A Distributed Computing Architecture for Small Satellite and Multi-Spacecraft Missions

Bryan Palmintier[*], Christopher Kitts[*], Pascal Stang[*] and Michael Swartwout[**]

[*] Robotic Systems Laboratory, Santa Clara University, Santa Clara CA 95053
(408) 554-4382, ckitts@me.scu.edu, bpalmintier@me.scu.edu, pstang@scudc.scu.edu

[**] Department of Mechanical Engineering, Washington University in St. Louis, St. Louis MO 63130
(314) 935-6077, mas@me.wustl.edu

Abstract. Distributed computing architectures offer numerous advantages in the development of complex devices and systems. This paper describes the design, implementation and testing of a distributed computing architecture for low-cost small satellite and multi-spacecraft missions. This system is composed of a network of PICmicro® microcontrollers linked together by an I^2C serial data communication bus. The system also supports sensor and component integration via Dallas 1-wire and RS232 standards. A configuration control processor serves as the external gateway for communication to the ground and other satellites in the network; this processor runs a multitasking real-time operating system and an advanced production rule system for on-board autonomy. The data handling system allows for direct command and data routing between distinct hardware components and software tasks. This capability naturally extends to distributed control between spacecraft subsystems, between constellation satellites, and between the space and ground segments. This paper describes the technical design of the aforementioned features. It also reviews the use of this system as part of the two-satellite Emerald and QUEST university small satellite missions.

Table of Contents

1. Introduction

Distributed computing architectures offer numerous advantages in the development of complex devices and systems. These advantages include well-defined interfaces, flexible composition, streamlined integration, straightforward function-structure mappings, standardized components, incremental testing, and other benefits.

In the context of this paper, *distributed computing* refers to computational decentralization across a number of processors which may be physically located in different components, subsystems, systems, or facilities. These processors may be general-purpose computers with data/application sharing capabilities (e.g. a typical personal computing network), they may have an architecture that enables collaborative processing focused on a specific task (e.g. parallel computation), and/or each may be optimized to efficiently execute particular tasks or control specific subsystems (e.g. smart peripherals).

The *architecture* of a computing system (whether centralized or distributed) refers to the physical and logical framework used to interconnect components. It determines the pathways for inter-subsystem data transfer and may have a large bearing on both wiring harness size and modularity. As depicted in Figure 1, typical architecture definitions include the following (in some cases, they are not mutually exclusive):

- A *star* (centralized or distributed) architecture consists of a central processing unit that is directly connected via dedicated links to every other computational unit; this approach often leads to large wiring harnesses and a dependency of the central computer's hardware/software on the design of the peripheral units. From a computational point of view, a star configuration is the prototypical example of a centralized architecture when the connected components don't have processors. A star configuration, however, can be used in a distributed framework if these components have processors.
- In a *ring* (distributed) computing architecture, processors are linked in a closed chain with

communication typically facilitated by a "token" which is passed from one processor to another; rings typically lead to small wiring harnesses, cause minor design dependencies among subsystem implementations, and may suffer from interrupted communication if/when subsystems are disconnected or fail.

- *Linear bus* distributed computing architectures typically consist of a standardized, shared, linear data bus to which all subsystems are connected; while this often leads to small wiring harnesses and non-problematic (dis)connections to/from the bus, it requires a well-designed communication protocol governing when processors can transmit data over the bus.
- *Hybrid* architectures occur when one or more instances of the aforementioned architectures are used to link different processors within a single system. This is often done given that different inter-processor communication requirements often are best addressed by different architectures.
- *Layered* architectures arise when more than one of the aforementioned architectures is used to link to the same processors within a single system. For example, the same processor may have a low data rate bus architecture for command and control information but a direct, dedicated, high data rate connection for science data. This is often done given that different processor functions often are best addressed by different architectures.

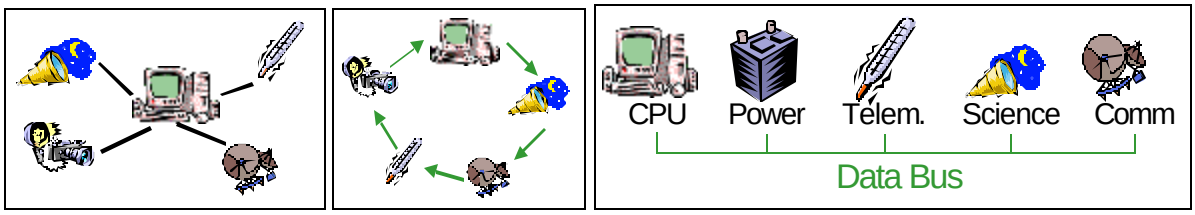
2. Advantages of a Linear Bus Architecture

The use of a linear bus distributed computing architecture leads to a simple and relatively small data bus that promotes standardized methodologies for interfacing at a range of levels. At the signal level, standardization of physical interconnections is a natural objective. At higher levels, standardization of data communication protocols for arbitration, error handling and other functions is a straightforward strategy. Ultimately, even the command and data handling interface can be standardized. For the current development effort, the ultimate goal of this standardization was to achieve component-level modularity such that components could be easily connected, disconnected, replaced, swapped and/or upgraded in a rapid and transparent manner.

While distributed computing systems and their advantages are common in modern personal computer, consumer product, industrial automation, automotive, and other industries, they have been slowly adopted in the satellite industry. Recent initiatives such as the NASA/JPL X2000 development program have recognized the advantages of distributed computing strategies and are beginning to develop such systems for space flight.

The authors (all of whom have served as systems engineers and/or managers of one or more university small satellite projects with centralized computing architectures) are pursuing the development and use of a linear bus command and data handling architecture for use in small and multi-spacecraft systems. The adoption of this strategy is in and of itself an experiment that explores the true benefits and costs of such an approach. The team currently believes that the following benefits will be realized:[1]

- In the design stage, the distributed system will simplify the command and data flow within the satellite by clarifying which specific component is responsible for each task and what information exchange is required to initiate the task; ideally, each functional block in the system's signal flow diagram will map directly to a physical box on the satellite. Furthermore, this characteristic will allow easy hierarchical scaling of the functional and data flow designs for multi-satellite missions and comprehensive space/ground segment systems.



(a) Centralized Architecture. (b) Ring Architecture (c) Linear Bus Architecture.

Figure 1 – Comparison of Typical Computing Architectures.

- During development and integration, the distributed architecture will promote the rigorous and independent test/verification and the controlled, incrementally integration of each subsystem/component. Such an achievement will assist in de-coupling the reliance of one subsystem's development on the operation of another (e.g. such as the central processing unit in most centralized architectures). Furthermore, with network bus "gateways", components can be easily integrated remotely using TCP/IP or other protocols to bridge and test subsystems being developed at different locations.
- On-orbit, the distributed architecture will simplify resource sharing (e.g. computational power, memory, etc.) among components, subsystems, and even satellites. It also promotes fault tolerance since computational functions can be supplied by other units (possibly even located in other spacecraft or on the ground) in the event of component outages.
- When exploited in a multi-satellite mission, the distributed architecture will allow components deployed across multiple satellites to interact in much the same manner as those within a single satellite. This has significant implications in the simplification of collaborative processing schemes both at the conceptual and implementation levels.

3. System Design

Given the aforementioned advantages of a distributed computing approach, the research team is focusing its efforts on the development of a robust and widely applicable linear bus design. Given this approach, the team sought to develop a system with a balance of simplicity and cost while also a) providing performance capable of supporting research-quality microsatellite instrumentation, and b) being feasible for use by student design teams.

The current design consists of a network of PICmicro® PIC-based processors linked by a hybrid bus consisting of a high-bandwidth Phillips I^2C (Inter-Integrated Circuit) data bus and a low-bandwidth Dallas Semiconductor "1-wire" microLAN for standard vehicle telemetry.

Subsystem Motherboard[2]

For the distributed architecture to work effectively, it is not necessary for all subsystems to use the same processor. As long as the standard bus interface is observed, processor differences are transparent to all other subsystems on the bus. But to simplify baseline subsystem development and to leverage economies of scale, a standard subsystem motherboard based around the PIC16F877 microcontroller was developed. This standard motherboard, shown in Figure 2, includes power/data connectors, power control circuitry, latch-up protection, and full access to the PIC's ports.

PIC16F877 Micro Controller

The PIC16F877 is a single chip computer, requiring only an external clock source. It resides in a 40-pin package with 8 kwords of ROM and 384 bytes of RAM. This particular PICmicro® has many built-in peripherals, including I^2C capabilities and an 8 channel A/D converter. Another attractive feature is its low power consumption: less than 100 mW at max speed (20Mz) and less than100μW in its low-power sleep mode. The PICmicro® also has a simple "RISC" instruction set, which allows fast and efficient execution.

One of the major shortcomings of the PICmicro® is its lack of support for external memory. The development team has worked around this deficiency by interfacing a simple custom SRAM capable of storing up to 2.5Mbytes of data for subsystems that require additional storage. Less memory intensive subsystems will store data in an external serial EEPROM.

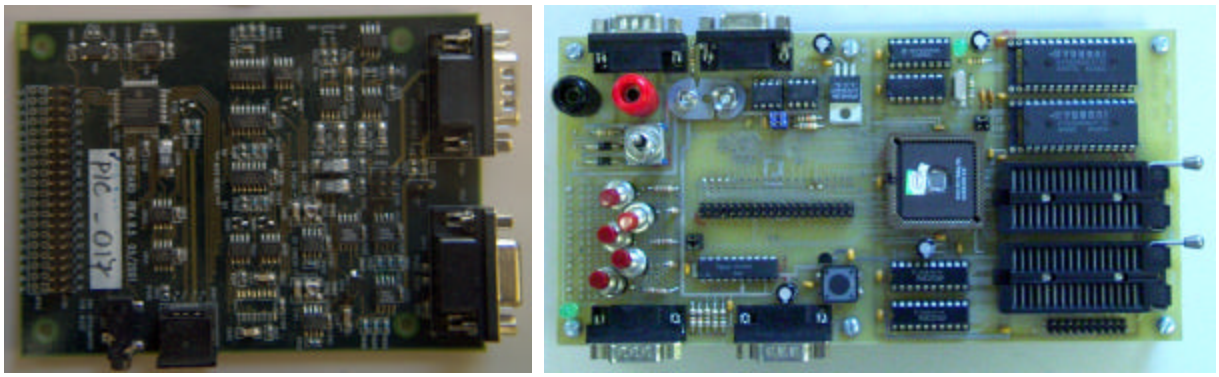
Software

A common library of low-level hardware routines, as well as standardized I^2C bus interfacing has been developed to simplify subsystem programming. The library includes support for:

- I^2C, including high level protocol
- RS232, for debugging
- A/D conversion at up to 18kHz
- Timer control to synchronizing fast events
- Real Time clock base for slower events, time-stamping, and scheduling.
- External memory support (1.5 MB SRAM or 64KB EEPROM)
- Simplified interface for translating I^2C commands into subsystem function calls
- Standardized, run-time configurable "super-loop" structure with flags to allow simple multi-tasking.

Configuration Control Computer

A separate Configuration Control Computer uses a 16MHz PIC17C56 (16Kx16bit ROM, 904Byte RAM, with built-in I^2C and 2 asynchronous serial ports) running the Pumpkin Salvo Real Time Operating System (RTOS).



(a) Standard PIC16F877-based Subsystem Board. (b) PIC17C56-based Configuration Control Board.

Figure 2 – Functional Motherboard Prototypes:

Originally envisioned to simply characterize data bus performance and to provide a redundant path between the communication system and the I^2C bus, the scope of this subsystem has grown due to difficulties with the originally selected commercial processor, the ability to leverage the capabilities of the Salvo RTOS, and the advantages of reusing software developed for the subsystem motherboards. A 17-series PIC was chosen as the heart of this system due to its expanded memory resources and RTOS compatibility.

On top of the RTOS, several software tasks execute in order to process I^2C and Dallas data packets, to service communication subsystem data flow, to efficiently enable software uploads, and to interface expert system execution with satellite commands and telemetry.

Data Bus

The data bus specification required careful consideration since it forms the data backbone of the entire satellite. There are numerous existing standards that span the high and low level aspects of distributed computing systems, especially in distributed industrial control applications.[3,4,5] However, these standards typically assume computational and power resources not available in small satellites. Therefore, the team researched simpler communication protocols, those typically used at the chip and board scale (instead of the room or facility scale). This allowed a design that met power, voltage, and computational constraints but required the custom development of the high level protocols.

Protocol Selection[2]

The team decided to only consider synchronous serial protocols given the desired balance of simplicity, reduced wiring, and speed. Additionally, protocol support for multiple nodes sharing control of the bus (multi-master) was desired, though not strictly required, for some experiments. Given these parameters, detailed consideration was given to three simple protocols commonly used in embedded systems:

- *Controller Area Network (CAN):* CAN is used extensively in automotive and industrial control applications. However, CAN is most suited to industrial-scale users, and is therefore overly complex for the current effort. The increasing commercial availability of stand-alone CAN chips and microcontrollers with integrated CAN support may make CAN more attractive in the future.
- *Serial Peripheral Interface (SPI):* SPI is simple and widely used, but the interface doesn't scale well to arbitrarily sized networks since it requires additional wires for address lines. In addition, multi-master support requires extending the baseline specification.
- *Inter-Integrated Circuit (I^2C):* I^2C is used in audio/visual equipment, on PC motherboards, and in "smart" batteries. This protocol easily scales to networks of arbitrary size and includes built-in support for multiple masters. However, it doesn't provide existing, suitable high-level communication protocols.

Given these options, the development team chose to use the I^2C bus. Conveniently, there are many simple, off the shelf devices available for I^2C, (digital I/O, A/D converters, EEPROM, microprocessors, etc.)

Low level I^2C

I^2C is a synchronous serial protocol that uses only 2 wires, one for data (SDA) and one for clock (SCL). It also requires a common ground. It operates at 100 kbps

in standard mode. Faster modes (400kbps and 3.4 Mbps) are also specified, but fewer devices support these modes. The protocol is specified to a fairly high level from reading and writing to multi-master support and arbitration.

Both lines are connected wired-AND, which allows for arbitrary numbers[†] of devices and which facilitates "hot swap" addition and removel of devices without interrupting the bus.

Communication is always initiated by a master, which also drives the clock. Each I²C message consists of an arbitrary number of 9 bit "words." These words are 8 bits of information (supplied by the current "transmitter") plus an acknowledge (from the "receiver). The first word of the message sets the address (7bits) and the communication direction (Read or Write). In read mode, after the first acknowledge, the slave begins transmitting and the master becomes the "receiver."[6,7]

High Level Messaging

I²C standardizes many layers of the communication protocol. However, because it does not specify any data integrity checks, the development team has developed a simple, error checking message format. Command packets coming from the Configuration Control processor (or any other master) are fixed to 5 bytes in length, plus a field for a return address and checksum (rolling 8bit sum). Reply packets include a field for length plus a variably sized data portion and finally a checksum byte for error detection.

A simple acknowledgement system was also developed. From the master's perspective, a complete communication includes sending a command packet, waiting for an acknowledgement byte, and then, if indicated, requesting a data packet in reply, and finally acknowledging that the data packet was successfully received. The complete message formats is shown in Figure 3.

This approach permits subsystem designers to choose the exact format for commands and data relevant to subsystem tasks. In addition, a variety of standard commands are defined for controlling common tasks for all subsystems on the bus. These include functions for checking subsystem status, synchronizing time, and querying the subsystems for a list of defined commands (help function). This last feature is very attractive because it allows subsystems to change and expand their functionality without requiring extensive system knowledge by operations personnel. Due to time limitations, these standard commands will only be incorporated into some of the nodes on the system.

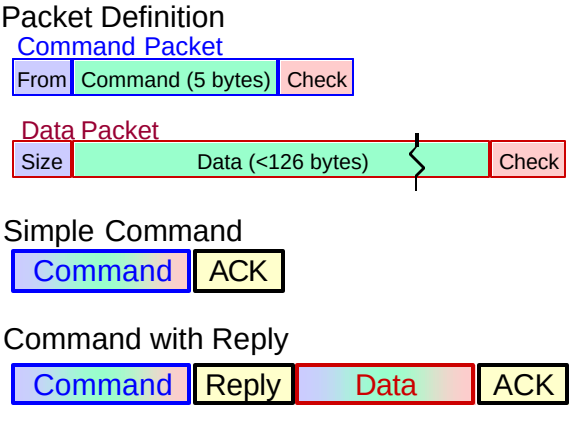


Figure 3: I²C Message Format

Telemetry Bus

In addition to the higher bandwidth I²C command/data bus described above, the current architecture also has a low bandwidth telemetry bus built on the Dallas 1-wire protocol. This multi-layer configuration is similar to the configuration chosen by JPL for the X2000 data bus. X2000 uses IEEE 1394 (Firewire) instead of I²C for data transfer and I²C instead of Dallas 1-wire for telemetry and control.[8]

Dallas microLAN supports an arbitrary number of devices connected with a single bi-directional data line. It is an asynchronous protocol that operates at 14.4kbps. Like many standardized serial protocols, a wide range of off-the-shelf components is available. Of particular interest for satellite telemetry are the temperature sensors (DS18B20) and analog-to-digital converters (DS2450, quad channel 8-bit accurate).

One particularly appealing characteristic of the Dallas technology is that each individual device has a completely unique 64-bit ID number for addressing. This eliminates the need for external address configuration pins, or chip select lines when connecting multiple devices to the bus. Adding another temperature node, or A2D node truly is as easy as just connecting 3 wires: power, ground, and the data line.

There are some 25 temperature sensors and nearly twice that many A2D converters on each Emerald satellite. Many of the A2D converters are actually not used for

[†] The number of devices is actually limited by the electronic signalling requirements of the I2C bus, such as maximum capacitance and resistance, but this is not a problem for even the maximum number of nodes ever considered, which is approximately 20.

analog input, but as 4 channel open-drain digital outputs for various control and switching applications on the satellite. In fact an A2D converter turned digital output, is used with a P-channel MOSFET as part of the standard power control circuitry in all of the subsystems.

Design-level Fault Tolerance

Since the nodes of the I²C bus are connected as open-collector, it is possible for an errant subsystem to hang the entire data bus. In other aerospace systems (airplanes, larger satellites, etc.), a common solution is to incorporate the additional complexity of multiple redundant back-up data buses.

For this design, the team has taken a less robust but appropriate approach given that tight power/mass/volume budgets and a desire for simplicity precluded such redundancy. This is done by controlling subsystem power over the Dallas microLAN, thereby allowing a component to be reset or completely shut down in the event of a component failure that results in an I²C bus lock-up. An analog switch at each I²C connection ensures that the subsystem circuitry is isolated from the I²C bus. Therefore, if a controller detects that the I²C bus is hung, it can selectively turn off power (using Dallas) to successive subsystems until the fault is eliminated.

While Dallas devices are not individually isolated, these devices receive power from a separately switched supply. This allows power to the Dallas bus to be cycled, hopefully clearing up any problems that have arisen. Latching circuitry in the standard subsystem power control module prevents subsystem power loss during this power cycling.

To date, this approach to bus-level fault tolerance has proved sufficient. This level of capability will be more fully tested in the future.

Comparison with Other Standards

The performance of the architecture's data bus compares favorably to other standards commonly used in aerospace avionics.[9] It is interesting to note that this was achieved without any initial knowledge of these other standards. In particular, the control architecture is nearly identical to that used in military aircraft (MIL-STD 1553), while the data rate is as high as found in most commercial airplanes (ARINC-429). Most importantly, the selected design uses 2 orders of magnitude less power per node than these standards. This reduction in power was crucial to the applicability of this system to small spacecraft mission. Table 1 provides a comparison of these and other relevant performance metrics.

4. Application to Small Satellite Missions

The development team intends to use the distributed computing architecture described in this paper in a variety of robotic systems ranging from undersea robots to land rovers to airships to spacecraft. With respect to the latter, the architecture is currently being incorporated into two multi-satellite small spacecraft missions: Emerald and QUEST.

	I²C as being used	I²C spec[9]	ARINC 429[10]	ARINC 629[11]	MIL-STD 1553[12]
Topography	Linear Bus	Linear Bus	Linear Bus	Linear Bus	Linear Bus
Control	Selectable master	Any master	Single transmit.	Any transmit.	BC + BM
Data Rate	100kbps	100k/400k/3.4Mbps	12.5/100kbps	2 Mbps	1Mbps
Max Num Nodes	~100	>>20	20	-	31
Coupling	Direct / Open Collector	Open Collector	Direct	Drct./Induct./Fbr.	Inductive
Redundancy	None	-	Varies	Varies	varies
Max Msg. Length	5/127 Bytes	-	2Bytes/34KB	-	64 Bytes
Data Integrity	8bit checksum	None			
Voltage	0/5V	0/5V	-10/0/+10V	-10/0/+10V	
Fault Tolerance	BM/D1W Pwr. Cntrl	-	Varies	Varies	BM/redundancy
Power Consumption	3.5mW	<50mW	Watts??	Watts??	1-2W[13]

Table 1: Comparison of Implemented I²C Data Bus to other bus standards comonly used in aerospace systems

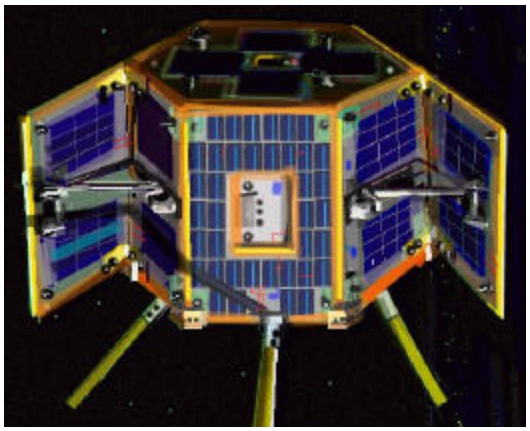


Figure 4 – An Emerald Spacecraft with Extended Drag Panels [Henning]

The Emerald Nanosatellites

Emerald is a two-spacecraft small satellite mission being jointly developed by Santa Clara University and Stanford University.[14] As a mission to explore the sensing, actuation, and control issues relating to distributed space systems, Emerald will attempt to characterize the performance of space-based GPS receivers, low-cost drag panel actuators, and a distributed formation control strategy. In addition, technology development is being driven by the needs of a simple but legitimate multi-satellite science experiment involving the distributed monitoring of lightning-induced VLF radio waves.

Depicted in Figure 4, each Emerald satellite consists of a 15 kilogram, 14-inch tall, 16-inch diameter hexagonal structures employing modular, stackable trays made of aluminum honeycomb. Custom, space-quality linear actuators articulate two drag panels mounted on he sides of the spacecraft.

Sun sensors and a magnetometer provide attitude information, and three torque coils provide control torques. The power system consists of 24% efficient, triple junction Gallium-Arsenide solar cells, a multi-cell NiCad battery, and high quality voltage regulators. Amateur radio communications provide 9600 baud communications in the 145 and 435 MHz bands; both satellite-to-ground and intersatellite communications are supported. Passive thermal control techniques are being emp loyed.

The distributed control system meets all requirements of the Emerald mission with the exception of the GPS relative position sensing system. The needs of this particular system include access to a high data rate communication link. While the I^2C bus could actually handle this data capacity, the team decided to use an existing, direct serial port connection to the flight processor so that the GPS unit could direct access the communication link without taxking the resources of the data bus; this was done in order to lower congestion on the I^2C bus given that the serial port was available for use at no additional developmenet cost. Figure 5 shows how the I^2C bus and Dallas bus are used to connect components within the Emerald system.

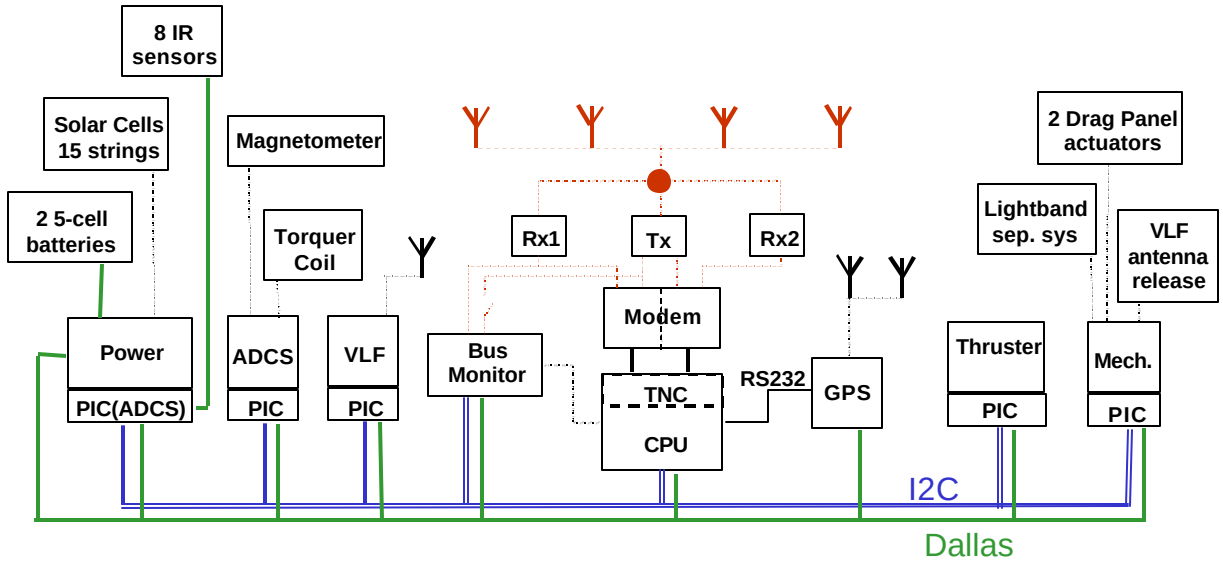


Figure 5 – The Emerald System Block Diagram Showing Connectivity of the I^2C and Dallas data buses.

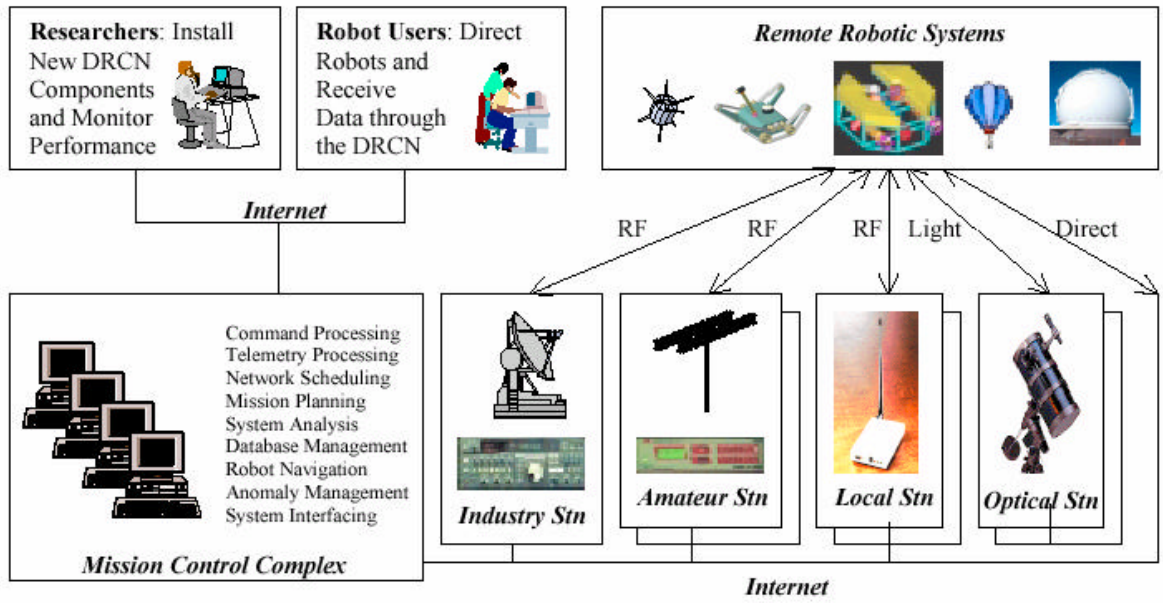


Figure 6 – The Distributed Control Network

Use of the distributed computing system is being orchestrated as a flight experiment within the Emerald operations plan. First, performance statistics regarding data packet transmissions, collisions, rates, and errors will be collected. Second, the system will host an on-board expert system capable of directly accessing the bus in order to issue subsystem commands in response to autonomy modes and realtime telemetry; this system will support a variety of functions ranging from task planning and execution to anomaly management. Third, the architecture will be used in a similar manner but with a scope that spans the multi-satellite formation. This will allow the development team to showcase processor resource sharing, fleet-level commanding with on-orbit task planning and execution, and coordinated opportunistic science.[15]

Finally, the system will be used within a comprehensive ground control network consisting of a centralized mission control center and several geographically distributed communication stations.[16] Depicted in Figure 6, this network currently includes operational stations in California and Hawaii with additional stations being planned for Missouri, Oklahoma, Texas and Alaska prior to Emerald's launch. The network's mission control center will use professional-grade command and telemetry software along with a variety of research-oriented analysis packages dedicated to resource allocation, anomaly managemenet, science data processing, and other functions.

The Quest Tethered Microsatellite

As an active program being conducted under the auspices of the Japan – U.S. Science, Technology, and Space Applications Program (JUSTSAP), QUEST is a joint spacecraft mission among four universities, two in Japan: Kyushu University (KU) & the University of Tokyo (UT), and two in the U.S.: Santa Clara University (SCU) & Washington University in St. Louis (WU). The primary QUEST mission is to demonstrate deployment and survivability of a 2 km space tether, marking the first use of a tether by a Japanese university and the first controlled tether on a very small spacecraft.[17] Additional missions include the provision of amateur radio store-and-forward services, demonstration of autonomous operations capabilities ranging from model-based anomaly management to collaborative multi-node mission operations, and several student-centered experiments.

The mission architecture consists of two similarly-sized spacecraft, one on each end of the deployed tether. The total launch mass of the mission is 50 kg and the combined QUEST vehicles will be less than 50 cm on a side. This mission intends to fly into low-Earth orbit as a secondary payload on the Japanese HII-A rocket in 2004 or 2005. KU will provide the tether & the launch interface, UT the communications subsystem, SCU the "daughter" spacecraft bus and the distributed ground station control network, and WU the "mother"

spacecraft, thermal control and pointing control subsystems. Figure 7 shows the prototype tether release mechanism, which has undergone extensive testing. Figure 8 depicts the tether deployment timeline.

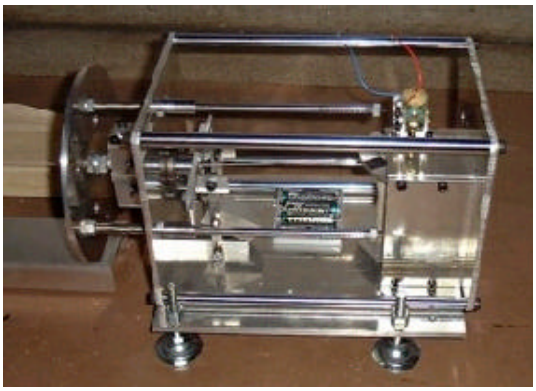


Figure 7 – Prototype of the QUEST Tether Release Mechanism

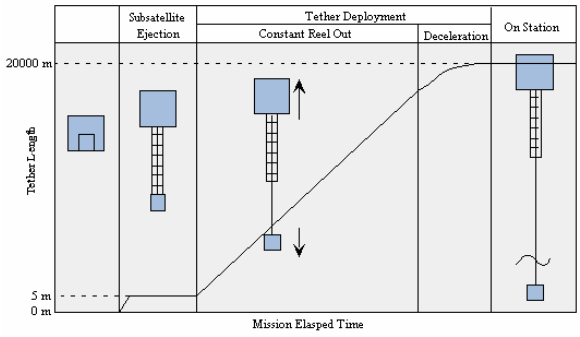


Figure 8 – QUEST Tether Deployment Timeline

The distributed computing system described in this paper is well suited to provide the computational infrastructure for the QUEST mission. It meets the majority of communication bandwidth and performance requirements, and the PIC-based motherboards are capable of controlling the majority of subsystem elements. In addition, it will naturally address the intricacies of multi-satellite control issues in a manner similar to the Emerald mission.

Depending on the evolution of the mission, one or two of the payload systems may require processing and/or communication capabilities beyond that baselined in the proposed architecture. Advanced processors will be accommodated by simply requiring them to adhere to the standard I^2C data protocol. A high-bandwidth communication requirement, which may arise in order to support a real-time video experiment, will be addressed by adding a dedicated communication transmitter directly connected to the camera payload.

5. The Architecture as an Experiment

The design team is treating the development of the distributed computing architecture described in this paper as an experiment to identify the true pros/cons of this approach and to gain insight into how its use can be effectively exploited by system design teams. Although early observations are anecdotal, several qualitative observations have been made to date.

First, the use of standard protocols and equipment has saved a significant amount of cost and time that would have otherwise been invested in a custom design activity (as the authors had done in previous programs).

Second, the modularity inherent in this approach has allowed entire subsystems to be completed, iteratively tested, and seamlessly integrated with other subsystems without being impeded by slow progress elsewhere within the program.[18] For example, the Emerald program originally baselined a commercially available flight computer; however, significant delays occurred due to the delivery schedule and the learning curve associated with this equipment. While such delay would have crippled a central computing architecture, the Emerald team was able to completely develop and accept several subsystems prior to the delivery of the final version of the commercial processor.

Third, the well-defined interface standard and the ability to test compliance and functionality of subsystems via a simple PC interface, as depicted in Figure 9, has had an interesting affect on the student design team. In previous projects, functional integration and test was often a somewhat mysterious exercise conducted by a core set of software developers responsible for implementing data interfaces ranging from debug signals all the way through to the final command and telemetry interface. The well-defined PC interface has made this process far more accessible to the broad student design team, thereby providing a significantly enhanced educational experience.

In addition, the PC interface provides a natural gateway for an internet-based bridge between data buses at geographically distributed locations as is depicted in Figure 10. Such a connection, using a ring-buffered network data server or a direct on-board ethernet, has been used at Santa Clara University for remote development and test of other robotic systems using alternate distributed computing architectures; use of a similar framework for the Emerald and Quest projects is being considered for the future.

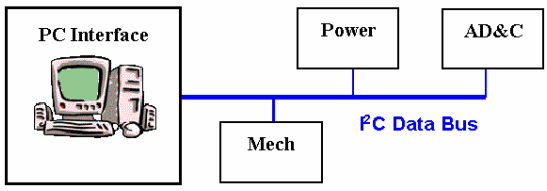


Figure 9 - The PC-based Integration and Test Setup

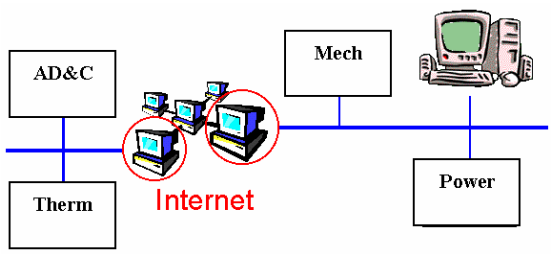


Figure 10 - An Architecture for Geographically Dispersed Test and Integration

6. Future Work

Future work remains at many levels. In the current phase of work, the development team must complete a final version of the data handling protocols and libraries suitable for flight. In addition, work remains on services for software uploads, the production rule system, and quantitative testing for data rate performance and fault tolerance. Furthermore, the team will continue integrating this system with the Emerald and Quest spacecraft and will identify new opportunities to incorporate this technology on other small spacecraft and robotic systems; the "off-the-shelf" nature of the system will hopefully make this a cost-effective choice for future design teams.

On a broader level, the team wishes to continue with research and development of distributed computing systems. Future designs might also migrate to a faster or more capable data bus standard such as CAN or possibly even internet-based protocols (as is being done on the distributed computing architecture for the team's current efforts in undersea robots). Another direction for improvement would be to develop a truly redundant bus based around a low power standard such as I^2C.

In addition to improving the raw performance of such systems, the team is also interested in developing and extending suites of autonomous services that can exploit such capability; this includes services such as dynamic command and data routing, resource sharing, and collaborative multi-robot operations.

7. Conclusions

The development team has made significant progress in developing a linear bus distributed computing architecture. This architecture balances simplicity and cost with the performance required to support research-quality, university-based microsatellite instrumentation and missions. This system consists of a network of PICmicro® PIC-based processors linked by a hybrid bus consisting of a high-bandwidth I^2C data bus and a low-bandwidth Dallas "1-wire" microLAN for standard vehicle telemetry. The current system is being incorporated into both the Emerald and the Quest multi-satellite small spacecraft missions.

To date, the system has been implemented and functionally verified with a variety of small satellite subsystems. Compared to previous experiences with developing small spacecraft with a centralized computing architecture, the team has observed significant improvements in the development process. These are largely attributed to the standardization and modularity that has been enforced by the distributed architecture. Observed improvements to date include reduced costs, a framework for independent development and seamless integration, a more open and accessible iuntegration and verification process, and a clear strategy for extending the distributed architecture's benefits to multi-satellite flight operations as well as to geographically distributed development operations.

8. Acknowledgements

Development of this distributed computing architecture has been made possible through sponsorship by the U.S. Air Force, NASA Goddard Space Flight Center, the Universities Space Research Association, the California Space Grant Consortium, and Santa Clara University.

Integration of this system with the distributed ground-based mission control architecture as well as work involving the internet-based network bus is based upon work supported by the National Science Foundation under Grant No. EIA0079815. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

The authors also wish to thank the student design teams at Santa Clara University, Stanford University, and Washington University in St. Louis for their feedback in the development of this system and its incorporation into several active flight projects.

References

1. Palmintier, B., "Emerald Nanosatellite Data Architecture," Stanford University AA254 Research Report, May, 2001.

2. Palmintier, et. al., "Distributed Computing on Emerald: A Modular Approach for Robust Distributed Space Systems," Proceedings of the 2000 IEEE Aerospace Conference, Snowmass MT, March 2000.

3. Fieldbus Homepage, Dec 1998 [Available Online at http://cran.esstin.u-nancy.fr/CRAN/Cran/ESSTIN/Fieldbus/page_norm.html#Fieldbus]

4. Synergetic Microsystems, Inc., Online Fieldbus Comparison Chart, December 1999 [Available Online: http://www.synergetic.com/compare.htm]

5. Wade D. Patterson, VMEbus Frequently Asked Questions, December 1999 [Available Online: http://www.vita.com/vmefaq/index.html]

6. The I²C-Bus Specification, version 2.0, December 1998 [Available Online: http://www-us.semiconductors.philips.com/acrobat/various/I2C_BUS_SPECIFICATION_2.pdf]

7. Paret, D., and Fenger, C., The I²C bus: from theory to practice, Wiley, Chichester ; New York, 1997.

8. Paschalidis, N., "A Remote I/O (RIO) Smart Sensor Analog-Digital Chip for Next Generation Spacecraft" In Proceedings of the 12th AIAA/USU Conference on Small Satellites, 1998.

9. The I²C-bus and how to use it (including specifications), Phillips Semiconductor, June 2001 [Available on-line: http://www.semiconductors.philips.com/acrobat/various/I2C_BUS_SPECIFICATION_1995.pdf]

10. ARINC Protocol Tutorial, Condor Engineering [cd-rom]

11. Spitzer, C., Digital Avionics Systems: Principles and Practice, second ed. The Blackburn Press, Caldwell, NJ, 2000, pp. 37-43.

12. MIL-STD-1553 Tutorial, Condor Engineering [cd-rom]

13. UT63M1XX: Power Consumption vs. Dissipation [Availible on-line: http://www.utmc.com/products/consumption.pdf, 6-7-2001]

14. Kitts, C., et. al., "Emerald: An Experimental Mission in Robust Distributed Space Systems," Proceedings of the 13th Annual AIAA/USU Conference on Small Satellites, Logan UT, August 1999.

15. Kitts, C., and Swartwout, M., "Autonomous Operation Experiments for the Emerald Nanosatellite Program," Proceedings of the 14th Annual AIAA/USU Conference on Small Satellites, Logan UT, August 2000.

16. Kitts, C., "The Distributed Robotic Control Network," Santa Clara Technical Document, Robotic Systems Laboratory, 2001.

17. Carlson, J., and Nakamura, Y., "The Kyushu/US Experimental Satellite Tether (QUEST) Mission, a Small Satellite to Test and Validate Spacecraft Tether Deployment and Operation," Proceedings of the 14th AIAA/USU Conference on Small Satellites, Logan UT, August 2000.

18. Townsend, J., et. al., "Effects of a Distributed Computing Architecture on the Emerald Nanosatellite Development Process," In Proceedings of the 14th AIAA/USU Conference on Small Satellites, Logan UT, August 2000.