins in a biological cell (or cells in a multicellular organism), agents in a multi-agent environment allow the decomposition of complex tasks into a set of simpler goals, which can be pursued by individual agents.

Agent technology provides a natural solution for achieving cooperation amongst the artificial proteins described in Section III. To date, many bespoke agent systems have been designed, with little or no compatibility between implementations. Therefore, to facilitate future interaction with other agent-based satellites and ground station services, the design of the agent middleware for the Satellite Stem Cell Architecture is based on the Foundation for Intelligent and Physical Agents (FIPA) Abstract Architecture and Agent Communication Language (ACL) specifications.

FIPA was formed in 1996 with the aim of standardising the interaction of heterogeneous agents and agent-based systems [18]. Figure 5a shows the architecture of a FIPA compliant software agent environment, typically called an agency. An agency is a piece of middleware which provides an execution environment and a set of services for agents. The execution environment allows agents to be dynamically loaded and executed. It is usually based on a virtual machine, such as the Java Virtual Machine (JVM), or a runtime interpreter. Services provided by the agency include communication channels, allowing agents to communicate with one another and other agencies, an Agent Management System (AMS), which controls the lifecycle of agents, and a Directory Facilitator (DF), allowing agents to discover services provided by other agents. In addition, since the agency and all local agents are typically running on the same physical hardware, an Agent Security Manager (ASM) ensures that agents cannot access


Agency
Agent Execution Engine (AEE) – VM/Interpreter
Agent 1
Agent 2
Agent n
Agent Communication Channel (ACC)
Agent Security Manager (ASM)
Agent Management System (AMS)
Directory Facilitator (DF)
a)

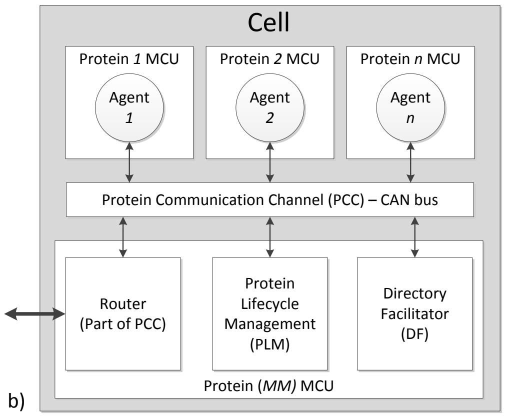

Cell
Protein 1 MCU
Protein 2 MCU
Protein n MCU
Agent 1
Agent 2
Agent n
Protein Communication Channel (PCC) – CAN bus
Router (Part of PCC)
Protein Lifecycle Management (PLM)
Directory Facilitator (DF)
Protein (MM) MCU
b)

Fig. 5. The architecture of a typical FIPA software agent environment a), and a distributed version optimised for the artificial cell hardware b).

restricted resources.

The Satellite Stem Cell hardware architecture focuses on reliability and low power consumption, making it poorly suited for porting existing agent environments. Instead, the FIPA Abstract Architecture is distributed, as shown in Figure 5b, to make best use of the fault tolerant characteristics of the hardware.

In the Satellite Stem Cell Architecture, each artificial cell is seen as an agency. Within each cell, agents are executed on the discrete MCUs of the artificial proteins. Thus, each set of protein hardware can be seen as a blank protein prototype, which becomes unique and active once loaded with an agent. Running agents on discrete MCUs allows for natural real-time operation. One protein on the cell is reserved for the task of providing agency services to the rest of the cell (previously referred to as the cell's macromolecular machinery). The unique design of the cell hardware ensures that the agency services are always provided, even as protein failures build up. The services include Protein Lifecycle Management (PLM), which is responsible for health monitoring and reprogramming proteins, and a Directory Facilitator. A physical communication bus links proteins within the same cell. This bus supports FIPA Agent Communication Language (ACL) communication between agents and agency services, as well as real time traffic. Inter-cell (inter-agency) communication occurs over a separate physical bus. Protein messages intended for recipients on other cells are automatically routed onto the inter-cell bus by a router service. Implementation details are given Section V.

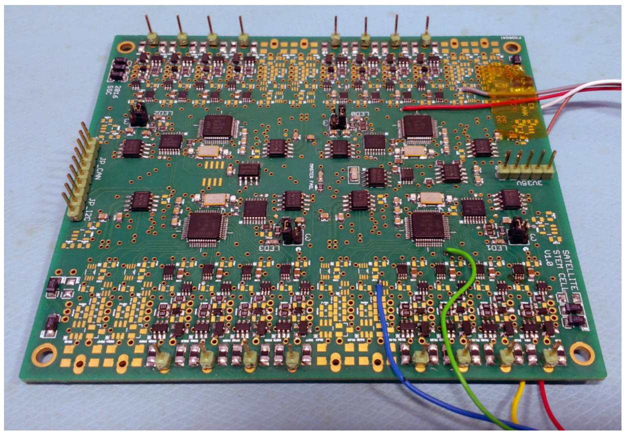


Fig. 6. Functional prototype cell with four proteins. Notice the GPIOs (partially populated) along the top and bottom edges of the PCB.

Advantages of the cell agent environment over traditional software agent environments include:

- Discrete hardware execution environments provide intrinsic real time agent execution and security
- Agents have direct access to hardware interfaces
- Agency services are ensured through redundant hardware

The main disadvantage of the cell agent environment is the lack of compatibility between agents developed for different implementations of the cell environment. Agents must be compiled for the specific MCU on which the artificial proteins are based. However, porting an agent between implementations is expected to simple, consisting mainly of adapting hardware abstraction layers for communication and GPIO control.

V. Implementation Details

This section provides preliminary implementation details of the cell hardware, GPIO circuitry, and agent middleware.

A. Cell Hardware

A full cell prototype is being developed to fully understand the implementation overheads of the Satellite Stem Cell Architecture. For comparison purposes, the prototype is being developed at a CubeSat scale, and will be compared in volume and power consumption to typical CubeSat avionics.

An operational cell prototype, containing four proteins, is shown in Figure 6, while Figure 7 gives implementation details on the cell design. Each protein is composed an ARM Cortex M0 MCU with internal CAN bus controller and transceiver, a secondary, external CAN bus controller and transceiver IC, GPIO circuitry, power switches, and a discrete I2C I/O node. Three communication buses span the cell: An internal CAN bus, an internal I2C bus, and an inter-cell (external) CAN bus. As described in Section III, one protein acts as the cell's macromolecular machinery (MM) and has certain control over the other proteins through the I2C bus. Using this bus to

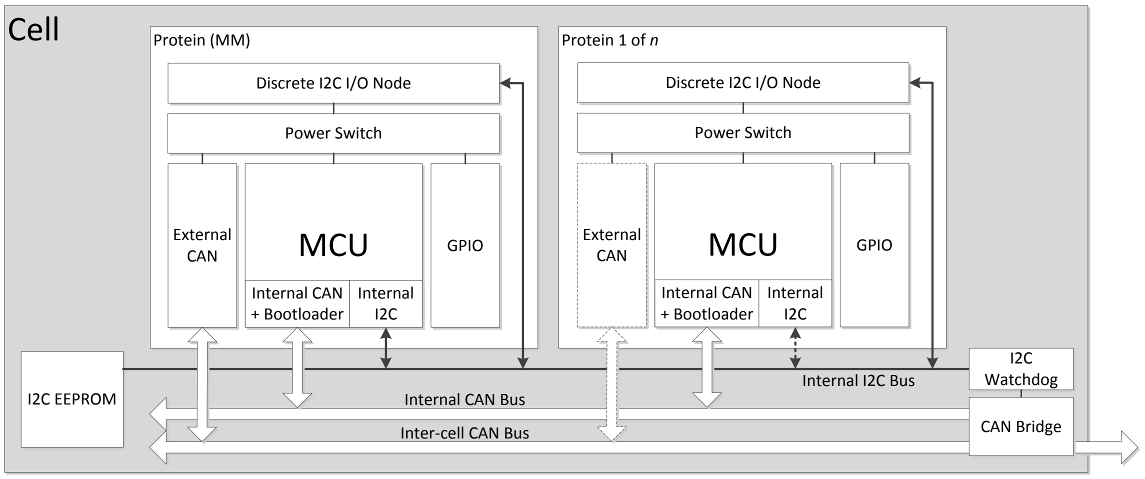


Fig. 7. Scematic representation of the artificial cell hardware design. Proteins are linked by internal Controller Area Network (CAN) and I2C buses. The protein acting as MM has a level of control over the other proteins through the I2C bus and discrete I2C nodes. Only the MM has access to the external CAN bus.

send commands to the discrete I2C nodes present in each protein, the MM can power cycle a protein, or put it into bootloader mode. Once in bootloader mode, a protein's MCU can be reprogrammed by the MM over the internal CAN bus, using firmware stored in central, non-volatile memory. Proteins communicate with one another and the MM protein using the internal CAN bus. Only the MM has access to the external CAN bus, onto which it forwards messages addressed to proteins on other cells.

The MM monitors the health of the proteins on its cell by polling them over the internal CAN bus and by polling the I2C nodes. If a failure is detected, the MM will attempt to activate a redundant protein and then recover the failed protein through power cycling or reprogramming. To guard against MM failures, every protein monitors for a loss of polling from the MM. In addition, the MM is required to continuously kick an I2C watchdog IC. In the event of an MM failure, one of the proteins will be first to notice the loss of polling. That protein will enter bootloader mode and wait. The other proteins will notice the loss of polling, too, but will detect that they were not first and proceed normally. Moments later, the I2C watchdog will timeout, activating a CAN bus bridge between the internal and external CAN buses. At this point, a second cell in the system will notice the presence of a bootloader-state protein on the external CAN bus, indicating the loss of an MM on another cell. The second cell will then proceed to reprogram the bootloader-state protein into a new cell MM.

The CAN bus bridge serves a secondary purpose. During normal operation, the MM acts as a router between the internal and external CAN buses. Therefore, MM failure cuts off communication between the proteins on the stricken cell and the rest of the system. The CAN bridge resolves this issue, allowing the proteins on the stricken cell to continue to operate while a new cell MM is being programmed.

The cell prototype measures approximately 100 x 110 mm, making it comparable to a single PC104 CubeSat subsystem. The MM protein, operating at 50 Mhz and utilising the external CAN bus, consumes 90mW, while the normal proteins, which can be operated at lower clock frequencies, consume as little as 15mW each. These values are in the range of typical CubeSat subsystems. A more accurate comparison to COTS CubeSat subsystems will be possible once the middleware has been completed and a representative set of tasks is loaded onto the system.

B. General Purpose I/O

Each protein of the cell prototype has six general-purpose I/O lines, which are visible along the edges of the PCB in Figure 6. The GPIO circuitry has multiple duties. Firstly, it must enable proteins to interface directly to a large variety of peripherals typically found on small satellites. Examples include magnetorquers, reaction wheels, and analogue and digital sensors. Secondly, the GPIO circuitry must allow peripherals to be cross strapped between proteins and should preferably fail in such a way as to not disable a cross-strapped peripheral. Finally, the GPIO circuitry must protect the cell from unexpected peripheral behaviour. To fulfill these duties, the GPIO circuitry was designed with the following specifications:

- Minimise physical changes required for different operation modes
- Operate at voltages from 3V3 to 9V
- Support digital I/O at baud rates of at least 100 kb/s
- Support analogue input voltages from 3V3 to 9V
- Provide drive and sink capability of at least 1A
- Provide current measurement when driving or sinking
- Support a high impedance state
- No single component failure should prevent entry into the high impedance state

High GPIO toggle rates are required to support digital communication protocols, such as Inter-IC (I2C), and for implementing pulse width modulation (PWM). A high current

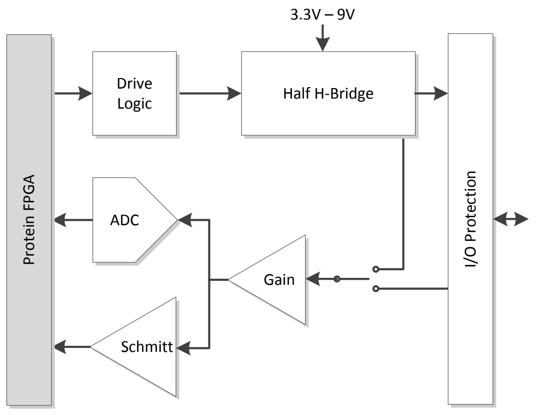


Fig. 8. Block diagram of the GPIO hardware design.

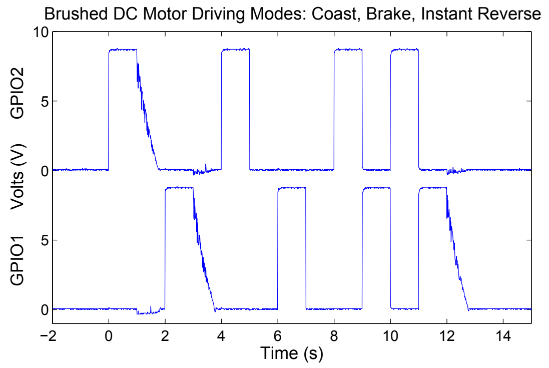


Fig. 9. Voltage traces demonstrating a pair of GPIOs operating as a full H-Bridge to drive a brushed-DC motor in coast and brake modes.

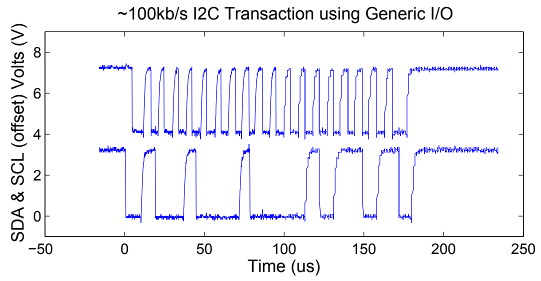


Fig. 10. A 100kb/s I2C transaction performed via the GPIO hardware.

drive and sink capability, together with current monitoring, are required to drive actuators. The high impedance state is used during input modes and when acting as a backup for a cross-strapped peripheral.

A key principle of the Satellite Stem Cell Architecture is that each cell is manufactured identically, leading to lower costs and simplified testing. Therefore, ideally, the GPIO circuitry should support all operational modes (eg. analogue input, high current output) without requiring any physical changes. However, since the operational mode of each GPIO will be determined by the attached peripheral, scenarios where on-orbit operational mode changes would be required are few. Therefore, small physical changes, such as installing jumpers or changing a minimal set of resistors during satellite development, are deemed acceptable.

A simplified representation of the GPIO circuitry is shown in Figure 8. It is comparable to the design of GPIO circuitry found in most MCUs, but with several enhancements. The output stage is based on a half-H-bridge, which is capable of driving up to one amp for directly interfacing to actuators (typical MCU GPIOs can deliver 5-20 mA). The input stage is based on an amplifier which feeds an analogue-to-digital converter and a Schmitt trigger. Unlike in most MCUs, the input stage can also be used to measure the current being delivered by the output stage.

GPIOs have been demonstrated directly interfacing to a brushed-DC motor (Figure 9), I2C devices such as gyroscopes (Figure 10), and analogue sensors.

C. Agent Middleware

The agent middleware is under development, but current implementation details are given.

The agent middleware operates on top of FreeRTOS, a free, real-time operating system which has been used on board several low-cost satellites [19], [20]. The middleware is composed of two parts. The first part consists of a set of agency services, such as the PLM, DF and Router, executed as FreeRTOS threads on the MM protein. The second part is a template on which agents are typically based. The template includes a set of FreeRTOS threads for handling ACL messages, and a set of libraries for interacting with the GPIO hardware, MM protein, and other proteins.

Agents interact with one another and the agency services over a physical CAN bus using the FIPA Agent Communication Language. Specifications for XML, string, and bit-efficient representations of ACL messages are provided by FIPA. The bit-efficient representation is chosen to limit processing overhead and power consumption. FIPA also provides specifications for two message transport protocols, namely HTTP and IIOP. However, neither of these protocols were designed to operate over a physical CAN bus. Therefore, an alternative message transport protocol, based on slightly modified ISO-TP, was developed.

ISO-TP, or ISO 15765-2, is an international standard for sending messages exceeding eight bytes over CAN networks. ISO-TP prepends one or more bytes of protocol control information (PCI) to each 8-byte CAN frame. The first PCI byte always indicates one of four frame types. A transaction begins with the transmitter sending a start frame, which includes a PCI byte describing the total message length, and the initial bytes of the message. A receiver will reply with a flow control frame, which describes how the transaction will continue. A flow control frame can request the transmitter to wait until further notice, or to proceed with sending the message at a defined rate.

A problem arises when considering the situation in which a single receiver is receiving ISO-TP messages from two or more transmitters simultaneously. All transmitters except one should be told to wait using flow control frames. However, the CAN bus protocol uses message IDs instead of sender and receiver addresses. A CAN network can be configured so that message IDs are treated as intended recipient addresses, however, the recipients have no information about message origins. Therefore, it is unclear where to send frame control messages. To solve this problem, the ISO-TP protocol is modified slightly by adding an additional PCI byte with the transmitter's address into each start frame.

The limited RAM available in the protein MCUs prompted placing some restrictions on the ACL implementation. Firstly, agent identifiers, which are encoded as strings, are limited to two characters each. Secondly, the message content of ACL messages is restricted to 11 bytes. Despite these restrictions and the use of the bit-efficient representation for ACL messages, significant overhead is incurred by following the FIPA specification. A FIPA ACL message, containing 11 bytes of message data, is typically 70 bytes long. Thus ACL messaging is only utilised for low data rate applications, such as system reconfiguration, health monitoring and task configuration. For all other communication, real time CAN messages can be passed throughout the system unrestricted. As an example, consider a producer-consumer situation between two proteins, which may or may not be on the same cell. ACL messages are used to initially tell the producer at what rate the consumer requires data, after which the data transfer proceeds using regular CAN packets.

The official port for the LPC11C24 MCU, based on FreeRTOS V7.1.0, was used, together with the LPCXpresso development environment. When compiled with size optimisation, the MM firmware is 28.7 kB, occupying 90 percent of the MCU's available flash memory. The protein template compiles to 24.9 kB, or 78 percent of the available flash. While the remaining space available for user protein code is limited, the template's extensive libraries should allow user code to focus on high level functionality. Nevertheless, MCUs with larger flash memories, or even softcore MCUs, should be considered for future cell implementations.

VI. Conclusion

This paper described the development and early implementation details of the Satellite Stem Cell Architecture, which aims to improve low-cost satellite reliability, while minimising overheads. The architecture is the result of contributions from three fields of research, namely, traditional reliability analysis, bio-inspired engineering, and agent-based computing. Novel, highly-fault tolerant, artificial cells form the core of the Satellite Stem Cell Architecture. Each cell is based on an enhanced k-out-of-n architecture and mimics the protein-driven reconfigurability of biological cells. To facilitate cooperation amongst the cells composing a complex system, a novel version of the FIPA Abstract Architecture for agent-based middleware has been developed. It makes maximum use of the fault-tolerant properties of the the artificial cell hardware. CubeSat-scale implementations of the cell hardware and agent-based middleware are well underway. Preliminary comparisons with traditional CubeSat avionics, in terms of volume and power consumption, suggest the practical feasibility of the architecture.

References

[1] J. Guo, L. Monas, and E. Gill, "Statistical analysis and modelling of small satellite reliability," *Acta Astronautica*, vol. 98, pp. 97 – 110, 2014.

[2] M. Swartwout, "The first one hundred cubesats: A statistical look," *Journal of Small Satellites*, vol. 2, no. 2, 2013.

[3] H. Pham, "Optimal design of k-out-of-n redundant systems," *Microelectronics Reliability*, vol. 32, no. 1, pp. 119 – 126, 1992.

[4] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti, "Toward self-repairing and self-replicating hardware: the embryonics approach," in *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on*, 2000, pp. 205–214.

[5] P. Bremner, Y. Liu, M. Samie, G. Dragffy, A. G. Pipe, G. Tempesti, J. Timmis, and A. M. Tyrrell, "Sabre: a bio-inspired fault-tolerant electronic architecture," *Bioinspiration and Biomimetics*, vol. 8, no. 1, p. 016003, 2013.

[6] M. R. Boesen and J. Madsen, "edna: A bio-inspired reconfigurable hardware cell architecture supporting self-organisation and self-healing," in *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, July 2009, pp. 147–154.

[7] J. Xu, Y. Dou, Q. Lv, and J. Zhang, "Etissue: A bio-inspired match-based reconfigurable hardware architecture supporting hierarchical self-healing and self-evolution," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, June 2011, pp. 311–318.

[8] B. Chen, H. H. Cheng, and J. Palen, "Mobile-c: A mobile agent platform for mobile c-c++ agents," *Softw. Pract. Exper.*, vol. 36, no. 15, pp. 1711–1733, Dec. 2006. [Online]. Available: http://dx.doi.org/10.1002/spe.v36:15

[9] D. E. Bernard, G. A. Dorais, C. Fry, E. B. Gamble, B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, P. P. Nayak, B. Pell, K. Rajan, N. Rouquette, B. Smith, and B. C. Williams, "Design of the remote agent experiment for spacecraft autonomy," in *Aerospace Conference, 1998 IEEE*, vol. 2, Mar 1998, pp. 259–281 vol.2.

[10] D. M. Surka, M. C. Brito, and C. G. Harvey, "The real-time objectagent software architecture for distributed satellite systems," in *Aerospace Conference, 2001, IEEE Proceedings.*, vol. 6, 2001, pp. 2731–2741 vol.6.

[11] C. P. Bridges and T. Vladimirova, "Agent computing applications in distributed satellite systems," in *2009 International Symposium on Autonomous Decentralized Systems*, March 2009, pp. 1–8.

[12] A. Erlank and C. Bridges, "Reliability analysis of multicellular system architectures for low-cost satellites," submitted for publication.

[13] A. O. Erlank and C. P. Bridges, "A multicellular architecture towards low-cost satellite reliability," in *Adaptive Hardware and Systems (AHS), 2015 NASA/ESA Conference on*, June 2015, pp. 1–8.

[14] J. Thoemel, F. Singarayar, T. Scholz, D. Masutti, P. Testani, C. Asma, R. Reinhard, and J. Muylaert, "Status of the qb50 cubesat constellation mission," in *65th International Astronautical Congress*, 2014.

[15] C. Boshuizen, J. Mason, P. Klupar, and S. Spanhake, "Results from the planet labs flock constellation," in *28th Annual AIAA/USU Conference on Small Satellites*, 2014.

[16] C. Jopling, S. Boue, and J. C. I. Belmonte, "Dedifferentiation, transdifferentiation and reprogramming: three routes to regeneration," *Nature Reviews Molecular Cell Biology*, vol. 12, no. 1, pp. 79–89, 2011.

[17] G. Weiss, Ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999.

[18] FIPA. Welcome to the foundation for intelligent physical agents. Accessed June. 09, 2016. [Online]. Available: www.fipa.org

[19] C. Bridges, S. Kenyon, P. Shaw, E. Simons, L. Visagie, T. Theodorou, B. Yeomans, J. Parsons, V. Lappas, C. Underwood *et al.*, "A baptism of fire: The strand-1 nanosatellite," in *AIAA/UTU Small Satellite Conference*, 2013.

[20] I. Sünter, "Software for the estcube-1 command and data handling system," Ph.D. dissertation, Tartu Ülikool, 2014.