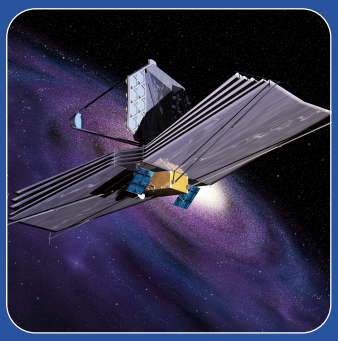
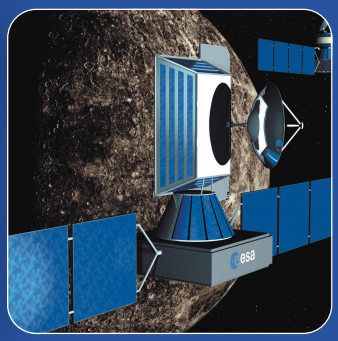
International SpaceWire Conference

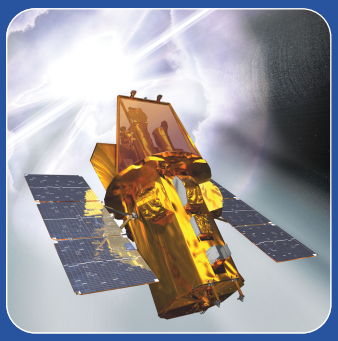
Dundee 2007



Monday 17th to Wednesday 19th September

Conference Proceedings





Space Technology Centre
University of Dundee

Sponsored by

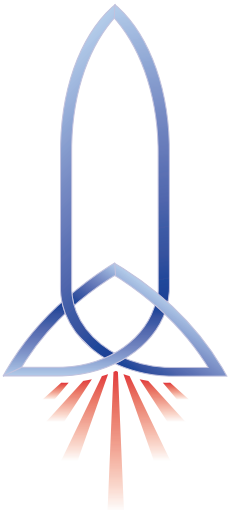
University of Dundee · esa · Dundee City Council Economic Development · IEEE · BNSC British National Space Centre · JAXA · NASA · Discover the Secret Dundee & Angus Convention Bureau

International SpaceWire Conference

Dundee 2007

Monday 17th to Wednesday 19th September

Conference Proceedings



Space
Technology
Centre
University of Dundee

International SpaceWire Conference
Dundee 2007
Conference Proceedings

ISBN: 978-0-9557196-0-8

Space
Technology
Centre
University of Dundee

Contents

Papers Indexed by Session

Monday 17th September

Mon 13:45-15:05 - Missions & Applications 1 (80 mins)

Mon 15:50-17:05 - Onboard Equipment & Software (75 mins)

Tuesday 18th September

Tues 9:00-10:30 – Components 1 (90 mins)

Tues 11:15-12:35 - Standardisation (80 mins)

Tues 13:35-16:00 - Networks & Protocols 1 (145 mins)

Tues 16:45-18:25 - Networks & Protocols 2 (100 mins)

Wednesday 19th September

Weds 9:00-10:30 - Test & Verification 1 (90 mins)

Weds 11:15-12:45 - Test & Verification 2 (90 mins)

Weds 13:45-14:15 - Missions & Applications 2 (90 mins)

Weds 14:40-18:35 - Components 2 (165 mins including 25 min break)

Additional Papers

* Abstracts only available for these papers

Papers Indexed by Author

First Author's Surname: A-D

First Author's Surname E-K

First Author's Surname: L-R

First Author's Surname S-Y

Missions & Applications 1

Monday 17th September

13:45 – 15:05

The Geostationary Operational Satellite R Series (GOES-R) SpaceWire Implementation

Session: SpaceWire Missions and Applications

William H. Anderson – NASA Goddard Space Flight Center/MEI Technologies

E-mail: William.H.Anderson@nasa.gov

ABSTRACT

The GOES-R program needed a simple high speed data interface for on-board communications. SpaceWire was chosen as the best solution to this need. The British Aerospace (BAE) SpaceWire Application Specific Integrated Circuit (ASIC) was developed under a NASA contract and selected by GOES-R as the model solution for instrument-to-spacecraft communications. The GOES-R project has developed ground support hardware, software, and a Reliable Data Delivery Protocol (RDDP) to meet mission needs. This paper presents GOES-R SpaceWire hardware and software development activities. Also discussed is the SpaceWire implementation and use on the spacecraft.

The GOES-R SpaceWire test card was designed to use the BAE ASIC providing a platform validating this approach to satisfying GOES-R requirements. The SpaceWire test card is Peripheral Component Interconnect (PCI) compliant and operates in a Windows work station and environment. There is on-board memory and all features of the ASIC are available to the user. Primary applications of the test card are spacecraft data system test and evaluation as well as ground support equipment. This test card has been used to prove the concept and functionality of the GOES-R flight data system. Also, the card has been used to develop a Reliable Data Delivery Protocol (RDDP) and verify GOES-R protoflight instrument interfaces.

The GOES-R spacecraft implements a point-to-point instrument-to-spacecraft interface. The RDDP is wrapped around CCSDS Source Packets for full duplex transmission of science, telemetry, command, time, and timing data between the instruments and spacecraft. Details of this development and implementation are presented in this paper.

1. Introduction

The Geostationary Operational Environmental Satellite Program (GOES) is a joint effort of NASA and the National Oceanic and Atmospheric Administration (NOAA).

Currently, the GOES system consists of GOES-12 operating as GOES-East in the eastern part of the constellation at 75° west longitude, and GOES-10 operating as GOES-West at 135° West longitude. These spacecraft help meteorologists observe and predict local weather events, including thunderstorms, tornadoes, fog, flash floods, and other severe weather. In addition, GOES observations have proven helpful in monitoring dust storms, volcanic eruptions, and forest fires.

The benefits that directly enhance the quality of human life and protection of Earth's environment include:

- Support the search and rescue satellite aided system (SARSAT)
- Development of worldwide environmental warning services and enhancements of basic environmental services
- Improvement of forecasting and providing real-time warning of solar disturbances
- Data that may be used to extend knowledge and understanding of the atmosphere and its processes

GOES-N is the first spacecraft in the current GOES-N/O/P series and launched on May 24, 2006 aboard a Boeing Delta IV Rocket.

The GOES R Series (GOES-R) is currently in the formulation phase and follows GOES - NO/P. The first launch is scheduled for the 2012 timeframe [1].

The GOES-R spacecraft uses European Cooperation for Space Standardization (ECSS) SpaceWire [2] for the transfer of sensor, telemetry, ancillary, command data, time code, and time synchronization between instruments and the spacecraft. The GOES-R Project has directed that all data transferred over SpaceWire implement a reliable data delivery protocol. Early in the GOES-R Project development a decision was made to develop GOES-R specific SpaceWire technology to aid in cost and risk reduction. In response to this direction reference hardware and software solutions have been fully developed and verified to be compliant with the SpaceWire standard and GOES-R Project requirements. Intellectual property (IP) developed by this effort is available through the NASA Technology Transfer Office.

2. SpaceWire Test Card

GOES-R instrument-to-spacecraft data rates are between 10 and 100Mbs. Also, error detection and correction, at the source packet level, is needed. In addition to meeting these key requirements, SpaceWire provides a simple interface with minimum cabling. The GOES-R Project set a goal that all SpaceWire computational requirements have a minimal impact on the on-board Single Board Computer (SBC).

Upon selection of SpaceWire for instrument-to-spacecraft communications, a search for suitable hardware solutions was completed with the selection of the British Aerospace (BAE) SpaceWire Application Specific Integrated Circuit (ASIC) as the GOES-R reference design. The block diagram of the BAE SpaceWire ASIC is shown in Figure 1. The BAE ASIC is an off-the-shelf heritage part modified to implement the NASA Goddard Space Flight Center (GSFC) SpaceWire 4 port router IP core.

The next step in this process was the design and fabrication of the SpaceWire Test Card. In an effort to minimize cost and utilize one of the BAE ASIC's two PCI ports, the test card is designed to plug into a standard Windows based motherboard. In addition to the on-chip 32 kbyte SRAM, the test card has 4 Mbytes of SRAM and 16 Mbytes of DRAM. Numerous test points and logic analyzer probe attachments are included in the card's design. Custom Windows drivers and support software were developed to make the test card fully operational.

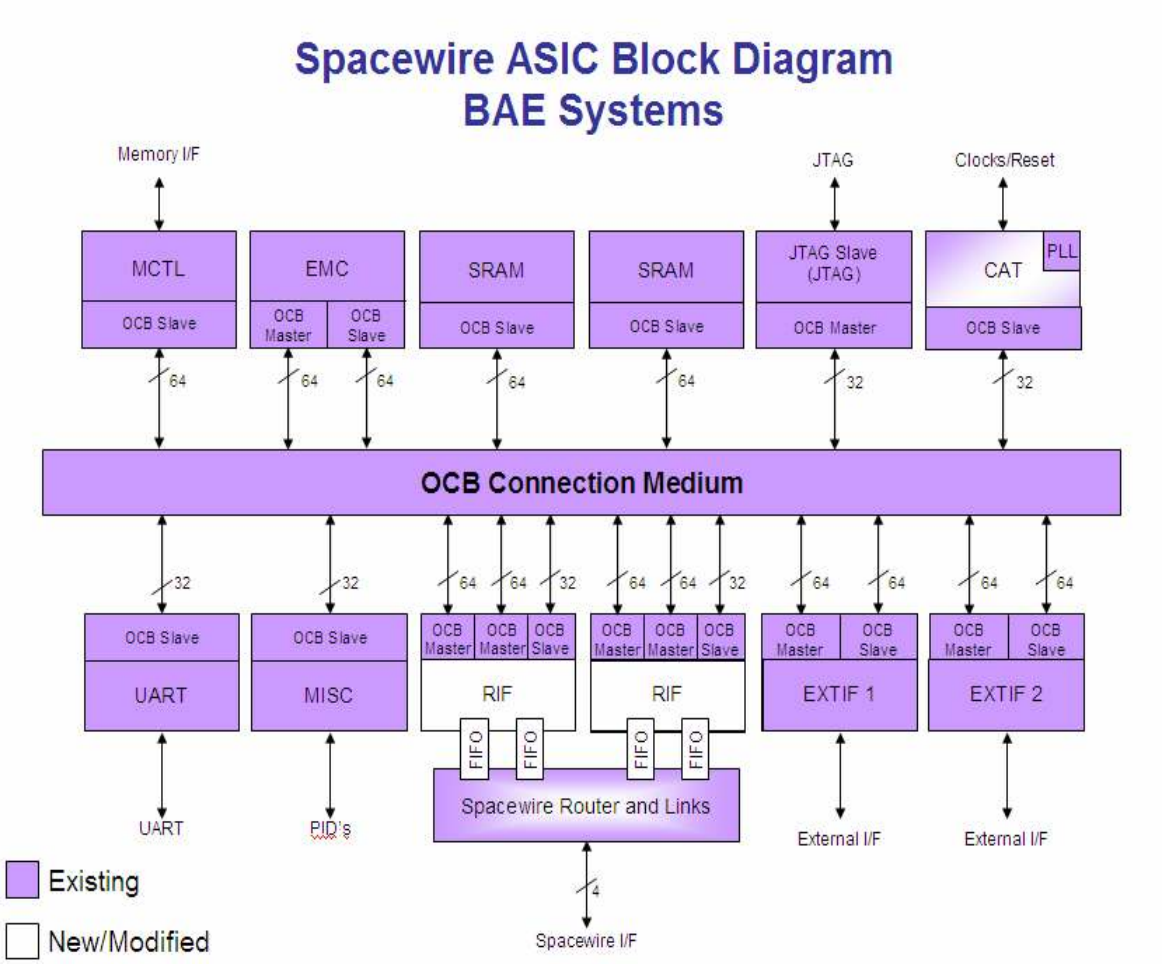


Figure 1. BAE SpaceWire ASIC Block Diagram

Once the test card, motherboard, and support software were fully integrated and tested work began on modelling the GOES-R flight data system. The test card mounted in a test station motherboard is shown in Figure 2. The current flight data system test configuration includes a Command and Data Handling (C&DH) using the test card, another test card based work station simulating a high rate instrument, and two Field Programmable Gate Array (FPGA) based work stations acting as low rate instruments.

Figure 2. GOES-R SpaceWire Test Card in test station

The BAE ASIC has an on-chip embedded microcontroller (EMC). The ASIC based work stations were used to develop EMC software for traffic and error management.

To date, the test card has served in a number of roles and is quite easy to use. In addition to modelling the GOES-R flight data system it has verified instrument interfaces, acts as Ground Support Equipment (GSE) and functions as an EMC software development platform. Plans are in process to incorporate the test card workstation as a "gold standard" for all GOES-R instrument and spacecraft providers.

2. Reliable Data Delivery Protocol Overview

Since the GOES-R Project requires data transferred between the spacecraft and instruments have error detection and correction, a search for a simple protocol was initiated early in the program. Modelling with off-the-shelf protocols indicated they required more computational resources than GOES-R desired. Work then shifted to develop a simple protocol above the SpaceWire packet layer requiring minimal computational resources while providing reliable data delivery. Since the BAE SpaceWire ASIC was already the GOES-R reference hardware solution, the target processor for protocol processing was the on-chip EMC. Software requirement definition and a formal protocol document development activities proceeded in parallel. The end result is the GOES-R Reliable Data Delivery Protocol (RDDP) [3] allowing up to 128 virtual channels, reliable data delivery (RD) mode, and urgent message (UM) mode over a single SpaceWire physical connection. The EMC software meets all protocol and GOES-R requirements with significant margin.

3. Data Reliability

A SpaceWire link bit error rate (BER) is to be 10^{-12} or less [4]. With this BER specification, data transmitted at 100 Mbps results in 1 error every 2.78 hours [4]. Since GOES-R is in constant view of the ground station, on-board sensors provide data around the clock without interruption. If an error occurs about once every three hours then 8 errors happen in a 24 hour period. In a SpaceWire link a data error can result in the loss of a packet. The SpaceWire standard [2] requires clearing the transmit buffer on detection of a SpaceWire link error. This creates an incomplete packet and is discarded by higher level processing. Assuming the largest packet sent over the link is 65 kbytes, a single bit error can cause the loss of approximately 500 kbits. This is a worst case scenario and yields a potential loss of 4 million bits per day, not accounting for orbit-related data loss. Implementation of the RDDP provides a mechanism to eliminate the effects of bit errors.

A process at a higher level than specified in the SpaceWire standard [2] detects data errors and requests packet retransmission. This is one of the two reasons GOES-R developed the RDDP. The second reason deals with implementation of an error management protocol. If the GOES-R Project did not develop the RDDP and impose it on the instrument and bus developers, every developer could potentially provide different solutions to the problem. This is a high cost and risk approach. The RDDP sets the rules for everyone on the project. With the RDDP no developer needs to spend time and money on a solution to the error management problem.

The RDDP complies with ECSS-E-50-12A [2] and functions as point-to-point or in a network. GOES-R made an effort to keep the RDDP from being GOES-R specific

and can be used by any organization wishing to do so. The SpaceWire Working Group assigned decimal 238 as the RDDP protocol ID.

4 Virtual Channels

The RDDP header destination and source SpaceWire Logical Addresses (SLA) are a data source and sink pair creating a virtual channel. For example: one SLA pair sends telemetry data from an instrument to the spacecraft C&DH. Another SLA pair sends command data from the spacecraft to the instrument. A third pair sends sensor data from the instrument to the spacecraft and so on. In addition, time code messages and time synchronization transfer between the spacecraft and instrument. This is all done over a single SpaceWire physical connection.

5. RDDP Operation

The RDDP accommodates two operational modes RD and UM. All RD packets require an acknowledgement (ACK) from the receiver. All UM packets are fire-and-forget. RD packets must utilize the header sequence number for sliding window, missing packet detection, duplicate packet detection, and packet order processing where UM packets do not. Since simplicity is a driving RDDP requirement, any error condition outside the protocol's management ability causes the RDDP to stop and report this condition to a higher level process. If any packet is regarded as a duplicate it is discarded. Figure 3 shows nominal packet processing with fast ACKs and no errors. Normal data sequencing begins when source and destinations SLAs have been reset and enabled. The source sends a packet and on error free reception by the destination SLA an ACK is sent back to the source SLA. The sliding window is advanced by 1 and the process is repeated.

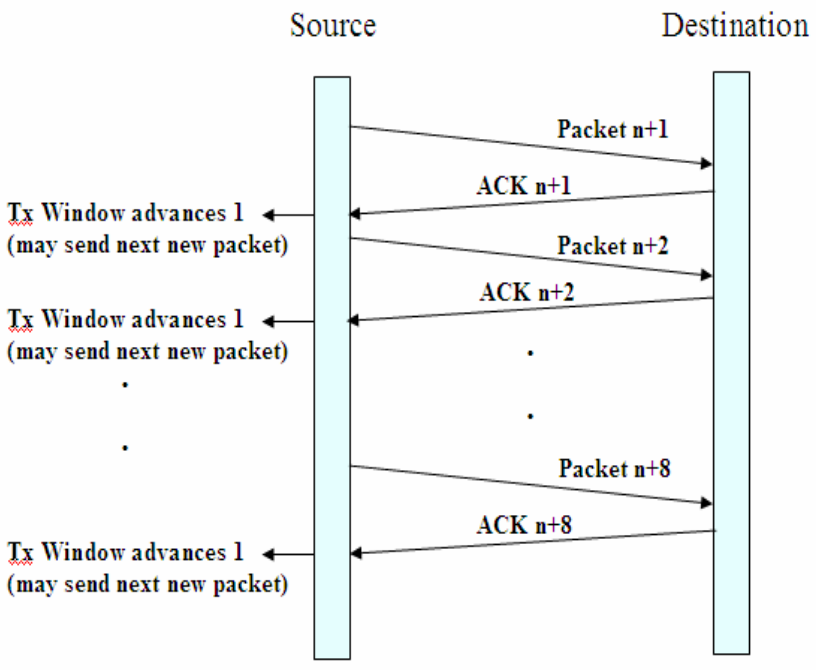


Figure 3. Nominal Operation with Fast ACKs.

Figure 4 is a case where packets are sent in burst mode or due to traffic congestion ACKs are delayed. Simply stated, the source SLA can send up to window size number of packets before having to wait to receive an ACK

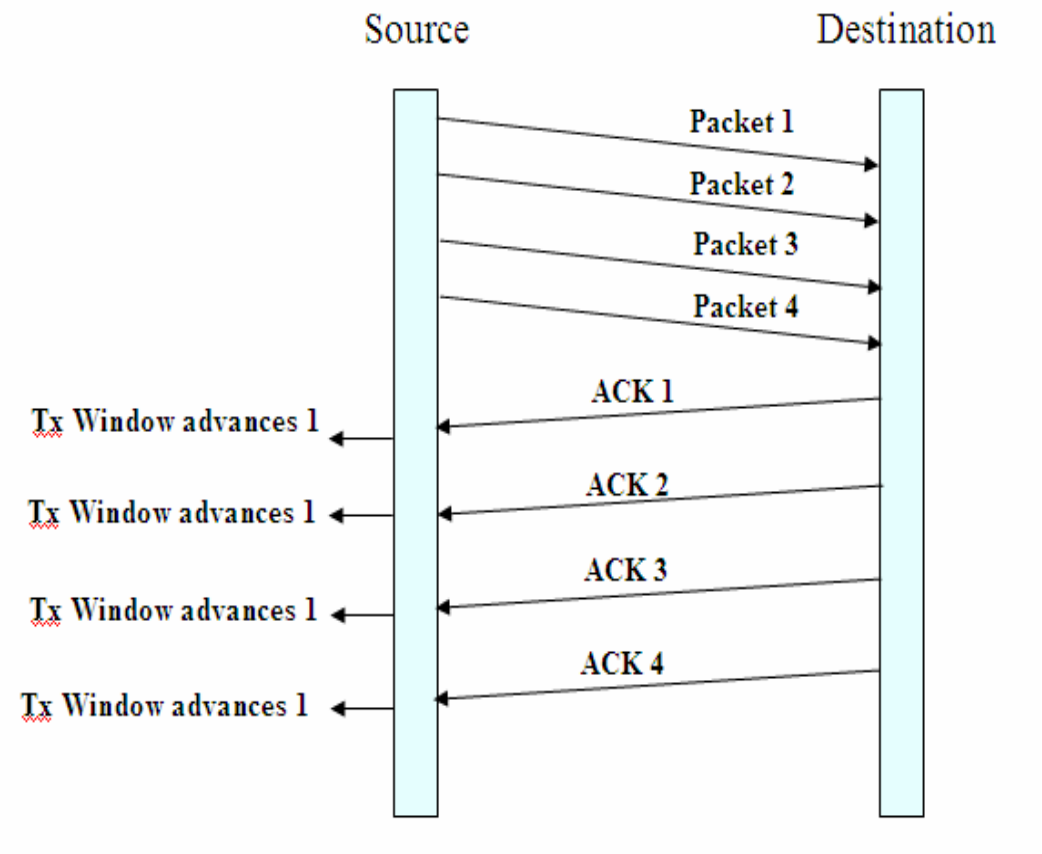


Figure 4. Burst packets or Slow ACKs

Figure 5 shows a case where the window size is set to 4 and the ACK for packet 2 is not received by the source SLA. The source window cannot advance beyond the number of contiguous ACKs received. The source SLA waits the timeout period and resends packet 2. If the ACK for packet 2 is received, the window is advanced and the virtual channel is clear to send the next set of packets. If the ACK for packet 2 is not received after the timeout period, the retry counter is incremented.

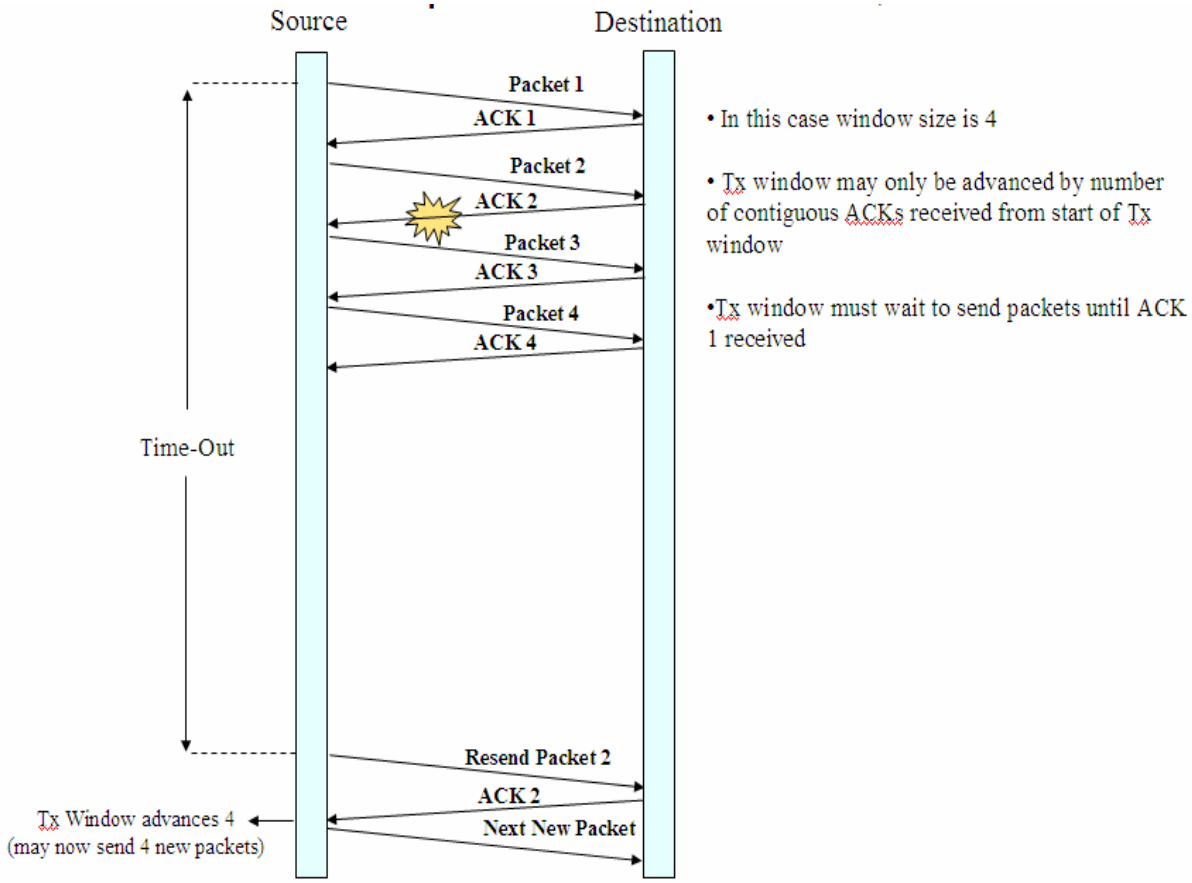


Figure 5. Destination SLA Fails to ACK Packet #2

6. GOES-R

The GOES-R C&DH serves as the hub for all data received by and sent from the spacecraft. The C&DH implements Consultative Committee for Space Data Systems (CCSDS) recommendations for both packet telemetry and telecommand communications. Multiple X-band and S-band services provide the space-to-ground and ground-to-space communications. Standard CCSDS packet multiplexing and Transfer Frame error coding are implemented on the communications links. The technology for moving uplink and downlink data between the on-board communications equipment and the C&DH is not yet known. The current state of the GOES-R Project has not moved completely into the implementation phase. Most instruments are under contract while the spacecraft procurement is still in process. It is not known if the use of SpaceWire beyond the instrument-to-spacecraft data interfaces will expand to other bus subsystems.

7. References

1. “GOES-R Web Page”, National Aeronautics and Space Administration Goddard Space Flight Center, http://goespoes.gsfc.nasa.gov/goes/project/index.html.

2. European Cooperation for Space Standardization, Standard ECSS-E-50-12A, “SpaceWire, Links, Nodes, Routers and Networks", Issue 1, European Cooperation for Space Data Standardization, February 2003.

3. NASA Goddard Space Flight Center GOES-R Project, “GOES-R Reliable Data Delivery Protocol”, 417-R-RPT-0050, January 2007.

4. S.M. Parks, SpaceWire Working Group, “SpaceWire: Serial Point-to Point Links”, Space Engineering, ECSS-E-50-12, January 2000

Plug-and-Play Satellite (PnPSat)

Session: Missions and Applications
Long Paper

Don Fronterhouse[1]
Scientific Simulation, Inc.
2951 Marina Bay Dr. #130-306, League City, TX 77573
Jim Lyke[2]
Air Force Research Laboratory, Space Vehicles Directorate
3550 Aberdeen Ave SE, Kirtland AFB NM 87117-5776

E-mail:donald.fronterhouse@kirtland.af.mil, james.lyke@kirtland.af.mil

[Abstract] The Air Force Research Laboratory's (AFRL) Space Vehicles Directorate has initiated a program to create the first satellite based entirely on the principles of plug-and-play as represented by the Space Plug-and-play Avionics (SPA) approach. Unlike other satellites, PnPSat is designed to be constructed extremely rapidly, based on design descriptions that can be eventually produced automatically from a push-button tool flow. The plug-and-play satellite (PnPSat) employs modular components, from the structural panels to the guidance and health/status devices, taking full advantage of the self-describing mechanisms inherent in the SPA approach. Panels contain SpaceWire routers and multiple connection sockets to support the arbitrary arrangement of spacecraft components on the panels and the connections between panels. In most regards, PnPSat reduces the integration of a satellite to a simplified assembly process, analogous to the assembly of components on a personal computer in which components are enumerated by the host as they are added. Since all components are based on the same, self-describing interface, the proliferation of disparate simulators and emulators are sharply reduced, and a unified "test bypass" mechanism is provided to facilitate hardware-in-the-loop simulation of a single component or the entire satellite at any point during integration of the system. This talk will describe the background and status of the PnPSat development program.

1. Introduction

To accelerate the development of complex space systems, fundamentally new approaches will be required over those used in conventional spacecraft today. The idea of "plug-and-play" (PnP) suggests an ease of integration is possible, and indeed the concept has found popular use in terrestrial systems. Evolution in aerospace systems has been decidedly more measured, cautious, and incremental. To achieve the benefits of PnP, it is necessary to "invest silicon" into the interfaces of components within a system, not to improve the performance of these components, but rather to improve the ability to more quickly make use of them. Given the considerable expense of radiation-hardened spacecraft components, the notion of diverting machine cycles into interfaces and away from primary performance may seem like profligacy. Or not, perhaps, in considering that the majority of cost in a space system can be traced to labor, and intelligent interfaces can reduce much of the labor associated with component / system integration.

[1] Chief Scientist.
[2] Technical Advisor, Space Electronics Branch, AFRL/VSSE, senior member.

The AFRL's Space Vehicles Directorate began a research program to understand the complexity of space systems and how, through technology, it would be possible to create them much faster. The core ideas of our approach, referred to as "Space Plug-and-play Avionics" (SPA), employ intelligent interfaces to accelerate component integration. In principle, SPA components are analogous to the USB components of a personal computer in that they embody the concepts of self-organization (networks are formed by simply plugging components together), self-description (through the use of electronic datasheets embedded in components), simplified connections to devices, and interchangeability of devices. SPA software concepts (namely the "Satellite Data Model" [4]) encourage the creation of PnP "awareness," striving in effect to make operational flight programs more insular to differences in components. A companion set of "push-button toolflow" concepts [5] promote the rapid construction of spacecraft through a set of ideas very analogous to menu-driven consumer product purchasing or more approximately to electronic design automation, in which ideas are germinated in a "capture" process and evolved into a buildable specification.

The ideas of SPA sidestep decades of "incrementalism" in favor of an architectural "clean sheet approach." Though applicable on a limited scale (and some limited demonstrations are under development), the ideas are more compelling when they can be conducted on the scale of an entire platform. Spacecraft are built to perform missions, and missions are driven by payloads. Unfortunately, many payloads, even in new procurements today, are based on non-PnP legacy interfaces. Our early attempts to create a PnP platform were met with strong resistance, as replacing a legacy payload interface was viewed as inevitably adding cost, complexity, and risk. Resolving this particular form of the "chicken vs. egg" dilemma could only be resolved through a new experimental development, designed as a Plug-and-play (PnP) satellite (PnPSat) from first principles.

This paper is organized as follows. In the next section, we review relevant background technology and events shaping the emergence of the PnPSat. The following section will describe the philosophy behind PnPSat, the mission template around which it has been cast, and the management approach for the project. The next section will discuss the spacecraft subsystems, and finally we will review the status and future plans for PnPSat and follow-on activities.

2. Background

AFRL's Space Vehicles Directorate has conducted several studies showing the confluence of a number of space electronics technologies are enabling to the objectives of creating a spacecraft rapidly, including microsystems (the combination of microelectronics, advanced packaging, and microelectromechanical systems), high-performance computing on orbit (HPCOO), and reconfigurable systems approaches (such as field programmable gate arrays, adaptive wiring, and software-definable radio concepts). One of the technologies involved the notion of "peel-and-stick" or appliqué sensor modules (the idea was not strictly limited to sensors, but the term "stuck"). The sentiments of "peel-and-stick" components evolved an approach referred to as Space Plug-and-play (PnP) Avionics (SPA) [1]. In the SPA concept, complex components are encapsulated with simple but intelligent standard electrical interfaces, such as USB (SPA-U) and SpaceWire (SPA-S) [3]. The principles of the SPA approach are summarized as follows:

1. Component Physical / Functional Encapsulation

The concept of encapsulation is important, as it serves to hide complexity within modular compartments, presenting inasmuch as possible an apparently clean and simple interface. In fact, much of the "magic" of PnP occurs below the surface. SPA components, for example, carry their own descriptions, referred to as XML-based electronic datasheets (xTEDS). As it is not normally necessary for a personal computer user to be concerned with the inner workings of common components (i.e., mice and keyboards), the xTEDS mechanism in SPA makes it possible for components to carry their own documentation.

2. Self-Forming Networks

Self-describing components can be used to automatically construct networks. In SPA, devices are "endpoints," connected together through hubs (SPA-U) or routers (SPA-S), and the structure of the network is induced through assembly and automatically inferred by the system. The "endpoints" range from traditional bus components, such as gyros and reaction wheels, to payload elements, such as cameras. Even spacecraft structures can be viewed as components, perhaps sub-networks of SPA endpoints and hubs/routers. The paradigm of a "machine-negotiated interface" was felt to be especially liberating for spacecraft developments, which have a notorious reputation for cost and schedule overruns. Reducing the need to "think" to first order allows system developers to concentrate on core challenges in developing a complex platform without being mired in the myriad details of simple components.

The supporting mechanisms within SPA to support self-forming networks are encompassed in hardware and software features. In hardware, endpoints ideally need only form a connected topology in which relative ordering is unimportant. This insensitivity to location in the network frees the system developer from worrying about where the network "needs" to place components, except for the "real-world" constraints that sensibly apply (i.e., such as the need to ensure that reaction wheels are placed in orthogonal/orthonormal geometric relation). The binding of geographic information can also be generated in principle automatically, derived from the placement of components in a PnP system. Power distribution is distributed, with much of the burden of switching being placed on SPA hubs/routers.

For that matter, a spacecraft need not have a central processing and power distribution functions, as it is common practice in spacecraft to have a centralized "command-and-data handling" (C&DH) element for processing and an "electrical power subsystem" (EPS) for power management and distribution. Eventually, through SPA, the artificial constraints induced by a fixed network structure can eliminated.

3. Plug-and-play "awareness"

It is then only necessary for components to "understand" each other relative to the features or "services" they require of each other. In the development of SPA, this need gave rise to the concept of the Satellite Data Model (SDM) [4], which provides self-discovery and self-configuration capabilities to SPA. SDM provides a number of key mechanisms that organize a network and the devices it contains, in a manner similar to web services [6]. In this sense, even the most complex software applications can be viewed of as compositions of primitive transactions, based on communications between SPA elements using the "service descriptions" contained in the xTEDS. Software applications at the spacecraft level, to be effectively "plug-

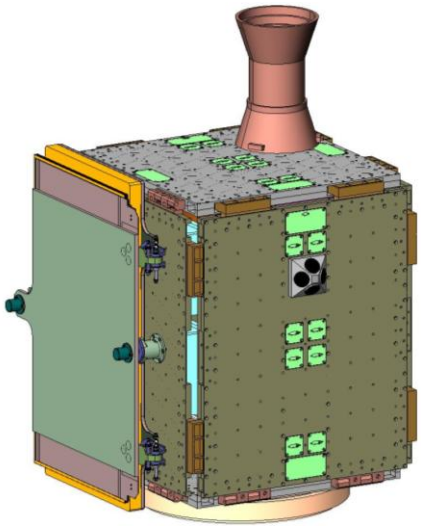
and-play," are written to harness the SDM infrastructure. One interesting side effect of this constraint is that "pieces of software" also typically contain xTEDS. Similarly, from the view of hardware components in a spacecraft, SPA devices (endpoints, hubs, and routers) must be developed to enable SDM to organize the spacecraft network automatically. For example, in the SpaceWire-based form of SPA (SPA-S), this requirement gave rise to a need to define a PnP protocol, whereas the automatic enumeration mechanisms already exist in the USB form of SPA (SPA-U).

4. Hardware-in-the-loop simulation (HWILS) and test bypass

A powerful concept in SPA, though not directly related to "plug-and-play," involves the ability to support the injection of synthetic information or instrumentation of data from components. In the simple example of a SPA thermometer, it is possible to substitute the ambient temperature values generated by the thermometer with a desired control value in a way that does not perturb the existing SPA network. Conceptually, the ability to insert or examine the state of variables throughout a SPA network is similar to the boundary scan principles in JTAG interfaces [7]. When coupled with a simulation infrastructure, the test bypass facility of a network of SPA devices becomes an *in situ* hardware-in-the-loop system. The philosophy behind test bypass is that rapid system development benefits (if it does not in fact require) improved abilities to dry run part of all of the system and to expose hooks to improve test and debug in the event of inevitable irregularities that are like to occur in development.

3. The Need for PnPSat

To gain a better appreciation of the considerations driving the PnPSat approach, we review conventional wisdom and the emergence of smaller spacecraft, oriented on shorter-term tactical needs. PnPSat represents an extension of the ideas of small, tactical satellites and modular design approaches, with an infusion of technology concepts that aim to simplify and accelerate the construction of spacecraft. The pursuit has not been without obstacles, and these have given rise to the need to consider PnPSat as a "clean sheet" approach, an alternative to a protracted, incremental technology insertion strategy.



Space electronic systems have evolved gradually over nearly half a century, and new architectures have been seldom attempted. A contemporary spacecraft is typically a polyglot of legacy hardware ("it worked before") fastened to new hardware, often requiring extensive additional engineering for each interface. These are in effect like a complex hardware/software "glue," applied liberally to achieve the desired effect. Even when a program offers a stubborn insistence on the use of an interface standard (popular choices include MIL-STD-1553 and RS-422), the likelihood that any two independently developed components would work in the same network without significant additional work is highly unlikely. Components, even if their interface designs are simple, are not typically designed to "explain themselves" to a system, but require significant human effort to reconcile. Spacecraft are not designed to accommodate unknowns or late

changes in a system's design. Wiring harnesses are painstakingly defined to implement specific configurations. For these and many other reasons, the engineering of systems may be robust at one level (designed to operate reliably in a harsh environment), but they are at the same time fragile to change. And the longer it takes to develop a system, the more likely indeed it will be necessary to change the system in some way. As such, typical spacecraft are very expensive and take a long time to develop.

Small satellites, in principle, are believed to be less expensive, quicker to develop, and faster to checkout on orbit and bring online to satellite operators [8]. Small satellites remain a controversial proposition in military space. Some equate "small" with "responsive", but there is a conventional wisdom that suggests that capability scales with larger spacecraft.

Small spacecraft are formed in much the same way as larger spacecraft, subject to the same complexities and integration challenges, albeit at a smaller scale. The challenges might be met decisively through the use of SPA, but the level of commitment has been too different perhaps from that dictated by conventional wisdom to attempt. Breaking this cycle, we ultimately felt, would not occur until someone attempted a PnPSat.

4. The PnPSat Spacecraft Approach and Developmental Philosophy

And attempted we have. PnPSat represents the first spacecraft of its kind, not from outward appearance but from first principles as platform based on a self-organized network of self-describing components. It is modular, but the application of modular approaches in spacecraft is not a new concept. PnPSat can be viewed as the combination of modularity and complexity hiding. Most of its wiring harness will be invisible, recessed within panels.

PnPSat is a pure science and technology experiment to establish the necessary technologies to implement an evolving breed of software defined systems. Over the last several years we have touched every aspect of satellite design and construction, as well as test and operation to find those areas that inhibit the six-day spacecraft. What we found is that we must simplify the interfaces by hiding complexity. In PnPSat we are applying the principles of plug-and-pay to the mechanical, electrical and software interfaces.

What is a plug-and-play satellite? It is a modular satellite with open standards and interfaces, self describing components, and an auto-configuring system. This results in system integration and testing tasks that can be automated and are themselves simplified. Modular spacecraft structures also allow components to be mounted either on the inside or outside on regular grids. We are currently using a 5 cm x 5 cm grid. The system also employs modular flight software that is both easy to maintain and can be reused for various satellites as well as is intrinsically autonomous. Our goal is to have a satellite capable of maintaining its own health and status and only needs to talk to the ground by exception and for user tasking. High-performance-computing-on-orbit (HPCOO) provides gigaflops of processing to the user to support both the autonomy and on orbit processing of sensor data. We want to be able to provide to the user not only raw data as appropriate but also processed information. We also are working on tactical user interfaces that allow the user to task a satellite based upon that satellite's capabilities. We are working with experimenters to develop plug-and-play experiments. Since this is a science and technology satellite, we use experiments as payloads. Distributed power systems support plugging in a

battery on one panel and solar arrays on another. Main bus power and charging grids are distributed throughout the spacecraft allowing access to the main power grid from anyplace on the spacecraft. This access is protected with circuit breakers in case something goes wrong. We are also investigating plug-and-play launch vehicle interfaces.

4.1 Requirements

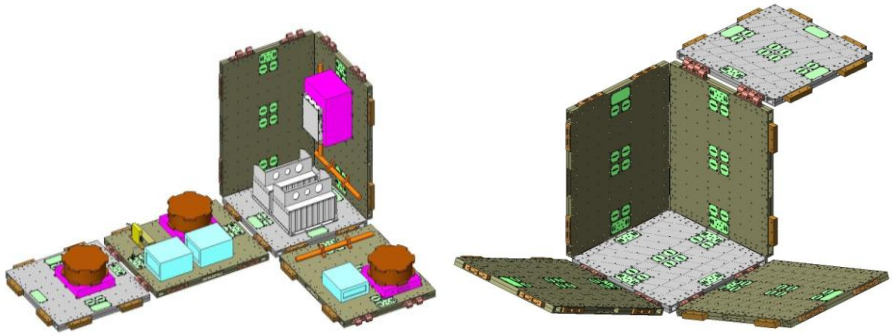
PnPSat requirements fall into three basic categories: the overall Responsive Space program requirements, the PnPSat program requirements, and the primary system capabilities that need to be demonstrated. The Responsive Space program requirements include demonstration of the viability and maturity of a modular plug-and-play architecture. It is important to be able to transition TRL-6 technologies to other satellite programs.

The primary system capabilities include being able to demonstrate rapid design, assembly, and test. Of course, we must be able to demonstrate modular plug-and-play, including both SPA and the Satellite Data Model. We must also demonstrate distributed systems including power, thermal, computing, and control. On the software side, we must be able to demonstrate robust autonomy including both dynamic schedules and activities.

4.2 PnPSat Architecture

One way to look at the PnPSat architecture is to break the spacecraft into three basic parts. First, the basic bones of the spacecraft upon which all components are attached. This includes the spacecraft structure, the power grids (both main and charging), the SPA infrastructure, and thermal control. Second, we add components that provide robust performance including the autonomous flight software; the quantity of high-performance computing; power generation and storage; guidance, navigation, and control components; and the communications radios for both tactical and TT&C. Finally, we add the mission sensors that provide customization for warfighter needs. From the perspective of building and testing the satellite, we must consider assembly, integration, and test; the ground systems; and the launch systems.

The PnPSat structure features modular panels to support quick assembly and the flexibility to mount components in multiple places. There are standard plug-and-play mechanical and electrical interfaces that can accommodate 48 experiments, and the components are located on either the interior or exterior surfaces. A tactical satellite requires approximately 25 to 28 components, which provides us with sufficient flexibility to mount the components based upon mass, thermal, power, and FOV requirements, among others. Electronics infrastructure and



harnessing is recessed within each panel to increase available footprint and volume for the plug-and-play components and experiments. Locking hinge joints allow panels to rotate about the

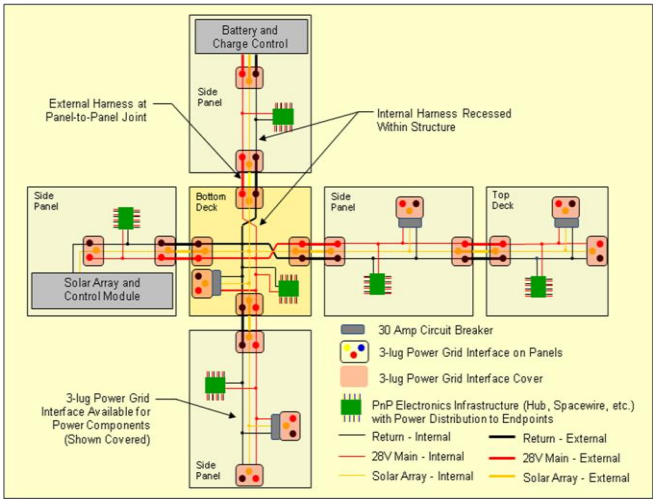
hinge line for easy access to the interior. Inter-panel jumpers, which harnesses across joints, allows the plug-and-play electrical network to remain intact throughout assembly, integration, and test. This means that we can determine if a component is working as it is assembled on the spacecraft. Currently, the panels are machined from 6061-T6 aluminum. The current structure is 51 x 51 x 61.2 cm and weighs 34.7 kg excluding the launch vehicle adapter.

One of the advantages of the folding PnPSat concept is that it can be changed easily to various configurations to support requirements for different stages of the project. At first it can be opened up into a flat configuration for internal components to be mounted and tested. Then it can be closed, while still active, for the external components to be mounted and tested. Panel to panel joints are pinned to allow panels to be rotated from the horizontal flat to vertical folded configuration. Then securing the joints with bolts provides for a rigid structure. Individual panels or sets of panels can be integrated and tested in parallel.

One of the modularity keys is to have a standard simple mechanical interface between the components and the structure. We have established a simple, standard mechanical interface to increase the flexibility and to speed integration. We have initially selected a 5 x 5 cm grid pattern that goes completely across the internal and external surfaces of all panels. The holes are threaded to support #8-32 fasteners. The hope is that eventually new components and experiments will be designed to accommodate this interface. In the meantime, existing components can be integrated with a simple adapter plate. This is the approach we are using on PnPSat to match legacy components to the modular structure.

The SPA electronics infrastructure is recessed within the interior of each panel including boards and inter-board harnessing. The power and data services provided to each of the eight SPA endpoints on each panel are handled by the robust hub. Panels are networked together, including power and data using the inter-panel harnessing. Once the SPA infrastructure has been installed and tested the panel halves are bolted together to form an EMI tight enclosure.

Each of the eight SPA endpoints has a standard electrical interface for components and experiments. For PnPSat the standard electrical connector is a 25-pin micro-D containing data (both Spacewire and USB), power (up to 4.5A @ 28v), time synchronization pulse, test bypass interface, and single point ground. Endpoints can be located on either the interior or exterior surface of the panel. Batteries, solar arrays, and power supplies have access to the power grids through 2-lug interfaces.



4.3 PnPSat Components

There are 25 PnPSat components plugged onto the structure. These include two coarse sun sensor assemblies, three reaction wheels, three magnetic torque rods, a fine sun sensor, a magnetometer, two batteries, FITS solar array, GPS radio, two packages of HPCOO processors, an Intelligent Data Store, and

a TT&C radio. We believe all components that plug onto the structure should be plug-and-play. Our initial studies have shown that by recessing the electrical infrastructure and harnessing inside the panels, we significantly increased flexibility for component and experiment mounting.

To enable a plug-and-play power system, the bus power grid is composed of two separate grids: the main power grid, and the battery charging grid. These grids extend across all of the panels, allowing batteries, and the solar arrays to be connected to their grids from anywhere on the bus. High power components can gain access to the main power grid via 30 amp circuit breakers. The battery charge control electronics and the solar array controller are also SPA components. By separating the charging and main power grids, we enable a Phoenix mode, where even if we disconnect the main power grid due to low battery charge, we can still use opportunistic photons to charge the batteries. After the batteries reach sufficient charge, the battery and charge control electronics reconnect the battery to the main power grid and the satellite reboots.

The SPA infrastructure consists of the Appliqué Sensor Interface Module (ASIM), robust hub, hardware in the loop router (for ground testing only), the SpaceWire router, and the high power circuit breakers. The ASIM is used to interface legacy components to the SPA network. The ASIM also has two major functions. First, it is charged with the care and feeding of the attached component. Second, it presents a standard plug-and-play interface to the SPA network. The ASIM contains the xTEDS that defines the devices' data products, accepted commands, supported interfaces, and services provided. This allows each component to be self describing to the SPA data network. In addition, the ASIM provides a very accurate, real-time clock, and the hardware-in-the-loop test bypass interface.

One of the fundamental changes being implemented in PnPSat is the concept of a data-centric architecture. Traditional systems engineering is component centric, relying upon a detailed component interface control document (ICD) to enable system configuration. SPA enables us to focus more on the data rather than the details of the component. Data can be described, moving from the more fundamental to the more specific, as the basic physics, measurable quantities measured through a measurement process yield variables and qualifiers that we provide names and formats, and gather all of this up into the ICD. Now if we were able to agree upon the meaning of measurable quantities - for example, attitude or position or pressure or temperature - and place that in a Common Data Dictionary (CDD) for all to share and place the variable names and qualifiers and their formats in the xTEDS, we could then implement a standard SPA interface and get rid of the ICD. In this way, we have defined both a plug and a play interface, where that data interface is based upon a common standard (CDD) of what data means that is distributed to all, a standard data interface expressed in a standard language (XML), and the electrical interface based upon a common SPA standard.

The robust hub provides both a USB hub and endpoint power distribution and monitoring. Each SPA endpoint can be supplied up to 4.5 A @ 28 V protected by a circuit breaker. In addition, there is a current monitor on each endpoint with a soft breaker that can be set based upon the power required for that component as described in its xTEDS. In addition, the robust hub provides control of the high power circuit breakers. The robust hub uses an ASIM to provide power interfaces and control functionality, much like any other component.

4.4 HPCOO Components

PnPSat will be the first space implementation of the AFRL-developed Wafer Scale Signal Processor (WSSP) high-performance computer. There are six processors per chip, organized either as voted triplets with 6 MB of EDRAM each, that can detect and correct SEUs or as six independent processors with 2 MB of EDRAM each. In addition, each chip has 2 SpaceWire ports and FIFO interfaces. This processor has been completely synthesized and produced providing greater than 1 GFLOP per watt at 125 MHz. PnPSat will be flying eight of these chips, providing 24 GFLOPS of processing power.

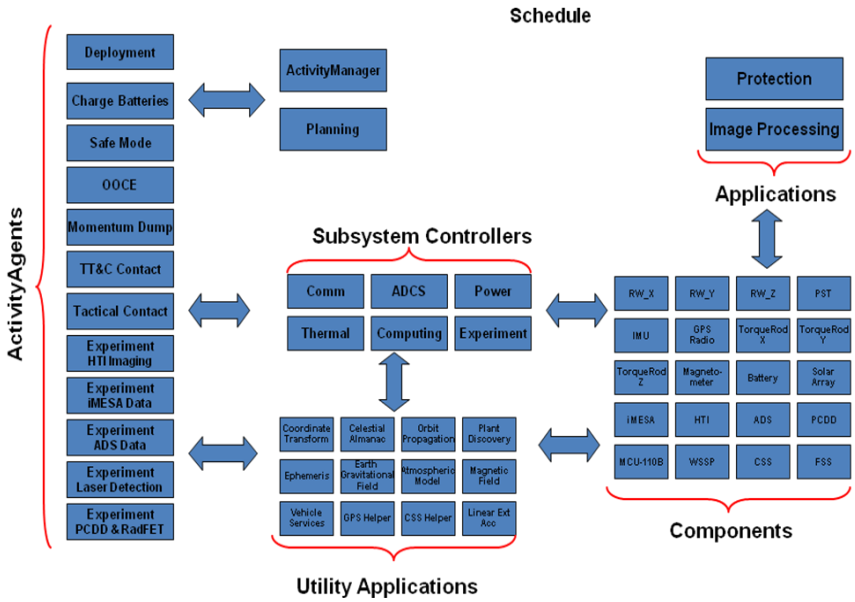
To provide on-orbit data storage, we are developing an Intelligent Data Store (IDS) that is fully SPA compliant. The IDS uses a Vertex 2 FPGA, with up to 4, 32-bit Microblaze processors and 512 MB (with a potential of 11 GB) of error corrected flash and 128 MB of error corrected RAM. The IDS resides on the SpaceWire high-speed data network and runs SDM applications and provides file storage and retrieval for system configuration data, application executables, and telemetry data.

4.5 Autonomous Flight Software

To assemble a spacecraft in two to three days means that we will not be able to write any custom software. Our focus on modularity includes the ability to develop software applications before the satellite mission or the specific components of the satellite are known. To facilitate the independent and concurrent development of hardware devices and software applications, we have developed a sideware application called the Satellite Data Model (SDM). SDM allows for the last-minute integration of independently developed hardware and software while supporting self configuration and self discovery. SDM is the play side of modular plug-and-play. It also provides a support model for fault tolerance to loss of devices, loss of software applications or services, and loss of SDM components.

There are five applications or managers that comprise the Satellite Data Model. A Processor Manager resides on each processor in the computing system and provides for the orderly execution of tasks on that processor. It is responsible for the underlying messaging services, and for dynamically selecting applications (tasks) to execute that are compatible with its resources. Tasks to be executed are posted to a Task List maintained by the Task Manager. The Processor Manager periodically reviews the current Task List and requests those that match its available resources. The Task Manager then assigns the task to one of the responding Processor Managers for execution. The Processor Manager also provides a heartbeat to the Task Manager to guard against processor failure. The Data Manager maintains a database of all xTEDS that have been registered by components and applications. The Data Manager provides a query and discovery mechanism whereby other applications can determine what data is available in the system, who provides it, and how to get a hold of it. The ability to be able to find single data elements within the data system is a key capability of a data-centric architecture. If the data network is composed of two or more sub-networks (for example SPA-U and SPA-S), a Sensor Manager is used to bridge the two networks. Finally, a Network Manager is used to discover the elements of the data network, their addresses, and in the case of SpaceWire the path routing between any two elements on the data network. We call the Satellite Data Model sideware because it is only used to discover the network and the data elements in it. Once a message has been subscribed, the data flows from producer to consumer as peers and SDM steps aside and does not get in the way.

There are two ways to look at the flight software architecture. The SDM discovery model provides for a flat architecture, where any application can get or provide data from/to any other application or from/to any component as necessary. This is an extremely flexible architecture, but more difficult to manage. We also have a more hierarchical architecture that is composed of controllers, agents, and managers that is conceptually easier to manage. It is important to remember that controllers do not own the devices they use to provide control. For example, the ADCS Controller does not own the reaction wheels, but does use them to control spacecraft attitude.



The PnPSat flight software core functionality is implemented as a group of autonomous activities. An activity is defined as a function that requires coordination of multiple subsystems and needs to be scheduled. Implementing flight software using SDM provides usability beyond just PnPSat. There are five basic categories of flight software in the hierarchical model. Subsystem controllers (for example Power, Communications, Computing, Thermal, ADCS, and Sensor) support both planning and commanding interfaces. System order is maintained by an Activity Manager that keeps the schedule and places activities to be executed in the schedule based upon time window and priority. We break priority into both a base priority associated with an activities importance to the satellite mission and an urgency that is time-dependent. For example, an activity to charge the batteries becomes more urgent the greater the depth of discharge. When it is time for an activity to be executed, the Activity Agents enables the associated Activity Agent. Activity Agents implement the basic activities of the satellite such as charging batteries, maintaining thermal control, collecting imagery, and safe mode. Activity Agents provide the heavy lifting to get things done and are required whenever more than one subsystem must be coordinated. Utility support applications such as coordinate transforms, orbit propagators, and celestial almanac, provide general-purpose support. Finally, there are the general purpose applications, such as satellite protection, image processing, etc. that are not associated with any specific activity.

Perhaps a PnPSat separation timeline will help to illuminate how all the various controllers, agents, and managers work together to bring the satellite up from cold at launch to a fully functional system. First, the separation switch closes as the satellite leaves the launch vehicle allowing the battery ASIM to provide battery power to the bus at which point the robust hubs boot providing power to the endpoints. As each ASIM boots, it provides control to its attached device. The WSSP ASIM boots the WSSP processors and loads and executes SDM. After network discovery by the Network Manager, the Task Manager is started and retrieves the initial

Task List from the IDS. The list includes the Subsystem Controllers, Activity Agents, Activity Manager, utility applications, etc. The Solar Array Activity Agent will place a deployment activity in the schedule via the Activity Manager and when executed by the Activity Manager will reduce tip off rates, deploy the solar arrays, and request ADCS go to sun point mode. Then, the normal activity agents take over and the satellite is up and running.

4.6 Building PnPSat

PnPSat will be built in the Responsive Space Testbed at AFRL's Space Vehicles Directorate, Kirtland Air Force Base, N.M. The schedule is very aggressive. We held a CDR last month and will have an AI&T Review by the end of this year. At that time, we will not have a full complement of flight components, but we will have tested with the engineering models. After AI&T, we will be taking the satellite apart, updating components, updating software, and testing to demonstrate that assembling and testing a semi-custom satellite in two to three days is achievable.

5. Acknowledgments

The authors would like to thank the AFRL's Space Vehicles Directorate and the Responsive Space thrust for its strong support of this effort.

6. References

[1] Lyke, J., Fronterhouse, D., Cannon, S., Lanza, D., and Byers, W. "Space Plug-and-play Avionics", proceedings of the AIAA 3rd Responsive Space Conference, Long Beach, CA, April 25-28, 2005.

[2] Lanza, D.; Lyke, J. ; Zetocha, P.; Fronterhouse, D.; and Melanson, D., "Responsive Space Through Adaptive Avionics", presented at 2nd AIAA Responsive Space Conference, Los Angeles, CA, April 19-22, 2004.

[3] Lyke, J., Cannon, S., Fronterhouse, D., Lanza, D., and Byers, T. "A Plug-and-play System for Spacecraft Components Based on the USB Standard", proceedings of the 19th Annual AIAA/USU Conference on Small Satellites, Logan, UT, 8-11 August, 2005.

[4] Sundberg, K., Cannon, S., Hospodarsky, and Fronterhouse, D., "The Satellite Data Model", International Conference on Embedded Systems and Applications (ESA'06), Las Vegas, NV, June 2006 (ISBN 1-60132-017-5/CSREA, Editor H.R. Arabnia).

[5] Strunce, R., Eckert, F., and Eddy, C., "Responsive Space's Spacecraft Design Tool (SDT)", Proceedings of the 4th AIAA Responsive Space Conference, Los Angeles, CA, 24-27 April 2006.

[6] Snell, J., Tidwell, D. and Kulchenko, P., *Programming Web Services with SOAP*, O'Reilly publishers, Sebastopol, CA, 2001.

[7] Joint Test Action Group (JTAG), "IEEE standard test access port and boundary-scan architecture", *IEEE Standard 1149.1*, 1990 and supplement *1149.1-2001*, 2001

[8] Singer, J. "U.S. Air Force Sees Operations Role for Small Satellite Platforms", Space News Business Report [online publication], 1 March 2004 (http://www.space.com/spacenews/archive04/platformsarch_030104.html, [cited 4 Apr 2007]).

OVERVIEW OF THE INTAμSAT'S DATA ARCHITECTURE BASED ON SPACEWIRE

Session: SpaceWire Missions & Applications

Short Paper

D. Guzmán[1], M. Angulo[2], L. Seoane[2], S. Sánchez[1], M. Prieto[1], D. Meziat[1]

1. Space Research Group. Dpto. Automática. Universidad de Alcalá

2. INTA (Instituto Nacional de Técnica Aeroespacial)

E-mail: dguzman@srg.aut.uah.es, angulom@inta.es, seoanepl@inta.es, ssp@aut.uah.es, mpm@aut.uah.es, meziat@aut.uah.es

ABSTRACT

This paper presents the INTAμSAT programme initiative and its data bus architecture, which will be based on SpaceWire for high data rate transfers. The first INTAμSAT-1 will be an Earth observation mission. This enlarged μSAT class is a further step after the NANOSAT programme success.

Inside the INTAμSAT-1 spacecraft, two types of data buses are foreseen. The OBDH data bus will be based on the CAN standard, and it will be a low data rate bus mainly used for TM/TC. In the other hand, it is foreseen a Payload Processor Unit (PPU) with SpaceWire interface for high data rate information exchange, with the cameras and the Mass Memory Unit.

1 INTAμSAT PROGRAMME INITIATIVE

After successful missions INTASAT (1974) and MINISAT-01 (1997), at the present time, INTA is focusing its efforts on small satellites, with NANOSAT & INTAμSAT programmes [1][2]. Most of the subsystems are developed at INTA with bilateral work with research institutions. At the same time, INTA tries to offer parts of HW or SW to the small business Spanish industries, to promote and encourage their entry to the space technology.

The launch of the first Nanosat-01 mission was performed on December 10th, 2004 by an Ariane-5 ASAP from Kourou. Nanosat-01B and it is still working healthy after almost 3 years in orbit, with the same store & forward communication mission and an enhanced UHF antenna, pointed to Earth, is planned to be launched in Oct. 2008 with a Dnepr from Yasny-Rusia. Just as a logical further step to the original Nanosat initiative, in October 2005, INTA started the INTAμSAT programme Phase-A feasibility study, to take full advantage of the Ariane-5 launch using ASAP. At present the preliminary design Phase-B is nearing the end (Dec. 2008).

INTAμSAT is conceived as a multimission system optimised for 600-700 Km LEO orbits, with inclinations ranging from 0º to 100º. The satellite design is modular with a neat physical separation between the Service Module (SVM) and Payload Module (PLM), that will make easy the design and integration phases for different payloads

(P/L) in parallel with the SVM. The subsystems (S/S) design philosophy is also modular to allow specific resources to be matched to each particular mission, most probably with different flight configurations.

For the first flight it has been selected an Earth observation mission, relying on previous experience with the following cameras at INTA: IRIS, OMC for Integral (2001), and OSIRIS NAC & WAC contributions for ESA mission Rosetta (Jan. 2004). Other astrophysics research payload and GPS signal reflectometry missions will follow. The Control Centre will be installed at INTA in Torrejón-Madrid (same for Minisat-01 and Nanosat-01). The launching date is foreseen for year 2010.

2 SYSTEM DESCRIPTION

The basic block diagram is shown in figure 1.

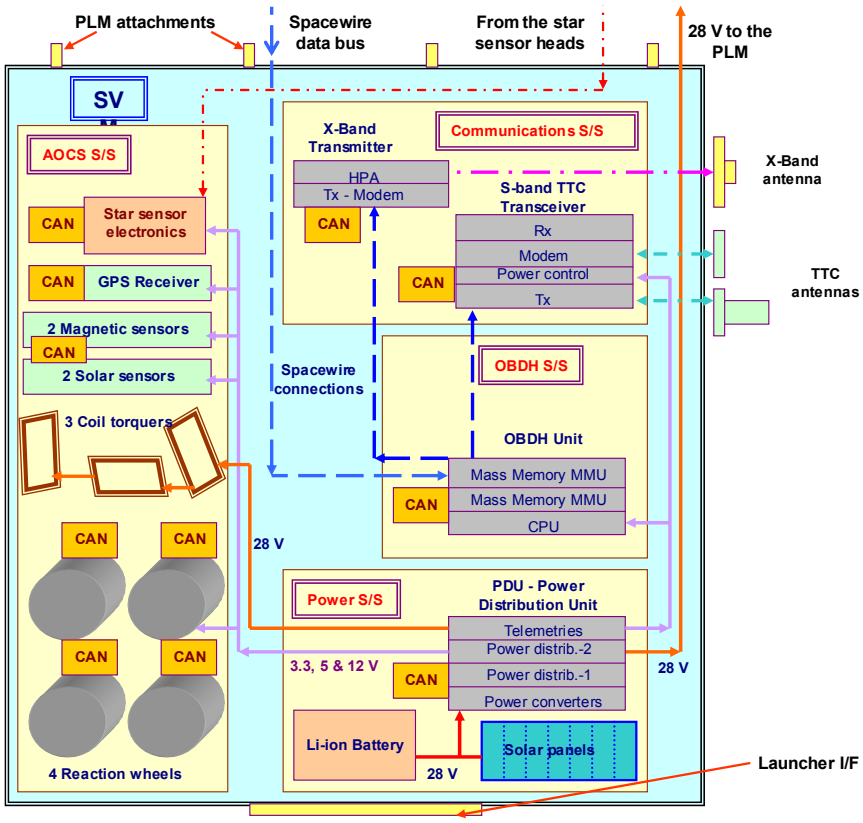


Figure 1. INTAμSAT's block diagram

Five subsystems can be identified:

- Structure and thermal subsystem. The structure design is based on a traditional concept with separated SVM and PLM, using a mix of alleviated aluminium plates for the primary structure and honeycomb panels for the secondary one and also the solar panels (in this case with carbon fibre skins). Thermal control is based on the classical passive design, using when required multilayer thermal isolations in the external sides, and radiators where required.

- Communications. This subsystem provides a S-Band channel for TM/TC (up to 2 Mbps) and a X-Band channel for high rate data download (20 to 40 Mbps). It

is based on a digital modem implemented in Actel RTAX-2000 FPGAs, mainly developed by AD Telecom in Barcelona. All the satellite antennas are developed at INTA, where we have long time experience and nice testing facilities.

- Attitude and Orbit Control subsystem (AOCS). An Earth pointing three axis stabilised concept with fast slewing capacity will be used, with A three axes magnetic sensors design for coarse pointing. It has dedicated electronics and a common CAN I/F with the CPU. Additionally, a Star Tracker and a new development Control Moment Gyro (CMG) also called Advanced Gyroscopic Actuator (AGA-150 patented in Spain) will be on board. The CMG will allow fast slewing manoeuvres up to 3º/s in a bang-bang mode.

- On Board Data Handling (OBDH). The OBDH is crucial for the mission success once in orbit and should provide the best performances available today for this kind of small satellites. The Central Processing Unit (CPU) uses Atmel microprocessor TSC-695 (ERC-32), that is a good compromise between performances and full space qualification (Radhard). This is critical, as the design is not redundant. The OBDH subsystem also comprises dedicated Remote Terminal Units (RTU) and a Mass Memory Unit, as it will be explained in section 3.

- Power Distribution Unit (PDU). The primary bus power provided by the 4 fixed solar panels will be a 28 V partially regulated bus concept (battery voltage tracker). The number of strings connected to this bus in real time will be in accordance with the required power in the satellite. The PDU will be responsible for this regulation and for the distribution of secondary regulated voltages (3.3, ±5 and ±12 V, etc.) to the SVM units that could need them.

3 DATA BUS ARCHITECTURE OVERVIEW

Two data buses can be found inside the spacecraft. The main OBDH bus is based on CANBus standard. This bus is used for low data rate transfers (<500 kbps) such as housekeeping, telecommand distribution, and for ACS sensors & actuators communications in real time with the CPU. CAN protocol provides a low cost and a high reliable bus which has proven suitable for small satellites.

For high data rates, SpaceWire standard has been selected. SpaceWire specification perfectly fulfils the requirements for INTAμSAT in terms of speed, reliability, cost and connectivity. SpaceWire will also place INTAμSAT in the same direction that ESA is sponsoring, being equipped with the latest technology in intra-satellite communications. Although this decision is challenging, due to its recent and still in development technology, SpaceWire is a guarantee of future.

As shown in figure 2, all subsystems are connected to the CAN bus. For those systems which do not have a CAN interface, a Remote Terminal Unit is provided fully functional to that particular user. The RTU is a System On Chip (SoC) design based on a 80C32 microcontroller. Apart from CAN, the RTU provides different I/O capabilities in order to fit the interface requirements for the most common low and middle range applications.

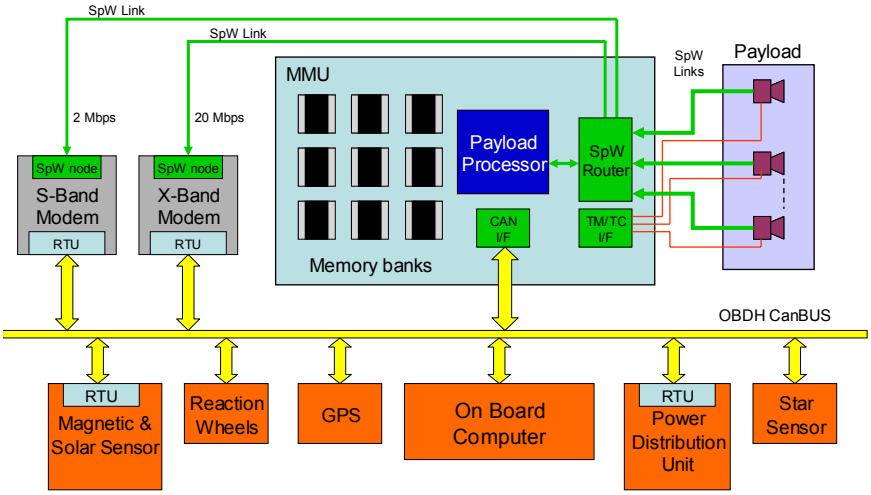


Figure 2. Data bus architecture

The MMU consists of a set of SDRAM memory banks and a Payload Processor as part of a SoC design. Together with the processor, the SoC will provide a SpW router, a low data rate interface with the payload and a CAN interface. The payload processor will handle and pre-process the payload telemetry. During the periods when there is no ground contact, the telemetry will be stored in memory. Otherwise, the stored data is driven to the S or X-band modem for its download to ground in realtime. Both modems are seen as SpW nodes in the SpW network. In the uplink, the S-Band modem sends out the TTC received information to the OBDH-CPU through the CAN Bus. When performing the downlink to ground, the data is taken either from the CPU RAM or the MMU, because both hold recorded data (Housekeeping or payload TM).

4 CONCLUSIONS

INTAμSAT-1 development is carrying out according to the foreseen schedule. Thanks to the R+D technology effort dedicated to new developments, INTA expect to have the first mission ready by year 2010. After the first evaluations, SpaceWire has shown as a good design decision, not only in performance, but also as strategic issue. SpaceWire together with other latest technologies will turn Microsat into an up-to date small satellite.

5 ACKNOWLEDGEMENTS

The Nanosat and MicroSat programmes are running thanks to several funding from the National Space Plan, either Scientific (PNE) or Technical (CDTI), and from INTA's internal budget. We would also like to thank INTA top level management, for their big support and encouraging recommendations along the past years.

6 REFERENCES

[1] M. Angulo, MR. Canchal, JM. Mi, P. de Vicente; Development and qualification of the Nanosat programme at INTA. 57th IAC Valencia 2006.

[2] M. Angulo, JM. MI, P de Vicente, M. Prieto, O. Rodríguez. E. de la Fuente, J. Palau. Development of the Microsat Programme at INTA. 6th Symposium on Small Satellites for Earth Observation. IAA-DLR Berlin, 2007.

SPACEWIRE FOR OPERATIONALLY RESPONSIVE SPACE AS PART OF TACSAT-4

Session: SpaceWire Missions & Applications

Short Paper

Paul Jaffe

Naval Research Laboratory Code 8243, 4555 Overlook Ave SW, Washington, DC 20375, USA

Greg Clifford

Silver Engineering Inc., 255 East Drive, Suite A, Melbourne, FL 32904, USA

Jeff Summers

MicroSat Systems Inc., 8130 Shaffer Parkway, Littleton, CO 80127, USA

E-mail: paul.jaffe@nrl.navy.mil, gclifford@silvereng.com, jsummers@microsatsystems.com

ABSTRACT

The rapid integration, launch, and deployment of satellites in response to emerging needs have been a focus of various organizations. This concept has been termed "Operationally Responsive Space" (ORS) by the United States Department of Defense (USDOD). One vision of ORS calls for the positioning in a depot of interchangeable satellite payloads and spacecraft buses with a common interface. Upon direction to deploy a particular mission, the appropriate payload would be selected and integrated with a bus, and the space vehicle would be launched. To support such a system, standardized hardware and software interfaces are needed between the payload and bus. For the development of ORS Bus Standards, the SpaceWire standard (ECSS-E-50-12A) has been specified as part of such a payload-bus interface for high rate data. Data interfaces can be modelled in a number of ways, such as with the OSI layer model. SpaceWire offers the appeal of standardization of physical, data, and network layers. The TacSat-4 satellite, part of the USDOD TacSat experiment series, is intended as a combination of a prototype Standardized Bus for small satellite national security missions and an example payload. This implementation includes an instance of the SpaceWire interface called out in the ORS Payload Developer's Guide. For the bus and payload SpaceWire interfaces, existing SpaceWire logic designs were used, notably the gate array core developed by NASA GSFC. This was intended as a demonstration for ORS that use of existing and freely available intellectual property can streamline design, enhance reliability, and empower instrument and payload vendors.

INTRODUCTION

Cost and schedule overruns of high-profile satellite systems in the United States [Powner et al, 2006] have led to an increased focus on the development of meaningful capabilities that are achievable relatively inexpensively and on shorter timeframes. Additionally, it is highly desirable to have the capability to deploy space assets rapidly, both in the sense of reducing the duration between call-up and launch, and in minimizing the time it takes to field new technology in orbit. Changes in international relations and the less predictable nature of threats to national and global security that have arisen in recent decades favor space assets that can be launched and utilized quickly.

In pursuing the development of an Operationally Responsive Space (ORS) system, the US Department of Defense has undertaken a multi-phase effort to establish the feasibility, requirements, architecture, and standards involved with such a system. Partnerships between industry, academia, government and research laboratories, and national and international standards organizations have been forged in an attempt to maximize the likelihood of success of the effort. Phase one, led by the Massachusetts Institute of Technology's Lincoln Laboratory, identified and analyzed the missions suitable for an ORS approach and the necessary spacecraft bus and payload capabilities [Brenizer et al, 2005].

Phase two, led by the US Air Force Research Laboratory (AFRL), has focused on ORS technology and device interface standards development. The level of modularity targeted has been at the level of individual spacecraft components, allowing for essentially unlimited combinations of hardware devices for possible satellite configurations. Capabilities have been demonstrated with AFRL's test bed and some will be employed

by TacSat-3, due for launch in late 2007. One of these technologies is SPA-S (Space Plug & play Avionics – SpaceWire) [Lyke et al, 2005]. A related AFRL effort, PnPSat, is pursuing the implementation of a satellite using solely plug-and-play components [Fronterhouse et al, 2007].

The third phase has delved deeper into the development of standards to support modularity at a larger granularity. Rather than making every satellite component modular, the space vehicle is split into only two components: the spacecraft bus and the payload. One or two different types of spacecraft buses could support the variety of a dozen or so different payloads identified in the phase one report's collection of missions appropriate for national security space using space vehicles of less than 500 kg [Brenizer et al, 2005]. It is accepted as a cost of ORS that using modular components incurs mass and other over-design penalties that are not present in tailored implementations. To assure the palatability of the resulting standards to industry, about ten companies that might be likely to build the spacecraft buses in accordance with the ORS Bus Standards [ORSBS-002, 2007] in response to a quantity buy request for proposal (RFP) were engaged early in the process by means of the Integrated Systems Engineering Team (ISET). The same group of industry and government representatives on the ISET also developed the ORS Payload Developer's Guide [ORSBS-003, 2007], intended for providers of payloads for ORS missions, and a Launch Interface Standards document [ORSBS-004, 2007].

One area that received considerable attention was that of the spacecraft bus and payload interface. Because the intention was to make the payloads interchangeable, this interface needed to accommodate a wide range of differing applications, corresponding to the phase one national security missions. The concept of operations outlines the existence of a launch depot, where launchers, spacecraft buses, and payloads are stockpiled. Upon call-up, a given payload is mated to a spacecraft bus, forming the space vehicle, and the space vehicle is integrated with the launcher for launch, the whole process perhaps being shorter than several days. Because of the multitude of digital interfaces available for space, a trade study was initiated to determine the best choice for the ORS Standards.

INTERFACE TRADE STUDY

The digital interface trade study [Jaffe 2006] dealt both with "high speed" and "low speed" interface options, in keeping with a division of electrical interfaces laid out by AFRL in phase two. These divisions were: power and ground, high speed data, low speed data, and time synchronization [Lyke et al, 2005]. For the purposes of the trade study, the boundary between "high" and "low" rate interfaces was considered to be 10 Mbit/s for mission data, excluding any overhead. Though SpaceWire could technically fall into both categories, it was considered as a "high" rate option in the trade study because of its ability to be run at speeds in the hundreds of Mbits/s. It also includes an inherent time synchronization capability with time codes that fulfils an additional interface need. Conceivably, a sufficiently easy-to-implement interface could obviate the need for a "low" rate interface.

To narrow the field of high rate interfaces under consideration, the criteria of supporting rates of at least 50 Mbits/s and of having had at least some spaceflight development heritage were applied. This left the following interfaces and their variants: IEEE-1394 (Firewire), SpaceWire, and Ethernet. Other trade studies have observed the relatively limited spaceflight heritage of Firewire and Ethernet [Walrod, Greeley 2003] [Stakem 2001] [Gwaltney, Briscoe 2006]. These particular three interfaces have been the subject of a considerable amount of comparative research [Walker 2004] [Rakow 2004] [Wolfram 2004] and developmental efforts [Wolfram 2004] [Joseph 2003] [Ivancic et al, 2005] as interfaces for space.

Because there is a dearth of direct empirical comparisons between the interfaces investigated, particularly pertaining to performance and power measures under similar circumstances, it was necessary in some cases to make partially subjective judgments or extrapolate approximations based on the available data. Different implementations also will yield varying figures for some of the qualities measured; the case judged most common was used as the baseline in these situations. The result is more of a meta-analysis than a true trade study, and it certainly not a substitute for actual lab testing.

Many factors can affect speed, throughput, and packet overhead. It may be difficult or impossible to determine optimal or actual performance without lab testing using different schemes and data types. Without controlling for various factors, comparisons may be misleading or invalid. Some such factors include: number of nodes, protocols used, the nature of the data, the desired error tolerance, latency requirements, and compressibility.

A summary of the results of the trade study are seen in Table 1. Shaded boxes indicate that for that particular metric, the interface compares favorably to the other interfaces.

Table 1. High rate data interface metrics

	Firewire	SpaceWire	Ethernet
Speed (Mbps)	100, 200, 400	2 through 600	10,100,1000
Packet Overhead	Medium	Low	Medium
Standard Maturity	High	Medium	High
Radiation Tolerance (TID/SEE)	High	High	Being tested
Space Heritage	Low	Medium	Low
Rad Tolerant Flight HW Availability	Possibly soon	Yes	>6 months
Ground Equipment Availability	Yes	Yes	Yes
Flexibility / Expandability	Medium	High	High
Power @ 50Mbps loading	~3W	~1W	~2W
Mass per node	Low	Low	Low
Complexity	High	Medium	High
Estimated monetary cost	Medium?	Medium	No basis

At the time of the conclusion of this trade study in January of 2006, SpaceWire appeared to be the most palatable choice for the high rate interface for the ORS Bus Standards. Ethernet was also attractive, but sufficiently radiation tolerant hardware to meet ORS requirements was not available at the time. (>30Krads total ionizing dose, 60MeV/(mg/cm2) linear energy threshold) It is possible in the future that this will change, and it will be worth revisiting the ORS standards periodically to determine if an update that includes Ethernet is warranted. Firewire had its appeal as well, but it would have been imprudent to select it for the standard before it had been demonstrated in flight. This may also be an interface worthy of revisiting in the future.

Practical considerations for the ORS phase three Standard Bus prototype to implemented for TacSat-4 dictated that the hardware be immediately available. The short schedule did not allow the selection of interface hardware that was not readily available. Relying on the development or future availability of new, unproven flight hardware was an unacceptable option. Additionally, NRL had used SpaceWire with STEREO; and both hardware and field programmable gate array (FPGA) designs were readily available. Ethernet for space, suitable for our orbit, (700 km x 12,050 km inclined 63.4 degrees) was not available. COTS Ethernet components are used for TacSat-1, but they were not appropriate for our HEO orbit. The availability of the 1394 chips used by NPOESS was not assured due to lack of stock and the need for a large order to compel fabrication; an option we could not pursue due to budgetary constraints. The development of the NPOESS 1394 chips (3 per node: APHY, DPHY, and Link [14]) took several years, more than three times as long as expected, and cost many millions of dollars. They are not flight proven as of this writing.

Software layer standards are a critical portion of the interface, but were largely beyond the scope of this trade study. Other efforts are striving to tackle this formidable challenge [McGuirk et al, 2007]

TACSAT-4 IMPLEMENTATION

The TacSat-4 SpaceWire link has two nodes: The Payload Data Handler (PDH) on the spacecraft bus side, and the Universal Interface Electronics (UIE) on the payload side. They are connected by a nonstandard SpaceWire cable to facilitate ease of rapid depot integration of the bus and payload [Schierlmann, Jaffe 2007]. CCSDS Space Packets and other CCSDS formats are used on top of SpaceWire.

Payload Data Handler

Beyond the payload SpaceWire connection to the UIE, the PDH also provides a second SpaceWire port for ground support equipment or a theoretical future wideband tactical downlink. In addition, it supplies the low rate interface to the payload, a 512 MB solid state data recorder and an eight channel CCSDS Channel Access Data Unit (CADU) multiplex function for the mission data S-Band downlink. Data packet routing is accommodated by a multi-channel chaining DMA controller supervised by the host processor and a Look-Up Table (LUT) that uses the packet destination identification byte to select the appropriate DMA channel for gathering the incoming packets into the required memory partition. The SpaceWire interface was implemented using the SpaceWire VHDL FPGA core developed by NASA's Goddard Spaceflight Center. This core is now widely available for

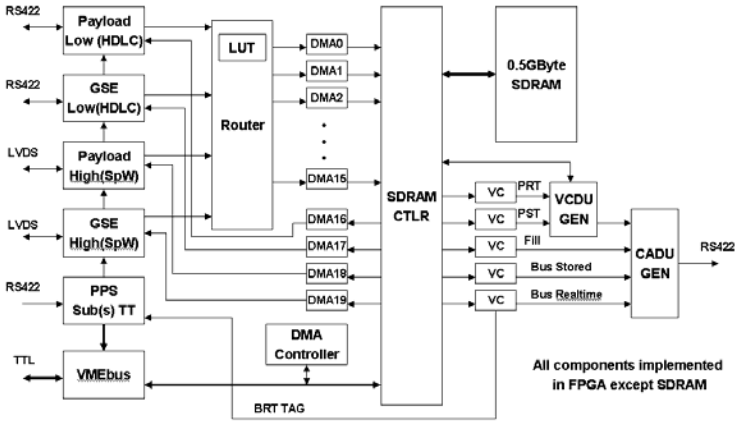


Figure 1. PDH block diagram

many users for no monetary cost. Digital logic was implemented in an Actel RTAX2000 FPGA utilizing the onboard RAM memories configured in Triple Module Redundancy (TMR) to mitigate Single Event Upset (SEI) effects. One of us (Greg Clifford) was the design engineer for the PDH.

Universal Interface Electronics

The UIE is intended as a versatile compact avionics system. Its different interfaces allow it to easily perform as a protocol translator. Depending on the desired configuration, it can have up to six SpaceWire ports. It can also offer several RS-422 ports, analog and digital I/O, power switching, and data storage. It employs the LEON3 processor in an Actel RTAX2000 FPGA and also includes a Xilinx Virtex II FPGA for mission-specific configurations. Both Goddard and Gaisler SpaceWire cores are utilized in the UIE. One of us (Jeff Summers) is the project manager for the UIE.

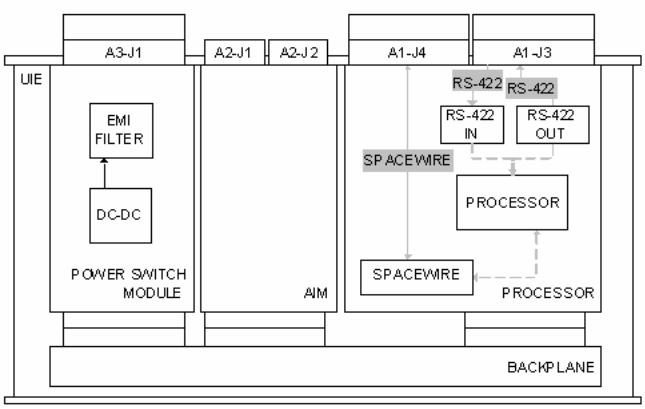


Figure 2. UIE block diagram

REFERENCES

D. A. Powner, C. Cha, N. Doherty, N. Glover, K. Malhotra, C. Phillips, K. Richey, "Geostationary Operational Environmental Satellites: Steps Remain in Incorporating Lesson Learned from Other Satellite Programs," US GAO, Washington, DC, Rep. GAO-06-993, Sep. 2006. Available: http://www.gao.gov/new.items/d06993.pdf

D. Brenizer, S. Andrews, G. Hogan, "A Standard Satellite Bus for National Security Space Missions: Phase I Analysis in Support of OSD/OFT Joint Warfighting Space Satellite Standards Efforts," MIT Lincoln Laboratory, Lexington, MA, Air Force Contract No. FA8721-05-C-0002, Mar. 2005. Available: https://projects.nrl.navy.mil/standardbus/

J. Lyke, D. Fronterhouse, S. Cannon, D. Lanza, W. Byers, "Space Plug-and-Play Avionics," presented at the 3rd Responsive Space Conference, Los Angeles, CA, Apr. 2005. Available: http://www.responsivespace.com/Papers/RS3%5CSESSION%20PAPERS%5CSESSION%205%5C5001-LYKE%20&%20FRONTERHOUSE%5C5001P.pdf

D. Fronterhouse, J. Lyke, S. Achramowicz, "Plug-and-play Satellite (PnPSat)," presented at AIAA Infotech@Aerospace, Rohnert Park, CA, May 2007.

Operationally Responsive Space (ORS) General Bus Standard (GBS), ORSBS-002/NCST-S-SB001 Revision 2, Feb. 2007. Available: https://projects.nrl.navy.mil/standardbus/

Operationally Responsive Space (ORS) Payload Developer's Guide (PDG), ORSBS-003/NCST-IDS-SB001 Revision 2, Feb. 2007. Available: https://projects.nrl.navy.mil/standardbus/

Operationally Responsive Space (ORS) Launch Vehicle Interface Standards (LVIS), ORSBS-004/NCST-IDS-SB002 Revision 2, Feb. 2007. Available: https://projects.nrl.navy.mil/standardbus/

P. Jaffe, "Trades for an Operationally Responsive Spacecraft Bus-to-Payload Digital Interface," NRL NCST Code 8243, Washington, DC, Dec. 2006.

J. Walrod, S. Greeley, "DSX Network Trade Study for networked Data Acquisition (NDAS)," PSI, Melbourne, FL, 2003.

P. H. Stakem, "Onboard LAN Technical Report," QSS Group, Inc., Greenbelt, MD May 2001. Available: http://flightlinux.gsfc.nasa.gov/docs/onboard_LAN.pdf

D. A. Gwaltney, J. M. Briscoe, "Comparison of Communication Architectures for Spacecraft Modular Avionics Systems," NASA MSFC, Huntsville, AL, Rep. NASA/TM--2006-214431, Jun. 2006. Available: http://www.tttech.com/technology/docs/protocol_comparisons/NASA_2006-06-Communication_Architectures.pdf

P. Walker, "IEEE 1394 compared with SpaceWire," 4Links Ltd., Milton Keynes, UK, 2004.

G. Rakow, "On-Board Space Flight High-Speed Protocol Comparison: Ethernet, FireWire, & SpaceWire," NASA GSFC 561, Greenbelt, MD, 2004.

K. D. Wolfram, "A New Radiation-Hardened Satellite Onboard LAN Based on IEEE Std 1394," AIAA 2004-5869, San Diego, CA, Sep. 2004.

J. Joseph, "Network Hardware for LEO Spacecraft," SpectrumAstro, Phoenix, AZ, NAS3-01147, Jun. 2003.

W. Ivancic, D. Stewart, D. Shell, L. Wood, P. Paulsen, C. Jackson, D. Hodgson, J. Northam, N. Bean, E. Miller, M. Graves, L. Kurisaki, "Secure, Network-Centric Operations of a Space-Based Asset: Cisco Router in Low Earth Orbit (CLEO) and Virtual Mission Operations Center (VMOC)," NASA GRC, Cleveland, OH, NASA/TM—2005-213556, May 2005.

D. Schierlmann, P. Jaffe, "SpaceWire Cabling in an Operationally Responsive Space Environment," NRL NCST Code 8243, Washington, DC, 2007, to be published.

P. McGuirk, G. Rakow, C. Kimmery, P. Jaffe, R. Klar, A. Bertrand, "SpaceWire Plug-and-Play (PnP)," presented at AIAA Infotech@Aerospace, Rohnert Park, CA, May 2007.

Onboard Equipment & Software

Monday 17th September

15:50 – 17:05

RAD750™ SPACEWIRE-ENABLED FLIGHT COMPUTER FOR LUNAR RECONNAISSANCE ORBITER

Session: SpaceWire Onboard Equipment and Software

Short Paper

Richard Berger, Alan Dennis, David Eckhardt, Suzanne Miller, Jeff Robertson, Dean Saridakis, Dan Stanley, Marc Vancampen

BAE Systems, 9300 Wellington Road, Manassas, Va 20110 USA

Quang Nguyen

NASA Goddard Space Flight Center, Greenbelt, Md 20771 USA

E-mail: richard.w.berger@baesystems.com, alan.dennis@baesystems.com, david.g.eckhardt@baesystems.com, suzanne.miller@baesystems.com, jeff.robertson@baesystems.com, dean.saridakis@baesystems.com, dan.stanley@baesystems.com, marc.vancampen@baesystems.com, quang.h.nguyen@nasa.gov

ABSTRACT

An additional version of the RAD750™ CompactPCI® 6U radiation hardened single board computer has been developed and delivered for use on the Lunar Reconnaissance Orbiter (LRO) mission developed by NASA Goddard Flight Center, scheduled to launch in late 2008 as the first mission in preparation for manned missions to the Lunar surface. This new variant of the RAD750 processor incorporates both a SpaceWire router and 1553 interface. The LRO mission processor architecture represents a hybrid implementation in which the SpaceWire links, 1553 bus, and PCI bus are all utilized to interconnect the flight computer and on-board instruments. The RAD750 computer includes 36 MB of radiation hardened SRAM, 4 MB of non-volatile memory, and the ability to support PROM or EEPROM in its SUROM locations. The computer also includes an additional 8 MB of radiation hardened SRAM dedicated to supporting the SpaceWire ASIC that provides a four port router through a PCI interface. The 1553 interface consists of an Actel FPGA, Aeroflex "SμMMIT DXE" ASIC, and dedicated memory. SpaceWire transport layer software was developed for the embedded microcontroller that resides on the SpaceWire ASIC using a C compiler developed by BAE Systems.

This paper discusses the definition of the new processor board configuration, definition and development of the software, and validation of the board through delivery to the mission. In addition, a preview of the next generation RAD750 processor board will be discussed, in which an improved four port SpaceWire router is incorporated directly into the third generation Power PCI bridge ASIC. This increased level of integration improves size, weight, and power of the flight computer while also adding features and increasing performance.

LUNAR RECONNAISSANCE ORBITER MISSION AND ARCHITECTURE

Lunar Reconnaissance Orbiter (LRO) is the first mission of NASA's Lunar Precursor Robotic Program (LPRP), supporting the planned return of astronauts to the Lunar surface. Scheduled to launch in October 2008, LRO will orbit the moon for one year in a 50km circular polar orbit [1] scouting possible manned landing sites, with potential extension of the mission for an additional four years. The spacecraft will deliver up to 450 Gigabits of data back to Earth each day. The spacecraft payload includes seven instruments, two of which are connected to the Command and Data Handling (C&DH) unit developed by NASA Goddard Space Flight Center via the SpaceWire network, as shown in Figure 1.

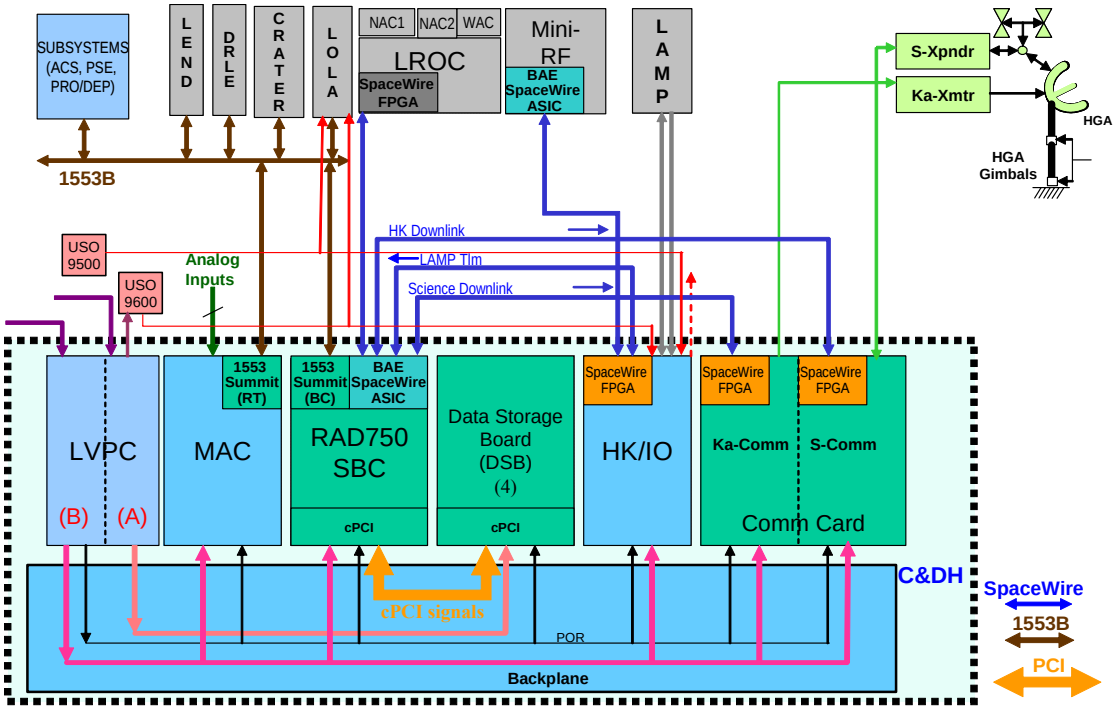


Figure 1: Lunar Reconnaissance Orbiter C&DH / Spacecraft Architecture

The Lunar Reconnaissance Orbiter Camera (LROC) instrument includes three cameras, two narrow angle with 0.5 meter resolution and one wide angle with 100 meter resolution. LROC implements the SpaceWire protocol and physical layer in a Xilinx FPGA. The LROC instrument will supply the majority of information sent back to Earth. The Mini-Radio Frequency (Mini-RF) instrument is a demonstration of S-band and X-band synthetic aperture radar, connected to the SpaceWire network through BAE Systems' SpaceWire ASIC [2]. The remaining instruments interface to the C&DH via the MIL-STD-1553 bus.

Within the C&DH unit, the RAD750 flight computer communicates with the instruments and other boards via three interfaces: a four-port SpaceWire router, a 32-bit, 33 MHz PCI bus, and a redundant MIL-STD-1553 bus. The SpaceWire router is implemented in BAE Systems' SpaceWire ASIC that is in turn connected to the RAD750 microprocessor via the PCI bus and the second generation enhanced Power PCI bridge ASIC. Both the Ka-band and S-band communications boards include SpaceWire interfaces with routers, implemented in Actel FPGAs. The LROC instrument is connected directly to one of the processor board's SpaceWire links, while the Mini-RF connects to the Housekeeping and Input/Output (HK/IO) board

that also implements a SpaceWire router using an Actel FPGA and is then routed to the processor board across the SpaceWire bus via the FPGA that implements another SpaceWire router. The 400 Gigabit LRO mass memory is implemented in synchronous DRAM that is interfaced to the RAD50 computer via the PCI bus on a custom C&DH backplane.

RAD750 SINGLE BOARD COMPUTER (SBC) ARCHITECTURE

The RAD750 flight computer is a CompactPCI 6U-220 card with two printed wiring boards (PWBs), an extension of the RAD750 CompactPCI 3U single board computer [3] that is currently flying on NASA's Deep Impact and Mars Reconnaissance Orbiter missions. The RAD750 microprocessor [4] operates at 132 MHz with a 66 MHz bus to I/O and memory, both of which are accessed through the enhanced Power PCI bridge ASIC [5]. A total of 36 MB of radiation hardened SRAM is available to the RAD750, along with 4 MB of EEPROM and 64 KB of Start-up ROM, all provided with additional bits for error correction code (ECC). The processor PWB as shown in Figure 2 is not a standard CompactPCI configuration in that there are three on-card PCI loads on the PCI bus that are made visible to the backplane. The enhanced Power PCI bridge ASIC, the SpaceWire ASIC, and the Actel FPGA programmed as a SμMMIT-PCI Interface (SPIF) for access to the 1553 bus all include PCI interfaces based on the same flight-proven reusable core developed by BAE Systems. In addition to the RAD750 microprocessor, the processor PWB includes 20 MB of SRAM memory with ECC, 8 MB of which is dedicated to the SpaceWire ASIC that includes its own external memory controller. The 1553 bus interface consists of the SPIF FPGA, the Aeroflex SμMMIT DXE ASIC, 64 KB of dedicated memory, and transformers. The back-side 160 mm memory MIB includes 24 MB of SRAM and 4MB of EEPROM, all with ECC. Connectors for the four SpaceWire links and the dual redundant 1553 are located on the single board computer's face plate, along with a connector for the RS-422 discretes and a test connector.

The board uses the standard CompactPCI 5 volt and 3.3 volt external power inputs supplied through the backplane. Separate, switchable, 5 volt input pins are provided to power the Local Memory EEPROM devices. The 2.5 volt supply required for the SRAM, ASIC and FPGA cores is generated on the board by International Rectifier linear regulators. Separate Q-Tech oscillators are employed for the main system clock (66 MHz) and the 1553 bus (24 MHz). Eight RS-422 (4 in, 4 out) discrete interfaces are provided. Power dissipation of the RAD750 single board computer has been measured at between 5.5 and 17 Watts in this configuration with calculated typical upper limit of 19.1 Watts. The mass of the RAD750 SBC is 3.5 pounds.

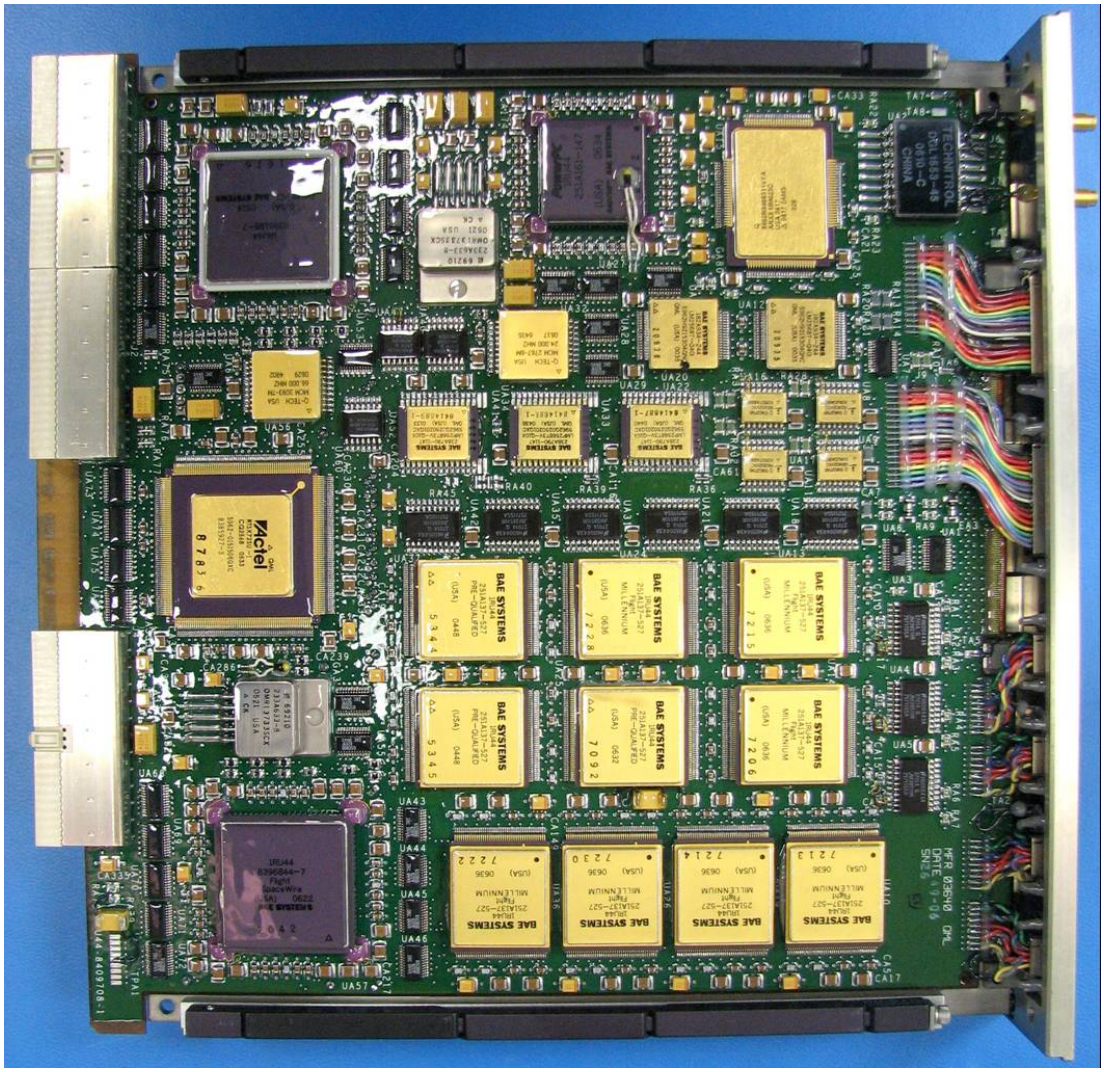


Figure 2: RAD750 Single Board Computer LRO configuration Front Side

SPACEWIRE INTERFACE IMPLEMENTATION ON THE SBC

The SpaceWire ASIC, shown in Figure 3 is based on BAE Systems' reusable core architecture. Its primary function is to perform routing of data using the SpaceWire protocol via a router with four external links and two internal connections to the On Chip Bus, the standard cross-bar switch connection medium of the reusable core architecture. In addition to the SpaceWire links and router, the ASIC is comprised of an external memory controller, dual PCI interfaces, a DMA controller, and cores for clocks and JTAG test support. Two 16 KB blocks of on-chip scratchpad memory are provided, as well as a 32-bit RISC processor called the Embedded Microcontroller (EMC) [5]. The EMC performs housekeeping functions as well as providing support for the SpaceWire router. A Phase Locked Loop (PLL) is provided for the SpaceWire link interface, which is capable of 280 MHz operation.

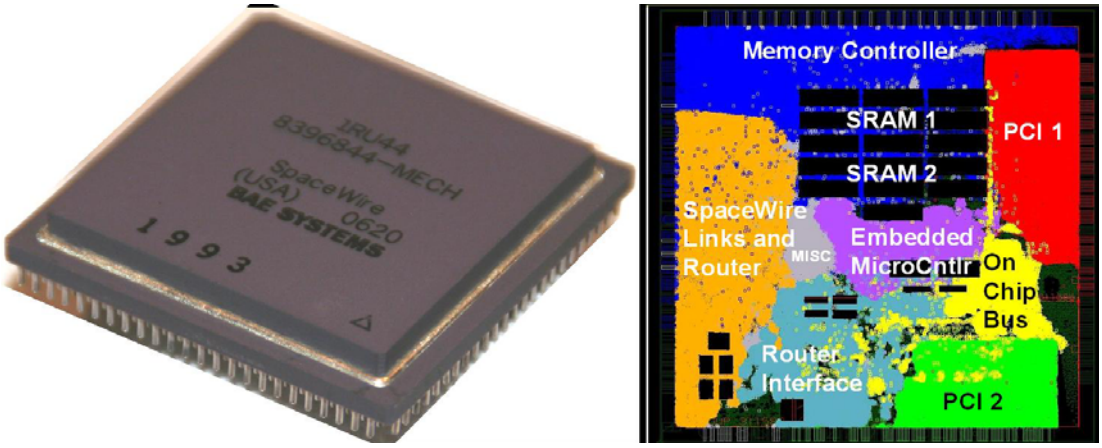


Figure 3: BAE Systems SpaceWire ASIC

In the LRO implementation, the PLL has been throttled back to only 132 MHz, supporting an aggregate system throughput of 170 Mb/s across the four ports of the SBC's SpaceWire ASIC. The gating link is the 100 Mb/s transmission between the processor and the K-band communications board. Data transmission from the LROC instrument to the processor requires 30 Mb/s throughput across a SpaceWire link. The Mini-RF instrument, which communicates with the processor via a chained SpaceWire connection through the HK/IO board, provides data at a 28.5 Mb/s rate.

SPACEWIRE SOFTWARE SUPPORT

Software drivers have been developed for the SpaceWire ASIC using C code and are included in the RAD750 Board Support Package (BSP). The BSP is designed for operation with the VxWorks (versions 5.4, 5.5, and 6.2) Real Time Operating System (RTOS). The device drivers include modules for the Programmable Interrupt Discretes (PIDs), interrupt routing, PCI and address translation, the DMA controller, the Router Interfaces (RIF) and the configuration of the SpaceWire router. The SpaceWire interface is controlled by a combination of the on-chip EMC core within the SpaceWire ASIC and the RAD750 processor across the PCI bus. The RAD750 control software also includes a network routing layer that resides above the device driver. This network layer provides queues for incoming packets. Code development for the EMC core is supported by a C compiler that BAE Systems has developed expressly for it.

Software support for the Consultive Committee for Space Data Systems (CCSDS) File Delivery Protocol (CDFP) is split between the RAD750 CPU and the EMC within the SpaceWire ASIC. The function of the software executing within the SpaceWire ASIC is to assist in CFDP download to maximize downlink throughput by sending batches of CFDP data packets known as Protocol Data Units (PDU) over the SpaceWire interface to the Ka-band communications link. Code developed for the EMC using its C compiler is used to partition large files for efficient transmission by generating PDU headers and control block "descriptors" on large (1 MB) blocks of file data. The RAD750 processor handles the supporting PDUs required for a CFDP transaction, including Metadata, End of File (EOF) in which a checksum is provided, and positive acknowledgement (ACK) or negative acknowledgement (NAK) in cases where a reliable service version of CFDP is employed.

SPACEWIRE-BASED DATA TRANSFER OPERATIONS

When data is sent to the SpaceWire ASIC on one of the SpaceWire links, it is sent through the router and on to the On Chip Bus. From there, it may either be transferred into the SRAM that is dedicated to the SpaceWire ASIC or across the PCI bus and through the enhanced Power PCI bridge ASIC into the RAD750 microprocessor's memory. A Direct Memory Access (DMA) transfer is then used to move the data into the mass storage file system.

When downlink of this data is desired, the data is extracted from mass memory with another DMA operation issued by the SpaceWire ASIC, is moved across the PCI bus and is stored in the SRAM associated with the SpaceWire ASIC. Supporting PDUs are generated by the RAD750 and the EMC within the SpaceWire ASIC issues DMA chains to packetize the data and send it across the SpaceWire link to the downlink transmitter.

Two sets of DMA engines support the transfer operations. One set of memory-to-memory DMA engines are used to transfer the data between RAD750 memory, SpaceWire memory and Mass Storage memory. A second set of DMA engines are used to transfer data between RAD750 memory or dedicated SpaceWire memory and the SpaceWire link.

INTEGRATION OF THE RAD750 COMPUTER WITH THE LRO SPACECRAFT

The RAD750 single board computer (SBC) has been successfully integrated into the LRO C&DH Engineering Test Unit (ETU). Extensive testing was performed on the single board computers at the box level prior to integration with the full spacecraft. A combination of built-in self-test (BIST) of individual components and functional tests were executed to validate all parts of the board. Functional tests included system stress tests, testing of error correction of the memory, and transfers across the PCI, 1553, and SpaceWire interfaces.

The LRO C&DH ETU has now been integrated into the FlatSat (Flat satellite) test bed at the NASA Goddard Space Flight Center and is performing well. SBC breadboard cards were delivered during 2006. SBC engineering units were delivered early in 2007 and two flight units were delivered to NASA in mid-2007. NASA is currently considering using this SBC configuration in additional missions.

The LRO RAD750 SBC is being used on NASA's Lunar Crater Observation and Sensing Satellite (LCROSS) C&DH. LCROSS is overseen by NASA Ames Research Center and is scheduled for launch in 2008. It will travel to the Moon as a co-manifested payload aboard the Saturn 5 Launch Vehicle [6] for the LRO.

NEXT GENERATION SPACEWIRE-ENABLED RAD750 SINGLE BOARD COMPUTER

A RAD750-based single board computer (SBC) is currently in development that increases the level of integration of the SpaceWire interface, as shown in Figure 4 in conceptual form. The next generation of the Power PCI bridge ASIC, designed in 150nm CMOS technology, incorporates both a SpaceWire router with four link interfaces and a dual redundant 1553 interface as internal cores. This will allow the board format to reduce to the more standard 6U-160 format with decreased power dissipation, as the SpaceWire ASIC, the SPIF FPGA, and the Aeroflex SμMMIT

ASIC with dedicated SRAM memory chips will no longer be required. On-chip memory has been extended to include two 64KB cores in addition to SRAM for the 1553 interface. In the new ASIC, performance of the SpaceWire interface increases to greater than 300 MHz.

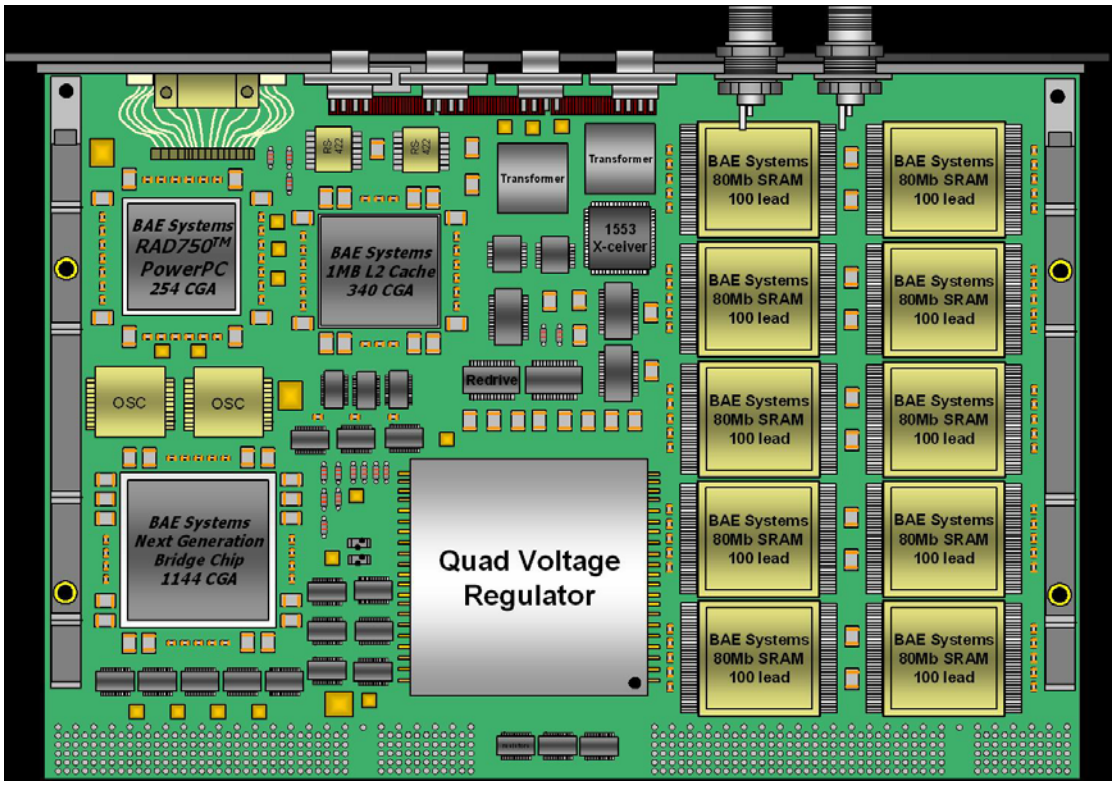


Figure 4: Next Generation RAD750 CompactPCI SBC Conceptual View (Front Side)

The RAD750 processor performance has been increased to 200 MHz, and an external 1MB Level 2 cache has been added. System SRAM capacity on the new SBC will also increase to approximately 200 MB, with new 16Mb SRAMs packaged in 80 Mb stacks replacing the current 4Mb chips that are packaged in 20Mb stacks. Startup ROM and any additional non-volatile memory will be implemented with BAE Systems' C-RAM™ memories. Prototype versions of this next generation RAD750 board are expected in 2008, with flight units planned for 2009.

SUMMARY

BAE Systems has developed a SpaceWire-enabled Single Board Computer based on the RAD750 microprocessor. Flight boards have been delivered for use in the Lunar Reconnaissance Orbiter mission, scheduled to launch in late 2008. BAE Systems continues to emphasize the integration of the SpaceWire protocol and plans to offer a next generation SpaceWire-enabled processor board by 2009.

REFERENCES

[1] C. Tooley, *"Lunar Reconnaissance Orbiter Spacecraft and Objectives"*, 2006 AIAA-Houston Annual Technical Symposium, May 2006

[2] R. Berger, et. al., *"A Radiation Hardened SpaceWire ASIC and Roadmap"*, 9th Military and Aerospace Programmable Logic Devices (MAPLD) International Conference 2006, September 26-28, 2006

[3] J. Marshall, R. Berger, *"A Processor Solution for the Second Century of Powered Space Flight"*, 19th Digital Avionics Systems Conference Proceedings, October 2000

[4] R. Berger, et. al., *"The RAD750™ - A Radiation Hardened PowerPC™ Processor for High Performance Spaceborne Applications"*, 2001 IEEE Aerospace Conference, March 2001

[5] J. Marshall, and J. Robertson, *"An Embedded Microcontroller for Spacecraft Applications"*, 2006 IEEE Aerospace Conference, March 2006

[6] G. Chin, et. al., *"Lunar Reconnaissance Orbiter Overview : The Instrument Suite and Mission"*, Space Science Reviews, May 2007, Netherlands: Springer

SpaceWire-RTC Development Suite

Session: Onboard Equipment & Software

Short Paper

Sandi Habinc

Gaisler Research AB, Första Långgatan 19, SE-41327 Göteborg, Sweden

Jørgen Ilstad

European Space Agency, Postbus 299, NL-2200 AG Noordwijk, The Netherlands

E-mail: sandi@gaisler.com, jorgen.ilstad@esa.int

ABSTRACT

The objective of the SpaceWire Remote Terminal Controller (SpaceWire-RTC) ASIC development has been to provide the European space industry with a single-chip solution for their SpaceWire and Controller Area Network (CAN) needs for the decade to come. With the ASIC development nearing its completion, the next challenge is to provide the users with a complete development suite to simplify the hardware and software design of applications using the SpaceWire-RTC ASIC.

1 SPACEWIRE REMOTE TERMINAL CONTROLLER ASIC

The SpaceWire-RTC ASIC [1] comprises the following functions:

- LEON2-FT 32-bit Fault Tolerant SPARC Processor and MEIKO FPU
- Debug Support Unit with Debug Serial Link UART
- Primary and Secondary Interrupt Controllers
- 32-bit Timers
- UART Serial Links
- 16-bit General Purpose Input Output
- 32-bit Memory Interface with EDAC support
- 64 kByte EDAC protected On-Chip Memory
- 8/16-bit FIFO Interface
- 16-bit ADC/DAC Interface
- 32-bit Timers, with external clock source and triggers
- 24-bit General Purpose Input Output with pulse generation
- CAN 2.0 Interface, with redundant interfaces
- 2x SpaceWire Link Interfaces

The ASIC has been designed using Intellectual Property (IP) [2] cores from various sources [6] [7]. The on-chip architecture is based on the AMBA AHB and APB on-chip buses that have been extended with a plug&play capability [3]. The ASIC will be manufactured by Atmel in their 0.18 um ATC18RT process and will be packaged in a MCGA 394 package. The ASIC has been designed by Gaisler Research and the prime Saab Space AB.

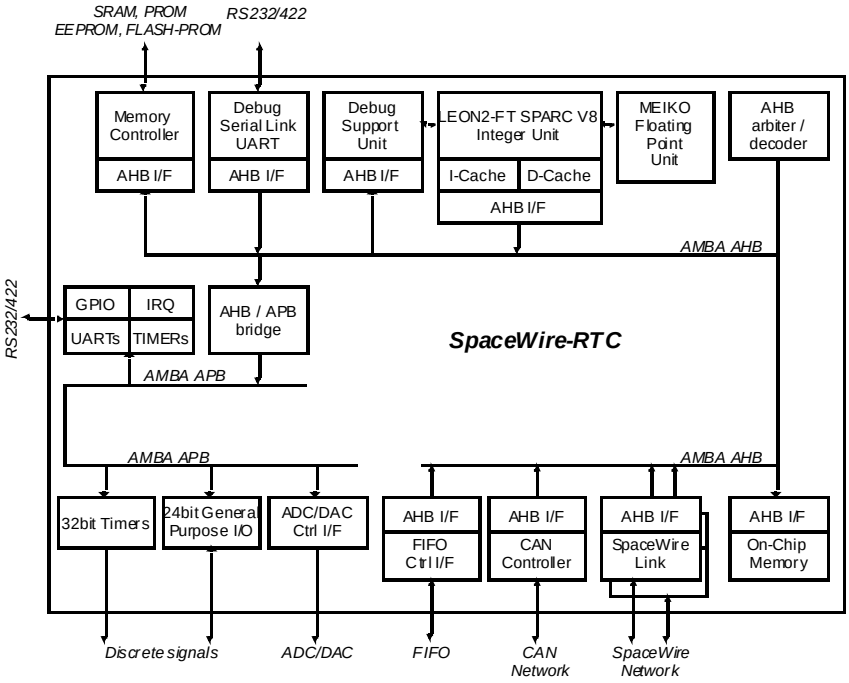


Figure 1. SpaceWire-RTC block diagram

2 SPACEWIRE-RTC DEVELOPMENT SUITE

The SpaceWire-RTC Development Suite is composed of several elements covering hardware, software and development tools. The objective is to provide the customers with a ready-to-use off-the-shelf product.

The development suite comprises the following elements:

- Hardware:
  - ASIC development board with housing
- Software:
  - LEON Bare-C Cross Compilation System (BCC)
  - RTEMS Cross Compilation System (RCC) - BSP and drivers
  - VxWorks – BSP and drivers
- Tools:
  - GRMON - debug monitor
  - GRESB - Ethernet to SpaceWire and CAN bridge
  - TSIM2 - instruction simulator with a loadable SpaceWire-RTC module

Each element is described in detail in the subsequent sections.

3 HARDWARE

The ASIC development board comprises all the memory, interfaces, transceivers and connectors that a designer would need. The board is enclosed in a housing to simplify its handling, but a can also be used without the housing and allows for expansion.

The development board comprises:

- SpaceWire-RTC ASIC
- On board Memory: SRAM / FLASH / EEPROM / PROM socket
- Interface driver circuits and connectors for UART, CAN and SPW interfaces
- 12 bit DAC (1 channel) and 12 bit ADC (4 channel input Multiplexer)
- FIFO circuits for 8/16 bit FIFO high-speed data transfer
- General Purpose Timer triggers
- On-Board Oscillators plus connectors for injecting external clocks
- DIP Switches, Push Button switches and LED indicators for configuration
- Power Supply Circuits

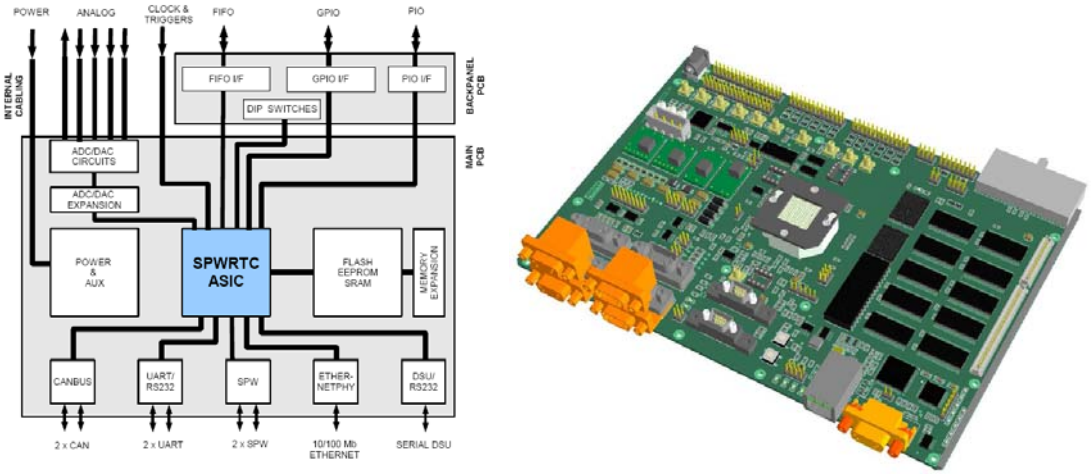


Figure 2: SpaceWire-RTC Development Suite Hardware

4 SOFTWARE

The two operating systems explicitly supported by the SpaceWire-RTC development suite are RTEMS and VxWorks. Although the two operating systems are provided under very different conditions, one being commercial and the other being for free, they have much in common concerning board support packages and drivers.

The RTEMS Cross Compilation System (RCC) [11] already includes support for the LEON2 and LEON2-FT processors. The following SpaceWire-RTC ASIC features are therefore already covered by RCC:

- LEON2-FT Integer Unit with MEIKO Floating Point Unit
- LEON2 Memory Interface
- LEON2 Debug Serial Link UART
- LEON2 UART Serial Links
- LEON2 34-bit Timer
- LEON2 16-bit General Purpose Input Output
- LEON2 Interrupt Controller

Newly developed drivers cover the following features of the ASIC:

- FIFO Interface
- CAN Interface
- SpaceWire Link Interface

The following SpaceWire-RTC ASIC features are covered by newly developed structures and functions:

- LEON2 Secondary Interrupt Controller
- On-chip Memory
- 32-bit Timers
- 24-bit General Purpose Input Output
- ADC/DAC Interface

The existing LEON2 Board Support Package (BSP) is re-used for the SpaceWire-RTC ASIC. It only required minor configuration changes to support this device. RTEMS version 4.6.5 is supported.

Similarly, VxWorks already includes supports for the LEON2 and LEON2-FT and therefore required a similar set of drivers, structures and functions to be developed for the SpaceWire-RTC ASIC. VxWorks version 5.4 is supported [12].

5 TOOLS

The tools explicitly supported by the SpaceWire-RTC development suite are GRMON, GRESB and TSIM2.

GRMON [9] is a debug tool that can communicate with the SpaceWire-RTC ASIC either via the Debug Support Link or via the RMAP protocol over the SpaceWire links. Since GRMON has been used during the development of the ASIC, the tool has been constantly adapted to support the ASIC fully. It requires therefore no updating.

The same applies to the GRESB Ethernet to SpaceWire bridge [10]. The GRESB allows communication with SpaceWire links through Ethernet, supporting functions such as SpaceWire packet routing, IP tunneling, remote debugging of LEON3 and LEON2 processors, etc. It required no update to support the SpaceWire-RTC ASIC.

TSIM2 is a simulator that can emulate LEON2 based computer systems [8]. It can also emulate user specific system-on-a-chip solutions by means of loadable modules. TSIM2 allows emulation of the LEON2 processor core with a completely user-defined memory and I/O architecture. The emulated processor core communicates with the AHB module using an interface similar to the AHB master interface in the real LEON2 VHDL model. The AHB module can then emulate the complete AHB bus and all attached units.

A TSIM2 loadable module has been developed to allow simulation of the SpaceWire-RTC ASIC. Note that only newly added functions needed to be developed in the loadable module, since TSIM2 already models the core of the LEON2 system including its peripherals.

6 CONCLUSIONS

The innovative contents of the work presented is to establish a development environment that combines software tools, hardware and test equipment in one package, with possibility for extension with user hardware or software, but still simple enough for office use (software development). This approach has been used in commercial activities and products of Gaisler Research and has now been employed in the space sector as well.

The resulting product has direct usefulness for space applications such as ESA's Bepi-Colombo programme where SpaceWire-RTC based instruments are being considered. The result will have a direct application to the development of software and hardware in projects where the SpaceWire-RTC ASIC will be used, fulfilling ESA's needs for a powerful hardware and software development suite.

The substantial improvement over existing technologies can be seen in the product integration aspect, where ease of use and turnkey solutions has been the leading goal. The benefits of programmatic improvements in terms of one complete development suite provided by a single support house should not be overlooked. It is rare that an SME has the in-depth knowledge about both hardware and software for a space component. The SpaceWire-RTC development suite puts the customer in focus. Instead of developing a board that only fits one system house, the development suite addresses multiple customers with varying needs.

7 REFERENCES

1. Jorgen Ilstad, Wahida Gasti, Peter Sinander, Sandi Habinc, "SpaceWire Remote Terminal Controller", International SpaceWire Conference 2007, Dundee (Scotland), 17-19 September 2007
2. Sandi Habinc, Tom Seeman, Jiri Gaisler, "Remote Terminal Controller Based on LEON2-FT and GRLIB", DAta Systems In Aerospace 2006, Berlin (Germany), 22–25 May 2006
3. ARM Limited, "AMBA™ Specification", 13 May 1999
4. SPARC International Inc., "The SPARC Architecture Manual - Version 8", 1992
5. Gaisler Research, "GRLIB IP Library User's Manual", 2006
6. Gaisler Research, "GRLIB IP Core User's Manual", 2006
7. Gaisler Research, "The LEON2-FT Processor User's Manual", 13 February 2007
8. Gaisler Research, "TSIM2 Simulator User's Manual", May 2007
9. Gaisler Research, "GRMON User's Manual", January 2007
10. Gaisler Research, "GRESB Ethernet/SpaceWire Bridge", August 2007
11. Gaisler Research, "RCC User's Manual", April 2006
12. Gaisler Research, "VxWorks 5.4 LEON BSP", September 2006

MODULAR ARCHITECTURE FOR ROBUST COMPUTING (MARC)

Session: Onboard Equipment & Software

Short Paper

Alan Senior & Philip Ireland

Systems Engineering & Assessment Ltd

Stuart D. Fowell & Roger Ward

SciSys UK Ltd

Dr. Omar Emam & Dr. Ben Greene

EADS Astrium Ltd

E-mail: alan.senior@sea.co.uk, philip.ireland@sea.co.uk, stuart.fowell@scisys.co.uk, roger.ward@scisys.co.uk, omar.emam@astrium.eads.net, benjamin.greene@astrium.eads.net

ABSTRACT

This paper describes the Modular Architecture for Robust Computing (MARC) concept, a modular processing system that is interconnected by a SpaceWire network. Additional processing, memory or IO modules may be added to improve system availability and/or performance. Future tests on the demonstration system will verify the FDIR system analysis and performance predictions.

The hardware architecture is closely coupled to the software aims of Generic Fault-tolerant Software Architecture using SOIS (GenFAS) software framework, which allows software builds to be allocated to available processor modules, and re-allocated in the event of a failed processor module, enabling a reduced amount of redundant processing modules through an n+m rather than 2n provision.

1 INTRODUCTION

The aim of the Modular Architecture for Robust Computing (MARC) project is to demonstrate the essential features of a heterogeneous, fault tolerant, high availability distributed avionics system based on a SpaceWire network, to a point where it is considered to be a Product that is "one-step-from-flight".

The derived MARC architecture must provide a scalable solution that can meet the demanding needs of future missions. For example, the SpaceWire network architecture must be scalable to include new functions and to provide duplicate paths to achieve the level of redundancy needed for a particular mission.

An important aspect of the demonstrator hardware is that the key components are space qualifiable parts; permitting the design to be upgraded to a fully space qualified system with minimal changes. Similarly the existing GenFAS OBSW software framework will be upgraded to a Product in accordance with space qualified software

standards (ECSS-E-40) and implementing the Spacecraft Onboard Interface Services (SOIS) communication standards as mapped onto SpaceWire.

The main applications foreseen for this architecture include missions requiring extensive distributed fault-tolerant on-board processing capabilities, such as advanced payload data processing systems and highly autonomous space exploration systems. To that end, an additional key objective is to demonstrate the applicability of the MARC architecture to the ExoMars Rover CDMS and PDHS and show that it can provide a viable solution option.

The FDIR handling features and associated test and validation shall be focused on the robustness of the SpaceWire network and its ability to reliably provide a medium for distributing both data and commands. The integrity of the commanding infrastructure is assessed in terms of guaranteed delivery and response while still complying with the stringent requirements imposed on command delivery, detection and execution latency by time critical embedded systems used for space applications. It is anticipated that the outcome of the MARC FDIR and system performance testing will show that the SpaceWire can be used alone to communicate critical data and commands and in principle replace the traditional methods of command and control distribution such as MIL-STD-1553B.

2 HARDWARE ARCHITECTURE

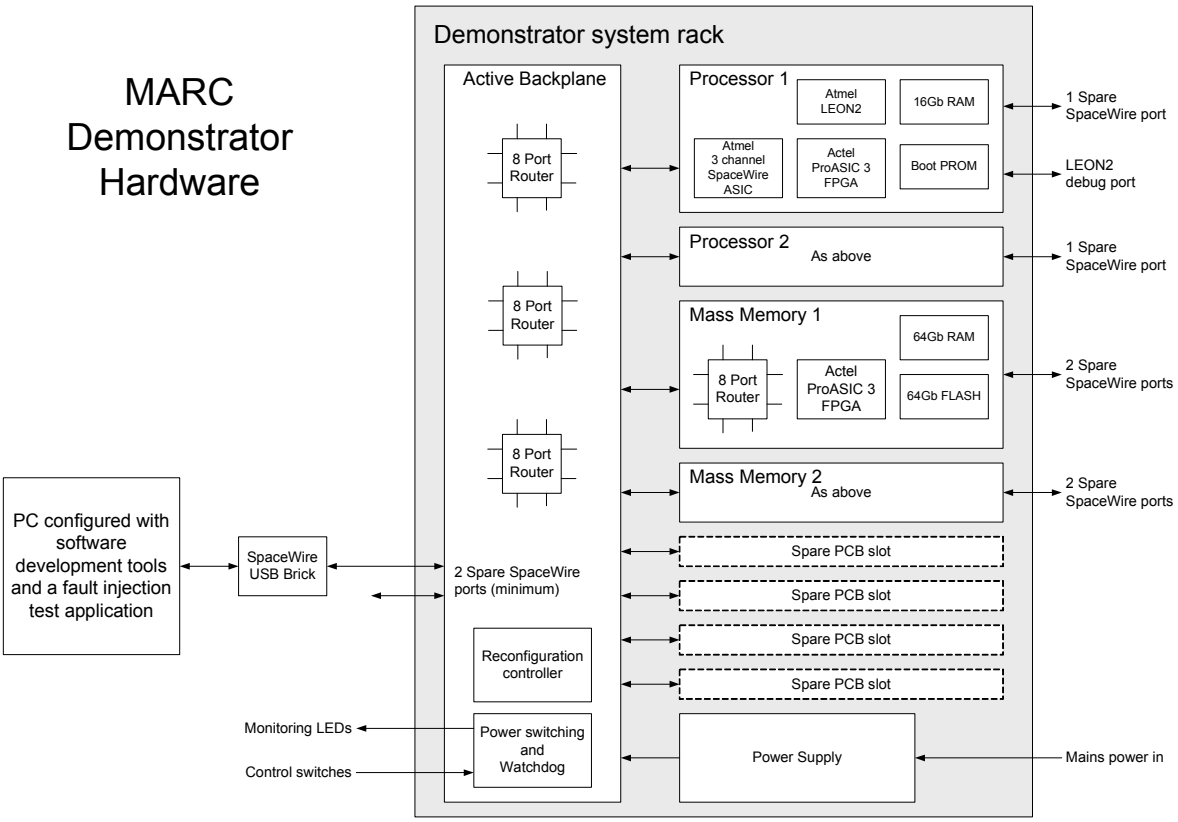


Figure 1: MARC Demonstrator Hardware Block Diagram

Figure 1 is a block diagram of the proposed MARC demonstrator system; it comprises the following principal functions:

- MARC rack containing the modules, backplane and power supply

- EGSE PC loaded with support tools software development and control of the hardware fault injection facilities
- USB brick to interface between the EGSE PC and the MARC rack

The developed modules for the demonstration rack are anticipated to be be:

- A Core Computing Module based on the LEON2 processor
- A 128Gb Mass Memory module incorporating an 8-port SpaceWire Router
- An active backplane incorporating 8-port SpaceWire Routers, power switching and a reconfiguration controller

Spare SpaceWire ports and spare module slots will be provided to permit expansion of the demonstrator at a future date. The Active Backplane incorporates three 8-port SpaceWire Routers. Of the 24 available SpaceWire ports on the backplane, 6 will be used to connect the routers together, leaving 16 available for the 8 backplane PCB slots with the remaining 2 for the spare external SpaceWire ports.

3 FAULT TOLERANT SOFTWARE

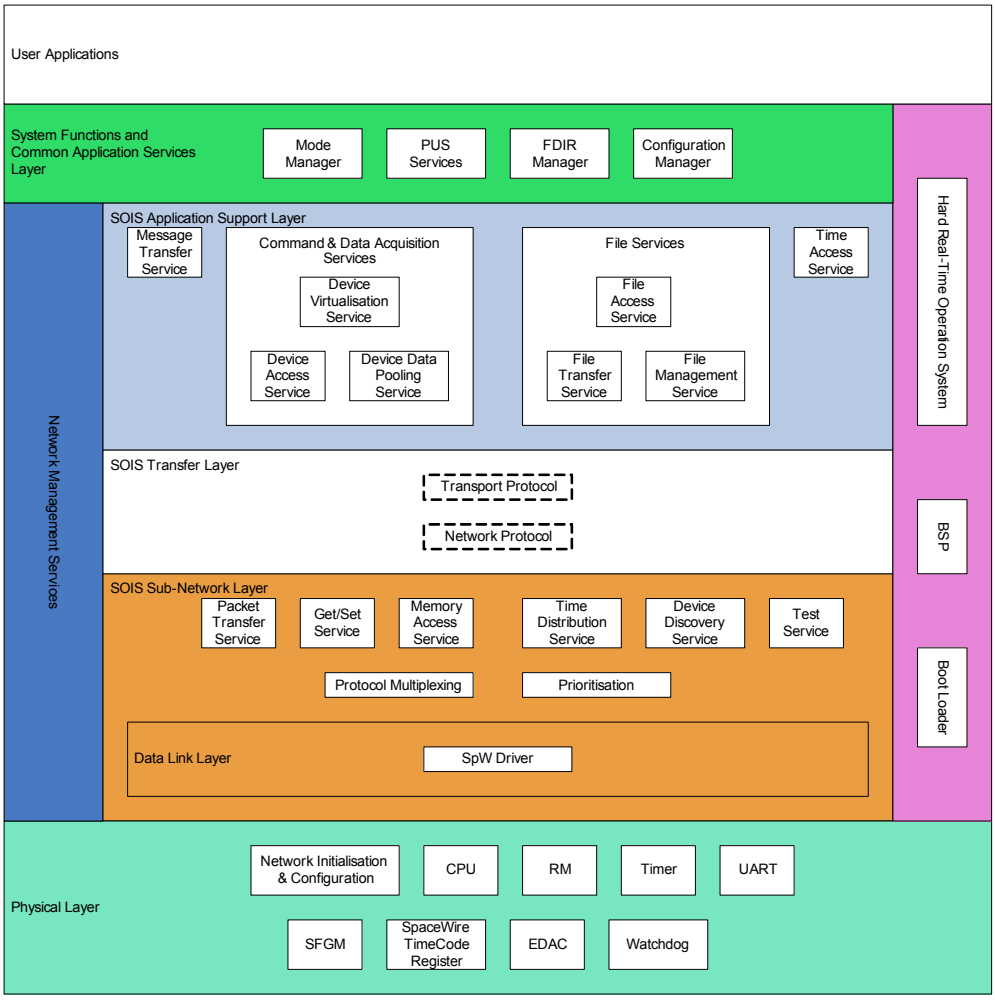


Figure 2: GenFAS Architecture

The Generic Fault-tolerant Architecture using SOIS (GenFAS) [2] is a distributed software framework for Onboard Software (OBSW) applications, implemented to ECSS-E-40 standards [3]. It is deployed upon the processing modules of the MARC demonstrator, has an architecture illustrated in figure 2 and provides the following features:

- **decentralised, distributed OBSW** – PUS Services [4] and a Data Pool are provided to OBSW applications that may be located upon any processing module in the MARC system. For example, TC packets are routed to the destination application based on APIDs and carried using the SOIS services [5] mapped onto SpaceWire protocols [6].

- **decentralised, distributed access to instruments and transducers** – all instruments and transducers are attached either directly to the SpaceWire network or via IO modules and are accessed by applications running on any processing module using the SOIS service libraries mapped onto SpaceWire protocols. Because of this an application running on any processing module has access to all instruments and transducers.

- **advanced mass memory architecture** – GenFAS implements the User libraries and System Controller of the MAGUS architecture [7]. This provides access control to different mass memory partitions held on Mass Memory modules, with the following partition types supported: File System, Packet Store, FIFO Buffer, Linked List and Raw Buffer. The Mass Memory modules are accessed using the SOIS services mapped onto the SpaceWire RMAP protocol [8].

- **fault-tolerance** – This implements an n+m redundancy approach to processing modules (n nominal processors and m redundant processors) rather than the traditional 2n approach. It relies upon and takes advantage of the other features of GenFAS – distributed software and universal access to all features using the SpaceWire network. OBSW Applications are linked together into a validated software build to be deployed together on a processing module. Each software build is stored within a build repository in the System Context area of Mass Memory. Upon start-up of the system, each software build is allocated by the GenFAS Configuration Manager to an available processing module of the appropriate type (it is assumed that there may be a small number of different types of processing modules, e.g. Leon2 and PowerPC), downloaded from Mass Memory and executed. Those processing modules not required to run the builds are held as a pool of spared. Should the processing module fail, the FDIR Manager will detect the failure and instruct the Configuration Manager to re-deploy the software build on one of the spare processing modules, i.e. a spare processing module is assigned to load the software build from Mass Memory and execute it.

4 FDIR ANALYSIS AND PERFORMANCE

The FDIR activity requires that the hardware provides facilities for the detection of faulty modules, containment of faults to a module and control over nominal and redundant functions. The details of the required level of fault monitoring are one of the main tasks in the initial phase of the project. The hardware will provide voltage level monitoring, watchdog timers and control to disable functions. The balance of software and autonomous hardware control of the system is an aspect that needs to be addressed within the project.

It is anticipated that the majority of the FDIR functionality will be implemented in software, where an adaptable decision making process can be supported. There should also be some hardware only FDIR mechanisms as it is anticipated that there will be credible failure modes that may stop processors and hence software from running. It is anticipated that there will be circumstances where it is desirable to stop all software from running and to diagnose and mitigate failures via TM/TC actions.

Two approaches will be considered for the FDIR architecture. One is a centralised architecture where the key FDIR mechanism, in particular these implemented in hardware shall be centralised and a classical prime and redundant hardware FDIR handling system is implemented with a prime and a redundant node. The other FDIR approach is a distributed one, where the FDIR handling mechanism is relocated on each node of every type. The distributed FDIR then relies on some arbitration or majority voting mechanism to reach consensus on the FDIR actions. It is anticipated that the software based FDIR will be distributed in either case.

The FDIR element of this project will be in two parts: the first is to develop an FDIR strategy which takes into account the modular architecture of the proposed system and the reliability characteristics built into the SpaceWire network standards. This work will culminate in an FDIR analysis tool which is intended to be implemented using the UML modelling language. The tool will include algorithms which can take as its input a definition of the system in terms of the type and number of nodes as well as how they are interconnected together via the SpaceWire network. The input will also specify the data paths and rates as well as constraints on commanding latency and priority. The tool will analyse the system and produce a table of FDIR actions which is used to define the conditions and paths of transitions for a state machine which shall be at the heart of the FDIR handling algorithm.

The FDIR handling algorithm which shall be implemented as part of the GenFAS software shall also be modelled in UML. The combined FDIR table generation, FDIR algorithm handling model and system definition shall be used to create a sophisticated FDIR model of the system which will enable the user to evaluate and optimise the system's architecture and performance when exposed to various scenarios of failure events.

As mentioned in the introduction, a major part of the analysis, and testing shall be focused on the reliability of the SpaceWire network when used within the MARC framework to handle the distribution of critical commands. An assessment shall be made to enable system architects to make a decision on whether such a system can rely on the SpaceWire network alone for distributing commands and their responses within a time critical embedded system.

5 REFERENCES

1. W.Gasti, “Advanced Robust Processing Architecture “ARPA” for Modular Architecture for Robust Computing “MARC””, TEC-ED/WG/2005-12, July 2006.

2. W.Gasti, “Generic Fault-tolerant Software using SOIS “GenFAS” for Modular Architecture for Robust Computing “MARC””, TEC-ED/WG/2005.14, July 2006.

3. “Software – Part 1: Principles and Requirements”, ECSS-E-40-1B, November 2003.

4. “Telemetry and Telecommand Packet Utilization”, ECSS-E-70-41A, January 2003.

5. “Spacecraft Onboard Interface Services – Informational Report”, CCSDS 850.0-G-1, Green Book, Issue 1.0, Washington, D.C, June 2007.

6. “SpaceWire – Links, Nodes Routers and Networks”, ECSS-E-50-12A, January 2003.

7. J.Stevens, D.Durrant, S.Fowell, “Generic Architectures for Future Mass Memories”, DASIA 2005 – Data Systems in Aerospace, Edinburgh, UK, May 2005.

8. “Remote Memory Access Protocol”, ECSS-E-50-11, Draft E, December 2005.

SPACE CUBE 2 - AN ONBOARD COMPUTER BASED ON SPACE CUBE ARCHITECTURE

Session: SpaceWire Onboard Equipment & Software

Short Paper

Tadayuki Takahashi, Takeshi Takashima, Seisuke Fukuda
ISAS/JAXA 3-1-1 Yoshinodai, Sagamihara, Kanagawa, Japan
Satoshi Kuboyama
IAT/JAXA, 2-1-1 Sengen, Tsukuba, Ibaraki, Japan
Masaharu Nomachi
Osaka Univ. 1-1 Machikaneyama, Toyonaka, Osaka, Japan
Yasumasa Kasaba
Tohoku Univ. Aoba-ku, Sendai, Miyagi, Japan
Takayuki Tohma
NEC Corp., 10, Nisshin-cho 1-chome, Fuchu, Tokyo, Japan
Hiroki Hihara
NEC TOSHIBA Space Systems, Ltd., 10, Nisshin-cho 1-chome, Fuchu, Tokyo, Japan
Shuichi Moriyama, and Toru Tamura
NEC Soft, Ltd., 2-22 Wakaba-cho, Kashiwazaki, Niigata, Japan

E-mail: takahasi@astro.isas.jaxa.jp, ttakeshi@stp.isas.jaxa.jp, fukuda@isas.jaxa.jp, nomachi@lns.sci.osaka-u.ac.jp, kasaba@pat.geophys.tohoku.ac.jp, t-tohma@bx.jp.nec.com, hihara.hiroki@ntspace.jp, moriyama@mxp.nes.nec.co.jp, tamura@mxm.nes.nec.co.jp

ABSTRACT

Space Cube 2 is a space-wire based onboard computer designed for future science missions in Japan. It has been developed as a successor of Space Cube 1, which is the commercial version of a minimum set of on board computer to be used on the ground. The combination of Space Cube on the ground and Space Cube 2 in space provides us with a user-friendly platform for the developments of satellites, since one could develop onboard software and flight equipments with low-cost commercial Space Cube and then easily integrate these Space-Cube2 based satellites .

INTRODUCTION

Scientific satellites often require different specifications of how the components are linked and controlled. The size and transmission speed between scientific instruments and data processors are also different from each other. Flexibility is thus the key when we investigate an architecture which can be used for a variety of scientific missions. Another issue is that mission components are often designed, built and tested by university members distributed in various places. Therefore, we need to have clearly-defined interface between the components and the bus system. Additionally, we need to prepare an easy-to-use (and inexpensive) ground support system to members, such

that most of the functionality tests including the interface can be done in the universities prior to integration.

In order to develop spacecrafts in a short time without losing reliability, Space Wire is an attractive standard. The standard interconnection between modules in both hardware and protocol levels is important to realize a system which can be easily tested. The Remote Memory Access Protocol (RMAP) for Space Wire provides a standard method for reading and writing to registers and memory among mission- and bus- components, leading to further standardization by introducing the abstraction of the access method to individual components. Once a set of middle ware is prepared based on the RMAP access, the time to write applications could be reduced significantly even with different implementations of the hardware. Another merit of the Space Wire standard is its simplicity. The logic can be readily implemented in Field Programmable Gate Arrays (FPGAs).

NET WORK BASED ARCHITECTURE AND "SPACE CUBE 1"

We have adopted the Space Wire standard for a series of small scientific satellites organized at ISAS/JAXA after some experiences in designing medium-size science missions, such as MMO (Bepi Colombo) and NeXT (X-ray astronomy) [1]. Space Wire enables us to define distributed system with a network topology. Figure 1 shows a diagram of the possible architecture to be adopted in future scientific satellites in Japan. Based on the interconnection via Space Wire, this concept could realize the modular structure and the scalability of the system and promote the re-use of resources. To make such an environment in real spacecraft, we need to prepare a standard computer for both on the ground and in space, as well as space qualified LSIs for the router and the SpW interface to be mounted in sensors and actuators.

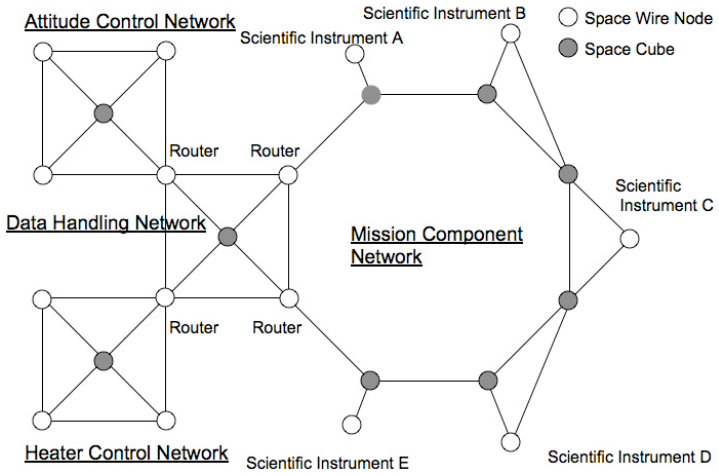


Figure 1. Architecture of the future distributed system based on Space Wire

Since the initial phase of the project, we have developed various kinds of sensor electronics with the Space Wire interface for ground-based and balloon-borne experiments [2]. Space Cube 1 was developed to control these electronics in collaboration with ISAS/JAXA and Shimafuji Electric[1]. Space Cube 1 features and

[1] Shimafuji Electric Incorporated, 8-1-15 Nishikamata, Ota, Tokyo, Japan 144-0051

various I/O interfaces include three channels of Space Wire interface with logics implemented in FPGAs, inside a very compact size, 52 x 52 x 55 mm. It includes 16 MB of Flash memory, along with audio, and a D-sub RS232 serial port. The board also offers a D-sub 15 video port. In order to have high throughput in data acquisition, Space Cube 1 uses the latest version of the TRON real time operating system, which is called the "T-Kernel"[2]. After having applied Space Cube 1 to various projects, we have learned that this kind of "standard computer" is very useful to help simulate network-based distributed systems with multiple Space Wire nodes.

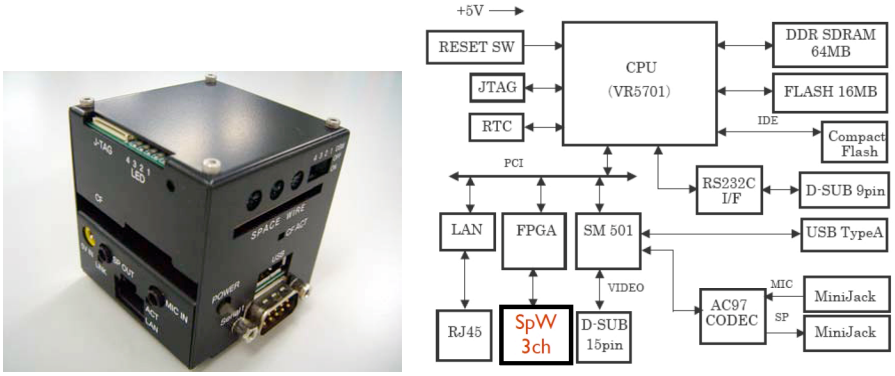


Figure 2 Photo of Space Cube 1 and the block diagram

"SPACE CUBE 2" COMPUTER

Space Cube 2 is a generic multi-mission platform developed by JAXA/ISAS and NTSpace in 2005 for space application. Space Cube 2 is based on the concept of Space Cube 1 and integrates in a single modular stack. The block diagram of Space Cube 2 is shown in Figure 2.It consists of three modules, the CPU, a Data Recorder and Power Supply modules. HR5000, which has been developed by JAXA, is the central processor chip of the system. The chip contains 64bit micro-controller based on MIPS 5kf architecture with maximum clock speed of 200 MHz With integrated peripheral devices the chip enables high-speed communication and control. The specifications of Space Cube 2 are summarized in Table. 1. We operate the CPU at 33 MHz to reduce the power. We use bus connectors to fabricate the stack structure to eliminate back planes. Furthermore, it is possible to add user modules as long as they meet the requirements of the stack module interface, mechanically, thermally and electronically. The first demonstration of Space Cube 2 (Figure 3) in space will take place in 2008 by using JAXA's technology demonstration satellite, SDS-1. In this mission, we will use Space Cube 2 as a data acquisition computer for user modules that measure acceleration of free-falling test masses.

Further developments to promote the Space Wire standard are on going. The activities include a router LSI with 15 port Space Wire and one PCI interface by using the cross bar switch technology. A small-scale Space Wire chip with simple parallel bus interface and DMA capability will also be prepared, this year.

[2] Space Cube 1 also supports Linux.

Figure 3 Space Cube 2

Table 1 Specifications of Space Cube2

CPU	HR5000 (33 MHz)
Space Wire Interface	3 ch (additional ports available)
System Memory	2 MB Flash Memory 4 MB Burst SRAM
	4 MB Asynchronus SRAM
Data Recorder Memory	1GB SDRAM 1GB Flash Memory
Size (mm)	71（W) x 220.5 (D) x 170.5 (H)
Weight	1.9 kg
Power Consumption	7 W

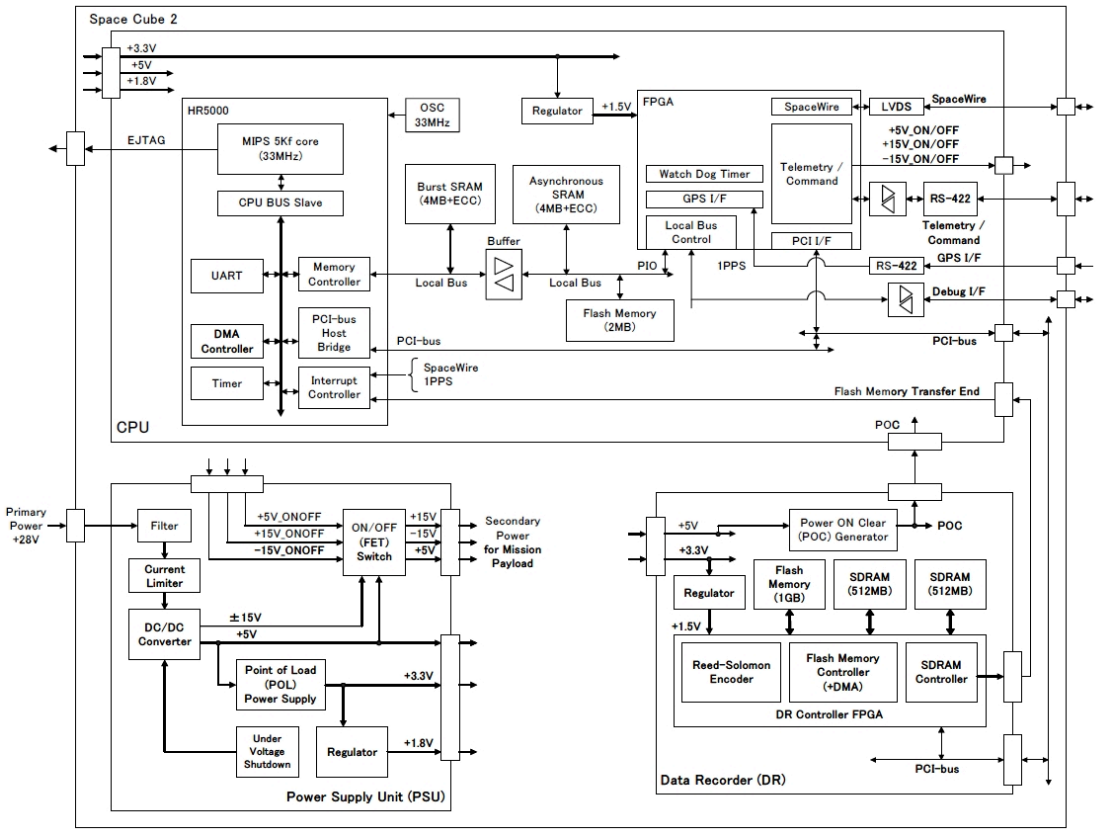


Figure 4 Block diagram of Space Cube 2

REFERENCES

1. Tadayuki Takahashi, Masaharu Nomachi, Shigeru Ishii, Yoshikatsu Kuroda, and Hiroki Hihara, “Space Wire activities in Japan for science missions”, The second SpaceWire Working Group meeting, ESA/ESTEC, November 11th 2004.

2. Takayuki Yuasa et al. “Development of a Space Wire/RMAP-based Data Acquisition Framework for Scientific Detector Applications. This conference.

Components 1

Tuesday 18th September

9:00 – 10:30

SPACEWIRE CABLING IN AN OPERATIONALLY RESPONSIVE SPACE ENVIRONMENT

Session: SpaceWire Components

Short Paper

Derek Schierlmann and Paul Jaffe
Naval Center for Space Technology, Naval Research Laboratory Code 8243, 4555 Overlook Ave SW, Washington, DC 20375, USA
E-mail: derek.schierlmann@nrl.navy.mil, paul.jaffe@nrl.navy.mil

ABSTRACT
Rapid integration, launch, and deployment of satellites in response to emerging needs have been the focus of various organizations. This concept has been termed "Operationally Responsive Space" (ORS) by the United States Department of Defense. One vision of ORS calls for the positioning in a depot of interchangeable satellite payloads and spacecraft buses with a common interface. Upon direction to deploy a particular mission, the appropriate payload would be selected and integrated to a bus, and the space vehicle would be launched. To support such a system, standardized hardware and software interfaces are needed between the payload and bus. For the development of ORS Bus Standards, SpaceWire has been specified as part of such a payload-bus interface for high rate data. Data interfaces can be modelled in a number of ways, such as with the OSI layer model. SpaceWire offers the appeal of standardization of physical, data, and network layers. However, the physical standards specified in the SpaceWire standard (ECSS-E-50-12A) regarding connectors are not ideal for the depot environment. The need for quick connection, use by minimally trained personnel, and few or no required tools combines with the usual mission considerations of cost, availability, signal integrity maintenance, and space worthiness in driving connector selection. The preliminary investigation and testing described in this paper details our recent efforts at the Naval Center for Space Technology concerning possible SpaceWire connectors for ORS. Proposed ORS test and flight cables were fabricated, tested, and analyzed using time-domain reflectometry, and compared with existing SpaceWire standard cabling.

INTRODUCTION

The commonplace cost and schedule overruns associated with space missions [Powner et al, 2006] have resulted in a focus on the development of meaningful capabilities that are achievable relatively inexpensively and on shorter timeframes. It is desirable to have the capability to deploy space assets rapidly, both in the sense of reducing durations between call-up and launch, and in minimizing the time it takes to field new technology in orbit. The US Department of Defense has undertaken a multi-phase effort to establish the feasibility, requirements, architecture, and standards involved with an Operationally Responsive Space (ORS) system.

For this approach, the space vehicle is split into two components: the spacecraft bus and the payload. One or two different types of spacecraft buses could support the variety of a dozen or so different interchangeable payloads.

The concept of operations outlines the existence of a launch depot, where launchers, spacecraft buses, and payloads are stockpiled. Upon call-up, a given payload is mated to a spacecraft bus, forming the space vehicle, and the space vehicle is integrated with the launcher for launch, the whole process perhaps being shorter than several days. SpaceWire was selected as the standard to be employed for the high rate data interface between the payload and spacecraft bus [Jaffe 2007].

PROBLEM

The cabling and connectors specified in the SpaceWire standard (ECSS-E-50-12A) are not ideal for ORS and the depot concept. The standard is insufficient for ORS for two reasons: 1) the lack of provision for an intermediate or bulkhead connection and 2) the specification of micro-D connectors, which have burdensome handling requirements.

The standard allows only for point-to-point cabling. This would force ORS busses using SpaceWire as their high-speed link to either restrict placement, routing, or access of SpaceWire devices, or perhaps necessitate the use of a hub placed on an external surface of the spacecraft. Such situations would add to the complexity and cost of an ORS bus and should be avoided.

Micro-D connectors have many advantages, but are not the best choice for a depot environment. In an ORS depot, a satellite will be created by mating a standard bus to a mission package-payload. This assembly will be undertaken by mid-level armed forces enlisted personnel whose expertise will most likely not be spacecraft engineering. The micro-D connectors called out in the SpaceWire standard are somewhat delicate and require small tools and conscientious handling. We would prefer something more ruggedized.

A SpaceWire connector for ORS requires signal integrity and impedance control fine enough to reliably support SpaceWire communication. The connector should be widely available for a reasonable cost. Such a connector should provide quick, reliable connection even when used by minimally trained personnel. It should not have torque requirements and ideally should not require tools.

We examined several commonly used spaceflight qualified connectors to determine their suitability for ORS. This investigation included the non-standard SpaceWire connectors used on the James Web Space Telescope and the International Space Station. We ultimately selected a circular 13 pin 38999 Series II connector similar the one chosen for the European Drawer Rack (EDR) of the ISS [Lanza 2002]. EDR chose a D38999/ 20FB35SN, however we chose a D38999/40FB35SN (447HS166M11-10-4 backshell) for the bulkhead connector and D38999/46FB35PN (same backshell) for the cable side.

The design of the TacSat-4 spacecraft dictates that the SpaceWire cable has two bulkhead connectors for a total of 6 connectors and 3 segments. The pin assignments for the connector were chosen graphically in an attempt to make the conductor configuration for each pair as uniform as possible and attempting to align **E** and **H** fields. The overall design of the cable is shown in Figure 1.

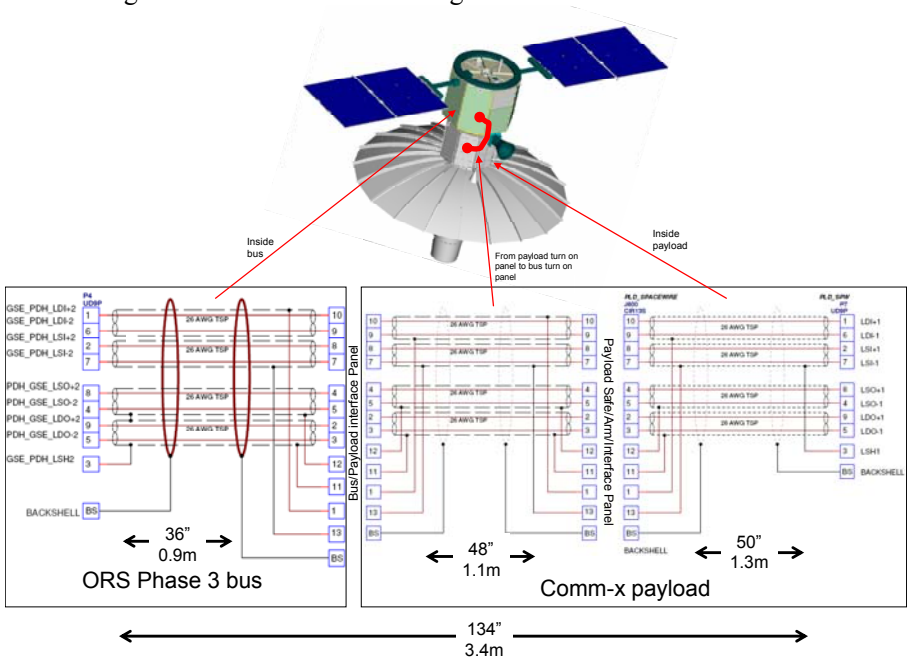


Figure 1: TacSat-4 SpaceWire cable drawing

The cable tested was the flight SpaceWire cable to be used on TacSat-4. The cable was hand fabricated by NRL's harness technicians in compliance with engineering drawings (2047249) developed by NASA GSFC for the James Webb Space Telescope. The cable assembly is made from Gore-Tex 26AWG (GSC-05-82730-00) SpaceWire cable (W. L. Gore & Associates GmbH). The Gore-Tex wire used to make this cable assembly has undergone extensive qualification and testing to prove its suitability for SpaceWire applications [Mueller 2006].

METHODOLOGY

NRL performed testing to evaluate the deviation from SpaceWire specifications. These tests will be compared to two intact, off-the-shelf, SpaceWire cables (0.5 m and 2 m). The testing included: evaluating the differential impedance (via time-domain reflectometry), comparing data and strobe waveforms, and bit error rate (BER) measurements. TDR traces and scope waveforms were analyzed qualitatively for pass or fail. Successfully passing TDRs traces were close to 100 ohms and had few discontinuities. Oscilloscope waveforms needed sharp, monotonic edges with minimal ringing to pass. An engineering team of SW, hardware, RF, and I&T engineers decided that bit error rate of less than 10^{-7} was required for BER test success. Operational testing (waveform capture and BER) testing was be done with a STAR-Dundee SpaceWire-USB

brick in loopback mode. These operational tests were performed using slightly modified versions of the scripts that ship with the STAR-Dundee brick.

All scope probing was done on a Tex TDS644A with a Tex P6246 400MHz differential probe. This probe has input capacitance of less than 1pF and an input resistance of approximately 200kOhms. The probes were connected to the system using a test board developed for this program. The board is known to introduce a discontinuity, but since test results are compared against a baseline cable, the effect can be subtracted. TDR testing was done using a Tektronix DSA8200 with a 80E04 differential TDR head. Impedance correction was performed using IConnect (80SICMX) software. The TDR analysis also required the use of the SpaceWire test board. Additional testing of the cabling will be achieved during normal flight vehicle qualification. These tests include, but are not limited to shock, vibe, EMI, and TVAC.

RESULTS AND ANALYSIS

Figure 2: Picture showing the flight (left, in unmated configuration) and baseline cables (middle is 2m DVI heritage cable and right is 0.5m 'blue' cable).

The cables shown in Figure 2 were tested as described above. The results and analysis are presented below and grouped according to test.

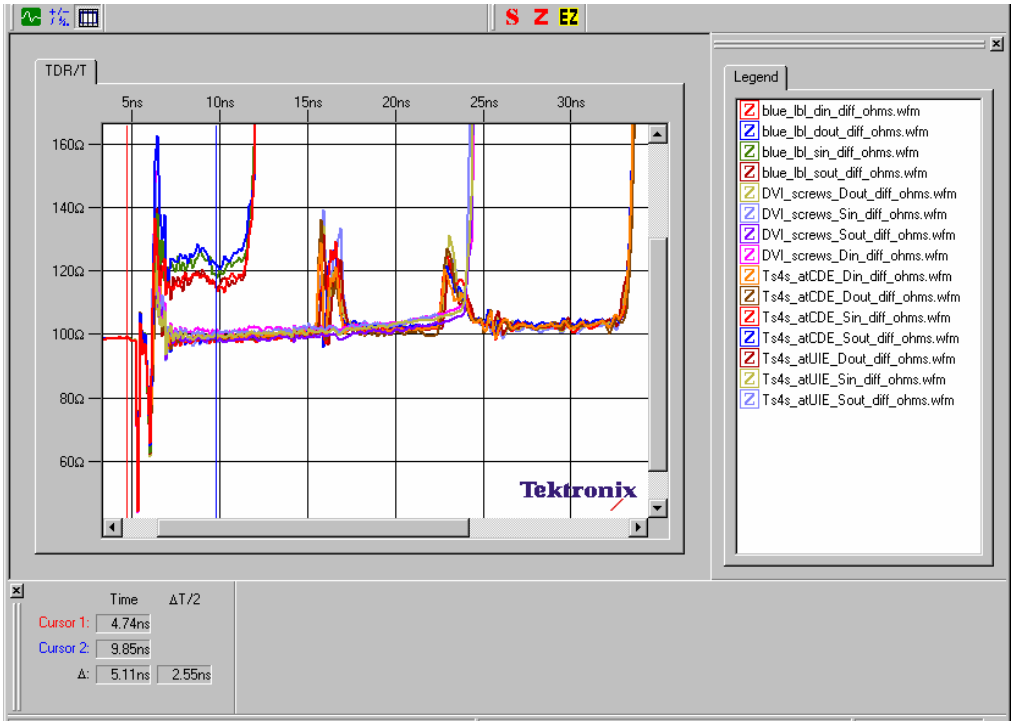


Figure 3: Raw (uncorrected) TDR traces for both reference cables and the TacSat-4 SpaceWire cable

As shown in Figure 3, the 38999 Series II connector has an impedance ranging from 98Ω to 140Ω. The discontinuities are approximately 2 ns wide. An impedance discontinuity of up to 40Ω would seem to be a failure, but the 38999 Series II impedance control is as well as and perhaps better than the standard micro-D

(60Ω to 180Ω for 1.5ns, not shown). It is clear that this connector is not perfect, but it appears to be good enough to support some SpaceWire communications.

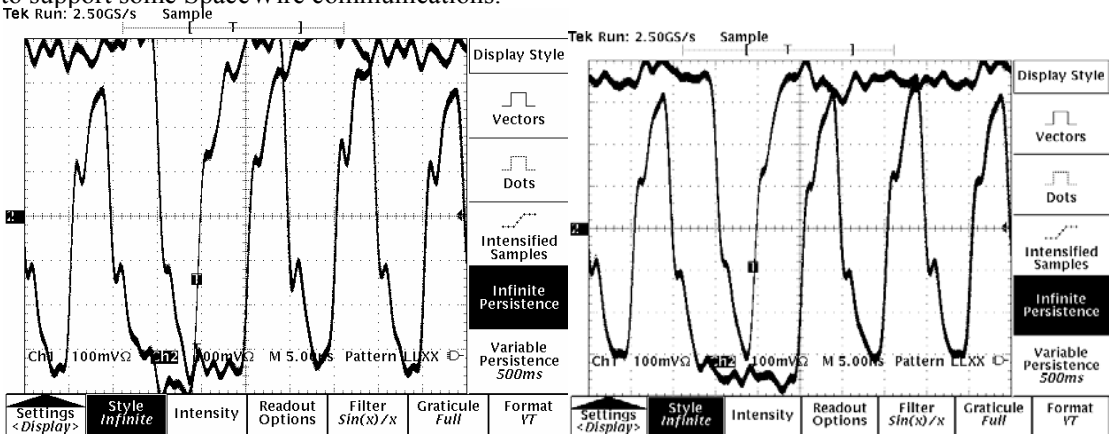


Figure 4: Comparing the reference cable (left) to the TS4 cable (right) Note that waveforms were *NOT* taken simultaneously.

Figure 4 shows typical scope traces comparing the flight and reference cables. As is seen in the image, there was no appreciable difference between scope traces taken of data transmitted on the flight or reference cables when evaluated using the criteria stated above. Although the scope traces show non-monotic edges and some ringing, these effects appear in all traces and may be attributable to the test set-up. The traces taken on the TacSat-4 cables show less ringing and are slightly attenuated when compared to the reference cable. This attenuation is most likely attributed to the difference in length between the TacSat-4 (134"/3.4m) and the reference cable (18"/0.5m) and actually improves the signal slightly.

The bit error rate test used in this study was designed to be a pass or fail test; meaning that there is no feedback from the test other than to say that it passed at the specified speed. Both reference cables and the TacSat-4 cable passed when running at the maximum rate of the SpaceWire device (136 Mbit/s). The 2m DVI heritage reference cable passed BERT testing, but would not run when inline with the SpaceWire test board. In bit error rate testing, there was no difference between the cables.

All testing performed to date has shown that 38999 Series II connector performs reliably in SpaceWire applications up to and above 100 Mbit/s. Furthermore, the concept of a depot-style multiple-segment SpaceWire cable is valid and we will continue to design ORS busses with SpaceWire bulkhead connectors.

FURTHER WORK

It is important to note that no effort was made to characterize the insertion and reflection losses, skew, crosstalk or eye pattern as performed in previous studies [Mueller 2006]. Such work would be useful in further characterizing the proposed cable solution.

The drivers used in this study were speed limited. It would be interesting to see how this cable performs at higher bit rates. How much will the quirks of this cable assembly affect its performance? This study suggests that the cable assembly should perform as good or better at higher rates, but where is the limit?

REFERENCES

S. Allen, "SpaceWire Physical Layer Issues," 2006 MAPLD International Conference, Washington, DC, September 2006.

D. Brooks, "Differential Impedance: What's the Difference?" Printed Circuit Design, August 1998.

T. Heikkila, "Differential Impedance Measurements with the Tektronix 8000B Series Instruments," Available http://www.tek.com/Measurement/cgi-bin/framed.pl?Document=/Measurement/App_Notes/85_16644/eng/&FrameSet=oscilloscopes

P. Jaffe, "SpaceWire Cabling in an Operationally Responsive Space Environment," NRL NCST Code 8243, Washington, DC, 2007, to be published.

H. Johnson, G. Martin, *High-Speed Digital Design: A Handbook of Black Magic*, New Jersey: Prentice Hall PTR, 1993.

H. Johnson, [Online Papers] Available http://www.sigcon.com/, 1998-2005.

P. Lanza, "EDR HSSL Protocol and Implementation," EDR-TN-AI0014, 2002.

J. W. L. Mueller, "Design Challenges of an Advanced SpaceWire Assembly for High Speed Inter-Unit Data Link," 2006 MAPLD International Conference, Washington, D.C. September 2006.

Operationally Responsive Space (ORS) General Bus Standard (GBS), ORSBS-002/NCST-S-SB001 Revision 2, Feb. 2007. Available: https://projects.nrl.navy.mil/standardbus/

Operationally Responsive Space (ORS) Payload Developer's Guide (PDG), ORSBS-003/NCST-IDS-SB001 Revision 2, Feb. 2007. Available: https://projects.nrl.navy.mil/standardbus/

C. R. Paul, *Introduction to Electromagnetic Compatibility*, John Wiley & Sons, 1992.

D. A. Powner, C. Cha, N. Doherty, N. Glover, K. Malhotra, C. Phillips, K. Richey, "Geostationary Operational Environmental Satellites: Steps Remain in Incorporating Lesson Learned from Other Satellite Programs," US GAO, Washington, DC, Rep. GAO-06-993, Sep. 2006. Available: http://www.gao.gov/new.items/d06993.pdf

D. Brenizer, S. Andrews, G. Hogan,, "A Standard Satellite Bus for National Security Space Missions: Phase I Analysis in Support of OSD/OFT Joint Warfighting Space Satellite Standards Efforts," MIT Lincoln Laboratory, Lexington, MA, Air Force Contract No. FA8721-05-C-0002, Mar. 2005. Available: https://projects.nrl.navy.mil/standardbus/

M. N. O. Sadiku, *Elements of Electromagnetics 2nd Edition*, Saunders College Publishing, 1994.

DESIGN AND IMPLEMENTATION OF SYNTHESIZABLE SPACEWIRE CORES*

Session: SpaceWire Components

Short Paper

P. Aguilar-Jiménez, V. López, S. Sánchez, M. Prieto, D. Meziat

Space Research Group. Dpto. Automática. Universidad de Alcalá

E-mail: paguilarj@srg.aut.uah.es, mpm@srg.aut.uah.es, vlopezalvarez@gmail.com, chan@srg.aut.uah.es, meziat@aut.uah.es

ABSTRACT

The Space Research Group of the University of Alcalá is developing a library of IP cores to provide main space systems and subsystem SoC implementation. One of these is an ECSS-E-50-12A [1] based SpaceWire core covering the standard specification in two different approaches: a basic codec and a router core. Our goal is to achieve a technology independent synthesizable core optimized for timing and area.

In this paper, a brief description of the developed core is presented. After improving the synthesis results adding internal FPGAs, the resulting programming file is tested on different devices to check the post layout simulation results.

1 INTRODUCTION

Into the research areas of the Space Research Group (SRG) one of them is dedicated to de development of synthesizable IP cores of main spacecraft systems and subsystems. Some of these cores are based in the SpaceWire standard [1], giving us a deep knowledge of the standard and the technologies involved in the development of IP cores for space applications.

The SpaceWire Codec and Router here presented are intended to be on board of future missions and project at which SRG is currently involved.

2 SPACEWIRE CODEC

The SpaceWire Codec [1][2] has been developed accordingly to the ECSS-E50-12A Standard and it is planned to provide a maximum data rate close to 400 Mb/s [1]. Its design constraints are synthesizable, device independent and optimized for area and time resources what makes analysis a key role in the process leading to the working core. Its design is based in the well known block diagram shown in [2]:

* This work has been supported by the CICYT (grant ESP2005-07290-C02-02)

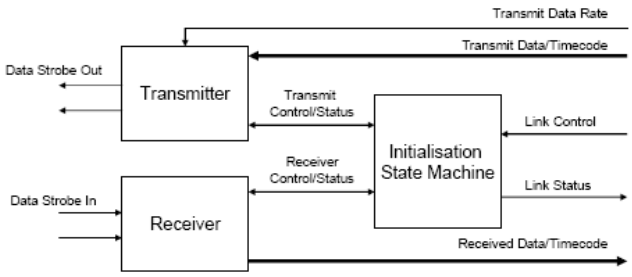


Figure 1. SpaceWire Codec block diagram.

Our approach is based in state machines for all three blocks. At the transmitter and receiver these state machines performs the serial to parallel conversion, flagging the different reception events to the main state machine and attending the transmit requirements issued by the reception buffer management and the host interface. Transmitter and receiver synchronization is based in the scheme described in [1] and in the size of the receive buffer, 56 N-chars.

One of the first concerns is that of dealing with high clock frequencies to provide as high data rates as possible. That is an affordable issue considering that the data signal of the standard's DS encoding holds one bit per each state of the recovered clock signal at the receiver [2], as shown in the next figure:

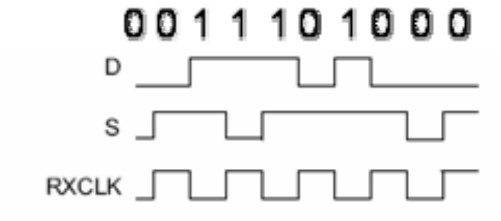


Figure 2. Receiver clock recovery.

Our proposal is base in the use of Double Data Rate for both transmitter and receiver. Using DDR registers give us the ability to manage clock frequencies half the desired data rate, with a 200 MHz top clock limit in the design. The cost of this advantage is an increased complexity of the design, working with two data flows, one for the even bits (b0, b2, b4, b6, b8) and another for odd ones (b1, b3, b5, b7, b9).

Strobe sequence generation at the transmitter is another challenging issue when optimizing design for area and timing. Our proposal is based in processing data stream as two independent data flows. Working with even and odd sequences, as stated before, strobe sequence is generated in two (odd and even) sequences too, using a quite simple algorithm based in the XOR logical properties and the receiver's XORing clock recovery scheme [1]. Taking into consideration that initial state for data and strobe signals is low ('0') and that first data bit transmitted after reset is a zero parity bit causing a high state transition on the strobe signal [1], it is clear that even bits are transmitted with clock's high state and odd bits with clock's low state. In that case, strobe even and odd sequences generation is as simple as

$$S_{par} = D_{par} \oplus 1 = \overline{D_{par}}$$
$$S_{impar} = D_{impar} \oplus 0 = D_{impar}$$

Both sequences are combined by a DDR register leading to the transmitted strobe signals. Additional delays due to the combinational logic are equalized using pipelining registers at the cost of one or two latency cycles at first bit transmission.

3 SPACEWIRE ROUTER

The SpaceWire Router implemented [1][3] is planned as a switching matrix to interconnect up to 224 SpaceWire codecs [1] rising up a SpaceWire network made of nodes (codec pairs) and at least one SpaceWire router, although complex topologies can easily grow up adding several routing switches. In its basic approach the router performs logical addressing and wormhole routing, maintaining a four nodes routing matrix as a constant VHDL array structure.

It is designed as a separate entity from the codec, having both a matching interface to easily connect. Router interface have a generics based interface to be configured for a different number of nodes, being three the minimum permitted. An internal resource was configured too accordingly to the number of nodes connected to.

In case of a desired output port is busy the router will stop reading packets from the transmitting node, avoiding its reception buffer gets empty and maintaining the flow control in the link. This simplified router behaves accordingly to the ECSS-E50-12A Standard. Although the router can support up to 224 addresses in logical addressing and 32 addresses in path addressing, the required number of pad for such configurations makes them a study case. Our router supports up to eight nodes in a star topology network without connecting to other routers.

Special care to crosstalk between adjacent paths belonging to different channels inside the FPGA was taken to ensure all time constraints are met in the design.

4 SYSTEM DEVELOPMENT AND TESTING

Simulations, both functional and post layout, and synthesis have been made using Mentor Graphics Corporation tools. For simulation Modelsim 6.0 was the application selected while for synthesis it was Leonardo Spectrum 2005. Post layout simulations are performed including VITAL libraries provided by Xilinx and Actel. Glitch handling differs at both, while in Xilinx VitalPathDelay arguments Mode, Xon and MsgOn are set to VitalTransport, true and false, in Actel are set to OnEvent, false and true; to avoid annoying To avoid those VitalGlitch warning messages simulations are run with the "+no_glitch_msg" optional argument at the vsim command.

The map and layout tools are device dependant. The initial codec prototype was tested in a Spartan IIE using Xilinx ISE tools to generate and load the bitstream file into the FPGA. That test showed the codec was right designed and it consumed a total gate count of 4952. Compared to other cores [4] the result is considered good taking into consideration it is an initial prototype. The next steps were synthesizing the codec for Actel proasic3 A3P250 using Designer, and provide it with connectors to test the equipment against a Star Dundee SpaceWire PCI2 board. Following this test a Spacewire-USB Brick was used to connect both cards, and finally the Star Dundee SpaceWire-USB Brick was replaced by our router implementation.

Although the design is intended as device independent, some details like system clock signals generation and distribution are treated as device dependant using FPGA resources

such as PLL and DLL. That gives a better performance in such critical areas for optimization and can be designed as external modules to the main design.

5 CONCLUSIONS

The implementation of synthesizable SpaceWire components based in the ECSS-E50-12A standard should take into account technology dependant issues such as design constraints, time and area optimization and accurate management of system clock frequencies. These issues will be common to any device selected, representing a physical limit to the standard.

When optimizing for particular details, like clock management or memory resources, device selected resources will play a key role, being the design tailored for a specific FPGA when looking for an improvement in the core performance.

It is necessary the support of qualified test equipment to prove the accuracy of our prototype. StarDundee commercial Spacewire equipment provides a good platform to perform the final stage of the test procedure.

6 FUTURE WORKS

With the 2010 horizon for the INTAμSAT-1 mission, additional works are being done to improve connectivity to other subsystems and payload components. One of these works is focused in the development of host interfaces for the Spacewire codec, having special attention to RMAP [5]. Data transfers and bus interfaces are also improvement goals for the codec, where DMA and AMBA are well known.

Concerning the router, a prototype card is the next step to further test its functionality between our own codec and the Spacewire PCI2 card. Improvements should be done to support both addressing schemes, giving chance to router interconnection and different and deeper network topologies. Redundant links between router and the associated management of the routing tables is also an improvement goal.

7 REFERENCES

[1] S.M. Parkes et al, “SpaceWire – Links, Nodes, Routers and Networks”, European Cooperation for Space Standardization, Standard No. ECSS-E50-12A, Issue1, January 2003.

[2] C. McClements, S. Parkes and A. Leon, “The SpaceWire CODEC”, International SpaceWire Seminar, ESTEC Noordwijk, The Netherlands, November 2003.

[3] S.M. Parkes, C. McClements, G. Kempf, S. Fischer and A. Leon, “SpaceWire Router,” International SpaceWire Seminar, ESTEC Noordwijk, The Netherlands, November 2003.

[4] Gaisler Research, “SpaceWire Codec with RMAP. GRSPW / GRSPW-FT CompanionCore Data Sheet”, Gaisler Research AB, September 2006, Version 1.0.0.

[5] S.M. Parkes and C. McClements, “SpaceWire Remote Memory Access Protocol”, submitted to DASIA 2005.

The GRSPW SpaceWire Codec IP Core and its Application

Session: Components

Short Paper

Sandi Habinc, Marko Isomäki, Jiri Gaisler

Gaisler Research AB, Första Långgatan 19, SE-41327 Göteborg, Sweden

E-mail: sandi@gaisler.com, marko@gaisler.com, jiri@gaisler.com

ABSTRACT

SpaceWire based devices are moving rapidly away from just being generic memory mapped interface chips to becoming complex processor-based system-on-a-chip solutions. The function of the SpaceWire link is changing from one being used for high-speed data transfers, to one also being used for remote control and debugging.

There are several key technologies that enable the development of powerful system-on-a-chip designs featuring SpaceWire links and embedded processors. By mastering all of them one can produce new exciting and advanced products for onboard application. This paper will present these key technologies and how they have been utilized in developing an efficient SpaceWire coded that is used in a large set of complex system-on-a-chip designs aimed for the space market.

1 ENABLING TECHNOLOGIES

With the introduction of the LEON processor family the space community has gained access to a set of powerful 32-bit SPARC V8 [2] based processors that can be targeted towards different FPGA and ASIC technologies. This is the single most important technology required for developing embedded fault-tolerant processing applications.

The adoption of the Advanced Microcontroller Bus Architecture (AMBA) [1] as the on-chip bus fabric used in ESA developments was made simultaneously with the development of the second LEON processor. The standardization has resulted in a multitude of synthesizable cores that are integrated in system-on-a-chip designs.

The SpaceWire Remote Memory Access Protocol (RMAP) [4] allows the implementation of a standardized communication method for reading and writing to remote memory and registers. This eliminates the plethora of existing ad hoc protocols that have been used in previous developments, allowing designers to focus their efforts on a single re-usable solution that can be transferred between projects.

The combination of the RMAP protocol and the Debug Support Unit (DSU), the latter has been developed for the LEON processor family [6], has opened up the possibility for fast and efficient uploading and remote debugging of software through existing SpaceWire networks, without the penalty of an additional low-speed control bus.

2 SPACEWIRE CODEC IP CORE

Although the enabling technologies are in place, their efficient implementation is of paramount importance to develop successful products. Gaisler Research has always developed building blocks required for system-on-a-chip design keeping efficiency and compatibility in mind.

The key building block is the GRSPW SpaceWire Codec IP core [6] that has been developed from scratch by Gaisler Research. Contrary to other codec developments, the GRSPW has been designed taking into account from the beginning the previously presented enabling technologies. In addition to implementing the SpaceWire link protocol, the design had to simultaneously meet the constraints posed by the AMBA Advanced High-performance Bus (AHB), providing plug&play capability, full RMAP functionality, tool and technology portability, and finally seamless interaction with the LEON3 and LEON2 processors through the DSU interface.

By taking this overall approach several functions could be combined, for example the data buffering in the front-end codec and direct memory access (DMA) on the AMBA bus, to reduced the size of the core. This has lead to a compact fault-tolerant implementation of the codec IP core that provides full SpaceWire and RMAP functionality and high-throughput using few on-chip resources, making the core suitable for both ASIC and FPGA implementation.

3 ADVANCED PRODUCTS

The development of advanced building blocks, such as the GRSPW SpaceWire IP core, has resulted in several advanced system-on-a-chip products that are already being shipped to customers.

The first flight products to be shipped to customers are from the LEON3FT-RTAX family. This is an implementation of the LEON3-FT SPARC V8 processor using the Actel RTAX FPGA technology. The fault tolerant design of the IP cores in combination with the radiation tolerant FPGA gives a total immunity to radiation effects. The LEON3FT-RTAX processor is provided in multiple configurations, covering both instrument and spacecraft control applications, and custom configurations are created on request. The configurations offer up to three SpaceWire links with support for implementing RMAP in software.

The first ASIC silicon with LEON3-FT and GRSPW is the GR702RC radiation-hard controller, which has been manufactured on the 180 nm technology from Tower with the objective to assess performance, design flow and radiation behavior of cores using the Ramon Chips' radiation tolerant cell library. The ASIC has been successfully validated and undergone static radiation testing.

The LEON3-FT processor core is being used as the design driver in an ESA activity [(1)] to validate and qualify the Design Against Radiation Effects (DARE) library, developed by IMEC under ESA founding, on the 180 nm technology from UMC. The ASIC will in addition to the LEON3-FT processor comprise a high-performance floating-point unit, a memory management unit and multiple SpaceWire interfaces with full RMAP support.

The LEON3-FT-MP project has been a preparation for the next generation ESA microprocessor [(2)] that has the ambition to provide 1,000 MIPS/MFLOPS in a single device. A verification and validation platform, consisting of a multiprocessing system based on the LEON3-FT and GRSPW cores, was successfully developed. Fault tolerant and multi-processing capabilities have been demonstrated and validated.

The planned LEON2-FT AT697F device is equipped with a powerful PCI (Peripheral Component Interconnect) interface, which is ideally suited for interfacing either a back plane bus such as Compact PCI (cPCI) or a companion chip. Gaisler Research has developed the GR701A companion chip that will add amongst others multiple SpaceWire interfaces, using the GRSPW core, to the AT697F device.

The GRSPW core is also being used in the SpaceWire test equipment developed by Gaisler Research. The GRESB Ethernet to SpaceWire Bridge is internally based on a Xilinx FPGA implementing a system-on-a-chip design featuring LEON3 and multiple GRSPW cores. The GRESB allows communication with SpaceWire links through Ethernet, supporting functions such as SpaceWire packet routing, IP tunneling, remote debugging of LEON3 and LEON2 processors, etc.

There are several more system-on-a-chip developments in progress using the LEON3-FT and GRSPW cores, made both in-house and by customers, of which many cannot be disclosed at this point in time.

4 CONCLUSIONS

By embracing the enabling technologies presented in this paper several powerful system-on-a-chip designs have been developed in a short period of time. The key factors have been efficient implementation of truly re-usable IP cores, such as the GRSPW SpaceWire codec, which have been designed with interoperation and portability in mind from the start. This has resulted in sophisticated flight products that are being shipped to customers.

5 REFERENCES

1. ARM Limited, "AMBA™ Specification", 13 May 1999
2. SPARC International Inc., "The SPARC Architecture Manual - Version 8", 1992
3. SpaceWire - Links, Nodes Routers and Networks, ECSS-50-12A, January 2003
4. Remote Memory Access Protocol, ECSS-E-50-11 Draft F, December 2006
5. Gaisler Research, "GRLIB IP Library User's Manual", 2006
6. Gaisler Research, "GRLIB IP Core User's Manual", 2006

Footnotes

(1) ESA Contract Number 19916/06/NL/JD

(2) ESA Contract Number 18533/04/NL/JD

SPACEWIRE IP FOR ACTEL RADIATION TOLERANT FPGAS

Session: SpaceWire Components

Short Paper

Steve Parkes[1], Chris McClements[1], Zaf Mahmood[2]

[1]STAR-Dundee Ltd, c/o University of Dundee, Dundee,DD1 4HN, Scotland, U.K.

E-mail: sparkes@computing.dundee.ac.uk

[2]Actel, UK

ABSTRACT

STAR-Dundee Ltd and University of Dundee have teamed with Actel to provide a range of SpaceWire IP cores optimised for the Actel RTAX-S radiation tolerant series of devices [1]. FPGA technology is ideal for instrument development, allowing one off equipment development without the large non-recurring engineering (NRE) costs of ASICs. The RTAX-S family provides a large number of available logic gates enabling complete instrument interface and control systems to be integrated on a single device.

SpaceWire [2] [3] was designed to provide a fast and efficient (low gate count) interface suitable for use onboard spacecraft. It is now being widely used by the major space agencies and aerospace industry across the world. SpaceWire is ideal for connecting an instrument into an onboard data-handling system and many missions are now selecting SpaceWire as the onboard interface of choice.

A ready laid-out SpaceWire interface with guaranteed timing, that fits into a corner of an Actel RTAX-S device would make the design of instrument interface and control FPGAs much simpler. Effort could then be concentrated on the application specific design without having to be concerned about the SpaceWire interface design.

The Actel "Block Flow" technology makes this possible. A functional block with fixed layout can be integrated with general user logic. STAR-Dundee are designing a series of SpaceWire interface blocks including a SpaceWire interface and a SpaceWire interface with Remote Memory Access Protocol (RMAP) support [4] and a SpaceWire router. The first of these IP blocks will be available in 4Q 2007.

SPACEWIRE RMAP IP ARCHITECTURE

The SpaceWire IP will be derived from the CODEC developed by University of Dundee for ESA [5], adapted to suit the Actel FPGA devices. The target is for this to operate at up to 200 Mbits/s in an Actel RTAX device. Time-codes will be supported in the SpaceWire interface.

The RMAP IP will again be tailored to the Actel devices. The architecture for a slave SpW/RMAP IP core is illustrated in Figure 1.

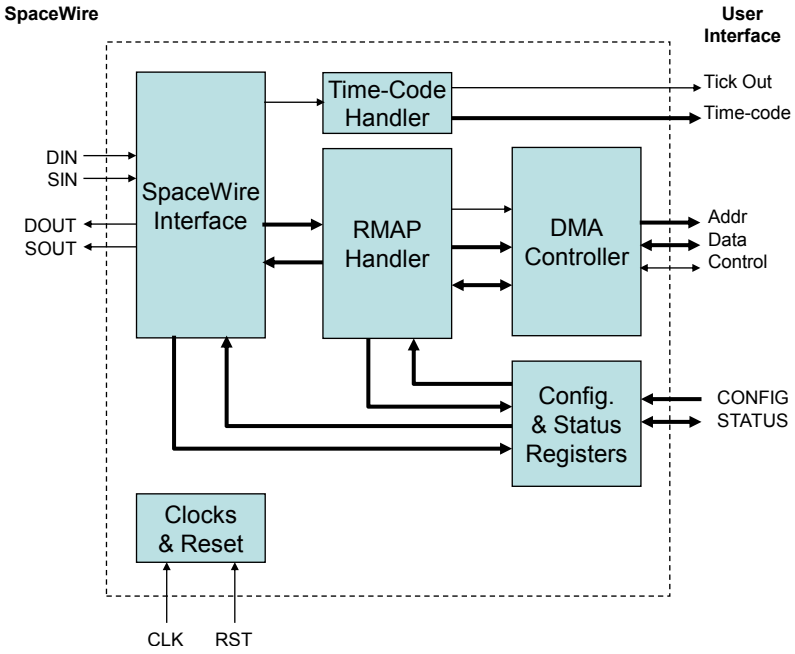


Figure 1 SpW/RMAP Interface Architecture

The SpaceWire Interface is responsible for receiving SpaceWire packets and time-codes and passing them to the RMAP Handler and Time-Code Handler respectively. It also transmits SpaceWire packets when requested to do so by the RMAP handler. The packets it sends are RMAP reply or acknowledgement packets.

The Time-Code Handler is responsible for checking time-codes and maintaining the value of the time-code counter. It will assert the TICK_OUT signal when a valid time code is received and put the value of each valid time-code on the TIME-CODE output.

The RMAP Handler is responsible for checking and responding to valid RMAP packets. It will set up the DMA controller to perform reads and writes to user memory and registers and will form the reply or acknowledgement to the RMAP command for sending by the SpaceWire interface.

The DMA controller provides the interface to user memory and registers. It is responsible for gaining access to the user data bus and performing memory or register, read or write operations.

The Configuration and Status registers hold configuration and status information for the SpaceWire interface and RMAP Handler. On power up certain configuration registers are loaded with default values specified by the CONFIG interface. Thereafter the configuration values may be changed by writing to the configuration registers either by a SpaceWire-RMAP command or by the user logic writing to the appropriate registers. Status information from the SpaceWire interface and RMAP Handler is held in status registers which can be read by SpaceWire-RMAP command or by the user logic. Certain status information is also available on dedicated signals, STATUS, from the SpW/RMAP IP core.

The Clock and Reset block is responsible for providing the user reset signal, RESET, to the relevant parts of the SpW/RMAP IP core ensuring a clean condition after the reset signal has been asserted. It is also responsible for generating any necessary clock signals from the single clock input signal, CLK.

An example sequence diagram for handling an RMAP verified write command with acknowledgement is given in Figure 2.

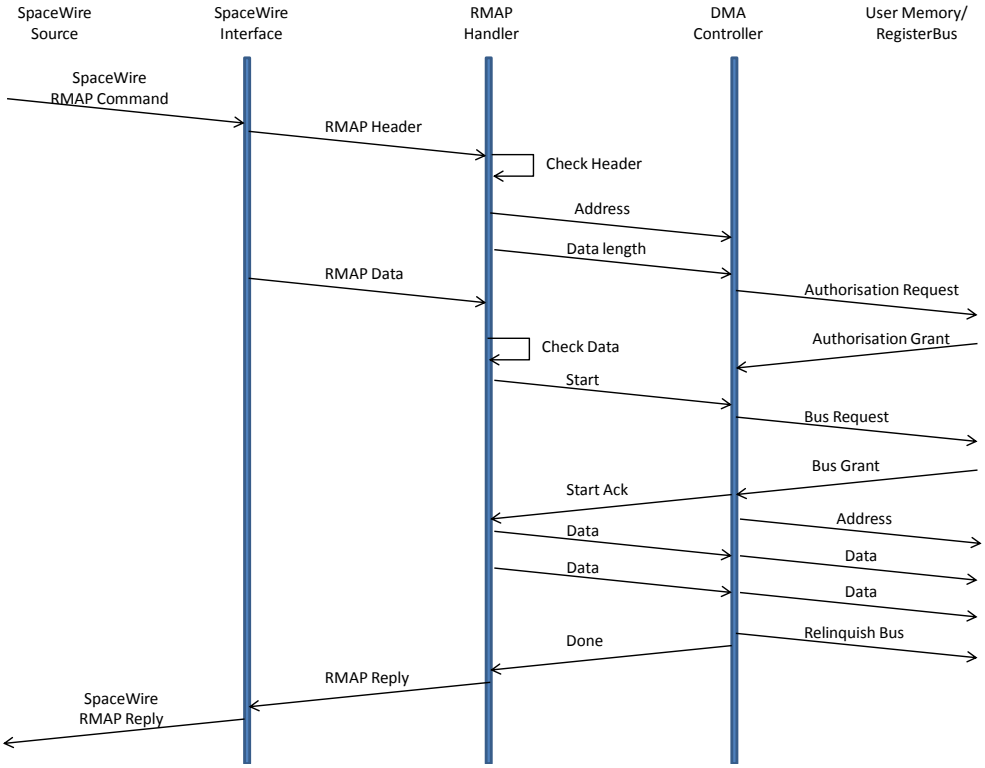


Figure 2 Sequence Diagram for RMAP Verified Write

The SpaceWire packet containing the RMAP write command arrives at the SpaceWire interface and is passed on to the RMAP handler. The RMAP header arrives character by character at the RMAP handler and is decoded and checked for errors. Assuming that there are no errors the address to write to and the amount of data to be written are passed to the DMA controller. The DMA controller checks with the user logic that it is authorised to write to this area of memory.

In the meantime the data has arrived at the RMAP handler it is buffered and the data CRC checked to make sure that the data is without error. The RMAP handler then tells the DMA controller to start writing the data to the specified area of user logic memory. The DMA controller requests use of the user bus and when granted access will transfer the data from the buffer in the RMAP handler to the user logic memory. Once the data has been transferred the user bus is relinquished and the RMAP handler is told that the DMA transfer is complete.

The RMAP handler now forms the RMAP reply with the status field set to zero to indicate success. This is sent back to the source of the original RMAP command by the SpaceWire interface.

CONCLUSIONS

An IP core containing a well proven SpaceWire CODEC with added RMAP support will be available for use in a radiation tolerant FPGA with fixed placement to ensure the timing. This will be a valuable addition to the current range of SpaceWire components.

REFERENCES

[1] Actel RTAX FPGA website http://www.actel.com/products/milaero/rtaxs/

[2] European Cooperation for Space Standardization, Standard ECSS-E-50-12A, "SpaceWire, Links, Nodes, Routers and Networks", Issue 1, European Cooperation for Space Data Standardization, January **2003**.

[3] European Space Agency, "SpaceWire Web Page", European Space Agency, http://spacewire.esa.int/

[4] Parkes S.M. et al, "Remote Memory Access Protocol", ECSS-E50-11 draft F, December 2006.

[5] ESA IP Cores http://www.esa.int/TEC/Microelectronics/SEMVWLV74TE_0.html

[6] STAR-Dundee Website www.star-dundee.com

Monolithic Radiation Tolerant Multi-Gigabit Space Wire Fiber/Copper Transponder with Minimal Delay Synchronization

Session: Space Wire Components

Vladimir Katzman, Glenn P. Rakow, Vladimir Bratov, Sean Woyciehowsky, Jeb Binkley.

Vladimir Katzman, Vladimir Bratov, Sean Woyciehowsky and Jeb Binkley are with ADSANTEC Inc., 27 Via Porto Grande, Rancho Palos Verdes, CA 90275, U.S.A.

Glenn P. Rakow is with NASA Goddard Space Flight Center, Maryland, U.S.A.

E:mail: vkatzman@adsantec.net; vbratov@adsantec.net; woycieho@adsantec.net; jbinkley@adsantec.net;

grakow@sunland.gsfc.nasa.gov

Abstract

Current and future programs of near-Earth and deep space exploration require the reconfigurable, high-speed, intra-satellite interconnect systems based on switching fabric active backplane architectures with high-speed multi-gigabit rate serial interfaces. Electrical and/or optical transponders operating with Space Wire or Gigabit Ethernet protocols are required to support the associated data interconnects. The systems must be easily upgradeable, power-efficient, fault-tolerant, EMI-protected, and capable of operating effectively for long periods of time in harsh environmental conditions including radiation effects. We are developing a monolithic, copper/optical, radiation-tolerant transponder, which will be implemented either as a stand alone ASIC for copper interconnect or as a hermetically-sealed fiber pigtailed multi-chip module with an FPGA-friendly parallel interface; featuring an improved radiation tolerance, high data rate, low power consumption, and advanced functionality. The transponder utilizes our patent-pending current-mode logic library of radiation-hardened-by-architecture cells. 8B10B encoding will be used to achieve data disparity equal to 0 and perform a minimal delay clock recovery scheme, combining a multiple phase interpolation technique and our proprietary radiation tolerant clock recovery scheme. Zero latency data interconnect capabilities of the copper version are achieved by applying patent pending radiation-hard-by-architecture multi-level interconnect techniques. The above described characteristics are implemented in an advanced SiGe BiCMOS technology. The prototype device will be available in the fourth quarter of 2008.

Standardisation

Tuesday 18th September

11:15 – 12:35

SPACEWIRE AND IEEE 1355 REVISITED

Session: Standardization

Long Paper

Dr Barry M Cook, C Paul H Walker …

4Links Limited, Bletchley Park, Milton Keynes MK3 6ZP, England

E-mail: [Barry, Paul] @4Links.co.uk

ABSTRACT

The ECSS (European Cooperation for Space Standardization) SpaceWire standard includes a list of changes from the standard from which it was derived, IEEE 1355. Several of these changes are improvements, for example fixing the initialization state machine, adding Time Codes, and adding a network layer. The list in the SpaceWire standard does not include a number of aspects of 1355 that SpaceWire has discarded, yet which are being requested by user missions — sometimes where the need is so strong that the missions are already using adaptations from the SpaceWire standard. This paper considers these user needs, with reference to where they were covered, at least in part, by IEEE 1355. It concludes that as SpaceWire is evolving to include SpaceFibre, it should also evolve in other directions to form a family of standards, all sharing the common link, 'exchange' and packet layers of the 1355/SpaceWire standards.

1 INTRODUCTION AND SCOPE

The IEEE 1355 standard [1] was broader than SpaceWire [2] in terms of the variety of transmission speeds and distances covered and the encoding and physical layers to achieve these speeds and distances. 1355 was less deep than SpaceWire in that it did not include the network layer. What is really important in both 1355 and SpaceWire is the common character alphabet and the separation of the link layer (initialization and flow-control) from the 'exchange' and packet layers.

1.1 MULTIPLE ENCODINGS AND PHYSICAL LAYERS

The IEEE 1355 standard defined a family of three separate encodings to cover different speeds and distances:

- Data-Strobe encoding, from which SpaceWire is derived;
- Three-of-Six (TS) encoding, which is already flying on space missions;
- High-Speed (HS) encoding, capable of transmission through copper or optical fibre at rates exceeding 1Gbit/s, much as is the intention for SpaceFibre [3].

IEEE 1355 Data-Strobe character encoding was identical to SpaceWire except for the EOP2 character (Exceptional End of Packet) which SpaceWire replaced by EEP (Error End of Packet). The electrical signals were very similar to SpaceWire, except

that Pseudo-ECL was mature technology in the early 1990s and so was used in preference to the lower-power LVDS which had yet to prove itself.

Three-of-Six encoding was used in IEEE 1355 for low-speed optical fibre connections. The Three-of-Six code takes four bits of data and encodes them into six bits such that each six-bit character has three ones and three zeros. There are 20 such characters, which conveniently gives 16 characters for data plus a further four characters for control. The code uses twelve bits to encode each Byte of data, which is a little less efficient than Data-Strobe encoding that uses 10 bits for each Byte, but the code has some useful advantages. It is DC-balanced over a very short period which means that it can be coupled through transformers or capacitors. The code is very simple when compared with the 8B10B code that tends to be used for Gbit/s links, which means less silicon, reduced power, and that it easier to verify and build.

The High-Speed (HS) encoding of IEEE 1355 allowed for communication exceeding 1Gbit/s, much as SpaceFibre is currently intended. Some choices would be made differently now, ten to twenty years from when they were made for 1355 HS, but the HS links were demonstrated in the early 1990s, including a routing switch.

1.2 Single Packet Structure for All Encodings and Physical Layers

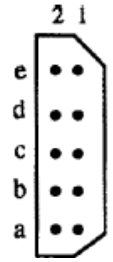
Although IEEE 1355 had several encodings and physical layers, the common threads through them all were:

- Flow-control, with the flow-control unit (Flit) chosen appropriately for the speed and distance;
- A common character alphabet, including 256 data characters plus two EoP characters and the link layer characters Flow-control, and Null. The optical fibre codes also had an init character ('comma' in communications terms) that did not occur during normal data transfer, to simplify initialization.
- A minimalist packet structure such that packets are simply delineated by the End of Packet character.

The authors suggest that these threads are the really fundamental aspect of SpaceWire that makes it effective, and that multiple physical layers are to be expected.

2 Data-Strobe Physical Layer

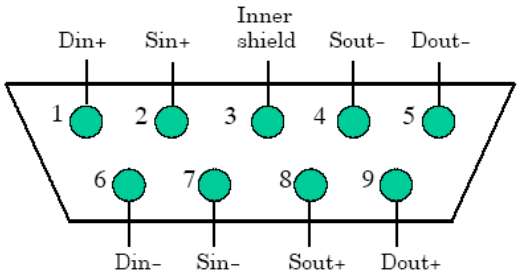
The IEEE 1355 connector was clearly not suitable for use in space, but its pinout and format have certain advantages over the 9-pin Micro-D selected for SpaceWire. The 1355 details are copied from Figure 5.7 and Table 5.9 of the 1355 standard:



	2	1	
e	DS_DE_data_in_plus	DS_DE_data_in_minus	(Chamfer corners at this side of the connector)
d	DS_DE_strobe_in_plus	DS_DE_strobe_in_minus	
c	No connect or optional power return	No connect or optional power	
b	DS_DE_strobe_out_minus	DS_DE_strobe_out_plus	
a	DS_DE_data_out_minus	DS_DE_data_out_plus	

2.1 CONNECTOR/CROSSTALK

From the above figure and table (copied from the IEEE 1355 standard), it can be seen that the connector had five rows of two pins. Each row except row c was used for a differential pair, with the signals coming in from a cable on rows e and d and out to a cable on rows b and a. Row c could either be left open or could carry optional power and return. In either case there is a significant gap between row b and row d, and so the two directions are separated and near-end crosstalk (or NeXt) is not significant.



The Micro-D (shown with its pinout, copied from Figure 10 of the ECSS standard) has nine pins rather than the 10 pins of the 1355 connector, and the pins are interleaved for mechanical efficiency with the "D" shape. Unfortunately this results in imbalanced coupling between the pins. In particular, while pin 2 (Sin+) is well separated from Pin 4 (Sout-), pin 7 (Sin-) is as close to pin 8 (Sout+) as it is to its pair, pin 2, and so there is crosstalk between these pins. Near end crosstalk between output and input can be a significant problem and in SpaceWire's case, almost the entire contribution to near end crosstalk is in the connector rather than the cable. The imbalance that this crosstalk introduces also results in EMC emissions which, while normally acceptable for CE and FCC regulations, may be undesirable on a spacecraft that needs to receive weak RF signals.

SpaceWire users have recognized these issues. NASA Goddard Space Flight Center has commissioned an alternative connector [4] that is rather larger than the micro-D but which presents much less of an imbalance or discontinuity to the signals. 4Links have described mitigation techniques for the EMC emissions that can be implemented with low-cost components and with negligible additional mass [5].

2.2 CABLE SPECIFICATION

The IEEE 1355 standard specifies parameters for physical components such as cable and connector as normative, and includes informative appendices that give examples of how the specification might be met. So 1355 includes parameters for cable attenuation and crosstalk. The SpaceWire standard omits these important parameters, but includes manufacturing details of how the cable should be constructed. As a result it is not possible, by external examination and test against parameters given in the standard, to determine if a SpaceWire cable will be adequate for its purpose.

2.3 POWER DISTRIBUTION

Like USB, IEEE 1394, and Power over Ethernet, the IEEE 1355 Data Strobe connector included provision for carrying power down the signal cable, using the two pins on Row c of the 1355 connector. During SpaceWire standardization, this was discarded.

AFRL's PnPSat is perhaps unique in the space industry, in taking modularity concepts further than most others [6]. Part of this is to recognize that a satellite needs a data network, a power network, and a test network, and to integrate them into a single connector. They still use Micro-D, but with 25 pins rather than 9 pins.

3 PACKET LENGTH CONSTRAINTS

3.1 MAXIMUM PACKET LENGTH

The T9000 transputer, which was the original source of the technology that has evolved to SpaceWire, had a maximum Payload/Cargo of 32 Bytes. In the IEEE 1355 standard, this restriction was removed and SpaceWire has kept the lack of constraint on packet size.

For point-to-point connections without routing switches, long packets are not a problem. When long packets go through routing switches, they block all the links on their path from carrying other traffic. If the large packet is a low-priority file transfer, and the blocked packet is a high-priority alarm, then the alarm is blocked until the file transfer has completed.

Problems can also occur if packets traverse both slow links and fast links, because the packet duration is determined by the slowest link on the path. One solution to mixing slow and fast links is to place the packets in temporary store-and-forward buffers when there is a change of link speed. But the buffers can only cope with packets that will fit, which imposes the traditional Maximum Transfer Unit (MTU) on packet size.

SpaceWire is still greatly superior to bus or ring topologies and standards, even when these issues are taken into account, because there are normally multiple paths available in a SpaceWire network. And we definitely do not recommend a return to packets as short as 32 Bytes. But it may be that recommendations on packet size (and on the buffering inside routing switches) would prevent unexpectedly low performance resulting from mixed link speeds or large packets. Alternatively, pre-emption of long or slow packets by high-priority packets would retain the freedom while providing the critical performance when needed [7].

3.2 MINIMUM PACKET LENGTH

Both the IEEE 1355 standard and the SpaceWire standard define packets to be:

(DEST) (PAYLOAD) (End_Of_Packet)

although SpaceWire uses the word CARGO instead of PAYLOAD

IEEE 1355 then goes on to say that either DEST or PAYLOAD can be zero length. If both are zero length, then the packet is empty. An empty DEST could be used, for example, between two nodes not connected by a routing switch network; and empty PAYLOAD could be used, for example, as an acknowledgement (it was used this way on the T9000) or as an interrupt or watchdog signal.

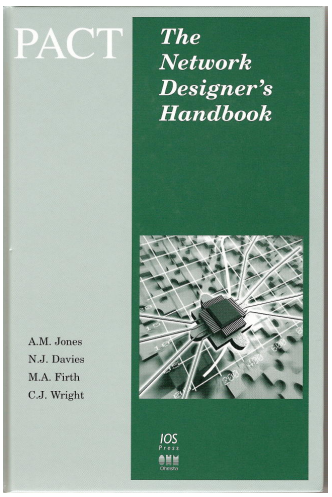
SpaceWire acknowledges that the DEST can be zero, but then goes on to require a minimum cargo of one data character.

In practice, as the SpaceWire standard notes, empty packets can occur as a result of fault conditions, and so they need to be handled. Unfortunately, products are on the market that discard empty packets without updating the flow-control credit, and so suffer from loss of flow-control credit after receiving a number of empty packets.

4 NETWORK AND RELIABILITY STUDIES

4.1 THE NETWORK DESIGNER'S HANDBOOK

Simulations performed on a wide variety of IEEE 1355 networks fed through to a book, 'The Network Designer's Handbook' [8], published in 1997. Most of the networks considered are arrays such as (hyper)cubes, toroids, and Clos networks. On the other hand, most of the SpaceWire networks that are being implemented are less structured than these so that the results may not be directly applicable. In spite of this, there is still much that can be gained from studying the results presented. A brief summary of the sort of information in the book is given in the authors' companion paper at this conference [9]. Those SpaceWire users who are doing their own simulations for architectural modelling may also find that the results inform their network architecture decisions.



4.2 CERN TEST-BED AND RELIABILITY TEST

Further work went into a test-bed for IEEE 1355, with 1024 nodes, at CERN [10]. This was used both to validate the simulation results and to determine reliability. The 1024-node network contained 1344 active Data-Strobe links, of which about a third used differential buffers and 2metre cables, with the others non-differential on printed circuit boards. The results gave a per-link bit error rate better than 9.6×10^{-18}.

5 EVOLUTIONS FROM IEEE 1355

5.1 ASYMMETRIC DATA RATE

Some years ago, the authors' company designed an evolution from IEEE 1355 for a consumer electronics application. The application had data flowing mostly in one direction and so had a real need for asymmetric data rates. Other requirements meant that the physical layer was completely different from both 1355 and SpaceWire, but the important 1355/SpaceWire protocols were retained.

Several potential users of SpaceWire have pointed out that their traffic is almost entirely in one direction, and that the harness mass of eight twisted pairs in the SpaceWire cable discourages them from using SpaceWire. One solution is a Simplex mode for SpaceWire, and there is indeed a paper on this topic later in this session [11]. Pure simplex is, however, likely to lose the benefits of flow-control and of being able to control the data stream from within the one network. Much better, we believe, would be a half-duplex implementation [12] of SpaceWire:

- that provides the required unidirectional bandwidth;
- that halves the cable mass;
- that retains the reverse channel for status and control;
- that still has a great deal of compatibility with the SpaceWire standard;
- and that builds on our experience of asymmetric data with 1355.

5.2 COPPER VERSION OF 1355 TS ENCODING FOR FIBRE

The TS Encoding of IEEE 1355, described briefly above, was intended in the 1355 standard to provide longer distance connection than the cabled Data-Strobe version, at similar data rates. A group of collaborators in the space industry realized, however, that the TS encoding, when carried on copper wire, reduced cable mass and power compared with the Data-Strobe version and gave a number of other advantages. We will leave further description to those who did the work (amongst the authors of [13]), but the result is an interface that is logically extremely close to SpaceWire and that is currently flying alongside 1355-Data-Strobe/SpaceWire on successful ESA missions.

6 CONCLUSIONS

While we appreciate that SpaceWire makes many improvements over IEEE 1355, we also recognize that potential users of SpaceWire are deterred by the absence of features that they need, and that some of these features were embodied in the 1355 standard but were deleted from SpaceWire. In a number of cases, including an electrically better connector, power distribution, and the use of the TS encoding, different parties in the space industry have overcome the issues themselves. We welcome their initiatives to improve on the ECSS standard.

There may be other aspects, such as consideration of packet lengths and network topologies — particularly building test-beds for such topologies — where work done ten or more years ago for 1355 will still be of great benefit to many users of SpaceWire.

Evolution of SpaceWire is ongoing, for example with SpaceFibre. Further evolution of SpaceWire such as pre-emption (below the link layer) or reduced harness mass (from a different physical layer), may also be appropriate. As long as it preserves the important layers of IEEE 1355/SpaceWire (the link, exchange, and packet layers), we believe that such evolution should be classed as belonging to an enlarged SpaceWire family, all of whose members share common fundamentals. This larger family could greatly increase the usefulness and hence the penetration and success of SpaceWire within the space industry — and beyond.

REFERENCES

In the following references, * indicates a paper that is being presented at this conference.

[1] Bus Architecture Standards Committee of the IEEE Computer Society, "**IEEE Std 1355-1995**, IEEE Standard for Heterogeneous InterConnect (HIC) (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction)" IEEE, 1995

[2]The ECSS-E-50-12 Working Group, "**ECSS-E-50-12A 24 January 2003**, SpaceWire - Links, nodes, routers and networks", published by the ECSS Secretariat, ESA-ESTEC, Requirements & Standards Division, Noordwijk, The Netherlands

[3]* Steve Parkes, Chris McClements, Martin Suess, "**SpaceFibre**"

[4]* Shaune S. Allen, "**SpaceWire Cable and Connector Variations**"

[5] Barry M Cook, Paul Walker, "**Reducing Electromagnetic Emissions from SpaceWire**", Data Systems In Aerospace (DASIA), Naples, 29 May to 1 June 2007, proceedings CD SP-638 – August 2007, ISBN 92-9291-202-8, ESA.

[6]* Donald Fronterhouse, "**PnPSAT**"

[7] Barry M Cook, Paul Walker, "**SpaceWire – Improvements in Support of Mission Requirements**", SDSS 2005, First Annual ESA Workshop on Spacecraft Data Systems and Software, Noordwijk, The Netherlands, October 2005.

[8] P. Thompson, A.M. Jones, N.J. Davies, M.A. Firth and C.J. Wright (Eds.) **"The Network Designer's Handbook",** Volume 51: Concurrent Systems Engineering Series, 1997, 309 pp., softcover, ISBN: 978-90-5199-380-6

[9]* Barry M Cook, C Paul H Walker, "**SpaceWire Network Topologies**"

[10]S Haas, D A Thornley, M Zhu, R W Dobinson and B Martin, "**The Macramé 1024-Node Switching Network**" Microprocessors and Microsystems, IEEE 1355 Special Issue, V21, Nos 7,8, 30 March 1998, also summarized in Chapter 8.2 of The Network Designer's Handbook

[11]* Eugenue Yablokov, "**Simplex Mode in SpW Technology**"

[12] Barry M Cook, Paul Walker, "**Asymmetric SpaceWire**", to be published, abstract available from 4Links

[13] C. M. Carr, E. Cupido, C. G. Y. Lee, A. Balogh, T. Beek, J. L. Burch, C. N. Dunford, A. I. Eriksson, R. Gill, K. H. Glassmeier, R. Goldstein, D Lagoutte, R. Lundin, K. Lundin, B Lybekk, J. L. Michau, G. Musmann, H. Nilsson, C. Pollock, I. Richter, J. G. Trotignon, "**RPC: The Rosetta Plasma Consortium**", Space Science Reviews, Volume 128, Issue 1-4, pp. 629-647, Springer, February 2007

SPACEFIBRE

Session: SpaceWire Standardisation

Long Paper

Steve Parkes[1], Chris McClements[1], Martin Suess[2]

[1]School of Computing, University of Dundee, Dundee, DD1 4HN, Scotland, U.K.

E-mail: sparkes@computing.dundee.ac.uk

[2]Martin Suess, ESTEC, Noordwijk, The Netherlands

ABSTRACT

SpaceFibre is a proposed very high speed serial data link intended to complement the existing SpaceWire high-speed data link standard. SpaceWire operates at speeds up to 200 Mbits/s in radiation tolerant technology. SpaceFibre will be able to operate over fibre optic and copper cable and support data rates of 2.5 Gbit/s and possibly higher.

This paper describes the work done by University of Dundee in developing SpaceFibre. It starts by looking at lessons learnt from the development of SpaceWire. The principal requirements for SpaceFibre are then listed and the baseline architectural design presented. It concludes with an overview of the SpaceFibre demonstration system developed by University of Dundee.

LESSONS LEARNT FROM SPACEWIRE

SpaceWire [1] [2] is an effective onboard communications link which is being used on a range of ESA, NASA, JAXA and other space missions. It does however have a number of issues that should be addressed in the development of a new onboard communication standard.

1. **Cable Mass:** SpaceWire uses a data-strobe signalling technique [1]. The data and strobe signals are differentially encoded using LVDS so that two twisted pairs are required for communications in each direction, giving a total of eight wires. This approach was adopted in preference to a self-clocking data stream because it removed the need for a phase-locked loop in the receiver and made implementation of the SpaceWire CODEC possible in any ASIC or FPGA technology. It does mean, however that the cable contains more wires and is thus heavier (~87g/m) than would otherwise be necessary. SpaceWire cable is specially manufactured for flight applications. It should be noted that the data-strobe technique gives better skew tolerance than a simple data-clock method.
2. **Data Rate:** The speed of transmission is limited by two primary factors: the attenuation in the cable at high frequencies and the skew between data and strobe signals. Attenuation reduces the amplitude of the SpaceWire signals as they

propagate down the cable making them more susceptible to noise. Skew between data and strobe pairs affects the signal timing reducing the maximum data rate that can be supported. The longer the cables the worse these effects become. SpaceWire will work at data rates of 200 Mbits/s over distances of 10 m. Shorted cables allow higher data rates while longer cables can only support lower data rates.

3. **Character Sizes:** There are two types of character sent over a SpaceWire link: control and data characters. In addition there are two control code sequences made up from control and data characters (NULL and time-code) [1]. This results in four different length codes sent over the SpaceWire link: control 4-bit, NULL 8-bit, data 10-bit and time-code 14-bit). Handling these different sized characters complicates the transmitter and receiver circuitry. The original intention (I believe) with having control characters shorter than data characters was to save link bandwidth when FCTs were sent and also, less importantly, when EOP markers were sent. In SpaceWire an FCT is exchanged for eight data-characters, which results in a 4-bit overhead to send 80-bits data (encoded). Note that the overhead is actually in the opposite direction of the link to the direction that the data is travelling. This is a 5% overhead. The same level of overhead could be achieved by exchanging one 10-bit FCT for every 200 bits data i.e. 20 data characters. This would increase the minimum amount of buffering needed, which was a possible issue in the original IEEE1355-1995 standard [3] from which SpaceWire was derived, but is not an issue today. It would allow the control characters to be the same size as the data characters simplifying transmit and receive circuitry.

4. **Parity Coverage:** Parity coverage in SpaceWire is rather peculiar. The parity bit covers the data/control field from the previous character and the data/control flag from the next character. This approach complicates both the transmitter and receiver because two characters have to be considered together to determine the parity value. The reason that this was done in IEEE1355-1995 is not known.

5. **Transmitted DC Component:** SpaceWire characters use all possible bit patterns of the 10-bit data and 4-bit control characters. This means that depending on the data pattern sent there will be a DC component to the transmitted signal. For example, if a stream of data characters of value 0xff is sent, there will be a positive DC bias. This degrades the transmission characteristics of the SpaceWire signals making them broadband and prevents AC coupling from being used.

6. **Galvanic Isolation:** SpaceWire does not provide a method of galvanic isolation. A technique using capacitive coupling and bus-hold circuits has been proposed for SpaceWire [4] as used in IEEE1394 [5] but has not been prototyped. A galvanic isolation unit has been designed at Dundee to support ground based isolation at speeds up to 100 Mbits/s.

7. **Matched Impedance Connectors:** The 9-pin micro-miniature d-type connectors used in SpaceWire are not controlled impedance. This can cause problems at

higher speeds when there will be reflections from the impedance mismatch adding noise and jitter to the SpaceWire signals. A matched impedance connector is essential at higher speeds than 200 Mbits/s. The 9-pin micro-miniature d-type connector was used in SpaceWire because it was readily available in space qualified form and was able to operate with the data rates that SpaceWire was then intended to work at.

8. **Initialisation Protocol:** The initialisation protocol in SpaceWire is based on part handshake and part timing. This can lead to false initialisation caused by noise and data characters being sent due to noise, before a parity error or other error is detected and the link is terminated. This approach was taken to allow backwards compatibility with IEEE1355-1995 and the SMCS devices. A full handshake scheme which did not use flow control tokens (FCTs) would be better and prevent the flow of data characters before a proper link connection has been made. Note that the use of bias resistors on the LVDS receiver inputs can provide a noise threshold which will eliminate the false start due to noise problem in most systems.
9. **Link Speed Negotiation:** The initial link speed is fixed in SpaceWire to 10 Mbits/s. This link speed is always used for link initialisation. Once a connection has been made either or both ends of the link can switch to different speeds. There is no link speed negotiation protocol in SpaceWire, so that if a low speed device is connected to one operating at higher speed the two devices will start communication at 10 Mbits/s, make a connection, then switch to their operating speeds. At this point the slower device will fail to receive the characters from the high speed device correctly resulting in an error and disconnection of the link. A link negotiation protocol would allow a more controlled setting of the link operating speed so that when a high speed device is connected to a lower speed device, they operate at mutually acceptable data rates.
10. **Initialisation Speed:** Because the initialisation protocol has a fixed timing and because the initial link speed is fixed, the time taken to initialise a SpaceWire connection is around 20 us. If data is being transferred at 20 Mbytes/s corresponding to 200 Mbits/s data signalling rate, then the 20 us initialisation time is equivalent to 400 bytes of data. If a link level retry mechanism is being implemented then each router or node will need to store at least this amount of data in the event of a temporary fault (e.g. parity error) occurring in a continuous data stream. In practice the buffer storage could be significantly higher than this. If the link is to always operate at high speed (e.g. 200 Mbits/s) then a faster disconnect and re-initialisation interval would result in less buffer space being needed for a retry mechanism. This approach is being considered by GSFC for JWST.
11. **Transport Layer:** SpaceWire lacks a transport layer so there is no consistent way of managing a connection in SpaceWire: buffer management and end-to-end flow control, fault recovery and packet retransmission are all missing.

While addressing these issues the important features of SpaceWire should not be lost:

- Simplicity
- Low implementation cost (gate count)
- Bi-directional, full-duplex communication
- Time-code distribution
- Group adaptive routing

REQUIREMENTS

SpaceFibre aims to extend the capabilities of SpaceWire: improving the data rate by a factor of 10, reducing the cable mass by a factor of four and providing galvanic isolation. It will also address the other issues with SpaceWire.

The principal requirements for SpaceFibre are listed below:

- Provide symmetrical, bi-directional, point-to-point link connection
- Operate at high speed (1-10 Gbits/s) with a target of at least 2.5 Gbits/s
- Operate over fibre cable lengths of up to 100 m
- Also operate over copper cable over shorter length of 5 m
- Have a fibre optic cable mass of less than 20 g/m for a full-duplex link
- Provide galvanic isolation
- Support arbitrary network architectures
- Support mixed SpaceWire-SpaceFibre networks using a SpaceWire-SpaceFibre router
- Be able to multiplex a scalable number of SpaceWire links over a SpaceFibre link

ARCHITECTURAL DESIGN

The baseline concept for the SpaceFibre interface is illustrated in Figure 1.

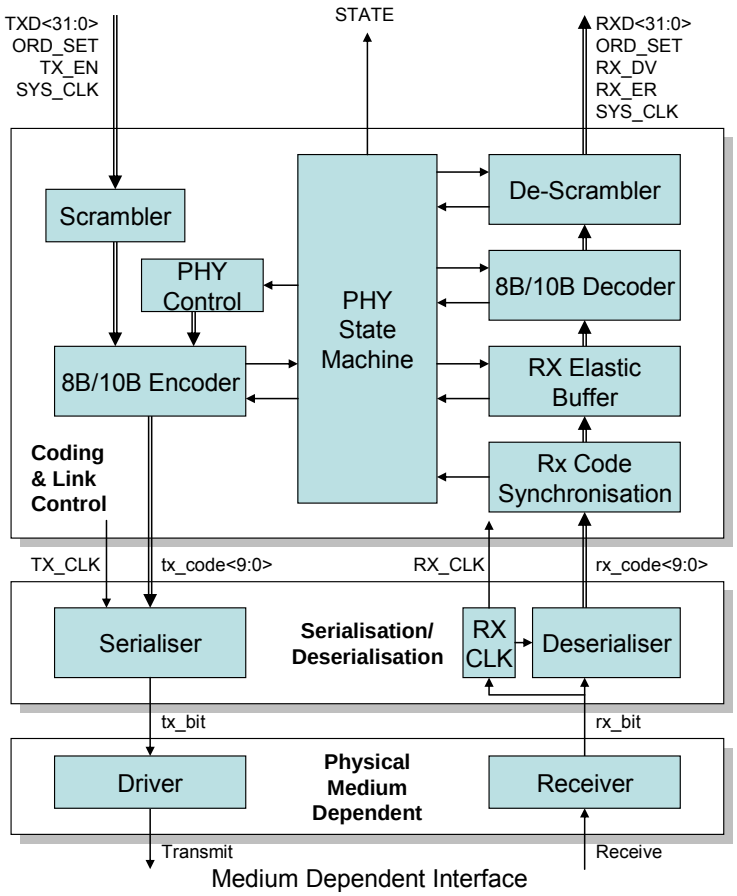


Figure 1 Baseline SpaceFibre Interface

This block diagram shows the possible interfaces that may be exposed.

Data words (32-bits) or ordered sets to be transmitted are passed though the transmit side of the port interface. They are first scrambled by a data scrambler which is re-seeded at the start of a frame. The scrambled data is then encoded byte by byte by the 8B/10B encoder [6] and passed on to the Serialiser. The bit stream from the Serialiser is driven onto the transmission medium by an appropriate medium dependent driver.

Whenever there is no data or ordered sets to transmit, idle ordered sets are automatically added to the data stream.

The bit stream is recovered from the medium by first converting the medium dependent signal into a digital signal. A receive clock is generated from this digital signal using a phase-locked loop and is used to recover the transmitted bit stream which is converted to 10-bit words using the de-serialiser. Using the 8B/10B "comma" contained in the idle ordered sequences the receive code synchronizer synchronizes to 10-bit word boundaries and passes a correctly aligned 10-bit word to the receive elastic buffer. The elastic receive buffer copes with any differences in system clock and receive clock by adding or deleting Idle ordered sets in the elastic buffer. This means that the interface to the system is synchronous to the system clock. The output from the elastic receive buffer is passed to the 8B/10B decoder. This decodes the 10-bit word to an 8-bit wide data stream plus comma codes. The decoded data stream is fed to a descrambler which recovers the data originally sent.

A link control state machine is responsible for link initialization, link-level flow control and for responding to errors.

SpaceWire time-codes may be transmitted using a subset of the ordered sets.

SPACEFIBRE PHYSICAL LAYER

Patria New Technologies Oy, VTT, INO, Fibre Pulse and Gore have been working on the physical fibre optic components for SpaceFibre with some encouraging results. INO selected and tested a fibre optic cable core which is radiation tolerant. Fibre Pulse recommended suitable robust fibre optic connectors. Gore assembled cables and connectors into rugged cable assemblies. VTT developed a fibre optic transceiver that can operate at well over 2.5 Gbits/s and is thought to be radiation tolerant. These physical components have been tested by Patria New Technologies Oy. These fibre optic components can be seen on the right hand side of Figure 2.

Figure 2 SpaceFibre Demonstration Unit

SPACEWIRE-SPACEFIBRE ROUTER

The prototype SpaceFibre CODEC has been designed in a SpaceWire-SpaceFibre router which sends SpaceWire packets over SpaceFibre. This prototype system was implemented on a Xilinx Virtex 4 device using an ML405 prototyping board. The SpaceFibre interface is implemented using the RocketIO facilities of the Xilinx FPGA device. In this demonstration system four SpaceWire ports can be multiplexed over a single SpaceFibre link which operates at 2 Gbits/s. The reduced speed compared to the target speed of 2.5 Gbits/s is due to a problem with the ML405 prototyping board. A block diagram of the design in the Xilinx FPGA is illustrated in Figure 3.

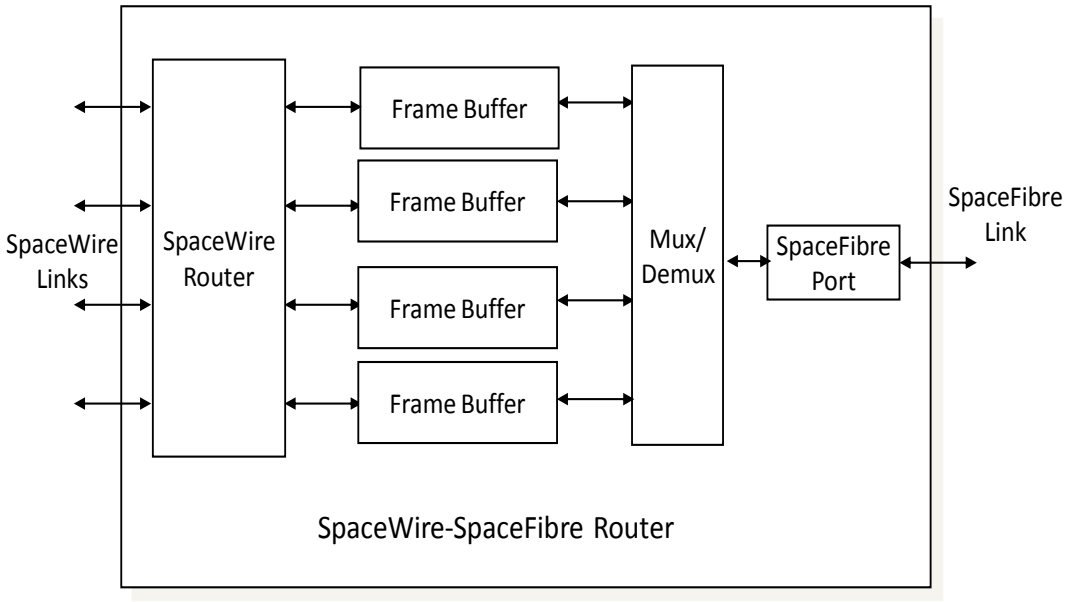


Figure 3 SpaceWire-SpaceFibre Router

The four SpaceWire links feed into an eight-port SpaceWire router which has four SpaceWire ports and four internal ports connected to the SpaceFibre interface. The internal ports are fed into frame buffers, which collect SpaceWire data-characters, EOPs, EEPs and time-codes and prepare them for sending over SpaceFibre. The prepared frames are multiplexed by the multiplexing/de-multiplexing block and fed into the SpaceFibre interface for transmission over SpaceFibre.

Frames arriving over SpaceFibre are de-multiplexed, and passed into frame buffers going the other way. Information is read out of these frame buffers and passed into the SpaceWire router and out of the appropriate SpaceWire port.

The ML405 board and SpaceWire connectors can be seen on the left hand side of Figure 2.

CONCLUSIONS

SpaceWire has been very successful for spacecraft onboard data handling and is now being widely used. There is a need now for a higher speed data link running at well over 1 Gbit/s. The development of SpaceFibre offers an opportunity to benefit from the widespread use of SpaceWire and also to overcome some of its deficiencies. The University of Dundee has developed an appropriate SpaceFibre CODEC. The next step is to provide a draft SpaceFibre standard document for wider review.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support of ESA for the work done on the SpaceFibre at the University of Dundee, which was done under ESA Contract Number 17938/03/NL/LvH.

REFERENCES

[1] European Cooperation for Space Standardization, Standard ECSS-E-50-12A, "SpaceWire, Links, Nodes, Routers and Networks", Issue 1, European Cooperation for Space Data Standardization, January **2003**.

[2] European Space Agency, "SpaceWire Web Page", European Space Agency, http://spacewire.esa.int/.

[3] IEEE Computer Society, "IEEE Standard for Heterogeneous Interconnect (HIC) (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction)", IEEE Standard 1355-1995, IEEE, June 1996.

[4] Parkes SM et al, "Review of Standard and Status", Digital Interface Circuit Evaluation Study WP1000 Technical Report, Document No. UoD-DICE-TN-1000, ESA Contract No. 12693/97/NL/FM, University of Dundee, July 1998.

[5] IEEE Computer Society, "IEEE Standard for a High Performance Serial Bus", IEEE Standard 1394-1995, IEEE, August 1996.

[6] Kembel, R.W., "Comprehensive Fibre Channel", Solution Technology, Northwest Learning Associates Inc, **2002.**

SPACEWIRE NETWORKS FOR PAYLOAD APPLICATIONS

Session: SpaceWire Standardisation

Short Paper

Christophe Honvault, Olivier Notebaert

Astrium Satellites SAS, 31 rue des cosmonautes 31402 Toulouse cedex 4

E-mail: christophe.honvault@astrium.eads.net, olivier.notebaert@astrium.eads.net

ABSTRACT

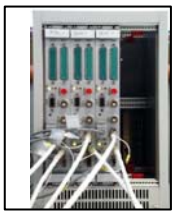
SpaceWire aims at becoming a candidate data-handling network for use onboard spacecraft, which could bring modularity and scalability to future onboard data processing systems. The use of routers brings several advantages as flexibility and scalability while the number and length of links can be decreased. However, the routers introduce constraints and limits in term of overall data throughput and latency. The increase of the link speed can push back the limits but do not solve the issue. Several other ways for improvement need to be explored: new router arbitration schemes, new multiplexer devices, implementation of dedicated communication protocols, the development of a methodology and associated tools for network analysis are possible ways for improving the scope of possible future usage of SpaceWire networks onboard spacecrafts.

1 SPACEWIRE EXPERIENCE

Astrium gain a strong experience on SpaceWire by leading several R&D studies and by contributing to the development of the main relevant building blocks for future space systems.

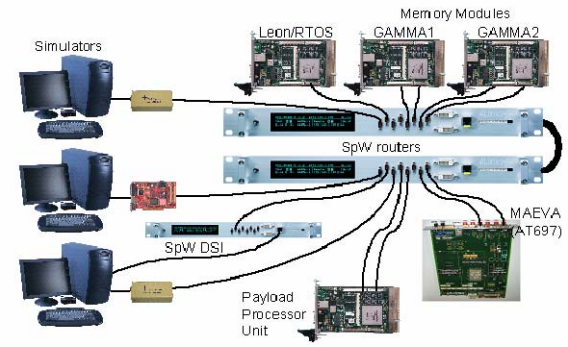
A SpaceWire codec IP core [1] has been developed and connected to Leon2 and Leon3 IP cores. It has been used in many studies making possible the identification of improvements. These improvements are presently under development and the new version of the IP core will be integrated in the SCOC3 system.

A3M [2] used point-to-point communication between three Leon based computers to implement and validate a safe and reliable distribution system. As an additional result, A3M has shown that SpaceWire disconnection detection can be used to implement fast failure detectors.



GAMMA [3] used a SpaceWire network including an eight ports router to implement and validate a distributed mass memory architecture in which several multi-threaded Leon based users can simultaneously access one or several mass memory modules.

PADAPAR [4] is an Astrium Satellites internal study that aims to define a generic architecture for future payloads that can match mid-term needs while optimizing the development process through standard building blocks definition. The main building blocks are implemented and interconnected by means of a SpaceWire network.



These studies and developments have made possible a fine characterization of the SpaceWire and make possible the confirmation of SpaceWire advantages and drawbacks.

2 IDENTIFICATION ADVANTAGES AND DRAWBACKS

The advantages and drawbacks of the SpaceWire can be easily identified in the ECSS standard [5] and are clearly confirmed by test. The advantages that are, among others, a great simplicity, the ease to use, a high data rate and a low consumption are not detailed in this short paper that prefers to focus on drawbacks in order to request or propose ways for improvement.

From mechanical point of view, with a maximum of 80 grams per meter, the SpaceWire links can be considered as heavy. This is weight can become an important problem when the SpaceWire is used in a point-to-point configuration. The use of routers can help to reduce the number and the length of the links. However, the routers themselves should be redounded.

The use of routers brings other numerous advantages as a great flexibility and scalability to the SpaceWire networks. However, the routers introduce constraints and limits in term of overall data throughput and latency. The overall data throughput of the system is reduced as soon as one of a device is not working at the maximum speed. A latency is introduced by each crossed router. The value of this latency mainly depends on the number and the size of all the messages that are transferred to a same output port. The phenomenon is amplified by the wormhole routing when several routers are crossed. If the worst case of data throughput and latencies can be easily determined, it can also be very restrictive at the time of the definition of a system. The increase of the link speed, the restriction of the size of messages exchanged and the use of the group adaptive routing can push back the limits but do not solve the issue.

Of course, these problems are not specific to SpaceWire and exist in other packet switched networks as Ethernet and explain, at least partly, why Ethernet is not used in real-time space systems.

3 POSSIBLE WAYS FOR IMPROVEMENT

3.1 DEVICES

The implementation of the arbitration policy in SpaceWire routers is generally limited to round-robin. The implementation of new arbitration policies (e.g. priority based), should make possible to extend the use of the SpaceWire networks to command and control. The implementation of configurable traffic controller in routers is another that could be considered.

The SpaceWire standard defines the broadcast and multicast distributions. However, no existing SpaceWire device presently implements one of these distribution mechanisms. When managed at the lowest level, these two distribution modes can be very useful to implement safe and distributed systems.

Multiplexer and high data rate devices should make possible the decrease of the overall latency while optimizing the data throughput by the supporting the concentration of the data traffic generated by the slowest units in the system. In addition, the number of links required to establish the connections would decrease.

3.2 PROTOCOLS AND STANDARD

New protocols are required to ensure the time constraints of applications. This approach has been followed with success on the Ethernet standard and is at the origin of AFDX [6] used on civil and military aircrafts as the Airbus A380 and A400M. Several time-triggered protocols could be envisaged as TTP [7], FlexRay [8] and even isochronous or asynchronous protocols as the one studied within the A3M activities.

The definition of new protocols that match the specific constraints of the space application must de defined. These protocols must comply with the low-level layers of the space communication networks as SpaceWire and take into account the needs of space applications. Of course, these protocols must also be compliant with the CCSDS SOIS standards.

3.3 TOOLS

The support tools must ease the definition, analysis and validation of system based on SpaceWire networks by taking into account the characteristics of all its components. Such tools exist for other standard networks (e.g. OpNet) and could be configured to SpaceWire. The possibility to mix simulation and connection of real hardware will be an asset.

4 CONCLUSION

The SpaceWire standard must remain as simple and efficient as possible. It must serve as the basis for the development of new devices, high-level protocols and support tools. The new devices and protocols should ensure the time constraints of critical applications and minimize the latency of the messages they exchange. The support tools must ease the definition and validation of system based on SpaceWire networks by taking into account the characteristics of all its components.

5 REFERENCES

1. Tam Le Ngoc –Astrium Satellite, “SpaceWire IP Hardware User Manual”, 25/04/2002.

2. Marc Le Roy - Astrium Satellite, “Advances Avionics Architecture and Modules (A3M) Final Report”, ESA Contract 13024/98/NL/FM(SC), September 2003.

3. Christophe Honvault - Astrium Satellite, “Generic Architecture for Mass Memory Access (GAMMA) executive summary” ESA Contract 17437/03/NL/AG.

4. Christophe Honvault, Olivier Notebaert – Astrium Satellites, “Prototyping a Modular Architecture for On-Board Payload Data Processing - PADAPAR”, Proc. ‘DASIA 2007 – DAta Systems In Aerospace Conference’, Naples, Italy, 29 May - 1 June 2007 (ESA SP-638, August 2007).

5. ECSS, “SpaceWire-Links, nodes, routers and networks”, ECSS-E-50-12A 24 January 2003.

6. ARINC 664P7, “Aircraft Data Network, Part 7, Avionics Full Duplex Switched Ethernet (AFDX) Network”

7. Hermann Kopetz, “Real-Time Systems: Design Principles for Distributed Embedded Applications”, 1997

8. Sev Gunes-Lasnet, Gianluca Furano – ESA/ESTEC – “FlexRay- An answer to the challenges faced by spacecraft on-board communication protocols”, DAta Systems In Aerospace Conference’, Naples, Italy, 29 May - 1 June 2007 (ESA SP-638, August 2007).

SIMPLEX MODE IN SPW TECHNOLOGY

Session: SpaceWire Standardisation

Short Paper

Eugenue Yablokov

Saint-Petersburg University of Aerospace Instrumentation. 67, B. Morskaya, Saint-Petersburg, Russia

E-mail: kabalenator@gmail.com

ABSTRACT

A very important task in developing SpaceWire interconnections is the development of the simplex mode. Reducing number of lines is a good solution for working with devices, which are not designed for working in full duplex. The simplex mode also reducing the number of lines, thus reducing square and weight, which is very useful on board of spacecraft. For example, simplex mode can be used working with video camera, the block will only receive information from camera. Simplex mode can be also used for control the block, for example moving source of light etc. All data will be transmitted using one Tx/Rx pair instead of two, thus decreasing the cost of the cable.

The SpaceWire controller can work in two possible directions of the simplex mode – transmitting and receiving. Transmitting part sends symbols due to standards and from time to time use the special mode of reconnection. The receiving part establishes connection and detects errors. The FCT symbols are not sending, so the receiving part is always ready to receive the symbol of data.

Due to changing the number of lines, however, several parts of the standard SpaceWire were changed. Sending FCT symbols for reserving eight words of buffer were not possible, so the credit system was not used. The problem of connection is very sufficient, transmitting part doesn't know if connection is established. If an error occurred during transmission, the receiver part was sent to reset state and after reset it had to establish connection again, whereas the connection had to be established on speed 10 MHz. These problems are solved in our SpaceWire controller.

Simplex mode is designed for one-side transmission or receiving data. This mode let us minimize the number of transmitting or receiving cable, reduce the number of gates of the block, thus reducing the weight of the whole block. The block with simplex mode included, give us the number of advantages, though there are some problems to solve, such as establishing the connection, because in normal block connection is established using two directions. The crediting system can't be used as it is in SpaceWire specification.

Our modification allows to use the simplex mode. Considerable changes in the standart SpaceWire were not made. Just one part of the standart Spacewire is modified – state machine. The new state machine will help us to use the simplex mode depending on signals designed for simplex mode. Using only two new signals – tx_simplex_enabled and rx_simplex_enabled will allow us to turn this mode on. If this two signals will not be set to active level – state machine will work as an ordinary state machine of the SpaceWire standart. On figure 1 our new state machine is listed.

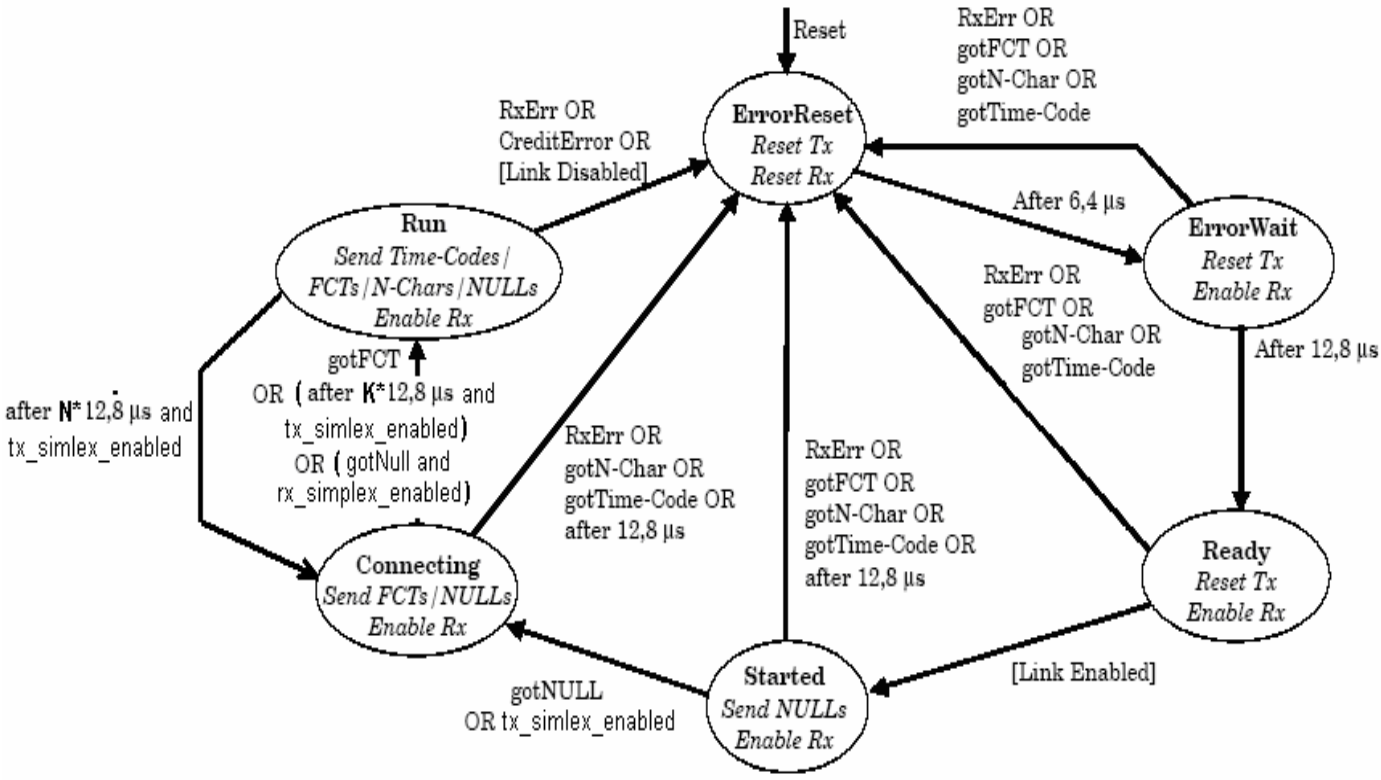


Figure 1. Modified state machine

All modifications of this state machine are made after state "Started". Also signal Link Enabled will be formed a little bit different from original state machine.

As we mentioned before – there are two types of the simplex mode – transmitting simplex mode and receiving simplex mode. Let us examine the behavior of the state machine in two types of simplex mode.

Receiving simplex mode is enabled by setting the active level of signal on line rx_simplex_enabled. In this mode the block will only receive data and/or time codes. Firstly state Machine will be in ErrorReset State. After 6.4 microseconds the block will go to ErrorWait state. After 12,8 microseconds it will be in Ready state. Link Enabled simplex in the receiving simplex mode is set only if the Null symbol is received and the AutoStart signal is in active level. The LinkStart signal is not used in receiving simplex mode. Thus, after the receiving of the NULL symbol the state machine will go to the state Started and then to Connecting. There is no sense waiting the FCT symbol in this state, because no data will be sent. So, after the block goes to Connecting state it goes to the Run state. In this state the block will be until the signal Link Disabled is set, or any other error occurred, like in the original state machine. In conclusion I can

say that state machine of the receiving simplex mode is simple – after the Null symbol is received and Autostart is set – it goes directly to Run state.
Transmitting simplex mode is enabled by setting the active level of signal on line tx_simlex_enabled. Firstly state Machine will be in ErrorReset State. No errors can be occurred in this mode. The only way to go to ErrorReset state is setting the Reset signal or Link Disable signal in Run State. After 6.4 microseconds the block will go to ErrorWait state. After 12,8 microseconds it will be in Ready state. The block will be in this state until the signal Link Start is set to active level. In the transmitting simplex mode AutoStart signal is not used. After the signal Link Enabled is set to active level – state machine go to Started state. The state machine goes to Connecting state at once, because there is no receiving channel. State machine will stay in this state for a sufficient time (K*12,8 microseconds). This time can be set by a designer, the only condition is that a NULL symbol must be sent on frequency 10 MHz. Such time-consuming state is made because the receiving must detect the first NULL. This state also will be used for reconnecting, which will be described further. After K*12,8 microseconds the state machine goes to Run state. In this sate the transmitter can send data and time codes. FCT codes are not sent. In this state the block stays for N*12,8 microseconds. After this time state machine goes to Connecting State and the transmitter begin to send only NULL symbols on the frequency 10 MHz. This period is called the period of reconnecting and is made for the receiving block. If an error occurred in the receiving block – the receiving block will go to the ErrorReset state. After some time it will go to the Ready State and will start the Connection only if the transmitting block will send the NULL symbol. The transmitting part doesn't know the situation in the receiving part. That's why such periods of reconnection are made. If an error occurred, the connection can be established again. Maximum time of reconnection period is 12,8*N+12,8*K microseconds.
In conclusion we can say that if the simplex mode enabled the state machine of the receiving part can be in the Run state till doomsday (if no errors occurred), whereas the state machine of the transmitting part will be changing its state, moving from Connecting to Run and from Run to Connection.
The FCT signal is not used at all, that's why on the receiving part the big-sized buffer have to be used, or the reading speed from the buffer must be higher than the writing speed from the receiving part.
Siplex mode can be used as the mode of the block, or the can be the block, using just one channel, receiving or transmitting. If the block will use only one channel, the size of block will be reduced significantly, because the other channel will not be even sinthesized. The use of simplex mode will reduce the power consumption of the block, also because the second channel will not be working.
The maximum traffic capacity of the SpaceWire channel is 320 Mbit per second on the frequency 400 Mhz. This traffic capacity can be achieved in half-duplex mode (when one channel is sending data symbols only and the other – FCT, Time and Null symbols). In full duplex mode traffic capacity is reduced to 305 because of the FCT symbols in each channel. Because of the period of reconnection the traffic capacity is lower in simplex mode than in half duplex mode. On Figure 2 the traffic capacity of the channel in simplex mode (we are using N=8 and K=1).

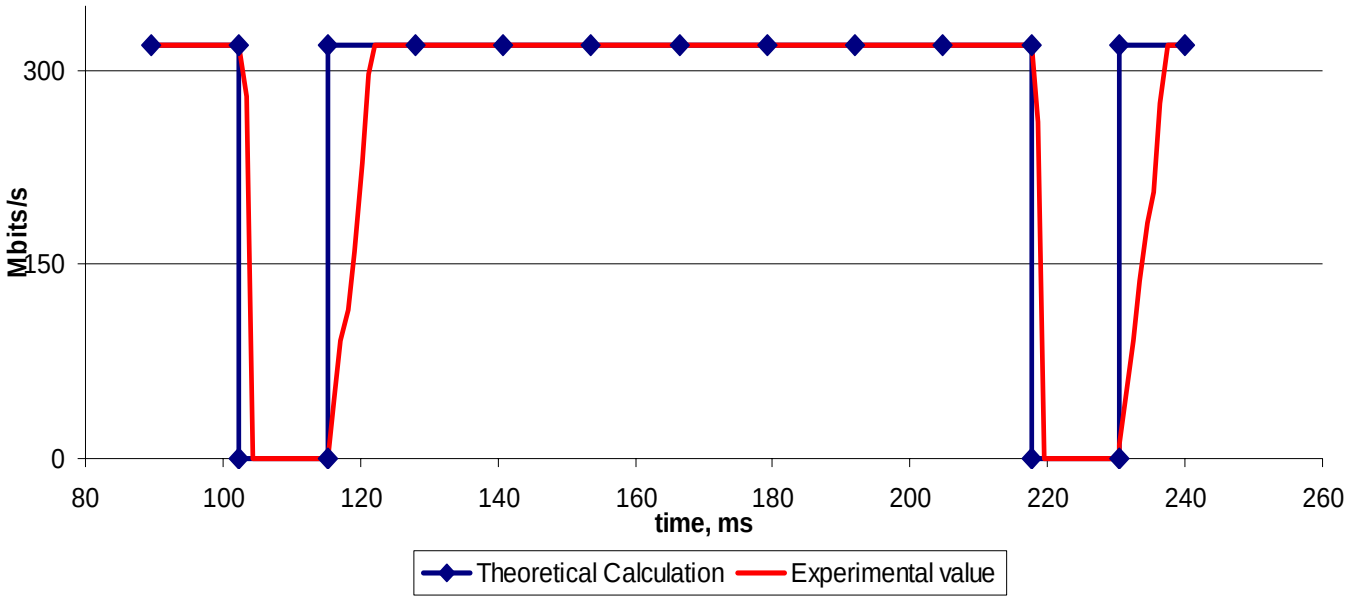


Figure 3 Traffic capacity in simplex mode (frequency 400 MHz).

The block in state Run is sending Data for a period 102,4 microseconds, and is sending Null symbols for 12,8 microseconds. As we can see from practical value – the traffic capacity is not going instantly to 0, the transmitter is sending some data, left in transmitting buffer, and after this it falls to zero. But the frequency is not 10 MHz, the PLL also can't set 10 MHz in a moment, frequency goes down smoothly. When State machine of the transmitting part goes to state Run – traffic capacity also don't go to maximum because of the PLL. Let us look through average statistical value of different modes of transmitting Data. The traffic capacity in simplex mode is significantly lower then in other modes because of the period of reconnection (figure 3).

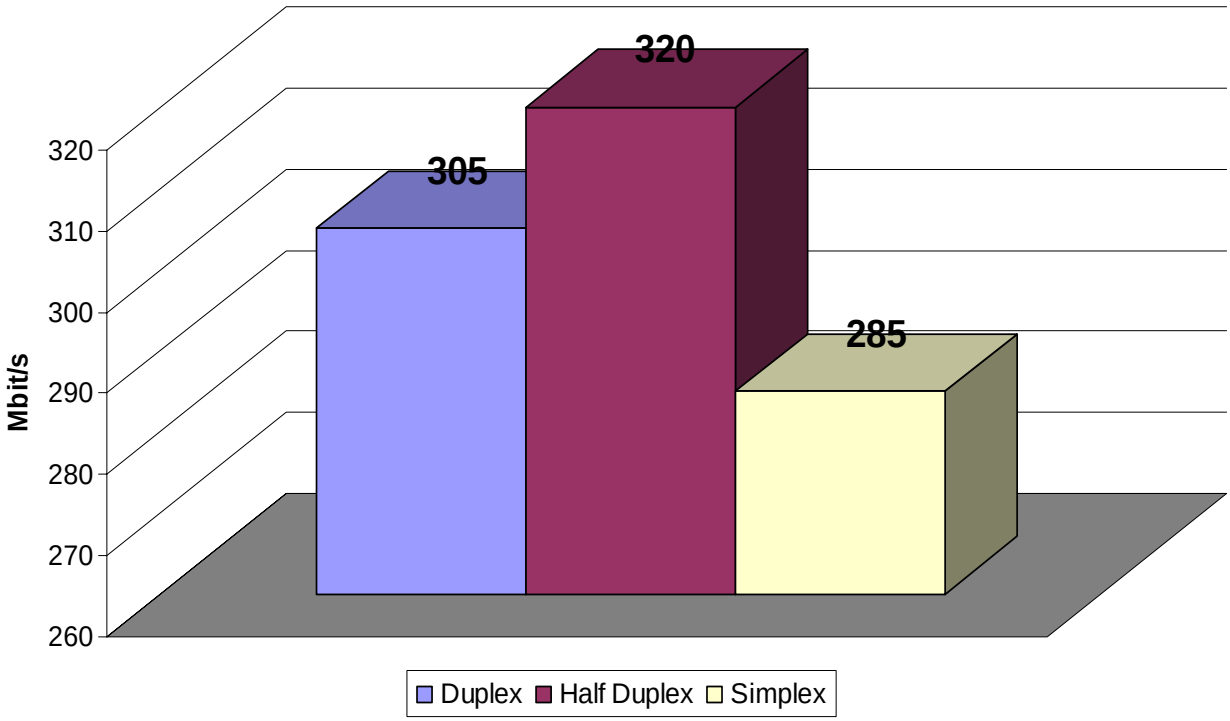


Figure 3. Traffic capacity of the block in different modes

The more is the period of sending data in simplex node and the less is the period of the reconnection – the higher is the traffic capacity, but that will increase the number of data lost if an error occurs. Because of this – parameters N and K must be set due to situation.

Networks & Protocols 1

Tuesday 18th September

13:35 – 16:00

NETWORK MANAGEMENT AND CONFIGURATION USING RMAP

Session: Networks & Protocols

Long Paper

Peter Mendham, Stuart Mills, Steve Parkes

Space Systems Research Group

School of Computing, University of Dundee, Dundee, DD1 4HN, Scotland

E-mail: petermendham@computing.dundee.ac.uk, smills@computing.dundee.ac.uk, sparkes@computing.dundee.ac.uk

ABSTRACT

The success of the SpaceWire standard has resulted in the availability of a wide variety of SpaceWire devices. Allowing these devices to inter-operate easily is an open-problem with growing importance. This problem has two different manifestations: interoperability in ground and test equipment, where ease of use is the main driver; and interoperability of flight hardware where improvements could ease both hardware and software reuse and lower costs.

This paper proposes a standard mechanism for network management and configuration of network devices building on the remote memory access protocol (RMAP). The paper argues that RMAP is an appropriate starting point as it layers well in a protocol stack, and many vendors have existing RMAP capabilities. Adding network management and configuration features in this way could be a fairly easy task for both existing and future SpaceWire equipment.

1 INTRODUCTION

Since the publication of the SpaceWire standard by the European Cooperation on Space Standardization (ECSS) in January 2003 [1], SpaceWire has emerged as one of the main data-handling networks for use onboard spacecraft. It is now being used on many ESA, NASA and JAXA spacecraft and by research organisations and space industry across the world. This widespread interest has resulted in the availability of a large number of SpaceWire devices, however, there is no standard way to interrogate or configure these devices, limiting the extent to which existing hardware can interoperate and both hardware and software can be reused between missions. This paper examines what would be required to ensure a basic level of interoperability, and proposes a solution using the Remote Memory Access Protocol (RMAP) [2].

The next section discusses interoperability, and its implications, for both ground and flight. The outcomes of this discussion serve as a set of requirements for the rest of the paper. The following section gives a brief overview of the RMAP protocol and presents the argument for using it here. The paper then presents the technical details of the proposal examining the two different parts of the interoperability problem separately: network management and network configuration.

This work has a close relationship with the proposed Plug-And-Play Standard for SpaceWire; the penultimate section of the paper clarifies the relationship and presents a small amount of historical background. The final section summarises and concludes the discussions.

2 SPACEWIRE INTEROPERABILITY

One of the greatest assets of SpaceWire, and perhaps the most important reason for its popularity, is its inherent simplicity. SpaceWire CODECs and routers require relatively few gates, saving cost, mass and power. SpaceWire packets are simply the routing addresses, followed by the data cargo and an end of packet marker. SpaceWire links are relatively simple; at all times they must be in one of six states, furthermore, the standard gives clear indications of the ways these states should be controlled. As part of the standardisation of SpaceWire addressing, the SpaceWire standard gives a clear discussion of a many features that routers must, or can, support. Despite this, there is no standard way to interrogate, or configure these features. A vendor-agnostic method for managing the standard features of SpaceWire Devices would bring a wide range of benefits.

2.1 GROUND AND TEST EQUIPMENT

Interoperability standards would provide benefits for ground-based systems in two main areas: development and test. In the first case, a developer may wish to set up a SpaceWire network for prototyping and simulation. It is likely that the network will use equipment supplied by a number of different vendors. Currently, this means that vendor specific software, or other mechanisms, must be used to configure each of the devices. Should any network management functions be required, such as detecting the status of individual links, or the topology of the network, vendor specific functions are again required, in some cases a single operation that involves multiple devices may be forced to use multiple software libraries from different vendors, each with their own conventions.

By providing standard methods for the interrogation of SpaceWire networks, test equipment can discover network topologies and provide diagnoses on network problems in real time. Such features could be especially useful during final spacecraft integration. With knowledge of vendor-agnostic methods of interrogating routers and devices, electrical ground support equipment (EGSE) could identify and pinpoint problems with flight devices, and network configuration.

2.2 FLIGHT EQUIPMENT

Although a spacecraft is generally a closed system, similar inter-operability problems are faced. If flight devices from multiple vendors are used, vendor specific mechanisms must be employed to, for example, configure routing tables. This limits the degree to which software and other spacecraft components can be re-used. Standard methods for interrogating a network could permit flight software to easily determine the current status of SpaceWire links to diagnose problems or confirm the availability of a device, perhaps in response to a mode change. Utilising a vendor-agnostic standard for these purposes reduces the amount of testing that must be done, lowers risk and increases the amount of reuse possible within, and between, missions.

3 THE REMOTE MEMORY ACCESS PROTOCOL (RMAP)

The Remote Memory Access Protocol is a simple yet flexible method for querying and configuring a SpaceWire device. RMAP provides commands for:

- Write
- Read
- Read/Modify/Write

Additionally, RMAP provides support for an 8-bit CRC which may optionally be used to check the data contents of a write operation (a verified write). The source of a write command may also request an acknowledgement (an acknowledged write) which will return a standard status code. The two may be combined to form a verified, acknowledged write. A read operation is straightforward, with the reply packet containing the requested data.

Read/modify/write (RMW) operations are slightly more complicated. An RMW packet contains two fields: a data value and a mask value. The operation first performs a read, and responds with the data. It then uses the mask and data fields to perform a write in an implementation specific way.

All RMAP commands use an address field to specify the data on which to perform the operation. As there are no restrictions on the values in this field, the protocol imposes no semantics above the three command types.

Additionally, RMAP provides:

- A transaction identifier to permit the tracking of transactions by the host requesting the operation. This requires no extra logic at the device.
- A destination key, essentially a 'magic cookie', included in every command must be matched by the device before the operation is allowed to continue.

Either path or logical addressing may be used with RMAP.

3.1 BENEFITS OF RMAP

Read and write operations form the basis for many protocols and, combined with the verification and acknowledgement facilities, RMAP represents a versatile lowest layer in a protocol stack. The lack of complex semantics ensures the protocol's flexibility. If facilities, such as transaction identifiers and destination keys, are not required the corresponding logic may not be implemented and the overhead in packet size is minimal.

A number of manufacturers and organisations have found RMAP useful and have implemented hardware and software solutions for devices and hosts. Any protocol using RMAP as a lower layer would be able to leverage existing intellectual property, limiting the risk for adopters of new technology.

3.2 Interaction with the Protocol Identifier

RMAP works within the proposed Protocol Identification standard [3], which places a centrally assigned numerical identifier after the address. If a logical address is not used, a byte of padding (essentially a fake logical address, usually with the value 254) must be inserted so that the packet arriving at the device is consistent.

To configure or manage routers, the standard specifies that port zero is reserved for this purpose. To permit the interrogation of all devices, without *a priori* knowledge of their type, nodes must also respect a packet arriving as if it is destined for port zero. In this case a node should recognise the packet as a configuration packet, if it supports that feature, or it should discard the packet. This behaviour has been proposed as an alteration to the Protocol Identification standard.

RMAP has a dedicated protocol identifier of 01h. Unfortunately, whilst this identifies the packet contents as RMAP, it does not describe any semantics imposed on the RMAP commands. For this reason, this proposal uses an alternative protocol identifier, which specifies network management and configuration using RMAP.

3.3 Specifying an RMAP Return Address

If a host is sending an RMAP read command to an unknown device in order to, for example, determine its type, the host may not know the port number to which it is connected. In this case it is impossible for the host to specify a return address. To solve this problem, we propose that any packet requiring a response which specifies a Source Logical Address of zero will have its return address modified to include the path out of the port through which the request was received.

3.4 Use of RMAP Address Space

RMAP has a 40-bit address field (including the extended address byte) of which this proposal uses only a fraction. To ease both packet formation and decoding, the address parameter is split into three fields, each a byte long. These form the lowest three bytes of the address field; the upper two bytes are not used. The fields are:

- A command field, which identifies the structure or table being addressed.
- An index field, which specifies an entry in a table; this is not used for structures.
- A byte offset field which permits a read or write command to offset into a structure or table entry.

Addressing is therefore byte-wide in this proposal.

4 Network Management Tasks

Network management, in this context, involves a network manager detecting or verifying the topology of the network, the devices that are connected, and the status of the SpaceWire links that connect them. This section first reviews the parameters that would be useful for network management and then proposes data structures for storing the parameters. In this proposal, all network management parameters are read-only.

4.1 Device Information Parameters

Device information permits network managers to identify SpaceWire devices, and their capabilities. The device would need to specify the following:

- Whether the device is a node or a router.
- The number of ports the device has.

Additionally, a device may wish to identify itself by specifying:

- A vendor identifier; centrally assigned to be unique for each vendor.
- A product identifier, assigned by the vendor.
- A device class identifier; centrally assigned to describe the function of this device.
- Version information.

It may also be important to uniquely identify a device on a network. This could be achieved by specifying:

- A globally unique device identifier.

For maximum flexibility, a vendor may decide not to assign this parameter, and to allow it to be written to. For this reason, a device identifier is included in the data structures for network configuration (see below).

An RMAP verified write operation requires the device to store the contents of the whole packet so that it can verify the contents against the CRC. Any practical device will have limited buffer space. A device may therefore wish to specify:

- The maximum allowed write packet size.

4.2 Link Status Monitoring Parameters

For network management, a host may wish to determine the following:

- The status of all links attached to this device (whether or not they are in the ‘run’ state).
- Whether any errors have been detected on the links.
- The maximum speed the links may be run at.

The SpaceWire standard specifically mentions the following errors:

- Disconnect errors.
- Parity errors.
- Escape errors.

- Transmit credit errors.
- Receive credit errors.
- Character sequence errors.

The maximum speed of a link must be greater than or equal to 10 Mbit/s, the start-up speed of SpaceWire links.

4.3 Network Discovery Parameters

Using the device information presented so far, a host may begin to explore the network and determine what devices are present. To identify the route that the host is using to interrogate the device, the device should make available:

- The port number through which the current request was received.

4.4 Router Information Parameters

During packet routing, routers may use a variety of arbitration mechanisms to determine access to an output port between competing packets. It may choose to use priorities for physical ports, logical addresses, or both. A router should therefore specify:

- The maximum priority that may be used for physical port arbitration.
- The maximum priority that may be used for logical address arbitration.

4.5 Device Information Structures

The parameters introduced above are organised into two structures:

- A device information structure for parameters that are common to both nodes and routers.
- A router identification structure for routers only.

These structures are shown in Figure 1 and Figure 2 respectively. The “Active Port Bitmask” in Figure 1 identifies which ports are in the ‘run’ state. Bit 1 set to a ‘1’ indicates that port 1 is running. Bit 0 is not used and is always set to a ‘0’.

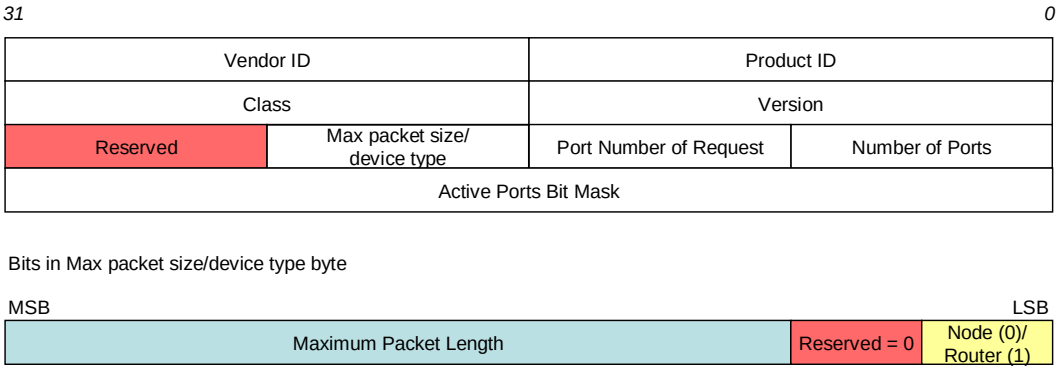


Figure 1: Device Information Structure

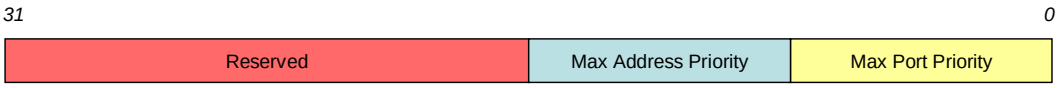


Figure 2: Router Information Structure

4.6 PORT STATUS TABLE

The port status table contains 32 entries, one for each physical port from 0-31. The first entry, referencing the configuration port, is not used. A port status table entry is shown in Figure 3.

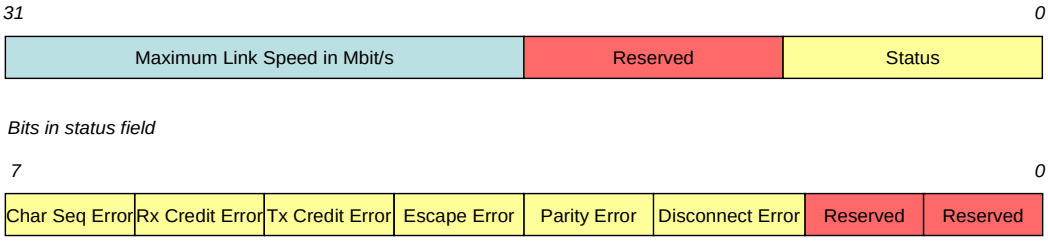


Figure 3: Port Status Table Entry

5 NETWORK CONFIGURATION TASKS

During network configuration a host configures the device for operation. For all devices, a host must be able to configure each of the SpaceWire links. For routers only, the host must be able to configure the way in which arbitration is carried out, and to set-up the routing table.

5.1 LINK CONFIGURATION

To configure a link, a host must be able to:

- Reset each of the errors in the port status table.
- Set the state of the link.
- Set the speed of the link.

The state of a link may be:

- Idle – if the link is not running, it won't start; if it is running it will continue to run.
- Start – start the link.
- Auto-start – listen for NULLs and start the link if they are received.
- Disable – if the link is not running, it won't start; if it is running it will be stopped immediately.

The link speed may be set to any value between 10 Mbit/s and the supported maximum. A device may choose to alter the value to the nearest supported speed below the one specified.

For arbitration, routers may need to associate a priority with each link (see below).

5.2 ROUTER CONFIGURATION

A routing device may be configured to set the arbitration mode that it uses to arbitrate between multiple packets competing for the same output port.

Using information provided in the SpaceWire standard, the arbitration of packets competing for an output port can be decided using a combination of up to three techniques, applied in the order specified below:

1. Address Priority uses the destination address of the received packet. Arbitration is carried out by comparing the priorities assigned to the destination addresses in the routing table (see below). The highest arbitration priority wins.
2. Port Priority uses the port at which the packet arrives. Arbitration is carried out by comparing the priorities assigned to the arrival port in the port configuration table (see below). The highest arbitration priority wins.
3. If competition still exists, an arbitration mode can be chosen by selecting a arbitration method such as random, round robin, fixed, etc.

A host should be able to enable and disable these features individually. If a router chooses not to support one or more of these methods the host will not be able to enable that feature. A router must support fixed arbitration as a minimum.

To enable logical address-based routing, a router must allow a host to associate the following with each logical address:

- One or more physical ports, more to permit group adaptive routing or packet distribution.
- The option to remove the logical address from the packet.
- The option to enable packet distribution where the packet is forwarded to every port in the group rather that only one. If the arrival port is included on the list the packet is only forwarded to other ports.
- A routing priority.

5.3 PORT CONFIGURATION TABLE

All devices have a port configuration table with as many entries as the device has physical ports. There is no entry for the configuration port. Entries are as shown in Figure 4.

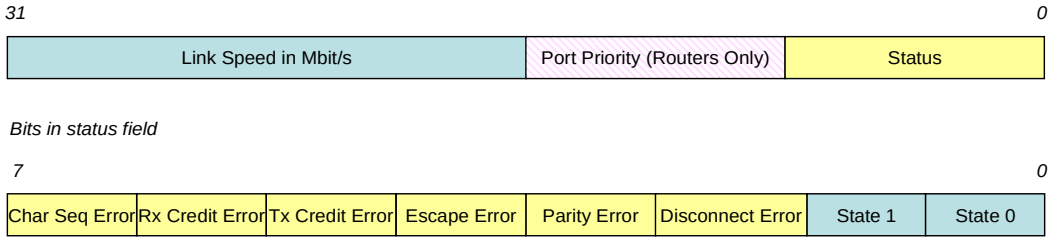


Figure 4: Port Configuration Table Entry

The status field is a copy of that given in the equivalent port status stable entry, except that writing a '1' to an error bit resets that error. The two state bits are encoded as follows:

0. Idle
1. Start
2. Auto-start
3. Disable

The port priority field is reserved for nodes.

5.4 ROUTER CONFIGURATION STRUCTURE AND ROUTING TABLE

To configure the arbitration mode on routers, a router configuration structure is provided; this is shown in Figure 5.

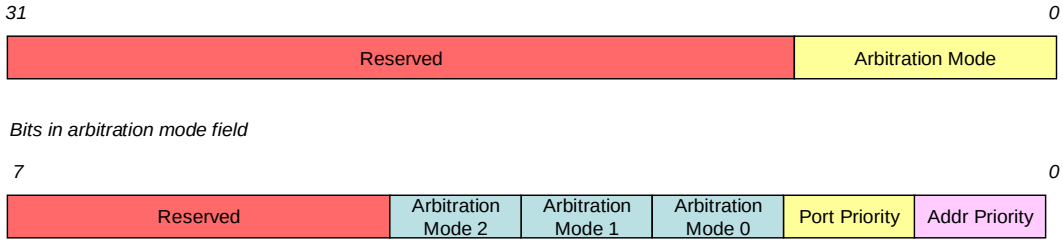


Figure 5: Router Configuration Structure

The arbitration mode field decodes as:

0. Fixed arbitration
1. Round-robin arbitration
2. Random arbitration

All other values are vendor specific.

The routing table has entries which control the routing of packets for logical addresses (see Figure 6). The most important field is the port association bitmask, which indicates the ports that are to be used for routing the selected logical address. Bit 1 indicates port 1 and so on. As routing is not permitted to the configuration port, the lowest bit is used to specify address deletion. A routing table entry is shown in Figure 6. An entry is provided for all addresses from 1 to 254.

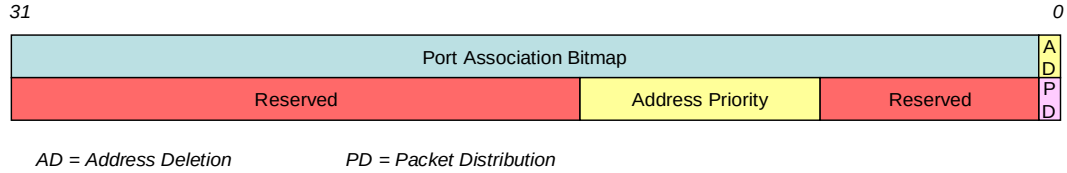


Figure 6: Routing Table Entry

6 Relationship with the Proposed Plug and Play Standard

This work has two starting points: the configuration space of the University of Dundee router, which is accessed using RMAP; and the proposals produced by the Plug-and-Play working party, a sub-group of the SpaceWire Working Group. The current proposal is accessible from the working party's discussion forum [4]. Although heavily indebted to the work that the plug-and-play group has done, this proposal has a slightly different focus, which is why the term plug-and-play has not been used here.

7 Conclusion

This paper argued the need for a standard method for network management and configuration in order to promote interoperability between SpaceWire devices from different vendors. Such interoperability would make equipment easier to use, permit operations such as network discovery in a consistent manner and enable higher levels of software and hardware reuse. As a network management and configuration protocol is largely based on the semantics of read/write (or set/get), this paper argues that RMAP, a protocol with increasing levels of vendor support, would be an appropriate starting point. To permit identification of the semantics imposed by the network management and configuration protocol, a distinct protocol identifier should be used.

The paper then identified the key features that could be considered as standard, derived from the SpaceWire standard itself, such as link status and speed, and router configuration. Facilities are also provided to permit network discovery.

This work has derived partially from the University of Dundee router configuration space, and partly from the draft specification produced by the SpaceWire plug-and-play working party. The next steps for this work are to: consider existing devices, and whether any special support is needed; include support for sharing the configuration of SpaceWire devices between multiple hosts without introducing conflicts; aligning this proposal closer with the work being done by the plug-and-play working party.

8 REFERENCES

1. European Cooperation for Space Standardization, Standard ECSS-E-50-12A, “SpaceWire – Link, Nodes, Routers and Networks”, Issue 1, European Cooperation for Space Standardization, January 2003.

2. European Cooperation for Space Standardization, Standard ECSS-E-50-11, “Remote Memory Access Protocol”, Draft F, European Cooperation for Space Standardization, December 2006.

3. European Cooperation for Space Standardization, Standard ECSS-E-50-11, “Protocol Identification”, Draft B, European Cooperation for Space Standardization, December 2006.

4. SpaceWire Plug-and-Play Working Party, “SpaceWire Plug-and-Play Specification”, http://tech.groups.yahoo.com/group/SpaceWirePnP/files/Draft%20Specification/, 2007

PROPOSED SOIS PLUG-AND-PLAY ARCHITECTURE AND RESULTING REQUIREMENTS ON SPACEWIRE MAPPING

Session: SpaceWire Networks and Protocols

Long Paper

Stuart D. Fowell

SciSys UK Ltd,
Clothier Road,
Bristol,
BS4 5SS,
UK

Chris Taylor

European Space Technology Evaluation Center,
2200 AG Noordwijk,
The Netherlands

E-mail: stuart.fowell@scisys.co.uk, chris.taylor@esa.int

ABSTRACT

This paper describes the initial concept of the proposed Plug-and-Play architecture of the SOIS area of the CCSDS, and the resulting requirements on the mapping of SOIS onto SpaceWire. Firstly, this paper defines the term "plug-and-play" in the scope of SOIS and a number of SOIS use cases for "plug-and-play", so as to derive requirements that must be met by a SOIS Plug-and-Play architecture. Secondly, this paper proposes a tentative SOIS Plug-and-Play architecture, based on initial analysis of the use cases and consideration of existing plug-and-play technologies, e.g. USB 2.0, IEEE 1451, 1-wire as well as proposals such as those already made for SpaceWire. Finally, this paper proposes draft requirements of the mapping of the SOIS Plug-and-Play architecture onto SpaceWire. As this is an initial concept paper, it is hoped that it will generate debate and feedback on the SOIS Plug-and-Play initiative that will, of course, be gratefully received and taken into account.

1 EXISTING SOIS ARCHITECTURE

The Consultative Committee for Space Data Standards (CCSDS) [1] was founded in 1982 by the major space agencies in the world to discuss and define common space communications issues, to enhance governmental and commercial interoperability and cross-support, while also reducing risk, development time and project costs.

As part of the CCSDS work, the Spacecraft Onboard Interface Services (SOIS) area is developing standards to radically improve the spacecraft flight segment data systems design and development process by defining generic services that will simplify the way flight software interacts with flight hardware and permitting interoperability and reusability both for the benefit of Agencies and Industrial contractors. As part of the standardisation process for SOIS, a subnetwork-neutral architecture of services has

been defined [2], as illustrated in Figure 1. Mappings of these services to capabilities of specific subnetworks are then defined, e.g. protocols on SpaceWire, MIL-STD-1553B and CAN. This allows, amongst other benefits, for satellite architectures to be re-used across different busses and standardised off-the-shelf devices and subsystems to be developed.

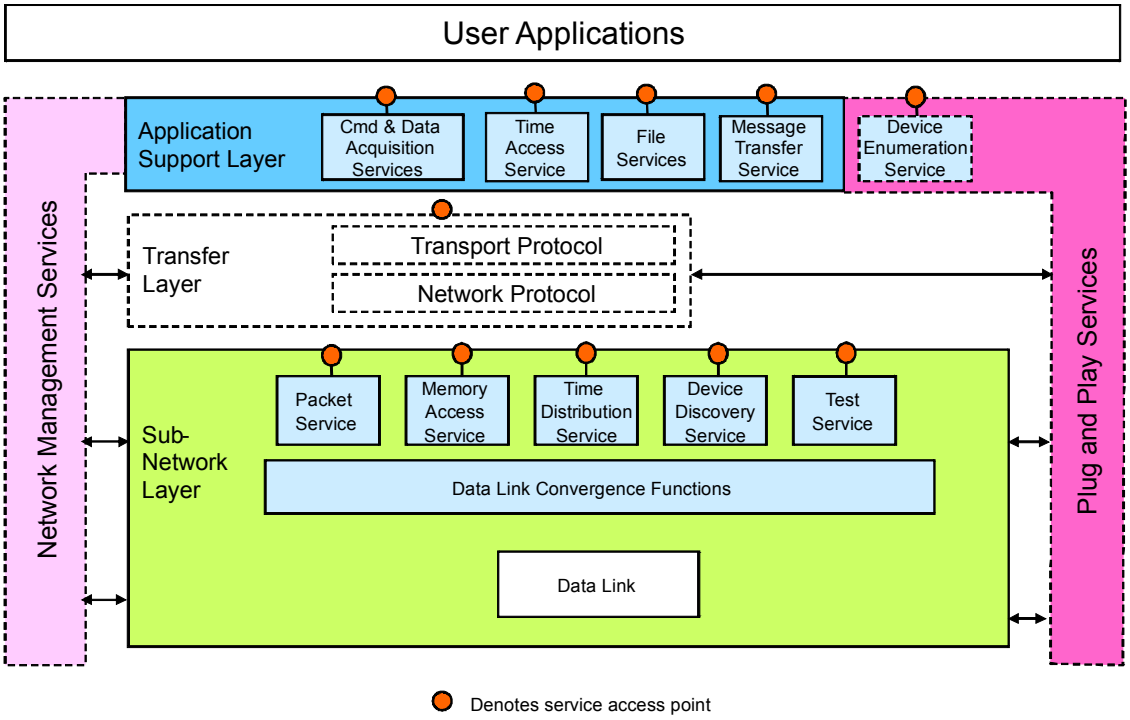


Figure 1: CCSDS SOIS Architecture

Of relevance here are the following services:

- **Command and Data Acquisition Services**, that provide mechanism for commanding of and acquiring data from devices within a spacecraft;
- **Message Transfer Service**, that provides transfer of messages between software applications within a spacecraft;
- **Packet Service**, that provides transfer of packets between data systems[1] within a subnetwork of a spacecraft;
- **Memory Access Service**, that provides access to memory locations of a data system from another data system within a subnetwork of a spacecraft.

The first set of standards is currently being reviewed by the various Space Agencies. The ECSS are currently developing protocols to provide the mappings onto

[1] Within ISO standards, communication is defined as being between "data systems", a generic term that can be taken, within the SpaceWire context, as mapping onto a node within a SpaceWire network. A node is defined as any addressable SpaceWire network entity, be it a processing, IO, or memory module, a transducer, an instrument etc.

SpaceWire and MIL-STD-1553B and a similar exercise is planned in 2008 for CAN. For SpaceWire, the RMAP protocol is suitable for the SOIS Memory Access Service and the SpaceNet project is prototyping a SpaceWire packet protocol for the SOIS Packet Service.

However, these standards only address a static or top-down configured communications architecture. In addition, support for "wireless" capabilities is being considered and by their nature can result in a more dynamic communications architecture.

To address these issues, it has been identified that the SOIS architecture needs extending to support "plug-and-play" concepts and a "Birds-of-a-Feather" (BoF) grouping has been organised to address this. This paper summarises the current position of the SOIS Plug-and-Play BoF, building on previous positional papers [3, 4].

2 PLUG-AND-PLAY REQUIREMENTS

Before we can define a SOIS Plug-and-Play architecture and its mapping onto SpaceWire, it is important to clarify the term "plug-and-play" and define appropriate use-cases within the SOIS domain. From these can be extracted requirements on the SOIS plug-and-play architecture.

2.1 Definition of "Plug-and-Play"

The Wikipedia definition of "plug-and-play" is as follows:

> **"Plug and play** is a computer feature that allows the addition of a new device, normally a peripheral, without requiring reconfiguration or manual installation of device drivers. … Modern plug-and-play includes both the traditional boot-time assignment of I/O addresses and interrupts to prevent conflicts and identify drivers, as well as hotplug systems such as USB and Firewire."

Translated to a spacecraft domain, "peripheral" should include onboard computing modules such as processing, IO and mass memory modules, as well as devices traditionally associated with avionics, from this simple (e.g. thrusters, magnetometers, thermistors) to the more complex (e.g. star trackers), and simple instruments. It has been considered that perhaps plug-and-play should extend to including the integration of whole sub-systems, but has been discounted as this is where the line between where device integration and software system integration occurs, and so is beyond the scope of SOIS. This limits SOIS Plug-and-Play to the establishment of communication of the SOIS Command and Data Acquisition Services of the Application Support Layer and the Subnetwork Layer Services.

Therefore, we limit the definition of "plug-and-play" in a SOIS context to:

> "the mechanisms necessary to establish communication services between two data systems in a spacecraft's onboard (sub-)network, without requiring reconfiguration or manual installation of device drivers by any user (higher-level service or OBSW application)."

2.2 SOIS Plug-and-Play Use Cases

In conjunction with defining the term "plug-and-play", the use cases to be solved by plug-and-play within the SOIS domain need to be captured. The following is a summary list of those use cases identified to date:

- **Dynamic Spacecraft Network Reconfiguration** – activation of redundant devices upon a flying spacecraft in response to faults. A Fault Detection, Isolation and Recovery (FDIR) system application simply powers up replacement. Reconfiguration happens automatically (bottom-up), rather than hierarchical (top-down);

- **Spacecraft Integration & Test** – Electrical Ground Support Equipment (EGSE) connection to Spacecraft under test using wireless technologies;

- **Rapid Spacecraft Assembly of Devices** – to reduce/eliminate the need for aspects of Spacecraft database for configuring OBSW;

- **Biometric Health Monitoring of ISS/Orbiter crew** – characterised as facilitating the incorporation of heterogeneous sensing and control devices in a wireless, heterogeneous communications network [8].

It is important to also identify the use cases that are out-of-scope for SOIS Plug-and-Play (though that is not to say that SOIS Plug-and-Play may not have a role to play within them):

- **Onboard Software Upgrade or Reconfiguration** – covering mode changes or software updates. This is purely a software change with no new data systems introduced, so there is no reconfiguration of the SOIS communication services required, and so out of scope of SOIS Plug-and-Play.

- **Rapid Spacecraft Assembly of Subsystems** – while SOIS Plug-and-Play will simplify at the subnetwork layer the integration of subsystems with other subsystems and/or complex instruments, integration also requires a complex exchange of information using perhaps a software framework or middleware that is beyond the present scope of SOIS. However, such a software framework would exchange messages using the SOIS Message Transfer Service. Therefore SOIS Plug-and-Play greatly aids but itself does not fully solve this use case.

2.3 SOIS Plug-and-Play Requirements

From these uses cases, a tentative set of requirements of SOIS plug-and-play can be extracted: the SOIS Plug-and-Play architecture shall:

1. support a mechanism to discover new data systems added to (powered up, mechanically inserted, sending announcement packet) a SOIS subnetwork;

2. support a mechanism to discover old data systems removed from (switched off, failed, mechanically removed, out of range, electing to withdraw) a SOIS subnetwork;

3. support a mechanism to discovery of capabilities of added data systems;

4. support a mechanism to reconfigure SOIS communication services to allow communication to and from added data systems;

5. support a mechanism to reconfigure SOIS communication services to remove information about removed data systems;

6. support a mechanism to notify users (applications and higher layer services) of added and removed data systems and their capabilities.

These are a preliminary set of requirements, which are expected to be expanded and refined over the process of defining the SOIS Plug-and-Play architecture within the multi-agency discussion and debate that is inherent in the CCSDS approach.

3 TENTATIVE SOIS PLUG-AND-PLAY ARCHITECTURE

Having established the requirements for a SOIS Plug-and-Play Architecture, the architecture itself must be designed. Within SOIS, the "adopt-adapt-innovate" approach is used, i.e. adopt an existing standard or technology if it meets the requirements, otherwise adapt (i.e. modify) an existing standard or technology if it can be changed to meet the requirements, and failing either, innovate (i.e. develop) a novel technology to meet the standards.

To do this then, a survey of existing standards, technologies, and prototypes is required.

3.1 Existing Plug-and-Play Technologies and Studies

From work sponsored and directly performed by NASA and ESA, the following plug-and-play technologies have been identified:

- **USB 2.0** – ubiquitous serial bus for data exchange between host computers (typically PCs) and a wide range of simultaneously accessible peripherals. The bus allows peripherals to be attached, configured, used and detached while the host and other peripherals are in operation. USB provides most of the characteristics required for plug-and-play. However, it has a rigid topology and allows for only a single host computer [5].

- **IEEE 1451** – a standard for a Smart Transducer Interface for Sensors and Actuators, designed to ease connectivity of sensors and actuators into a device or field network. Uses a common object model for smart transducers along with a smart transducer interface module (STIM), a transducer electronic data sheet (TEDS), and a digital interface to access the data. A variety of standards within this set define access to analogue, digital and smart transducers across a variety of communication protocols, including wireless [6].

- **1-wire** – a device communication bus system from Dallas Semiconductor that provides low-speed data, signalling and power over a single wire. The bus includes a mechanism for recovering the address of every device on the bus. As the device address includes the device type so that this can produce an inventory of all devices on the bus [7].

- **wireless technologies** – by their very nature of integrated independent data systems, wireless technologies support plug-and-play concepts. The architectures of wireless technologies are still being considered for their inclusion within SOIS Plug-and-Play.

In addition, a number of projects have produced prototypes of plug-and-play architectures and/or technologies:

- **BioNet** – BioNet is a network-transparent device-driver and application-client framework. It is a general middleware solution for the integration of disparate data-producing endpoints over heterogeneous wired and wireless networks. The BioNet architecture is a highly-scalable, decentralized, asynchronous, publish/subscribe message bus [8].

- **SpaceWire Plug-and-Play Prototyping** – within the SpaceWire community, a working group coordinated by NASA GSFC has been established and a number of prototyping initiatives for supporting plug-and-play within SpaceWire networks, primarily focussed on network mapping and router (re-) configuration [9]

Of course, it is a requirement on SOIS that it is independent, but mappable onto, network specific plug-and-play technologies.

3.2 Proposed SOIS Plug-and-Play Architecture

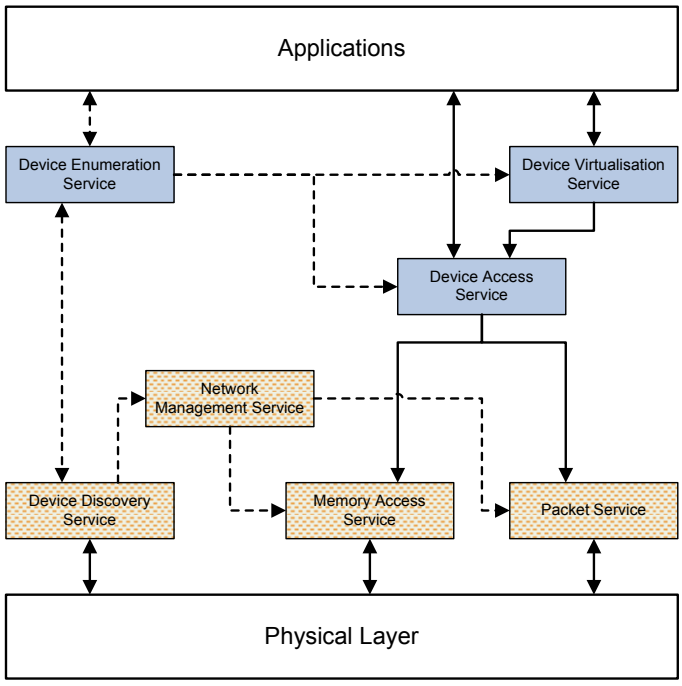


Figure 2: Proposed CCSDS SOIS Plug-and-Play Architecture

The following is a tentatively proposed SOIS Plug-and-Play Architecture, see figure 2. It is primarily focussed on device plug-and-play with particular consideration for wired devices. It should be expected that this will evolve as plug-and-play of more generic data systems and wireless "busses" are considered.

At the heart of the SOIS Plug-and-Play architecture is the **Device Enumeration Service**, which is responsible for managing the discovery of a new device and its insertion into the SOIS communications architecture.

A Subnetwork **Device Discovery Service** is used to discover new devices. This may implement a specific discovery mechanism, e.g. by broadcasting for new devices, or react to a subnetwork-specific event, e.g. a trigger that a new device has been powered up or inserted into the subnetwork. It is assumed that this service is also responsible for allocating or obtaining a subnetwork-specific address for the new device. This allows the Subnetwork Layer Services to be reconfigured to allow communication from within the SOIS communications architecture with the new device.

It is expected that a Subnetwork **Network Management Service** (to be provided as part of subnetwork protocol definitions) is responsible for any reconfiguration of the Subnetwork Layer services that may be required, e.g. updates to SpaceWire Router GAR Tables.

Another key concept is the provision by each SOIS Plug-and-Play-enabled device of an **Electronic Data Sheet** (EDS), that defines the device type and capabilities (e.g. functions, protocols and classes-of-service supported). This, together with the subnetwork-specific address of the new device, allows the Command and Data Acquisition Services and/or Message Transfer Service to be reconfigured to allow application-level communication with the new device. It is assumed that either the Subnetwork Memory Access or Packet Service will be used to obtain the device's EDS.

So, to summarise the steps required to "plug in" a new device (i.e. power up):

1. Device is powered up.

2. Subnetwork specific Device Discovery Service discovers the device and either discovers its address or allocates it.

3. The subnetwork specific Network Management Service performs any necessary configurations to allow communication with the device from any other data system in the subnetwork.

4. The Device Enumeration Service is notified of the new device, including its address. The Device Enumeration Service uses the subnetwork specific Memory Access Service to read the device's EDS to discover its capabilities, e.g. device class. The device may be configured to a default setting.

5. The Device Enumeration Service configures the Device Virtualisation Service and Device Access Service so that users may command and acquire data from the device.

6. Finally the Device Enumeration Service notifies a registered OBSW application that a new device has been plugged into the system.

This approach allows for both static and dynamic systems to be deployed, where the dynamic SOIS Plug-and-Play services manipulate the configuration of the potentially static services.

4 REQUIREMENTS ON SPACEWIRE MAPPING

Given the current status of the SOIS Plug-and-Play initiative, anything written in this section of the paper must be considered speculative!

To date the SpaceWire community have focused on network mapping and discovery of attached and detached nodes. These map nicely to the functionality of the SOIS Device Discovery Service.

Beyond this, the Device Enumeration Service will require a standardised mechanism to discover the capabilities of a node is required, i.e. provision of the device's EDS. RMAP provides a generalised method that can be used to access the EDS data. However, where is EDS located, what address is it at? Is this standardised or perhaps dynamically obtained? In addition, the content of the EDS (structure, types, etc.) remains to be defined. Probably it will not be specific to SpaceWire.

However, one thing is certain. The SOIS Plug-and-Play Architecture has to date primarily focussed on SpaceWire. As it develops, it must encompass other bus types, both wired and wireless. The diverse requirements and technologies must be taken into account in the difficult balancing act between the contradictory requirements of the genericity of the SOIS architecture and the resource constraints of actual spacecraft.

REFERENCES

1. Consultative Committee for Space Data Standards, www.ccsds.org.

2. "Spacecraft Onboard Interface Services – Informational Report", CCSDS 850.0-G-1, Green Book, Issue 1.0, Washington, D.C, June 2007.

3. David P, "Online Remote Maintenance", CCSDS Fall 2003 Working Meeting, Location, 20th October 2003 (available from author).

4. Plummer C, "Plug and Play for Spacecraft Onboard Systems", CCSDS Spring 2005 Working Meeting, Athens, Greece, March 2005 (available from author).

5. "USB 2.0", http://www.usb.org/home.

6. IEEE 1451 Standard for a Smart Transducer Interface for Sensors and Actuators, http://ieee1451.nist.gov/intro.htm.

7. "1-Wire", http://www.1wire.org/.

8. Gifford K, Kuzminsky S, Williams S, Saiz J, "BioNet: a developer-centric middleware architecture for heterogeneous devices and protocols", IEEE Wireless Communications and Networking Conference, Las Vegas, NV, USA, 3rd-6th April 2006.

9. McGuirk P, "SpaceWire Plug-and-Play", 2006 MAPLD International Conference, Washington, D.C, USA, September 2006.

APPLICATION OF THE SPACEWIRE PLUG-AND-PLAY PROTOCOL

Session: Networks and Protocols

Long Paper

Clifford Kimmery

Honeywell International Space Electronic Systems

13350 US Highway 19 N, Clearwater, FL, 33764

Patrick McGuirk

Micro-RDC Space Microelectronics

8102 Menaul Blvd NE, Albuquerque, NM, 87110

Glenn Rakow

NASA Goddard Space Flight Center Flight Electronics and Radiation Effects Branch

8800 Greenbelt Rd, Code 561, Greenbelt, MD, 20771

Paul Jaffe

Naval Research Laboratory Naval Center for Space Technology

Code 8243, Washington DC, 20375

Allison Bertrand and Robert Klar

Southwest Research Institute

6220 Culebra Rd, San Antonio, TX, 78238

E-mail: clifford.kimmery@honeywell.com, Patrick.McGuirk@micro-rdc.com, paul.jaffe@nrl.navy.mil, Glenn.P.Rakow@nasa.gov, abertrand@swri.org, robert.klar@swri.org

ABSTRACT

The proposed SpaceWire Plug-and-Play (SpaceWire PnP) protocol [SpaceWire PnP working group 2007] provides an infrastructure for network management. The protocol defines a set of common features (i.e., parameters and behaviors) to facilitate recognition of and interaction between SpaceWire PnP devices. Network management involves the automatic discovery of arbitrary network topologies, the configuration of device parameters to establish communication, and the timely detection of changes to the network. Network discovery entails the recognition of the type of each device (i.e., router or node) encountered, the identification of key device characteristics, and the mapping of the connections between devices. The facilities provided by the SpaceWire PnP protocol can be used for network discovery and configuration in several different ways depending upon the specific network management approach. The SpaceWire PnP protocol provides mechanisms for arbitration between multiple network managers to prevent collisions and minimize network traffic in dynamic ad-hoc networks. It also provides support for both polled and asynchronous notifications of changes to the network. Lastly, it offers standard

support for accessing device configuration parameters including those required for Plug-and-Play and those defined by the ECSS-E50-12A SpaceWire standard [ESA 2003] (e.g., routing table, logical address, etc.). We describe each SpaceWire PnP feature and give an example of its use in network management.

1 OVERVIEW OF THE SPACEWIRE PNP PROTOCOL

The SpaceWire PnP protocol provides facilities for identifying and configuring SpaceWire PnP-compatible devices (nodes or routers). The protocol defines exchanges of packets (Read, Response, Write, Reset and Notification) between network managers and network elements to perform network management operations. Each packet is tailored to an operation by the associated data type.

To avoid limiting the functionality of higher-level network manager applications, the SpaceWire PnP protocol is based on a "let software do it" philosophy. For example, Write and Reset transactions do not include acknowledgement packets, so a follow-up Read transaction is necessary to confirm success.

1.1 DATA TYPES AND PARAMETERS

Each SpaceWire PnP data type combines one or more SpaceWire PnP parameters into a convenient structure associated with a specific purpose. A SpaceWire PnP data type is used as the cargo of a SpaceWire PnP packet. Table 1 – SpaceWire PnP Data Types identifies the device parameters included in each data type.

		Data Type														
		Device Description	Active Ports	Device Identifier	Valid Logical Addresses	Arbitration Mode	Port Table Entry	Link Speed Table Entry	Routing Table Entry	Network Manager Identification	Notification Table Entry	Detachment Timeout	Notification Information	xTEDS File	Error	Return Address (implicit)
Device Parameters (Read-Only)																
Parameters	Device Class	√														
	Vendor Identifier	√														
	Subsystem Identifier	√														
	Version	√														
	Ports	√														
General Use Parameters (Read-Only)																
	Incoming Port Number	√														
	Active Ports	√	√										√			
xTEDS Parameters (Read-Only)																
	xTEDS File													√		
Network Discovery Parameters (Read-Write)																
	Device Identifier	√		√												
	Granted Port Number									√						

		Data Type														
		Device Description	Active Ports	Device Identifier	Valid Logical Addresses	Arbitration Mode	Port Table Entry	Link Speed Table Entry	Routing Table Entry	Network Manager Identification	Notification Table Entry	Detachment Timeout	Notification Information	xTEDS File	Error	Return Address (implicit)
	Network Manager Logical Address									√						
Event Notification Parameters (Read-Write)																
Parameters	Detachment Timeout											√				
	Notification Table Size	√														
	Notification Table Entry – Acknowledgement Timeout										√					
	Notification Table Entry – Router Identifier										√		√			
	Notification Table Entry – Notification Address										√					
Network Configuration Parameters (Read-Write)																
Parameters	Logical Address	√			√											
	Valid Logical Addresses				√											
	Arbitration Mode					√										
	Port Table Entry – Arbitration Priority						√									
	Port Table Entry – Link State						√									
	Port Table Entry – Link Status						√									
	Port Table Entry – Link Speed							√								
	Port Table Entry – Maximum Link Speed							√								
	Routing Table Entry – Port Association								√							
	Routing Table Entry – Header Deletion								√							
	Routing Table Entry – Arbitration Priority								√							
	Routing Table Entry – Packet Distribution								√							

Table 1 – SpaceWire PnP Data Types

1.2 PACKET TYPES AND BEHAVIORS

A network manager transmits a SpaceWire PnP Read packet to request a specific SpaceWire PnP data type. The SpaceWire PnP device that receives the SpaceWire PnP Read packet transmits a SpaceWire PnP Response packet to the network return address contained in the Read packet. The SpaceWire PnP Response packet contains the requested data type or the Error data type.

To reduce the chance of conflicting values being written to a device, SpaceWire PnP Write transactions are conditional. The write request fails if the current values of any of the parameters associated with the specified data type do not match the default (reset) values (see Section 2.2.2). A network manager transmits a SpaceWire PnP Write packet containing the specific SpaceWire PnP data type to be modified. The SpaceWire PnP device that receives the SpaceWire PnP Write packet checks the device parameters associated with the data type to determine if all current values are the default (reset) values. If all the parameter values are default, the device replaces them with the packet contents. Since SpaceWire PnP Write packets are not acknowledged, the network manager should confirm the success of the write transaction by reading the data type.

A network manager transmits a SpaceWire PnP Reset packet to reset a specific SpaceWire PnP data type. The SpaceWire PnP device that receives the SpaceWire PnP Reset packet initializes the writable device parameters associated with the data type to the default (reset) value (the network manager issuing the Reset packet does not need to know the default value of the parameters). SpaceWire PnP Reset packets are not acknowledged, so the network manager should confirm the success of the reset transaction by reading the data type.

The SpaceWire PnP Notification packet is optionally transmitted by a SpaceWire PnP router when a notification event is detected. The Notification packet contains the Notification Information data type. The router independently notifies each network manager that has identified itself in the router Notification Table. To ensure reliable event notification, a SpaceWire PnP router can optionally retransmit notification packets periodically until a Read packet requesting the Notification Information data type is received from the network manager.

1.3 SPECIAL CONSIDERATIONS

To identify an unknown device, a network manager reads the Device Description data type. The SpaceWire PnP Device Description data type contains detailed information about the responding SpaceWire PnP device. Since the network manager doesn't know whether the unknown device is a node or a router, the SpaceWire PnP protocol requires that SpaceWire PnP devices support a common packet structure based on the ECSS-E-50-11 Draft B Protocol Identification specification [ESA 2005] augmented with a leading-zero path address.

SpaceWire PnP Read, Write and Reset packets arrive at the destination device with a prefix consisting of single-byte path address (value 0) and a single-byte logical address (value 254 decimal). SpaceWire PnP routers pass the packets to the router configuration port (port 0) for handling. SpaceWire PnP-compatible nodes must accept such packets, but the handling method is implementation-defined.

2 NETWORK MANAGEMENT

Network management can be viewed as consisting of initialization and maintenance phases. Each of these phases performs both network discovery and configuration.

Network initialization can interweave the network configuration and network discovery activities (discover a network element, configure it, and then proceed with discovery). Alternatively, it can be decomposed into a distinct network discovery phase followed by a network configuration phase.

Network maintenance naturally combines the network configuration and network discovery activities when a single network element is added, but can be treated similarly to network initialization when an entire subnet is added.

2.1 NETWORK DISCOVERY

Network discovery is the process of identifying the network topology. It is accomplished by exploring the network, identifying each network element encountered and mapping the connections between the network elements.

The SpaceWire PnP protocol provides the tools for network discovery, but leaves implementation to higher-level network manager applications.

2.1.1 DEVICE RECOGNITION

Device recognition involves two aspects of device identity: (1) the characteristics of the device (e.g., this device is a SpaceWire PnP router with eight ports) and (2) the name of the device (e.g., this is SpaceWire PnP router 25). While the device characteristics are inherent attributes, the device name may be assigned by the discoverer and is a potential source of confusion.

As a network manager explores the network and recognizes each device, it must resolve issues of conflicting identity (devices with the same name) and connectivity (separate links between the same devices, one link identified twice, etc.).

The Device Description data type establishes the detailed description of a SpaceWire PnP device needed for device recognition. The Device Class parameter provides a generic categorization of the device while the Vendor Identifier, Subsystem Identifier and Version parameters combine to form a “fingerprint” for identifying the device in a device database.

The Device Identifier parameter is the primary means of uniquely identifying a specific SpaceWire PnP device on a network-wide basis. Since the defined default (reset) state of the Device Identifier parameter is zero, a non-zero value indicates that the device has been recognized and claimed by a network manager. The assigned Device Identifier value should be unique within the set of devices discovered by that network manager. However, to avoid identity confusion, the Device Identifier must be network-unique, so higher-level network manager applications must communicate with each other to resolve name-space partitioning and device ownership issues. Note that SpaceWire PnP routers with Event Notification support offer a means for each network manager to uniquely identify each router independently using the Router Identifier field in the Notification Table.

2.1.2 IDENTITY RESOLUTION

The SpaceWire PnP protocol provides the Network Manager Identification parameters to aid the network manager applications in finding each other. The Network Manager Identification parameters (Granted Port Number and Network Manager Logical Address) are labeled to indicate their defined purpose, but can be used in any way desired by higher-level network manager applications. The Granted Port Number and Network Manager Logical Address parameters each offer similar capability implemented in different ways.

The Granted Port Number parameter is defined to contain the router port that should be used to send a packet to the network manager that owns the SpaceWire PnP router device. If the owner network manager is directly attached to that router device, then the Granted Port Number parameter value is the full path address to the network manager. If other SpaceWire PnP router devices exist between the initial router device and the owner network manager, they are presumed to be owned by the same network manager and the Granted Port Number parameter value of each router device provides a partial path address to the network manager (note that the Network Manager Logical Address parameter can be used to confirm the presumption of common ownership of intermediary router devices by comparing the logical address values). A second network manager wishing to communicate with the owning network manager can interrogate each device in the chain of Granted Port Number parameter values to determine the complete path address. Note that all intermediary router devices must be SpaceWire PnP compatible (i.e., support the Granted Port Number parameter) for proper functioning of this mechanism.

The Network Manager Logical Address parameter is defined to contain the logical address that should be used to send a packet to the network manager that owns the SpaceWire PnP router device. The owning network manager is presumed to have established valid routing table entries in all router devices between the initial router device and itself. Note that intermediary router devices can be non-compliant with SpaceWire PnP.

2.1.3 TOPOLOGY MAPPING

Topology mapping builds a database (a map) of the network connections. Each network entity and its interconnections are uniquely identified such that all paths between any two network elements are known. A complete and accurate topology map is critical when assembling path addresses between two nodes of a SpaceWire network. A SpaceWire network based on logical addressing can tolerate lesser global topology detail for node-to-node traffic (the routing table in each router must still be locally accurate), but needs a detailed map for router configuration traffic since SpaceWire routers do not have logical addresses.

2.1.4 DYNAMIC NETWORK CHANGES

An important aspect of the SpaceWire PnP protocol is support for dynamic changes to the network topology. The network topology changes when a network element is attached or detached. The change may be minor (a single node) or major (an entire network region, or subnet). If the change is an attachment, the new network region must be mapped and configured (or reconfigured) to merge it with the existing

network topology. If the change is a detachment, the network topology must be pruned to the new boundary.

The Event Notification capability of SpaceWire PnP routers eliminates the need for periodic polling of each router to detect changes to the network. It increases network responsiveness to changes by directly notifying interested network managers as soon as the event is detected. It also reduces the network traffic associated with event detection by limiting interaction to the router that detected the event.

An attachment event occurs when a router port becomes active while in the detached state. A detachment event occurs when a router port stays inactive for a configurable period while in the attached state. The Detachment Timeout parameter establishes the delay between a router port becoming inactive and recognition of the detachment event. Since a port may become inactive due to transient conditions such as error recovery, etc., the Detachment Timeout acts as a filter to reduce the responsiveness of the router to such situations. A notification event is either an attachment event or a detachment event that occurs while the SpaceWire PnP router is configured to issue Notification packets.

When a SpaceWire PnP router detects a notification event, it transmits a Notification packet to each network manager that has configured an entry in the router Notification Table (the packet contains the router identity, the current state of the router ports and the Notification Table entry index).

The Notification Table contains an entry for each network manager supported by the SpaceWire PnP router (the Notification Table Size parameter indicates the number of entries available in the table). Each entry contains a Notification Address field, a Router Identifier field and an Acknowledgement Timeout field. The Notification Address field contains the network address of the network manager to receive the Notification packet. The Router Identifier field is included in the Notification packet to indicate to the associated network manager which router sent the packet. The Acknowledgement Timeout field supports reliable delivery of Notification packets by allowing the associated network manager to specify a delay period before the router retransmits the Notification packet (a Notification Information Read transaction indexed to the Notification Table entry is used to acknowledge the Notification packet).

2.2 NETWORK CONFIGURATION

Network configuration optimizes network behavior. During or after network discovery, the configuration of each network element can be adjusted to conform to the needs of the network applications. The degree of customization performed is dependent on the capabilities of each network element and the choices made by the network architect.

The SpaceWire PnP protocol defines the tools for determining the basic capabilities and customizing the behavior of each network element. Higher-level network manager applications can make use of the basic SpaceWire PnP configuration features to provide additional functionality.

2.2.1 CONFIGURATION RESOURCES

The ECSS-E50-12A SpaceWire standard defines various device configuration parameters (e.g., routing table, logical address, etc.). The SpaceWire PnP protocol defines additional parameters required for Plug-and-Play applications and offers standard support for accessing both types of parameters. The parameters are defined in an abstract manner to allow flexibility of implementation. Any parameters required by the SpaceWire standard and certain SpaceWire PnP parameters needed for network discovery are considered mandatory for SpaceWire PnP compliance. Other SpaceWire and SpaceWire PnP parameters are either mandatory or optional depending upon perceived utility in most SpaceWire applications. Some mandatory parameters could be rendered unnecessary since an implementation might choose to limit the number of allowed values to as few as one.

The Port Table and Routing Table each contain a mix of mandatory and optional parameters. The Port Table fields Link State, Link Status, Link Speed and Maximum Link Speed are mandatory while the Arbitration Priority field is optional. The Routing Table fields Port Association and Header Deletion are mandatory while the Arbitration Priority and Packet Distribution fields are optional.

The optional Arbitration Mode parameter supports routers that implement more than one of the arbitration methods defined by the SpaceWire standard. Similarly, the optional Port Table Arbitration Priority field supports routers that implement priority arbitration between input ports and the optional Routing Table Arbitration Priority field supports routers that implement priority arbitration based on packet addresses.

2.2.2 CONFLICT RESOLUTION

Device configuration can be complicated in complex SpaceWire networks that have multiple simultaneously active network managers. The SpaceWire PnP protocol defines parameters and associated behavior to aid higher-level network managers in resolving collisions between multiple network managers.

Competing network managers that attempt to configure the same network element parameter inevitably create confusion and contention. The SpaceWire PnP protocol provides the conditional-write behavior as a partial solution. The conditional-write behavior is a classic atomic test-and-set mechanism with the test condition predefined by the network element implementation. The current parameter value must match the default (reset) value for the write transaction to succeed. After the parameter value is changed to a new value (and the new value doesn't match the default value), all attempts to overwrite the value will fail. Only a Reset packet selecting the parameter can affect the value.

The conditional-write behavior enforces serial access to a parameter, but does not address a conflict between network managers over ownership of that parameter. Higher-level network manager applications must define a protocol for resolving such conflicts.

The simplest conflict resolution protocol is that device parameters only be reset by the network manager that successfully wrote the current value. A network manager can access the parameter only after the previous network manager has finished using it (a fault in the owning network manager that causes it to leave a shared parameter in a

non-default state must be resolved by a higher-level mechanism, e.g., a time-limit convention between network managers sharing the parameter). While feasible for network discovery, a first-to-use arbitration scheme may not satisfy the optimization needs of the network. A role-based or region-based mechanism can be used to avoid conflicts during network configuration.

With an accurate network topology map, a network manager responsible for configuring the logical address assignments (Routing Tables for routers and Valid Logical Addresses for nodes) can perform that role independently of network managers with other roles (Device Identifier assignment, Link Speed configuration, etc.). Since all network manager roles cover the entire network, each network manager can independently discover the network topology. The only negotiation necessary between network managers is over the roles to be performed by each.

Similarly, with the topology map of a network region and an allocation of Device Identifiers and logical addresses from the network namespace, multiple network managers can independently configure the separate network regions. The non-trivial problem of partitioning the network topology into regions (while avoiding overlaps and gaps) is the most difficult aspect of region-based conflict avoidance. The SpaceWire PnP Network Manager Logical Address parameter can be used to mark region assignments by indicating the network manager responsible for configuring the associated network element.

3 CONCLUSION

The SpaceWire PnP protocol defines capabilities that support discovery and configuration of a network of compatible devices. It is based on a philosophy of providing basic facilities and leaving as much behavior to higher-level network manager applications as possible. The SpaceWire PnP packets and behaviors are transparent to legacy SpaceWire routers (except when addressed to a legacy router).

In addition to the Plug-and-Play facilities that are the primary focus, most features defined by the ECSS-E50-12A SpaceWire standard are explicitly supported by the SpaceWire PnP protocol. The Plug-and-Play features support device identification and event notification, both critical to network discovery and maintenance.

Support for contention resolution between multiple active network managers is included, but the detailed mechanism is relegated to the network manager applications. Similarly, some features are optional to facilitate development of minimal products.

4 REFERENCES

[1] SpaceWire Plug-and-Play Specification: http://tech.groups.yahoo.com/group/SpaceWirePnP/files/Draft%20Specification/

[2] SpaceWire Specification: http://spacewire.esa.int/content/Standard/ECSS-E50-12A.php

[3] SpaceWire Protocol ID Draft Specification: http://spacewire.esa.int/content/Standard/Draft_ECSS-E50-11.php

SPACEWIRE PLUG-AND-PLAY: FAULT-TOLERANT NETWORK MANAGEMENT FOR ARBITRARY NETWORK TOPOLOGIES.

Session: SpaceWire Networks & Protocols

Short Paper

Albert Ferrer Florit, Martin Suess

On-Board Payload Data Processing Section, ESA/ESTEC,

Noordwijk, The Netherlands

E-mail: albert.ferrer.florit@esa.int, martin.suess@esa.int

ABSTRACT

The SpaceWire Plug-and-Play protocol (SpW PnP) aims to provide a set of common features for SpaceWire devices to facilitate recognition and configuration of SpaceWire networks.

This paper presents a methodology for network discovery and configuration compatible with the SpW PnP protocol defined by the SpW PnP Working Group. It supports arbitrary network topology changes and provides fault tolerance features without requiring any manual configuration. Nodes and routers with identical hardware can be uniquely identified, and polling or active notification methods are used to register a new device in the network.

The proposed approach relies on the use of one or more intelligent nodes, called Network Node Managers, with the capability to independently configure any SpW PnP compliant device. Following suitable mechanisms to avoid race conditions, a network fully configured is only supervised by one Network Node Manager, with the others acting as hot backups.

Our methodology has been successfully prototyped with ordinary computers acting as network node managers, using RMAP protocol to emulate SpW PnP. Results proved that its use could be especially helpful for fast prototyping in the laboratory or in space manned missions.

1. INTRODUCTION

The SpaceWire Plug-and-Play protocol (SpW PnP) is a joint effort together with NASA and other partners in the frame of the SpW PnP Working Group [1]. It will enable the implementation of different network discovery and configuration methodologies depending upon a specific network management philosophy. The SOIS Plug-and-Play architecture is expected to be based on this protocol [2].

This paper presents a methodology for network discovery and network configuration without considering other typical Plug-and-Play services such as device drivers installation. The network management philosophy is based on the following assumptions:

A) SpaceWire network:

- A.1 The network topology is unknown and arbitrary.

- A.2 Arbitrary network topology changes can occur at any time due to failures or user intervention.

- A.3 Devices or subnets can be plugged and unplugged to/from any element of an existing network at any time. New devices plugged may not be in reset status.

- A.4 Multiple devices with the same hardware configuration may be present in the same network.

B) Network Discovery:

- B.1 It shall be executed by an intelligent node, called Network Node Manager.

- B.2 It shall detect plug/unplug events of any SpaceWire link, device or subnet.

- B.3 It shall uniquely identify all devices in the network.

- B.4 It shall support redundancy by using multiple Network Node Managers. Implementation shall avoid race conditions.

2. PROPOSED APPROACH

The proposed approach successfully deals with the previous assumptions and it is based on the use of intelligent nodes, called Network Node Managers (NNM).

Basic network discovery algorithm

A NNM interrogates routers about the status of their ports or links in order to discover new devices (nodes or other routers). Path addressing and configuration port is used for this purpose. Every device is configured with a different device identifier in order to avoid identifying multiple times the same device when there are loops in the network. This process ends when all devices in the network has been identified and programmed with a unique identifier.

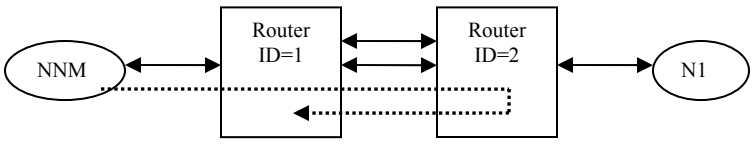


Figure 1: Simple network example. Dashed line shows how a NNM detects a loop.

Multiple Network Node Managers

Fault tolerance capability is implemented by using multiple NNMs. However, only one NNM, called Master NNM, should be active when the network is fully operative. This ensures that all devices have a unique identifier and all routing tables are consistent. Each NNM is configured with a different priority level used to determine who will continue mapping the network when another NNM is detected.

Race conditions

When more than one NNM is actively trying to discover the devices of a network, some mechanisms must be implemented in order to avoid conflicts when configuring devices, i.e. writing the device identifiers. In our approach these mechanisms are based on forcing all routers to be configured by only one NNM (that may be different for each router). This is achieved by programming Network Node Managers to write to a specific register, called GPN (Granted Port Number), the port number used to access to the router. If a NNM discovers a router with a valid GPN value that does not match the port number used to access it, the NNM will not immediately try to configure this router and will not look for other devices attached to this router.

Router configuration

As each router can be configured by only one NNM, some information must be hold by the router regarding its NNM. Our approach relies on the use of logical addressing to provide direct communication with the NMM of the router. Its logical address is stored in a specific register and the routing table is programmed to route incoming packets with this logical address to the GPN port. If the router configuration is consistent with the status of the network, the packets will be routed to other routers with the same NNM owner, until they reach the NNM itself.

Network Node Manager arbitration

The previous mechanism is used by a NNM to interrogate the NNM that is the owner of a router, regarding its status and priority level. If no reply is got, or the reply contains a priority level lower than the interrogation, the GPN will be overwritten with the value of the new NNM, and the router configuration will be updated accordingly. On the other hand, if the NNM receives back the message, it means that it already owns the router and a loop in the network has been found.

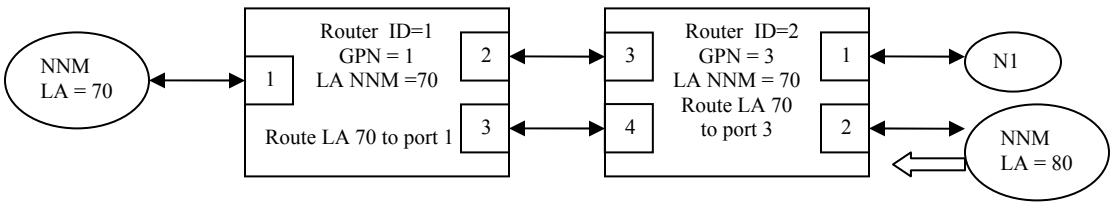


Figure 2: A new NNM is connected to an already configured network. The new NNM interrogates the NNM owner of the router to check its status and priority level.

The NNM interrogation message contains the identifier and the priority level of the NNM, the router identifier, and the return path from the router. The total return path is built by the NNM receiver. The reply message contains the identifier and the priority level of the NNM receiver and includes the interrogation message. The NNM receiver shall stop mapping the network if it receives a message with a priority level higher than their own.

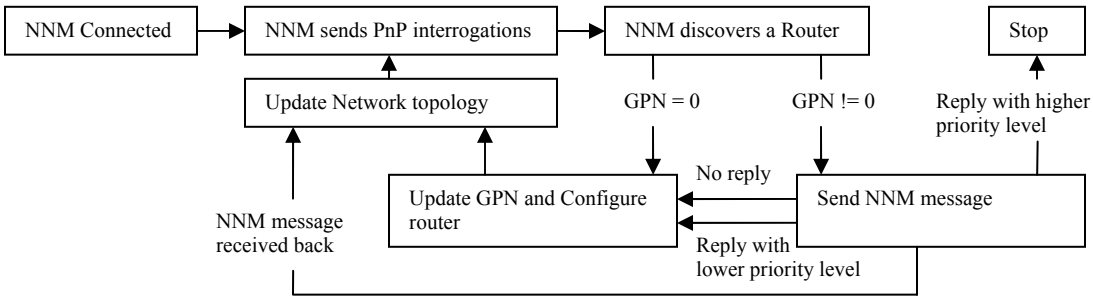


Figure 3: Basic flow of the Network Discovery procedure proposed.

Once a Master NNM is configured in an operative network, the other deactivated NNMs act as hot backups by periodically polling the Master NNM with NNM interrogation messages.

New device announcement

When a new intelligent device is connected to a router, it can notify its presence using the NNM logical address stored by the NNM Master. In case of passive devices, the Master NNM can detect them by polling mechanisms or by active notification from the router, as described in the PnP standard. If the new device is a router, network discovery procedure should be executed to map a possible new subnet.

CONCLUSIONS

A methodology for network discovery and network configuration has been presented using the SpaceWire Plug-and-Play protocol (SpW PnP) defined by the SpW PnP Working Group. The proposed approach supports any arbitrary change in the SpaceWire network topology and provides fault tolerant capabilities by using multiple Network Node Managers. It does not require previous manual configuration and it may become the first step towards the definition of a complete set of PnP services for SpaceWire.

REFERENCES

[1] SpW PnP Working Group, “SpaceWire Plug-and-Play Draft A”

[2] Proposed SOIS Plug-and-Play architecture and Resulting Requirements on SpaceWire Mapping, International SpaceWire Conference 2007.

BENCHMARKING SPACEWIRE NETWORKS

Session: Networks & Protocols

Long Paper

Asaf Baron, Isask'har Walter, Ran Ginosar and Isaac Keslassy
VLSI Systems Research Center, Elec. Eng. Dept., Technion, Haifa 32000, Israel

Ofer Lapid
Israel Ministry of Defense

E-mail: {ab@tx, zigi@tx, ran@ee, isaac@ee}.technion.ac.il, ofer.lapid@gmail.com

ABSTRACT

Measuring and comparing performance, cost, and other attributes of SpaceWire networks is a significant challenge. A new benchmark for SpaceWire-based satellites is presented. The benchmark contains a potential architecture of a multi-mission satellite, as well as specifications of the traffic flow. The proposed benchmark is demonstrated on an OPNET-based network simulation for various network configurations. The simulated SpaceWire network supports priority in multiple alternative manners: (1) a simple SpaceWire network that ignores priority, (2) a network that supports packet priority according to the SpaceWire specifications, (3) a non-standard support for N-Char interleaving on multiple virtual channels, and (4) support for both packet priority and N-Char interleaving.

1. INTRODUCTION

SpaceWire [1][2][3] has been designed to facilitate high-performance onboard data handling systems (DHS), to help reduce system integration time and costs, to enhance the reliability and efficiency of space systems, to promote compatibility between DHS and subsystems, to encourage re-use of DHS across several different missions and to facilitate spacecraft housekeeping and maintanance.

Payload processing involves several functions: controlling instruments, calibrating them, collecting data from instruments, storing the instrument data, processing and compressing the data and sending the data to ground via a down-link.

SpaceWire supports different architectures by using routers and point-to-point links that can be configured in different manners, customized and tuned to the requirements of specific missions. Links and routers may also be used redundantly, to enhance reliability, fault tolerance and availability.

The SpaceWire specification is still evolving. SpaceWire parts and networks are being proposed, and new amendments are being discussed [4][5][6]. However, no benchmark has been published in order to enable the comparative testing of standard variants or proposed networks and components.

In this paper we present a spacecraft model that can be used as a benchmark for SpaceWire implementations, networks, parts and protocols. The benchmark comprises typical spacecraft components such as sensors, processing units, mass-memory units and down-link telemetry systems. It defines the number of active components of each type and the data communication bandwidth that they require. The benchmark is

demonstrated by performance simulations of several alternative SpaceWire networks that could be employed for the benchmark spacecraft.

Benchmarks are necessary for performance evaluation, for comparison of network features and topologies, and for the development of simulation methods, common terminology and accepted figures of merit. So far, on-board spacecraft communications have typically been implemented without benchmarks and extensive simulation since they were based on ad-hoc connections of buses and dedicated serial links and on empirical design based on time division multiplexing.

In Section 2 we present the proposed benchmark. Section 3 describes the simulator used in this paper, and section 4 presents the simulation results, showing that the benchmark helps in identifying problems in various network configurations. We note that in this paper we have no intention of finding the best configuration for the network, but merely present the demands for it and demonstrate, by considering several alternative configurations, the usefulness of the benchmark and simulations.

2. THE BENCHMARK

Table 2, at the end of this paper, specifies the spacecraft components that are included in the benchmark. A real spacecraft may carry significantly more redundant inactive units for fault tolerance. Table 3 specifies the traffic requirements presented by the benchmark. Each row corresponds to one type of traffic. High bandwidth sensor data (Payload) are destined for the downlink through the storage. Other packet types (unmarked in the table) are either telemetry data (TM), comprising measurements sent to the DHS for processing, or tele-command data (TC), consisting of commands from the DHS to other units. In some cases the TM and TC traffic are symmetric, and in other cases they are not. Each traffic flow comprises BW bandwidth, in addition to addressing, parity and control overhead. Some flows happen in bursts, according to specified cycle times (as further detailed in Section 3). End-to-end latency expectation, if any, is also specified. Flows that constitute control loops include messages from the DHS to sample data by various units and messages back to the DHS with such sampled data. Finally, flows are tagged by high, medium or low priority (H, M, L respectively).

3. THE SIMULATOR

An OPNET-based wormhole network-on-chip simulator [7][8][9] was adapted for SpaceWire network simulations. The simulated networks comprise end nodes, as specified in Section 2, and routers [10][11]. End nodes are characterized by two parameters: average number of N-Chars per packet and average time between packets. These parameters are either constant or exponentially distributed.

The average packet size for cyclic traffic with cycle time c and bandwidth requirement BW is set at $BW \times c$, generating one packet per cycle. Note that in the benchmark of Section 2 such packets are limited to 100 N-Chars long, and are typically high priority. Payload packets are low priority, 60,000 N-Chars each. Other packets are either 100 N-Chars long (when BW>100 Bytes/sec) or $|BW|$-long, sending one packet per second. Ten overhead N-Chars are added to payload packets, to provide for control data such as for RMAP or RDDP, and one overhead N-Char is added to all other packets, to accommodate the logical address.

The average time between packets of cyclic control-loop packets is the cycle time c. For other packets, the inter-arrival time is exponentially distributed with parameter packet-size/BW.

All routers have eight ports, and each port may or may not be equipped with several Virtual Channels (VCs); different ports may have a different number of VCs. Routers perform priority arbitration, and round-robin arbitration among same-priority packets, first on different input ports and then on different VCs within each port.

The routers support Group Adaptive Routing (GAR), which means that more than one link can be connected in parallel between two routers, and messages may be routed between the routers over any one of multiple alternative links. Note that GAR behaves differently from a single connection with a bandwidth equal to the total bandwidth of all parallel GAR connections. Consider, for example, a low-priority packet transmitted from router A to router B. If there is a single connection between A and B, a high-priority packet will wait for the low-priority

packet to complete. In contrast, if there are two connections of half-the-speed each between A and B, the transmission of the low-priority packet takes longer, but a high-priority packet need not wait and may be transmitted in parallel over the second connection.

The simulator enables the implementation of N-Char Interleaving (NCI, presently disallowed by SpaceWire) [8]. NCI may outperform GAR. For instance, in the last example, if there are two connections, then the high and low priority packets can be transmitted in parallel but at a low rate. In contrast, if there is only one connection that is twice faster, then the high-priority packet can take over the line, transmit faster and arrive sooner, while the low-priority packet is blocked. Thus, resource sharing is improved. On the other hand, N-Char interleaving requires adding overhead bits to each N-Char.

The simulation does not implement physical routing. Rather, the routers support only logical routing, using the path with the minimal number of hops (routers). When there are multiple alternative minimal paths, traffic is divided equally among them, enabling arrival out of order. Routing tables are created automatically upon starting the simulation, using a BFS algorithm.

Flow control tokens are implemented in the simulator according to the SpaceWire specification. Time Codes and Errors are not implemented. Link data rates are assigned automatically: the bandwidth needed for each link is computed and then multiplied by a predetermined factor. Other simulation parameters include:

- Number of priority levels;
- Number of VCs [12] for each unit and each priority level;
- Buffer size for each unit and each VC (all VCs of the same unit and same priority level have the same buffer size);
- Network topology;
- Packet-Level Priority (PLP) [5][13], disabled or enabled in the routers; end units always support it and issue new packets in priority order;
- NCI disabled or enabled; in the latter case, service-level bits are added to each N-Char.

Only a small subset of the many possible configurations has been simulated using the benchmark, as presented in the next section.

4. SIMULATION RESULTS

This paper does not present an optimal network configuration. Rather, it demonstrates the simulation study of several alternative configurations using the proposed benchmark. Five different configurations are considered, all sharing the same linear topology and same network resources, as shown in Figure 1. In all routers, a 128-byte buffer is allocated to each input port, and when more than one virtual channel (VC) is used, then the same buffer is divided among the VCs [12]. Table 1 summarizes the configurations.

Table 1: Configurations parameters

Config.	N-Char Interleaving (NCI)	Packet Level Priority (PLP)	Virtual Channels (VCs)
1	No	No	1
2	No	Yes	1
3	Yes	No	4
4	Yes	Yes	4
5	Yes	Yes	8

Without NCI, each N-Char is 10-bit long. For NCI, 2 and 3 overhead bits are added to each N-Char to identify the VC when using 4 and 8 VCs, respectively.

Most traffic comprises low priority payload packets from the sensor to the storage and subsequently to the downlink units. We scaled this traffic by several factors to simulate different loads and present the average End-To-End (ETE) delay for each priority level as a function of that load. ETE delay is measured as the time from packet creation until the arrival of the tail N-Char at the destination.

Different capacities are assigned to different links, but they are kept constant per each link over all simulations. Capacity is determined by considering all data and overhead traffic requirements over each link and then selecting the slowest link rate from the set of {2, 10, 50, 100, 400} Mbit/sec which still accommodates the required traffic. Each simulation measures 10 seconds of real-time operation.

Figure 2 through Figure 4 show the ETE delay of all packets for configurations 1-5. Low-priority payload traffic from the sensor to the storage unit traverses links having capacity of 400 Mbit/sec. The ETE packet delay charts of low-priority payload packets in configurations 1 and 2 (Figure 2) reveal a relatively low upper bound, which is explained as follows. The downlink units can receive data at 60 Mbit/sec and are therefore connected by links with capacity of 100 Mbit/sec. The storage unit can send packets only to one of the downlink units at a time. Therefore, due to wormhole routing, the maximum rate in which the storage can send traffic to the downlink units is 100 Mbit/sec, even though the link emanating from the storage unit is designed with 400 Mbit/sec capacity. Most traffic generated by the storage unit is destined to downlink units and therefore its effective working rate is only 100 Mbit/sec when it can actually generate 300 Mbit/sec. Thus, for payload traffic above 1/3 of the total benchmark requirement, the queue in the storage is overloaded and ETE delay of packets to the downlink units increases dramatically. The results are unaffected by PLP (configuration 2) because payload traffic is low priority.

Still in Figure 2, configuration 3 exploits priority-based NCI and succeeds in eliminating the upper bound. Four VCs are established among the storage and downlink units. The storage can send packets to the downlink units at the maximum rate of 400 Mbit/sec. However, since an overhead of 2 bits per N-Char is required to identify the service level of the N-Char, the effective rate is now 360 Mbit/sec (the previously noted 300 Mbit/sec with 2 bits added to each 10-bit N-Char). As shown in Figure 2, the full generated bandwidth is managed successfully by the network.

Configuration 4 also employs four VCs among the storage and downlink units. However, PLP is also enabled, so the same number of VCs must be distributed among the various priority levels. We allocate 1-1-2 VCs to the H-M-L levels of priority. The two VCs at the lowest level (with the highest traffic load) are insufficient. The result is similar to that of configurations 1 and 2, except that the storage can send traffic at 200 Mbit/sec. Thus, an upper bound of 200/360 Mbit/sec is encountered.

Configuration 5 attempts to mend the problem of configuration 4 by using eight VCs, assigned 2-2-4 to priorities H-M-L, respectively. However, due to the added overhead bit, the storage needs to send data at the rate of 390 Mbit/sec. Furthermore, the storage does not only send data to the downlink units, but also flow control tokens (FCT) to the sensor. In a steady state, for each eight arriving N-Chars, one FCT must be sent. Since the storage receives traffic at 390 Mbit/sec, it also sends about 26 Mbit/sec of FCTs. This requirement creates a total flow of 416 Mbit/sec out of the storage. However, since the maximum link rate is only 400 Mbit/sec, the storage is limited to serving only 0.96 of the required benchmark traffic. This limitation is shown as a 0.96 upper bound in Figure 2.

Regarding medium and high priority packets (Figure 3 and Figure 4, respectively), for configurations 1 and 2 we observe that the ETE delay increases about linearly with the load over the first 1/3 of the benchmark traffic. At this point the network is overloaded by the low-priority packets and therefore any increase in the load beyond this point will have little effect on high and medium priority packets.

When using NCI and PLP (configurations 4 and 5) the load of the low-priority packets has almost no effect on the ETE delay of high-priority traffic. However, when we use NCI but no PLP (configuration 3), the load of low-priority packets affects the ETE delay of high-priority packets since in this case there is no guarantee that a high-priority packet will find an available VC.

The average ETE delay of high-priority packets (Figure 4) is longer than that of medium-priority ones (Figure 3) because of the larger average number of hops that high-priority packets must traverse in the example network topology.

5. CONCLUSIONS

We have proposed a benchmark for studying SpaceWire networks. The benchmark specifies spacecraft units, as well as requirements for inter-unit communication traffic. To demonstrate the usefulness of the benchmark, we considered a network topology and five different network configurations. The network has been simulated using OPNET, and end-to-end delay characteristics are presented. The simulated configurations include packet-level priorities and N-Char interleaving.

The simulations show that both packet-level priorities and N-Char interleaving may improve network performance. It appears that allowing atomic transfer units larger than N-Characters may improve network efficiency.

6. REFERENCES

[1] spacewire.esa.int

[2] Parkes, S. M., “SpaceWire: Links, Nodes, Routers and Networks”, European Cooperation for Space Standardization, standard number ECSS-E50-12A, January 2003.

[3] Parkes S. M., McClements C., “SpaceWire Networks”, DASIA 2002,

[4] S Mills, S Parkes, TCP/IP Over SpaceWire, Proc. DASIA 2003.

[5] G Rakow, R Schnurr, S Parkes, SpaceWire protocol ID: what does it means to you? IEEE Aerospace Conference, 2006.

[6] S. Parkes *et al*., CCSDS Time-Critical Onboard Networking Service, SpaceOps 2006.

[7] OPNET Modeler, www.opnet.com

[8] E. Bolotin, I. Cidon, R. Ginosar and, A. Kolodny, "QNoC: QoS Architecture and Design Process for Network on Chip", Journal of Systems Architecture, Volume 50, February 2004

[9] P. Fourtier, Simulation Of A Spacewire Network, DASIA 2007, 2A

[10] S.M. Parkes, C. McClements, G. Kempf, S. Fischer and A. Leon, “SpaceWire Router,” *Int. SpaceWire Sem.,* ESTEC, 2003.

[11] W.J. Dally and C. Seitz, "The Torus Routing Chip", Distributed Computing, vol. 1, no. 3, 1986.

[12] W. Dally, "Virtual Channels Flow Control", Proc. ISCA, May 1990.

[13] SM Parkes, P Armbruster, SpaceWire: a spacecraft onboard network for real-time communications, Real Time Conf., 2005.

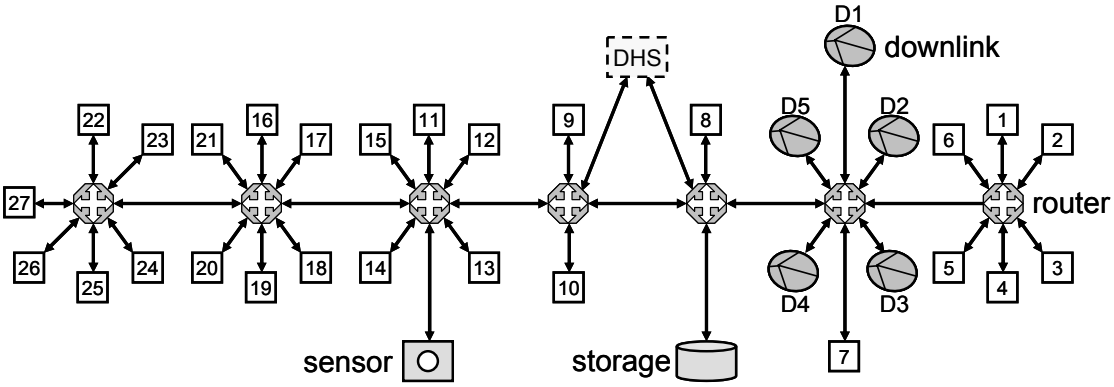


Figure 1: Example SpaceWire Network

Table 2: Benchmark Spacecraft Components Interconnected by SpaceWire

System	Name	Description	Num. Units	Num. Active Units
DHS	DHS	CPU	2	1
DHS	RU	Reconfiguration Unit	1	1
PLD	SENSOR	Camera, Particle Detector, etc.	3	1
PLD	STORAGE	Mass Memory for Storing Sensor Data	3	1
PLD	DOWNLINK	High-speed Communication Link	6	5
PLD	DAP	Downlink Antenna Positioning	2	1
PLD	DAS	Downlink Antenna Selection	2	1
TT&C	TTCGCS	Interface to Ground Control Station	2	1
TT&C	TTCAP	TT&C Antenna Positioning	2	1
TT&C	TTCAS	TT&C Antenna Selection	2	1
	THERMAL	Thermal Control	50	50
Elect	PCU	Electric Power Conditioning Unit	2	1
Elect	PDU	Electric Power Distribution Unit	2	1
Elect	BAPTA	Solar Array Bearing, Power Transfer Assembly	2	2
AOCS	STR	Star Tracker	3	3
AOCS	RW	Reaction Wheel	4	4
AOCS	MGM	Magnetic Flux Meter (Magnetometer)	2	2
AOCS	CSS	Coarse Sun Sensor	4	4
AOCS	FSS	Fine Sun Sensor	2	2
AOCS	GPS	GPS	2	1
AOCS	HPS	Hydrazine Propulsion System	2	1
AOCS	HETS	Hall Effect Thruster System	2	1

System acronyms:

DHS	Data Handling System
PLD	Payload
TT&C	Tracking Telemetry & Command
AOCS	Attitude and Orbit Control System

Table 3: Benchmark Traffic Specifications

From	To	Type	BW B/Sec	Cycle [Sec]	Latency	Control Loop	Priority
SENSOR	DHS		1,000	0.1			M
DHS	SENSOR		1,000		Low		H
STORAGE	DHS		1,000				M
DHS	STORAGE		1,000				M
SENSOR	STORAGE	Payload	100M				L
DHS	DOWNLINK		100				M
DOWNLINK	DHS		100				M
STORAGE	DOWNLINK	Payload	20M				L
DAP	DHS		500	0.1		Yes	H
DHS	DAP		200	0.1		Yes	H
DAS	DHS		1				M
DHS	DAS		1				M
DHS	TTCGCS		10K				M
TTCGCS	DHS		5K				M
TTCAP	DHS		500	0.1	Low	Yes	H
DHS	TTCAP		200	0.1		Yes	H
TTCAS	DHS		10				M
DHS	TTCAS		1				M
THERMAL	DHS		2.5	60			M
DHS	THERMAL		0.25	120			M
PCU	DHS		100				M
DHS	PCU		10				M
PDU	DHS		500				M
DHS	PDU		50				M
BAPTA	DHS		50	1		Yes	H
DHS	BAPTA		10	1		Yes	H
STR	DHS		2,000	0.1	Low	Yes	H
DHS	STR		1,000	0.1		Yes	H
RW	DHS		500	0.1	Low	Yes	H
DSH	RW		500	0.1	Low	Yes	H
MGM	DHS		50	1		Yes	H
DHS	MGM		1	1		Yes	H
CSS	DHS		20	1		Yes	H
DHS	CSS		1	1		Yes	H
FSS	DHS		20	1		Yes	H
DHS	FSS		1	1		Yes	H
GOS	DHS		100	1			M
DHS	GPS		17	30			M
HPS	DHS		100	0.1			M
DHS	HPS		100		Low		H
HETS	DHS		1000	0.1			M
DHS	HETS		100				M
RU	DHS		100	0.1	Low		H
DHS	RU		10	0.1			M

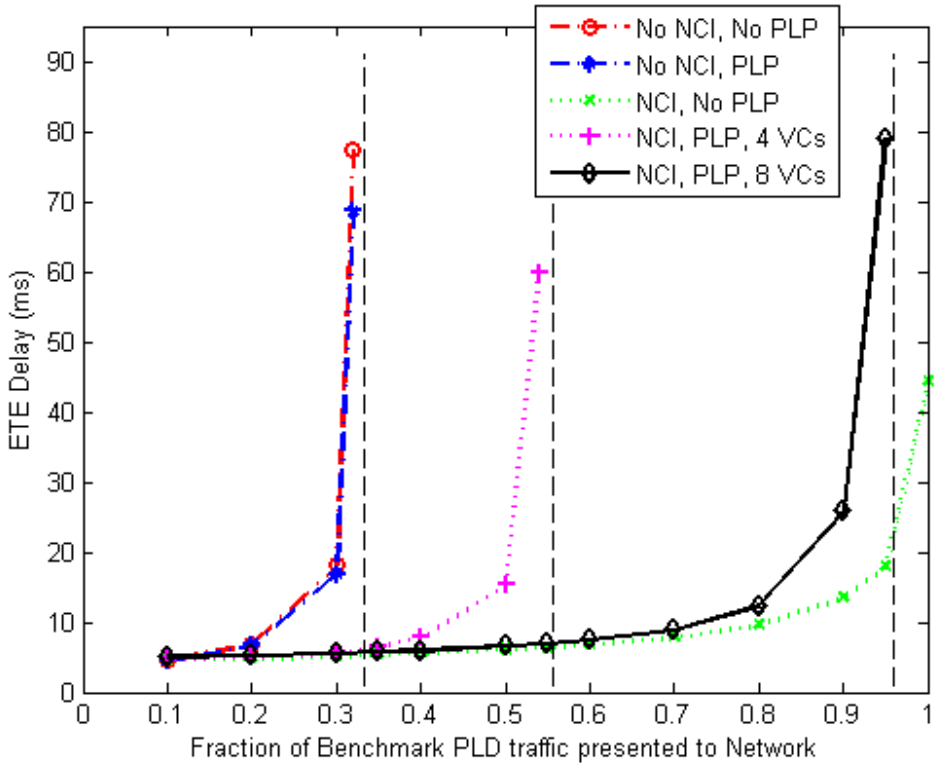


Figure 2: Average ETE delay of low-priority packets

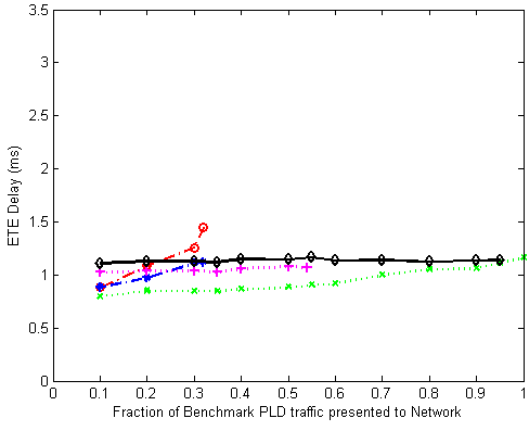


Figure 3: Average ETE delay of medium-priority packets

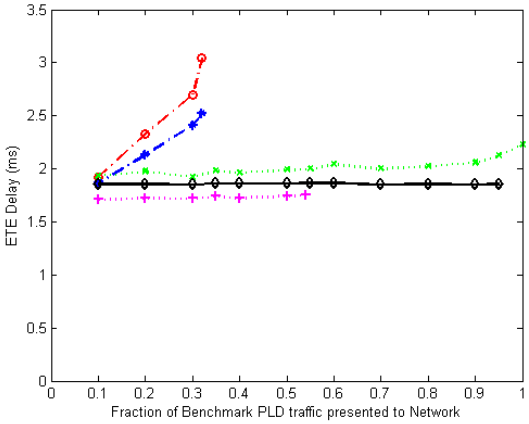


Figure 4: Average ETE delay of high-priority packets

INTEGRATION OF INTERNET PROTOCOLS WITH SPACEWIRE USING AN EFFICIENT NETWORK BROADCAST

Session: SpaceWire Networks and Protocols

Short Paper

Robert Klar, Sandra G. Dykes, Allison Bertrand, Christopher C. Mangels

Southwest Research Institute, San Antonio, TX, 78238

E-mail: robert.klar@swri.org, sandra.dykes@swri.org, allison.bertrand@swri.org, christopher.mangels@swri.org

ABSTRACT

SpaceWire is gaining popularity for space applications because of its simple circuitry, low power consumption, and high link speeds. Unlike the ubiquitous Ethernet which dominates the terrestrial Internet, SpaceWire does not include a link-layer broadcast. Common protocols that provide support for automatic network configuration such as the Address Resolution Protocol (ARP) and the Dynamic Host Configuration Protocol (DHCP) often rely on broadcast for discovery. Since SpaceWire does not explicitly provide provisions for these, Internet Protocol (IP) addresses and address resolution tables on each host must typically be manually configured. For large networks, this can be both a cumbersome and error-prone process.

This paper describes an efficient, loop-free, link-layer broadcast service for SpaceWire that supports automatic IP network discovery, configuration and management. Because it is implemented in the host software device drivers, our simple approach requires no changes to existing router or host interface hardware. It facilitates an easy integration path for existing protocol stacks and enables the use of standard ARP and DHCP implementations. The broadcast protocol is based on the concept of a SpaceWire subnet which consists of a router and its directly connected hosts. One host per subnet acts as the subnet server. The subnet server distributes broadcast messages locally to hosts on the same router. To extend a broadcast globally, the subnet server sends the message to the other subnet servers. We include performance results from simulation experiments, analytical results, and a prototype implementation. This research enables the convenient use of existing higher-level protocols and applications, thereby providing much promise for reducing the cost and time required to develop SpaceWire systems.

INTRODUCTION

Since its standardization by the European Space Agency (ESA) [1], SpaceWire, a high-speed low-power network, has rapidly been adopted for application on space missions by NASA, ESA, and other major space agencies [2]. By design, the standard was kept simple. The standard specifies SpaceWire "as a means of sending packets from a source node to a destination node" but does not specify packet contents beyond basic addressing [1]. Consequently, system designers have developed different communication protocols, most requiring static configuration.

Such implementations can become costly because recurring engineering is required to adapt the network configuration to support each different mission. This is contrary to one of the primary purposes of the standard – to reduce system integration costs.

Fortunately, the SpaceWire Working Group (SWG) provided some remedy for this situation by creating a standard encapsulation header [3]. This enables the multiplexing and demultiplexing of packet types belonging to different higher-level protocols, including IP stacks. The encapsulation header may also be used for automatic network configuration protocols such as ARP and to support emerging Plug-and-Play protocols for SpaceWire.

Underlying automatic protocols are the fundamental mechanisms of node discovery, route discovery, and address assignment. To that end, this paper presents three key concepts:

- A mapping of SpaceWire nodes to network unique addresses,
- A loop-free link-layer broadcast for SpaceWire to support integration of higher-level protocols, and
- Application of the broadcast to support some standard Internet protocols, i.e. Address Resolution Protocol (ARP) and Dynamic Host Configuration Protocol (DHCP).

UNIQUE ADDRESSES FOR SPACEWIRE

When transmitting, a SpaceWire node addresses a packet using a consecutive sequence of preceding bytes. An address byte in the range (1 ... 31) is a *path address* and specifies a physical output port on the router attached to a node for transmission. An address in the range (32 … 255) is a *logical address* that must be resolved by a SpaceWire router using a forwarding table to determine the output port for transmission. Logical addresses 254 and 255 are reserved. The SpaceWire standard provides for further extension of the address space by assigning each node to a unique region where logical addresses are not duplicated [1]. Typically, regions can be inferred from the router forwarding tables.

The problem for packet delivery is that SpaceWire logical addresses may be reused in different regions and therefore are not necessarily unique. We solve this problem by assigning each region a unique identifier called the RegionID, and using the combination of RegionID and logical address as the unique hardware address.

SPACEWIRE BROADCAST

In order to describe the broadcast, we define the term *SpaceWire subnet* (Figure 1) to indicate a single router and all of the nodes directly attached to its ports. A node on each subnet serves as a *subnet server*. (Although a subnet server could easily be embedded in router hardware, a node implementation was chosen to make use of off-the-shelf routers.) The subnet server has a special role and requires more information about the network than other nodes. It must be aware of the output port(s) on the local router from which it receives packets and have an address table for the subnet servers on other subnets. This information can be gained through methods of network discovery such as the Plug-and-Play protocols currently being proposed [4].

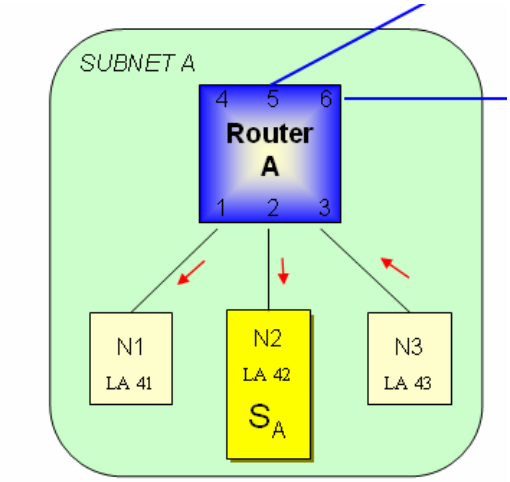


Figure 1. A SpaceWire Subnet

A protocol identifier (PID) of 253 was used to designate a broadcast in the encapsulation header. Following the header, the next byte was used to designate a broadcast message type. To implement broadcast, only two types are needed. These are designated Type 0 and Type 1. Type 0 messages are always sent using path addressing as a simple way to target messages to all devices attached to the subnet router. Type 1 messages are sent using logical addressing.

When a node wishes to send a broadcast, it sends a Type 0 message to all nodes on its local router. When the subnet server receives the Type 0 message, it resends the message as a Type 1 message to the other subnet servers in its broadcast table.

When a subnet server receives a Type 1 message, it resends the message as a Type 0 message to each port on its local router except itself. It passes the message contents directly to its higher-level protocol message handlers. In this way, the message is distributed efficiently throughout the network.

In networking, a broadcast loop occurs when the same packet is received more than once and resent, causing a network blockage or possibly a broadcast storm. In order to prevent such loops, the routers on the network are configured to drop messages with the logical address of 254. Since Type 0 messages are sent with path addresses where the output port is removed by the router, this guarantees that neighboring routers receiving the Type 0 message will discard the packet. Also, subnet servers must never send messages to themselves.

INTEGRATION WITH INTERNET PROTOCOLS

When a node receives a message, it strips off the broadcast encapsulation header to find another encapsulation header. The protocol identifier in this second encapsulation header can be used to designate higher-level protocols such as IP, ARP, and DHCP, supporting integration with a standard IP network stack.

ARP provides a way to translate a network address to a hardware address used for transmission at the link-layer. Because we have defined a unique network address based on a region identifier and a logical address, we can now make use of ARP. In our implementation, we chose to use a single byte for the region identifier.

Integration with a network stack also allows DHCP to be used for configuration of SpaceWire networks. DHCP dynamically assigns a unique IP address to a device and can provide other optional configuration information.

SIMULATION, IMPLEMENTATION AND TESTING

To demonstrate the broadcast, we developed a software simulation and a network device driver as internal research project funded by Southwest Research Institute (SwRI®). The simulation and the driver use the same core primitives.

Simulation and analytical study of the broadcast protocol were used to demonstrate correctness and provide comparison to a sequential unicast. For broadcast, an optimization where the broadcast only transmits to active ports on the attached local router was also considered. The results from the analysis matched those produced by the software simulation.

Although many topologies were considered, as an illustrative example, we chose to describe the results for two:

- A linear topology comprised of a linear sequence of eight routers with six hosts on each router. Each router has a single link to each adjacent router.
- A mesh topology comprised of a network with redundant links as might be found in a fault-tolerant spacecraft. Four routers with six hosts each were attached with redundant links between them.

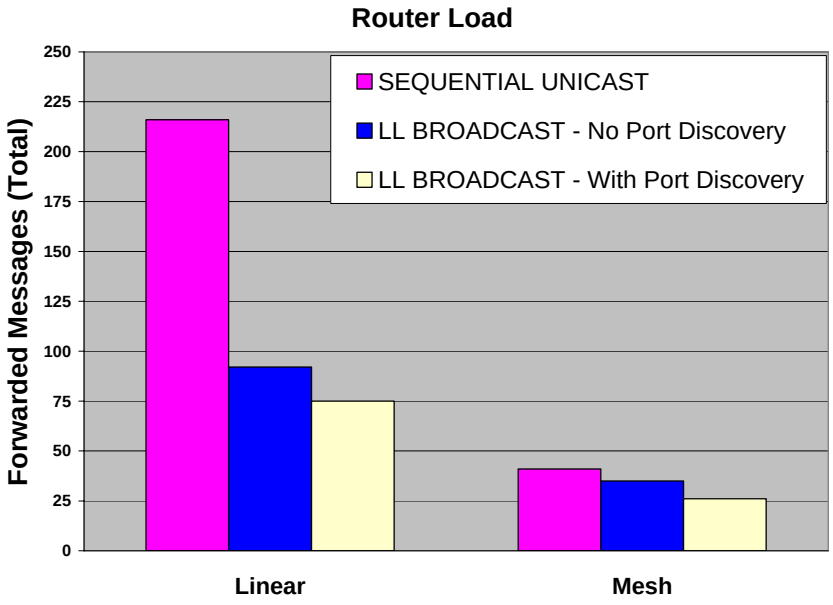


Figure 2. Comparison of Broadcast vs. Sequential Unicast

A study of the total number of packets required to complete the network-wide broadcast (Figure 2) shows that for the linear topology, our broadcast protocol reduces the number of packets transmitted by more than half. For the redundant mesh topology, our broadcast also requires fewer messages than sequential unicast.

A network device driver was developed for the SpaceWire Link Interface Module (SLIM), a 3U CompactPCI network card developed at SwRI® [5]. The driver was developed on an embedded Linux 2.6 kernel and implements the SpaceWire encapsulation header and broadcast service. The device driver is being tested using a configuration of three SLIM cards and two eight-port STAR-Dundee SpaceWire routers [6].

CONCLUSION

We have introduced a broadcast for SpaceWire that may be used as a foundation for support of standard network software stacks. Enabling the use of standard network stacks provides potential to reduce recurring engineering costs by enabling more standard, off-the-shelf applications to be used in space mission contexts.

REFERENCES

[1] CSS-E-50-12A, "Space Engineering: SpaceWire – Links, nodes, routers, and networks," ESA-ESTEC, January 2003.

[2] ESA SpaceWire Website: http://spacewire.esa.int/content/Missions/NASA.php

[3] CSS-E-50-11 Draft B, "Protocol Identification," ESA-ESTEC, February 2005.

[4] Patrick McGuirk, et. al. "SpaceWire Plug and Play (PnP)," AIAA Infotech 2007.

[5] Mark A. Johnson, et. al., "Design of a Reusable SpaceWire Link Interface for Space Avionics and Instrumentation," MAPLD 2005, September 2005.

[6] STAR-Dundee SpaceWire Routers, http://www.star-dundee.com

A SpaceWire Implementation of Chainless Boundary Scan Architecture for Embedded Testing

Session: SpaceWire Networks & Protocols

Short Paper

Mohammed Ali & Dr. Omar Emam

EADS Astrium Ltd

E-mail:, Mohammed.ali@astrium.eads.net, Omar.Emam@astrium.eads.net

Abstract

Electronic equipment for complex digital payloads have become increasingly more sophisticated incorporating thousands of ASICs with intensive interconnect between them. Most of these ASICs have to be monitored and controlled to satisfy the mission operational requirements. One scheme for implementing this is by connecting all these ASICs using a reliable network and interconnect such as SpaceWire. The ASIC in the system act as nodes which include their own SpaceWire routers. The testing and validation of the all electrical interconnects in such a complex system becomes a major contributor to increasing the schedule and cost of a project. Therefore built in self test, embedded test, strategies have been employed to alleviate these issues.

Traditionally IEEE Std 1149.1(JTAG) Boundary scan testing standard has been used to implement embedded test. Components in the system have to be daisy chained and driven by the JTAG control bus in parallel. Such an architecture has a number of limitations: Firstly, a faulty ASIC within a daisy chain will prevent testing of the whole chain. Secondly the parallel JTAG control bus puts undesirable constraints on the design of reliable systems. To overcome these limitations, and make the boundary scan test infrastructure transparent and independent of the design of the system under test, a generic chainless boundary scan architecture has been developed. This architecture is well suited to systems which use SpaceWire networks.

This paper proposes a dual function for the SpaceWire network. One, to facilitate the performance of embedded test. The other, is its standard role in communicating data and commands between nodes. A UK patent [1] application has been filed covering the method, architecture and technology described in this paper.

1. Introduction

The validation of modern complex electronic systems [2] now demand that facilities and features to test the system are incorporated as an integral part of its architecture requiring very little and possible no external test equipment at all. This approach is referred to as embedded test.

Traditionally IEEE Std 1149.1 (Standard Test Access Port and Boundary-Scan Architecture) [3] has been used to implement embedded test. Components in the system have to be daisy chained and driven by the JTAG control bus in parallel. Such architecture has a number of limitations: Firstly, a faulty ASIC within a daisy chain will prevent testing of the whole chain. Secondly the parallel JTAG control bus puts undesirable constraints on the design of reliable systems.

The solution is to develop and incorporate advanced highly integrated embedded test elements which will enable a built-in test infrastructure to evolve simply as by-product of the architectural design of the system it is intended to test. Such an advanced embedded test solution is inherently independent to the architecture configuration of the system it is built-in to test. This proposed embedded test solution is referred to as “Chainless Boundary Scan” and is well suited to systems which use SpaceWire networks as shall be described in this paper.

A typical SpaceWire [4] network is illustrated in Figure 1. In this example, the processors (P1 to P4) in the processor array are directly connected to one another. The sensors, memories and processor array are interconnected via the routers. Redundancy is provided in this example network by the use of redundant links and a pair of routers. If data is being sent from Sensor 1 to Memory 1 via Router 1 and the link between the sensor and the router fails then data can be sent from Sensor 1 to Memory 1 via Router 2. As described earlier, the SpW network interconnecting the components of the system is designed to enable information to be passed around the network even when at least one SpW link or the component itself has failed.

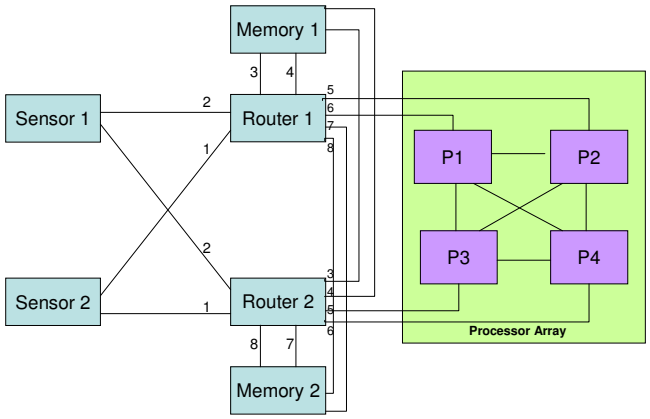


Figure 1: A typical SpaceWire Network

2. Background to Boundary Scan IEEE-1149.1 Standard

IEEE 1149.1 testing standard (also referred to as JTAG standard) is a test architecture which is used for designing boundary-scan test circuitry into digital integrated circuits for the purpose of testing the IC and the interconnections between them when assembled into a system.

All boundary scan compatible devices have a Test Access Port (TAP) with four required pins: Test-Data-Input (TDI), Test-Data-Output (TDO), Test-Mode-Select (TMS) and Test-Clock (TCK). An optional fifth Asynchronous Test-Reset (TRST) may be provided. At a system level, devices to be tested using boundary scan are connected in a chain referred to as a "scan chain". In the chain, the TDO of one device is connected to the TDI of the next device in the chain. The TDI of the first device in the chain is typically connected to the external boundary scan equipment via a "scan port". Similarly the TDO of the last device in the chain is also connected to the "scan port". The other signals listed above are passed through the "scan port" and are connected in parallel to all devices in the "scan chain".

Typically, a module incorporating boundary scan is designed with only a single "scan chain". However, there are instances where buffering requirements, redundancy management, the use of different interface logic (e.g. ECL/TTL) and power supply levels and clock domains on the same board, impose the need for multiple "scan chains", with each chain having its own "scan port". At system integration, modules that require boundary scan tests to be carried out between them must all have their scan ports connected to the same boundary scan test equipment. This is typically implemented using a multi-drop connection scheme.

3. Chainless Boundary Scan Architecture

The proposed 'chainless boundary scan architecture is based on a mesh network topology and allows the system design to be independent and free of the constraints imposed by a traditional daisy chain boundary scan architecture. One way to implement this is by exchanging the boundary information to the components under test via asynchronous communication links. These links can be individually routed or passed via the components under test which shall be referred to as 'nodes'. This strategy/approach would enable the system components themselves to facilitate the implementation of chainless boundary scan architecture as opposed to the traditional approach thus reducing component count and simplifying system design.

A system implementation of the "chainless boundary scan archetectiure" using SpW as the asynchronous communication links to distribute boundary scan information will be described with reference to Figure 2. The system consists of four boundary scan compatible integrated circuit devices C1 to C4 with a number of SpW communications links interconnecting each device with the other devices of the system. Each component may be regarded as a SpW node and incorporates a SpW interface as part of their normal functional requirements.

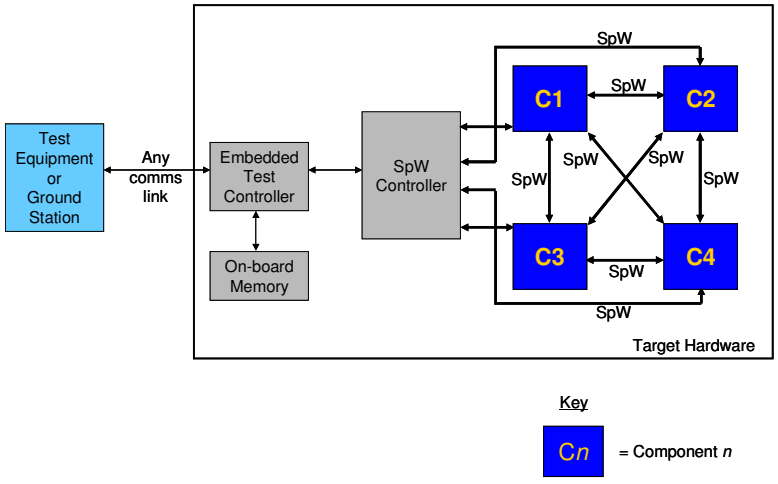


Figure 2: An example embodiment of the Chainless Boundary Scan Architecture

A key feature of the chainless architecture is that there is no requirement that the components to be tested using boundary scan are connected in a "scan chain" with the TDO of one device being connected to the TDI of the next device in the chain.

Instead, the system's SpW network is used to pass the boundary scan information when performing embedded test by using an integrated SpW router to boundary scan adapter IP block shown in Figure 3.

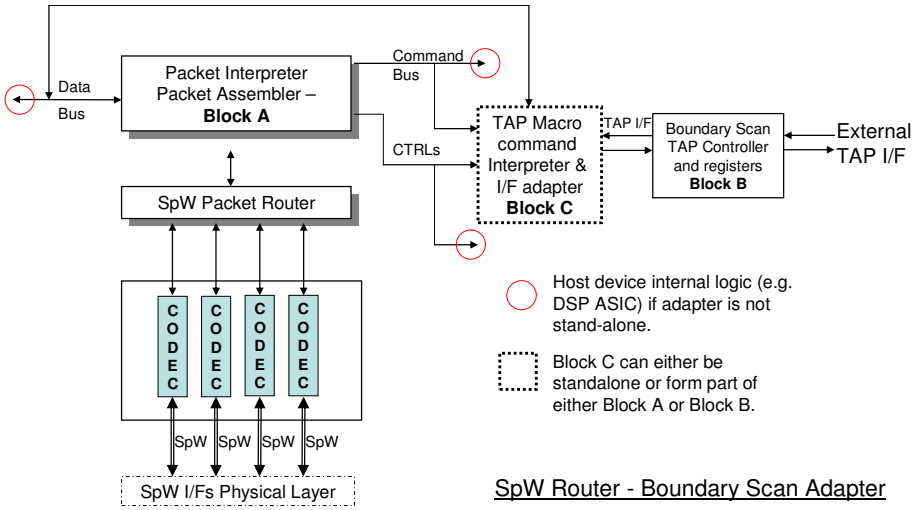


Figure 3: SpW Router to Boundary Scan Adapter

This IP block can be used either stand alone of as an integral part of the system components be it an ASIC, an FPGA or a unit. The basic functions of this IP block are:

- 4 x SpW I/F.
- 4 x Space Wire CODECs
- 1 x SpW Packet Router. The router also interfaces with a parallel interface where data on the Space Wire network can be presented if it was addressed to it.
- Space Wire packet interpreter and assembler block – Block A. This block is responsible for decoding commands arriving on the Space Wire interface and routing or processing the associated data or/and command appropriately. For the chainless boundary scan architecture described in this implementation of the claim it is necessary for this block to connect to the TAP controller block and associated boundary scan data and command registers.
- The Boundary scan TAP controller block – Block B. This block is responsible for generating all the necessary signals to drive the device's boundary scan circuitry and interfaces. This block also has an external Boundary scan TAP interface which would allow it to drive an external boundary scan chain of one or more boundary scan-able devices. The interface between the TAP controller block and the Space Wire packet interpreter and assembler block as designated as block C. The TAP controller would have a specific address or designation so that associated boundary scan commands and data transported on the Space Wire network can be communicated to it by the Space Wire packet Interpreter and assembler block.
- External conventional boundary scan IEEE 1149.1 Test Access Port (TAP) to enable testing of standard boundary scanable components that do not have SpW interfaces.

To perform a boundary scan test on this system, a boundary scan embedded test controller (ETC) as shown in Figure 2 is required to generate IEEE1149.1 boundary scan commands and data as well as receive and analyse scan data from the devices under test. This boundary scan information exchange is performed over SpW links between all the components under test and the boundary scan test controller. The primary function of the ETC is to sequence the tests in an appropriate order and execute the IEEE 1149.1 test vectors for in-orbit level test and diagnosis. To enable embedded and stand-alone boundary scan tests to be carried out, the test sequences are stored in a flash memory block and are accessed via the ETC under the control of the on-board processor. For System level test, at least one ETC is required.

4. Conclusion

This paper has demonstrated the viability of using asynchronous links to implement boundary scan without relying on the traditional daisy chain boundary scan infrastructure.

The advantage of the chainless boundary scan architecture described above, is that boundary scan tests can still be performed even after one or more components have been disabled, configured in functional mode or have failed.

The proposed scheme is advantageous in that it allows boundary scan test vectors to be transported via any appropriate communications link to the target hardware and directly executed on it without having to convert from and to an IEEE1149.1 interface. The method also facilitates communication between components on the target hardware to enable IEEE1149.1 testing without having to use external TAP controller signals.

This reduces the non-recurrent as well as recurrent costs of developing and producing a system. The added benefit is that this type of embedded test solution can be used to test and validate the system throughout its lifecycle, from leaving the assembly line to and including in-orbit testing in any permitted configuration.

5. References

[1] Patent Application Number 0712373.0, Inventors M Ali and O Emam., "Embedded Test System and Method", 2007.

[2] M Ali, O Emam, P Horwood and P Collins, "Implementation of 1149.1 and Scan chip set solution to Test Astrium Communication Space Satellites", International Test Conference (ITC), Baltimore, 2002.

[3] IEEE1149.1a – 1993 Standard, IEEE Standard Test Access Port and Boundary scan Architecture

[4] "SpaceWire – Links, Nodes Routers and Networks", ECSS-E-50-12A, January 2003.

Networks & Protocols 2

Tuesday 18th September

16:45 – 18:25

SPACEWIRE HOT MODULES

Session: SpaceWire Networks and Protocols

Long Paper

Asaf Baron, Isask'har Walter, Israel Cidon, Ran Ginosar, and Isaac Keslassy
VLSI Systems Research Center, Elec. Eng. Dept., Technion, Haifa 32000, Israel
Ofer Lapid
Israel Ministry of Defense
E-mail: {ab@tx, zigi@tx, cidon@ee, ran@ee, isaac@ee}.technion.ac.il, ofer.lapid@gmail.com

ABSTRACT

SpaceWire networks may include units, such as memory or processor units, which are bandwidth limited and in high demand by several other units. Such modules are termed Hot Modules. SpaceWire uses wormhole routing to deliver packets comprising multiple flits (N-chars). Wormhole routing helps minimize the number of buffers and the transmission latency. On the other hand, under high load, the network can become congested due to long packets that occupy resources at multiple network nodes, and block the paths of many other packets. This situation may be exacerbated at the presence of Hot Modules. In this paper we demonstrate that a single Hot Module can both dramatically reduce the network efficiency and cause an unfair allocation of system resources. First, the network efficiency is reduced because many packets can wait in different routers for the Hot Module to be available, while blocking other packets that are not destined to the Hot Module. Second, the allocation of system resources is unfair because in order to reach the Hot Module, packets from farther nodes need to win more arbitration cases than packets from nodes that are closer to the Hot Module. We explore several solutions for the Hot Module problem in SpaceWire networks. Possible solutions include provisions for priority-based routing and end-to-end access regulation mechanisms. We employ network simulations to investigate how the solutions address the network-efficiency and access-fairness problems.

1. INTRODUCTION

SpaceWire is a new set of specifications for onboard data-handling networks [1][2][3]. SpaceWire networks rely on wormhole routing [4] to transfer packets. In wormhole routing, each packet is divided into small fixed-size parts called N-chars (or flits), which are transmitted to the next hop without waiting for the entire packet to be received. This causes transmitted packets to be segmented and "spread" along the path between the source and destination in a pipeline fashion. Consequently, wormhole routing has relatively small buffer requirements, and incurs low latencies at light load. However, the main drawback of wormhole routing is its sensitivity to packet blocking, which can quickly fill up buffers along the entire path. In order to alleviate congestion effects, wormhole networks might employ virtual channels [5] to enable high utilization. Nonetheless, SpaceWire networks do not include virtual channels to allow packets to bypass blocked ones. Therefore, SpaceWire networks are especially vulnerable to high congestion.

The congestion problem in SpaceWire networks further intensifies at usage peaks, when the aggregated traffic demand exceeds the bandwidth of the destination module. Similarly, the destination module may also occasionally operate at a slower than average speed (e.g., a variable-speed encoder, decoder or storage device) and become congested, coincidently or not with an incidental usage peak. We term such a bandwidth-limited, high-demand unit a *hot module* [6]. In such situations, the hot module is unable to consume incoming packets fast

enough. A possible example for such a module onboard a satellite is a central processing unit or the main mass-storage device.

Congested modules exist in systems with any communication infrastructure, but wormhole-based systems are much more sensitive to hot modules, as the entire network may be affected: The hop-by-hop backpressure, associated with wormhole routing, causes buffers at the router adjacent to the hot module to be filled up and become stalled, blocking new arrivals to this router. This creates a domino effect, by which the delivery of packets to ports of more distant routers is slowed down, forming a *saturation tree* [7] with the hot module as its root.

Figure 1 and Figure 2 illustrate such a hot-module saturation tree. First, Figure 1 shows how the elongated shape of a typical satellite makes the network topology to be typically linear or near-linear, when short cables are used to protect signal integrity. Such a topology is exemplified in Figure 2, which represents a possible network topology with all its modules (using the benchmark traffic introduced in [8]). In this network, the storage unit is a hot module. The links leading from the satellite sensors to the storage unit form a saturation tree, shown in bold. (Note that a second saturation tree exists on the links from the storage unit to the downlinks.) The delivery of packets along the links in this saturation tree will be especially slow. Moreover, note that the domino effect stretches beyond the traffic on the saturation tree, as packets that are destined to other modules find no free buffers at certain routers on their route. For instance, packets sent from unit 29 to the Data Handling System (DHS) might get blocked in the network, blocking in turn packets sent from unit 25 to unit 23, which did not have any common link with the saturation tree. Thus, the network suffers from increased delays in packet delivery, as well as from unfair network utilization. This double threat is particularly troublesome in wormhole-based architectures due to packet "stretching" across multiple hops causing the hot-module effects to extend network-wide instantly. This phenomenon is *independent of links and router bandwidth*. In fact, such a network freeze may build up even in a system with infinite capacity links because of a single heavily-loaded module causing full buffers. Consequently, even largely over-provisioned networks suffer from poor performance if potential hot modules are left unhandled.

RELATED WORK

The negative effects of hot modules were partially explored in off-chip interconnection networks (e.g. [7]-[14]). In that literature there is no clear distinction between the issue of hot-module and the congestion of a network port. Typically, suggested solutions attempt to prevent regular traffic from being affected by the traffic of a saturation tree, either by not allowing one to form or by allocating hot traffic exclusive network resources. Unfortunately, such solutions do not bring a fair allocation of the hot resource. For example, some works modify the network routers in order to throttle packet injection at high loads (e.g., [10], [11]), discard packets (e.g., [12]), deflect packets away from loaded locations (e.g. [13], [14]), use separate buffers for traffic destined to a hot module (e.g. [9]), or simply use a large number of virtual channels. However, such solutions complicate the design of the network and may significantly increase its hardware cost. Note that most of the previous techniques can only slightly postpone the hot-module effect as they increase the number of buffers used by non-hot-module traffic (by adding buffers or routing non-hot-module traffic away), and in some cases they can even exacerbate the hot-module effect by throttling non-hot-module packets.

In [6], similar problems were addressed in an on-chip environment. In this paper, we partly use the solution proposed in [6]. However, since N-Char interleaving is not allowed in SpaceWire networks, we also show how the solution needs to be further improved.

Finally, other solutions to network congestion problems were proposed in SpaceWire networks as well [15][16], but none of these specifically address the hot-module problem.

RESULTS

In this paper, we examine two solutions to alleviate the hot-module effects in SpaceWire networks. First, we explore a simple mechanism that supports Packet-Level Priority (PLP) [8]. In this scheme, when competing for

network resources at the routers, packets traveling to the Hot Module receive a low priority, while other packets enjoy a higher level of priority. We show how this PLP-based mechanism manages to isolate the hot-module traffic from non-hot-module traffic, and therefore to mitigate the detrimental effects of hot modules on non-hot-module traffic.

In addition, we discuss a one-to-many end-to-end credit-based access regulation mechanism [6]. An allocation controller is introduced to arbitrate between the requests sent by the traffic sources. The controller allows the system designer to regulate hot-module access according to the quality of service requirements of the specific system application. The allocation algorithm employed by the controller is system-specific, since the hot module is independent of the network. Requests and grants are transmitted as small high-priority *credit* packets. In this paper, we explain how the credit mechanism prevents the accumulation of packets destined to a hot module within the network buffers. Consequently, we show how non-hot-module traffic remains unaffected even as the hot-module load increases significantly. In addition, we demonstrate how this credit mechanism alleviates the unfairness among flows resulting from their distance to the hot module. Finally, we show how combining the two solutions based on PLP and credits yields the best results among the proposed solutions.

The rest of this paper is organized as follows: In Section 2, the effects of hot modules on SpaceWire networks are discussed. Section 3 introduces two mechanisms to mitigate these effects, and Section 4 presents simulation results for the suggested techniques. Finally, Section 5 concludes this paper.

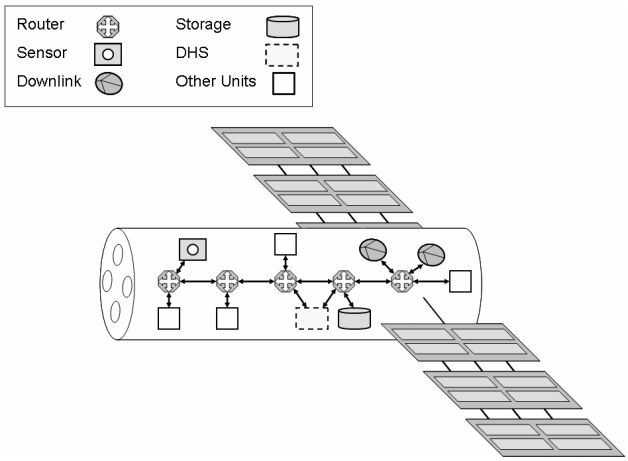


Figure 1: Elongated Shape of a Typical Satellite

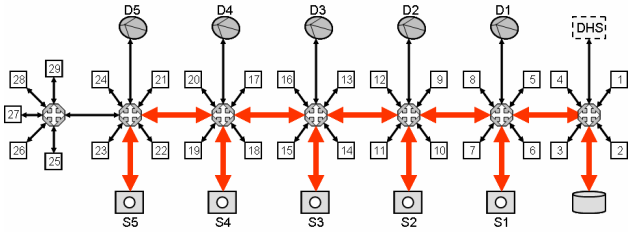


Figure 2: SpaceWire Network Topology

2. HOT-MODULE EFFECTS

The network buffer congestion due to hot modules has several negative effects on system performance. First, the hot-module *access latency* is increased, as packets destined to it contend for the limited hot-module bandwidth. As a result, when the load increases, the saturation tree becomes more and more congested, and packets traveling on the links of the saturation tree experience higher delays.

In addition, significant fairness problems arise, because different source modules are typically located at different distances from the hot module (as illustrated in Figure 2). More precisely, modules close to the hot module enjoy a much larger share of the hot-module bandwidth than distant ones. This is caused by the fact that routers typically employ a locally fair, round-robin arbitration between packets (or N-Chars) of similar priority waiting at different input ports and contending for the same output port. Therefore, when its inputs are saturated, each router that is part of the hot-module saturation tree equally divides the bandwidth available at its upstream port among its input ports. Consequently, hot-module throughput at a source can be modeled as dropping exponentially as a function of the number of hops between the source and the hot module. Further, the presence of other hot modules can also cause significant differences in access latency as a function of the number of hops —

in fact, packets sent by more distant sources are more likely to be blocked by traffic destined to another hot module. As a consequence, units that are located relatively far from the hot module experience extremely long access times when the hot-module load increases.

Furthermore, performance degradation due to hot-module load is not restricted to the hot-module traffic itself. In typical networks, hot-module and non-hot-module traffic compete for the same network buffer space and router ports. Therefore, hot modules that slowly service incoming data hinder the delivery of non-hot-module packets, as blocked hot-module packets wait inside the network occupying expensive buffers. As a result, packets destined to lightly loaded modules are also being stalled by the network, suffering delays and fairness problems similar to hot-module packets.

Note that the above discussion applies to any network in the presence of congested end-points. However, left unhandled, hot-module effects in the SpaceWire wormhole network are more severe than in a store-and-forward network, as packets are blocked across multiple routers and buffering space is limited. Further, hot-module effects are especially severe in SpaceWire networks, because N-Char interleaving is not allowed, and therefore it is harder to bypass blocked packets.

Also, it is important to note that these delay and fairness effects are symptoms of hot modules and not of an inadequately provisioned network. In fact, a wormhole network would suffer from the presence of a hot module even with links and routers of infinite capacity.

3. REDUCING HOT-MODULE EFFECTS

In order to reduce the dramatic effects that hot modules have on SpaceWire networks, we examine two possible solutions: a simple, inexpensive packet priority scheme and a many-to-one end-to-end credit-based arbitration mechanism. We first evaluate each of the suggested mechanisms separately, and will later consider using them jointly.

PACKET-LEVEL PRIORITY (PLP)

In this scheme, packets from different flows are assigned different packet-level priorities [8]. In particular, long packets destined at the hot-module have lower priority than short packets that are destined to the Hot Module and packets destined at other modules. Packet priorities are considered during output-port arbitration in network routers; if packets with different priorities contend for a single router output port, the packet with the highest priority wins the arbitration.

CREDIT-BASED ACCESS REGULATION PROTOCOL

The second scheme implemented in this paper is a credit-based access regulation mechanism [6]: each source can inject a packet into the network only after being granted permission by a *hot-module allocation controller*. The idea underlying this scheme is that packets that cannot be serviced by the hot module are not injected into the network and therefore do not wait inside the network. Consequently, a saturation tree cannot form and traffic not destined to the hot module remains unaffected during congested periods.

Two types of control messages regulate the access to a hot-module: a *credit request* packet, by which a source asks for permission to transmit a packet to the hot module; and *a credit reply* packet, which is sent by the hot-module allocation controller to grant permission. Due to their significance and short length, request and reply messages are given a high priority level. All other traffic in the network has normal priority.

The hot-module allocation controller can implement any arbitration scheme to select the next source module to be served, according to the designer's will. In this work, we assume simple round-robin service.

Finally, we distinguish between two possible implementations of the credit approval mechanism. First, in the original mechanism described in [6], new credit replies are transmitted just before the end of the transmission of packets currently received by the hot module. However, since credit replies may need to wait a long time for the

transmission of the current packets to be over, such an implementation might incur higher delays. It makes sense in [6] thanks to flit interleaving, which is unavailable in SpaceWire networks. Therefore, unless mentioned, we use a new credit approval mechanism, in which the credit approval is sent by the hot module as soon as the first N-Char of the packet currently received at the hot module has arrived. In the next section, we further discuss the effects of these two alternative implementations.

4. SIMULATION RESULTS

In this section, the performance of the two schemes is examined by means of simulation. We present four sets of results: (1) a standard SpaceWire network; (2) a SpaceWire network that supports PLP; (3) a SpaceWire network with a credit-based access regulation protocol; and (4) a SpaceWire network with both PLP and a credit-based access regulation protocol.

The term "end-to-end latency" in this paper refers to the time elapsed since the packet is created at the source until its last flit enters the destination. Therefore, the measured latency accounts for source queuing, network blocking, multiplexing, link bandwidth limitations, and overhead of the access regulation protocol. The results are generated using the OPNET based simulator [17], modeling a SpaceWire network at the N-Char level. The model includes all network-layer components, including wormhole flow control, routing, finite router buffers and link capacities.

SIMULATION SETTINGS

The topology of the simulated network can be seen in Figure 2. We use the benchmark from [8], but instead of using only one sensor that sends Payload (PLD) Data in a rate of 300 Mbit/sec, we use 5 sensors that send 60 Mbit/sec of PLD Data each. All the PLD traffic created by the sensors is destined at the storage in packets of 60,000 bytes each. Likewise, the storage also creates PLD traffic at a rate of 300 Mbit/sec that is destined to the downlink units also in packets of 60,000 bytes. The rest of the traffic is composed of telemetry data that is sent to the DHS by all other units and telecommand data that is sent from the DHS to all other units.

RESULTS

Figure 3 (respectively Figure 4) illustrates the average end-to-end delay of payload packets destined to the storage hot module (respectively to the DHS module). Both figures display four different plots, each based on a different scheduling methodology. Further, each hot-module plot displays five different curves and each non-hot-module plot displays seven curves, corresponding to the average ETE (end-to-end) delay in milliseconds over all sources located a given number of hops away from the destination.

Each plot shows the average ETE packet delay as a function of the hot-module link utilization. In order to illustrate the influence of the hot-module link utilization on the ETE delay, on each plot, the traffic demand is kept identical, while the hot-module link capacity is gradually reduced so as to increase the hot-module link utilization (the ratio of the fixed demand by the variable capacity), and accentuate the corresponding hot-module effects. The x-axis of each plot indicates the hot-module link utilization, comprised between 0 and 1.

Figure 3 (a) illustrates the average ETE delay of packets from the sensors to the storage hot module, when the network strictly follows the standard SpaceWire specifications. It clearly shows how the average ETE delays are large and increase as the hot-module link utilization increases. It also demonstrates the hot-module unfairness effect, since the average ETE delays increase for packets with more hops on the way. Similarly, Figure 4(a) illustrates the average ETE delay of traffic that is destined to the DHS when the network follows the standard SpaceWire specifications. Again, the average ETE delays highly depend on the hop location, causing extreme unfairness between flows.

Figure 3 (b) shows that using PLP does not improve the ETE delay of long packets destined to the hot module, because they all have low priority. Further, it also does not significantly mitigate the unfairness problem. On the contrary, Figure 4(b) shows that using PLP does help in reducing the ETE delay of traffic that is *not* destined to

the hot module, because the priority of this traffic is higher. However, the improvement is not dramatic, because the network is still congested with the long packets that are destined to the hot module

Figure 3(c) illustrates the use of the credit-based mechanism. It demonstrates how this mechanism manages to significantly reduce delay unfairness, because the hot module fairly allocates credits among all sources. In addition, Figure 4(c) shows how the credit-based scheme reduces the average delay of packets destined to the DHS unit. This is because the network is less congested by hot-module traffic, and therefore non-hot-module traffic is less affected by the hot-module congested flows. Obviously, since the credit-based mechanism only concerns hot-module traffic, it does not significantly reduce unfairness among flows destined to the DHS unit. Also, note that as the hot-module link utilization goes to 1, non-hot-module traffic is not directly affected, and therefore its average ETE delay will stay bounded, and oscillate around some value depending on the random traffic — hence the simulation artifact in the figure.

Finally, Figure 3(d) and Figure 4(d) show the results when jointly using both mechanisms. These figures illustrate how the combination of both mechanisms significantly mitigates the hot-module effects, by isolating hot-module and non-hot-module traffic, reducing hot-module traffic delay, and reducing the unfairness in the average ETE delay of hot-module traffic.

Figure 5 illustrates how the scheduling scheme strongly affects the hot-module unfairness effect. It plots the average ETE delay in milliseconds as a function of the source location, with a hot-module link utilization of 97%, using three different scheduling schemes. In the first scheme, a standard SpaceWire network is used, as in Figure 3(a) and Figure 4(a). It can clearly be seen that the delay increases with the number of hops that separate the source from the destination, for hot-module traffic as well as for non-hot-module traffic.

The two other schemes are both credit-based schemes. The first scheme uses the original implementation that posts a credit upon the arrival of the last N-Char, and the second one uses the new implementation that posts a credit upon the arrival of the first N-Char. As seen in Figure 5(a), the delay unfairness among hot-module flows nearly disappears, especially with the new implementation, which additionally reduces the average ETE delay by issuing credits upon the arrival of the first N-Char instead of the last one. Further, since credit-based schemes reduce the amount of hot-module traffic in the network, non-hot-module traffic finds less congestion and therefore encounters lower delays, as shown in Figure 5(b). In particular, in the original implementation of the credit-based scheme, credits take more time to arrive than in the new implementations, and therefore non-hot-module traffic finds less hot-module traffic congestion, thus leading to even lower delays.

5. CONCLUSION

In this paper, we showed how SpaceWire networks can contain hot modules that might cause congestion throughout the network and engender high delays and delay unfairness among traffic sources. In order to solve these problems, we have analyzed two mechanisms: first, a packet-level priority (PLP) scheme; and second, a credit-based end-to-end arbitration scheme. We showed how a combination of these schemes can efficiently mitigate the effects of hot modules and protect non-hot-module traffic.

Many of the problems mentioned in this paper can hardly be analyzed, or even detected, without simulation and benchmarking tools. For instance, we showed how non-hot-module traffic can be affected, even when it does not share any link with the saturation tree. We believe that such examples, sometimes quite counter-intuitive, further support the use of benchmarks and simulations, as argued in our companion paper [8].

REFERENCES

[1] Internet reference: spacewire.esa.int

[2] S.M. Parkes, “SpaceWire: Links, Nodes, Routers and Networks”, European Cooperation for Space Standardization, standard number ECSS-E50-12A, January 2003.

[3] S.M. Parkes and C. McClements, “SpaceWire Networks”, DASIA 2002.

[4] W.J. Dally and C. Seitz, “The Torus Routing Chip”, Distributed Computing, 1(3), 1986.

[5] W. Dally, “Virtual Channels Flow Control”, Int. Symp. Computer Architecture (ISCA), 1990.

[6] I. Walter, I. Cidon, R. Ginosar, and A. Kolodny, “Access Regulation to Hot-Modules in Wormhole NoCs”, First IEEE/ACM Int. Symp. on Networks-on-Chip (NOCS07), 2007.

[7] G. F. Pfister and V. A. Norton, “Hot Spot Contention and Combining in Multistage Interconnection Networks”, IEEE Trans. Comp., C-34(10), 1985.

[8] A. Baron *et al.*, “Benchmarking SpaceWire Networks”, Int. SpaceWire Conf., 2007.

[9] J. Duato *et al.*, “A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks”, High-Performance Computer Architecture (HPCA), 2005.

[10] E. Baydal, P. Lopez, and J. Duato, “A Congestion Control Mechanism for Wormhole Networks”, Ninth Euromicro Workshop on Parallel and Distributed (PDP '01), 2001.

[11] A. Smai and L. Thorelli, “Global Reactive Congestion Control in Multicomputer Networks”, 5th Int. Conf. on High Performance Computing, 1998.

[12] W. S. Ho and D. L. Eager, “A Novel Strategy for Controlling Hot-spot Congestion”, lnt. Conf. Parallel Processing, 1989.

[13] T. Lang and L. Kurisaki, “Nonuniform Traffic Spots (NUTS) in Multistage Interconnection Networks”, J. Parallel and Distributed Computing, 1990.

[14] P. Gawghan and S. Yalamanchi, “Adaptive Routing Protocols for Hypercube Interconnection Networks”, IEEE Trans. Comp., 1993.

[15] S Mills, S Parkes, “TCP/IP Over SpaceWire”, DASIA 2003.

[16] S. Parkes et al., “CCSDS Time-Critical Onboard Networking Service”, SpaceOps 2006.

[17] OPNET Modeler, www.opnet.com

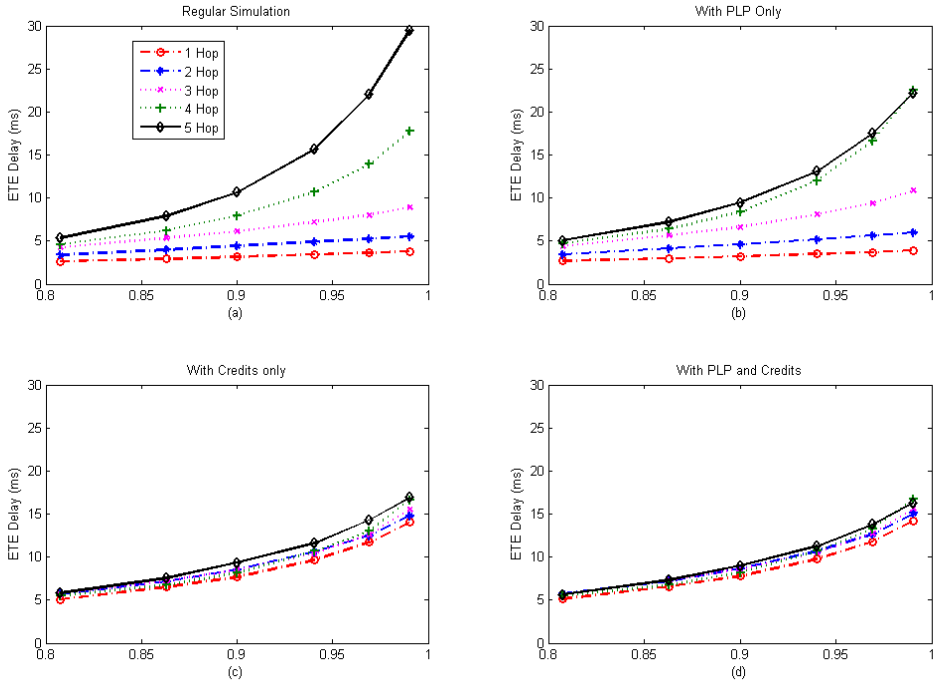


Figure 3: Average End-to-End Delay to the Storage Hot Module

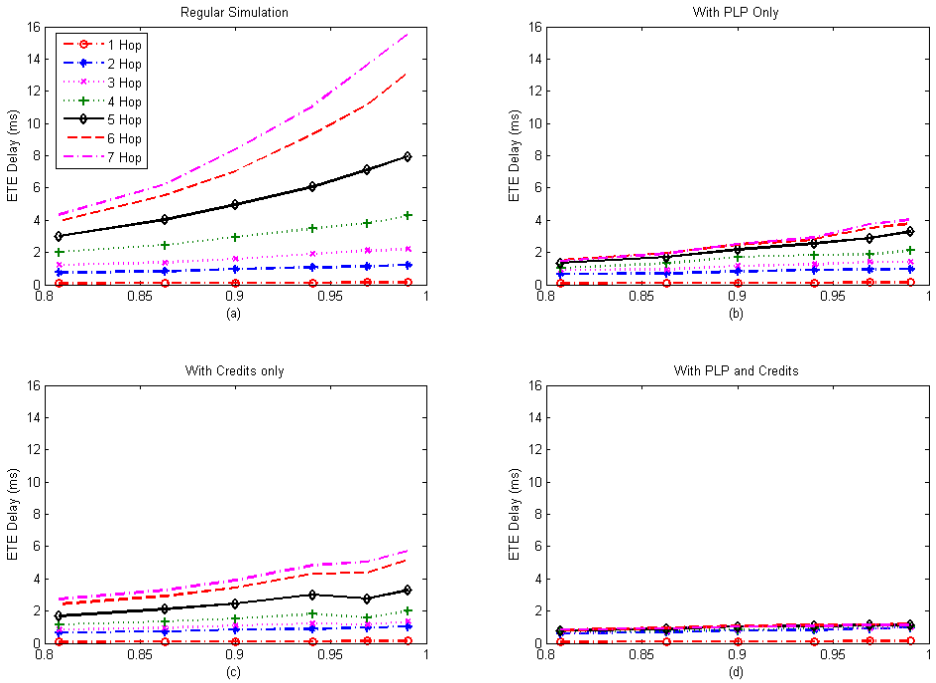


Figure 4: Average End-to-End Delay to the DHS Unit

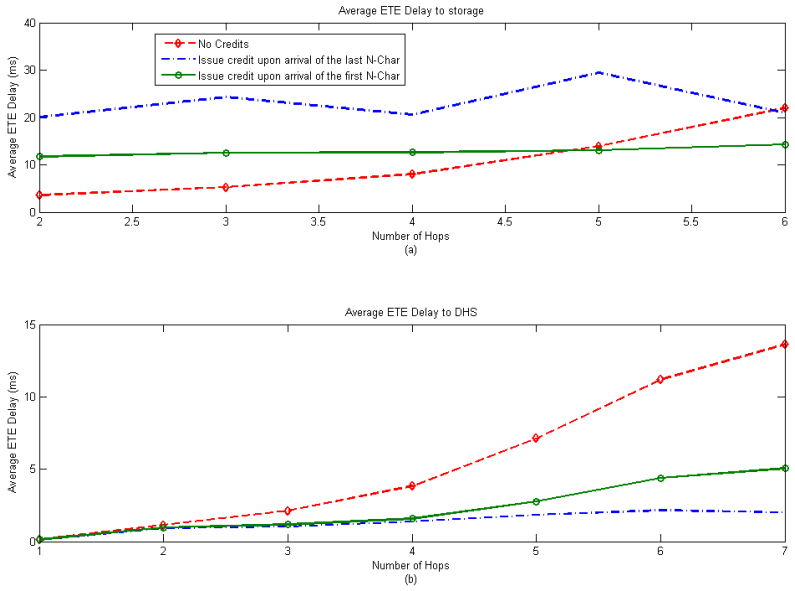


Figure 5: Delay Unfairness

SPACEWIRE NETWORK TOPOLOGIES

Session: Networks and Protocols

Short Paper

Dr Barry M Cook, C Paul H Walker …

4Links Limited, Bletchley Park, Milton Keynes MK3 6ZP, England

E-mail: [Barry, Paul] @4Links.co.uk

ABSTRACT

In contrast to other 'bus' standards, SpaceWire offers unprecedented flexibility in the choice of network topology. Choice can be made to optimize parameters such as performance, harness mass, cost, or fault-tolerance, or to provide an optimum balance of these, for a particular mission. This paper discusses some simple topologies and how they affect these parameters. It suggests hybrid topologies that might provide an optimum balance for some missions. It points to work on graph theory and on IEEE 1355 that may give useful design insights. The subject is large enough to warrant a course lasting several days, and in a short paper we can only scratch the surface. We hope that this will inform SpaceWire users and encourage them to find out more.

1 TRADE-OFFS: WHAT IS THE TOPOLOGY TRYING TO ACHIEVE?

Each mission has its own set of goals, and one of SpaceWire's [1] great benefits is to give the mission a wide range of topologies that provide a balance between a wide range of requirements. Inevitably, there are trade-offs between the different requirements.

Parameters that may influence, or be influenced by, the choice of topology are:

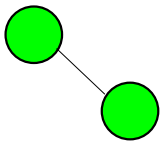
- The mass of the cable harness;
- The performance required: sometimes this will be different for different parts of the spacecraft, sometimes it is purely bandwidth, sometimes latency or delay through the network;
- The extent of fault-tolerance required, which may be different for different subsystems on the satellite.
- The power consumed by the network
- The cost of the components to build the network
- The lead time required for the satellite: can it be assembled from off-the-shelf building blocks, or is there time and budget to develop something new?

In considering the variety of topologies, we will suggest the effect that the topology might have on some of these parameters (particularly performance, harness mass and fault-tolerance). Of course topology is not the only design aspect to affect these parameters and harness mass, for example, might be reduced in networks where traffic is mostly unidirectional if a half-duplex version of SpaceWire [2] were used.

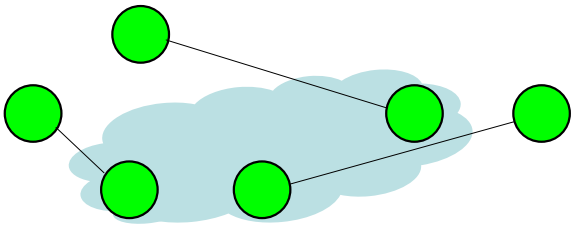
2 INTRODUCTION TO DIFFERENT NETWORK TOPOLOGIES

2.1 POINT-TO-POINT LINKS

Perhaps the simplest network is a single link between two nodes. Several spacecraft are flying with a number of such connections to make an ad-hoc network, perhaps more with the IEEE 1355-based components that have been available for some years.

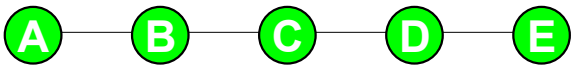


Of course there is, in practice, a (hidden) network between the disjoint nodes, because in almost all cases they will need to be controlled, and will need to communicate with the TM/TC down-link. The hidden network is shown in the figure here as a cloud. At first sight, it may seem easier to build the system with the hidden network, but the same tradeoffs need to be made and a full SpaceWire network has benefits.

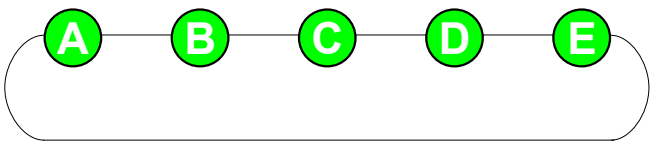


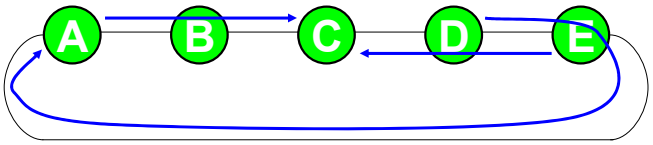
2.2 CHAINS AND RINGS

We can extend the point-to-point link by adding another node, and then another, etc., to make a chain of links, with small routing switches in each element of the chain. If connections are made between physically adjacent nodes, the chain offers the lowest harness mass of all the networks, but in other respects it is less than ideal. A packet from A to E will potentially experience contention and delay at all the intermediate nodes, limiting bandwidth and increasing latency. If a single fault occurs, such as in node C, the network becomes split into two networks, A,B and C,D which can not communicate.



The susceptibility to single points of failure can be overcome by connecting up the ends of the chain to make a ring. With this, any fault in a link or a node turns the network into a chain. Performance of the ring is potentially higher than of a chain because both directions can be used to make the shortest path connection between the nodes. Furthermore, the ring can carry multiple packets concurrently, for example one packet between A and C, one between D and A, and a packet in the other direction between E and C. Cable harness mass stays low if all the connections are made between nodes that are physically close to each other.

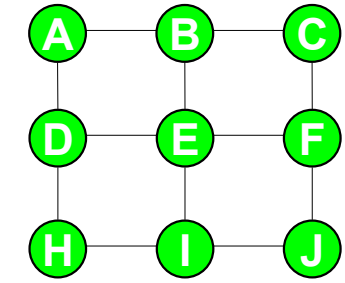




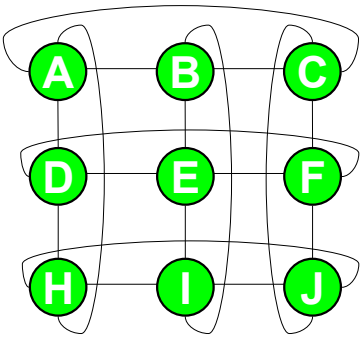
Strong claims are made of some ring-based networks such as Scalable Coherent Interconnect (SCI) and IEEE 1393 about ‘Spatial Re-use’, with several packets being transferred concurrently. It should be noted that SpaceWire provides the full benefit of Spatial Re-use while also allowing additional topologies beside the ring.

2.3 Multi-dimensional Chains and Rings, or Grids and Toroids

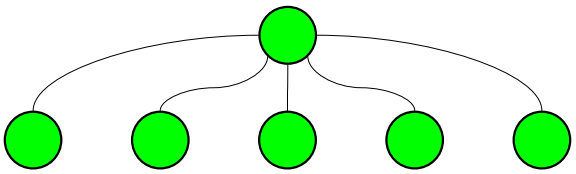
The chain can be extended into two or more dimensions to form a grid. This uses more ports on each node, and increases harness mass, to provide increased performance and fault-tolerance.



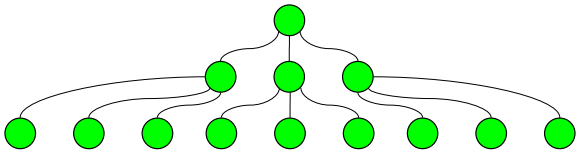
The multi-dimensional ring (or toroid) forms a loop on each row and column, which for a small additional cost and an increase in harness mass can provide more fault-tolerance and performance than the grid.



2.4 Centralized switches, Concentrators and Trees

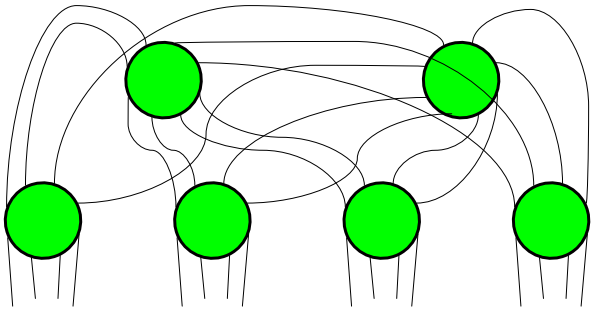


A typical application for a SpaceWire network is to connect a set of instruments to mass memory and processing, and a convenient way to do this is with a centralized routing switch, perhaps in the mass memory or processor. In many cases, a simplified routing switch which just concentrates the traffic is adequate, particularly if the instruments do not need to communicate with each other.



A tree can be built with a hierarchy of routing switches or concentrators. By placing the intermediate devices near their nodes, this can result in a substantial reduction of harness mass. If the leaf nodes need to communicate with each other, care needs to be taken that the root switch does not become a bottleneck. If the root node and the intermediate nodes are just concentrators, with no communication between the leaf nodes, then the root node is unlikely to be such a bottleneck.

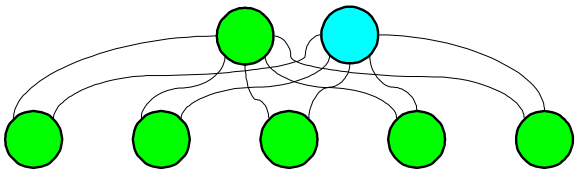
The bottleneck in the case of routing being required between leaf nodes can be overcome by building a 'fat tree' which preserves bandwidth at all levels of the hierarchy. This also provides a means of building a routing switch with twice as many ports as are available on a single chip. In the sketch here, six eight-port switches provide a single 16-port switch.



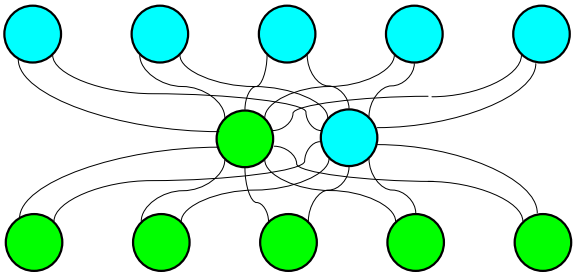
The cost of needing six eight-port switches to make a single full-bandwidth 16-port switch makes a strong argument for building switches with as many ports as possible. But, as we shall consider next, a single large switch is a large single point of failure.

3 FAULT-TOLERANCE CONSIDERATIONS FOR CENTRALIZED SWITCHES

The centralized switch or concentrator has many attractions. It is simple; each node is dependent only on the switch/concentrator and does not have to handle traffic from other nodes. If one node fails, none of the other nodes is affected, so fault-tolerance with respect to the nodes is good. On the other hand, if a single central point fails, it causes the whole mission to fail. And the cable lengths and hence harness mass is considerably higher with all the cables having to come together to one place rather than just go to nearest neighbours.



We can improve fault-tolerance by adding a cold-redundant switch or concentrator, at the cost of approximately doubling harness mass.

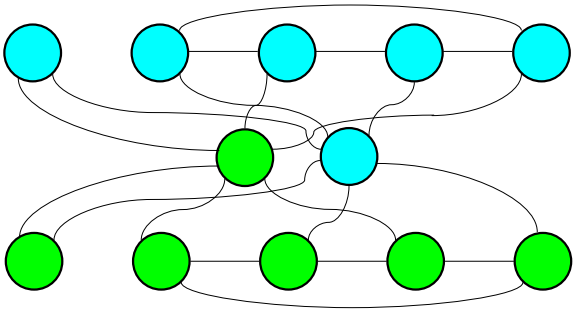


Many missions also need redundant instruments as well as redundant switches, and the harness mass increases again. Effectively this topology provides a 2x2 crossconnect between each node (both nominal and redundant) and each routing switch.

4 BALANCING THE TRADE-OFF, HYBRID NETWORKS

It has been seen that rings provide low harness mass at the expense of performance and centralised switches provide performance at a major cost in harness mass. Is it possible to mix the ring topology (where performance is less critical) with the centralised switch (where performance is needed)? SpaceWire does indeed provide the opportunity for such mixed or hybrid networks.

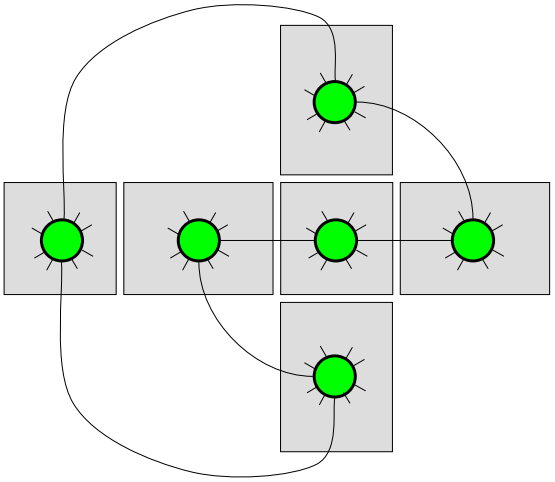
In the sketch here, the left nodes represent processors or mass memories or high-bandwidth instruments that need the dedicated connection to each routing switch. The remaining nodes do not need such high performance and so can be connected in a ring, with taps from the ring going to the switches. The number of Spacewire links is not changed from the previous example but eight long connections to the central switches have been replaced by eight much shorter connections to make the rings.

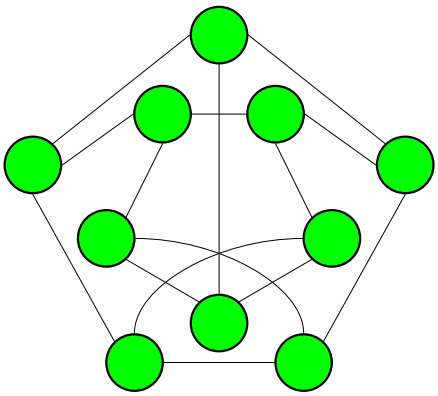


Many variations on this theme are possible and a trade-off study is likely to show for a particular mission that an alternative is preferable. In this example, we have chosen different connection techniques to balance between performance and harness mass, another balance might be with other parameters such as fault-tolerance, where some parts of the satellite may be more critical than others.

4.1 FOCUS ON LEAD-TIME, PNPSAT

PnPSat [3] has been designed as a kit of parts that can be built and launched within a few days of the mission requirement. They construct the satellite out of standard panels, each containing a routing switch which has local connections to nodes that are bolted to the panel, together with backbone connections to other panels. The initial network chosen is a hybrid network, with a single switch per panel and a ring to make the backbone between the panels, shown notionally here.

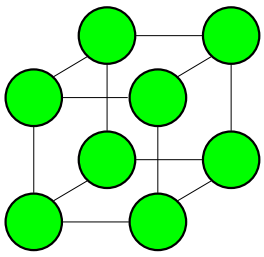




5 FURTHER STUDY

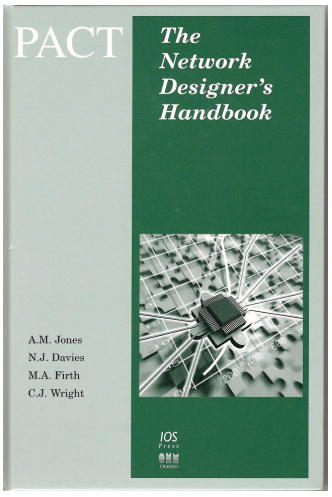
5.1 GRAPH THEORY

There is a large volume of work on graph theory, much of it targeted on finding networks that, for a given number of ports per switch, give the maximum number of nodes for a given maximum path length between nodes. There is a 'Moore bound' that sets the upper limit on the possible number of nodes for a given number of ports and path length (called degree and diameter respectively in graph theory papers). Reference [4] is one of many examples of papers that consider the Moore bound and list networks that come closest to meeting the bound. Very few networks actually meet the Moore bound — one that does is the classic Petersen graph, discoverd in the 19th century, which achieves ten nodes with three ports per node and with a maximum path-length of two between any nodes. Compare this with a cube, which has eight nodes, again has three ports per node, but has a path length of three between nodes on opposite corners — fewer nodes than Petersen yet a longer path length — but possibly easier to build physically.



5.2 THE NETWORK DESIGNER'S HANDBOOK

The development work on IEEE 1355 [5] produced a wide-ranging study of different network topologies and their relative performance in terms of throughput and latency. The results were published in 'The Network Designer's Handbook' [6].

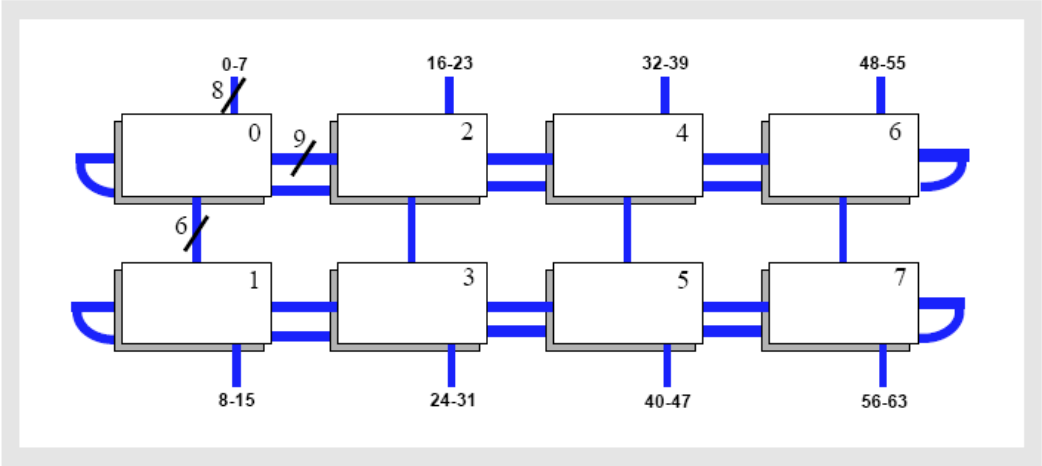


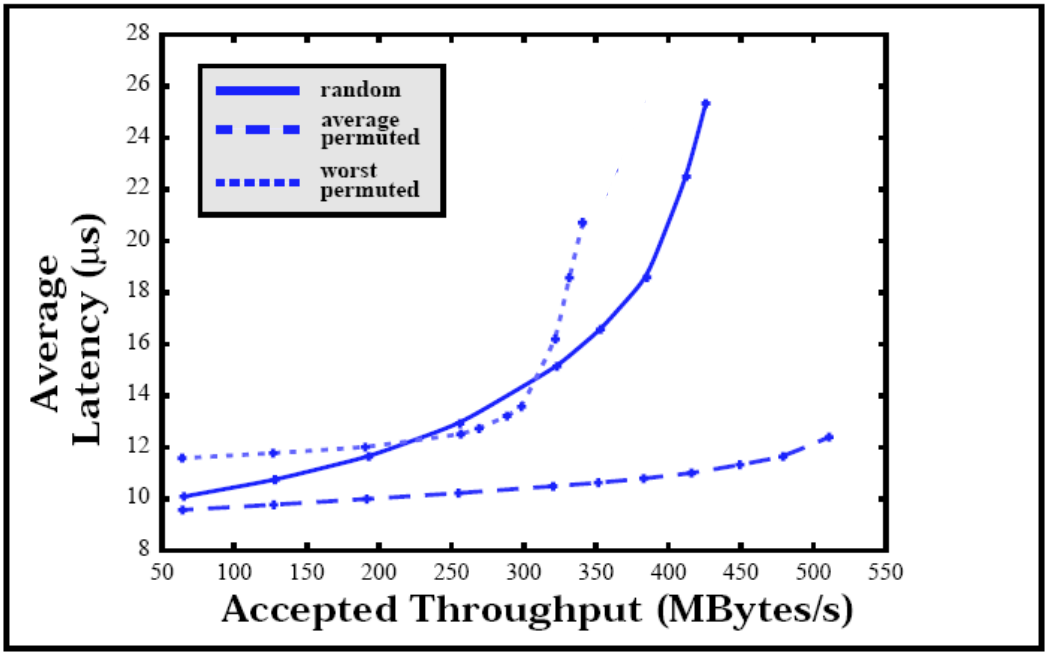
The results do not carry across to SpaceWire directly, because the work was based on the INMOS C104 routing switch which had 32 ports and used a different routing algorithm from SpaceWire's path and logical addressing. Furthermore, the networks considered were all regular

arrays, and many SpaceWire networks are, for good reasons, not such regular arrays. Also, the traffic in the simulations was assumed to be random, or had sequences of every node communicating in turn with every other node, whereas traffic in a SpaceWire network is likely to be more predictable.

Nevertheless, study of the results yields insights that could well benefit SpaceWire users. They might also indicate where projects might benefit from network modelling by software such as OpNet (as has been done by a small number of SpaceWire users), or with hardware [7], as was done for IEEE 1355.

An example set of results from The Network Designer's Handbook is reproduced here for a two dimensional ring or torus which acts as a backbone between eight ports per switch that go to local leaf-nodes, so that it connects a total of 64 leaf nodes. The extracts shown here are taken from a draft of the book and appear on pages 78 and 79 of the published version.





6 CONCLUSIONS

SpaceWire provides unprecedented flexibility in network topology. Two extremes are (daisy) chains, where most traffic has to visit several nodes before arriving at the destination node, and large central switches, where all traffic goes via a single switch.

A number of missions have chosen some form of hybrid topology, comprising elements of rings (daisy-chains with the loop closed) and of central switches. One example is PnPSat which uses a local switch on each panel to connect to the nodes on the panel, and then uses a ring as a backbone between these switches.

By studying some of the topologies that are possible for SpaceWire, perhaps together with academic results from graph theory, the industry may gain a more complete understanding of the effect of topological choices on missions. Similarly, results obtained from simulations of IEEE 1355 networks may also give useful insights.

SpaceWire's topological flexibility brings great new opportunities for improvements in all of the parameters such as performance, mass, cost, and reliability. Increased awareness of the effects of topology on these parameters could have a significant effect on mission lead-time, cost-effectiveness, and success.

REFERENCES

In the following references, * indicates a paper that is being presented at this conference.

[1] The ECSS-E-50-12 Working Group, "**ECSS-E-50-12A 24 January 2003**, SpaceWire - Links, nodes, routers and networks", published by the ECSS Secretariat, ESA-ESTEC, Requirements & Standards Division, Noordwijk, The Netherlands

[2] Barry M Cook, Paul Walker, **"Asymmetric SpaceWire"**, to be published, abstract available from 4Links

[3]* Donald Fronterhouse, **"PnPSAT"**

[4] Michael J Dinneen, Paul R Hafner, **"New Results for the Degree/Diameter Problem"**, accessed from http://arxiv.org/PS_cache/math/pdf/9504/9504214v1.pdf on 28 August 2007

[5] Bus Architecture Standards Committee of the IEEE Computer Society, "**IEEE Std 1355-1995**, IEEE Standard for Heterogeneous InterConnect (HIC) (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction)" IEEE, 1995

[6] P. Thompson, A.M. Jones, N.J. Davies, M.A. Firth and C.J. Wright (Eds.) **"The Network Designer's Handbook",** Volume 51: Concurrent Systems Engineering Series, 1997, 309 pp., softcover, EOS Press, ISBN: 978-90-5199-380-6

[7] S Haas, D A Thornley, M Zhu, R W Dobinson and B Martin, **"The Macramé 1024-Node Switching Network"** Microprocessors and Microsystems, IEEE 1355 Special Issue, Ed. Paul Walker, V21, Nos 7,8, 30 March 1998, (this paper is also summarized in Chapter 8.2 of The Network Designer's Handbook)

THE SYSTEM APPROACH FOR A SPACEWIRE NETWORK

Session : SpaceWire Network and Protocol session

Bruno Masson[1] bruno.masson@thalesaleniaspace.com
Stéphan Dethève[1] stephane.detheve@external.thalesaleniaspace.com
Bernard Alison[1] bernard.alison@thalesaleniaspace.com

[1] *Thales Alenia Space 100 bd du Midi BP 99 06156 Cannes la Bocca Cedex France*

Introduction

The need of high speed board-to-board communication links has led to define a new standard, extending the previous 1355 point-to-point, to SPACEWIRE network standard concept. However each space mission will need a specific network architecture in terms of its objectives. So, it is important to establish an approach of SpaceWire networks at system conception level in order to optimize their performance, their robustness and their weakness. For this reason, a top-down approach is promoted to be sure that top level needs are well implemented. It is why SpaceWire technology is now fully considered in the overall THALES ALENIA SPACE (TAS) avionics roadmap.
This paper introduces the different ways which are implemented to master and to optimize SpaceWire standard features for future space missions:

- A full numerical simulator
- A scalable and adaptable HW/SW mock-up

MOST, a representative simulation tool

MOST (Modeling Of Spacewire Traffic) is built upon the OPNET tool dedicated to network modeling. A specific SpaceWire library is provided and developed by TAS, containing validated models of SpaceWire elements (routers and nodes). The models are fully representative of the standard and of the characteristics of available hardware components (i.e. ESA router).

The nodes models let users implement application specific behaviors as packets consumers and/or providers. These models can be enhanced anytime, when the design of the nodes becomes more precise.
The tool lets user configure network models with the different options proposed by the standard: logical or physical addressing, Group Adaptive Routing (GAR), …
In addition, MOST implements the main features of RMAP protocol layer (selectable options) and lets user provoke anomalies chosen in a pre-defined set of reasonable cases, at chosen dates within simulation sequences.

MOST is based on a realistic representation of the traffic at character level.
This option has been chosen due to the dependence between traffics on both directions of a SpaceWire bidirectional link (see SpaceWire protocol description and FCT characters (Flow Control Token) in SpaceWire communication standard, titled ECSS-E50 12A).

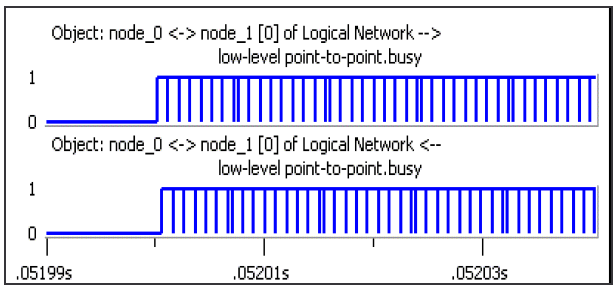


Figure 1 : MOST provides an accurate character level modeling

When the network topology is available, the designer develops the network model under the OPNET environment, by picking the components in the SPACEWIRE library. The graphical editor makes easy connections between representations of routers and nodes.

Once created, each node and router can be parameterized, using the pre-defined set of properties accessed via graphical editor commands (i.e. data rates, routing table, buffer size…).

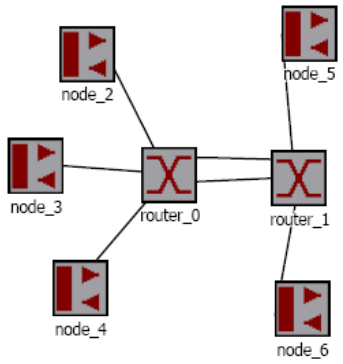


Figure 2 : OPNET Graphical User Interface lets user build quickly any network topology

When the scenarios have been specified (nodes behavior defined, network parameterized and output parameters selected), the network model can be executed for a programmed simulation duration as provided by the OPNET tool. This run results in a recorded history of outputs.

MOST is fully representative of the previous properties and of the components implementing the standard. Prototyping with MOST gives quick return on design weakness or risks of any simple or complex network. For instance, the influence of key parameters (i.e. packet size and frequency) can be accurately measured.

The advantages of this tool are the ability to restitute the traffic of the whole network, when the analyzer logs traffic on one point-to-point link and the customization of different network elements in terms of needs for a specific space mission.

This way, results of simulations can be correlated with results obtained on avionics test bench, containing SpaceWire traffic analyzer in order to qualify MOST simulator. As it was not yet an industrial tool because it needs to be formally validated, how do we validate this tool ?

Breadboarding mock-up of a SpaceWire network

MOST is a simulator based on behaviour of real hardware components. So it is necessary to make a cross-checking between a network simulated by MOST and a breadbording mock-up using real hardware components which will be for us a reference point.

This comparison study has been developed in two steps.

- The first one is the system specification for SpaceWire network architectures and high level protocols (including a dedicated FDIR definition).

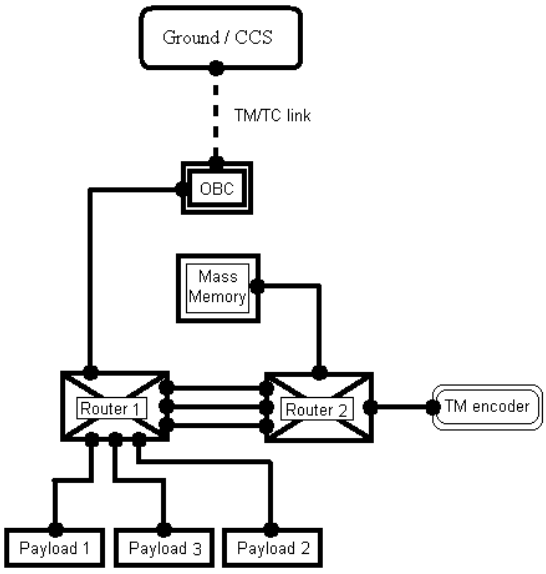


Figure 3 : Representation of network topology

Here, the experiment specification is defined by :

- a specific topology,
- scenarios and network characteristics,
- SpaceWire features,
- ways to monitor the network state,
- ways to detect failures and blocking,
- improvements of SpaceWire utilization.

And, the experiment network is consisted of :

- a Center Check-out System (simulating ground station),
- an On-Board Computer,
- a Mass Memory,
- a Telemetry Encoder,
- three Payloads,
- two Routers.

- The second one is the development of avionics mock-up which includes TM/TC functions, processor module, mass memory module, SpaceWire routers and payload modules providing science data.

Applicative software in middleware layers has been developed in order that it is in charge of the SpaceWire handling. Global validation has been performed in order to verify the concept.

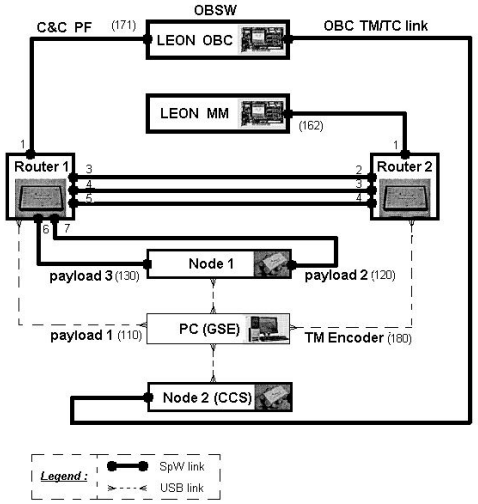


Figure 4 : Representation of mock-up topology

For the breadboarding, the hardware is :

- LEON boards,
- bricks & routers.

The TAS avionics software mock-up is a software emulating a platform which is reused and adapted to SpaceWire standard.

Applications of middleware layers have been developed to simulate :

- On-Board Computer & Mass Memory by LEON boards,
- Payload, Telemetry Encoder & ground station behind bricks or routers by a host PC (Ground Support Equipment)

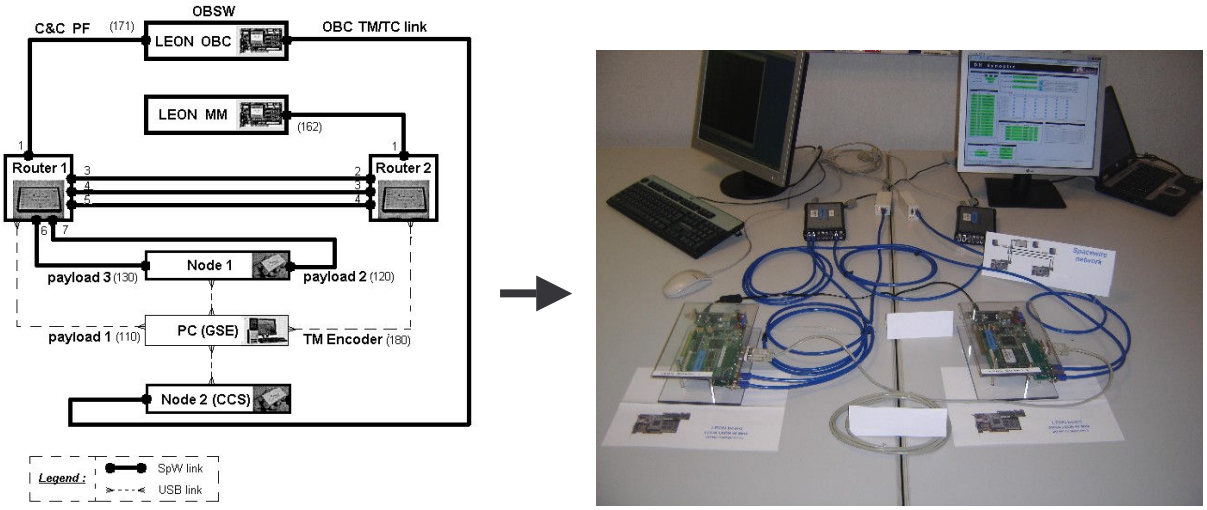


Figure 5 : SpaceWire mock-up overview

Once node behaviours have been simulated (MOST level) and breadboarded (mock-up level), an identical scenario is played for both networks, then their results are compared.

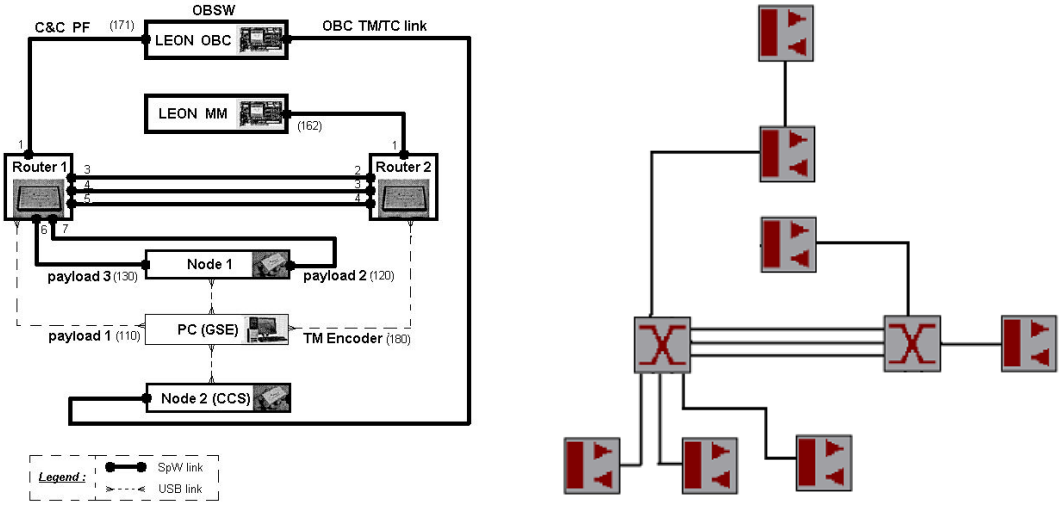


Figure 6 : Representation of the breadboard network on MOST

MOST validation is mainly based on SpaceWire standard features (for instance, packet size, packet generation, FCT generation, …) and the visualization of eventual bottlenecks and failures.

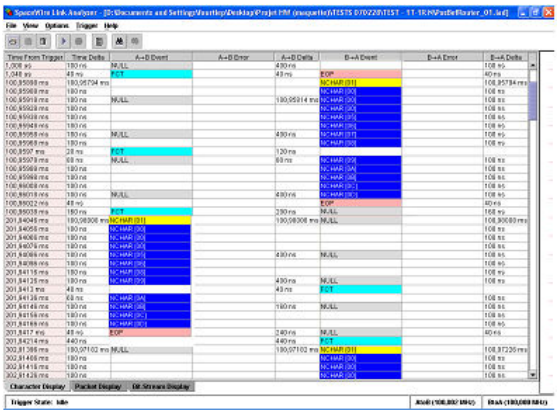


Figure 7 : Breadboarding network traffic on SpaceWire link analyzer

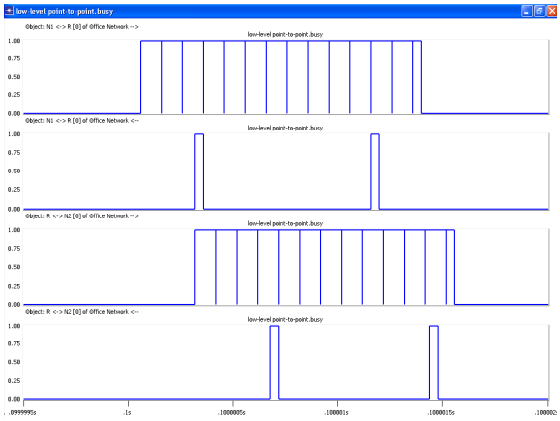


Figure 8 : Breadboarding network traffic simulated on MOST

MOST helps to identify slowing down factors observing the network behaviour in terms of data flows and allows to identify bottlenecks.

Avionics mock-up particularities

The internal avionics mock-up is based on an existing avionics software mock-up completed with SpaceWire nodes in such a way that it can be adapted to any SpaceWire network architecture.

Two high level protocols used for the mock-up have been tested :

- Remote Memory Access Protocol (RMAP),
- Packet Utilization Standard (PUS) protocol (including CCSDS protocol).

On the one hand, RMAP protocol is reserved to the routers during initial configuration and for their updates, if needed.
On the other hand, PUS protocol has been used to the traffic management and monitoring. Thanks to PUS protocol, our high level protocol includes a dedicated FDIR definition in order to make a failure monitoring from On-Board Computer. That permits to improve the control of the network health state without use of analyzers on each link, because it is unwieldy and expensive to have an analyzer for each link. The aim of this monitoring is to identify the lost packets, lost time-codes, EEP, bottlenecks … then to find, and even to track, the problem origin. It is why it is important to establish a strategic protocol to visualize the network behaviour with information exchanges which are correctly defined upstream.

This way, the avionics mock-up includes the PUS protocol based on following services : telecommand verification (1), HouseKeeping and diagnostic data reporting (3), event reporting (5), memory management (6), time management (9), on-board operations scheduling (11), on-board monitoring (12), large data transfer (13), packet forwarding control (14), test (17), on-board operation procedure (18), event action (19) and equally TAS PUS services dedicated to SpaceWire.

In the avionics software architecture, a virtual router has been implemented at middleware layer level to manage a redunded SpaceWire port at node level. This way, a specific routing table will be configured and updated, if needed. It permits to route packets to the redundant SpaceWire port if the nominal portfails.

At user application level, the avionics software mock-up has been developed to be in accordance with Spacecraft On-board Interface Services (SOIS) which are based on CCSDS_850.0-G-1 informational report. In fact, mandatory (and some optional) features of SOIS services have been implemented :

- at application support layer: network management, command/data acquisition, time access and message transfer services,
- at sub-network layer of the avionics software: packet, memory access and time distribution services,
- including the SpaceWire mapping in data link layer convergence functions (redundancy, prioritization and protocol multiplexing functions).

That provides a set of services to SpaceWire users compliant with CCSDS SOIS standard and completes the avionics software architecture.

Benefits

MOST brings support all along the project life, consolidating the design from early steps of the definition by an incremental approach. Once qualified, this tool will permit to support design/development of SpaceWire network in phase A/B, to verify FDIR concepts (at network level), to extend its use towards verification/validation in phases C/D, and if needed towards maintenance in phase E (including anomaly cases).

So network can be specified and breadboarded reusing the existing mock-up. That means it is possible to evaluate specification of :

- any SpaceWire network,
- any monitoring strategy to detect any bottleneck or permanent failure of a piece of the network,
- and any SpaceWire high level protocol (RMAP, CCSDS, PUS, …), synchronized or not.

Together they bring a continuous support for future developments during specification, design, development, assembly and validation phases of future projects.

Up to now, the network is synchronized via time-codes which provide features to build a high level protocol dedicated for the synchronization of SpaceWire networks. In a next future, MOST simulator and mock-up could test the feasibility of a synchronous and adaptive SpaceWire high level protocol.

Conclusion

The spacecraft avionics mock-up constitutes the best way to :

- build easily a representative SpaceWire mock-up for any space system application,
- specify any SpaceWire network including monitoring strategies (e.g. status and error messages),
- evaluate any high level protocol for any network architecture.

Moreover, it is important to guarantee that SpaceWire simulation run on MOST tool is deeply representative and efficient in order to provide a support for any project. In case we want to optimize the network performances, the nominal campaign of scenarios can be appended with specific elements, e.g. :

- some anomaly cases which are impossible to set in use on hardware bench, in order to verify FDIR concepts at network level,
- eventual improvements on SpaceWire high level protocol in order that SpaceWire networks become more robust : adaptive synchronous protocol.

This breadboarding mock-up and its possible SpaceWire improvements are an opening on the creation of hardware benches in order to realize a specific platform dedicated for SpaceWire networks. This SpaceWire platform could be able to optimize the design, development, and test of networks entirely SpaceWire.

SERIAL BACKPLANE FOR SPACEWIRE

Session: Networks & Protocols

Short Paper

Masaharu Nomachi, Shuuhei Ajimura

Osaka Univ. 1-1 Machikaneyama, Toyonaka, Osaka, Japan

E-mail: nomachi@lns.sci.osaka-u.ac.jp, ajimura@rcnp.osaka-u.ac.jp

ABSTRACT

ATCA and µTCA are serial backplanes introduced for telecom and embedded applications. They provide point-to-point star connections between modules. Because the connection is LVDS signals, most of serial data link protocols including SpaceWire can be hosted. PICMG (PCI Industrial Computer Manufacturers Group), which defines ATCA/µTCA specifications, assigns PCI express and Gigabit Ethernet and user area. We report an example to use the user area as SpaceWire connections. SpaceWire on the backplane will provide ideal prototyping for onboard modules. It is also useful for ground support systems.

1 INTRODUCTION

SpaceWire allows us to make flexible configurations. Because of its flexibility, we often see variety of board sizes and power supplies on one's desk. Here, in this paper, a standard for prototyping is proposed. In order to make prototyping easy, standardization of board size and power supplies are helpful. Moreover, SpaceWire interconnection over the backplane can makes system compact. Advanced Mezzanine Card (AMC) [1] and micro TCA [2] standards are candidates to host SpaceWire systems. In this paper we introduce such possibility.

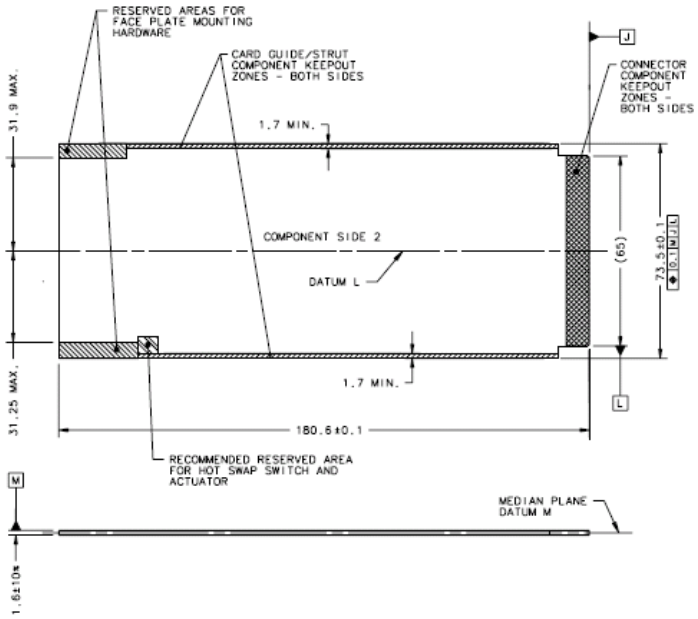


Figure 1 AMC Single Module PCB dimensions. The figure is taken from the reference 1 page 25.

2 AMC AND MICROTCA

Mezzanine cards are widely used for interface modules. So far, those have parallel interfaces. PCI mezzanine card (PMC) is commonly used now. Recent developments on Gigabit Ethernet and PCI express, serial interface becomes

preferable. Based on those requirements, new standard called AMC (Advanced Mezzanine Card) is developed by PICMG [1]. AMC has LVDS point to point interfaces. Micro TCA is hosing for AMC modules [2]. AMC modules can communicate over LVDS point to point link on the backplane.

2.1 AMC

AMC is designed for telecom applications. Therefore, many features about reliability are integrated. Although, those capabilities are not required on the present system, it will be useful if the system is integrated in a shelf. Board size of AMC single size card is 180.6 mm by 73.5 mm as illustrated in the figure 1. It is suitable for prototyping. It may have 20 differential input/output pairs. The backplane and connectors are designed for Gigabit data transfer. Thus, they have sufficient performance and may be also good enough for SpaceFiber. Some of those data links are recommended to be used for Gigabit Ethernet and PCI express. Possible assignment is shown in the figure 2. The "Fat pipe region" and "Common option region" are assigned for Gigabit Ethernet and/or PCI Express. After coexisting with Gigabit Ethernet and PCI express, AMC still has "extended option region" to be assigned for SpaceWire interconnections. We propose to use this region for SpaceWire interconnection.

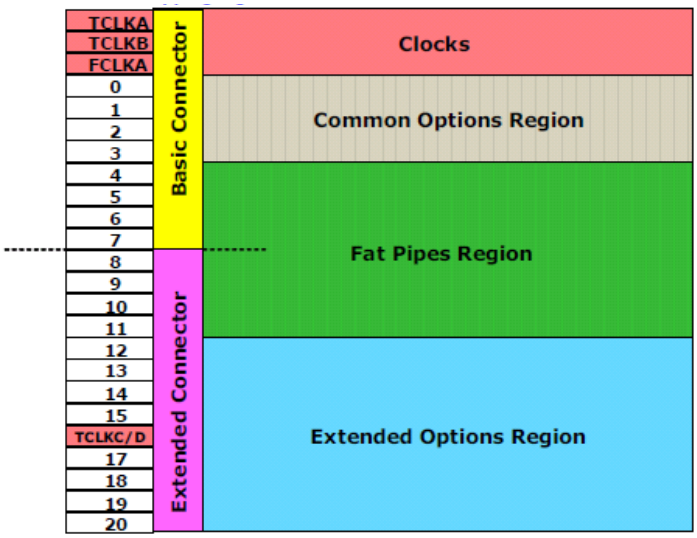


Figure 2AMC port mapping regions. The figure is taken from the reference 1 page 52.

Power supply for AMC module is 12 V. Maximum power dissipation on one single size board is 40 W. A point of load (POL) regulated power distribution is applied. AMC also has hot-swap capability. It makes developments easy.

2.2 Micro TCA

Micro TCA is a standard of shelf and backplane. AMC modules can be plugged into micro TCA system directly. Micro TCA provides power to AMC modules. It also provides interconnections over the backplane. Powerful system management capability is defined in the standard. It is a great advantage for ground applications for reliable system such as medical application. Micro TCA can hold around 10 AMC modules. The serial data links are connected to Micro TCA Carrier Hub (MCH). MCH works as a hub or a router module. A micro TCA can have tow MCH modules. It provides dual star configuration. One MCH may be a switch for PCI express for example. Another MCH can be a SpaceWire router module. Coexistence of SpaceWire and PCI express is not so difficult.

3 Applications

Standardization of board size and power supply makes module reusability high. Developing a new system, reuse of existing modules helps. Moreover, a system can be integrated in a compact chassis. It opens many ground applications. Micro TCA has some military products [3]. Micro TCA can be used not only on the ground but also on aero-applications and balloon-borne-applications.

4 Examples

AMC is originally developed for the mezzanine for Advanced TCA module (ATCA) [4]. ATCA is also defined at PICMG like AMC and micro TCA. Micro TCA is a smaller system standard of ATCA. ATCA also have serial backplane with larger board size.

In Osaka University, we have developed a 500 MHz FADC system on ATCA system [5]. The figure 3 is the FADC module. SpaceWire interconnection over the serial backplane is successfully demonstrated.

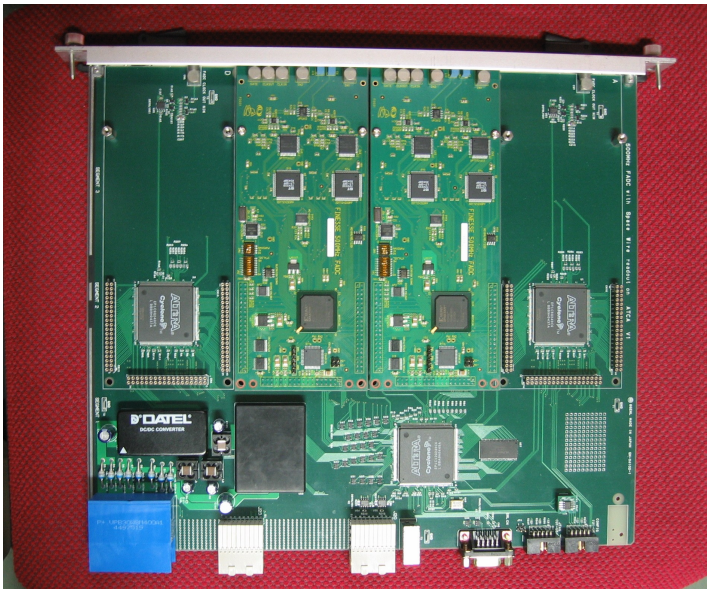


Figure3. 500MHz FADC module in ATCA board.

5 Summary

Many commercial systems in ground application will go forward to serial data link, probably PCI express. Thanks to the flexibility of AMC/micro TCA standard, SpaceWire interconnection can co-exit with those commercial modules. Use of such standard makes easy to test the system and is also good for ground applications.

6 References

1. PICMG® AMC.0 R2.0, Short Form Specification.

2. PICMG® MicroTCA.0 Draft Specification Release Candidate 1.0RC2, Micro Telecommunications Computing Architecture Base Specification

3. http://www.hybricon.com/

4. Advanced TCA PICMG 3.0 Short Form Specification, January 2003

5. Serial Data Link on Advanced TCA Back Plane
   Nomachi, M.; Ajimura, S.; Nuclear Science, IEEE Transactions on
   Volume 53, Issue 5, Part 2, Oct. 2006 Page(s):2849 - 2852

SPACEWIRE NETWORK FUNCTIONAL MODEL

Session: Networks & Protocols

Short Paper

Elena Suvorova, Liudmila Onishchenko, Artur Eganyan

Saint-Petersburg University of Aerospace Instrumentation. 67, B. Morskaya, Saint-Petersburg, Russia

E-mail: wildcat15@yandex.ru, luda_o@rambler.ru, artfla@rambler.ru

ABSTRACT

The complex of simulation programs for building models of SpaceWire distributed systems and investigation of their characteristics is presented. It includes a specification of basic SpaceWire network elements: a node, a routing switch and a link, allows to assemble a SpaceWire interconnection system of required structure, implements wormhole routing, time flow and distributed interrupts mechanisms, generation and transmission of data packets. This configurable tool enables to estimate an efficiency of SpaceWire based interconnections.

1 INTRODUCTION

An important task for devices design, network protocols development and distributed systems building is their simulation. For this purpose the configurable software simulation tool programs was developed. It implements SpaceWire network functional model (SpWNM). To input distributed system structure (topology) and their parameters the MS Visio based GUI is used. Software system model tool is written in SystemC. It could be used in different modeling environment where SystemC is supported, for example, in IUS(5.1–5.7) environment (Cadence Design Systems), under Linux Red Hat. Each type of network elements is implemented as an independent module, so a network of required topology could be composed from these modules without changing their programs. It is easy to learn this tool because of its interactive documentation.

2 SPWNM DESCRIPTION

The SpWNM package implements:

- data packets generation, receiving and transmission;
- generation, receiving, transmission and handling of control-codes (Interrupt-codes, Interrupt_Acknowledge-codes, Time-codes), NULL and FCT;
- Wormhole routing and symbol flow priority according to SpaceWire standard;
- Path, logic and regional-logic addressing;

- Adaptive group routing;
- Data packet blocking in switch where output port is unavailable (busy) or has buffer overflow;
- Timeout mechanism in switches and nodes;
- Error modeling at the channel level and credit errors.

SpWNM composes a SpaceWire network model according to a special input file generated by MS Visio. For this purpose the application in VBA was made. It allows users to compose distributed system model, their structure and parameters in a graphical way, using drag and drop and dialog windows. The generated by application system structure is saved in a file to be used by the systemC application. Thus user doesn't write a systemC model. All devices interconnections and their parameters settings are done automatically according to the input file. So complex distributed systems with big number of devices and links could be composed easily. To compose such a system as SystemC application by hand would be quite difficult.

SpWNM's structure and parameters which are set in Visio wholly describe SpaceWire network devices. A user can set types of generated distributed interrupts and parameters of their generation, types of distributed interrupts and parameters of their handling, signals transmission rate for every channel, parameters of time-codes' generation, timeouts values for switches and nodes, routing tables and adaptive group routing, packets' generation parameters in the nodes, parameters of channels error imitation. Setting of these properties is convenient for the user and are separately applied for each device.

Before simulation started the SpWNM can be configured for specific devices implementation that are used in it. Particularly, user can set a lot of time intervals required by the device to perform different actions – to process a signal which is received from input port, to write some value to the device's internal buffer, to read a value from this buffer, addressing to the application which is used by this device, etc. Besides times, the buffers size and numbers of input and output ports are configured for the every device separately. So there can be several devices of the same type but with different parameters and different numbers of ports in the system. It provides flexibility in models' building.

The SpWNM provides the user with ability to for writing a special program for each application. This program will receive and process data packets which are got by application over simulated SpaceWire network and return generated result. Each application is linked with a node in the network.

3 SPWNM SIMULATION RESULTS

The result of modeling is statistics that is stored during the system simulation. It consists of several files. Some files describe the history of system's simulation in special tables. Each table's row contains the time, the device ID, its action and additional information about it. These tables can be parsed, for example, using filters in Excel. Other files contain statistical data such as channels' workload by symbols of different types, average time of channels' blocking inside switches, time from sending

concrete type of distributed interrupt to receiving of this interrupt, time from sending answer for distributed interrupt to receiving of this answer, full time of each type of distributed interrupt handling used in the system. There is also information about average time from sending data packet which is sent by concrete node from its concrete port to receiving this packet. Information about data transmission errors in channels is also stored (these errors are imitated by the program, parameters of their generation is configured too). Diagrams can be built for all these results.

Automatic analysis of system simulation is performed by the special application. It reads generated xml-files (they are simultaneously generated with html tables), takes from them statistics and also detects errors which occurred in system simulation. The system simulation validity is checked according to all types of addressing and routing used in SpaceWire. Errors of receiving a control code that is not registered in the system are also detected – these errors can occur because of imitation of noises in channels.

4 SpWNM objectives

The SpWNM provides means to estimate a wide range of characteristics for different research. Many characteristics are set for every distributed system element so it is possible to find various dependences of estimated time characteristics from input parameters that could be important for a distributed system and a task set. Simulation could be used for different research, for example, to select a distributed system topology, to define parameters (distributed interrupts, time-codes, routing table, timeout values and so on) in a way, that required system characteristics correspond to requirements specification; to research load of every router, node and link caused by data flow and control-codes flow with different intensity; to validate a SpaceWire network analytical model.

In networks with wormhole routing a deadlock is possible. To solve this problem a correct routing table and timeout mechanism are used. These parameters strongly effects network capacity and data packet latency so it is useful to use simulation to select them.

The SpaceWire simulator can reflect some details of hardware implementation of real devices so it is possible to use simulation results during devices design. For example, we can investigate different output port arbitration schemes in data packet switch, or different buffer size, and so on.

5 Conclusion

The SpWNM provides an efficient tool for users to compose distributed system from ready made configurable modules and set their parameters in a simple way. As a result of simulation it is possible to estimate a wide range of characteristics that that are useful for research during building distributed systems, to define their parameters. The results can be used also to validate analytical models and devices in design.

REAL-TIME SIGNALING IN SPACEWIRE NETWORKS

Session: Networks & Protocols

Short Paper

Yuriy Sheynin, Sergey Gorbatchev, Ludmila Onischenko

St. Petersburg University of Aerospace Instrumentation

67 B. Morskaya str., 190 000, St. Petersburg, Russia

E-mail: sheynin@aanet.ru

ABSTRACT

In the paper we consider a complementary mechanism of Distributed Interrupts as additional type of control codes for the SpaceWire standard extension. Give main ideas of its operation, recovery in case of errors. Estimate latency characteristics of control codes distribution in SpaceWire networks, reason about timeout values for error recovery mechanism.

1. INTRODUCTION

SpaceWire is a prospective networking technology for building on-board interconnections in prospective satellites and spacecrafts. Based on it integral interconnection infrastructure can efficiently substitute several (3-5) separate interconnections that are used in modern distributed on-board systems: *sensor buses* for data streams from sensors and instruments; *command buses* for commands from control units to instruments and spacecraft equipment; *telemetry busses* for telemetry data; *data buses* for data exchange between computing modules in the course of data and signal processing. Time-codes - a specific SpaceWire standard feature that distinguishes it from one of its predecessors the IEEE 1355, give a facility to incorporate into SpaceWire interconnections on-board clock synchronization also, thus eliminating customary separate *time synchronization buses*.

Still, sideband extensions custom signals are often used for hard real-time signaling and control: system event signals (e.g. interrupts, error notification), data sampling, data transfer coordination, etc. To integrate such signal into SpaceWire interconnections also we proposed a complementary mechanism of Distributed Interrupts. Implemented as additional types of specific control codes, Interrupt codes, together with Time-codes of the origin SpaceWire standard, they form an extended set of control codes for low latency real-time signaling in SpaceWire interconnections.

The SpW control codes, due to their specification at the low levels of the SpaceWire protocol stack, are distributed by the same cables, channels that data packets are sent and switched, but their distribution does not depend on data packets flow intense and can traverse even blocked by data channels and paths. This core feature differs the SpaceWire from other high-rate interconnection standards and makes it most appropriate for real-time distributed systems interconnections.

2. DISTRIBUTED INTERRUPTS

To implement Distributed Interrupts we extend control codes by 2 additional ones. As the Time-Code, they are formed from ESC followed by a single data character, Firg.1.

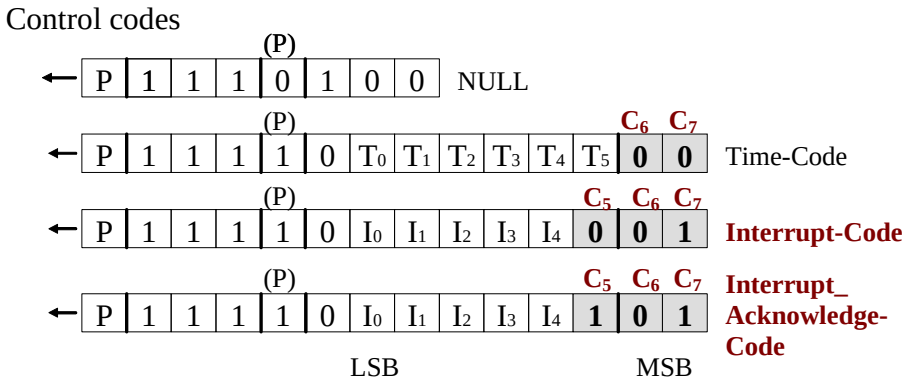


Fig.1. Extended control codes, with Interrupt/IntAck codes

Five bits of interrupt information are held in the least significant five bits of the Interrupt-Code (I0-I4); three most significant bits (C5=0. C6=0, C7=1) contain control flags that are distributed isochronously with the Interrupt-Code. Five bits of interrupt acknowledge information are held in the least significant five bits of the Interrupt_Acknowledge-Code (I0-I4); three most significant bits (C5=1, C6=0, C7=1) contain control flags that are distributed with the Interrupt_Acknowledge-Code.

Interrupt-Code represents a system signal request, e.g. an interrupt that is formed by a node in a SpaceWire network. It is issued by a node link that will be considered as the source node for this interrupt (Interrupt Source). It is distributed over the network to other nodes. An Interrupt-Code should be accepted for handling in some node of the SpaceWire network, which will be called the Interrupt Handler. The host of the node is supposed to implement some interrupt processing routine. One of 32 interrupt request signals (interrupt source identifiers) could be identified by the Interrupt-Code.

Interrupt_Acknowledge-Code represents a confirmation that the Interrupt-Code has reached some Interrupt Handler and has been accepted by it for processing. The Interrupt Handler node should send an Interrupt_Acknowledge-Code with the same five-bit interrupt source identifier as in the accepted Interrupt Code.

The node that sends an Interrupt-Code does not know where an Interrupt Handler in the network is. The Interrupt-Code is broadcasted to find an Interrupt Handler node. To eliminate infinite cycling of the broadcasted control code specific mechanisms for its handling in nodes and routers are provided. Each link controller of a node and each router contains one 32-bit Interrupt Source Register (ISR). When the link interface receives from its host an interrupt request with a five-bit interrupt identifier it sets appropriate bit to '1' in the 32-bit ISR. Then it sends out the Interrupt-Code with the five-bit interrupt source identifier field. If the correspondent bit in the ISR is in ‘1’ state already then the Interrupt-Code is not sent out. A subsequent Interrupt-Code with the same interrupt source identifier can be sent by the link only after receipt of an Interrupt Acknowledge with the correspondent interrupt source identifier.

When a node, which host system can be an Interrupt Handler, receives an Interrupt-Code it checks the correspondent bit in the 32-bit ISR to be reset to '0'; in this case it

sets the ISR bit to '1' and assert its INTR_OUT output signal at its interface with the host. When the link controller in a node receives from its host an of Interrupt Acknowledge signal with a five-bit interrupt source identifier it shall reset appropriate bit in the 32-bit ISR to '0' and then sends out an Interrupt_Acknowledge-Code.

In a router, when a link interface receives an Interrupt-Code it checks the correspondent bit in the ISR. If the bit is '0' it sets the ISR bit to '1' and the input port asserts its INTR_OUT output signal at the link controller interface; it is accompanied by the five-bit interrupt source identifier of the incoming Interrupt-Code. The signal propagates to all the router output ports (except the port that have issued the signal) so that they all shall emit the Interrupt-Code with the same five-bit interrupt source identifier field, which was received by the router. But if the correspondent bit in the 32-bit ISR is equal to '1' the Interrupt-Code will be ignored (to prevent repeated Interrupt-Code propagation in networks with circular connections); the router shall not retransmit the Interrupt-Code to its output ports.

In a link an a node or in a router the Interrupt -Code and Interrupt_Acknowledge-Code control codes shall be sent out as soon as the current character or control code has been transmitted. However, the Time-Code has priority for transmission over an Interrupt_Acknowledge-Code; an Interrupt_Acknowledge-Code has priority for transmission over an Interrupt-Code.

3. INTERRUPT CODES DISTRIBUTION RECOVERY IN CASE OF ERRORS

In a SpaceWire network faults and errors may occur: link disconnect error or parity error can cause an Interrupt -Code/Interrupt_Acknowledge-Code loss; there may be spontaneous change of an ISR bit state as a result of intermittent faults in a node or in a router. The Interrupt mechanism is made to be tolerant to them. SpaceWire networks with redundant links and circular connections (e.g. mesh, torus, fat tree) are tolerant to an Interrupt -Code/Interrupt_Acknowledge-Code loss: such an error will not stop the Interrupt control codes distribution to network nodes. To ensure tolerance against multiple faults and spontaneous changes in ISR special timers are used.

Each ISR in a node or in a router has a timer per ISR bit. A timer starts at the receipt of an Interrupt-Code with correspondent five-bit interrupt source identifier and resets at receipt of an Interrupt_Acknowledge-Code with the same interrupt source identifier. In case of timeout Ti before the timer is reset, the ISR timeout event arises; the correspondent ISR bit should be reset to ‘0’. In the Interrupt Source link, the link also sends an Interrupt_Acknowledge-Code with the five-bit interrupt source identifier that corresponds to the ISR bit. ISR reset timeouts recover Interrupt –Codes distribution for following interrupt requests, both after Interrupt–Code and Interrupt_Acknowledge-Code losses.

4. CONTROL CODES DISTRIBUTION LATENCIES AND TIMOUTS

Efficiency of SpaceWire real-time signalling mechanisms depends on latency characteristics. They depend on several factors: link bit rates, router architecture, topology of the network interconnection, control codes flow rates. Taking some indexes for these factors we can estimate the control codes characteristics. Let $\boldsymbol{T_D}$ be worst propagation time in the network with diameter $\boldsymbol{D}$ (depends upon the SpaceWire network interconnection topology); $\boldsymbol{T_{bit}}$ – one bit transfer time; $\boldsymbol{T_{wtc}}$– Time-code

transport through router delay (ignoring interference with previous characters/codes; depends upon implementation); T_H – delay in an Interrupt Handler node that should send an Interrupt_Acknowledge-Code (depends upon implementation).

The Time-code has highest priority among control codes and in a hop it can wait only completion of a previous character or code transmission. Its maximum time-code delivery delay, T_{Tmax}, is:
$T_{Tmax} = (T_{wtc} + 13\ T_{bit})*(D\text{-}1) + (14\ T_{bit})*D = T_{wtc}\ (D\text{-}1) + T_{bit}\ (27\ D - 13)$.
For D=5, T_{wtc} = 200 ns it will give T_{Tmax} = = 1,1 μs.

Several Interrupt codes (up to 32 ones) can run concurrently in the network, and theoretically can come simultaneously to a router, thus forming a queue. They will have to let pass ahead a Time-code that can also appear at the moment. So the maximum Interrupt-code delivery delay, T_{Imax}, will be rather pessimistic:
$T_{Imax} = (D\text{-}1)*(T_{wtc} + 13T_{bit} + 14T_{bit} + 31*14*T_{bit}) + (14T_{bit})*D =$
$(D\text{-}1)*(T_{wtc} + 461T_{bit}) + 14T_{bit}*D$. For D=5, at 400 Mb/s it will give T_{Imax} = 5,6 μs. However, such situation is quite artificial. As a worst propagation time T_D it is reasonable to take more realistic estimations. Let q be an average number of waiting control codes in a router at the moment of the Interrupt code arrival. Then an estimation will be $T_D = (D\text{-}1)*(T_{wtc} + 27T_{bit} + 14\ q*T_{bit}) + (14T_{bit})*D$. For the same data, it will give more realistic picture, fig. 2; for q less then 6, T_D will be under 2μs.

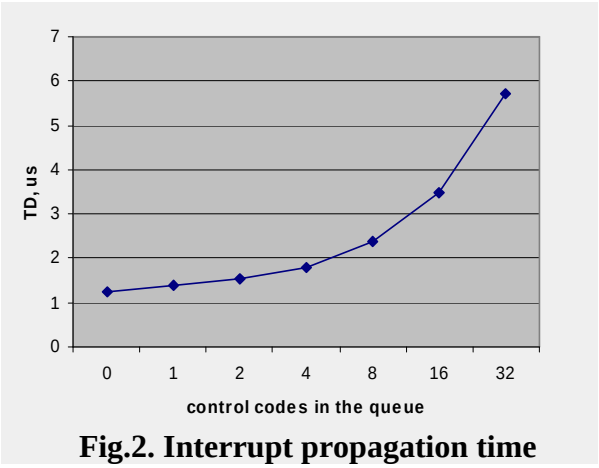


Fig.2. Interrupt propagation time as a function of control codes queue

Timeout intervals for SpW RT signals should be enough to ensure that right Interrupt/IntAck codes will reach their destination. Thus, for an Interrupt Source link the reset timeout $Ti=T_1$ shall be $T_1 > 2*T_D + T_H$. Reset timeout for routers and non-Interrupt-Source nodes can be set $Ti=T_2$, where $T_2 \geq T_1$. Specific constraints are set on relationship between T_D and T_H : $T_H > 2*T_D$. It is required to prevent cycling in some particular cases with possible Interrupt-code and Interrupt_Acknowledge-code with the same Interrupt ID simultaneous distribution in networks with cycles.

5. REFERENCES

1. ECSS-E-50-12A “SpaceWire - Links, nodes, routers and networks”, European Cooperation for Space Standardization (ECSS), 2003, 124 p.

2. Sheynin Y., Solokhina T., Petrichkovitch Y., SpaceWire Technology for Parallel and Onboard Distributed Systems. “Electronika”, 2006, №5, pp.64-75, 2007, No.1, pp.38-49 (in Russian).

Test & Verification 1

Wednesday 19th September

9:00 – 10:30

SPACEWIRE CABLE AND

CONNECTOR VARIATIONS

Session: Test & Verification

Long Paper

Shaune S. Allen

NASA Goddard Space Flight Center, Greenbelt, MD, 20771

E-mail: Shaune.S.Allen@nasa.gov

ABSTRACT

SpaceWire applications have grown steadily since the introduction of the standard and with them come new and varied requirements. In many cases, these requirements make it difficult to implement the data protocol in the standard method. The need to adapt to these new applications has fuelled the natural growth and development of the standard. The basis of SpaceWire circuits is the physical layer or level, which may consist of the SpaceWire Cable assemblies, including the bulk cable and the SpaceWire connector. This paper summarizes testing done on both standard and non-standard SpaceWire physical layer components, which are suitable for SpaceWire applications and in some cases necessary for systems that have strict performance requirements.

Comparisons will be made of standard SpaceWire cable and non-standard AWG 26 SpaceWire cable. Additionally, comparisons will be made between the standard SpaceWire connector and non-standard Twinax connectors, which offer potential applications through bulkhead interfaces and other applications that must meet tight performance specifications. These new components offer the potential for suitable applications and should be considered as suitable alternatives to the SpaceWire connector and cable.

VIRTUAL SATELLITE INTEGRATION AND THE SPACEWIRE INTERNET TUNNEL

Session: SpaceWire Test and Verification

Long Paper

Stuart Mills

STAR-Dundee,

c/o School of Computing, University of Dundee, Dundee, DD1 4HN, Scotland, UK

Steve Parkes

University of Dundee,

School of Computing, Dundee, DD1 4HN, Scotland, UK

Raffaele Vitulli

European Space Agency,

ESTEC, Keplerlaan 1, 2201 AZ Noordwijk, The Netherlands

E-mail: stuart@star-dundee.com, sparkes@computing.dundee.ac.uk, Raffaele.Vitulli@esa.int

ABSTRACT

A major area of risk in spacecraft development is the integration of sub-systems late in the development cycle. Sub-systems are often developed by different organisations in different countries, making it expensive to bring them all together for testing. Although the SpaceWire standard has helped reduce incompatibility problems at the data link and physical layers, there is still the potential for problems at the higher layers. These problems may only be found during integration testing, which is late in the development cycle, and may therefore be very costly to address.

This paper presents a solution to reduce that risk: virtual satellite integration and the SpaceWire Internet Tunnel. Virtual satellite integration is a process whereby the sub-systems of a satellite are brought together virtually for testing. The SpaceWire Internet Tunnel is a tool which makes it possible to virtually integrate sub-systems which are to be connected using SpaceWire.

1 INTRODUCTION

1.1 LIMITATIONS OF EXISTING METHODS OF INTEGRATION

A common feature of spacecraft development, particularly in Europe, is that the partners involved are separated by great distances. Development teams may be in different states, countries or continents, and may be separated by culture and language, in addition to their geographical separation.

To cope with these boundaries and ensure the correct operation of the system, a great emphasis is placed on using documentation during spacecraft design. Interface specifications are written for each subsystem and the accuracy of these documents is of great importance. Any errors or inaccuracies in these specifications could result in sub-systems being developed which cannot communicate without major modification.

Once the sub-systems have been developed, integration and testing is performed. This requires each unit to be moved to a common location. Any faults or errors identified at this late stage can be very costly to correct. This late integration is therefore a major area of risk in any project.

One method of reducing this risk is through the use of standard interfaces. SpaceWire [1] [2] provides a standard for the physical and data link layers of the interface, greatly reducing the likelihood of issues at this level during integration testing. However, there is still the potential for problems at the higher layers, for example in the interface between applications, each of which may be developed and tested in isolation.

1.2 VIRTUAL SATELLITE INTEGRATION

Virtual satellite integration [3] can be used to greatly reduce these risks. It provides a means of interconnecting the sub-systems at an earlier stage of the development life cycle, without moving the units to a common location. Instead the Internet is used to remotely connect the sub-systems, allowing them to be virtually integrated.

An example of a very simple spacecraft network using SpaceWire and containing two sub-systems is shown in Figure 1. These sub-systems could be developed by separate organisations in different countries.

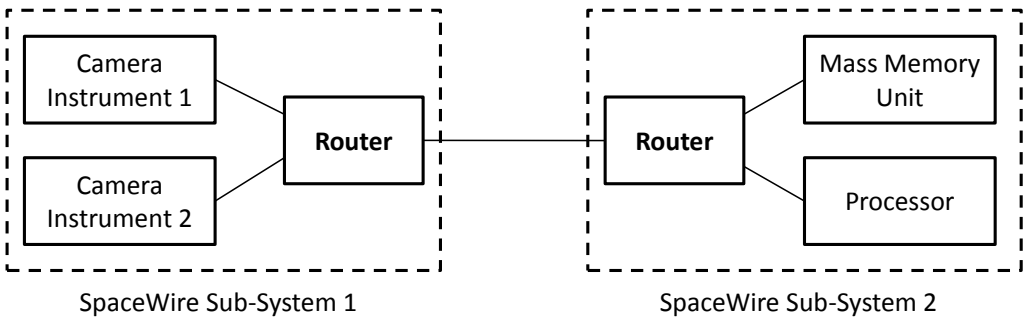


Figure 1: SpaceWire network with two distinct sub-systems

In the past these sub-systems may not be tested together until late in the development cycle. Using virtual satellite integration, these two sub-systems can be tested together

at a much earlier stage, as shown in Figure 2. In this updated network diagram, the link between the two routers has been replaced with links to two SpaceWire IP-Tunnel devices each connected to a PC, with the PCs connected over the Internet. Each PC runs the SpaceWire IP Tunnel software which manages the connection between the two PCs, sending and receiving traffic on the Internet connection and the SpaceWire link.

Virtual satellite integration can also be used to test sub-systems with simulators. In Figure 2, for example, sub-system 1 could be replaced with a camera simulator in order to test sub-system 2.

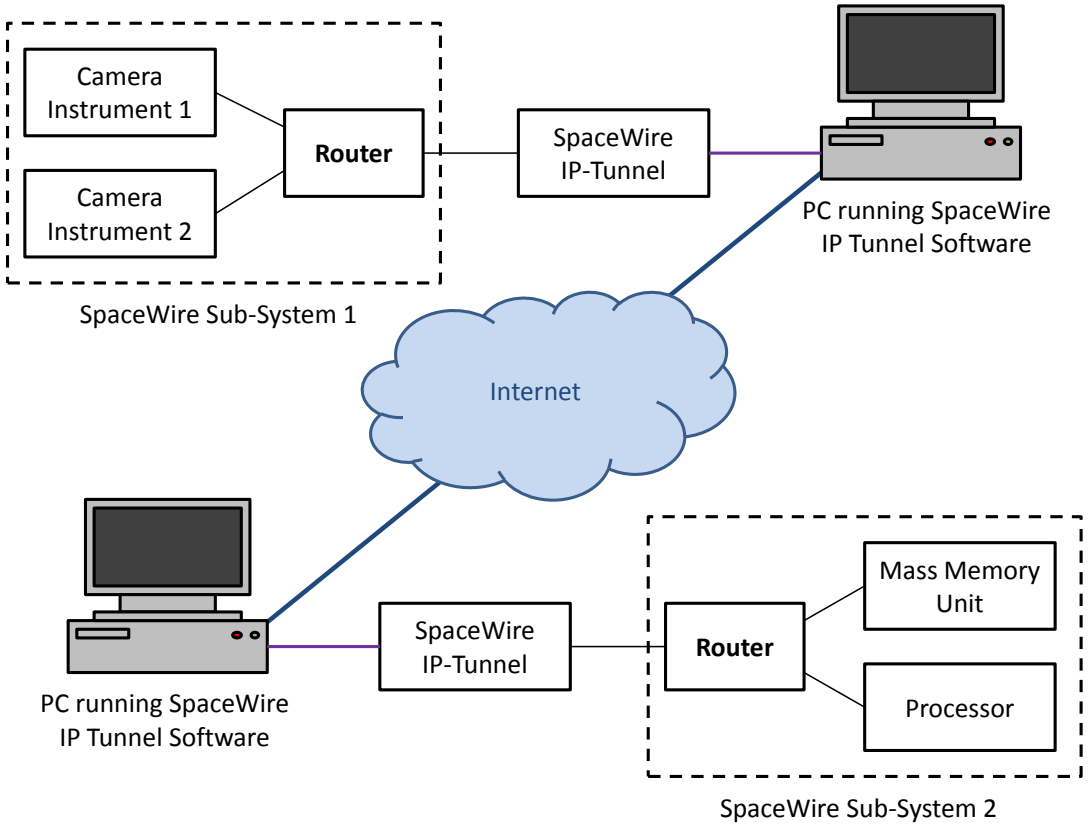


Figure 2: SpaceWire Internet Tunnel between two distinct sub-systems

There are some scenarios in which the use of virtual satellite integration is not appropriate. As a network such as the Internet is used to replace a link in the network, traffic passing across this link will be subjected to the bandwidth limitations and latency issues of this network. The SpaceWire Internet Tunnel provides features to limit the effect of these problems but despite this, virtual satellite integration may not be suitable for systems in which traffic must be transferred within a bounded time period, for example. Many systems can be used with the Tunnel, however, and indeed the slow response times of the Internet can often be useful in identifying problems in devices and applications.

2 THE SPACEWIRE INTERNET TUNNEL

The SpaceWire Internet Tunnel [4] consists of hardware to send and receive traffic from the SpaceWire link, and software to route traffic between the hardware

component and the Internet. Protocol Analysis software is also provided to allow the traffic crossing the Tunnel to be monitored.

A SpaceWire Internet Tunnel transparently replaces a SpaceWire link. All traffic entering a Tunnel will exit the Tunnel in exactly the same order. This means, for example, that a time-code will exit the Tunnel between the same two data characters it entered.

2.1 SpaceWire IP-Tunnel Hardware

The hardware component of the SpaceWire Internet Tunnel requires a STAR-Dundee SpaceWire IP-Tunnel device [5]. The reason for the use of a specialised device is that the device does not simply send and receive packets. The Space IP-Tunnel device must perform a number of tasks in order to cope with the expected reduction in bandwidth and increase in latency introduced by the Internet. This is necessary to avoid timeouts at routers, for example, which occur when the time between two data characters in the same packet exceeds a defined limit.

The device must also treat traffic differently than the SpaceWire-USB Brick [5] on which it is based. In a normal SpaceWire device, time-codes should be treated with higher priority than data packets. However, a SpaceWire Internet Tunnel must not alter the order of time-codes and data characters. Similarly, link start and disconnect events must be passed up to the application, in sequence with data characters and time-codes.

2.2 SpaceWire IP Tunnel Software

The SpaceWire IP Tunnel application is written in Java, which means it can be run without recompilation on any platform on which Java is supported and a driver is available for the SpaceWire IP-Tunnel device. Currently Windows and Linux drivers are provided. This driver provides the interface between the application and the hardware and is an updated version of the SpaceWire USB Driver provided with STAR-Dundee SpaceWire-USB Bricks and SpaceWire Router-USBs [5]. Modifications to the driver were required in order to support the interface used by the Tunnel device. The updated driver also keeps data characters, time-codes and link start and disconnect events in sequence when passing them up to the user. In addition, changes were made to the driver to improve performance with the unusual traffic flow produced by the SpaceWire IP-Tunnel device, when compared with the SpaceWire-USB Brick and SpaceWire Router-USB.

When the user creates a Tunnel in the SpaceWire IP Tunnel application, a secure connection is established over the network to the other end. The secure connection ensures that no-one can view or modify the traffic crossing the network, while a password can be provided for each Tunnel to provide further authentication. Any form of network interface (typically Ethernet) can be used by the Tunnel software, provided there is a network driver available for the operating system.

Once the connection is established, traffic received on the SpaceWire link is sent over this network connection to the other end. At the same time, the software is also receiving traffic from the network connection, which it then sends over the SpaceWire

link. Traffic received from both the SpaceWire link and the network connection is also passed to the SpaceWire Protocol Analyser for analysis.

2.3 SpaceWire Protocol Analyser

As the SpaceWire Internet Tunnel is intended for use during testing, it is important to be able to identify and investigate any problems encountered while performing a test. The SpaceWire Protocol Analyser is integrated in to the SpaceWire IP Tunnel software and allows the user to view the statistics of traffic crossing the Tunnel in real time, as shown in Figure 3.

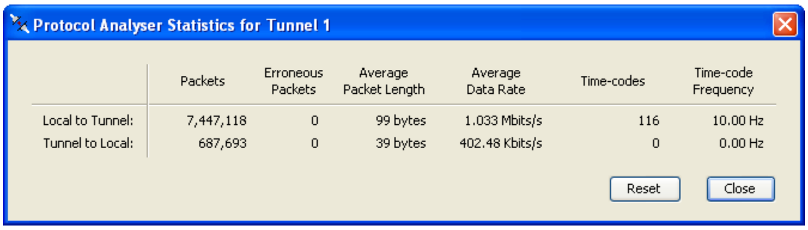


Figure 3: Protocol Analyser Statistics dialog

Traffic crossing the Tunnel can also be recorded using the Protocol Analyser. The type and characteristics of traffic to be recorded (e.g. data packets with a particular address or data pattern, or time-codes with specific flag values) can be specified, as can a trigger point and the amount of traffic to record before and after the trigger point. Once traffic is recorded, it is displayed in the Protocol Analyser for further analysis, as shown in Figure 4.

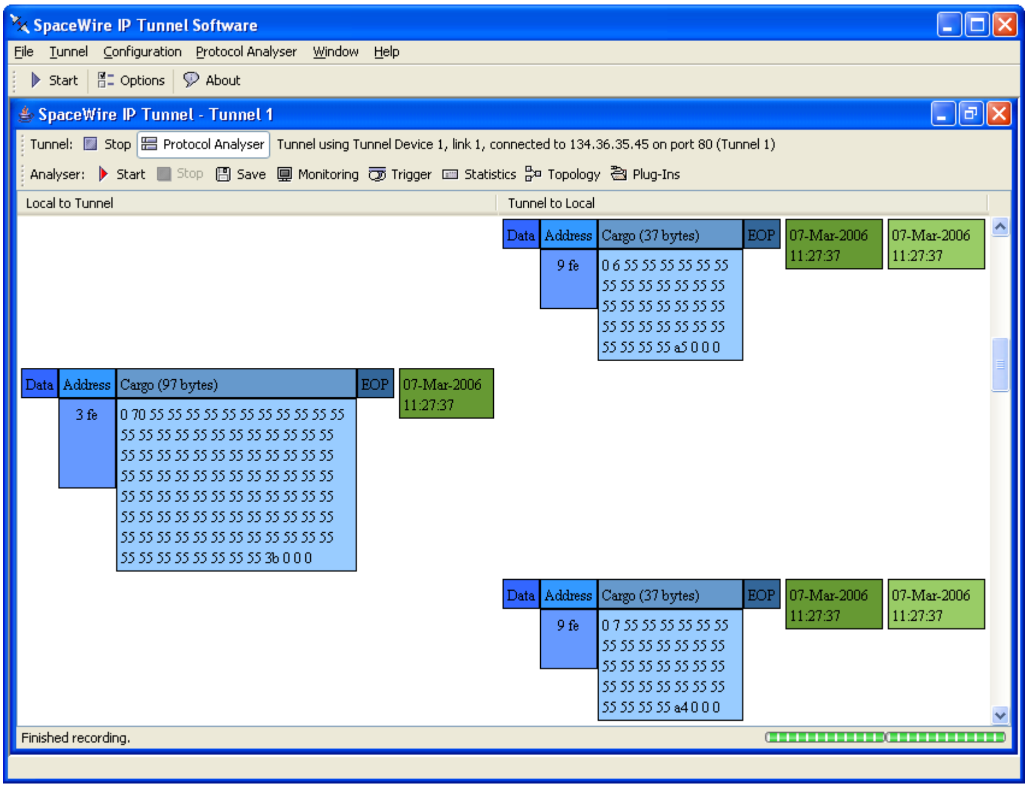


Figure 4: Protocol Analyser showing recorded traffic

The Protocol Analyser also allows higher layer protocol plug-ins to be loaded. These can be written by users to identify and display packets containing a particular higher layer protocol. With a higher layer protocol plug-in loaded, the Protocol Analyser can be used to record and/or trigger on packets containing the protocol defined by the plug-in. Recorded traffic containing the protocol is also displayed using the formatting information provided by the plug-in.

Figure 5 shows traffic recorded by the Protocol Analyser containing Remote Memory Access Protocol (RMAP) [6] packets. The RMAP plug-in (which is provided with the SpaceWire IP Tunnel and Protocol Analyser Software) has been used to format the packet, and so the fields in the RMAP packet can easily be viewed without the need to look at each of the individual data bytes to determine a field's value.

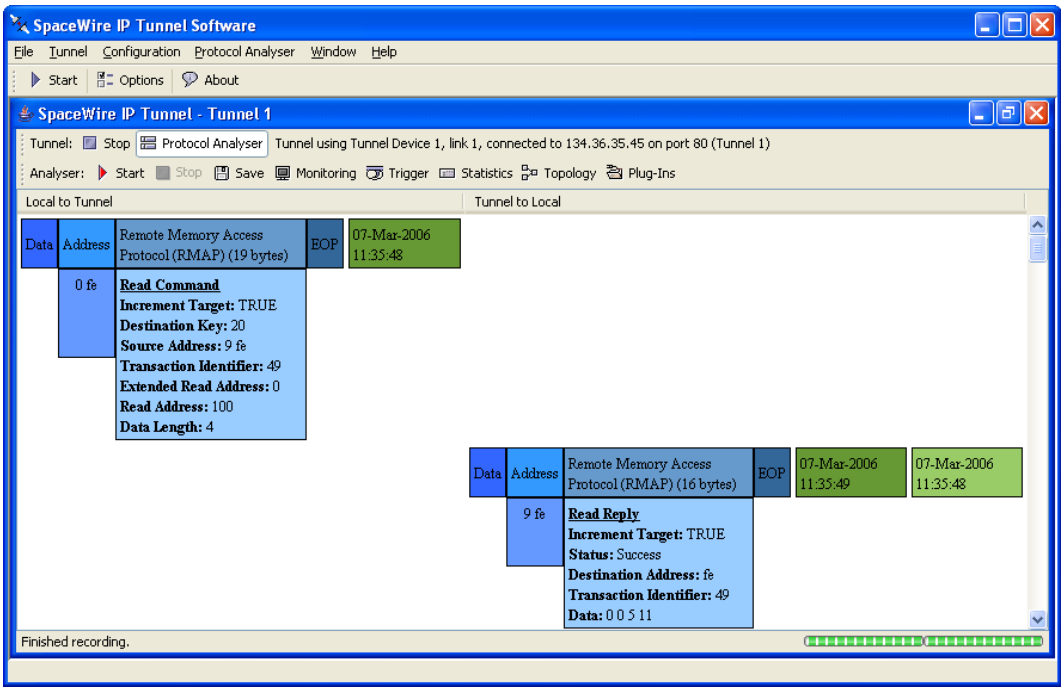


Figure 5: Protocol Analyser showing recorded traffic formatted by the RMAP plug-in

As users can write their own protocol plug-ins, it can be much easier to monitor the traffic crossing the network. For example, if an application sends packets in a particular format, someone can write a higher layer protocol plug-in describing that format, which can then be loaded by anyone using the Tunnel software. Protocol plug-ins are written in Java and can be very simple or as complex as is required. Example source code is provided to assist in development of new plug-ins.

3 THE PILOT ACTIVITY

An ESA pilot study is currently being undertaken to test the suitability of the SpaceWire Internet Tunnel and virtual satellite integration. This study involves a number of organisations in different countries, each using tunnels to test concepts, hardware and software. On completion of their experiments they will report back on their experiences.

It is hoped that in addition to proving how useful virtual satellite integration can be, the pilot study will also identify any issues and possible areas of improvement within

both the concept and the hardware and software in use. Although the pilot activity is still in its early stages, already some of the organisations involved have successfully tested Tunnels between their locations and a network set-up at the University of Dundee.

4 SpaceWire Internet Tunnel Server

A limitation of the existing SpaceWire Internet Tunnel is that firewalls in some organisations are configured to stop the Tunnel accessing the Internet, and actions must be taken to allow a Tunnel to pass through the firewall. The reason for this problem is that one end of the Tunnel must operate as a server while the other operates as a client. The server end requires the same firewall settings as a web server, for example, which may not be possible in some organisations.

To address this problem, development has started on a SpaceWire Internet Tunnel Server. A PC acting as a Tunnel Server will run software which accepts connections from clients and establishes connections between pairs of clients. The software will then route all traffic from a client to the other client in the pair. As both ends of the Tunnel are now clients the problems with firewalls should be eliminated. The Tunnel Server must be situated outside the firewall, or given the same access permissions required by a Tunnel client acting as a server. However, a single Tunnel Server will be capable of handling a number of Tunnels, so a single server could handle Tunnels from multiple organisations.

5 SpaceWire Internet Tunnel Improvements

Although the SpaceWire Internet Tunnel and Protocol Analyser software performs the task it was designed for, there are a number of improvements that could be made. After some time identifying possible enhancements, development will soon commence on improving the software. These improvements include some minor bug fixes to the Tunnel, attempts to improve performance, and the addition of new features to both the Tunnel and Protocol Analyser.

New features which are being considered include:

- A chat window for communication between the users at each end of a Tunnel
- Support for proxy servers
- Improved facilities for viewing recorded traffic
- Play back of recorded traffic
- Context sensitive help for all areas of the software

6 Summary

The SpaceWire Internet Tunnel is now available to buy from STAR-Dundee. Tunnelling has been successfully demonstrated between a number of locations across Europe. The Pilot Activity will demonstrate and test the virtual satellite integration concept more comprehensively. The feedback from these experiments will drive future improvements to the SpaceWire Internet Tunnel, some of which are currently under development.

Following recent improvements, data rates greater than 25 Mbits/s have been viewed using the Tunnel. As such rates are unlikely to be achieved over the Internet and most wide area networks, the Tunnel hardware and software are unlikely to be the cause of any limits in the data rate when tunnelling. The SpaceWire Tunnel Server will make it easier for users behind firewalls to use the Tunnel, while the improvements being made to the Protocol Analyser will mean that more detailed analysis of tunnelled traffic can be performed.

A demonstration of the SpaceWire Internet Tunnel will be provided at the SpaceWire Conference.

7 ACKNOWLEDGEMENTS

The authors would like to acknowledge the support of ESA for the work done on the SpaceWire Internet Tunnel and Tunnel Server at the University of Dundee.

8 REFERENCES

1. European Cooperation for Space Standardization, Standard ECSS-E-50-12A, "SpaceWire – Link, Nodes, Routers and Networks", Issue 1, European Cooperation for Space Standardization, January 2003.

2. European Space Agency, "SpaceWire Web Page", European Space Agency, http://spacewire.esa.int.

3. S. M. Parkes, "Virtual Satellite Integration", Proceedings Data Systems in Aerospace (DASIA) conference, Nice, France, May 2001.

4. S. Mills, S. M. Parkes and R. Vitulli, "SpaceWire Internet Tunnel", Proceedings Data Systems in Aerospace (DASIA) conference, Edinburgh, Scotland, May 2005.

5. STAR-Dundee, "STAR-Dundee Web Site", STAR-Dundee, http://www.star-dundee.com.

6. European Cooperation for Space Standardization, Draft Standard ECSS-E-50-11, "SpaceWire Remote Memory Access Protocol", Draft F, European Cooperation for Space Standardization, November 2006.

SPACEWIRE CABLE CHARACTERISATION

Session: SpaceWire Test and Verification

Long Paper

Martin Suess

European Space Technology Centre, PO Box 299, 2200 AG Noordwijk ZH, The Netherlands

Steve Parkes, Paul Crawford

Space Technology Centre, University of Dundee, Dundee, DD1 4HN, Scotland, UK

E-mail: martin.suess(at)esa.int, sparkes(at)computing.dundee.ac.uk, psc(at)sat.dundee.ac.uk

ABSTRACT

The SpaceWire physical layer comprising twisted pair cables and micro-miniature D-type connectors is specified in detail in the SpaceWire standard ECSS-E-50-12A. In some cases the physical layer defined in the standard is not fulfilling all requirements of a specific application. The cables as defied in the standard are quite rigid and not handy for use in the laboratory and for SpaceWire connections over very long distances (>10 m) cables with lower loss are required. For this purpose cables with stronger gauge as well as different types of connectors have been proposed. In order to be able to assess the consequences of these modifications a test setup for SpaceWire cable and connector characterisation has been developed and a set of performance parameters to be measured have been defined. The testing uses a Vector Network Analyser (VNA) and a set of specially developed interface boards to measure S-parameters. From these S-parameters the performance figures for return loss, insertion loss, near- and far-end crosstalk can be derived. The paper describes the different test setups and how the performance figures are obtained from the measurements.

1 SPACEWIRE CABLE SPECIFICATION

The SpaceWire cable and connector is specified in the SpaceWire standard [1] in the chapters 5.2 and 5.3. For the cable a detailed specification of the construction is provided in the standard (Figure 1). It specifies the diameter of the twisted pair conductors and of the wires in the braided shields, the materials used for the filler and the protective sheath etc. as well as the physical properties of the resulting cable like diameter, bent radius, mass and colour.

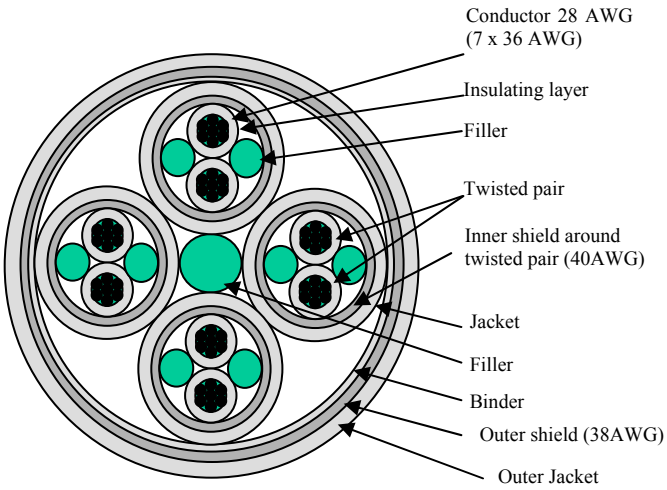


Figure 1: Section though a SpaceWire cable

While such a detailed specification ensures that any cable constructed accordingly will fulfil the performance requirements and directly provides mass, diameter and bent radius figures for the system engineering this kind of specification does not leave room for further improvements or other specifically adapted cable solutions.

In terms of electrical requirements only the characteristic differential impedance (100±6 Ω), the DC resistance of the inner conductors (≤ 256 Ω/m) and the skew between the differential signal pairs (≤ 0.1 ns/m) are specified. The performance in terms of insertion loss over frequencies and cross talk between the twisted pairs is only specified implicitly by the detailed requirements on the construction of the cable.

One objective of the test setup described here is to get a complete electrical characterisation of the SpaceWire compliant cables.

2 S-PARAMETER DEFINITION

Scattering or S-parameters can completely characterise linear networks with respect to their electrical behaviour as a function of frequency. They are commonly used when describing and designing RF and microwave components and systems. Today they are measured with the help of a VNA up to frequencies beyond 100 GHz.

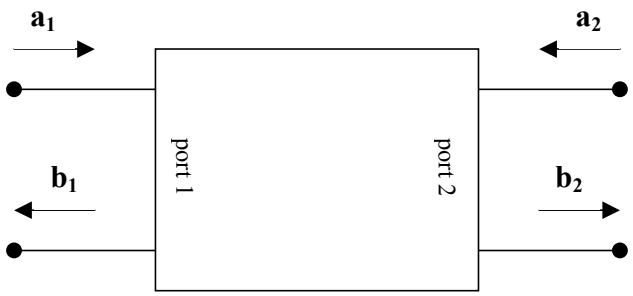


Figure 2: 2-port network with incident and reflected power waves

A 2-port linear network can be described with the complex scattering matrix containing 4 S-parameters which are calculated from the incident and reflected power waves which are measured by the VNA.

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \quad (1)$$

S_{11} and S_{22} are commonly called return loss while S_{12} and S_{21} are usually named insertion loss for passive networks.

3 ELECTRICAL PERFORMANCE PARAMETERS

The need for the electrical characterisation of twisted pair cables is not unique for SpaceWire. Twisted pair cables are widely used in computer networks in offices like Ethernet. The characterisation of this type of cables is specified in two IEC standard documents [2] and [3]. The most important electrical measurements defined in the standards are Return Loss, Insertion Loss, Near-end Crosstalk (NEXT) and Far-end Crosstalk (FEXT). All these electrical characteristic is measured as a function of frequency with the use of an VNA. One difficulty when using a VNA for this type of measurements is to go from the single ended configuration of the VNA to the balanced configuration needed for the twisted pairs. This is achieved by using baluns (balanced to unbalanced RF transformers) which have a sufficient high bandwidth for the measurements. The test setup for these measurements as defined in [2] and [3] is described in the following.

3.1 RETURN LOSS

The return loss is the ratio of power delivered to power reflected at the input port of the cable assembly, when the far end is terminated with its nominal characteristic impedance.

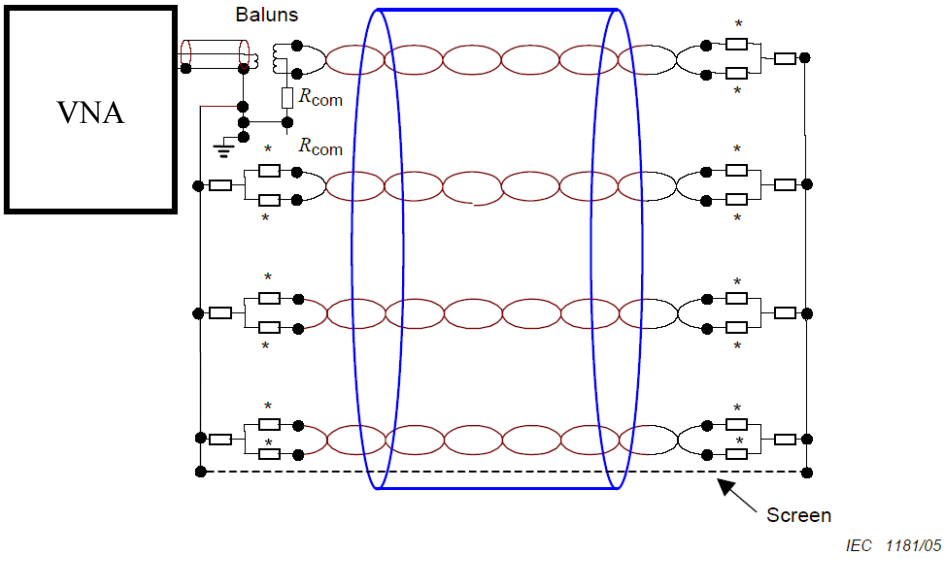


Figure 3: Return Loss test configuration according to ICE 61935-1

The return loss measured S_{11} can be directly translated into a measurement of the differential impedance Z_c of the cable and its termination.

$$|S_{11}| = 20\log_{10}\left|\frac{100\,\Omega - Z_c}{100\,\Omega + Z_c}\right| \tag{2}$$

This test setup could also be used to determine the quality of a SpaceWire receiver termination in SpaceWire equipment.

3.2 INSERTION LOSS

Insertion loss is measured by determining the signal loss of the cabling element under test, relative to the signal loss of a short connection between the test ports of the Vector Network Analyser.

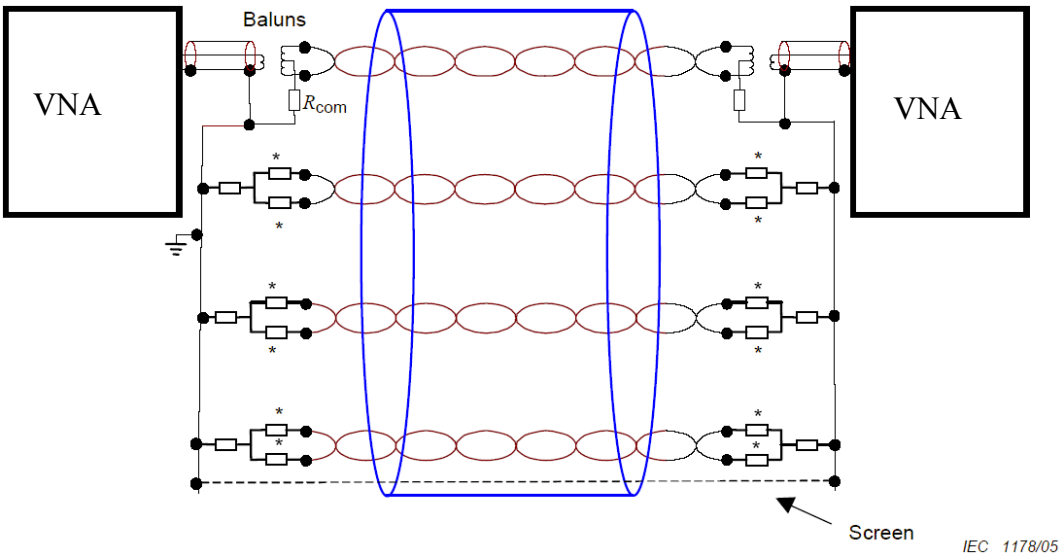


Figure 4: Insertion Loss test configuration according to ICE 61935-1

Each of the twisted pair cables is measured subsequently while the other twisted pairs are terminated at both sides with the reference impedance.

The same insertion loss measurement S_{12} can be used to determine the skew between the data and strobe lines. For this the phase of the complex scattering parameter S_{12} is unwrapped and compared with the phase in the S_{12} of the second twisted pair. A skew between the two twisted pairs will show up as an additional linear phase component as a function of frequency.

3.3 NEAR-END CROSSTALK (NEXT)

The objective of this test is to determine the coupling between a transmitted signal applied at the near end of one twisted pair to the received signal at the near end of a different twisted pair.

NEXT is measured by applying the signal at the near end of one twisted pair and measuring the coupled signal S_{12} at the near end of a different twisted pair. The coupling between the transmitting twisted pairs and the receiving twisted pairs which is of particular interest requires two times four separate measurements.

All cables need to be terminated with a connector at the far end with matched loads at each pair. Pairs that are not used in the measurement shall have terminations at the near end as well.

The over all influence of the transmitters on the receivers of the near end is described by the Power Sum NEXT (PSNEXT) which can be calculated but summing up the linear values (not dB) of the measured pair to pair NEXT.

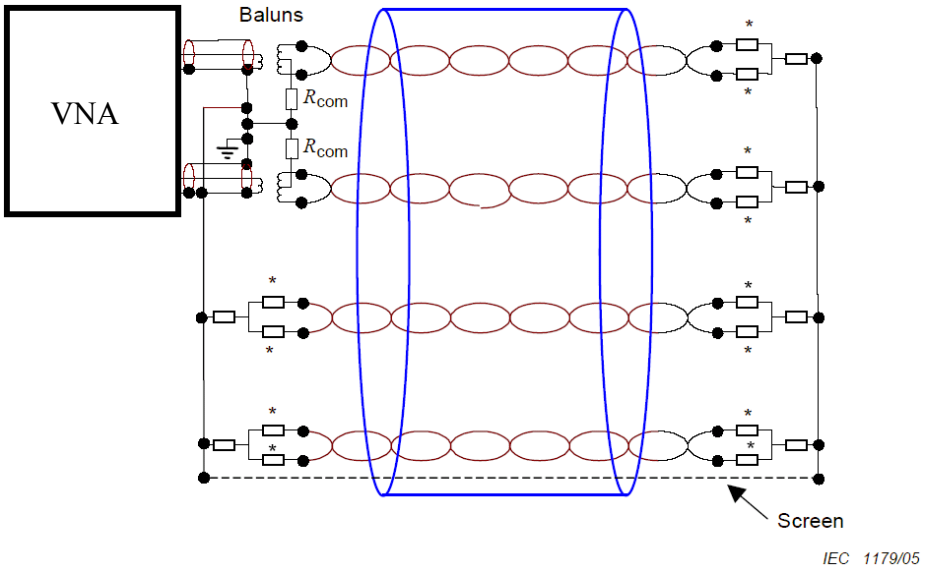


Figure 5: Near-end Crosstalk test configuration according to ICE 61935-1

3.4 FAR-END CROSSTALK (FEXT)

The objective of this test is to determine the coupling between a signal applied at the near end of one wire pair to the signal received at the far end on a different wire pair by measuring the insertion loss S_{12} between the two twisted pairs at different ends.

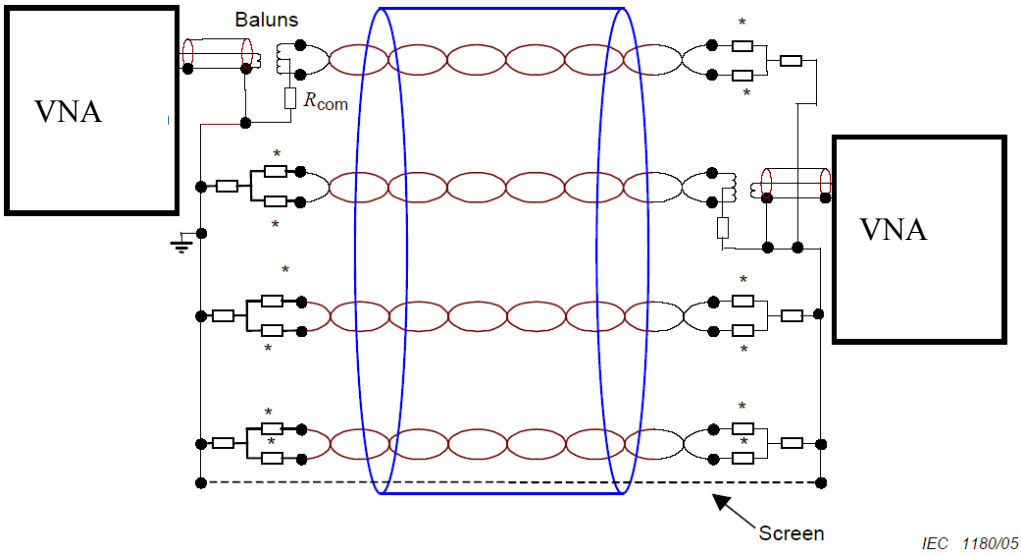


Figure 6: Far-end Crosstalk test configuration according to ICE 61935-1

FEXT is measured by applying the signal to the near end of one wire pair and measuring the coupled signal at the far end of a different wire pair. The coupling between the data and the strobe lines which is of particular interest requires two times two separate measurements.

4 SpaceWire Test Connector Board

In order to be able to perform the tests discussed previously on SpaceWire cables a special test connector board has been designed. It allows to connect the VNA with SMA connectors on one side and to interface the SpaceWire cable via a microminiature D-type socket on the other side. The two different structures described in the following have been integrated on this PCB in order to perform the measurement by using two different measurement techniques. A comparison of the results will be used to get an estimate of the measurement accuracy and to select the best measurement technique for the future. Four identical test connector boards of this type have been built.

5.1 BALUN DRIVEN TEST CONNECTOR

This test connector is in line with the measurement setup described in [2] and [3].

The balun driven connector allows for a conventional 2-port VNA use, as each of the SpaceWire differential pairs are driven from single-ended 50Ω SMA connectors via a 1:2 impedance ratio balun. The centre tap of each of the baluns is terminated for common mode AC signals by a 51Ω resistor and a π–section RC low-pass filter Figure 7.

The baluns used (Mini-Circuits TX-2-5-1) are specified as having a –2dB bandwidth of 30-1100MHz in a 75Ω system and have shown a very similar performance in the 50Ω system used for the tests here.

When used to test the match (return loss) of real SpaceWire receiver in normal operation it is important that the LVDS line receivers are biased to the nominal 1.2V

potential. For this purpose a RC low-pass filter and a jumper link to an on-board regulated supply have been implemented. This is not needed for testing passive devices.

In order to investigate the impact of a DC disconnection of the outer shield of the SpaceWire cable on the crosstalk performance, the chassis support bracket is mounted on a small section of copper plane which is connected to the PCB's ground plane via two capacitors for some AC grounding at all times and by a set of 4 paralleled jumpers which can be used to break up the DC connection.

4.1 DIRECTLY DRIVEN TEST CONNECTOR

The baluns have been introduced to perform the transformation of the 50 Ω single ended interface of a conventional 2-port VNA to the 100 Ω differential line of the twisted pair cable Figure 7. Where a 4-port VNA is available the directly driven test connector can be used with a direct connection from one 50Ω SMA connectors to one pin in the SpaceWire connector. From the measurements not only the differential mode but also the common mode S-parameters can be calculated as well as the signal transitions between common and differential mode. This setup further avoids the maximum frequency limitations on the measurement introduced by the baluns.

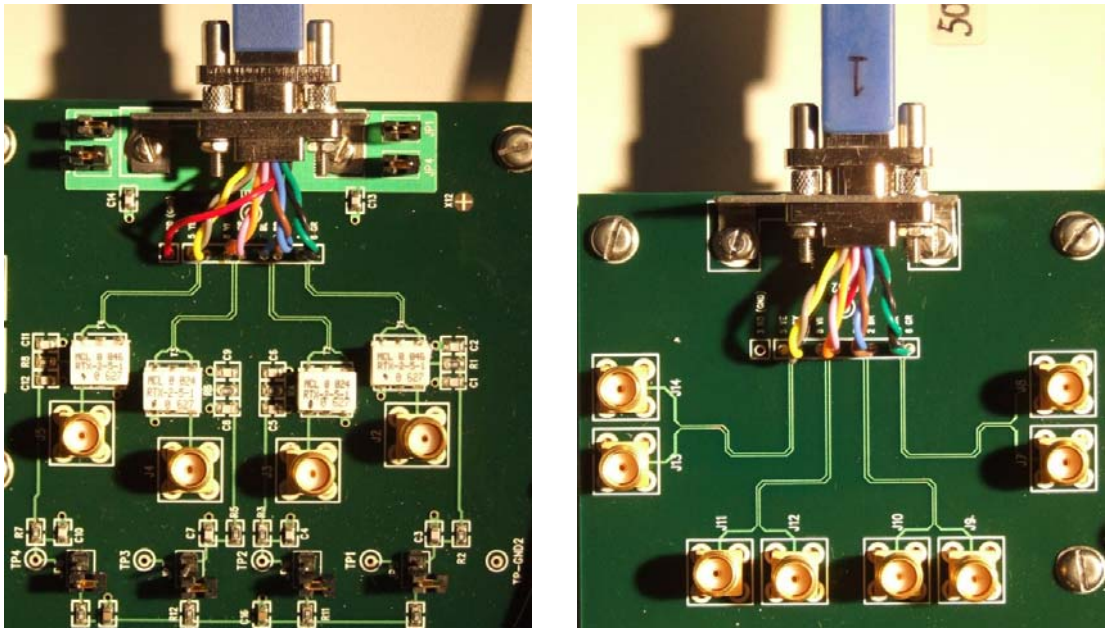


Figure 7: Balun driven test connector (left) and Directly driven test connector (right) as implemented on SpaceWire test connector board

5 CALIBRATION

A good calibration is always a crucial point for an accurate VNA measurement. This calibration has to be performed at the reference plane for measurement which is in this case the SpaceWire connector. For the micro-miniature D-type connector there is no standard calibration kit commercially available but the calibration standards have to be designed and self-built. A number of different calibration techniques like

Short/Open/Load/Through, Trough/Reflect/Line and adaptor removal will be tried out and compared to each other.

The test connector boards further contain some special structures which are intended to be used for calibration and its verification.

6 CONCLUSION

In the current standard the SpaceWire cable is specified by a detailed specification of its construction. Its electrical performance is only partially specified. Particularly no cross-coupling performance and insertion loss performance as a function of frequency is specified. A definition of electrical performance parameters to cover this gap in the specification and a technique to measure them has been described.

A characterisation of the specification-compliant SpaceWire cables with the presented performance parameters will allow it to be compared with other cable designs with respect to the electrical performance.

7 REFERENCES

[1] ECSS-E-50-12A, SpaceWire - Links, nodes, routers and networks

[2] IEC 61935-1, Testing of balanced communication cabling in accordance with ISO IEC 11801 Part1 Installed cabling

[3] IEC 61935-2, Testing of balanced communication cabling in accordance with ISO IEC 11801 Part2 Patch cords and work area cords

MEASURING TIME AND TIME-RELATED ASPECTS OF SPACEWIRE

Session: Test and Verification

Short Paper

Dr Barry M Cook, C Paul H Walker

4Links Limited, Bletchley Park, Milton Keynes MK3 6ZP, England

E-mail: [Barry, Paul] @4Links.co.uk

ABSTRACT

This paper presents a variety of measurement techniques that give information about how a device or system behaves with respect to time. We show how these techniques can be used to measure different parameters of SpaceWire, such as the receive link speed, the times of arrival of Time Codes, the durations and latencies of packets, and the operating margins of the SpaceWire receiver. One such operating margin is the disconnection timeout, and we report on the discovery of non-compliance with the ECSS SpaceWire standard on this parameter, and of other design errors that can be exposed by the measurement of time-related parameters.

We comment that asynchronous design can be difficult, that as a result its testing should be particularly rigorous, and that a synchronous SpaceWire receiver overcomes some of these issues.

We then show how the basic timing measurement techniques can be expanded to cover a potentially large SpaceWire system, and how the time-related issues of interface bandwidth and latency can be overcome to give tens of Gb/s overall SpaceWire test bandwidth.

1 INTRODUCTION

Users of our SpaceWire [1] test equipment have told us for many years how important time is in the design of their systems.

In response to these user needs we have developed a variety of techniques for measuring different time-related characteristics. We reported on some of this work at DASIA 2006 [2], and in this paper concentrate more on the how the measurements can be done and what they can be used for. Much of this is in helping users to remove bugs, but we have also found that our equipment exposes bugs that would not be seen without the measurement of time-related parameters.

We also show how the capabilities have been developed to cover systems from a single chip up to a large system with many ports and tens of Gb/s SpaceWire throughput.

The bugs that we have seen, including those in supposedly mature designs, lead us to sound a note of caution about asynchronous design and the traps that it can cause for the unwary.

2 MEASUREMENT TECHNIQUES

We present a variety of measurement techniques. All are simple in concept, but achieving the resolution, sensitivity and synchronization that users require is far from trivial.

Frequency: This is simply the inverse of time, and may be measured by timing a number of cycles of the signal that is being measured.

Time tagging: The occurrence of an event is sampled at high frequency, so that the time at which the event occurs can be recorded. The recording resolution on early products was 100ns, and this has been progressively improved to less than 1.5ns.

Fine control of transmit speed: Transmit speed can be set in small increments. The increments are 0.1Mb/s steps (or smaller) up to 40Mb/s, and 1Mb/s steps up to 400Mb/s.

Increased bit periods: Unlike most communication standards, a bit on SpaceWire can have any duration, and successive bits do not need to have the same duration. A particular bit duration may therefore be extended by a certain number of clock cycles.

Synchronization between outputs: A set of outputs can be delayed, without transitions, until all are ready to transmit, much as a barrier prevents horses from leaving the starting line in a race. When the barrier is lifted, all the packets leave at the same time, or with accurately pre-determined delays. Within a single EGSE box, this synchronization can typically be sub-nanosecond.

Synchronization between units: Some SpaceWire networks are being built with many SpaceWire nodes, and to test these, it is useful to have a common view of time across all the test equipment being used. Synchronization is achieved for time tags to ±3ns, over any number of boxes up to a total synchronizing cable length of 50metres.

3 MEASURING THE BIT RATES

It can be useful to be able to see not only that a link has connected, but also to see the transmit and receive bit rates. If the bit-rate is not as expected (whether as a result of a hardware or software failure or because the user has set it up incorrectly), it is not always obvious that this is the cause of unexpected behaviour. Displaying both the transmit and receive speeds on the front panel and making them available to software, for example to show on a GUI, makes such errors more readily visible.

The measurement is also useful for validation, as the ECSS SpaceWire standard requires that a link starts up between a rather narrow range of bit-rates and we have observed designs that start up at a transmit bit-rate outside this range.

4 MEASURING TIME CODES

SpaceWire uses Time Codes to distribute a common view of time throughout the spacecraft. Test equipment needs to ensure that the distributed time codes arrive correctly. Time tagging their arrival and logging the results makes it possible to see if

any time codes get lost, to see what the time code interval is, and to see what long-term drift and short-term jitter there is in the time code source and distribution.

5 MEASURING PACKET DURATION

There are many places, both in an individual SpaceWire node and across a SpaceWire network, where packets can be delayed and so have Nulls inserted where they are not expected. Time tagging the start and end of packet to record the packet duration enables the user to determine whether such Nulls have been inserted.

6 MEASURING LATENCY OR DELAY

In a similar way to measuring packet duration, packet delay can be measured with time tags. In this case the transmitted packet is time tagged as well as the received packet. The measurement is so sensitive that it can easily discriminate differences in cable length, but is equally capable of measuring routing-switch or host-interface latency even if these are milliseconds or seconds. Indeed the measurements can be used at both ends of an Internet connection to measure local latency and remote latency, the difference being the round-trip-time over the Internet.

7 MEASURING OPERATING MARGINS

Many SpaceWire circuits operate at a limited number of transmit speeds. For example a design might run at 200Mb/s, 100Mb/s, 50Mb/s and 10Mb/s. The receiver may have been designed to operate at up to 220Mbits/s, but the circuit itself provides no means of testing such operation. Test equipment with the finely tuned transmit frequency can determine whether the design does indeed operate up to 220Mbits/s or whether it fails at a much lower speed and would therefore have inadequate design margin.

As well as being used to qualify completed designs, users have found this test capability extremely useful in development. One user had two equipments that worked individually but not together. It transpired that both had narrower than intended operating margins --- one only worked below a certain operating point, the other only worked above a certain operating point and there was no overlap between the two operating regions. Using the finely tuned transmit frequency, they were able to expand the operating margins so that they overlapped and the two equipments were able to work together.

8 MEASURING THE EFFECT OF HIGH-FREQUENCY NOISE

Asynchronous input circuits for SpaceWire can be susceptible to high-frequency noise. If the noise results in consecutive bits having less than the minimum edge separation that the receiver can tolerate, the regenerated clock may have inadequate pulse width. Behaviour in such circumstances is indeterminate and the receive logic may enter an invalid state. We have found several designs where the receiver gets into a state that can not be recovered by the normal disconnect and exchange of silence — the only way to recover was to reset the whole device.

Testing at deliberately higher speeds than the device can tolerate may seem to be excessive, but the high-speed testing is only simulating noise which may occur in flight as a result of radiation effects.

We would recommend:

- that this test should be performed on all asynchronous SpaceWire receivers;
- and that if asynchronous designs are used, then a mechanism is invoked that is able to reset the device if it gets into an invalid or otherwise irrecoverable state.

9 MEASURING DISCONNECTION TIMEOUT

The timing measurement techniques can be used to create a precise gap between transitions that is much longer than the normal bit-period, For example the gap can be set to 700ns, which should never result in a disconnection timeout, or to 1000ns, which should always result in a disconnection timeout. And of course a scan can be performed that tests at a range of gaps between edges, from less than 700ns to more than 1000ns. This can be done when the link is idle, just transmitting Nulls, or when it is transmitting data.

With one SpaceWire design, we found that a gap of 840ns never generated a disconnect timeout, and that a gap of 860ns always generated a timeout. With another SpaceWire design, we found that when receiving Nulls, it always generated a timeout at 1000ns, but when receiving data, a significant proportion of gaps above 1000ns failed to generate the timeout. This is not the sort of bug that would jeopardize a mission, but it is clearly not compliant with the ECSS SpaceWire standard.

10 TESTING ASYNCHRONOUS ARBITRATION

Synchronous arbiters are comparatively easy to test, and indeed if a simulation reports functional correctness and timing analysis reports that the required speed is met, confirmatory testing is possibly adequate.

The situation is very different in the case of asynchronous arbitration. Firstly, very few of the simulation tools adequately handle asynchronous circuits. And secondly, synchronous timing analysis is irrelevant when all the signals are asynchronous.

We therefore provide synchronization on packet outputs, with controllable offsets of one or more outputs with respect to the others. The synchronization is remarkable, typically below our capability of measuring it and definitely well under one nanosecond of skew between outputs. This is as close as possible to what is needed to test an asynchronous arbiter, for correct functionality let alone for performance.

One of the bugs found with this technique was a routing switch with an arbiter that was not ‘fair’, so it could allow some ports to block other ports permanently.

11 GENERAL COMMENTS ON ASYNCHRONOUS DESIGN

The last few sections have highlighted problems with different aspects of asynchronous design. While SpaceWire’s asynchronous nature has many advantages, it presents enticing traps for the unwary. The standard design tools are excellent for synchronous design but — except in expert hands — are almost non-existent for asynchronous design. Consequently those designers of SpaceWire who have chosen the asynchronous approach have given themselves an extremely challenging job.

Several years ago, 4Links abandoned asynchronous design and our receivers have a free-running clock that samples the asynchronous inputs at the device pins. Thereafter, the whole receiver is synchronous, and can be handled by the excellent design tools that are available. The result is a much more robust SpaceWire core than is possible with asynchronous design.

12 MEASURING TIME ACROSS AN ENTIRE SYSTEM

The above measurements have proved to be invaluable both for detecting and for guidance in fixing bugs. But as described so far, they are only available from a single test unit, which in our product family provides up to eight SpaceWire ports. SpaceWire systems are often smaller than this, but there are several spacecraft in design that have many more than eight SpaceWire ports. So it would be useful for time to be accurately coordinated across the system.

Synchronization has been achieved with a coax bus connecting all the SpaceWire test sets. The bus carries not only the synchronization clock, but also all the management of which unit is synchronization master (to which all the other units synchronize) together with calibration of the delay to each unit. The result is synchronization for time tags across any number of boxes up to a total coax length of 50metres to within ±3ns. Synchronized outputs are not quite so precise at ±20ns, but they can be time-tagged with the ±3ns precision.

Putting this into perspective, a time-tagged packet output on one unit can be received by a unit 50 metres away, with the synchronizing cable visiting ten or more other units, and with the ambient temperature changing and different at the two units — and the transit time reported by comparing time tags from the sending and receiving units is accurate to within less time than it takes for a signal to go along about half a metre of SpaceWire cable.

13 OVERCOMING BANDWIDTH LIMITATIONS

If both directions of a SpaceWire link are running at 400Mb/s, through eight ports per unit, the total bandwidth of actual data can be up to 5.12Gb/s. Add instrumentation, for example with time tags, and total bandwidth can increase to above 6Gb/s on a single unit.

To provide such bandwidth, we have extended our test equipment to include a Packet Generator for each port, which can be programmed to generate a long sequence of different packets with little or no interaction across the interface to a control computer. The computer just sets up the algorithm to generate the sequence of packets.

In the receive direction, the equipment has a packet checker that can use, for example, such information as the header and checksum to check that the received packet is correct. If it is correct, no (or negligible) information needs to be sent to the computer, and so bandwidth is conserved in this direction as well. In fact the data-rate multiplication in the generator and data-rate compression in the checker are such that full-bandwidth tests are conceivable over the Internet.

The packet generator and checker are actually implemented by small, highly specialized, processors — specialized for the jobs of generating and checking packets, with remarkable flexibility in so doing.

When the packet generator is combined with multiple units and their synchronized time-tags, systems can be tested with tens of Gb/s of SpaceWire traffic.

14 OVERCOMING LATENCY LIMITATIONS

Much work goes into overcoming bandwidth limitation but, in many cases, the actual bandwidth achieved is determined more by latency than by the native bandwidth limitations. This is particularly the case when a PC (with its operating system) has to respond to a request. The whole process can take many milliseconds.

Having a programmable packet generator and checker on each port, with the generator and checker able to transfer information between each other, provides an exceptionally low-latency response. The checker not only checks the packets it receives but extracts the information needed by the generator to return a packet with the right size and content to the correct place in the network. The response may be, for example, to a read request, or it might be a simple acknowledgement of the received packet.

As with the generator and checker working in isolation but over multiple units, responsive systems can be tested with tens of Gb/s of SpaceWire traffic.

15 CONCLUSIONS

Time is an important parameter for SpaceWire, and users need to know how their systems behave with respect to time. This includes being able to measure and record parameters such as the receive link speed, the times of arrival of Time Codes (and the variations in those arrival times), the durations and latencies of packets, and the operating margins of the SpaceWire receiver.

The asynchronous nature of SpaceWire, while offering many advantages, also offers enticing traps for the unwary. These traps can result in failure to comply with the ECSS SpaceWire standard, or failure in operation, for example as a result of a noise upset.

The tests that we have performed on a wide range of SpaceWire designs have exposed a variety of bugs. We observe that, even in the hands of experts, common design and simulation tools lead to a false sense of security and lead us to conclude that compliance with the standard can only be assured if a design has been subjected to the timing tests that we have outlined here.

If subjecting a SpaceWire design to noise causes a lock-up condition that can not be recovered by the standard disconnection and exchange of silence, we have to question whether the design is fit for purpose.

At least some of the problems we have uncovered are because asynchronous design is difficult. We and our users have found our own synchronous SpaceWire design to be robust.

If time is important over a single SpaceWire link, it is equally important across a large system with 100 or more SpaceWire nodes. Test equipment is now available that

provides a consistent view of time from small systems to large systems, and to within a few nanoseconds.

Any interface to a large computer and operating system is likely to limit bandwidth and give excessive latency, distorting system behaviour with respect to time. Adding specialized packet processors at each port of the test equipment overcomes such limitations for many aspects of system test, and scales to tens of Gbits/s of SpaceWire traffic.

REFERENCES

[1] The ECSS-E-50-12 Working Group, "**ECSS-E-50-12A 24 January 2003**, SpaceWire - Links, nodes, routers and networks", published by the ECSS Secretariat, ESA-ESTEC, Requirements & Standards Division, Noordwijk, The Netherlands

[2] Barry M Cook, Paul Walker, "**Verifying the Temporal Behaviour of SpaceWire Components and Systems**", Data Systems In Aerospace (DASIA), Berlin, 22 to 25 May 2006, proceedings CD SP-630 – July 2006, ISBN 92-9092-941-3, ESA.

Test & Verification 2

Wednesday 19th September

11:15 – 12:45

DEBUGGING SPACEWIRE DEVICES USING THE CONFORMANCE TESTER

Session: Test & Verification

Short Paper

Dr Steve Parkes, Dr Martin Dunstan

School of Computing, University of Dundee, DD1 4HN, Scotland, UK

E-mail: sparkes@computing.dundee.ac.uk, mdunstan@computing.dundee.ac.uk

1 INTRODUCTION

The SpaceWire Conformance Tester is a device developed by STAR-Dundee for ESA to perform black-box testing of SpaceWire devices against the SpaceWire ECSS-E-50-12A standard [1]. The tests performed by the Conformance Tester are those which can be performed over a single SpaceWire link to the unit under test without cooperation from that unit. Thus link initialization behaviour and the response to data and control characters can be investigated while PCB layout and connector compliance cannot. The tests to be performed are launched from easy-to-use software running on the host PC with concise test results providing important feedback on how the unit under test performed.

This paper examines two of the errors/issues that have been identified with different SpaceWire devices through the use of the SpaceWire Conformance Tester.

This paper also introduces a novel test procedure which measures the speed at which the unit under test recovers from link errors at different points of the link initialisation process. The graphical output of this test can be used to identify anomalous behaviour of a SpaceWire device which might be hard to detect in other ways and to provide measurements of the link initialisation timing parameters.

2 MEASURING LINK START-UP SPEED

Clause 6.6.5 of the SpaceWire ECSS-E-50-12A standard specifies that after a reset or disconnect the SpaceWire link transmitter shall initially commence operating at a data signalling rate of (10±1) Mb/s and shall operate at this rate until commanded to operate a different data signalling rate. Clause 6.6.6 specifies that the operating rate shall not be changed before the link has reached the *Run* state.

To validate the (10±1) Mb/s link initialisation speed the test link interface (TLI) of the Conformance Tester is instructed to restart the link to the UUT but is forbidden to transmit any FCTs. A compliant UUT will begin error recovery and link initialisation eventually reaching the *Connecting* state. Since the TLI is forbidden to send FCTs, the UUT must return to *ErrorReset* after the nominal 12.8 µs time out period. From the time at which the UUT enters the *Started* state and begins transmitting NULLs to the time at which it stops the link to move to *ErrorReset* the UUT must operate at a data signalling rate of (10±1) Mb/s. It ought to be a trivial matter to measure the link rate of the UUT and confirm that it lies within the specified limits.

However, the standard doesn't specify the conditions under which the data rate limits are to be satisfied. Should the duration of every bit transmitted during start-up lie within 90.9 ns to 111.1 ns? Should the duration of every recovered transmit clock period lie within 181.8 ns to 222.2 ns? Should the signalling rate be averaged over multiple bits: how many bits? When does link initialisation end and shutdown begin?

The Conformance Tester software will report the minimum and maximum bit-to-bit rates, the minimum and maximum clock period from rising-edge to rising edge and from falling-edge to falling-edge, and the bit rate averaged over all bits received. If any bit-to-bit rate lies outside the (10±1) Mb/s range the UUT is considered to have failed. It is the responsibility of the user to examine the reported limits to determine whether the test failure corresponds to a violation of the standard. By applying a stringent interpretation of the standard we encourage the user to investigate the issue.

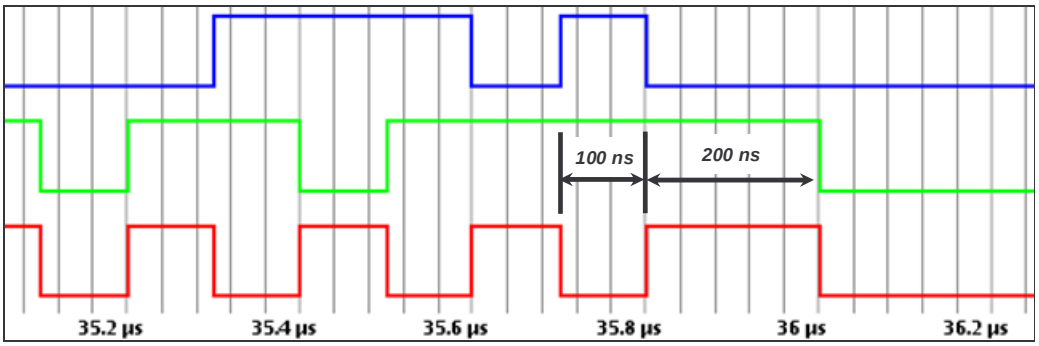


Figure 1: link shutdown with the strobe signal held for an extra bit period

An example of a test failure which has been observed with several UUTs is shown in Figure 1. The UUT has held the strobe signal stable for an extra bit period while stopping the data signal during link shutdown to avoid a simultaneous D/S transition. This doubles the duration of the last bit which causes a violation of the bit-to-bit rate. Since the failure applies to the last bit of the transmission during link shutdown one could argue that the (10±1) Mb/s rate does not apply. However, by reporting a test failure the user is made aware of an issue that not be detected otherwise.

3 Empty Packet Credit Counting Failures

Clause 8.3 of the standard specifies that the transmitter shall not transmit any N-Chars unless its credit counter is greater than zero. The transmit credit counter shall be decremented by one for each N-Char transmitted and shall be incremented by eight for each FCT received. A similar counter is maintained by the receiver. Clause 8.2.1 states that data characters as well as EOP and EEP are N-Chars. Clause 8.9.3.2 states that empty packets may be silently discarded.

As a result of these clauses being considered in isolation, the Conformance Tester has discovered UUTs which discarded empty packets before updating their receive credit counter. If a UUT suffers from this problem then empty packets received by the UUT will reduce the effective size of the UUT receive buffer. By sending a non-empty test packet to the UUT followed by a number of empty packets the Conformance Tester is able to detect this bug. If the bug is present the Conformance Tester can also report the size of the UUT receive buffer.

4 NULL ARRIVAL TIME TESTING

The NULL arrival time test is a recent addition to the Conformance Tester suite and is one of the most powerful for probing the UUT.

Consider the following: the Conformance Tester TLI is connected to a UUT with the link in the *Run* state. At time T_0 the TLI sends a parity error to the UUT. A compliant UUT will respond by moving to *ErrorReset* and begin the link initialisation process. Let the TLI ignore the UUT disconnection and remain in the *Run* state transmitting NULLs indefinitely at the maximum achievable rate. A compliant UUT will detect the TLI NULLs when it reaches the *ErrorWait* state. The UUT will reach the *Started* state at approximately time $T_{null}=T_0+19.2\ \mu s$ where it will begin transmitting NULLs before moving to the *Connecting* state. We define $T_{null}-T_0$ as the NULL arrival time.

Now consider what happens if the TLI sends another error at time $T_1>T_0$. If the UUT is in the *ErrorReset* state it will be unable to detect this error because its receiver is disabled. Thus errors received while the UUT is in *ErrorReset* will have no effect on the expected NULL arrival time of 19.2 µs. In contrast, if the UUT is in any other state when the error arrives then the UUT must return to *ErrorReset* and begin the link initialisation process again. This will add T_1-T_0 to the expected NULL arrival time. An example graph of the NULL arrival time against T_1-T_0 is shown Figure 2.

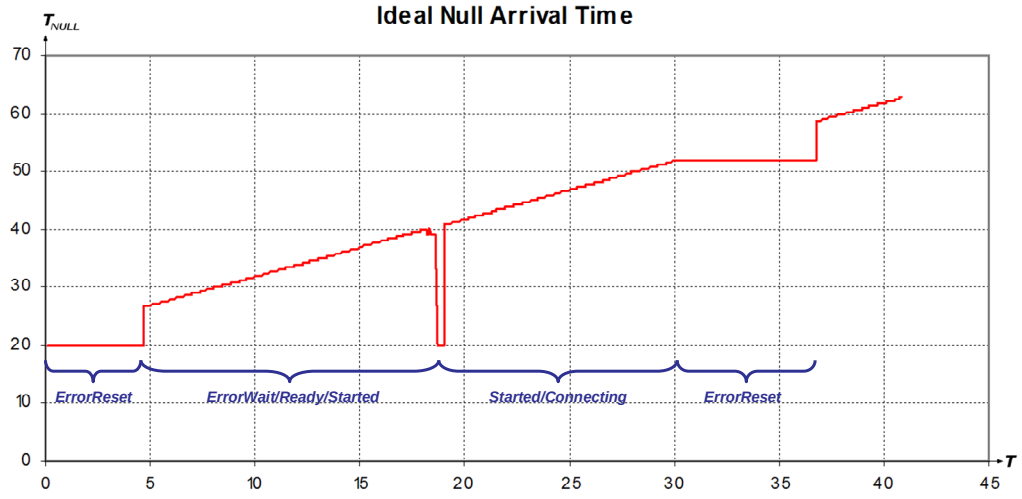


Figure 2: expected NULL arrival time graph

The horizontal segment of the graph from T=0 µs to T=4.64 µs corresponds to the period when the UUT is in the *ErrorReset* state: TLI errors are ignored and have no effect on the 19.8 µs NULL arrival time. The vertical step at T=4.64 µs marks the point where the UUT enters *ErrorWait* and the height of the step corresponds to the time the UUT spends in *ErrorReset* (approximately 6.4 µs). Note that the horizontal position of the vertical step is not 6.4 µs because the TLI waits until the UUT has disconnected before it starts the NULL arrival timer.

The diagonal section of the graph from T=4.64 µs to T=18.64 µs corresponds to the *ErrorWait*, *Ready* and *Started* states: if T_1 is increased by Δ then the NULL arrival time is expected to increase by Δ also producing a diagonal line. When the UUT is in the *Started* state its start-up NULLs will be detected by the TLI before the TLI is

ready to issue the second error. In this situation the TLI waits until the UUT has disconnected after issuing the error at T_1 before enabling the start-up NULL detector. The dip in the middle of the graph corresponds to the period when the UUT has started sending NULLs in the *Started* state but the TLI has not detected them yet. The TLI sends the error at time T_1 and detects these NULLs shortly afterwards.

The end of the diagonal line at T=30.32 µs indicates where the UUT leaves the *Connecting* state because it hasn't received any FCTs from the TLI. The graph repeats at this point with another horizontal segment corresponding to *ErrorReset*.

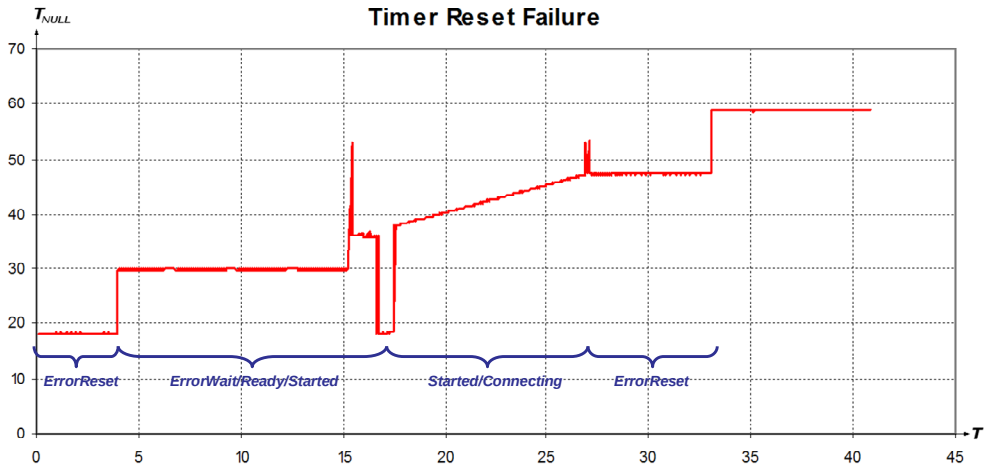


Figure 3: failure to reset 6.4 µs /12.8 µs timers

An example of NULL arrival times from a faulty device is shown in Figure 3. By comparing this graph with that shown in Figure 2 it can be seen that there is a problem with the *ErrorWait* state. It seems that the *ErrorWait* state is ignoring errors just like the *ErrorReset* state. However, since there is a step at T=3.96 µs the UUT appears to be detecting errors after leaving *ErrorReset*. This problem was due to a failure to reset a shared 6.4 µs/12.8 µs timer when moving from *ErrorWait* to *ErrorReset*. This caused the second *ErrorReset* period to be between 0 µs and 12.8 µs instead of 6.4 µs.

5 CONCLUSIONS

The Conformance Tester is very effective at detecting bugs in SpaceWire devices that were not uncovered by the development test suites of the devices. The development of the Conformance Tester has also raised questions about how parts of the SpaceWire standard ought to be interpreted such as the (10±1) Mb/s link initialisation rate measurement. Finally, the novel NULL arrival test described here can be used to probe the SpaceWire link initialisation state machine. It enables the 6.4 µs *ErrorReset* duration of any UUT to be measured and verifies the correct response to errors in the different link initialisation states.

6 REFERENCES

[1] "SpaceWire - Links, nodes, routers and networks", ECSS-E-50-12A, February 2003.

INTEGRATED DEVELOPMENT TOOLS SUITE FOR THE SPACEWIRE RTC ASIC

Session: SpaceWire Test and Verification

Short Paper

Walter Errico (1), Jørgen Ilstad (2), Annamaria Colonna (1), Fabrizio Bertuccelli (1)

(1) Aurelia Microelettronica Srl, Via Vetraia 11, I-55049 Viareggio, Italy

(2) ESA-ETEC, Keplerlaan 1, NL-2201 AZ Noordwijk, The Netherland

E-mail: walter.errico@aurelia-microelettronica.it, Jorgen.Ilstad@esa.int

ABSTRACT

The SpaceWire RTC device is a fully integrated LEON2-FT based System on Chip that among other features, provides the capability to bridge traffic between the SpW network and the CAN bus. The SpW RTC Development Suite is the collection of HW/SW components designed to stimulate all the RTC interfaces and to drive all its communication links.

In this paper a brief description of the RTC ASIC and the Development Suite Tool are presented before focusing on the SpaceWire aspects of the two devices and their test chain.

1 SPACEWIRE RTC

The SpaceWire Remote Terminal Controller (RTC) device is a system on chip for space application, including the LEON2-FT processor unit with on chip memory, a memory controller for external banks, two SpW Links, a CAN bus controller, ADC/DAC interfaces for analogue acquisition/conversion, FIFO controller, standard interfaces and resources (UARTs, timers, general purpose input output).

The embedded LEON2-FT microprocessor, its abundance of interfaces and the possibility to access the system via SpW and CAN makes the RTC suitable for many onboard applications. i.e as an intelligent node due its processing capability, or it can be remotely managed via its SpW link interfaces using the Remote Memory Access Protocol (RMAP)

The SpW-RTC can play the role of payload data processor inside the Instrument Controller Unit (ICU). It can receive payload data from instruments, process them and send down via SpW. Its several programmable interfaces allow RTC to acquire analogue and digital data, generated by peripherals. The on-chip interfaces have DMA capability, thus data transfers between SpW links and i.e FIFO can sustain high bitrates. The CAN bus allows the On-board Computer (OBC) to monitor and control the SpW-RTC device that acts as a remote terminal.

Alternatively, the networking features of the RTC may be exploited even if the device is integrated in the OBC. The CAN controller capability can be utilised for node

management and time distribution, while the SpW links are utilized to communicate and manage the SpW network itself.

2 DEVELOPMENT SUITE

The SpW RTC development suite is composed of two HW modules; PCI-SpW/CAN board, SpW RTC ASIC test bed, and a SW library for generating traffic on all of the SpW RTC communication ports, independently of the LEON2 debugger tool.

The PCI-SpW/CAN is a standard PCI peripheral FPGA based board housing 2 SpW links plus 2 CAN ports and a large parallel service connector. Ones plugged into a Linux PC it is used as a stimuli source for the RTC test bed.

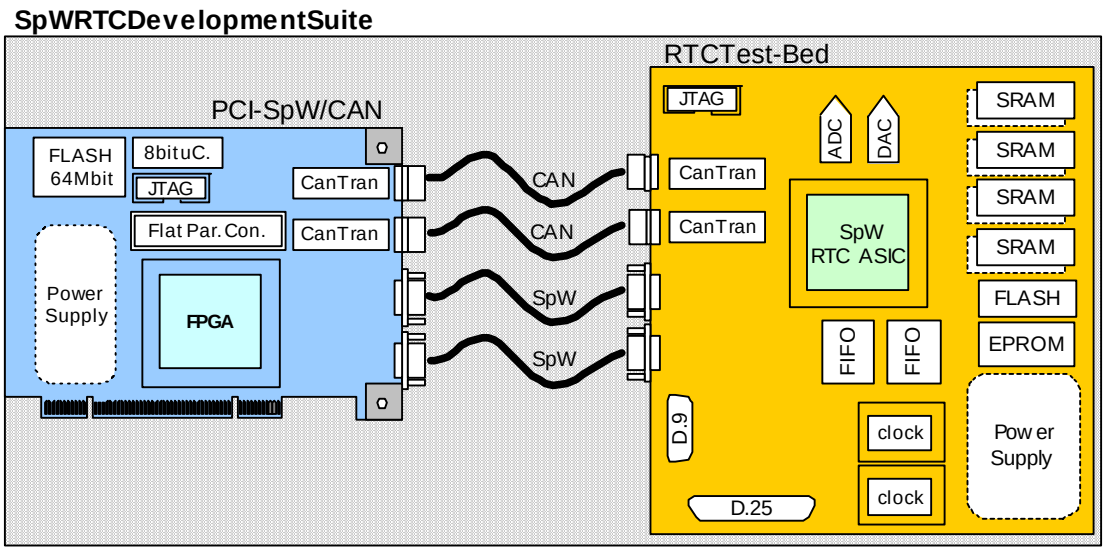


Fig 1 SpW RTC Development Suite HW modules

The "RTC test-bed" is the external test board where the SpaceWire RTC ASIC is mounted in addition to peripheral units such as SRAM, FLASH, EEPROM, ADC-DAC, FIFO etc. The testbed includes possibility to interface custom made daughterboards via mezzanine connectors. Users can take advantage of the FIFO interface or ADC/DAC interfaces for breadboarding of i.e. instrument control modules or custom made FPGA designs.

Communication with the PCI-SpW/CAN board is achieved through the CAN and SpW links. The RTC test-bed is provided with the standard serial (UART) connector to be directly connected to the test PC where also the LEON2 GRMON debugger tool may be independently running.

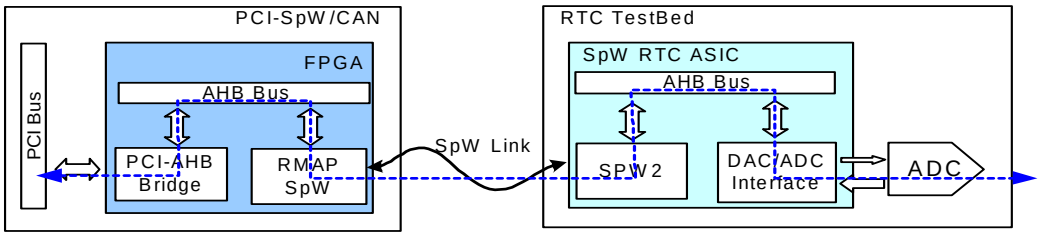


Fig 2 TestBed ADC measurement controlled via SpW

The development suite SW library incorporates drivers and comm. functions to perform data exchange between the PCI-SpW/CAN and the testbed via SpW and CAN links. The comm. functions for the SpW links utilize RMAP which enables access to all internal registers and memory space of the SpW RTC ASIC.

Fig 2 shows an example of measurements done on the RTC ADC interface directly transferred via SpW to the test PC.

The division of the development suite in two distinct parts leaves the user with the possibility to combine the test suite with other 3rd party equipment.

In addition several innovative concepts are implemented in the proposed architecture of the PCI-SpW/CAN board:

- *Compactness:* Two CAN Controllers and two SpW ports are hosted on the same single board.
- *Flexibility:* CAN and SpW modules are RTL Core instances embedded in large FPGA. They can be easily modified and/or enhanced to follow any new protocol updates and features.
- *Maintainability:* A custom FPGA programming scheme (Aurelia A-AltPrg1) is implemented on the board to allow the users to update the FPGA image without need of any vendor specific SW/HW tool or licence. The users can load the new programming image directly through the PCI bus.
- *Simplicity:* The software developer is provided with a single API for all the communication links. All the internal buffers and control registers are easily accessible as they are mapped in a uniform memory segment in the PCI space.

3 SPACEWIRE TEST

The development suite structure foreseen for the SpaceWire Test is shown below. The RTC ASIC SPW2 modules are connected to the Aurelia RMAP-SPW interfaces implemented in the FPGA of the PCI-SpW/CAN board. Both types of modules are provided with an AMBA AHB interface.

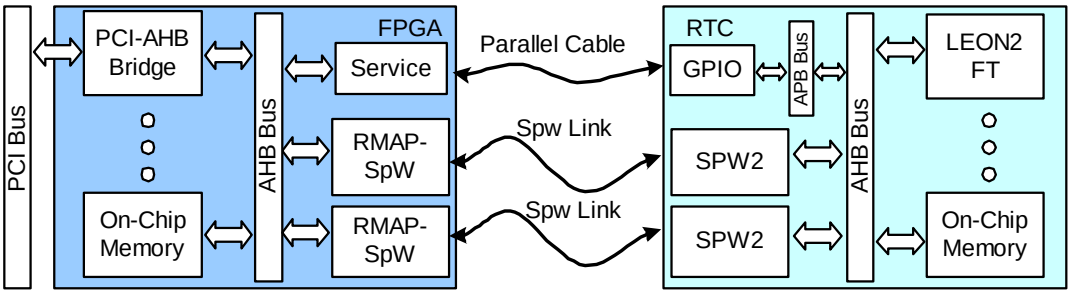


Fig 3 SpaceWire test chain

The SPW2 has access to the full range of all RTC registers and memory space via the AHB bus, thus its DMA engines can perform transmission and receptions of data without involving the LEON2-FT processing unit support during the transfers.

[1] A-AltPRG is the Aurelia integrated solution for Altera FPGA programmability on-board without need of proprietary software and external cables.

Due to the complex memory hierarchy of the Linux operating system the reception of RMAP packets is buffered in the on-chip memory of the FPGA. A DMA controller is included in the PCI-AHB bridge that optimizes large data transfers to and from this buffer.

4 SPACEWIRE RTC SPW2 MODULE

The RTC ASIC is equipped with two instances of the SPW2 Interface module.
The SPW2 module supports RMAP (Remote Memory Access Protocol) commands and the VCTP (SpW Virtual Channel Transfer Protocol) encapsulating the University of Dundee SpW CODEC for the low layers SpW link protocol. High-level non-supported commands and protocols are redirected to the software.

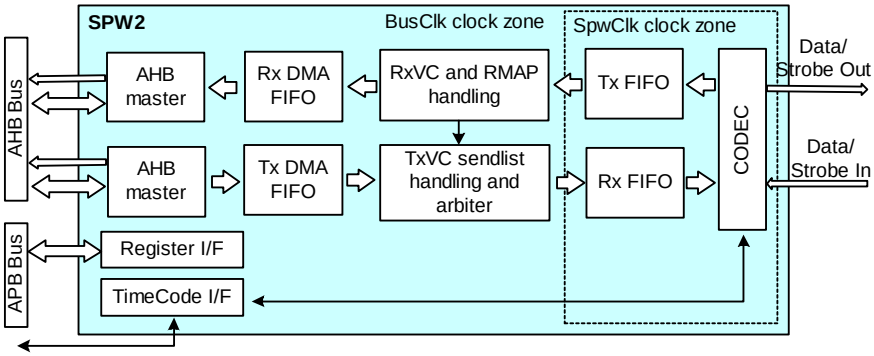


Fig 4 SpaceWire RTC SPW2 module

The RMAP allows remote users to access the entire RTC memory space using the explicit AMBA address to access also the system control and configuration registers. The RMAP is robust in that it offers CRC protection of the cargo and destination key check. It effectively allows for system control via SpW.

The VCT protocol utilises a pre defined memory area within the RTC memory space, defined using configuration registers. This effectively establishes a virtual channel which is well adapted to transfer larger data payloads.

5 DEVELOPMENT SUITE RMAP-SPW CORE

The RMAP-SpW core internal structure is drawn in Fig 5. The module is composed of a SpW link interface, a RMAP receiver and two independent AMBA AHB Master and Slave interfaces.

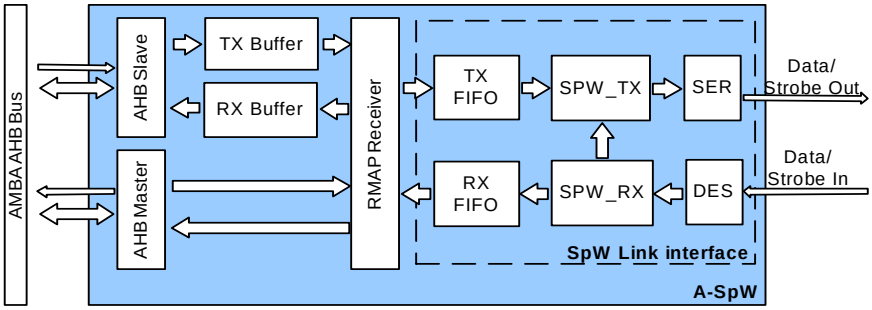


Fig 5 RMAP-SpW core block diagram

The RMAP receiver filters all the incoming packets. It recognizes the remote memory access requests and transfers them on the AMBA bus through the AHB Master interface. Then it automatically sends back the required acknowledges.

The incoming non-RMAP (or RMAP response) packets are transparently written in the RX Buffer, and outgoing data are fetched from the TX Buffer and forwarded toward the SpW link without any filtering. TX and RX buffers are accessed from the AMBA bus through the AHB Slave interface.

The TX Buffer is designed for efficient transmission of outgoing packets without any limitation on format and length. A 1024 x 32 transmission buffer is managed as an outgoing FIFO connected to the physical link like a free running output pipe. Specific locations are dedicated to transmit non d-word (32 bit) aligned bytes and End Of Packets flags. The RX Buffer is capable of simultaneously manage up to 8 packets. A 1024 x 32 memory buffer temporary stores input data in an “almost-FIFO” structure. The data inside the buffers are not automatically removed after reading, but each application has to explicitly notify the number of data and packets that have been consumed; afterward logic shift take place in the buffer so that the offset of the first valid entry is always 0x000. The packets delimiter pointers are automatically updated.

6 CONCLUSIONS.

The ever-increasing focus on reducing cost for space missions by reducing development time and time-to-test puts the SpW-RTC ASIC together with the development suite in the spotlight. The device is well suited for a number of applications, both at platform and payload level. The development suite toolbox allows users to start breadboarding activities at an early stage as both S/W and H/W tools included in the suite contributes to rapid prototyping.

The new SpaceWire interface (SPW2) integrated in RTC devices includes the Star Dundee CODEC core connected with the HW implementation of the two RMAP and VCTP high-level protocols. Both protocols permit the direct connection (DMA) between the system memory and the SpW link relieving the processing unit from the transfer data flow management. In addition, the RMAP protocol can be utilized also for the system remote control via SpaceWire.

On the Development suite side the RMAP-SpW core embedded in a fast FPGA is used. The RMAP-SpW interface operates up to 400 Mbit/sec and includes a RMAP receiver capable of accessing all the FPGA AMBA memory space. A PCI-AHB bridge makes the FPGA AMBA space available on the host Linux PCI space and includes a DMA controller for fast memory transfers. The task to fully utilize the new protocols features of the SPW2 is assigned to the Suite SW library that will be running on the Linux test PC.

The SpW implementations of the device under test and the tester tool come from separated sources and for several aspects follow different approaches. This matter increases the validity of the test activity reducing the risk that common problems mask each other.

SPACE WIRE PROTOCOL ANALYSER ON SPACE CUBE®

Session: Test & Verification

Short Paper

Hiroki Hihara

NEC TOSHIBA Space Systems, Ltd., 10, Nisshin-cho 1-chome, Fuchu, Tokyo, Japan

Shuichi Moriyama, and Toru Tamura

NEC Soft, Ltd., 2-22 Wakaba-cho, Kashiwazaki, Niigata, Japan

Takayuki Tohma and Kenji Kitade

NEC Corp., 10, Nisshin-cho 1-chome, Fuchu, Tokyo, Japan

Steve Parkes, and Stuart Mills

Univ. of Dundee/Star-DundeeLtd., Dundee DD1 4HN, Scotland UK

Masaharu Nomachi

Laboratory of Nuclear Studies, Graduate School of Science, Osaka University,

1-1 Machikaneyama, Toyonaka, Osaka 560-0043

Tadayuki Takahashi, and Takeshi Takashima

Department of High Energy Astrophysics, Institute of Space and Astronautical Science (ISAS), Japan Aerospace Exploration Agency (JAXA), 3-1-1 Yoshinodai, Sagamihara,Kanagawa 229-8510, Japan

E-mail: hihara.hiroki@ntspace.jp, moriyama@mxp.nes.nec.co.jp, tamura@mxm.nes.nec.co.jp, t-tohma@bx.jp.nec.com, k-kitade@cq.jp.nec.com, sparkes@computing.dundee.ac.uk, smills@computing.dundee.ac.uk, nomachi@lns.sci.osaka-u.ac.jp, takahasi@astro.isas.jaxa.jp, ttakeshi@stp.isas.jaxa.jp

ABSTRACT

Protocol analyser for SpaceWire with RMAP (Remote Memory Access Protocol) has been developed for heterogeneous computer platforms. SpaceWire CUBA software (Space Cube Analysis Software) is a portable protocol analyser supporting RMAP, which is developed in collaboration among University of Dundee (UoD), NEC TOSHIBA Space Systems (NTSpace), Osaka University and ISAS/JAXA (Institute of Space and Astronautical Science / Japan Aerospace Exploration Agency).

SpaceWire CUBA software is now used for the development of routing devices for the integrated onboard computer of MMO (Mercury Magnetospheric Orbiter) of Bepicolombo project, which is the joint collaboration mission between JAXA and ESA, and the software supports participants to establish interoperability among SpaceWire community in Japan and Europe.

1 USER NEEDS FOR RMAP IN JAPANESE SATELLITES MARKET

Since RMAP was proposed on the basis of SpaceWire basic protocol, it has gained powerful access method between satellite system management units (SMUs) and payloads. The control software on SMU accesses resources inside payload components through system-wide linear address space like I/O mapped registers.

Japanese scientific satellites have had almost the same capability on their original access module, which is called PIM (Peripheral Interface Module), and the function has been proven to be easy-to-use and flexible for many years through various scientific achievements using satellites. Therefore we have found RMAP as suitable in order to standardize Japanese satellite onboard embedded networks for applying next generation high-speed transmission capability.

2 PROTOCOL ANALYZER FOR RMAP

SpaceWire specification is simple and clear, so we had succeeded in developing SpaceWire interface module by referring written specification and it passed compatibility trial between European SpaceWire community members in 2004 [1]. On the other hand, close investigation should be carried out for various operation cases in dynamic conditions including off-nominal state through space craft development process, and we need protocol analyzer for that purpose. Since RMAP was in reviewing phase, we had joint collaboration among UoD, ISAS/JAXA, Osaka Univ., and NTSpace in order to clarify the understandings acquired from proposed specification.

2.1 DEVELOPMENT AIMS OF THE PROTOCOL ANALYZER

The protocol analyzer software should be portable, because hardware used for the equipments are mostly commercial products and have short product life. In addition to that, Space Cube is used in Japanese SpaceWire community as a platform for space and high energy physics research and development activities [2], so we need to use the protocol analyser on Space Cube as well as the popular test equipment as STAR-Dundee SpaceWire-USB Brick. Figure 1 shows Space Cube.

Fig. 1. Space Cube®

Therefore, the aims of the analyser are:

a) To design and develop software to support the development and testing of SpaceWire units that can run on both the STAR-Dundee SpaceWire-USB Brick and Space Cube.

b) To support the SpaceWire RMAP protocol with this tool.

c) To define a suitable driver API (Application Programming Interface) based on the API used for the SpaceWire-USB Brick.

The software is based on UoD PETRI (The Powerful Easy-to-Use Transmit Receive Interface) software, and versatile API is developed in order to make the software

portable for various operating systems. The API is developed on Windows operating system for UoD USB-bricks and T-Kernel real-time operating system running on Space Cube, and the same protocol analyser software is now running on Windows personal computer and Space Cube by exploiting the API.

2.2 THE ARCHITECTURE OF SPACEWIRE CUBA SOFTWARE

The protocol analyzer, which is called SpaceWire CUBA Software, is realized on the common API. The API is developed on Windows operating system and T-Kernel real-time operating system, and almost the same program source code runs on both operating systems. The software architecture on T-Kernel real-time operating system is shown in figure 2. T-Kernel is open-source real-time operating system based on TRON architecture. Boot strap code and hardware interface routines are programmed in T-Monitor. File system, network system, and graphics user interface system are included in standard extension layer, which give complete personal computer capability on palm top size embedded computer. SpaceWire device driver has been developed for user own SpaceWire IP (Intellectual Property), which encapsulates hardware specific interfaces. Since T-Kernel is independent from microprocessor architecture and the API for CUBA software uses T-Kernel features, SpaceWire CUBA Software runs on various microprocessors on which T-Kernel is implemented.

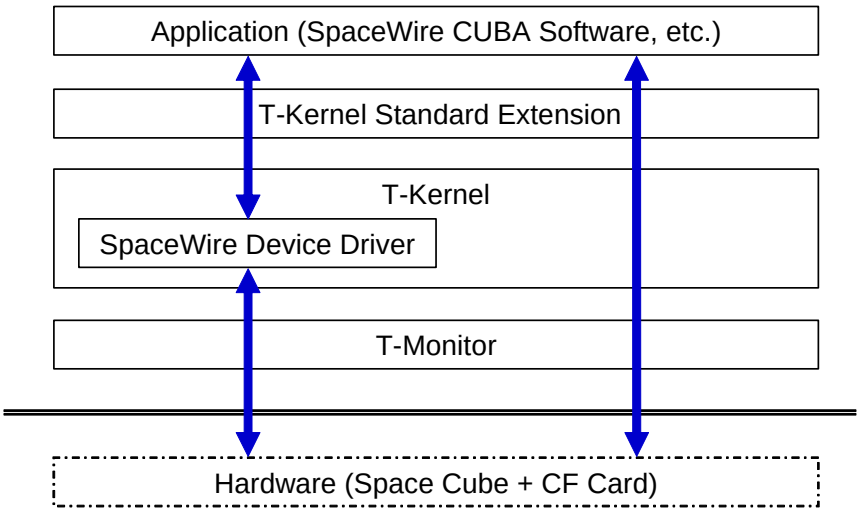


Fig. 2. SpaceWire CUBA Software architecture on T-Kernel

Figure 3 shows the working configuration of SpaceWire CUBA Software on Space Cube with PC terminal.

Fig. 3. SpaceWire CUBA Software on Space Cube

2.3 SATELLITE TESTING WITH SPACEWIRE CUBA SOFTWARE

SpaceWire CUBA Software is developed through joint collaboration among European and Japanese SpaceWire community, so it can be used as a reference through satellite development process especially for joint project like Bepi-Colombo/MMO. Since many members with various point of view participated in the development, detail understanding of SpaceWire and RMAP specification including off-nominal state are clarified through the development of the software and some concerns are reflected on the latest RMAP specification. Consequently, SpaceWire community members have already had benefits acquired though the development activity before using the software. SpaceWire CUBA Software is distributed from STAR-Dundee, Ltd. without any restriction for SpaceWire user community.

3 REFERENCES

1. Tadayuki Takahashi, Masaharu Nomachi, Shigeru Ishii, Yoshikatsu Kuroda, and Hiroki Hihara, “Space Wire activities in Japan for science missions”, The second SpaceWire Working Group meeting, ESA/ESTEC, November 11th 2004.

2. Masaharu Nomachi, Shigeru Ishii, and Hiroki Hihara, “SpaceWire activities in Japan”, The fourth SpaceWire Working Group meeting, ESA/ESTEC, July 20th 2005.

SPACEWIRE-CPCI VXWORKS SUPPORT

Session: SpaceWire Test & Verification

Short Paper

Iain Martin, Steve Parkes, Stuart Mills

STAR-Dundee

c/o School of Computing, University of Dundee, Dundee, DD1 4HN, Scotland, UK

E-mail: iain@star-dundee.com, steve@star-dundee.com, stuart@star-dundee.com

ABSTRACT

STAR-Dundee [1] provides both PCI and cPCI boards based on the SMCS-SpW-FPGA (Field Programmable Gate Array) device from Astrium. This FPGA is functionally representative of a radiation tolerant chip created by Astrium/Atmel [2]. These PCI and cPCI devices are therefore ideally suited to support the development and testing of on-board SpaceWire components and systems intending to use the SMCS chip. VxWorks from WindRiver [3] is a widely used real-time operating system (RTOS) in the embedded industry and is an important tool within the space industry. VxWorks driver and support software have therefore been developed for the SpaceWire-cPCI and PCI-2 boards. This paper presents the SpaceWire VxWorks driver architecture, discusses integration with Board Support Packages (BSPs) and describes the support software.

The driver performance is given by showing packet transfer rates with varying packet size and an example of a real-time application of jitter measurement using the SpaceWire-cPCI as a time code master is also presented.

1 SOFTWARE OVERVIEW

The SpaceWire VxWorks driver API is a custom interface VxWorks driver in the form of a compiled C library. Board initialisation support is provided as customisable source code with full working examples for Intel x86 and PowerPC 750 targets. Test software is provided as source code and can be used as a template to quickly develop data transfer applications. The software interface is similar to the SpaceWire PCI-2 Windows and Linux driver interfaces but it contains differences specific to an RTOS. The SpaceWire-USB RMAP [4] and Router Configuration libraries, have been ported to VxWorks and a routing table download example is shown.

1.1 SOFTWARE COMPONENTS

This section describes the software components provided to support the SpaceWire cPCI with VxWorks. The software is provided in four separate sections: Driver initialisation code which must be added to the VxWorks build, the VxWorks custom driver library, RMAP and Config support libraries and a test code suite.

Figure 1 below shows a standard development VxWorks system with the SpaceWire cPCI software integrated within a VxWorks image and a downloadable module. The initialisation code is compiled within the VxWorks image. The test code is compiled as a downloadable module and linked to the pre-compiled driver library. The RMAP and Router Configuration libraries can also be included.

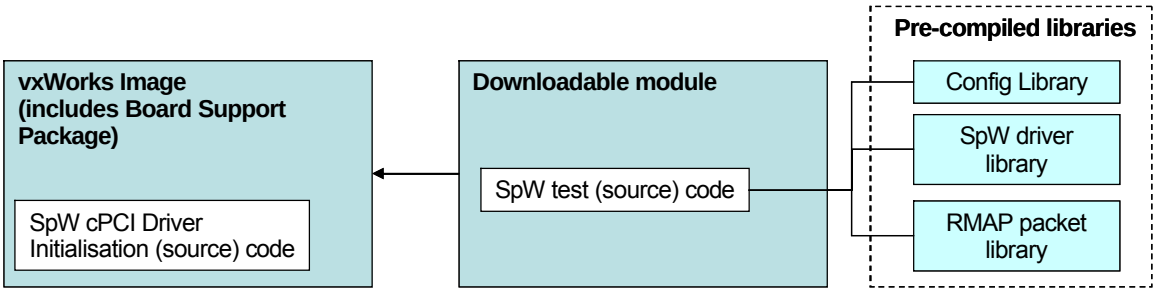


Figure 1: SpaceWire VxWorks driver software components

1.2 DRIVER INITIALISATION CODE

Driver initialisation code may need to be modified for specific targets and so is provided as source with working examples for Kontron cp620, Maxwell SCS750 and Intel x86 targets. In particular, memory mapping and the interrupt vector calculation can vary with different Board Support Packages. During initialisation, the PCI bus is scanned and an information table of detected cPCI devices is created. For each detected device this includes PCI BAR memory mapping addresses, a big/little endian flag, any error codes and an interrupt vector which is used to connect an ISR when the driver is opened. It is recommended that the initialisation code is run during VxWorks booting.

1.3 DRIVER LIBRARY

The driver library contains the VxWorks custom driver interface functions and is provided as a compiled archive. This is functionally similar to the PCI-2 Windows and Linux driver with functionality provided to support the following operations.

- Open and close devices with multiple devices supported,
- Link control which includes starting and stopping links, controlling link speed and obtaining status information,
- Data transfer functions to receive and transmit data directly to or from user buffers. Single and multiple packet transfers are supported as are transmitting and receiving raw unpacketed data. User tasks can be pended until data transfer is complete.
- Error injection and recovery,
- Time code support, and
- Low level DMA, register and memory access functions.

1.4 RMAP AND CONFIG LIBRARIES

The RMAP and Config libraries provide support for performing RMAP commands and for configuration of routers across a SpaceWire network. A router table download application is included to demonstrate the use of these libraries.

1.5 TEST CODE

The test code suite contains a range of confidence and performance test functions to demonstrate the functionality of the driver and also to provide example code which can be used as templates for data transfer applications.

2 DRIVER ARCHITECTURE

The driver architecture is tailored to the design of the SMCS3323 and PLX9056 PCI interface chips as shown in the block diagram in Figure 2. The Communication Memory Interface (COMI) of the SMCS332 transfers data to and from SpaceWire links to the dual port memory. There are two DMA channels which are used to transfer data to and from user buffers to the dual port memory. The Host Control Interface (HOCI) which bypasses the dual port memory is also supported but is significantly slower.

2.1 DATA TRANSFER

To transmit data from a user buffer out of a SpaceWire link, DMA is used to transfer data into the duel port memory. The data is then transmitted out of a SpaceWire Link using the COMI interface. Transparent internal DMA queues are used to share a single DMA channel between the three links. When a link is started, the COMI receive operation is started so any incoming data will immediately be received into the dual port memory. If a user buffer has been registered to receive data then this data will be transferred directly using DMA into the user buffer as it arrives. If no user buffer is available the data can be pended or discarded as desired. Again an internal queue is used to manage concurrent requests to DMA data from multiple links.

Each of the three SpaceWire interfaces can transmit and receive data concurrently. To enable efficient double-buffered data transfer separate internal packet transfer queues are maintained for each SpaceWire interface for both transmitting and receiving data. Each user buffer registered to transmit or receive data is allocated a buffer identifier. This can then be used to obtain the data transfer status or suspend a task until the transfer is complete or an error is detected.

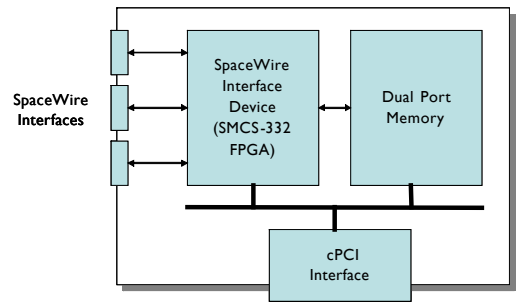


Figure 2: SpaceWire cPCI block diagram

2.2 INTERRUPT HANDLING

The driver uses a dedicated task to manage data transfer. This task acts as a "deferred procedure call" to perform any work required after an interrupt is received. The alternative model is to handle the driver interrupts directly in the Interrupt Service Routine (ISR) and so provide faster interrupt response which leads to smaller delays between packets. However the deferred procedure call model limits interrupt latency for other devices or services. The system designer can also set the priority of the driver task and so control the processor resource allocation so this is likely to be the best approach for a system with other drivers and time-critical interrupt servicing. However when servicing a high volume of small packets the task switch overhead may be an issue.

Figure 3 shows task scheduling timings during a single loopback test with packet sizes of 20,000 bytes. Task tSW0 on the left column task list is the driver task. Figure 4 shows a single loopback test with packet size of 100 bytes showing the increased processor time required to service the many extra interrupts required when using small packet sizes.

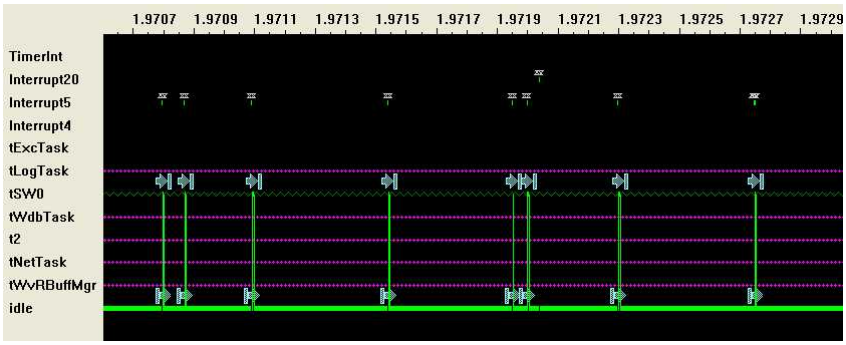


Figure 3: SpaceWire cPCI-2 driver task scheduling during a single loopback test with packet size of 20,000 bytes

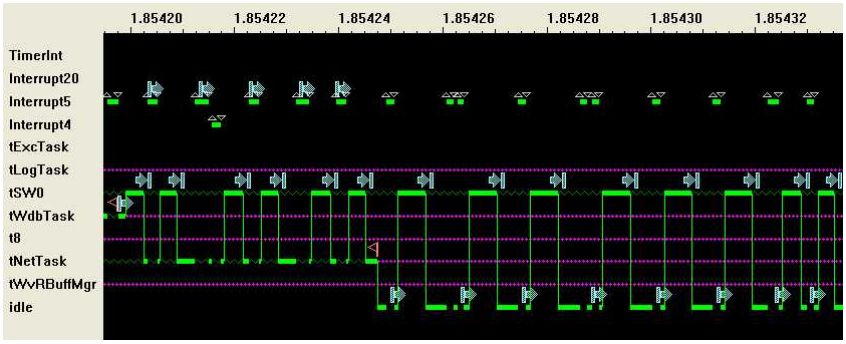


Figure 4: SpaceWire cPCI-2 driver task scheduling during a single loopback test with packet size of 100 bytes

3 PERFORMANCE

This section contains data rate transfer timings for the SpaceWire cPCI-2 board used with a Kontron cp620 bus master. Figure 5 shows the results of three data transfer tests all with varying packet size:

1. Single loopback where data is transferred out of a SpaceWire link and simultaneously read into another link connected by a SpaceWire cable. This tests two concurrent streams of data.
2. Double loop back where data is simultaneously transmitted and received out of two links connected with a SpaceWire cable. This tests four concurrent streams of data.
3. Triple loopback where data is simultaneously transmitted and received from all three links at the same time. Two links are connected with a SpaceWire cable and the other link has a single loopback cable connected. This tests six concurrent streams of SpaceWire data.

The tests use a varying packet size from 100 bytes to 5,000 bytes in steps of 100 bytes. Smaller packets sizes result in slower data rates due to increased interrupt processing overheads. When larger packets are used with multiple data streams, bus saturation is the limiting factor and is approximately 310 MBits per second when using the Kontron cp620 although this is target specific and tests with the Maxwell SCS750 gave a saturation level > 400 MBits/sec.

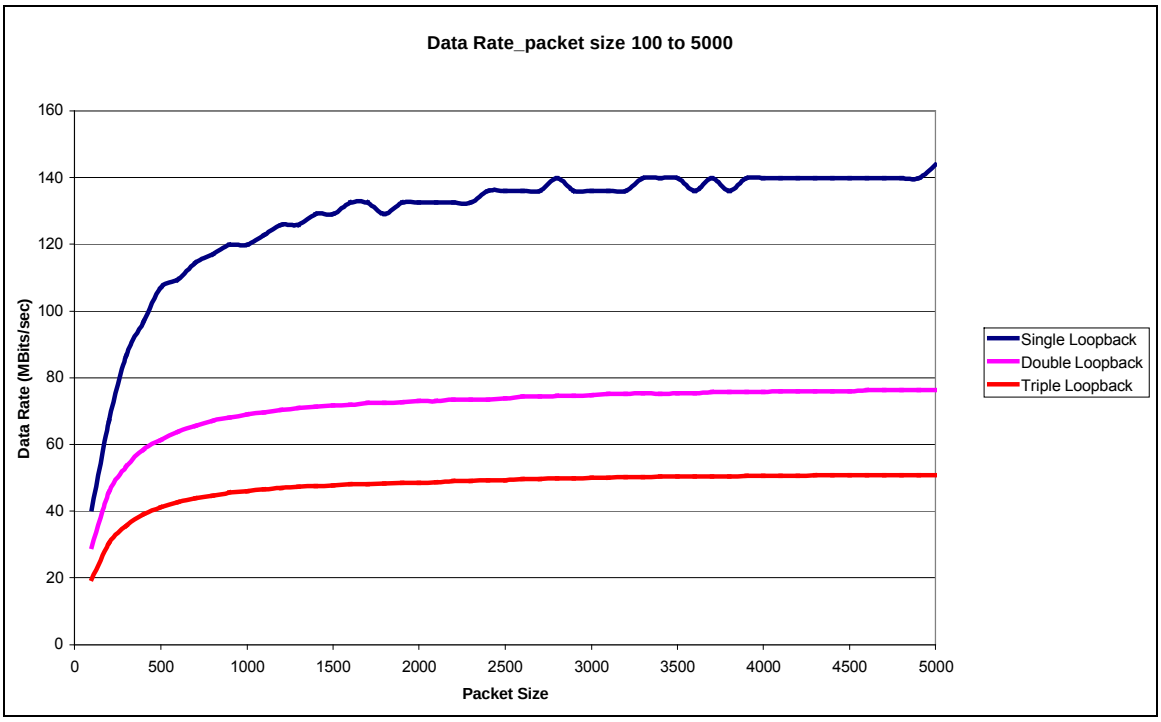


Figure 5: SpaceWire cPCI VxWorks data transfer rates with varying packet size

4 TIME CODE MASTER EXAMPLE APPLICATION

An example application was created to measure the jitter when using the cPCI board to send data packets at regular intervals controlled by time codes. A packet source application transmits a packet an exact specified time after receiving a time code. The time taken to send the packet is measured as the difference between the time code and

the EOP (End-Of-Packet-marker) being detected on the SpaceWire Link. Jitter is measured as the difference between the time to send each packet and the average. The SpaceWire cPCI board was used as a packet source, time codes were provided by a SpaceWire Router-USB and the accurate timing measurements were obtained using a SpaceWire Link Analyser. The largest jitter figure measured was 1.5 microseconds.

The router is connected to the cPCI via the Link Analyser as shown in Figure 6. The router is used as a time code master and also as a packet sink to receive the packets generated by the cPCI. The Link Analyser is used to generate accurate, independent timing data. The cPCI is used to transmit packets.

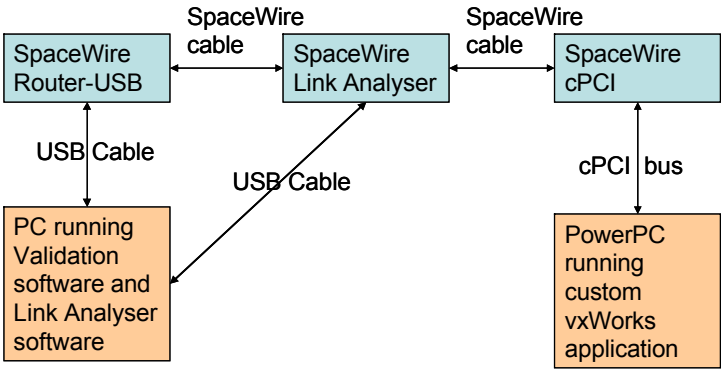


Figure 6: Jitter test setup diagram

5 CONCLUSIONS

VxWorks driver and support software has been created for the SpaceWire cPCI device. This software has a similar interface to the driver API available for the SpaceWire PCI-2 device for Windows and Linux. This allows applications using these devices to be easily ported to and from VxWorks. The RMAP and Config libraries are direct ports from the Router USB versions and so provide continuity between these products.

6 REFERENCES

1. "STAR-Dundee Web Site", STAR-Dundee, http://www.star-dundee.com.

2. "The SMCS332SpW / SMCS116SpW SpaceWire Communication Controller ASICs", S. Fischer, L. Stopfkuchen, U. Liebstückel, P. Rastetter, L. Tunesi, EADS Astrium GmbH, 2005 MAPLD International Conference, Washington, D.C., September 7-9, 2005.

3. "WindRiver Web Site", WindRiver, http://www.windriver.com

4. "SpaceWire Remote Memory Access Protocol", S. Parkes, C. McClements, DASIA 2005, 30 May - 2 June, 2005, Edinburgh, Scotland. Edited by L. Ouwehand. ESA SP-602. European Space Agency, 2005. Published on CDROM., p.18.1

SPINSAW – THE SPACEWIRE NETWORK SYSTEM ADMINISTRATOR WORKSTATION

Session: Test & Verification

Short Paper

Liudmila Onishchenko, Elena Suvorova, Alexander Cherny

Saint-Petersburg University of Aerospace Instrumentation. 67, B. Morskaya, Saint-Petersburg, Russia

E-mail:luda_o@rambler.ru, wildcat15@yandex.ru, alexblack@inbox.ru

ABSTRACT

SpiNSAW (SpaceWire Network System Administrator Workstation) is a software tool with GUI for setting of local and remote switch configuration. SpiNSAW can work in simple and in extended modes. In the simple mode the main switch components available for software are set, such as routing tables and links transmission speeds. In the extended mode you can set all the switch registers that are available for software. SpiNSAW can be used in a network with different types of a switch and nodes, it can be a useful means for network administration and for device testing during system and distributed algorithms developing.

1 INTRODUCTION

An important task in SpaceWire interconnections is network switches administration. A user should have an opportunity to set any parameters of any switch in a network. For this task the software tool SpaceWire Network System Administrator Workstation (SpiNSAW) was built. This tool provides state monitoring and operating modes settings for SpaceWire switches in a SpaceWire network. SpiNSAW is an application with graphical user interface working on a PC. A switch can be either directly connected to the PC through the COM-port for performing read and writing operations of all switch components available for software by using direct connection to a switch or by the RMAP (Remote Memory Access Protocol). Both modes provide means for saving current settings of every switch in the SpaceWire interconnection network to a file and for downloading them back.

The main task of the SpiNSAW is the network administration, but it may be useful for devices testing in the network and for distributed algorithms testing.

2 SPINSAW OPERATING MODE

The SpiNSAW can work in two different modes. One, simple mode is for users who would not like to know any details about switch internal structure. In this case only most important parameters could be set in quite a simple way: the SpaceWire switch routing table, transmission and receiving speed, and link ports' state. The example of the simple mode is shown on the Figure.

The second mode is an extended one and requires a user to understand the functions of the routing switch components that are accessible for software. It allows monitoring the current switches state in details, i.e. look the state of every channel, set transmission speed, determine adaptive group routing, send Time-codes and distributed interrupts, generate data packets and send them, monitor the error statistics, read|write configuration and operation mode registers of the switch.

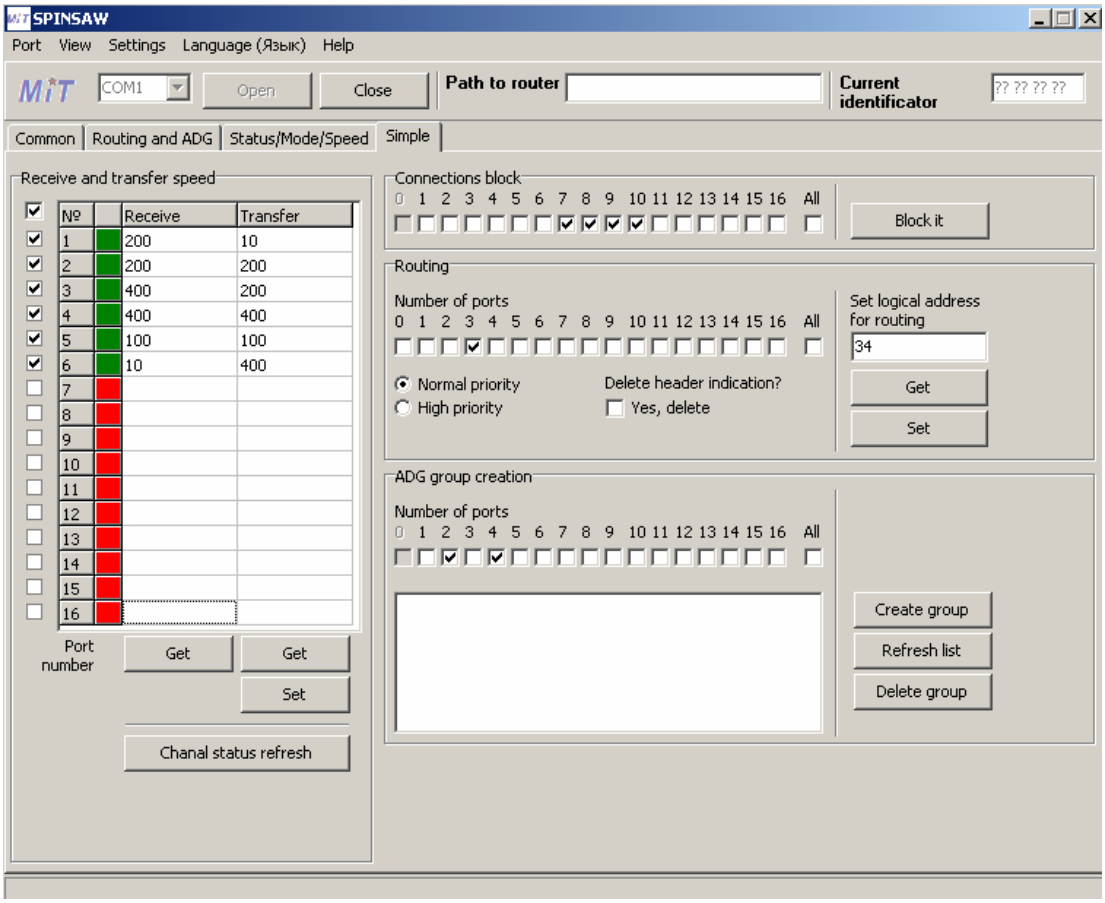


Figure. SpiNSAW simple operating mode

3 USING SPINSAW

3.1 LOCAL SWITCH CONFIGURATION

In the simplest case the SpiNSAW can be used for configuration of one switch MCK01, which is connected to the PC by a COM-port. In this case SpiNSAW forms instructions according to user's operations in a certain format and sends them to the switch through the COM-port. Switch MCK01 processes such instructions, form answers on them and sent them back to the SpiNSAW.

3.2 REMOTE SWITCH CONFIGURATION

To provide SpiNSAW access to the network several ways could be used. It depends on the devises' type that are used in the network.

The simplest way is when a switch, which has a special handler of instructions from the COM-port, is connected to the PC through the COM-port (for example, MCK01

switch). A remote configured switch must support processing of RMAP-packets and its address space distribution must be known. For other switches in the network it is sufficient that they transmit packets according to the SpaceWire standard. In this case configuration settings of the remote switch could be presented as a sequence of operations. By using SpiNSAW GUI the user specifies an instruction, for example sets the logic address in the routing table. According to the selected operation SpiNSAW forms an instruction in the form of a RMAP-packet. SpiNSAW sends RMAP-packet through the COM-port to the attached to the PC switch, which in turn sends it to the network. The RMAP-packet reaches the requested switch that processes it and sends a reply RMAP-packet. The directly connected to the PC switch receives this reply RMAP-packet and transmits it through the COM-port to the PC, where SpiNSAW is expecting it.

The SpiNSAW can be used also, with some limitations, if in the network there are switches with an unknown for SpiNSAW address space distribution.. At least one switch connected to the PC through the COM-port which can process messages from the SpiNSAW should be in the network. So the user can form RMAP-packet himself giving addresses, and SpiNSAW will send generated packet to the network and then receives an answer from the network.

In case the address space distribution of configured switch is known, but there is no switch connected to the PC through the COM-port the SpiNSAW also can be used. It is possible to use such off-the shelf devices as PCI-SpaceWire Bridge, USB brick, etc. In this case SpiNSAW forms RMAP-packet and transmits it to the software of the device which is used. The device software transmits it to the network through the SpaceWire channel. When reply RMAP-packet is received, the device's software should send it to the SpiNSAW for the further processing.

4 CONCLUSION

So SpiNSAW is a convenient tool for SpaceWire switches network administration. It makes possible to set switches' operating modes in the network and to control and monitor their state in a simple way. The SpiNSAW allows to send time-codes, distributed interrupts, RMAP-packets and data packets that can be convenient for distributed system work testing. SpiNSAW can be used for the network administration with different types of devices.

A METHODOLOGY AND THE TOOL FOR TESTING SPACEWIRE ROUTING SWITCHES

Session: SpaceWire Test & Verification

Short Paper

Elena Suvorova

Saint-Petersburg University of Aerospace Instrumentation. 67, B. Morskaya, Saint-Petersburg, Russia

E-mail: wildcat15@yandex.ru

ABSTRACT

SpaceWire routing switches could include functions of multicasting and adaptive routing. But supporting of these functions essentially complicated verification and testing process.

The article presents the developed test shell for verification and parameter evaluation of routing switches with different number of ports and for networks with different topology and number of switches; for data packet flows automatic generation with different parameters. Also the test shell could be used for testing of RTL models and post-synthesis netlists.

We use SystemC and Cadence SimVisio (simulator ncSim) design tools for these test shell development. SystemC provides development of parameterized models with flexible behaviour. SimVisio tools allows integration of mixed language models (SystemC, VHDL, Verilog) in one project.

1 INTRODUCTION

A network model is represented by the parametrized shell that includes models of SpaceWire routing switches, terminal nodes, interconnection lines, and a control and monitoring block. When a user specifies parameters (number of nodes, switches, interconnections topology, and other parameters) the shell configures the network model automatically. We use two network topology specification methods: a universal one – a network is described by a full interconnection graph specification, and a method for regular topologies specification (e.g. different meshes, hypercubes). In the second method a network structure is described analytically; topology type and number of nodes is user defined. The examples of networks are showed on the figure 1.

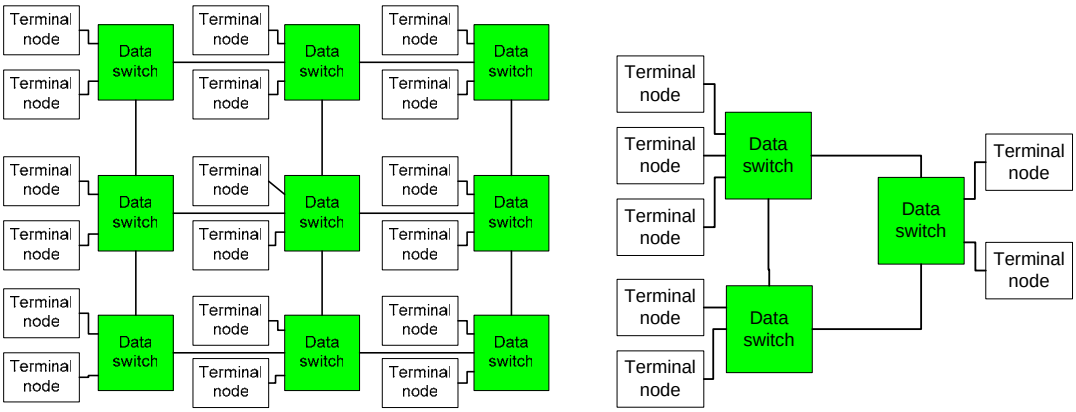


Figure 1. Examples of switch based networks

Users could apply this model for verification and parameter evaluation of his SpaceWire routing switches.

We not present on figure 1 the control and monitoring block. It connected to all terminal nodes and data switches for performing simulation control (initialization of routing switches and terminal nodes, starts data transmission, global result checking). The models of the shell, routing switches, terminal nodes, interconnection lines, and a control and monitoring block are written in SystemC. RTL routing switches models on VHDL or Verilog and netlists (VHDL, Verilog) also could be used.

The included into the network components could be described with different detalization. This allowed us choose between the resultant accuracy and reasonable simulation time. We develop some switch models on systemC with different detalization for preliminary simulation and parameter evaluation. We are using these models for selection bufferisation method and buffer size, for selection and verification arbitration scheme, and for decision some other questions affect to switch performance. The network models, included RTL models of data switches or netlists, are more detailed. The structure of these models illustrated by figure 2

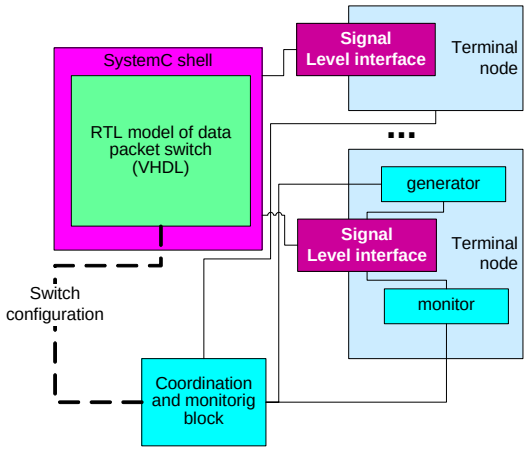


Figure 2. Example of network model included RTL model of data switch

These models includes some special components: SystemC shells for components described on VHDL and Verilog (these shells required by Cadence design tools) and

signal level interfaces for terminal nodes (basically terminal nodes have not signal but character level interfaces)

The models, suggested in this article, are correlated with SpWFM model created in SUAI, but oriented to detailed simulation of different arbitration schemes and to test and verification of RTL and netlists switch descriptions.

2 VERIFICATION AND TESTING PROCESS

The test process in our model includes next three stages:

1. The initial configuration stage. On this stage the coordination and monitoring block executes initial settings for switch models and terminal nodes models. Initial settings for switch models includes writing information for logical addresses to routing table, writing appropriate values to adaptive group registers and if RTL models of switches are used writing appropriate values to implementation specific registers. Initial settings for terminal nodes include data flow and control codes flow parameters.

2. The data packet transmission stage. On these stage data packets are transmitted between the switches and the terminal nodes. The duration of this stage could be assigned as absolute value or user could assign number of packets that are transmitted from every terminal node. The terminal nodes execute preliminary control of data packet transmission and timing parameters evaluation at this stage.

3. The global data packet control and collects statistics.

During data packet transmission testing we need to control: packet header must be excluded from the packet or not excluded according routing table settings; packet contents and end-of-packet symbol must follow without changes. The set of ports, which received this packet, must correspond to its address and adaptive routing settings. For transmission control every packet contents includes special information. The packet generator writes every transmitted packet into a log file. When the monitor receives the packet it uses log files for control packet contents and end-of-packet symbol, control of packet header processing. But only the coordination and monitoring block executes final packet header processing control. It controls multicast transmission and adaptive group routing. It control that multicast packet received by all terminal nodes includes in appropriate multicast group and not received by any other terminal nodes. Also it check that packet addressed to terminal node includes into adaptive routing group (or translated via switches which ports includes in adaptive routing group) is received by only one terminal node from appropriate adaptive routing group. The and monitoring block used adaptive routing registers values, routing tables ad special information includes into packets. This special information includes number of source terminal node, address field (for some addressing types address field is deleted from packet and ordinal number of packet in terminal node). Ordinal number of packet need for reordering control.

3 PERFORMANCE EVALUATION

One of tasks that can be done with such a model is a choice of effective buffering scheme when the ports' load is asymmetric. The rate of source ports is several times

less than the rate of port connected to handler port. If the fly-by commutation is used, then interconnection line to the handler will stand idle and waiting time for packets from other sources will increase. As a result the line to handler will stand idle during time enough for some symbols transmission after transmission of every one symbol. Using of switching with buffering allows decreasing of high rate lines idle time due to preliminary data accumulation in a buffer. However efficiency of buffering is essentially decreased when the packet size is greater than the buffer size due to the packet tail that is not placed into buffer when its transmission to destination port is started would be transferred with slow rate with it is arrived into switch. The developed switch model allowed to evaluate maximal acceptable packet size when buffer size is fixed or evaluation of necessary buffer size for a given packet size (average packet size and distribution law).

The model could be used for evaluation of a ratio between the switch fabric data channel throughput and the port throughput (interconnection lines throughput). Increasing of switch fabric data channel throughput could be reached by increasing local clock frequency or by increasing of number of bytes transmitted in parallel in a switch fabric data channel. Increasing of the switch fabric data channel throughput in combination with packet buffering leads to increasing packet transmission rate through switch fabric, but the port throughput will be a limitation factor. We obtain next results with using our model. The throughput ratio essentially affects the average packet transmission time when input port load is more than 95%. When the input port load is 99% increasing of throughput ratio from 1:1 to 4:3 allow decreasing the average packet transmission time to 11%. Further increasing the throughput ratio is less effective: an increase of throughput ratio from 4:3 to 2:1 leads to only 5%decrease of the average packet transmission time.

Missions & Applications 2

Wednesday 19th September

13:45 – 14:15

SWIFU: SPACEWIRE INTERFACE UNIT FOR VERSATILE SENSOR INTEGRATION ON THE EXOMARS CHASSIS BREADBOARD

Session: Spacewire Missions and Applications

Short Paper

Lee C. G.-Y., Obstei R.

von Hoerner & Sulger GmbH, Schlossplatz 8, Schwetzingen, D-68723, Germany

E-mail: lee@vh-s.de, obstei@vh-s.de

ABSTRACT

This paper describes a versatile interface unit which will allow quick integration of a distributed sensor system into a Spacewire network. Each unit contains two Spacewire interfaces plus an RMAP decoder and functions both a simple router and a data interface: Spacewire packets are accepted by the unit and passed to the RMAP decoder only if the packets logical address matches the units node ID. Otherwise the packet is passed to the alternate interface for transmission to the next node in the chain. In this way the network is built up by daisy chaining units together.

Accepted packets are decoded using the RMAP protocol which access a generic IO bus. Various different IO peripherals may be attached to this bus as required by the system and thus the unit can act as both a gateway to other protocols such as RS232 and CAN as well as direct interface to sensors from which data is generated. As well as servicing RMAP requests, the unit may also be configured to initiate RMAP requests upon specific events – data arriving on a port or if a given Spacewire time code is received. In this way resources may be free from the task of polling all the SWIFU units for data.

The current implementation of the unit is target to mixed signal FPGA technology to allow a single chip sensor solution and is being used as an integral part of the Exomars Chassis Breadboard being built by von Hoerner & Sulger GmbH.

1 INTRODUCTION

1.1 MOTIVATION

Over the last 30 years von Hoerner & Sulger GmbH (vHS) have designed and built custom space instruments for a large number of space missions belonging to US, European and Russian space agencies. Examples of which are COSIMA on ESA Rosetta and CIDA on NASA's successful Stardust mission. Each agency and even each mission has required different TM/TC protocols and the company has had to been able support a variety of interface protocols from simple RS-232 style protocols to more sophisticated MIL-STD-1355 interfaces.

The use of Spacewire within spacecraft data system data architecture has now become more and more common and it increasingly required that instruments and subsystems interface to directly to the on-board data handling system through this medium. It is highly likely that projects that the company are currently involved in, such as BELA, the laser altimeter on ESA Bepi-Columbo and Exomars Chassis and Locomotion subsystem will require Spacewire interfaces to able to communicate and with this in mind it was decided to develop the company expertise in this area by initiating this internal development project.

1.2 CONCEPT

The aim was the development project was to raise the expertise in the Spacewire area and to provide a cost effective method to implement a prototype Spacewire compatible systems. The project has two design goals:

- Development of a RMAP request server module with versatile peripheral interface.
- Development of simplistic routing module to allow a daisy chain network topology to be created.

A single Spacewire Interface Unit (SWIFU) will contain both modules and all the necessary peripheral interface logic implemented on a single FPGA device.

2 NODE DESIGN

The functional block diagram of a node is shown in

Figure 1. Each node contains two space wire CODEC link interfaces. The two interfaces are connected together by routing logic that is also attached to the RMAP decoder. Each node also possesses an 8 bit unique identifier set in hardware which acts as a logical address for the Spacewire network. Requests to the RMAP decoder may access any of the IO modules via a peripheral address bus.

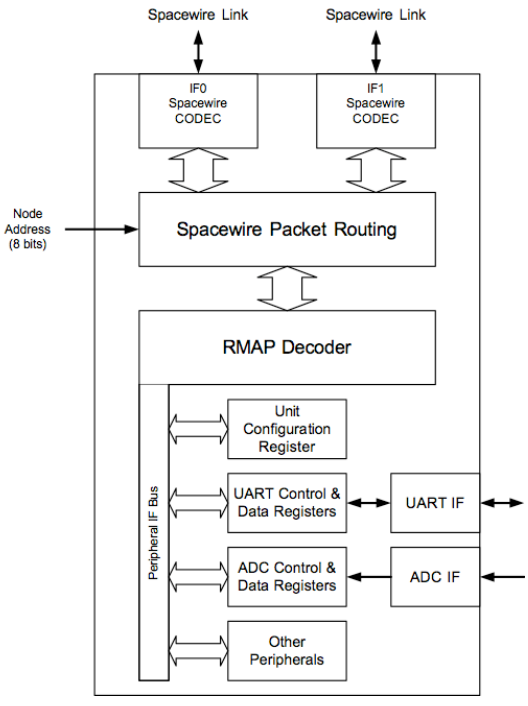


Figure 1 SWIFU block diagram

2.1 ROUTING LOGIC

When a packet arrives at a node, the node stores the first byte inside the receiving interface and checks where the packet should be routed to on the basis of the destination byte. If the destination address matches the node's identifier, the packet routing module directs the packet to the RMAP decoder if it is idle. If the RMAP decoder is busy with another request, Spacewire's flow control mechanism is used to hold the packet until the current request is completed.

If the destination address does not match the node's identifier then the packet will be transmitted through to the alternate Spacewire interface using a wormhole routing mechanism. In the special case that the alternate interface has not established an initialised link the packet is looped back on the receiving link.

The value of the received time codes are stored in the node and made available all parts of the node. Received time codes are also routed to the alternate interface as prescribed by the standard – i.e. codes which have time values equal to current time code value are suppressed.

2.2 RMAP REQUEST DECODER

As the RMAP decoder receives a packet, the header CRC is validated and the header is stored. If the packet contains a read request, the decoder uses the header information to retrieve the requested data into the data buffer, which is then used to form the data block in response to the request. Conversely for a write command, the packet's data block is stored in the data buffer and if no errors detected by the data CRC is written to the location defined in the header information. In either case, the response modifies the stored header information and a new CRC calculated for the header and data block if necessary. The response if then transmitted via the same interface as the request was received on.

2.3 PERIPHERAL INTERFACE

The RMAP decoder is connected to the node peripherals using a bus system following the Wishbone specification. In addition to the signals defined by the Wishbone specification, the bus includes discrete signals to allow each peripheral to indicate an occurrence of an event and to interpret information relating to the receipt and value of Spacewire time codes.

The peripheral address map is divided into 8 separate address spaces of 12 bits each. The first address space is reserved for configuration and status registers of the node. Other address spaces are then available for any peripheral module that may be required for a given the application. Each peripheral module must respect provide the following rules:

- The peripheral may utilise as little or as much of the address range 0x000 to 0x7FF.
- Address 0x800 to 0x803 Event Description registers (all other addresses are ignored) which must provide valid data if accessed.
- Access to an invalid address location is signalled with using the ERR signal in the Wishbone bus.

Apart from these rules the peripheral model may comprise of type of peripheral that can be implemented in programmable logic such as CAN interfaces, UART and lower level digital and analog input/output. In all cases the Spacewire network is transparent to the peripheral.

2.4 EVENT DATA TRANSFER

The design also allows the node to act as a client and to generate a RMAP write request on the occurrence of predefined event. The destination address for the request is predefined in the nodes configuration registers whilst the circumstances that initiate the event are defined with by the peripheral module logic itself. Typically an event could be generated upon the receipt of a specific Spacewire time or when the peripheral itself contains a certain amount of data.

When an event occurs, the peripheral signals the RMAP decoder that it has a pending event. The decoder then generates the RMAP message using information contained in the event configuration registers at locations 0x800-0x803 to collect the specified data from the peripheral.

3 NETWORK TOPOLOGY

By connecting several of these node units together a simple daisy chain network can be built. An operational network would consist not only of these slave nodes (SN) which receive and process normal RMAP requests but also a master node (MN) which generates the requests. The master node is also configured as the destination of RMAP packets generated by events at the SN's. Typically the MN would be the application's processing environment and the SN's are remote peripherals to which it communicates with. The advantage of using such a daisy chain system is that it is possible to build a space wire system without the need for a dedicated router. Once implemented it is also simple to expand the number of SN with out the need of expensive routing components.

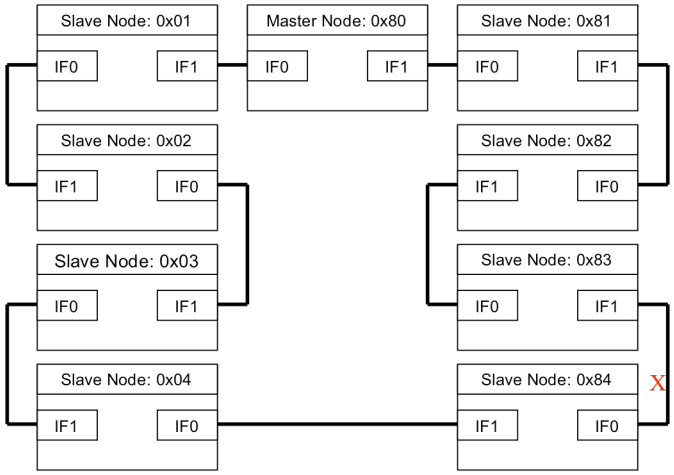


Figure 2 SWIFU Ring Network

A weakness of such a daisy-chained system is reliability. If a node or link fails then the any SN further down the chain from the MN is cut off. One method to improve reliability is to form a complete ring with the chain. A possible implementation is shown in Figure 2. Here the MN is designated with address 0x80. The SN addresses are set so that if the number of steps to the MN via interface 0 is less than interface 1, the most significant bit of the node's identifier is set. The master node is initialised so that it has a routing table where all packets destined to addresses whose MSB is set is sent to interface 1and otherwise to interface 0. If a link failure occurs at point X in the diagram, RMAP requests destined for Node 0x84 would be "looped back" at Node 0x83. Thus the MN will be able to detect the broken network and modify its routing for Node 0x84 to go via IF0. Such a routing table would require little resources in the MN as each entry can be represented with a single bit or two if unreachable addresses were also to be identified.

4 ROVER APPLICATION

The current SWIFU development will be used as part of the Phase B1 Exomars Chassis Breadboard by the Oerlikon-Space led consortium of which vHS is the major German partner. It is required that the electrical data system is implemented as a Spacewire network. The current design comprises of 6 steerable wheel drives each containing a single node providing an RMAP data interface to:

- COTS steering and wheel motor controller
- Steering position sensors
- Force and Torque sensors on the wheels
- General voltage, current and temperature monitoring

Additional nodes provide interfaces to system housekeeping, suspension position sensors and the IMU. The Master node is provided by the chassis's Onboard Computer.

Although this implementation is for the breadboard only, some advantages could be seen for a flight implementation. An over-riding requirement for the Exomars rover is mass which should be minimised. For a system critical function such as rover motion it would also be desirable to have fully redundant space wire links to the motor controllers so that a single electrical harness failure would not cause the rover to be immobile. However if these controllers were located on the suspension members the amount of harness required for a standard star network would be excessive. The use of a daisy chain network such as this would provide some fault tolerance to such failures whilst economising on mass.

5 FUTURE

The development of the node unit is nearing completion and hardware will soon be fabricated for use on the Exomars breadboard.

It also is hoped that SWIFU will be used on other future projects and in other fields. It is also being considered to extend its features to allow small ring networks to be incorporated into networks which utilise the path addressing mechanism.

6 ACKNOWLEDGEMENTS

The Exomars Breadboard Consortium is comprises of Oerlikon Space (lead) von Hoerner & Sulger GmbH (Germany) , Bluebotics SA (Switzerland), DLR-RM Oberpfaffenhofen (Germany), and ETHZ (Swiss Federal Institute of Technology Zurich).

APPLICATION OF SPACEWIRE TO FUTURE SATELLITE DATA PROCESSING SYSTEM

Session: SpaceWire Missions and Applications

Short Paper

Kenji Matsuda, Kazunori Msukawa, Shigeru Ishii, Yo Watanabe, Yoshikatsu Kuroda

Mitsubishi Heavy Industries Ltd, 1200 O-Aza Higashi Tanaka, Komaki, Aichi 485-8561, Japan

Motohide Kokubun, Masanobu Ozaki, Tadayuki Takahashi

Department of High Energy Astrophysics, Institute of Space and Astronautical Science (ISAS), Japan Aerospace Exploration Agency (JAXA), 3-1-1 Yoshinodai, Sagamihara, Kanagawa 229-8510, Japan

Masaharu Nomachi

Laboratory of Nuclear Studies, Graduate School of Science, Osaka University, 1-1 Machikaneyama, Toyonaka, Osaka 560-0043, Japan

E-mail: kenji4_matsuda@mhi.co.jp, kazunori_masukawa@mhi.co.jp, shigeru_ishii@mhi.co.jp, yo_watanabe@mhi.co.jp, yoshikatsu_kuroda@mhi.co.jp, kokubun@astro.isas.jaxa.jp, ozaki@astro.isas.jaxa.jp, takahasi@astro.isas.jaxa.jp, nomachi@fn.lns.sci.osaka-u.ac.jp

ABSTRACT

In managing project of development of satellites, development period, cost and risk have been a pain in the neck. One of the reasons of this is that many different communication protocols have needed to be developed for each satellite or communication line to connect many electrical components because those protocols were chosen as the most adequate protocol for each communication line. We aim to adopt SpaceWire protocol for all communication lines in satellite to shorten development period and reduce cost and risk of the development.

1 INTRODUCTION

Up to now, we have often developed satellite in which electrical equipments are connected as shown in fig.1. In this system, all equipment or system is designed for the best performance to a satellite as a whole. That is, different types of analogue sensors output raw measured data, and after processing those data, they are transmitted to satellite bus system through a dedicated communication line or by many different protocols. Therefore, Analogue sensor components, data processing components, communication protocol between them and sometimes communication

protocol between data processing components and satellite bus system needed to be developed individually. That has caused long development period, high-cost, and high-risk development.

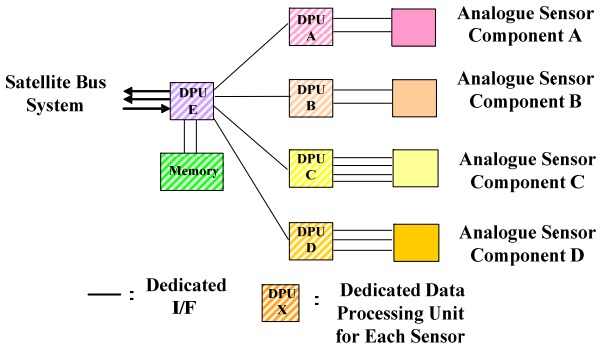


Fig. 1 Ordinary Satellite System

2 OUR SATELLITE DEVELOPMENT

2.1 DEVELOPMENT POLICY

To resolve the above mentioned issue, we develop our product based on the following policy; (1) To use SpaceWire protocol as the only one protocol that is used as inter-component protocol. (2) To link data processing units together. (3) To develop standard hardware. (4) To use standard software platform (TRON-OS).

2.2 OVERALL SYSTEM

We are about to organize a network between electrical components in satellite as shown in fig.2. The feature of this system is as follows; all of the communication protocol between all components (between analogue sensor component and data processing unit, between data processing unit and other data processing unit and between data processing unit and satellite bus system) is SpaceWire protocol. Thus, we are trying to shorten development time and reduce the cost of development, although the development of the very first satellite that is designed by this concept may take some time and money. One failure tolerance is achieved by having one extra data processing unit and linking them as a ring, because even if one component or one cable is disabled, data can be transferred via another cable or component.

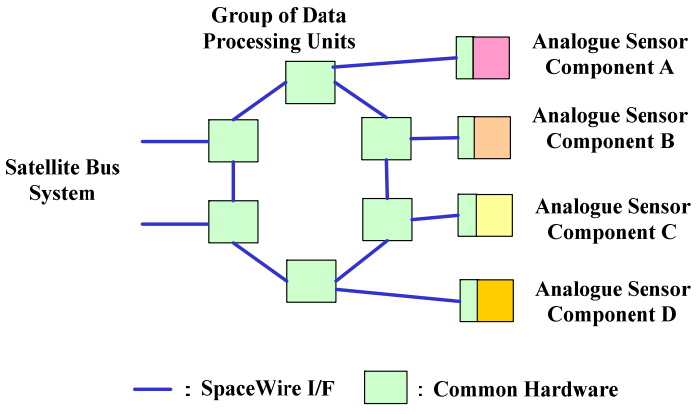


Fig. 2 New Satellite System

2.3 COMPOSITION OF EACH COMPONENT

Next, we would like to focus on the each component itself. Fig.3 shows an example of composition of an analogue sensor component. SpaceWire Interface Section is standardized and communication protocol between SpW Interface Section and User Section is provided as a well-defined protocol for efficient development, so SpaceWire Interface Section and User Section can be developed at the same time if necessary.

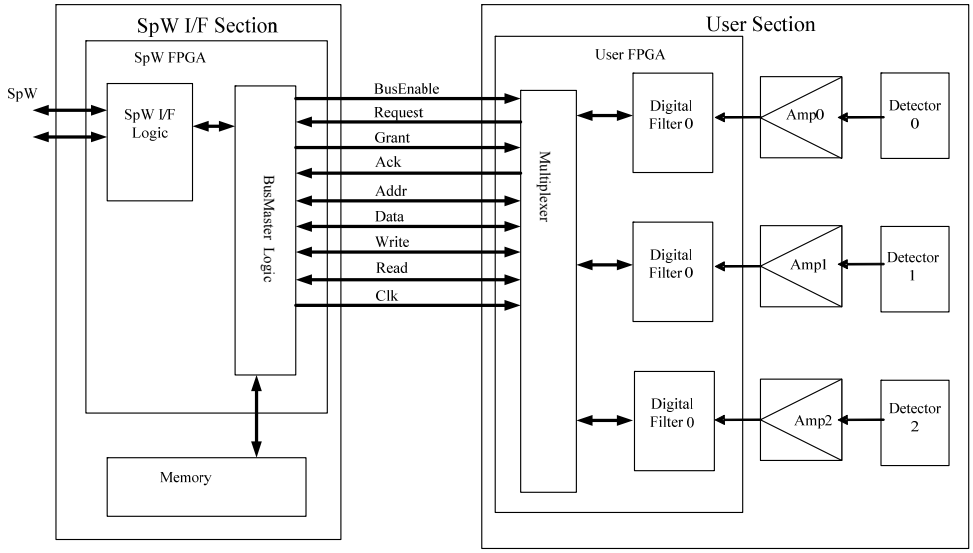


Fig. 3 Analogue Sensor Component

Construction of a data processing unit is illustrated in fig.4. It has SOI-SOC (Silicon on Insulator – System on Chip), SRAM, SDRAM, and EEPROM and can deal with processing of data such as image processing. SOI-SOC consists of RISC, memory interface and even SpaceWire interface in one chip, and features SEU-tolerance and is SEL-free. And it works on TRON-OS, which many engineers are familiar to use, for the reduction of the period of the development of software.

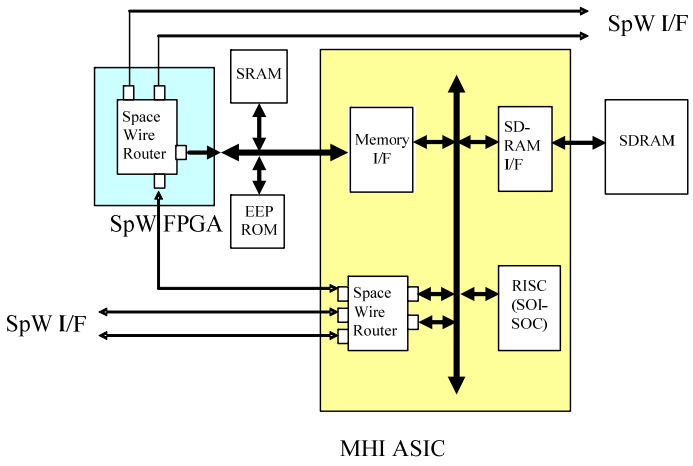


Fig. 4 Data Processing Unit

3 SUMMARY

For the purpose of shortening of development period and reduction of cost and risk of development of satellite, we use SpaceWire protocol for the communication not only between satellite bus system and data processing unit but also between data processing unit and analogue sensor unit. Furthermore, we are trying to achieve redundancy without too much components and cables by linking data processing unit together like a ring. This concept will be applied to the NeXT (New X-ray Telescope) satellite, which is scheduled to be launched in 2012 by JAXA.

DEVELOPMENT OF SPACEWIRE-BASED DATA ACQUISITION SYSTEM FOR A SEMICONDUCTOR COMPTON CAMERA

Session: SpaceWire Missions and Applications

Short Paper

Hirokazu Odaka, Motohide Kokubun, Shin Watanabe, Shin'ichiro Takeda,
Takeshi Takashima, Tadayuki Takahashi,

Department of High Energy Astrophysics, Institute of Space and Astronautical Science (ISAS), Japan Aerospace Exploration Agnency (JAXA),
3-1-1 Yoshinodai, Sagamihara, Kanagawa 229-8510, Japan

Takayuki Yuasa, Kazuhiro Nakazawa, Kazuo Makishima,

The University of Tokyo, 7-3-1 Hongo, Bunkyo, Tokyo, Japan

Masaharu Nomachi,

Laboratory of Nuclear Studies, Graduate School of Science, Osaka University,
1-1 Machikaneyama, Toyonaka, Osaka 560-0043

Hiroki Hihara,

NEC TOSHIBA Space Systems, Ltd. ,
10, Nisshin-cho 1-chome, Fuchu, Tokyo 183-8551, Japan

Takayuki Tohma,

NEC Corporation, 10, Nisshin-cho 1-chome, Fuchu, Tokyo 183-8551, Japan

E-mail: odaka@astro.isas.jaxa.jp, kokubun@astro.isas.jaxa.jp, watanabe@astro.isas.jaxa.jp, takeda@astro.isas.jaxa.jp, ttakeshi@stp.isas.jaxa.jp, takahasi@astro.isas.jaxa.jp, yuasa@amalthea.phys.s.u-tokyo.ac.jp, nakazawa@phys.s.u-tokyo.ac.jp, maxima@phys.s.u-tokyo.ac.jp, nomachi@fn.lns.sci.osaka-u.ac.jp, hihara.hiroki@ntspace.jp, t-tohma@bx.jp.nec.com

ABSTRACT

We have been developing a compact data acquisition system using SpaceWire interface for a semiconductor Compton gamma-ray camera. The system, which is based on our multi-purpose data acquisition framework, consists of a very small computer called SpaceCube and an electric circuit board with SpaceWire interfaces. Adoption of remote memory access protocol (RMAP) allows us simple handling of data. We can operate the detector and read data from the detector by software using RMAP libraries. We apply the system to a balloon-borne experiment which utilize the semiconductor Compton camera as a focal plane detector of hard X-ray imaging optics. In this paper we describe overview of the data acquisition system of the Compton camera based on SpaceWire technology and its application to the balloon-borne experiment.

1 INTRODUCTION

We have developed a semiconductor Compton camera which achieves gamma-ray spectral imaging with high energy resolution and high angular resolution[1]. It is a promising detector for the next-generation astrophysical observation of high energy universe. The camera consists of four double-sided silicon strip detectors (DSSDs) and 32 cadmium telluride (CdTe) pixel detectors (Fig. 1). The DSSDs form into a four-layer stack and the CdTe pixel detectors surround the bottom and horizontal direction of them in order to detect gamma-ray photons scattered at the DSSD part. Since each individual DSSD has 64 strips on each side and each CdTe pixel detector has 8x8 pixels, we have to read out many channels (total 2560 channels). Thus a high-speed and multi-channel data acquisition (DAQ) system is required under limitation of resources on a balloon. For this purpose, we have been developing a new compact DAQ system based on SpaceWire interface (I/F) and remote memory access protocol (RMAP) for a balloon-borne experiment.

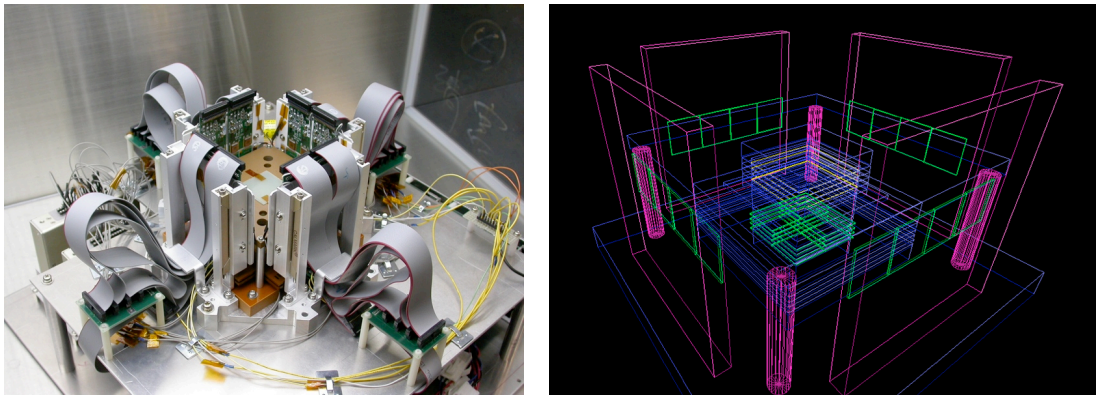


Fig. 1 Left panel: a picture of the Si/CdTe semiconductor Compton camera. Right panel: structure of the Compton camera. Si and CdTe detectors are drawn as yellow line and green line, respectively.

2 DATA ACQUISITION SYSTEM OF SEMICONDUCTOR COMPTON CAMERA

For a data acquisition system of the Compton camera we utilize our SpaceWire-based multi-purpose DAQ framework[2]. The framework includes common functions of data acquisition for scientific detectors, and therefore users of it are allowed focusing on development of application-dependent functions. The DAQ system of Compton camera consists of a very small computer called SpaceCube with SpaceWire I/F (the size of SpaceCube is 52x52x55 mm) and SpaceWire Digital IO board (Fig. 2). The circuit board has digital inputs/outputs (LVCMOS, LVDS), a 16 MB SDRAM for data buffer, two FPGAs (SpaceWire FPGA and User FPGA), and SpaceWire I/F. SpaceWire FPGA has SpaceWire RMAP engine. User FPGA, which connects to digital I/Os and SpaceWire FPGA, can be designed by users for data processing.

Figure 3 illustrates the DAQ system of the Compton camera. The Compton camera is composed of six independent detector modules (two DSSD modules and four CdTe detector modules). Each module has six digital ports (*TrigOut*, *TrigIn*, *Command*, *Command-clock*, *Data*, *Data-clock*) to comunicate with User FPGA on SpaceWire Digital IO board. The structure of User FPGA is based on the DAQ framework, which

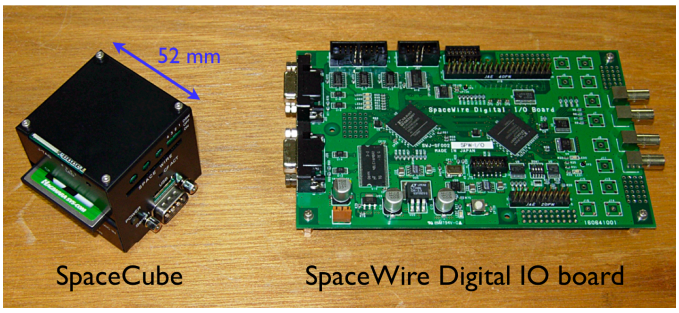


Fig. 2 SpaceCube and SpaceWire Digital IO board

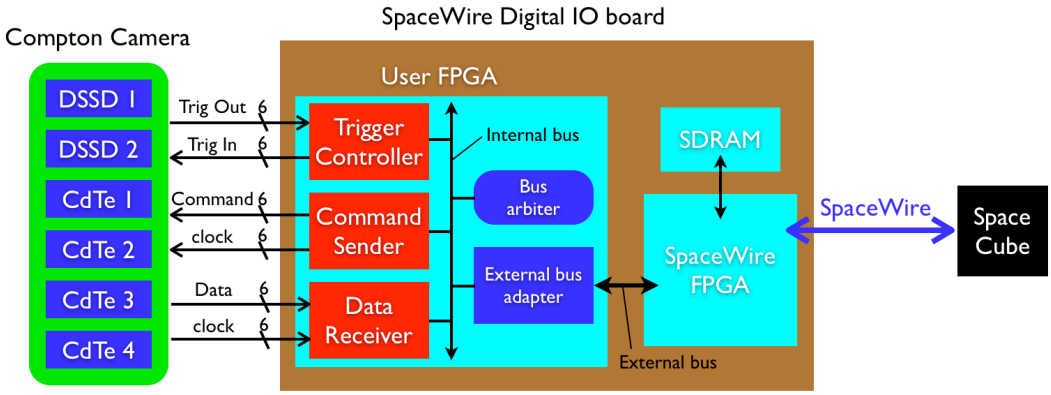


Fig. 3 A block diagram of the DAQ system of the Compton camera. While the DAQ framework offers the internal bus system and the external bus adapter in User FPGA, red modules in User FPGA are user-dependent.

provides modularized internal bus system including its arbiter and the external bus adapter to SpaceWire FPGA.

In User FPGA three user-dependent modules which connect to the internal bus are implemented for data readout from the Compton camera: Trigger Controller, Command Sender, and Data Receiver. The individual detector module composing the Compton camera digitalize signals from the semiconductor detectors by an analog-digital converter (ADC) and outputs serialized bit array via *Data* and *Data-clock* lines. The serialized data consists of the header part (8 bytes) and the data part. The data part structure is array of "4-bit flags + 12-bit ADC value" and its length is equal to the number of channels. The total size of one event data from all the detector modules is 5168 bytes. The data are received by Data Receiver and transfered through the internal bus and external bus to the SDRAM. Since photon-detection events occurs randomly and simultaneous readout from all the detector modules are required, Trigger Controller controls timing of readout via *TrigOut* and *TrigIn* lines. Command Sender sends various commands to the detector modules via *Command* and *Command-clock* lines.

From SpaceCube all modules in User FPGA and the SDRAM can be accessed by RMAP. All operations of the system, for example reading data from the SDRAM, sending commands to the detector, or setting registers of modules in the FPGA, are implemented by using RMAP Read/Write. SpaceCube program which operates the

Compton camera is written in C++ language and uses RMAP library of the DAQ framework.

We expect that the average event rate is less than 100 events per second. Thus required data transfer rate is more than 4 Mbps. Now our system in which SpaceWire IP core beta version is implemented have achieved the data transfer rate of 0.6 Mbps. In order to increase the data rate, we are developing new SpaceWire IP core which transfer data to RAM in SpaceCube by using direct memory access (DMA).

3 HEFT BALLOON-BORNE EXPERIMENT

We will perform a balloon-borne experiment with the Compton camera at Australia in 2008 summer. The mission named High Energy Focusing Telescope (HEFT) by California Institute of Technology, Columbia University and ISAS/JAXA is aimed at hard X-ray and soft gamma-ray imaging spectroscopy of high energy universe at 20 - 80 keV. The balloon carries three hard X-ray telescopes; our Compton camera (ISAS detector) is arranged at one of their focal planes.

Figure 4 is a block diagram of the data processing system for the HEFT experiment. The system has three SpaceWire I/F boards connected to SpaceCube via SpaceWire: SpaceWire Digital IO board for data acquisition of the Compton camera, SpaceWire FADC board, which has an ADC and digital I/Os, to read temperature and pressure sensors and to control high-voltage biases of the detectors, SpaceWire Digital IO board for communications (telemetry/command) with the central computer of the balloon system developed by Caltech.

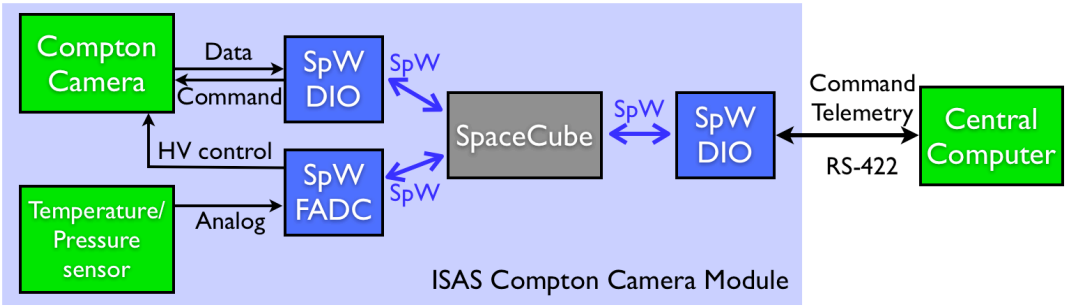


Fig. 4 A block diagram of the HEFT data processing system. There are three SpaceWire I/F boards.

4 REFERENCES

1. S. Takeda et al., “A new Si/CdTe semiconductor Compton camera developed for high-angular resolution” in “Hard X-Ray and Gamma-Ray Detector Physics IX”, Proceeding of SPIE, Vol. 6706, 2007.

2. T. Yuasa et al., “Development of a SpaceWire/RMAP-based Data Acquisition Framework for Scientific Detector Applications”, this volume.

THE SPACEWIRE INTERFACE FOR HERSCHEL/SCORE SUBORBITAL MISSION

Session: SpaceWire Missions & Applications

Short Paper

M.Pancrazzi[1]; A.Gherardi[1]; M.Focardi [1]; G.Rossi[1]; E.Pace[1]; M.Romoli[1];

1)Dipartimento di Astronomia e Scienza dello Spazio, Università di Firenze,

largo E.Fermi, 2, 50125, Firenze, Italy

E.Antonucci[2].

2) INAF – TO Osservatorio Astronomico di Torino, Via Osservatorio 20, 10025 – Pino Torinese, Torino, Italy

E-mail: panc@arcetri.astro.it, gherardi@arcetri.astro.it , mauro@arcetri.astro.it , gugpilot@arcetri.astro.it , pace@arcetri.astro.it , romoli@arcetri.astro.it , antonucci@to.astro.it

ABSTRACT

The HERSCHEL (HElium Resonant Scattering in the Corona and HELiosphere) experiment is a suborbital mission that will observe the Sun and the solar corona in the EUV and in visible light. One of the HERSCHEL instrument is the coronagraph SCORE (Sounding CORona Experiment) aimed at providing images of the extended solar corona in the EUV lines HI 121.6nm and HeII 30.4nm and in the broadband linearly polarized visible light. The two SCORE cameras are developed at the XUVLab of the Department of Astronomy and Space Science of Florence University, in collaboration with the Naval Research Laboratory, Washington, DC. The SpaceWire ESA standard protocol has been selected as the rocket interface. Since HERSCHEL is a suborbital mission and its operation time will be only about 300s, the SCORE SpaceWire interfaces have a customized design to perform specific and automatic procedures. The prototypes and the flight models of the SCORE cameras and the SpaceWire interface have been developed and tested.

1. THE HERSCHEL MISSION

HERSCHEL is conceived as a NASA Sounding Rocket Program providing new EUV/UV (H Lyα and HeII Lyα) and visible-light observations of the solar disk and corona [1].

HERSCHEL will investigate coronal heating and solar wind acceleration from a range of solar source structures by obtaining simultaneous observations of the electrons, protons and helium abundances in the solar corona.

The mission aims at developing instrumentation that for the first time will directly image and characterize on a global coronal scale the two most abundant elements: hydrogen and helium. This will directly address three outstanding questions in the Sun-Earth Connection theme:

1. Origin of the slow solar wind,
2. Acceleration mechanisms of the fast solar wind, and
3. Variation of Helium abundance in coronal structures.

Additionally, HERSCHEL will establish a proof-of-principle for the SCORE Coronagraph, which is in the ESA Solar Orbiter Mission baseline making easier future investigations of CME's kinematics, and solar cycle evolution of the coronal plasma.
The HERSCHEL instrument package consists of the HERSCHEL Extreme Ultraviolet Imaging Telescope (HEIT) for on-disk coronal observations and two coronagraphs, SCORE and HECOR (HERSCHEL EUV Coronagraph) for off-limb observations of the corona. The monochromatic H I (121.6 nm) and He II (30.4 nm) images obtained by the coronagraphs will provide simultaneous abundance, densities and velocities diagnostics from 1 R⊙ to 3 R⊙. HEIT will obtain the intensity of the resonantly scattered He II line below the eld of view of the coronagraphs, as well as the He II disk images necessary for the analysis of the coronal emission.

The HERSCHEL instruments will be launched at the end of 2007 with a Terrier-Brant booster drawing a ballistic trajectory with an apogee of 335 km. This altitude is needed to avoid the atmospheric absorption of coronal radiation; because of the ballistic trajectory, HERSCHEL will observe only for ~300 sec, when the rocket will be beyond 250 km of altitude. Due to the short duration of the flight and the relatively small reached altitude, all HERSCHEL instruments must be vibration and vacuum resistant but they do not require a space qualification.

2. THE SCORE INSTRUMENT

The SCORE coronagraph consists of an externally occulted, off-axis Gregorian telescope with multilayer coated optics [1]. The coronagraph has a visible (VLD) and an UV (UVD) detectors and it is capable of simultaneous images acquisition in the visible light (VL) and in two UV narrowbands. A filter mechanism (FM) swaps a Aluminium and an Al/MgF_2 filter to enable the acquisition of the only HeII Lyα line or of HI Lyα line and visible light images. The two independent SCORE detectors will acquire images simultaneously with the Al/MgF_2 filter.

The selected detectors for SCORE coronagraph are a CCD camera for the visible channel and an Intensified CCD (ICCD, a CCD coupled with a MCP detector) for UV. The visible detector is an E2V CCD47-20 1024x1024, frame transfer, operating in 2x2 binning mode in order to enhance the weak coronal signal. It has a 16-bit dynamic range and produces 4Mb images. The UV CCD is the E2V CCD42-40. The UVD camera will acquire both He and H images, by selecting different filter configurations in the telescope. The images acquired by both CCDs are readout at 300 kpx/s [2], [3]. Both cameras are developed by XUVLab with the collaboration of the Naval Research Laboratory (NRL).

The SCORE visible channel includes a polarimetric group to measure the polarized brightness of K-corona [4]. The polarimeter works with an innovative variable retarder plate that uses liquid crystals to select the proper polarization. Modulating the voltage at the input of the retarder plate it is possible to control the alignment of liquid crystal and so to set the proper polarization [5]. SCORE will be the first instrument to use a LCVR (Liquid Crystal Variable Retarder) polarimeter for space applications.

3. THE SPACEWIRE INTERFACE

During the flight, a single board computer will manage the entire scientific payload and the acquisition sequences of each camera; moreover it will work to save and store data and to send them to the ground station through telemetry.

To prevent an overwork of the computer, each camera must reduce the computer control and must keep busy the downlink as short time as possible. Due to the short mission duration the communication protocol must ensure an high data rate avoiding time losses for the images download and maximizing the observation time. For these reasons we chose SpaceWire as the SCORE cameras communication interface [6]. Moreover this standard protocol is reliable, low power, and space compliant. This standard has been chosen not only for the SCORE cameras but it has been adopted as the protocol of all on-board instruments. Both the SCORE cameras are provided with a SpaceWire (SpW) interface developed in our laboratories. Actually the on-board computer implements the revised IEEE-1355 [8] DS/DE sub-standard protocol with LVDS signalling that is however SpaceWire compatible. We chose to implement the same protocol on the SCORE cameras using the SMCSlite device (Atmel T7906E).

In order to minimize the download time of the images toward the computer, the communication interface has two FIFO memories which work together, in order to store a whole image (4 Mb). Every acquired image is readout and stored into the FIFOs, in ~1.7 s (VLD); then the image is downloaded to the computer through the Spacewire link as soon as the computer is ready to received it; then the image is stored in a mass memory and transmitted to the ground station.

A CPLD (Complex Programmable Logic Device) manages the FIFOs writing procedure and it produces all the initialization signals. The reading procedure from the FIFOs is managed by the SMCSlite afterwards a CPLD enabling signal. The SMCSlite transmits images properly coded via the IEEE-1355 link directly to the rocket computer. Each image is packed with some auxiliary informations, placed in the header, as acquisition number, exposure time and housekeeping. The SpW interfaces and the CCD cameras must have an high level of automation: since in the brief observational time is not possible to send commands from ground station, the electronics must recognize errors and must recover them. To realize this smart management we introduced in the interface design a microcontroller as host controller. This device controls the entire acquisition and sets the configuration of the SMCSlite and the CPLD. The host controller manages the communication with the other camera boards through 3 UART links Through this connections it checks that each camera board work properly

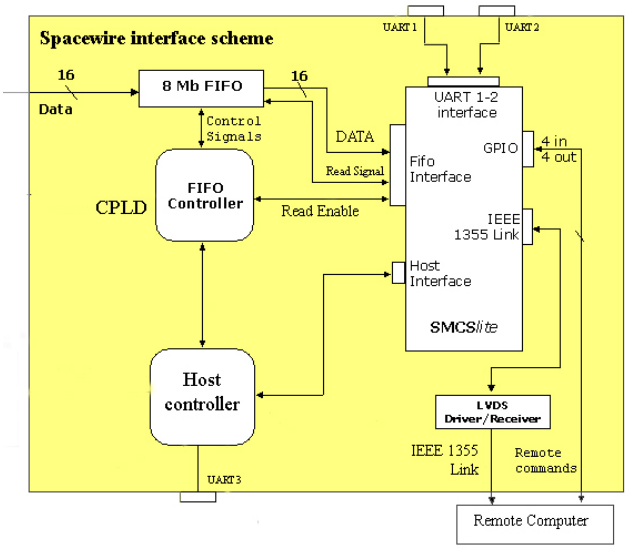


Fig 1: the Spacewire interface scheme showing the interface structure, the main devices and connections.

and it accomplishes housekeeping for the detector and (only for VLD) for the retarder plate writing these data in the header of every image. Therefore the microcontroller is a sort of CCD camera "brain". It accomplishes again the communication with the on-board computer exchanging commands and acknowledgements through the SpaceWire link or over a dedicated (8 bits) parallel link(Rem_Cmd link). Note that the possibility to send and receive data over different links increases the interface reliability. The microcontroller firmware implements also check and recover procedures: if a camera error occurs, the microcontroller points out the error to the on-board computer requiring a camera reset.

Although our SpaceWire interface has been developed for a custom application, its design is quite versatile. In fact the several procedures and functionalities that it offers, are all realized by means of software implementation The choice of reprogrammable devices enables to fit the performances of the interface to a specific applications easily changing the microcontroller firmware. The SCORE SpaceWire interface have also a flexible hardware; for example, it is able to write and store data in the internal FIFOs with an input rate ranging from 0 to 1 MHz without any change. Main SCORE interface features and performances are summarized below:

- 16 bits input data bus with a writing rate from 0 to 1 MHz.
- 8Mb Storing capacity
- 1 SpaceWire link (IEEE-1355 DS/DE) with a maximum data rate of 130Mb/s currently
- 1 general purpose 8 bits parallel link
- 3 UARTs
- Customizable procedures

This versatility will permit us to reuse our SpaceWire interface for different applications and to fit or upgrade it to future instrumentations.

REFERENCES:

[1]M.Romoli, E.Antonucci et al "*The Ultraviolet and Visible-light Coronagraph of the HERSCHEL experiment* "AIP Conf. Proceedings, Vol 679, pp. 846-849 (2003)
[2] A.Gherardi, L.Gori, "SCORE CCD camera: Overall Description", Technical Report, 2004.
[3] A. Gherardi "Elettronica di controllo per la camera CCD del canale polarimentrico di HERSCHEL, ", Degree Thesis, Univ. di Firenze, 2002.
[4] S.Fineschi et al, *"KPol: liquid crystal polarimeter for K-corona observations from the SCORE coronagraph"* Proceedings of the SPIE, Vol 5901, pp. 389-399 (2005).
[5] G. Rossi, " Realizzazione dell'elettronica di controllo per il polarimetro di HERSCHEL/ SCORE". ", Degree Thesis, Univ. di Firenze, 2005.
[6] M.Pancrazzi, "Realizzazione del sistema di comunicazione Spacewire per la missione SCORE", Degree Thesis, Univ. di Firenze, 2006.
[7] S.M. Parkes et al, "SpaceWire: Links, Nodes, Routers and Networks,"*European Cooperation for Space Standardization, Standard N. ECSSE50-12A,*Issue 1,Jan 2003.
[8] IEEE Computer Society, "IEEE Standard for Heterogeneous Interconnect (HIC) (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction)", *IEEE Standard 1355-1995,* IEEE, June 1996.

INTRODUCTION OF SPACEWIRE APPLICATIONS FOR THE MMO SPACECRAFT IN BEPICOLOMBO MISSION

Session: SpaceWire Missions and Applications

Short Paper

T. Takashima, H. Hayakawa and H. Ogawa

ISAS/JAXA 3-1-1 Yoshinoda, Sagamihara, Kanagawa 229-8510 Japan

Y. Kasaba

Tohoku Univ. 6-3, Aobadai, Aobaku, Sendai, Miyagi 980-8578 Japan

M. Koyama, K. Masukawa, M. Kawasaki, S. Ishii and Y. Kuroda

Nagoya Guidance and Propulsion Systems Works, MITSUBISHI Heavy Industries LTD., 1200 Higashi-Tanaka, Komaki, Aichi 485-8561

BepiColombo MMO Project Data-Handling team

NEC TOSHIBA Space Systems, Ltd., 10, Nisshin-cho 1-chome, Fuchu, Tokyo, Japan

NEC Soft, Ltd., 2-22 Wakaba-cho, Kashiwazaki, Niigata, Japan

e-SOL Co.Ltd., Harmony Tower 22,23,24F, 1-32-2 Honcho, Nakano-ku, Tokyo 164-8721

E-mail: ttakeshi@isas.jaxa.jp, hayakawa@isas.jaxa.jp, ogawa@isas.jaxa.jp, kasaba@pat.geophys.tohoku.ac.jp

ABSTRACT

The Mercury exploration BepiColombo mission is the joint planning of ESA/JAXA. The Mercury Magnetospheric Orbiter (MMO) is mostly dedicated to the first detailed study of magnetic field, waves, and particle environment of the planet Mercury. The MMO satellite is manufactured by Japan. They are expected to significantly advance comparative studies of the magnetic fields and magnetospheres of terrestrial planets: 1) Structure and origin of Mercury's magnetic field, 2)Structure, dynamics, and physical processes in Mercury's magnetosphere,3) Structure, variation, and origin of Mercury's exosphere, and 4) The inner solar system. The MMO payload selected by JAXA in 2005 consists of 5 instruments / instrument packages, wide-range observational capabilities for charged particles and energetic neutral atoms, magnetic field, electric field / plasma waves / radio waves, dust, and exospheric constituents. Those scientific payload groups are under unified and coordinated controls of the observational mode and time resolution by MDP (Mission Data Processor) provided

by JAXA and MHI, in order to fulfill the science objectives of this mission. These interfaces which control these observation equipments are defined by the SpaceWire with Remote Memory Access Protocol (RMAP).

It does introduction and a report about the SpaceWire with RMAP to have applied to both the observation equipment system and the bus systems of MMO.

.

DEVELOPMENT OF A SPACEWIRE/RMAP-BASED DATA ACQUISITION FRAMEWORK FOR SCIENTIFIC DETECTOR APPLICATIONS

Session: Missions & Applications

Short Paper

Takayuki Yuasa, Kazuhiro Nakazawa, Kazuo Makishima,
The University of Tokyo, 7-3-1 Hongo, Bunkyo, Tokyo, Japan

Hirokazu Odaka, Motohide Kokubun, Takeshi Takashima, Tadayuki Takahashi
Department of High Energy Astrophysics, Institute of Space and Astronautical Science (ISAS), Japan Aerospace Exploration Agency (JAXA), 3-1-1 Yoshinodai, Sagamihara, Kanagawa 229-8510, Japan

Masaharu Nomachi,
Laboratory of Nuclear Studies, Graduate School of Science, Osaka University, 1-1 Machikaneyama, Toyonaka, Osaka 560-0043

Iwao Fujishiro, and Fumio Hodoshima
Shimafuji Electric Incorporated, 8-1-15 Nishikamata, Ota, Tokyo, Japan 144-0051

E-mail: yuasa@amalthea.phys.s.u-tokyo.ac.jp, nakazawa@phys.s.u-tokyo.ac.jp, maxima@phys.s.u-tokyo.ac.jp, odaka@astro.isas.jaxa.jp, kokubun@astro.isas.jaxa.jp, ttakeshi@stp.isas.jaxa.jp, takahasi@astro.isas.jaxa.jp, nomachi@fn.lns.sci.osaka-u.ac.jp,fujishiro@shimafuji.co.jp, hodo@shimafuji.co.jp

ABSTRACT

We have been developing a multipurpose modularized data acquisition framework based on SpaceWire and Remote Memory Access Protocol (RMAP). The framework consists of a SpaceWire/RMAP library that runs on a small size computer SpaceCube, and a template hardware description language code for an FPGA which is used on several kinds of circuit boards with SpaceWire protocol stack. Users need to implement a VHDL module which receives data from their detectors and sends commands to them. Data transfer is done by accessing user defined registers or memory blocks in the FPGA from the SpaceCube user program. In this paper, we present an overview of this framework, and describe an example of its implementation in our gamma ray imager experiment is also presented.

1 A DATA ACQUISITION FRAMEWORK ON SPACEWIRE/RMAP

Remote Memory Access Protocol (RMAP) provides standard read/write methods over SpaceWire connection. We have been developing a data acquisition (DAQ) framework for scientific detectors based on SpaceWire and RMAP. As shown in figure 1, this DAQ framework consists of a small size computer called SpaceCube, a circuit board, and software libraries for them. Below are detailed explanations of SpaceCube, circuit boards, and the prepared framework.

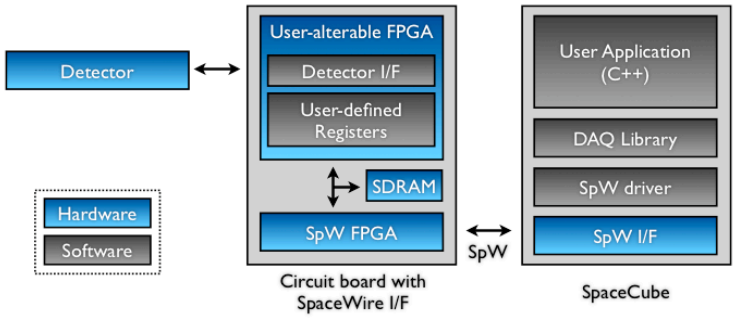


Figure 1 The block diagram of the DAQ framework. Users have to code a detector I/F module and user-defined registers in VHDL, and construct a user application in C++.

1.1 SpaceCube

SpaceCube is a small size (52x52x55 mm) computer with SpaceWire I/Fs developed by Shimafuji Electric, Inc. and JAXA. Figure 2 shows its appearance, and its major characteristics are listed in table 1. SpaceCube is operated by The Real-time OS Nucleus (TRON) which is widely employed as an operating system for embedded systems. In this DAQ framework, a user application which runs on SpaceCube works as a data/command handler and recorder. It fetches data and sends commands from/to a user-alterable FPGA via RMAP. SpaceWire protocol stack (an IP core and an I/F driver) is provided by NEC Software, Ltd. Our RMAP library is written in C++ and provides users with simple operations, such as *read(address)* and *write(address, data),* to access their boards.

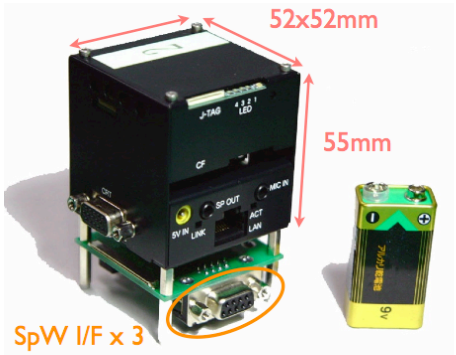


Figure 2 An appearance of SpaceCube. Right is a 006P type battery for size comparison.

Table 1 Specification of SpaceCube.

CPU	VR5701(MIPS core) 200, 250, 300 MHz
RAM	DDR SDRAM 64 MB
Flash Memory	16MB
I/F	SpaceWire x 3, CF, XGA, USB, LAN, Stereo Audio, RS232C
Power	+5 V
Size	52x52x55 mm

1.2 Circuit Boards with SpaceWire I/F and the Internal Structure of User-alterable FPGA

We have been developing a several kinds of circuit boards with SpaceWire I/F, for such purpose as analogue-digital conversion, digital-analogue conversion, and digital input/output. Each circuit board is equipped with a SpaceWire/RMAP protocol stack FPGA (hereafter, SpaceWire FPGA), a user-alterable FPGA, SDRAM, and circuit for independent functionality. Figure 3 is an appearance of an example of digital input/output board. Between the two FPGAs, a simple data bus (hereafter we call External Bus) is defined. An RMAP engine, implemented in SpaceWire FPGA, interprets an RMAP command packet, and then access the user-alterable FPGA's address space via External Bus if the requested address is of that FPGA. The SDRAM, which is connected to SpaceWire FPGA, is also accessible from SpaceCube and the user-alterable FPGA via SpaceWire and External Bus, respectively.

As shown in figure 4, a user-alterable FPGA consists of function modules connected to an on-chip bus. The on-chip bus provides a standard way for data transfer among modules, and enables SpaceWire FPGA to access each modules' address space via External Bus and on-chip bus (figure 4). In VHDL, we have written several template functional blocks for

the user-alterable FPGA, such as an on-chip bus arbiter, a bus interface module, a data-buffering module, and an External Bus adapter module. Users can construct their own read-out chip by connecting those modules.

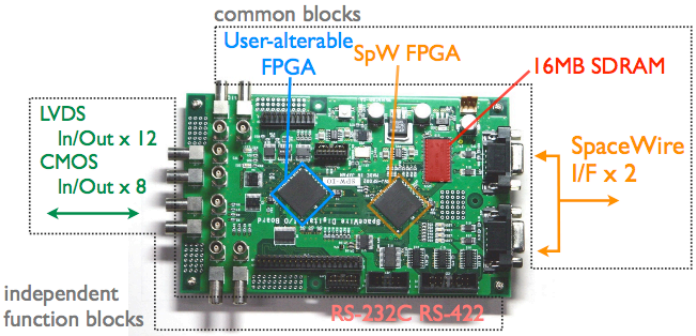


Figure 3 An appearance of a digital input/output board. Two FPGAs (Xilinx Spartan-3, XC3S1000 for SpaceWire FPGA and XC3S400 for the user-alterable FPGA) are used to separate commonly needed SpaceWire/RMAP functionality and user dependent blocks.

The whole framework structure is modularized in both software and hardware. Especially, in the user-alterable FPGA, all functional blocks, including a detector I/F module which should be coded by each user, only communicate with on-chip bus I/F modules as shown in figure 4. Therefore the portability of user-coded modules is reasonably high. This allows users to use the same modules both in their bread board models and in the flight model, and hence to reduce costs for the entire projects.

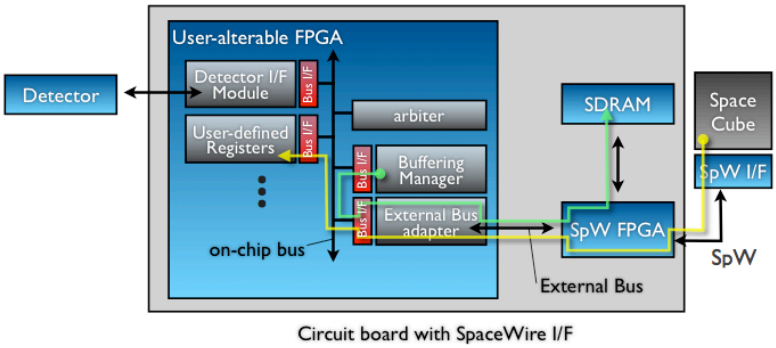


Figure 4 The internal structure of a user-alterable FPGA. For instances, yellow and green arrows represent routes of access from SpaceCube to user-defined registers in the user-alterable FPGA, and from Buffering Manager in the FPGA to the SDRAM, respectively.

2 An Example of Implementation of the Framework

To verify the performance of the developed framework, we experimentally employed it as a read-out system for a gamma ray imaging detector, which has been developed by our group [1]. The detector consists of pixelated inorganic scintillator array (4x4x10mm/pixel, 10x10=100pixels), viewed by a 256 channel multi-anode photo multiplier tube (PMT) and a 12 bit ADC unit. The AD converted signal is received by a digital input/output board, then read out by SpaceCube. A block diagram and a total picture of the system are shown in figure 5 and 6, respectively.

When a gamma ray photon is absorbed in one of the scintillation crystals, produces charge pulse in some or all of the 256 anode channel of the PMT. The analogue signals from the 256 channel PMT anodes are converted to serial digital signals in the 12 bit ADC unit. In the user-alterable FPGA, we have developed a particular detector I/F module for the ADC unit (De-serializer in figure 5). It de-serializes the data transferred from the ADC unit, and stores them into the SDRAM across the on-chip bus , an SDRAM buffering manager, and External Bus. The size of an event is

(ADC 12 bit + Header 4 bit) * 256 ch + Time (32 bit) = 4128 bit = 516 bytes.

The stored data are transferred from the SDRAM to SpaceCube via SpaceWire, and then sent to recorder PC as TCP/IP packet. Spectral and image analyses are done on an offline environment with the ROOT system. From the 256 pulse-height data, we can reconstruct the gamma-ray energy, and determine the interaction position as their centers of gravity.

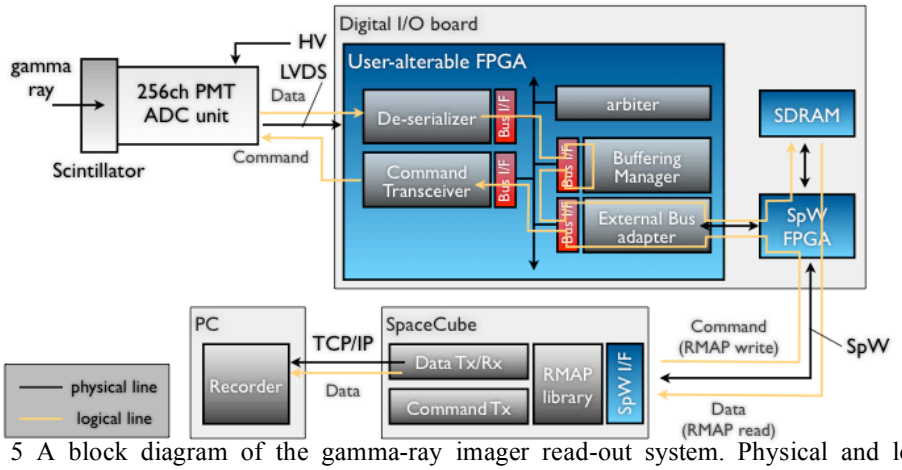


Figure 5 A block diagram of the gamma-ray imager read-out system. Physical and logical connections between blocks are shown in black and orange arrows, respectively.

We irradiated gamma rays for radioisotope ^{137}Cs to the whole area of the imager. The DAQ framework worked well and gamma ray event data were successfully transferred via SpaceWire/RMAP. Configurable parameters of the ADC unit (trigger mode, trigger threshold, peaking time, etc) were also properly controlled according to change commands from SpaceCube. Figure 7 shows a reconstructed image of 500,000 gamma ray events (~250 MBytes). In the image, each bright point corresponds to single scintillator pixel. All of 100 pixels, each 4x4 mm^2, have been successfully resolved.

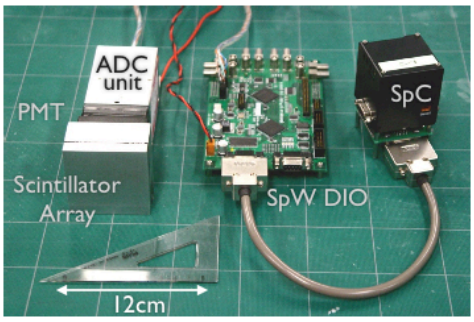


Figure 6 A photograph of our gamma-ray imager (left), digital input/output board (center),, and SpaceCube (right).

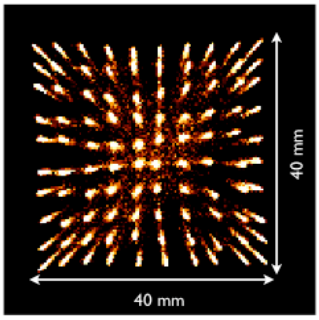


Figure 7 A reconstructed image of 5x10^5 gamma ray events from ^{137}Cs.

3 THE FRAMEWORK IN OTHER EXPERIMENTS

This DAQ framework based on SpaceWire and SpaceCube is planned to be used in such science missions as,

PoGO : Balloon-borne gamma-ray polarimeter experiment by SLAC, Hiroshima U., & JAXA
HEFT : Balloon-borne hard X-ray imaging experiment by JAXA, Cal Tech et al.
SDS-I SWIM : SpaceWire I/F verification Module on a Japanese small satellite by JAXA& UT
NeXT : Next Japanese cosmic X-ray Satellite by JAXA et al.
TEM : Transmission Electron Microscope with X-ray micro calorimeter by JAXA et al.

[1]. M. Kokubun and S. Hirakuri et al., "Development of an Active Gamma-ray Imaging Spectrometer with Pixelated Scintillators", IEEE NSS-MIC 2006, October 29 - Nov. 4, 2006, at San Diego

Components 2

Wednesday 19th September

14:40 – 18:35

ATMEL SPACEWIRE PRODUCTS FAMILY

Session: Components

Short Paper

Nicolas RENAUD, Yohann BRICARD

ATMEL Nantes – La Chantrerie – 44306 NANTES Cedex 3

E-mail: nicolas.renaud@atmel.com, yohann.bricard@atmel.com

ABSTRACT

This paper presents the status of the SpaceWire products available at ATMEL for space applications. This includes the SpaceWire SMCS communication controller ASICs (three SpaceWire links or one SpaceWire link version), the SpaceWire Router ASIC and the SpaceWire Remote Terminal Controller (RTC). It gives a high-level overview of the product features, their main characteristics (packaging, operating ranges...) and availability dates for engineering and flight models. A focus is done on the technologies on which these products rely, as well as on the expected radiation capabilities (total dose and single event effects). The paper also describes the way to interface these SpaceWire chips with other ATMEL rad-hard devices, such as the AT697 Sparc V8 processor or the ATF280 SRAM based FPGA. The building of a complete SpaceWire network based on ATMEL rad-hard products is underlined.

1 ATMEL SPACEWIRE PRODUCTS FAMILY OVERVIEW

The ATMEL portfolio of SpaceWire devices includes :

- the AT7910E, or SpaceWire Router
- the AT7911E, or SMCS332SpW
- the AT7912E, or SMCS116SpW
- the AT7913E, or SpaceWire RTC (Remote Terminal Controller).

An overview of the functionalities and the main characteristics of these chips are provided in the rest of this first section.

1.1 STANDARD ASICS

ATMEL is regularly working with most of its key space customers to make their ASIC designs available as standard ASICs when they correspond to commonly used functions on the market. The four SpaceWire chips described in this paper meet this requirement. The SMCS-SpW devices have been designed by EADS Astrium in Germany. The SpaceWire router is designed by Austrian Aerospace in Austria and the University of Dundee in Scotland. The SpaceWire RTC is designed by Saab Space in Sweden. All these designs are made under ESA contracts.

1.2 SMCS-SPACEWIRE CHIPS

The **AT7911E**, or **SMCS332SpW** for "Scalable Multi-channel Communication Subsystem for SpaceWire", provides an interface between three SpaceWire links compliant with the SpaceWire standard ECSS-E-50-12A specification and a data processing node consisting of a Control Processing Unit and a communication data memory. The AT7911E can connect modules with different processors (e.g. TSC695F or AT697E) and modules without any communication features such as special image compression chips or mass memory. The AT7911E may also be used in systems containing "non-intelligent" modules such as A/D-converter or sensor interfaces. More details on the AT7911E features can be found in EADS Astrium 'SMCS332SpW User Manual' [1], available on the ATMEL web site together with the AT7911E datasheet [2]. The AT7911E is packaged in a MQFPL 196 pins and can operate in both 5V or 3.3V voltage ranges. The engineering samples are available and flight models can be ordered.

The **AT7912E**, or **SMCS116SpW**, provides an interface between a SpaceWire link compliant with the SpaceWire Standard ECSS-E-50-12A specification and several different interfaces such as ADC/DAC, RAM, FIFO, GPIOs and UARTs. It supports both the standard SpaceWire link protocol (transparent mode) and the STUP (Serial Transfer Universal Protocol) for efficient packet oriented data transfer. More details on the AT7912E features can be found in EADS Astrium 'SMCS116SpW User Manual' [3], available on the ATMEL web site together with the ATMEL AT7912E datasheet [4]. The AT7912E is packaged in a MQFPF 100 pins and can operate in both 5V or 3.3V voltage ranges. The engineering samples are available and flight models can be ordered.

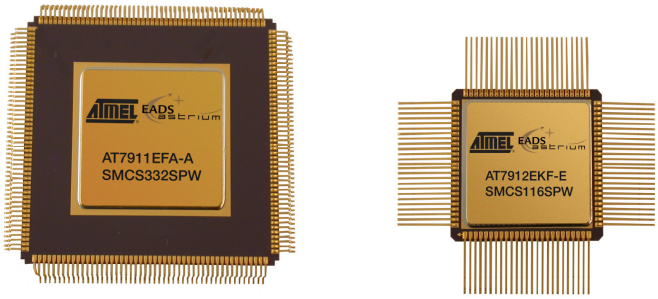


Figure 1 – SMCS-SpW chips

1.3 SPACEWIRE ROUTER

The **AT7910E SpaceWire Router** provides eight SpaceWire ports, two external parallel ports and an internal configuration port. It can be used as a standalone router or as a node interface using the external parallel ports. For more details on the AT7910E functionalities, it should be referred to the University of Dundee 'SpaceWire Router datasheet' [5]. The AT7910E is packaged in a MQFPF 196 pins and operates in 3.3V voltage range. The ATMEL AT7910E datasheet will be available in Q4 2007 on the ATMEL web site. The engineering samples will be available in Q1 2008. The order entry for flight models will open in Q1 2008.

1.4 SPACEWIRE REMOTE TERMINAL CONTROLLER

The **AT7913E SpaceWire Remote Terminal Controller** provides a bridge between a SpaceWire network and a CAN bus, and includes a LEON2-FT processor with additional interfaces to ADC/DAC, RAM, FIFO, GPIOs and UARTs. This chip allows the

interfacing of high speed serial SpaceWire network and low-speed spacecraft control bus based on CAN. The possible use of the embedded LEON2-FT processor also allows the SpaceWire RTC to contribute to instrument controller processing tasks. For more details on the AT7913E functionalities, it should be referred to the Saab Space 'SpaceWire RTC datasheet' [6] that will be publicly available in Q4 2007. The AT7913E is packaged in a MCGA 349 pins and operates in 3.3V voltage range for the I/Os, and 1.8V voltage range for the core. The ATMEL AT7913E datasheet will be available in Q4 2007 on the ATMEL web site. The engineering samples will be available in Q1 2008. The order entry for flight models will open in Q2 2008.

2 GENERAL INFORMATION ON TECHNOLOGY, QUALITY, RADIATION AND TOOLS

2.1 TECHNOLOGIES AND ASSOCIATED QUALITY ASPECTS

The above mentioned four SpaceWire chips rely on well proven ATMEL ASIC families for space :

- the AT7911E and AT7912E (SMCS-SpW) are based on the MG2RT radiation tolerant 0.5 µm CMOS sea of gates ASIC family,
- the AT7910E (SpaceWire Router) is based on the MH1RT rad-hard 0.35 µm CMOS sea of gates ASIC family
- the AT7913E (SpaceWire RTC) is based on the ATC18RHA rad-hard 0.18 µm CMOS cell-based ASIC family.

Space production screenings and qualification are compliant either with ESCC 9000 or MIL-PRF-38535. Atmel is fully DSCC QML qualified for level Q (military) and V (space). Qualification is granted for all the above mentioned technologies. Atmel is ESCC qualified for MH1RT ASIC family according to ESCC 2549000 (ESCC QML) and plan to get by the end of 2007 the ESCC certification and qualification process for the ATC18RHA ASIC family. Detailed space quality flows can be found in [7]. The four SpaceWire chips are or will be available in QML-Q and QML-V quality flows with SMD.

2.2 RADIATION

The radiation test results obtained on the MG2RT, MH1RT and ATC18RHA ASIC families are applicable to the SpaceWire chips. As a consequence :

- the AT7911E and AT7912E can be tested successfully up to a total dose of 50 Krad(Si), according to MIL STD 1019 method,
- the AT7910E and AT7913E can be tested successfully up to a total dose of 300 Krad(Si), according to MIL STD 1019 method.
- there is no Single Event Latchup at a LET of 70 MeV/mg/cm^2.
- the four chips use the SEU hardened flip-flops of the associated library for all critical bits, thus the SEU error rate in space is very low.

Radiation test reports for the associated technologies and/or for the above SpaceWire chips are available and can be obtained from ATMEL on demand.

2.3 TECHNICAL SUPPORT AND TOOLS

Technical support can be obtained by contacting the following ATMEL hotline :
assp-applab.hotline@nto.atmel.com

A number of development boards, software tools and test equipments to facilitate the design using one of the above mentioned SpaceWire chips are available, for example at Star-Dundee, 4Links, Aurelia Microelectronica or Gaisler Research.

3 BUILDING A SPACEWIRE NETWORK BASED ON ATMEL RAD-HARD PRODUCTS

3.1 OVERVIEW OF A COMPLETE SPACEWIRE NETWORK BASED ON ATMEL PRODUCTS

The four above mentioned chips, associated together and complemented by other ATMEL rad-hard products such as processors, memories, ASICs or FPGAs, allow the building of a complete and cost-effective system using a SpaceWire network based on rad-hard products.

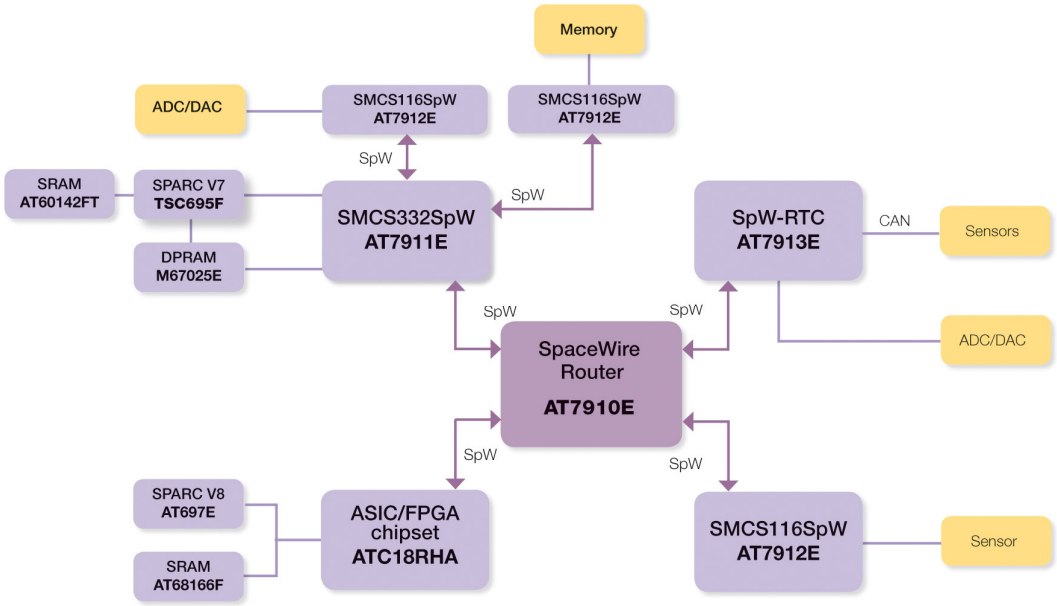


Figure 2 – Complete SpaceWire network based on ATMEL rad-hard products

3.2 INTERFACING ATMEL SPARC RAD-HARD PROCESSORS IN A SPACEWIRE NETWORK

The TSC695 Sparc V7 processor and the AT697 Sparc V8 processor families do not embed the SpaceWire interface capability.

Processor architectures requiring a single link to SpaceWire network can integrate the AT7912E single SpaceWire link high speed controller together with its communication DPRAM in order to provide a direct access from the processor to the SpaceWire network. Direct connections of the processor to both the control interface and the communication interface of the SpaceWire controller are sufficient to provide a SpaceWire communication front-end to the processor, as shown in the figure 3.

For processor architecture requiring more than a single SpaceWire link to communicate on the SpaceWire network, up to 3 SpaceWire links can be integrated in the architecture using the AT7911E triple Spacewire links high speed controller. External LVDS drivers should be used as the AT7911E and the AT7912E do not implement them.

Interfacing between the Sparc processor and the SpaceWire network can also be done by the use of a dedicated ASIC or FPGA chipset.

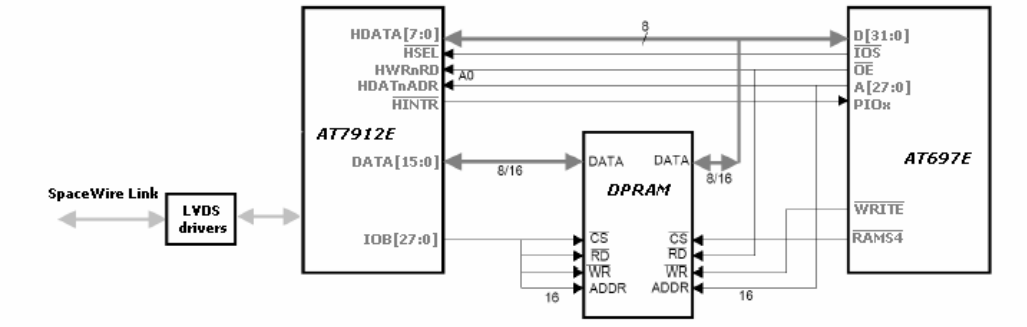


Figure 3 : SpaceWire Front End addition to AT697E processor principle

3.3 INTERFACING ATMEL RAD-HARD FPGAS WITH A SPACEWIRE NETWORK

The ATF280E FPGA integrates 280K equivalent ASIC gates that make it possible to build a SpaceWire communication interface. Integration of a SpaceWire IP on the FPGA provides a direct access to the SpaceWire network without any need for external LVDS drivers as the FPGA implements them on-chip. The same applies to MH1RT or ATC18RHA based ASIC designs when they embed the ESA SpaceWire IP.

4 CONCLUSION

The SpaceWire standard was developed by a european space industry working group sponsored by ESA and is now recognized by NASA. It ensures reliable and fast serial links for data handling between equipments and subsystems, allowing their standardization and re-use for several kinds of missions. ATMEL supports and promotes the SpaceWire standard through cooperations with ESA and space users. The availability of four new rad-hard SpaceWire devices, the AT7910E, AT7911E, AT7912E and AT7913E, designed by EADS Astrium, Austrian Aerospace and Saab Space will allow the space users to build complete and cost-effective systems using SpaceWire networks based on rad-hard devices.

5 REFERENCES

[1] EADS Astrium SMCS332SpW User Manual – SMCS_ASTD_UM_100 Issue 1.5 – 10/07/07

[2] ATMEL AT7911E datasheet – 7737A-AERO-07/07

[3] EADS Astrium SMCS116SpW User Manual – SMCS_ASTD_UM_116 Issue 1.0 – 10/07/07

[4] ATMEL AT7912E datasheet – 7743A-AERO-07/07

[5] University of Dundee SpaceWire Router datasheet – UOD_SPW_10X_Datasheet – Issue 2.0 – 18/08/2006

[6] Saab Space SpaceWire RTC datasheet – P-ASIC-NOT-00256-SE – Issue 6 – 29/05/2007 – Preliminary version

[7] ATMEL Aerospace products quality flows – 4288C – AERO- 11/05

[8] ATMEL web site – aerospace products : http://www.atmel.com/products/radhard/

[9] ESA SpaceWire web site : http://spacewire.esa.int

SPACEWIRE ROUTER ASIC

Session: SpaceWire Components

Short Paper

Steve Parkes [1], Chris McClements [1], Gerald Kempf [2], Stephan Fischer [3],
Pierre Fabry [4], Agustin Leon [4]

[1]*School of Computing, University of Dundee, Dundee, DD1 4HN, Scotland, UK.*

Email: sparkes@computing.dundee.ac.uk.

[2]*Austrian Aerospace*

[3]*Astrium GmbH*

[4]*European Space Agency*

ABSTRACT

A SpaceWire routing switch [1],[2] comprises a number of SpaceWire link interfaces and a routing matrix. The routing matrix enables packets arriving at one link interface to be sent out of another link interface on the routing switch depending on address information at the start of each packet. Thus, SpaceWire packets from one node can be routed, through the switch, to any other node connected to the routing switch.

This paper describes the SpW-10X SpaceWire Routing Switch ASIC designed by University of Dundee and Austrian Aerospace, validated by EADS Astrium GmbH and manufactured by Atmel.

SPW-10X ARCHITECTURE

The architecture of the SpW-10X SpaceWire Routing Switch is illustrated in Figure 1. It has the following main features:

- Eight SpaceWire ports.
- Two external parallel ports, each comprising an input FIFO and an output FIFO.
- A non-blocking crossbar switch connecting any input port to any output port.
- An internal configuration port accessible via the crossbar switch from the external parallel port or the SpaceWire ports.
- A routing table accessible via the configuration port which holds the logical address to output port mapping.
- Control logic to control the operation of the switch, performing arbitration and group adaptive routing.
- Control registers than can be written and read by the configuration port and which hold control information e.g. link operating speed.
- An external time-code interface comprising tick_in, tick_out and current tick count value.

- Watchdog timers on all ports.
- Internal status/error registers accessible via the configuration port using the RMAP protocol [2].
- External status/error signals.
- Implemented in 0.35um MH1RT technology from ATMEL (latch-up immunity guaranteed up to 80 MeV/mg/cm2).
- Prototypes will be in a 196 pin Metric Quad Flat Package (MQFP) whereas the flight models should be in a Ceramic package.

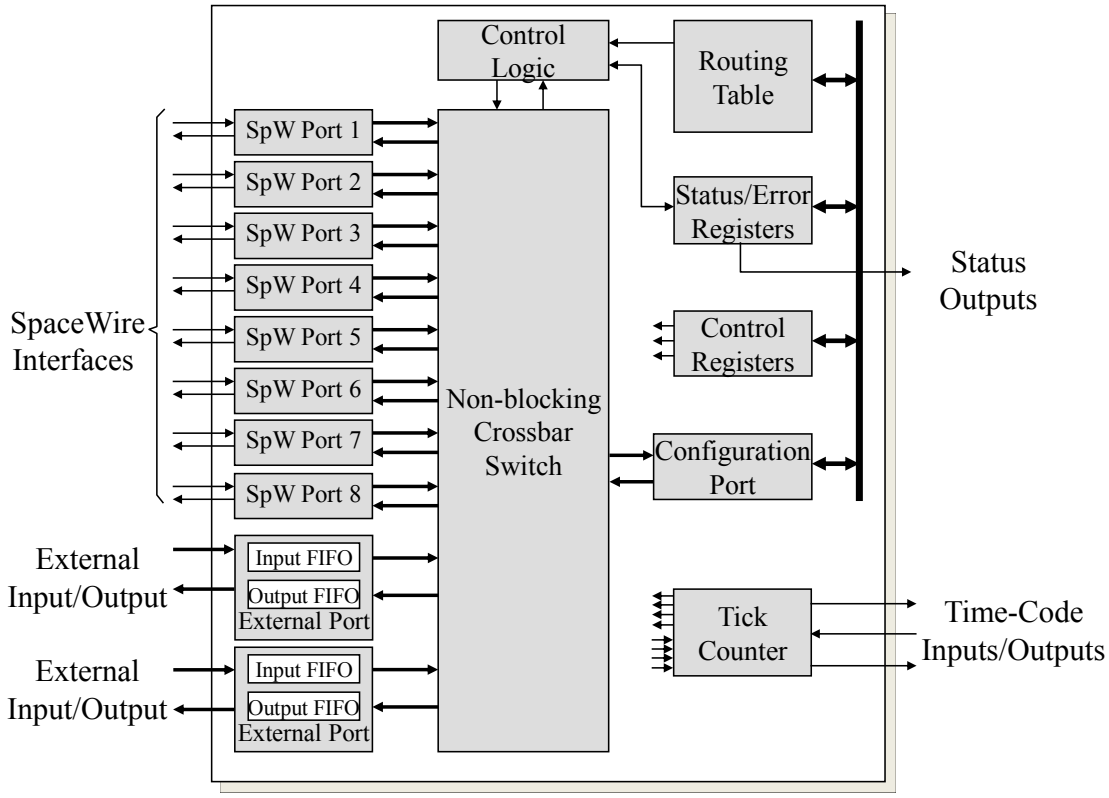


Figure 1 SpW-10X SpaceWire Routing Switch Architecture

SPW-10X DEVELOPMENT AND TESTING

The SpW-10X IP was designed by University of Dundee. Austrian Aerospace prepared this design for implementation in an Atmel radiation tolerant ASIC, EADS Astrium GmbH validated the design and Atmel implemented the ASIC. Funding and overall management was provided by ESA.

The SpW-10X device has been tested in several different ways:

- During design by University of Dundee a VHDL tests bench was used for initial testing. The architecture of this test bench is shown in Figure 2.

- An independent test bench was developed by Austrian Aerospace providing extensive tests and identifying several issues with the initial VHDL code.

- The Router IP was implemented in several STAR-Dundee devices [3] and has been widely used by many organisations.

- The SpW-10X device was implemented in a Xilinx FPGA with the design kept as close as possible to the final VHDL code used for the ASIC design. This SpW-10X FPGA was extensively tested by EADS Astrium GmbH (see Figure 3).

- A second SpW-10X FPGA device was implemented as a mezzanine board in preparation for final ASIC prototype testing (see Figure 4). A similar board has been designed to carry the SpW-10X ASIC.

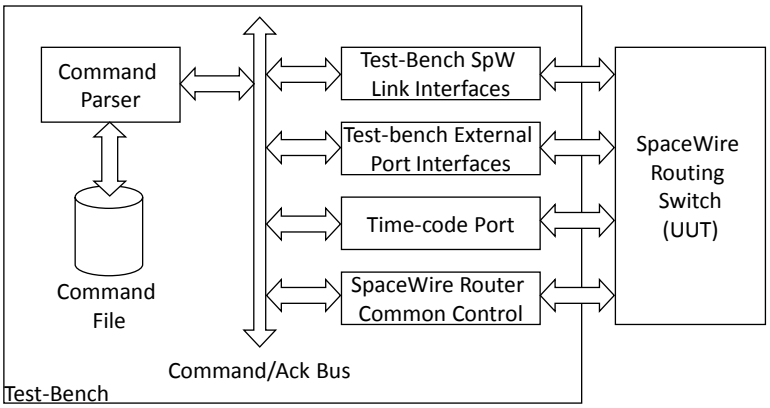


Figure 2 VHDL Test Bench

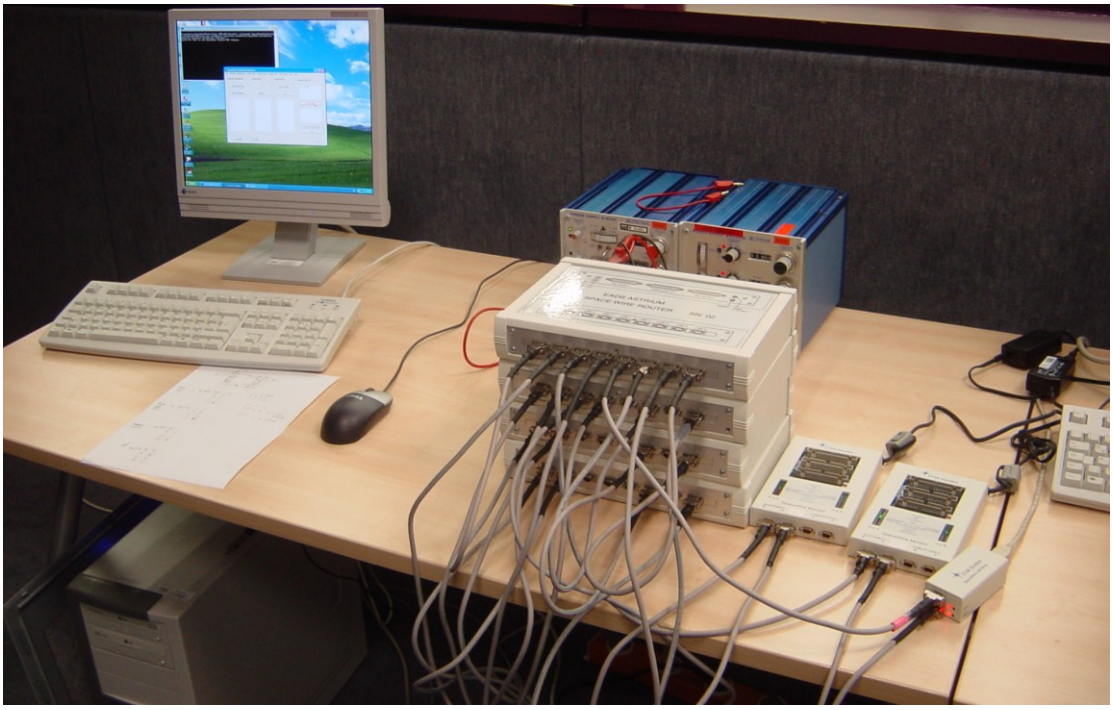


Figure 3 Four SpW-10X FPGA Prototypes Under Test

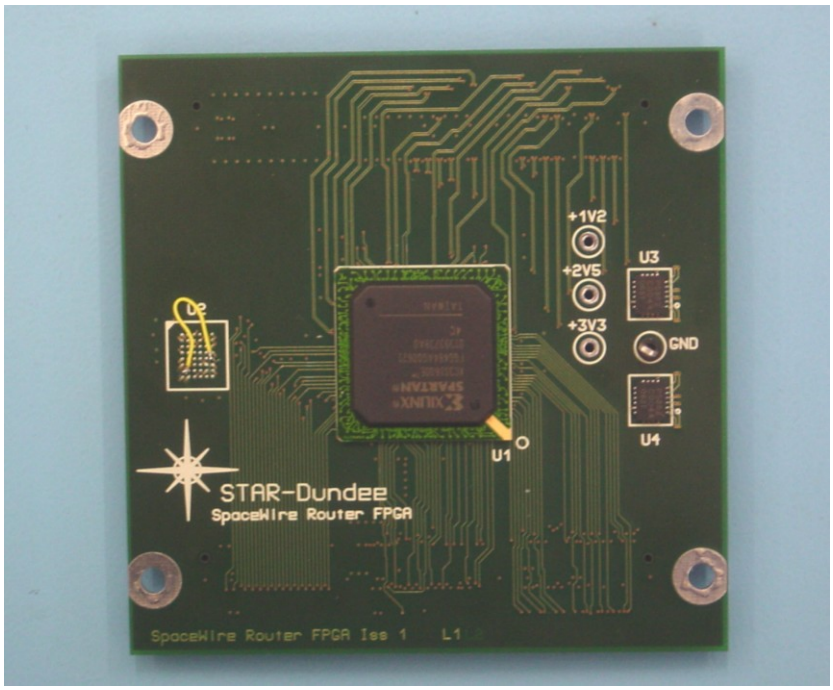


Figure 4 SpW-10X FPGA Mezzanine Board

The prototype ASIC devices are currently being manufactured and are expected in November. A test campaign will then start to characterise the devices. Production devices are anticipated in 1Q08 and will be available from Atmel as the AT7910E. Support for the device will be provided by STAR-Dundee Ltd [4].

SpW-10X Performance

The interfaces on the SpW-10X ASIC are designed to operate at up to 200 Mbits/s. Packet switching times are approximately 0.5 microseconds. Other performance parameters for data character and time-code transmission are detailed in the following tables:

Description	Value	Units
Router Latency – SpaceWire to SpaceWire port	547	ns, max
Router Latency – SpaceWire to External port	317	ns, max
Router Latency – External to SpaceWire port	364	ns, max
Router Latency – External to External port	167	ns, max

Table 1 SpaceWire Router Latency (200Mbit/s example)

Description	Value	Units
Time-code Latency – SpaceWire to SpaceWire port	410	ns, max
Time-code Latency – SpaceWire to External port	317	ns, max
Time-code Latency – External to SpaceWire port	360	ns, max
Time-code Jitter	117	ns, max

Table 2 SpaceWire Router Time-code Latency and Jitter (200Mbit/s example)

These performance values are preliminary and derived either from simulation and/or measurements on the FPGA version.

CONCLUSIONS

A SpaceWire routing switch is a key component in a SpaceWire network. The SpW-10X ASIC has been specifically designed for use in spacecraft onboard data-handling systems. Extensive testing was done on the design prior to manufacture. Prototype devices will be available for further testing and characterisation in November 2007. Following this test campaign they should be available as an ASSP.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support of ESA for the work done on the SpW-10X device, which was done under ESA Contract Numbers 15803/01/NL/JA and ESM-CON-001-AAE.

REFERENCES

[1] European Cooperation for Space Standardization, Standard ECSS-E-50-12A, "SpaceWire, Links, Nodes, Routers and Networks", Issue 1, European Cooperation for Space Data Standardization, January **2003**.
[2] European Space Agency, "SpaceWire Web Page", European Space Agency, http://spacewire.esa.int/
[3] Parkes S.M. et al, "Remote Memory Access Protocol", ECSS-E50-11 draft F, December 2006.
[4] STAR-Dundee Website: www.star-dundee.com

SPACEWIRE REMOTE TERMINAL CONTROLLER

Session: SpaceWire Components

Short Paper

Jørgen Ilstad, Wahida Gasti
European Space Agency, Postbus 299, NL-2200 AG Noordwijk, The Netherlands
Peter Sinander
Saab Space AB, SE-40515 Göteborg, Sweden
Sandi Habinc
Gaisler Research AB, Första Långgatan 19, SE-41327 Göteborg, Sweden
E-mail: jorgen.ilstad@esa.int, wahida.gasti@esa.int, peter.sinander@space.se, sandi@gaisler.com

1. Introduction

ESA initiated the development of the SpW Remote Terminal Controller (SpW-RTC) ASIC within a global strategy context based on the following driving aspects:

1) Ensure ESA payload program development with a set of ASSP[1]s capable of answering most of the On-Board computing and control needs for the future decade.
2) Adopt new ASIC developments to ESA ASSP strategy to reduce development time and recurring cost.
3) Ensure SpW nodes developed by ESA (i.e. component, module, unit…etc) to be easily integrated in ESA On-Board Distributed Computing and Control System.

Hence, the objective of this paper is threefold:

- To demonstrate that SpW-RTC development is fully in line with ESA strategy
- To demonstrate that the SpW-RTC design has functionalities that fulfils a great number of on-board needs.
- To announce the last phase of its development as the SpW-RTC is currently in its manufacturing and validation phase.

Thus, this paper summarises ESA strategy for the SpW-RTC development and it presents the resulting description of the ASIC. Focus on its multi-function aspects, low power and high performance capability is provided by presenting a set of applications and related architectures where the SpW-RTC is well suited. For the conclusion, ESA ASSPs development strategy is assessed for the SpW-RTC case and related methodology for the SpW-RTC validation is presented.

2. SpW-RTC and ESA strategy for On-Board ASSPs

For the development of the SpW-RTC, ESA initiative can be summarised as follows:

1) Review and synthesis of power computation and communication requirements extracted from a large survey of applications ranging from housekeeping data acquisition to on-board processing. In addition to this, requirements inherent to the component's ability to monitor without loss of key functionalities were added. The results lead to the SpW-RTC specification.
2) The SpW-RTC design took advantage of number of ESA technical developments such as: pre-validated IP cores (i.e. LEON2-FT SPARC V8 core, SpW codec, CAN core), pre-validated rad-tolerant technology developed for the AT697. Atmel will manufacture the ASIC in ATC18RHA technology.
3) ESA is leading and funding SpW-RTC development suites (based on the RTC-SpW FPGA board developed by Gaisler Research) which encompass:
   - SpW-RTC Development suite (H/W and S/W) [RD3], [RD6]
   - SW tools mainly based on SpaceWire device driver for the remote terminal controller [RD4]

The RTC-FPGA development kit is already integrated and validated at system level through TOPNET[2] initiative and related EGSEs[3] with results demonstrated in this conference [RD7].

[1] Application Specific Signal Processors
[2] Technology for Onboard Processing Networks with Extended Throughput

3. SpW-RTC Description

The SpW-RTC is a single chip embedded system, its architecture [RD1, RD2], is presented in figure 1 and provides the mixed capability to effectively perform data handling at platform level and powerful data processing at payload level.

Indeed it is judiciously based on:

- ♣ LEON2-FT SPARC V8 core with MEIKO FP unit.
- ♣ Appropriate amount of on-chip memory EDAC protected 64k SRAM, and EDAC protected interfaces to external memory devices such as SRAM, (EE) PROM .
- ♣ FIFO i/f (8 or 16 bit) with parity check.
- ♣ Two SpaceWire interfaces (compliant to SpW IP codec ECSS-E-50-12A and RMAP / ECSS-50-11) a Controller Area Network (CAN IP HurriCANe) bus controller
- ♣ ADC/DAC i/f (16bit) for payload data acquisition/conversion.
- ♣ Standard interfaces and resources (UARTs, 32bit timers, JTAG, general purpose input and output). It can provide up to 96 I/O lines as general purpose signals depending on its application.

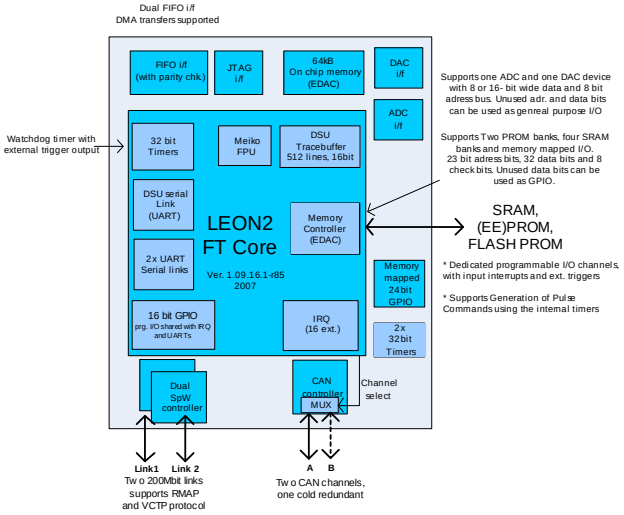


Figure 1: Functional block diagram of the SpW-RTC

The SpW-RTC ASIC is entering the manufacturing stage as of Q3 2007 and ESA is expecting flight prototype by the end of the year.

4. SpW-RTC and On-Board Computing Architecture

The SpW-RTC is a versatile high performance processor mainly designed for modular, reliable and fault-tolerant signal processing and on-board monitoring applications. Hence it is simultaneously maximising both communication aspects and processing power. It offers numerous configuration options considering its resources. Figure 2 illustrates some of its possible applications within an On-Board Computing System. Following sections are detailing some of these examples showing that the SpW-RTC usage is not limited only to payload needs as it was intended in a first perspective but can also be extended to platform demands as well. This list of examples is not exhaustive, but still instructive in terms of the SpW-RTCs capabilities, as other applications could be derived from these basic examples presented in this paper.

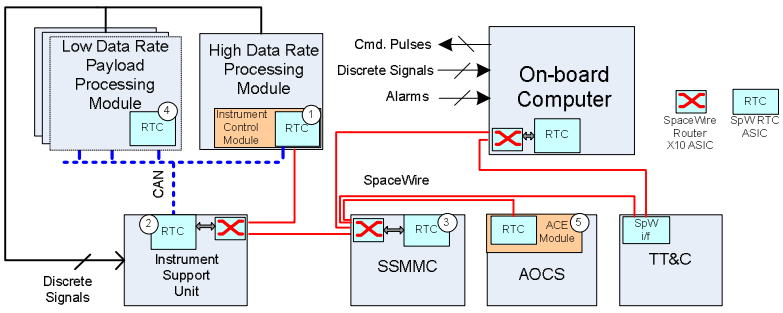


Figure 2:SpW SpW-RTC in a distributed architecture.

[3] Electronic Ground Support Equipement

4.1 SpW-RTC usage in On-Board Computing & Control Modules

4.1.1 Instrument Control Module

Instrument control needs to feature electronics for not only sensor control and monitoring but also data acquisition as well. The SpW-RTC is then the key component for such applications. Figure 3 illustrates how the SpW-RTC can be used as a part of the instrument control unit. High transfer rates can be obtained due to the DMA capability between the SpaceWire interface and the FIFO interface.

One could envisage the LEON2 being used for TM packet generation which is buffered in SRAM and ultimately stored as files in the SSMM unit. The ICU can command and control the other instrument through the CAN bus

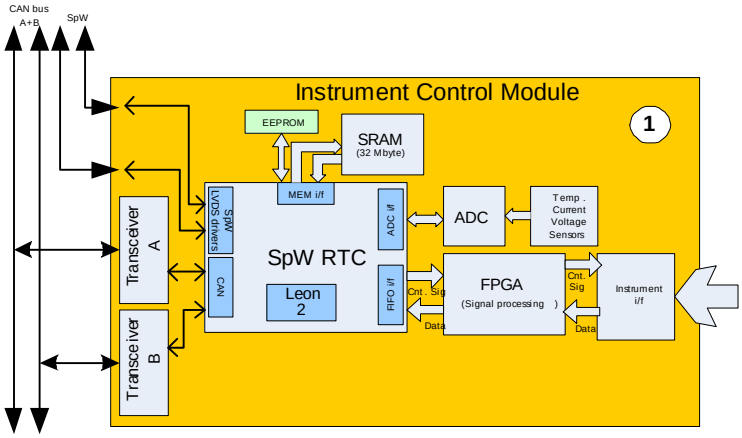


Figure 3: SpW-RTC embedded as a part of the ICU.

4.1.2 Solid State Mass Memory Controller (SSMMC)

Future ESA missions require significantly increased functionality and performance for the SSMM. Main requirements can be summarised as:

- ♣ To be able to handle a significant number of high speed SpW link simultaneously
- ♣ To be adaptable and configurable to different missions.
- ♣ To be able to support both payload and platform needs.
- ♣ To provide standard PUS[4] and SOIS[5] services at the application level for all types of memory access and control.
- ♣ To be able to implement SW file management services
- ♣ To easily upgrade capacity with memory chip technology improvement.

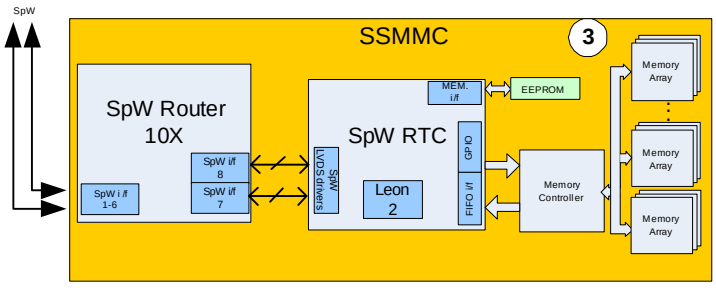


Figure 4:SpW-RTC embedded as a part of the SSMM

As represented in Figure 4, the SpW-RTC associated with a SpW_10X router [RD5] can be a part of the core elements the SSMM controller. As indicated in the illustration on the left, the SpW-RTC might act as a protocol handler / file management device towards the memory controller sub system.

[4] Packet Utilization Standard, ECSS-E-70-41A
[5] Spacecraft Onboard Interface Services, CCSDS SOIS Draft Greenbook 850x0g0b

4.1.3 Payload Processing Module

Figure 5 illustrates the SpW SpW-RTC being used in a digital module as a part of the payload processor module. General purpose I/O is used for discrete signalling and its UARTs are utilized to establish serial links with peripheral units. In this particular example, the SpW-RTC can largely reduce its power consumption by reducing the system clock or the external clock SpW clock speed. The SpW codec allows for a receiving bit rate 7 times faster than the frequency of the SpW clock. The PLL used for SpW codec supports multiplication with factor 2 or 3. Selecting a 30 MHz SpW clock, still allows for reception of 200Mbps, while transmit rate is 60Mbps.

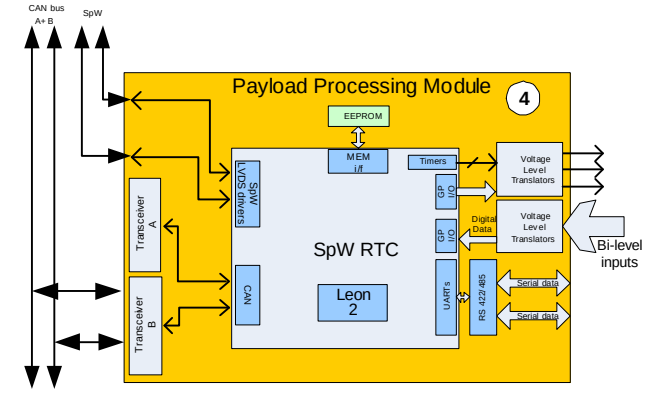


Figure 5:SpW-RTC embedded as a part of the PPM

4.1.4 Attitude, Control Electronics Module

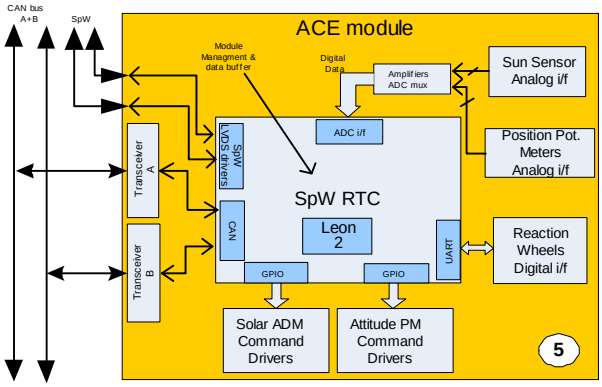


Figure 6: Attitude, Control Electronics Module

Figure 6 suggests how the SpW-RTC could be implemented in an attitude, control electronics module. Using the devices standard interfaces in conjunction with parallel interfaces offered by the GPIO, most applications can be accommodated for. Although the CAN bus in most cases is sufficient for command and control as well as data transfer, the SpaceWire links offers the added capability to remotely upload software using RMAP.

4.1.5 Instrument Support Unit

Monitoring of housekeeping parameters, both analog and digital, are typical application areas where the SpaceWire SpW-RTC is well suited. Referring to figure 7 the SpW-RTC is used as a separate analog acquisition board within the instrument support module. Measurements from temperature sensors, voltage and current sensors are typical HK parameters that will be accessible through the CAN bus or the SpaceWire links. The ICU may command and control the other instruments through the CAN bus.

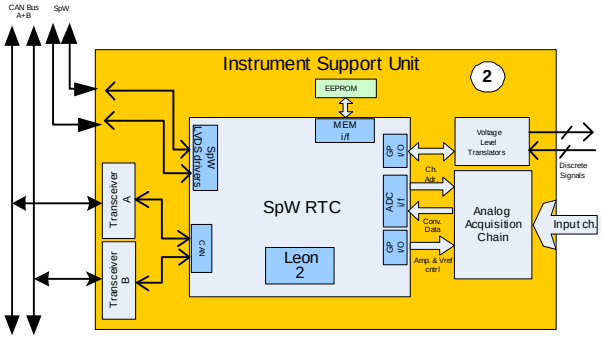


Figure 7: Instrument Support Unit general block diagram.

5. Conclusion

Overall, the modular architecture concept promoted by ESA and based on SpW interfaces and the development strategy adopted by ESA proved to be beneficial and fully applicable for the SpW-RTC development. Beside very specific applications driven by high data rates, the SpW-RTC design complies with very challenging On-Board computing and control needs.

ESA methodology for ASSPs validation is based on a set of selected alpha customers that will test the device in different context applications. For the SpW-RTC case, a community of scientific instrument designers related to ESA Bepi-Colombo mission has acquired the FPGA version of the SpW-RTC development kit and related tools. This has been done with the support of ESA to help these users for the implementation of their applications with the advantage of:

- Rapid prototyping
- Physically implementation of the algorithms, that concludes in a rapid execution of tasks
- Combine the tools of both hardware and software making the integration and validation phases easier and significant reduction in time-to-test and integration

The goal of the community is to assess the SpW-RTC device as common back-end candidate for most of Bepi-Colombo instruments. The benefit and the return of information of this community will pave the road of the SpW-RTC usage not only for Bepi-Colombo mission but to the following ESA programs XEUS, Cosmic Microwave Background Polarization Mapper (CMPM), to name a few of them.

Some of the most generic architectures using the SpW-RTC device for On-Board computing system have been presented. The goal of this survey was to summarize the ideas that emerge during its development as validation applications. The idea of parallel and pipelined architectures based on multiple SpW-RTC devices also emerged to solve problems that require higher level of computation. The related set will be forming fast processors for implementation of different tasks in payload processing. This type of architecture will need to be assessed and can constitute the basis of further work

6. Reference Documents

[1]: SpaceWire Remote Terminal Controller, User Manual v1.7 (issued Feb 2007)
[2]: SpaceWire Remote Terminal Controller datasheet v.5 (issued May 2007)
[3]: "*Integrated Development Tools Suite for the SpaceWire RTC ASIC*"
W.Errico, J.Ilstad, A.Colonna, F.Bertuccelli
International Spacewire Conference 17-19 September, 2007 Dundee (Scotland)
[4]: "*SpaceWire device driver for the remote terminal controller*"
A.Ferrer Florit, W.Gasti
International Spacewire Conference 17-19 September, 2007 Dundee (Scotland)
[5]: SpaceWire Router Data Sheet Issue 2.0 August 18, 2006
[6]: SpW-RTC Development Suite,
S.Habinc, J.Ilstad
International Spacewire Conference 17-19 September, 2007 Dundee (Scotland)
[7]: TopNet demonstration
R.Vitulli, A.Ferrer Florit, J.Ilstad
International SpaceWire Conference 17-19 September, 2007 Dundee (Scotland)

SPACEWIRE DEVICE DRIVER FOR THE REMOTE TERMINAL CONTROLLER

Session: SpaceWire Components

Short Paper

Albert Ferrer Florit, Wahida Gasti

European Space Agency, Postbus 299, NL-2200 AG Noordwijk, The Netherlands

E-mail: albert.ferrer.florit@esa.int, wahida.gasti@esa.int

ABSTRACT

The SpW Remote Terminal Controller (RTC) ASIC is a single chip embedded system designed to effectively perform data handling at platform level and powerful data processing at payload level. One of its key features is the implementation of two highly configurable SpaceWire interfaces.

This paper presents an implementation of a SpW software device driver for the RTC. The driver provides the necessary interrupt service routines, functions and procedures to handle the SpW interface. It includes the definition of an Application Programming Interface (API) for SpW compliant embedded systems. Reception scheme is based on early packet identification so that a decision can be made whether a packet is immediately processed, discarded or saved in the system for further processing. Transmission scheme supports multicasting, send queues, and priority schemes. The driver implementation overcomes the problematic of arbitrary packet lengths and network blocking, providing sustained high data rates transfers.

1 HARDWARE DESCRIPTION

The SpaceWire-RTC device is a single chip that includes an embedded LEON2-FT SPARC V8 processor with a Floating Point Unit, two SpaceWire interfaces, a Controller Area Network (CAN) bus controller, ADC/DAC interfaces for analogue acquisition/conversion, and standard interfaces and resources (UARTs, timers, general purpose input and output). Software can be executed from the on-chip 64Kbyte memory or from external SRAM. [1]

The actual implementation of the SpaceWire interface is interrupt driven and performs DMA transfers for the reception and transmission of SpaceWire packets. The interface must be programmed with the memory location and the size of one receive and one transmit buffer, i.e it implements a single virtual channel for each link direction. Multiple packets can be stored in the same receive buffer but the information about their lengths is not preserved.

As additional features, the SpaceWire interfaces provide an extra virtual channel reserved for VCTP packets [1], and a hardware implementation of the RMAP protocol [2]. This last characteristic allows to read/write all memory space of the RTC from a SpW remote node without the intervention of the embedded processor.

2 DESIGN CONSIDERATIONS

After reviewing some on-board recurring applications SpW-based, the following design considerations for the SpW Driver have been identified:

Variable packet length: The SpaceWire standard [3, 4] defines a data link protocol layer with arbitrary packet sizes. In a robust SpW network all nodes should be able to handle or tolerate SpW packets of arbitrary size, despite the limited memory available in some embedded systems.

Network congestion: One of the important points for any SpaceWire node design is its networking issues and accordingly, its capability to avoid network congestion. When a destination node sends more data than the receiver one can store or process, the whole data path gets blocked through multiple routers due to the use of wormhole routing. Therefore, a SpW driver should be able to discard incoming data automatically when the receive buffer is full.

Limited resources: Spacecraft data handling is based on embedded systems with limited resources in terms of memory and processing power. A SpW driver implementation should avoid memory transfers when possible. For example, basic packet identification, i.e. size and protocol used, should always be obtained without requiring the application to read and store the complete packet. Considering that the link speed can be a multiple of the processor speed, in case of high network traffic non DMA memory transfers can be very demanding for the system.

Multi-threaded applications: For complex applications, the driver should also allow multiple threads to send packets with different priorities, optionally, in the context of an operating system. Blocking and non blocking functions should be provided.

Extended functionalities: Link error reporting capability and time-code functionality has to be considered, as the possibility to provide time stamp information on packet reception.

3 SPECIFICATIONS

Based on the above design considerations, the following main SpW driver features were implemented.

Performance and efficiency:

- Support for sustained bidirectional high data rate transfers (up to 160Mbit/s, for low demanding data processing applications)
- User application can obtain the length of a packet and may read a complete packet without performing any memory copy.

Memory requirements

- Driver implementation has a small code and data footprint, and does not require any external library.
- Receive buffers can have arbitrary sizes and can be dynamically adjusted by the user application. A receive buffer may contain multiple SpW packets.

SpaceWire functionality

- Three different packet transmission functions, including multicast packet function. Information about time transmission is provided upon completion.
- Multiple send requests can be queued and an identifier is provided to supervise their status. They can have two levels of priority and be cancelled before being executed.
- Information provided on packet reception: packet length, protocol ID, EOP marker, CRC and other errors.
- Multiple receive buffers can be queued or added dynamically. Receive buffers are actually implemented as optimised receive FIFOs.
- Capability for hardware packet rejection and software packet filtering on packet reception. Statistical information about packets rejected and filtered is provided.
- There is no limitation in packet sizes. Packets that are bigger than the size of receives buffers available can be read in multiple chunks of bytes.
- User application can be notified when a packet or Time Code is received and/or a send request is completed.
- The driver can be configured to automatically discard incoming data in case there is no more memory available for packet reception, and/or notify the user application of this event.
- The driver can configure the hardware to act as a time-master (sending Time Codes periodically) or time-receiver (retrieving the last received Time-Code).
- The driver provides complete link configuration and error notification and recovering. It also provides configuration functions for the RMAP and VCTP hardware support.

4 OPERATION

The SpW device driver provides the data link layer services defined in the SpW standard, plus configuration functions for the RTC specific hardware support of VCTP and RMAP protocols. The user application access to these services using an API (Application Programming Interface) specifically designed to simplify the operation of all SpW driver features [5].

Upon a successful initialization of the driver and a SpW Link, a buffer previously allocated in the memory by the user must be provided to the driver to be used as a receive buffer for the incoming packets.

The application requests information about each packet received, and obtains a pointer to the memory location where each packet is stored. The driver implements a circular FiFO, trying to avoid splitting a packet in multiple buffers. However, in some situations with large packets, this may not be possible. In that case, a specific function must be called that copies the packet to another user buffer.

The memory space used by the received packets in the receive buffer should be explicitly freed by the user after the packet has been copied or processed.

It is also possible to keep the packet in the driver's receive buffer. But in that case when the buffer gets full the driver will not reuse the memory space used. Therefore, the application must provide a new receive buffer (it may be queued, see figure 1)

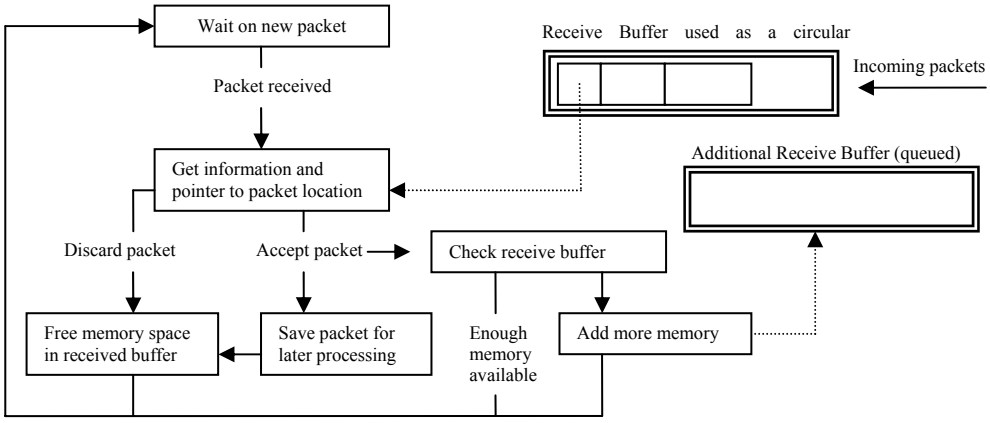


Figure 1: Simplified program flow for packets reception. Each box represents an API function call and an abstract representation of the receive buffers is included. Note that depending on the application, packets can be kept in the circular RxFIFO until they are processed.

Complex applications dealing with multiple protocols may have a single thread, which monitor a link and identifies the packets, and then forwards them to other threads that handle specific protocols. For packet transmission, multiple threads may execute and supervise send requests independently.

An application using some of the Driver's features can be found in [6].

5 CONCLUSIONS

An implementation of a SpW device driver has been presented targeting the Remote Terminal Controller. The Driver provides a full set of data link layer services, including time stamp information, priorities, packet filtering, link error reporting, and time-codes. The resulting design has a small memory footprint and low processing power needs, while providing enough flexibility to support a wide range of applications.

6 REFERENCES

1. SpaceWire Remote Terminal Controller, User Manual v1.7 (issued Feb 2007)
2. Remote Memory Access Protocol. ECSS-E-50-11 Draft F
3. SpaceWire - Links, Nodes Routers and Networks, ECSS-50-12A
4. ECSS-E-50-12 Part 2 Draft B January 2005 (also in ECSS-E50-11 chapter 5)
5. SpaceWire driver for the Remote Terminal Controller, User Manual.
6. TopNet Demonstration, R.Vitulli, A.Ferrer Florit, J.Ilstad
   International SpaceWire Conference 17-19 September, 2007 Dundee (Scotland)

"MCFLIGHT" – THE CHIPSET FOR DISTRIBUTED SIGNAL PROCESSING AND CONTROL WITH SPACEWIRE INTERCONNECTIONS

Session: SpaceWire Components

Long Paper

*Tatyana Solokhina, Alexandr Glushkov, Ilya Alexeev,**
*Yuriy Sheynin, Elena Suvorova, Felix Shutenko***

**) ELVEES RnD Center, Moscow*
***) St. Petersburg University of Aerospace Instrumentation*

E-mail: tanya@elnet.msk.ru, grisly@elvees.cm, sheynin@online.ru

ABSTRACT

The article presents the "MCFLIGHT" family of chips for distributed architectures with SpaceWire interconnections for signal processing and control. The chipset chips was developed for aerospace on-board data processing and control systems on the base of customizable System-on-Chip (SOC) design and is produced by the "ELVEES" RnD Center (Moscow). Main features and characteristics of the chips are presented.

1. INTRODUCTION

The chipset "MCFLIGHT" was developed for aerospace on-board data processing and control systems on the base of modern technology of customizable System-on-Chip (SOC) design. It includes 5chips, Table 1, for building distributed architectures with SpaceWire interconnections.

Table 1. Chipset "MCFLIGHT" with SpaceWire for aerospace applications

Chip		Type
	**MC24R**	**Dual-core DSP**; 600MFLOPs peak performace; 2 embedded **SpaceWire** links, 5-400 Mb/s
	**MCT01**	**Peripheral controller** RISC-core (MIPS32-64FP); 4-channel ADC/DAC; 2 embedded **SpaceWire** links, 5-400 Mb/s
	**MCB01**	**Bridge** **SpaceWire** /PCI/Memory bus, 4 SpaceWire links, 5-400 Mb/s embedded SRAM 2 Mb
	**MCK01**	16-channel **SpaceWire routing switch**; SpaceWire links 5-400 Mb/s, embedded RISC-core (MIPS32 architecture); Processor bus; Memory bus.

The chipset incorporates ideas of multi-core chips with RISC- and DSP-cores, new high-rate on-board communication technology SpaceWire, scalable distributed

architecture support and advanced embedded software for real-time parallel processing and control in distributed architectures. The chipset includes the high-performance multicore (RISC + DSP cores) processors MC24R with embedded SpaceWire links, the control processor MCT01 with embedded SpaceWire links, the 16-port SpaceWire routing switch MCK01, the 4-channel SpaceWire bridge for PCI bus and Memory access bus. Chips are produced in 0,25 un technology. Used in these chips IP-blocks of SpaceWire link controllers (NICs) run with rates up to 400 Mbit/s and implement extended version of the SpaceWire standard with Interrupt/IntAknowledge codes support. SpaceWire NICs are supported by drivers in Linux that run on the chips. Integrated software development package MCStudio is provided for software development and debugging.

2. DUAL-CORE DIGITAL SIGNAL PROCESSOR MC24R

Digital signal processor MC24R was designed for application in on-board control and data systems. It is the heterogeneous dual-core processor chip: a 32-bit DSP-core for high-performance signal and data processing and a 32-bit RISC-core for control and scalar data processing. Its structure is presented at the fig. 1., its main characteristics in the Table. 2.

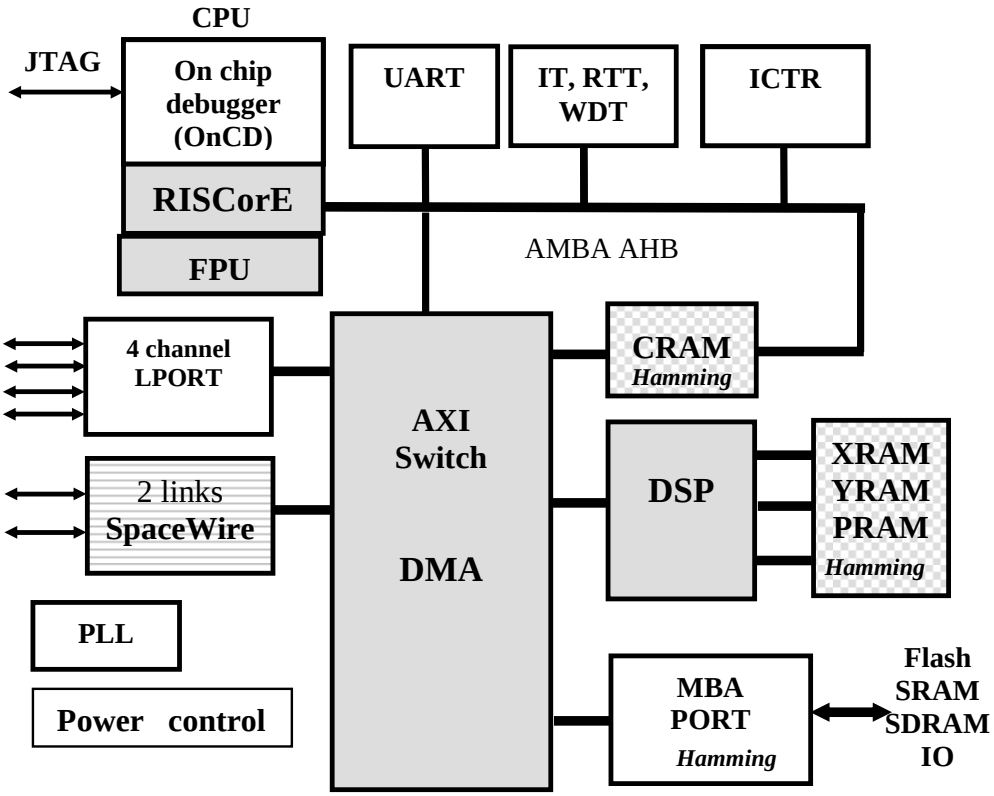


Fig. 1. Digital signal processor MC24R structure

The DSP-core has the original microarchitecture ELcore-24, which is optimized for massive signal and vector data processing, incorporates VLIW, pipelining, macro load/store and 2-way SIMD architecture features. It also includes 3 high-rate memory blocks with parallel access for DSP-core program codes (PRAM) and data (XRAM and YRAM). The 32-bit RISC-core has MIPS32-compatible architecture, with a cash and FPU, but is the originally designed by ELVEES processor core. Low latency

memory CRAM stores RISC-core program codes and data. The 16-channel DMA-core supports multiple data flows inside the chip and with off-chip components.
One of the MC24R specifics is extended support for building distributed on-board systems. Embedded in the MC24R 2 SpaceWire links provide duplex information flows with rates up to 400 Mb/s each. They ensure interfacing the MC24R with remote sensors and information flow recipients. It can be directly connected to other SpaceWire-compliant devices, routing switches included (e.g. MCK-01). Thus the MC24R is a ready-made component for building distributed on-board systems with SpaceWire based interconnections.

The chip was design with the "Rad Hard By Design" methods, with 0,25 μn CMOS technology. For fault-tolerance Hamming coding is used also.

Table 2. Digital signal processor MC24R characteristics

Technology	CMOS ASIC, 0,25 μn/ 5metal layers, 3,3 V (peripherals) / 2,5 V (core)
Frequency	100 MHz for the RISC-core 200 MHz for the DSP-core
RISC– core	32-bit, MIPS32-compatible architecture Performance 100 Mips Instructions cash – 16 Kbytes
FPU	Works in parallel with the main RISC-core, single and double ANSI/IEEE754 flow point operations.
DSP-core	Original microarchitecture ELcore-24. Peak performance: 600 Mflop– 32-bit IEEE754, 3600 Mips – 8-bit fixed point, 1600 Mips - 16-bit fixed point 800 Mips – 32-bit fixed point
On-chip memory	CRAM: 32 Kbytes (dual-prt) XRAM, YRAM 64 Kbytes PRAM 16 Kbytes
Memory port (MBA)	Address bus - 32 bits, Data bus - 64 bits Supports SRAM, Flash, SDRAM (100МГц) Programmable wait cycles; 4 interrupt lines. 4 external DMA requests
SpaceWire links	2 SpaceWire links Compliant with the ECSS-E-50-12A standard, with extensions (distributed interrupts). 5-400 Mb/s. LVDS-signals (ANSI/TIA/EIA-644).
Other interfaces	4 byte-wide ports (compliant with ADSP21160), . GPIO mode. UART: UART 16550 type, 50 baud – 1Mbaud. JTAG IEEE 1149.1
OnCD	On-chip debug facilities, access through JTAG IEEE 1149.1.
Power saving modes	Several modes for power saving, programmable.
Fault tolerance	All memory blocks are protected by modified Hamming code
Package	HSBGA-416

3. PERIPHERAL CONTROLLER MCT01

Peripheral controller MCT01 was designed as a terminal controller of distributed on-board systems for direct interfacing with on-spacecraft equipment and payload instruments. For interfacing the MCT01 has 4 channels of ADC and 4 channels of DAC, GPIO pins, UART. The MCT01 structure is presented at the fig. 2. and its main characteristics in the Table. 3.

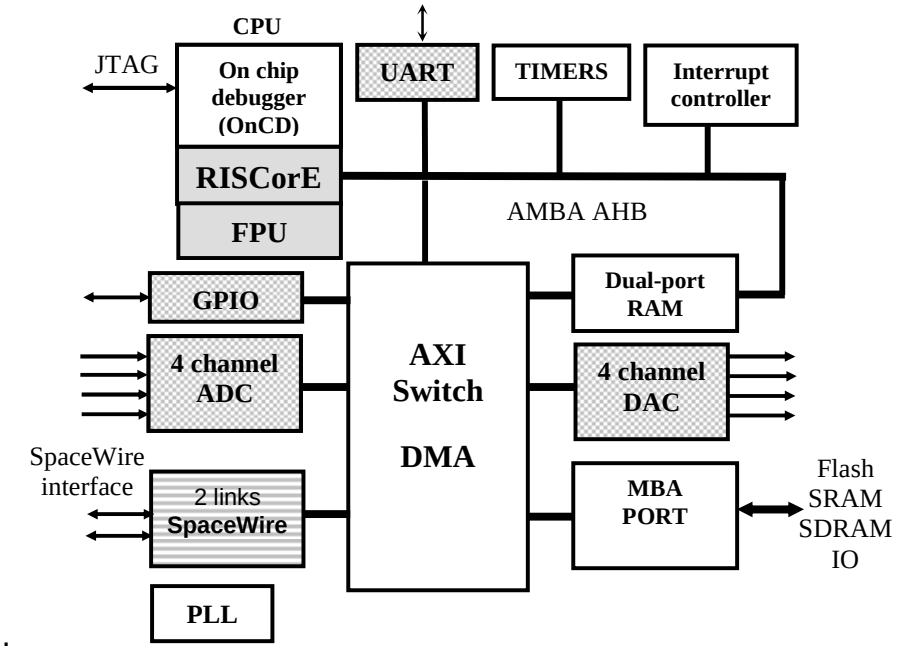


Fig. 2. Peripheral controller MCT01 structure

For local control, medium rate signal and data processing the MCT01 has the MIPS32-compatible 32-bit RISC-core, with a cash and FPU. There is on- dual-port SRAM for program codes and data. The RISC-core with FPU are similar to the RISC-core in the MC24R. Additional memory and IO devices can be attached by the programmable MBA port. Two 2 SpaceWire links embedded in the MCT01 provide duplex information flows with rates up to 400 Mb/s each. Thus MCT01 can be directly connected to other SpaceWire-compliant devices, routing switches included, processors, other peripheral controllers, as well as to instruments and sensors with embedded SpaceWire links. As the other **"**MCFLIGHT**"** chips MCT01 is a ready-made component for distributed on-board systems with SpaceWire interconnections.

Table 3 Peripheral controller MCT01 characteristics

Technology	CMOS ASIC, 0,25 μn/ 5metal layers, 3,3 V (peripherals) / 2,5 V (core)
Frequency	100 MHz
RISC– core	32-bit, MIPS32-compatible architecture Performance 100 Mips Instruction/Data cash – 16 Kbytes/16 Kbytes
FPU	Works in parallel with the main RISC-core, single and double ANSI/IEEE754 flow point operations.

On-chip memory	SRAM: 62 Kbytes (dual-port)
DMA	16 channels
32-bit timers	Interval timer; Real-time timer;Watchdog timer
Memory port (MBA)	32 bits, Supports SRAM, Flash, SDRAM (100 MHz) Programmable wait cycles; 4 interrupt lines. 4 external DMA requests
SpaceWire links	2 SpaceWire links Compliant with the ECSS-E-50-12A standard, with extensions (distributed interrupts). 5-400 Mb/s. LVDS-signals (ANSI/TIA/EIA-644).
Other interfaces	GPIO 16 bits. UART: UART 16550 type, 50 baud – 1Mbaud. JTAG IEEE 1149.1
OnCD	On-chip debug facilities, access through JTAG IEEE 1149.1.
ADC/DAC	4 channels, 13 bits, 1 Mhz.
Package	BGA-292

4. MULTI-CHANNEL BRIDGE MCB01

The MCB01 (MultiCore Bridge) is a multipurpose chip for interfacing processors and subsystems, which have not a SpaceWire interface, with SpaceWire interconnections. It can be also used to extend number of SpaceWire links, e.g. extend number of SpaceWire links of a MC24 based unit from2 to 6. The MCB01 structure is presented at the fig. 3. and its main characteristics in the Table. 4.

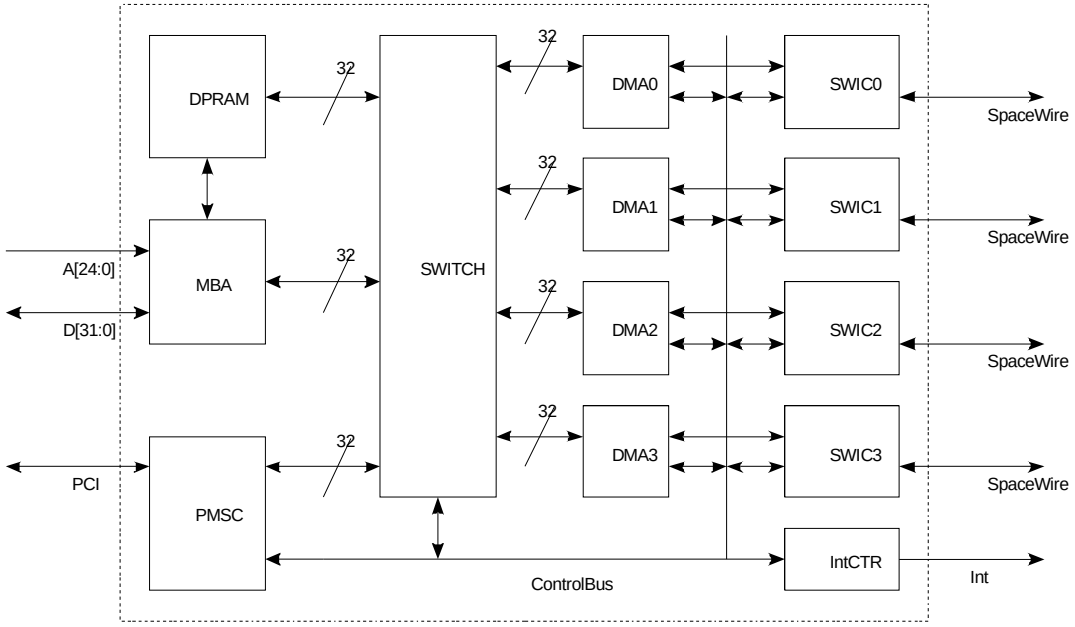


Fig. 3. Multi-channel bridge MCB01 structure

MCB01 has 4 independent SpaceWire link controllers, SWIC0 – SWIC3. They work in parallel and have independent operation mode control, link transmission rates included. SpaceWire links controllers generate signals on a set of events in the links operation (connection/disconnection; parity error; EOP, time-code, interrupt-code

receipt; etc.). They are stored in the links' status registers and can be transferred to a host by interrupt signals (can be masked), both to MBA and PCI ports.
From the side of a host the MCB01 provides two types of parallel interfaces: the MBA port (a typical for microcontrollers bus) and the PCI bus interface. By the MBA port the MCB01 can be attached, for instance, to all the processor and controller chips of the "MCFlight" family. By the standard for industry PCI port it can be connected with a vast variety of chips, computers and systems that use PCI bus. The PCI port (slave) can operate in 32 bit or 64 bit modes, at 33 or 66 MHz. The MCB01 architecture supports simultaneous hosts operation by the MBA and the PCI ports.
Embedded MCB01 dual port high-rate memory (128 Kbytes) supports full rate operation of all the 4 duplex links. The SpaceWire link controllers work with the memory by embedded in them multi-channel DMA. From the host side, the MCB01 embedded memory and registers of the SWICs are represented in the host memory address space (both from the MBA and from the PCI sides).

Table 4. Multi-channel bridge MCB01 characteristics

Technology	CMOS ASIC, 0,25 μn/ 5metal layers, 3,3 V (peripherals) / 2,5 V (core)
SpaceWire	**4** SpaceWire links. Independent link operation modes and transmission rates programming Compliant with the ECSS-E-50-12A standard, with extensions (distributed interrupts). 5-400 Mb/s. LVDS-signals (ANSI/TIA/EIA-644).
Memory port (MBA)	32 bits data bus, 25 bits address bus, 100 MHz)
PCI	PCI bus port (slave); Local Bus Specification. Rev. 2.2. 32-bit and 64 bit modes; 33 MHz / 66 MHz;
On-chip memory	SRAM 128 Kbytes (64Kx32), dual-port, up to 100 MHz
Package	HSBGA-416

5. SPACEWIRE ROUTING SWITCH MCK01

The MCK01 chip is the 16-channel SpaceWire routing switch. It was designed as the "MCFLIGHT" basic component for building scalable SpaceWire interconnections for on-board aerospace systems. The MCK01 is designed in conformance with the SpaceWire layered protocol stack. It can be used for interconnections between processing modules of parallel and distributed data and signal processing systems, as well as for organizing programmable interconnection between sensors/actuators and processing/control on-board computers. It provides efficient, low latency switching for different types of information flow: from heterogeneous flows of short packets (typical for inter-module traffic in parallel processing and distributed control) up to continuous homogeneous data flow from sensors to DSP. The MCK01 structure is presented at the fig. 4. and its main characteristics in the Table. 5.
MCK01 has 16 SpaceWire links with individually programmable mode of operations and transfer rates. 16 parallel link controllers and non-blocking switch provide high throughput and low latency packets and control codes switching through the router.
The MCK01 implements in hardware wormhole routing with all the specified by the SpaceWire standard addressing modes: path, logical, regional-logical addressing. It is

supported by the programmable full scale routing table in the chip. The MCK01 supports adaptive routing also, thus providing a mechanism for information flow distribution between multiple terminal nodes, with automatic dynamic information flows redistribution between channels, adaptation to channel faults for reliable on-board system configurations with redundancy.

For building real-time distributed systems the MCK01 supports an extended set of control codes. Along with the specified by the ECSS-E-50-12A standard time-codes it implements also the distributed Interrupt/Interrupt Ack codes that were proposed for the next release of the SpaceWire standard. As much as 32 Interrupt codes can be simultaneously distributed with low latency and without blocking or retarding by data flows in a SpaceWire interconnection with the MCK01 switches.

This routing switch is an intelligent one. While multi-megabit packet flows on 16 duplex links are processed by the MCK01 in hardware, on the fly, it has the embedded RISC-processor core also.

The configuration port (port 0) is assigned to the RISC-core unit. Thus the MCK01 has particular facilities for scalable intelligence and adaptability for application environments as well as to evolution of SpaceWire family standards. For instance, a PnP protocol that is under development now by the SpaceWire community, whatever final form it will have, will be readily implemented in firmware of the MCK01 that is in production already; it could be doped even into an installed and operating equipment or instrument with MCK01 inside.

Two parallel ports expand the MCK01 intelligence scalability and flexibility of its application in systems with different requirements and configuration. By the MPORT (master) additional memory (SRAM, Flash, SDRAM) or peripherals can be attached to be accessed and controlled by the embedded RISC-core. By the MBA (slave) external processor can interface the MCK01 for data interchange and taking control over the MCK01 operation. It will have access to the embedded memory and registers of the MCK01 that will be seen in its memory address space. The UART port makes interfacing the MCK01 to external test, debug or monitoring tools quite easy.

Table 5. SpaceWire routing switch MCK01characteristics

Technology	CMOS ASIC, 0,25 μn/ 5metal layers, 3,3 V (peripherals) / 2,5 V (core)
SpaceWire links	16 SpaceWire links Compliant with the ECSS-E-50-12A standard, with extensions (distributed interrupts). 5-400 Mb/s. LVDS-signals (ANSI/TIA/EIA-644).
RISC– core	32-bit, MIPS32-compatible architecture Performance 100 Mips
Frequency	100 MHz for the RISC-core Individually programmable frequencies of the 16 SpaceWire links
On-chip memory	SysRAM: 16 Kbytes Routing table 1 Kbyte Packet RAM(for cofiguration port) 8 Kbytes
MBA	Slave. 32 bits.
MPORT	Master. 32 bits; Supports SRAM, Flash, SDRAM (100МГц) Programmable wait cycles;
Other interfaces	UART: UART 16550 type, 50 baud – 1Mbaud. JTAG IEEE 1149.1
OnCD	On-chip debug facilities, access through JTAG IEEE 1149.1.
Package	HSBGA-416

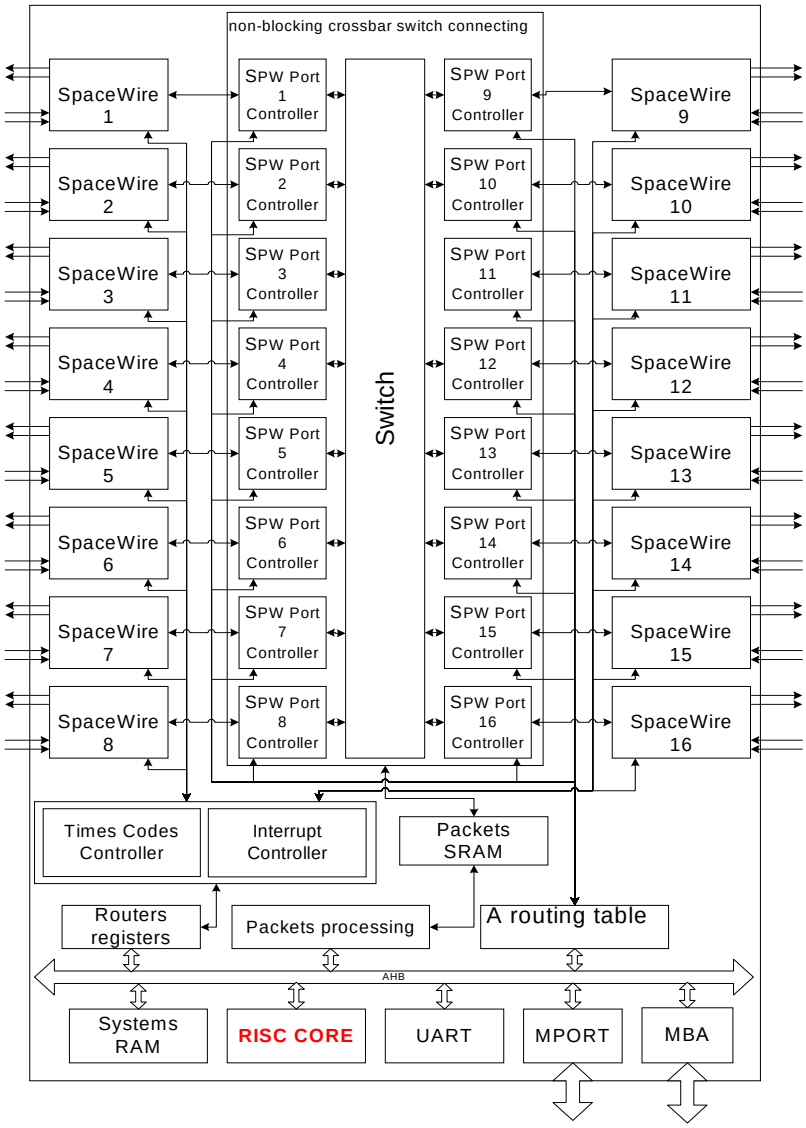


Fig. 4. SpaceWire routing switch MCK01structure

Conclusions

1. The engineering samples of MCFlight - “MULTICORE” platform based chipset (0.25-u) with SpaceWire links for distributed aerospace systems have been received and its functionality is proved on the HW modules;
2. ASICs and FPGA SpaceWire Links provide up to 400 Mbps and higher throughput (>5m) in the cross modules connections;
3. Chipset provides a lot of the innovational features and supports high performance, programmability, scalability and flexibility;
4. A lot of ELVEES ‘s technologies for “MULTICORE” platform (SDR, Adaptive signal/image processing , 3D graphics , Networks information security, multimedia, Intellectual systems of video observation) will be transferred on the MCFlight chipset;
5. MCFlight can be easily modified (during some months) by others platforms IP-cores (for example, for SPARC RISC core) or for others interfaces.

A SYSTEM-ON-CHIP RADIATION HARDENED MICROCONTROLLER ASIC WITH EMBEDDED SPACEWIRE ROUTER

Session: SpaceWire Components

Long Paper

Richard Berger, Laura Burcin, David Hutcheson, Jennifer Koehler, Marla Lassa, Myrna Milliser, David Moser, Dan Stanley, Randy Zeger

BAE Systems, 9300 Wellington Road, Manassas, Virginia 20110 USA

Ben Blalock, Mark Hale

University of Tennessee, 1508 Middle Way Drive, Knoxville, Tennessee 37996 USA

E-mail: richard.w.berger@baesystems.com, laura.burcin@baesystems.com, david.hutcheson@baesystems.com, jennifer.koehler@baesystems.com, marla.lassa@baesystems.com, myrna.milliser@baesystems.com, dave.moser@baesystems.com, dan.stanley@baesystems.com, randy.zeger@baesystems.com, bblalock@ece.utk.edu, mhale1@utk.edu

ABSTRACT

A mixed-signal radiation hardened computer ASIC that includes a four port SpaceWire router is currently in development. Based on the RAD6000™ microprocessor currently flying on numerous space missions and commanding the Mars Exploration Rovers, this massively integrated system-on-chip is capable of performing flight computer and instrument controller functions and can reuse existing RAD6000 software and test infrastructure. The ASIC will be manufactured in a 150nm radiation hardened CMOS technology. Initiated in 2005 as a NASA technology project, development has continued with funding from the Air Force Research Laboratory's Space Vehicles Directorate, Kirtland Air Force Base, N.M..

The ASIC incorporates an enhanced version of the reusable SpaceWire router core with four SpaceWire links and dual internal ports that was previously created for the BAE Systems SpaceWire ASIC. This newer version reduces both die area and power dissipation while improving link performance. The ASIC employs a flight-proven reusable core architecture with a common bus medium. In addition to the RAD6000 microprocessor and SpaceWire cores, the ASIC includes a pipelined 12-bit A/D converter with a programmable multiplexer, three channels of 12-bit D/A conversion, 192KB of on-chip SRAM, 32KB of chalcogenide-based C-RAM™ non-volatile memory, a 64-bit PCI interface, a 1553 interface, a DMA controller, and an external memory controller. A 2nd smaller microcontroller core called the EMC has also been incorporated on the ASIC. It is supported by a compiler developed by BAE Systems and software supporting the SpaceWire transport layer has already been developed.

This paper discusses the architecture and functions of the microcontroller ASIC, including the SpaceWire core implementation and features. Operational configurations matched to a variety of applications will also be shown.

INTRODUCTION TO THE RAD6000MC™ MICROCONTROLLER

Based on the BAE Systems flight-proven RAD6000™ microprocessor and integrating a wide variety of digital and analog interfaces and both SRAM and non-volatile memory, the RAD600MC is a true system-on-a-chip. It is being designed for a variety of applications that include both flight computer processing and instrument control. It supports both legacy buses such as MIL-STD-1553 and the rapidly growing SpaceWire serial bus through an integrated four-port router with dual internal interfaces. The RAD6000MC also supports up to 48 analog input channels, making it ideal for interface with spacecraft sensors and includes three channels of D/A conversion. Non-volatile memory is implemented with a new embedded C-RAM memory macro. Much of the RAD6000MC design is reused from ASICs already proven in hardware and/or currently in development, decreasing both the design cost and risk associated with the program. The ASIC will be manufactured in a radiation hardened 150nm CMOS technology.

RAD6000™ MICROPROCESSOR BACKGROUND AND CORE TRANSLATION

Originally developed by IBM in 1990 as the first single chip implementation of the RISC System/6000 "Power" architecture [1] and the predecessor to the PowerPC line of microprocessors, the RAD6000 microprocessor [2] was created in 1995 in a 0.5 micron radiation hardened CMOS technology by modifying the circuitry for reliable spaceborne operation. This processor was quickly adopted and has been employed in many NASA missions as the flight computer, as well as the command processor in both generations of the Mars rovers. As shown in Figure 1, the RAD6000 central processor unit (CPU) includes both fixed point and floating point execution units in a superscalar RISC architecture executing up to three instructions per cycle. A two-way set associate unified 8 KB cache memory is also included. The existing microprocessor component is capable of operation at up to 33 MHz, with a throughput of 35 MIPS. The processor was originally developed using IBM's proprietary design languages and design tools. Separate buses are provided for I/O and memory. The RAD6000 is supported by the VxWorks real time operating system from WindRiver and a Green Hills C Compiler. Diagnostics are supported using the RAD6000's Common On-Chip Processor (COP) function.

When ported to a 150nm CMOS technology, the RAD6000 decreases in size by almost six times, making it quite viable for use as an embedded processor core. However, there were several steps required to bring that to fruition. The first step was to translate the logic design into VHDL to create a model compatible with industry-standard simulation tools that would ease simulation with other cores. The translated model was validated both with functional simulations and simulations of the original design's manufacturing test patterns. All existing Interfaces and the original Level Sensitive Scan Design (LSSD) latches were maintained in order to allow validation post-manufacturing using the current test patterns as well.

The COP function was enhanced with the addition of a slave interface to the On Chip Bus [3] to complement the existing off-chip interface for connection to existing RAD6000 diagnostic tools. This change will allow access to the COP from the JTAG cores as an alternative diagnostic approach.

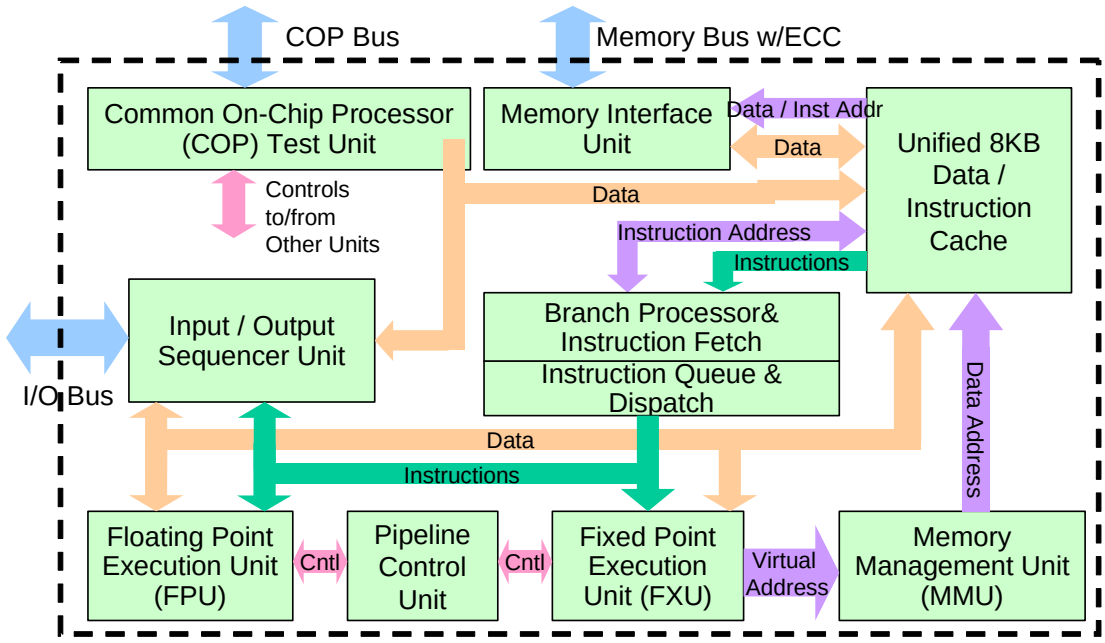


Figure 1: RAD6000 Microprocessor Architecture

The next important step was to develop an interface between the RAD6000 processor and the On-Chip Bus connection medium employed with all of BAE Systems' core-based designs. The original bridge ASIC developed for the standalone RAD6000 microprocessor was called the LIO ASIC. It was determined that while the key functions of the LIO were needed, a direct translation of the LIO into a core was not appropriate.

The Local Interface Function (LIF) core was designed specifically for this purpose. It provides direct connections to both the I/O bus and memory bus of the RAD6000, direct connection to the newest version of the external memory controller core, and three direct connections to the On-Chip Bus (OCB). The LIF is implemented as a set of sub-cores as shown in the block diagram in Figure 2, connected by a cross-bar switch to minimize contention for its many interfaces. In addition to the typical master and slave interfaces to the OCB, the LIF features a second slave interface designed to allow other cores to access external memory without interfering with the operation of the microprocessor.

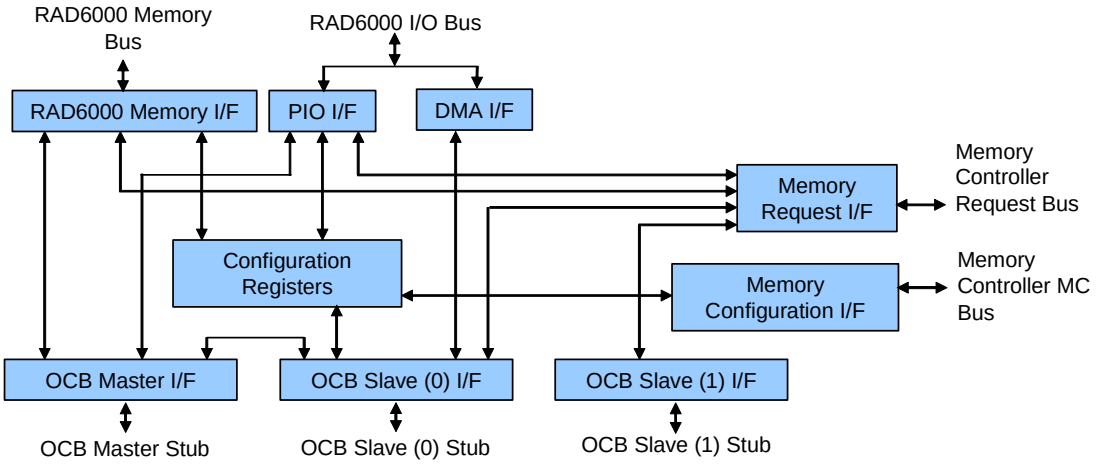


Figure 2: Local Interface Function Core Block Diagram

The two slave interfaces are used to partition the address space, which has been massively expanded beyond that supported by the standalone processor chip. The memory address bus of the RAD6000 was 27 bits, corresponding to 128 MB of addressable memory. The LIF extends that to 32 bits through address translation, as shown in Figure 3. The uppermost three bits are used to select one of eight 16 MB pages whose base address is stored in base address registers. The memory address map is defined to provide a total of 2 GB of RAD6000 address space. The other 2 GB of the 4 GB total address space is dedicated for use by the other cores that reside on the On Chip Bus.

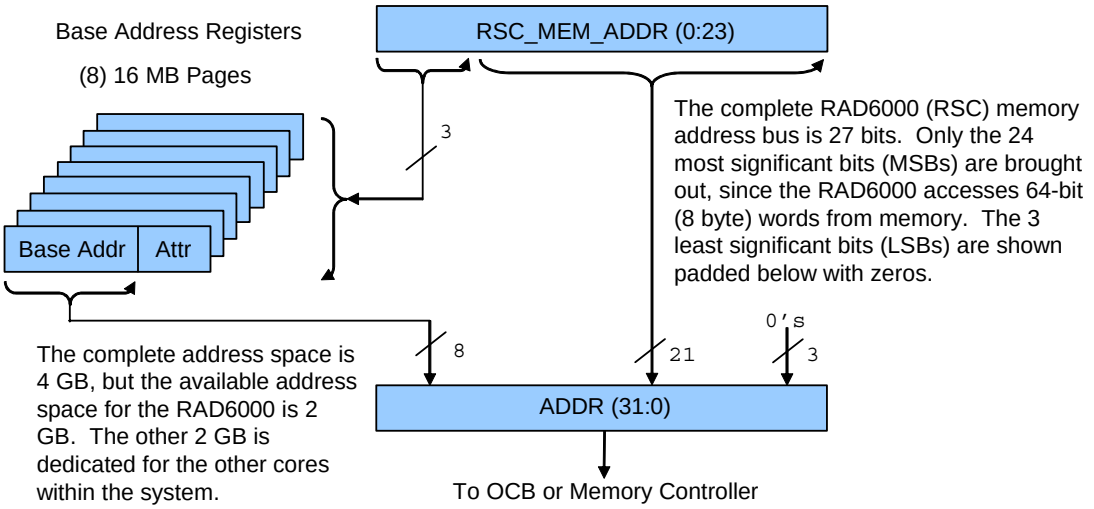


Figure 3: RAD6000 Memory Address Translation

In the LIF, only the first OCB slave (0) interface is used for the 128 MB of memory that corresponds to the original address space. When the memory access from the OCB falls within the original 128 MB RAD6000 memory address space, the interface accesses the base address registers and supports cache snooping if required. This operation is shown in Figure 4. The second OCB slave (1) interface does not support the original 128 MB of memory. It has no connection to the cache and is designed to support high speed direct access of memory by other cores through the OCB.

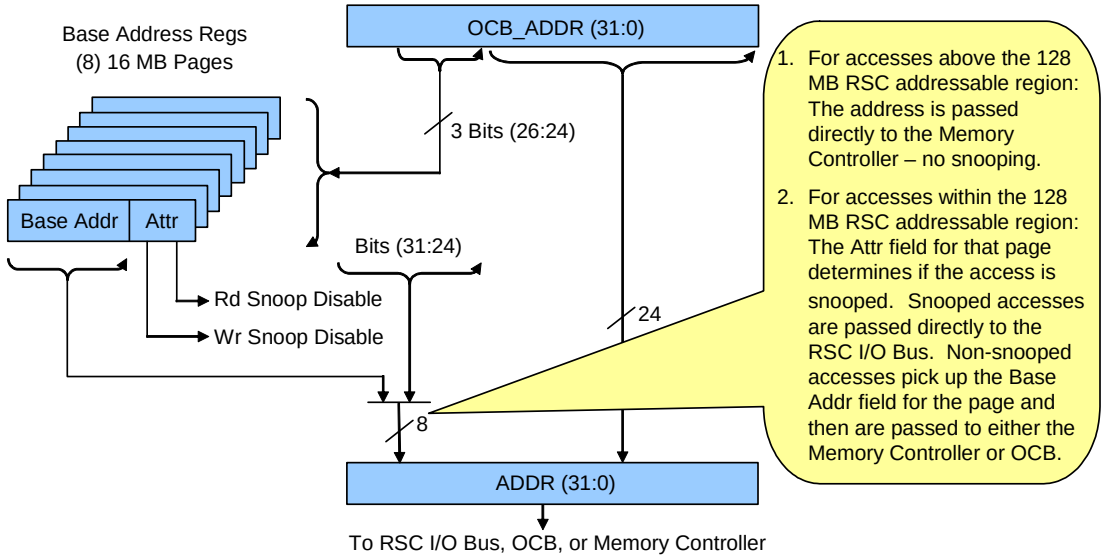


Figure 4: OCB Slave Address Translation

External memory is accessed by a memory controller core connected to the LIF. The core provides for two different types of error correction, single bit correction with double bit detection (SECDED) and nibble-level error correction. It can support a combination of SRAM, SDRAM, and non-volatile memory.

ON-CHIP BUS CONNECTION MEDIUM

The On-Chip Bus (OCB) is the standard connection medium used with all of BAE Systems' reusable cores and is the central element of the reusable core architecture [3]. It consists of two data buses configured as a crossbar switch that prevents contention between cores unless multiple cores are trying to access the same destination simultaneously. The high speed data bus is 64 bits and the low speed bus is 32 bits, connected by an internal OCB bridge. The higher speed data bus includes word level parity. The lower speed bus supports word, half-word, and byte parity. There is a separate 32-bit address bus, also with parity at the word level.

Connection between the OCB and reusable cores is provided by standard Master and Slave OCB "stub" logic. An OCB bus transaction consists of an address phase that begins with a request issued by the Master stub, followed by a data phase once the Slave has acknowledged the request. Once the request is acknowledged, optimum bus utilization is achieved through the use of address and data pipelining. In the OCB configuration for the RAD6000MC microcontroller ASIC, the higher speed bus has 11 master and 9 slave stub connections. The lower speed bus includes a single master with 9 slaves. This is shown on the RAD6000MC block diagram in Figure 5.

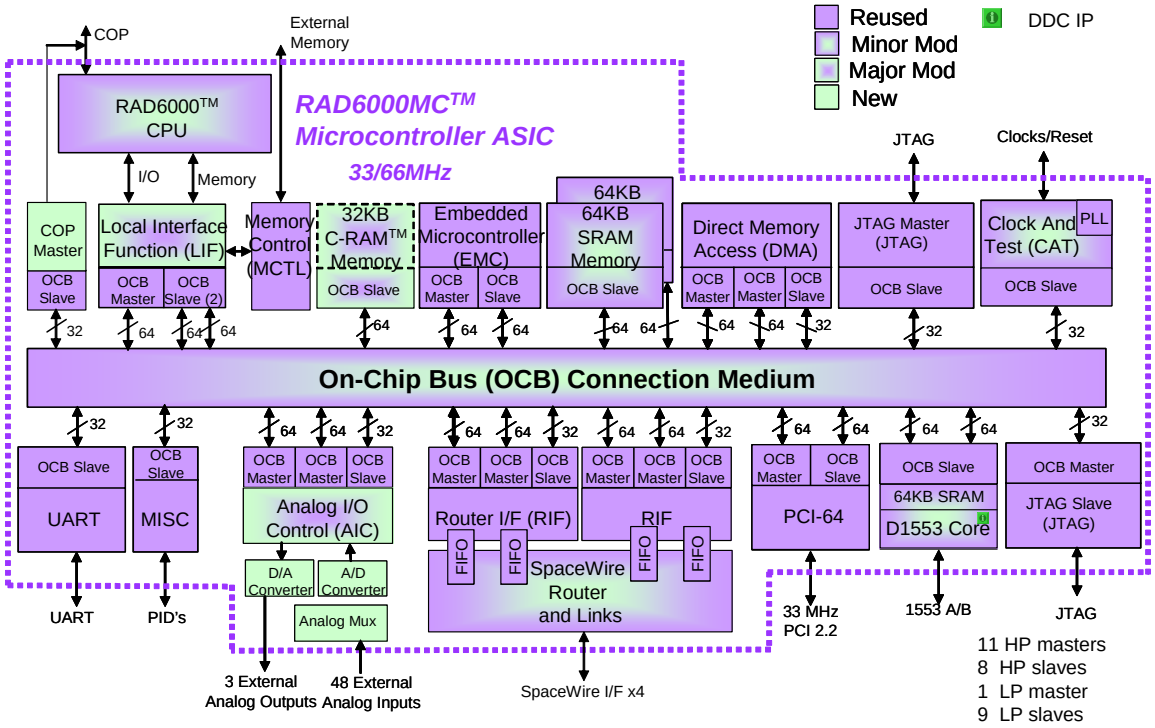


Figure 5: RAD6000MC ASIC Block Diagram

DIGITAL BUS INTERFACES

The RAD6000MC supports three digital bus interfaces that offer the user tremendous flexibility in implementation. For high reliability serial interfaces with low throughput requirements, a MIL-STD-1553 interface is supported. This interface is based on a synthesizable core from Data Device Corporation (DDC), a well-known provider of certified 1553 components. It supports Bus Controller and Remote Terminal (BC/RT) functions and provides dual redundant links. The 1553 interface includes a dedicated 64 KB SRAM memory block. If the 1553 interface is not being used, the user can treat the 64 KB SRAM as a third block of memory usable by the RAD6000 microprocessor or any of the other cores.

The SpaceWire interface is based on a router and serial link design originally developed for NASA Goddard Space Flight Center (GSFC), with features specific to the BAE Systems implementation currently in production in the BAE Systems SpaceWire ASIC [4]. The router includes four external links and two internal interfaces to the OCB via dual Router Interface (RIF) cores that minimize the risk of internal bottlenecks, as shown in Figure 6. The links have integrated LVDS drivers and receivers with cold sparing support. A unique bypass mode allows a fixed routing assignment, improving throughput in cases where the routing location is always known. A maximum packet length feature minimizes the chance of network stalling. This second generation version of the core also decreases the die area required for the router and operates it at a higher clock rate, improves performance of the time code function, and operates at a higher link rate (300 Mbps) when implemented in 150nm CMOS technology.

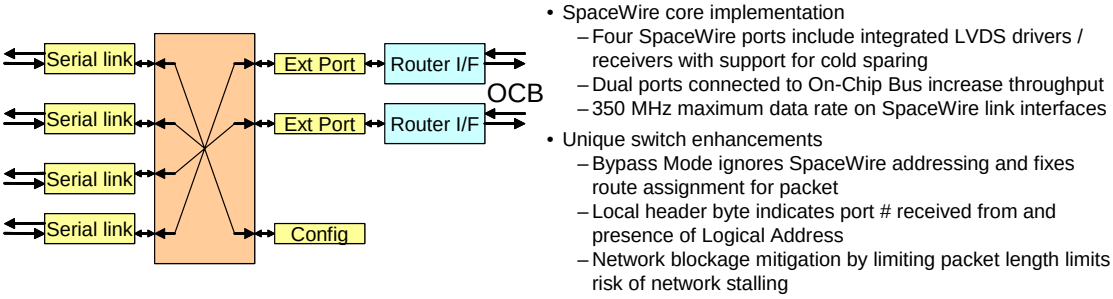


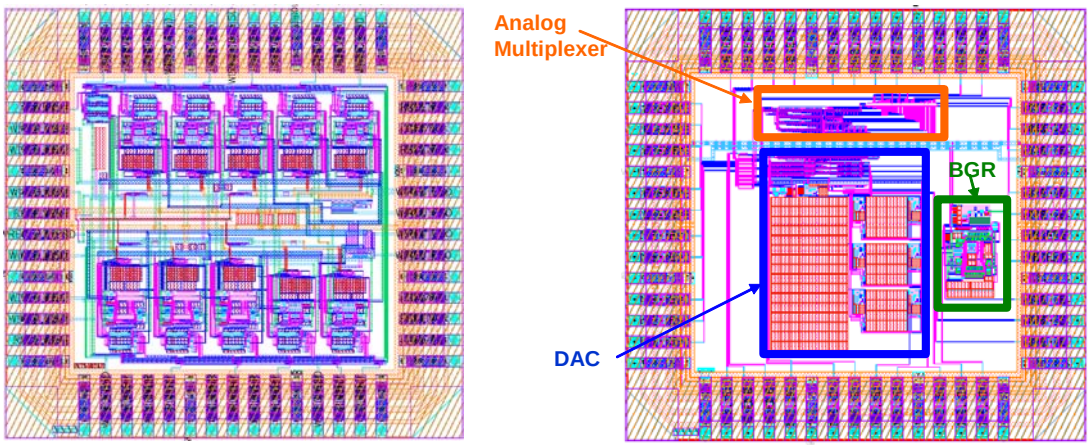
Figure 6: SpaceWire Links and Router With Dual On-Chip Bus Interfaces

High throughput parallel data transfers are supported by a 64-bit Peripheral Component Interconnect (PCI) bus (version 2.2) interface that can operate at up to 66 MHz on a board or at 33 MHz across a CompactPCI backplane. Bus throughput @ 33 MHz is up to 264 MB per second. The PCI interface includes word level parity on the 32-bit address bus and on both words of the 64-bit data.

ANALOG CIRCUITS AND INTERFACE CORE

The RAD6000MC includes a 12-bit Analog-to-Digital converter (ADC) operating at 8.25 Msps, with a matching programmable analog multiplexer and differential output Bandgap Reference (BGR) circuit. The ADC implements a 1.5-bit per stage pipeline architecture that is capable of sampling rates up 10 Msps at full resolution. It accepts a 4 volt, fully differential input. A high accuracy sample and hold circuit is incorporated into the ADC. The circuit includes a folded-cascode amplifier with four high-gain Operational Transconductance Amplifiers used as regulation amplifiers to

enhance gain and minimize propagation of errors through the stages of the ADC. A digital error correction algorithm is employed so that comparator errors up to 500 mV can be tolerated.



Pipeline A/D Converter test chip

Wilkinson D/A Converter, analog multiplexer, and Bandgap Reference test chip

Figure 7: Analog/Digital Converter and Digital/Analog Converter Test Chip Layouts

The entire ADC function was designed with 3.3 volt transistors. This provided for the highest dynamic range on analog inputs and resulted in lower power dissipation, rated at 200 mW. If the ADC is not being used, bias to the sample and hold amplifiers can be disabled via a small PMOS transistor within the current mirror of the A/D converter's current bias circuit to reduce power. The ADC has been manufactured on a test site and hardware is currently being tested. The circuit occupies 4.4 square millimeters (sq. mm.) of silicon, as shown in Figure 7.

The accompanying programmable analog multiplexer has been designed to support up to 48 single end inputs, up to 24 differential inputs, or limited combinations of both (8 differential and 32 single ended or 16 of each) to provide users with maximum flexibility. Area of the multiplexer circuit is 0.75 sq. mm.

The Bandgap Reference (BGR) circuit provides the differential reference voltage required by the ADC. Designed with continuous-time common-mode feedback, the BGR is very stable across the mil-spec temperature range varying by only 12 mV in simulations. Estimated power dissipation is 48 mW and circuit area is 0.27 sq. mm.

The D/A converter circuit is based on a novel Wilkinson architecture that dissipates only 5.5 mW at a 1 Ksps conversion rate and occupies a moderate silicon area of 1.54 sq. mm. while providing three channels of 12-bit resolution. The Wilkinson architecture minimizes the area increase for multiple channels by employing a shared ramp generator and pipelined Gray code digital counter with a 4.125 MHz clock. The track and hold circuits for each channel track the ramp generator voltage until it receives a pulse sent from the matching comparator. The track and hold amplifier then holds that analog value until the next voltage conversion.

All three of these circuits have been released on a test chip that is currently in manufacturing, shown in Figure 7. Hardware is expected by the end of 2007.

Connection of the analog circuitry to the OCB is achieved with the Analog I/O and Control (AIC) reusable core. This function provides the clocks and control signals required by all of the analog circuits. A set of six select bits is used to program the multiplexer.

EMBEDDED C-RAM NON-VOLATILE MEMORY AND ON-CHIP SRAM MEMORY

The 32 KB chalcogenide-based C-RAM macro is the first implementation of an embedded C-RAM core, adapted from BAE Systems' 4 Mb, C-RAM non-volatile memory chip [5]. The principle behind chalcogenide-based non-volatile memory is that of phase change. Current is applied to change a small quantity of chalcogenide material that represents the memory bit storage node between the amorphous and crystalline states, thereby achieving "0" and "1" stored values. Organized as 1,024 words by 32 bits, this core was sized specifically to provide built-in start-up ROM functionality for the RAD6000 that requires 26 KB of memory. The core is currently in manufacturing and testing will be performed late in 2007. A version of the macro with embedded error correction code (ECC) is now being designed to enhance manufacturing yield.

There are two 64 KB SRAM memory macros directly connected to the OCB. Splitting them allows multiple cores to access blocks of memory simultaneously across the OCB, taking advantage of the crossbar switch architecture. A third 64 KB SRAM resides within the 1553 core, and is dedicated to that interface when it is in operation. However, when the 1553 interface is not required, the user can reallocate it as a third generally available on-chip memory core.

EMBEDDED MICROCONTROLLER

In addition to the RAD6000, the microcontroller ASIC includes a smaller processor called the Embedded Microcontroller (EMC) [6]. The EMC is a synthesizable, 32-bit RISC processor with moderate throughput, based on a BAE Systems architecture and instruction set. It incorporates a 2 KB instruction cache and is supported its own C compiler, also developed by BAE Systems. Executing a single fixed point instruction per cycle, the EMC is employed to control reset sequences for the RAD6000, for housekeeping functions, and for support of some reusable cores. As an example of this, code has already been written in support of the SpaceWire interface, creating descriptors to partition large data files to provide a more manageable downlink.

CONFIGURABILITY FOR VARIOUS APPLICATIONS

The RAD6000MC will be configurable as required for various applications through a series of personalizations. RAD6000 processor clock speed will be selectable for either 33 or 66 MHz, emphasizing throughput vs. power dissipation. Clocking to the various interface cores may be gated to decrease power dissipation, if they are not required for a specific application; although some functions must always be operational. As mentioned above, one 64 KB memory macro may be used either dedicated to the 1553 bus or made available for general use. Additional external memory may be added where required.

Shown in Figure 8 in an example of a specific operation as part of a larger distributed computing system, the microcontroller is operating a robotic arm in a system that is

based on distributed computing. In this example, communication and data transfers occur across the SpaceWire network. Analog control of the joints of the arm would be achieved through use of the D/A converters. Tactile feedback via sensors would be transmitted through the multiplexer and A/D converter. As a subsystem with limited functions, a robust operating system is probably not required, so a microkernel can be employed. This likely eliminates the need for external memory, making the microcontroller completely self-contained. Because the 1553 interface is not being employed, a third 64 KB SRAM block is freed up for use in this application. System boot would be performed from the on-chip non-volatile C-RAM memory.

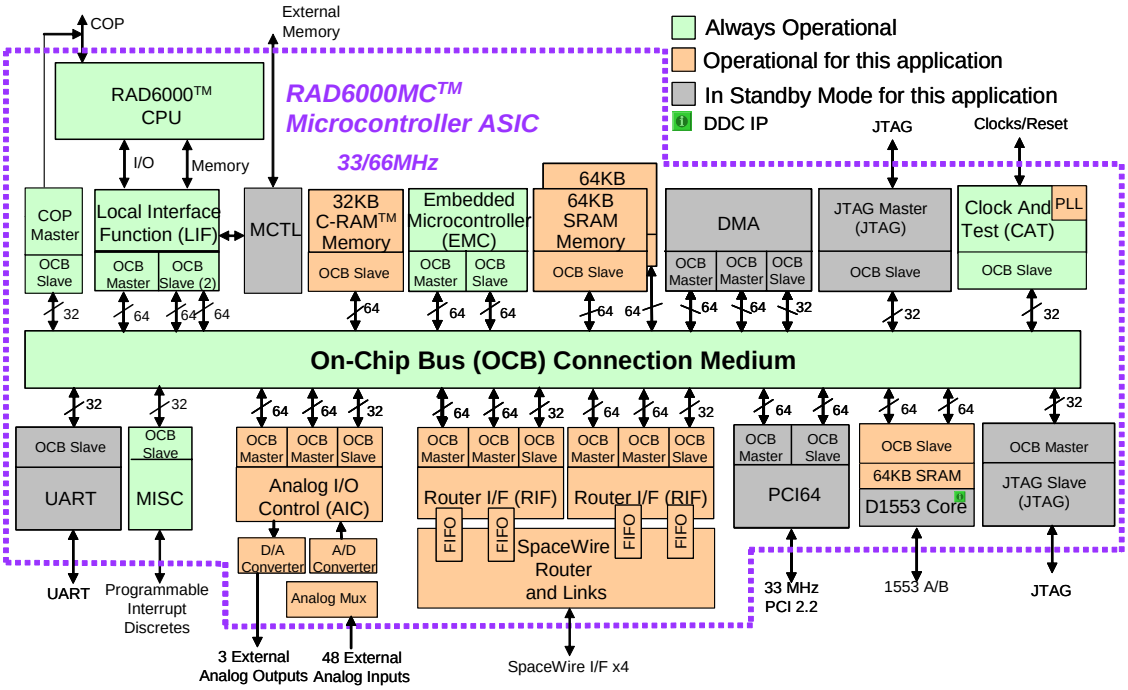


Figure 8: Example RAD6000MC Configuration - Robotic Arm Control

In a minimal configuration running at 33 MHz, the RAD6000MC is expected to dissipate about 2 Watts. With the full configuration operating at 66 MHz, estimated power dissipation is roughly 6 Watts.

SUMMARY

A highly flexible microcontroller ASIC is being designed in a 150nm radiation hardened CMOS technology. It is based on the RAD6000 microprocessor and incorporates a variety of digital and analog cores. Many of the digital cores have already been demonstrated in other ASICs, and the analog circuitry is being manufactured in test chips to provide early hardware validation.

Power and PowerPC are registered trademarks of the IBM Corporation.

RAD6000 and RAD6000MC, and C-RAM are trademarks of BAE SYSTEMS.

REFERENCES

[1] C.R. Moore, et. al., *"IBM Single Chip RISC Processor (RSC)"*, IEEE 1992 International Conference on Computer Design, pp. 200-204

[2] N. Haddad, et. al., *"Radiation Hardened COTS-based 32-bit Microprocessor"*, GOMAC 1998 Digest of Papers and Journal of Radiation Effects Research and Engineering, Vol. 17, Number 1, April 1999.

[3] J. Marshall, et al, *"Application of Reusable Cores to System-on-a-Chip"*, 2001 Government Microcircuits Applications Conference (GOMAC), March 2001

[4] R. Berger, et. al., *"A Radiation Hardened SpaceWire ASIC and Roadmap"*, 9th Military and Aerospace Programmable Logic Devices (MAPLD) International Conference 2006, September 26-28, 2006

[5] R. Ramaswamy, et. al., *"Progress on Design and Demonstration of the 4 Mb Chalcogenide-based Random Access Memory"*, Proceedings of the 2004 Non-Volatile Memory Technology Symposium, November 2004

[6] J. Marshall, and J. Robertson, *"An Embedded Microcontroller for Spacecraft Applications"*, IEEE Aerospace Conference 2006, March 2006

A ONE CHIP HARDENED SOLUTION FOR HIGH SPEED SPACEWIRE SYSTEM IMPLEMENTATIONS

Session: Components

Long Paper

Joseph R. Marshall and Richard W. Berger

BAE Systems, 9300 Wellington Road, Manassas, Virginia 20110 USA

Glenn P. Rakow

NASA-GSFC, Code 561, Greenbelt, Maryland 20771 USA

E-mail: Joe.Marshall@baesystems.com, Richard.W.Berger@baesystems.com, Glenn.P.Rakow@nasa.gov

ABSTRACT

An Application Specific Integrated Circuit (ASIC) that implements the SpaceWire protocol has been developed in a radiation hardened 0.25 micron CMOS technology. This effort began in March 2003 as a joint development between the NASA Goddard Space Flight Center (GSFC) and BAE Systems. The BAE Systems SpaceWire ASIC is comprised entirely of reusable core elements, many of which are already flight-proven. It incorporates a router with 4 SpaceWire ports and two local ports, dual PCI bus interfaces, a microcontroller, 32KB of internal memory, and a memory controller for additional external memory use. The SpaceWire cores are also reused in other ASICs under development. The SpaceWire ASIC is planned for use on the Geostationary Operational Environmental Satellites (GOES)-R, the Lunar Reconnaissance Orbiter (LRO) and other missions. Engineering and flight parts have been delivered to programs and users.

This paper reviews the SpaceWire protocol and those elements of it that have been built into the current and next SpaceWire reusable cores and features within the core that go beyond the current standard and can be enabled or disabled by the user. The adaptation of SpaceWire to BAE Systems' On Chip Bus (OCB) for compatibility with the other reusable cores will be reviewed and highlighted. Optional configurations within user systems and test boards will be shown. The physical implementation of the design will be described and test results from the hardware will be discussed. Application of this ASIC and other ASICs containing the SpaceWire cores and embedded microcontroller to Plug and Play and reconfigurable implementations will be described. Finally, the BAE Systems roadmap for SpaceWire developments will be updated, including some products already in design as well as longer term plans.

1. SPACEWIRE PROTOCOL AND USAGE

The SpaceWire protocol was defined in the European Cooperation for Space Standardization (ECSS) group as ECSS-E-50-12A. It grew out of the IEEE 1355 standard and earlier efforts to develop high performance point to point standard interfaces between computer nodes. With the advent of Low Voltage Differential Signaling (LVDS) devices being used in space, it has become possible to marry these physical elements with straightforward high performance logic and create space qualifiable devices that deliver the power and performance required.

The current SpaceWire standard defines the interface at the signal, physical, character, exchange, packet and network levels. Further standardization of higher and parallel sections

such as those needed for plug and play are in development. At the signaling level, data stream encoding is used to embed clocks and data across two signals in each direction. Each of these are physically transmitted using LVDS circuitry. Thus a total of eight logic wires and a ground are in each standardized cable, with 9 pin micro-D connectors on each end. At the character level, there are two types, control and data, each with embedded parity for fault detection. Control characters are used for flow control, packet completion and special combinations with data characters for the link. Data characters carry the message and time codes. Null characters are sent when no other character is available in order to maintain a link. Flow control, which manages the sending of characters when there is space at the receiver and detection of parity and disconnect errors, is handled at the exchange level. The packet level uses a destination header on the front and an end of packet marker on the back to encapsulate packets. At the network level, there are two addressing modes defined – path addressing and logical addressing. Path addressing concatenates the path as a series of addresses, each one stripped off as the packet traverses a SpaceWire network. Logical addressing maintains a lookup table and a global addressing scheme across a SpaceWire network.

SpaceWire links provides a scalable full duplex communications that may be expanded through the use of routers to most any size network. Three typical uses of the SpaceWire in a spacecraft are shown in Figure 1. A Control Tree has one CPU (or two with redundancy) performing the normal C&DH functions and controlling and sending and receiving data from several instruments and other subsystems. The 4 port router block shown provides direct connections between the CPU and three additional instruments. If more instruments or subsystems were present, the router could be expanded to more ports, a second router added and daisy-chained or routers could be added to some instruments. We will discuss router configurations further later in this paper.

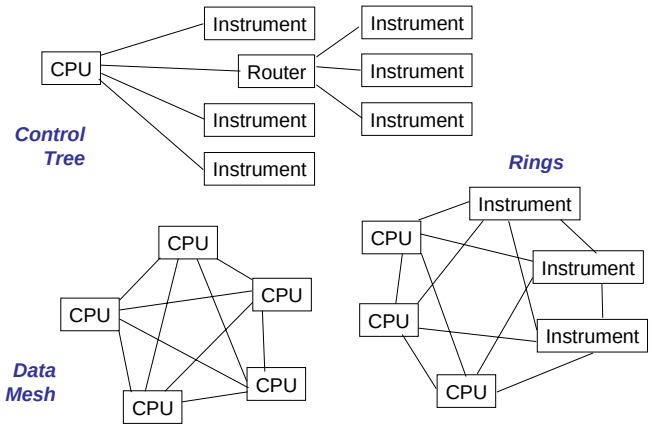


Figure 1: Typical SpaceWire Topographies

Data meshes are used for pure communications between a set of peers. If a CPU has four SpaceWire ports, it can take part in a five CPU mesh and have direct connections with four others. If more CPUs / peers are in the system, routers could be used to expand the connectivity either within the CPUs or as separate switches. In the Ring topology, one or more CPUs have connectivity with several instruments or other subsystems and they are connected by rings. In this picture, six nodes are connected by an outer ring and three nodes each are connected to each other by sub-rings. These could also be done as a second outer ring. This structure is easily expanded simply by adding nodes and the fault tolerance remains the same.

The SpaceWire interface is constantly being compared to other interfaces for potential space usage. Table 1 shows a comparison between SpaceWire (*based on BAE's implementations) and three other interfaces it may be used to replace. With its higher potential data rate per network, SpaceWire offers the most scalability and flexibility in building high performance networks.

Standard	SpaceWire	MIL-STD-1553B	1394a	cPCI
Attribute				
Width	Serial	Serial	Serial	32/64
Topology	Point to Point	Bus	Point to Point	Bus
Maximum Frequency	10-264 MHz*	1 MHz	100-400 MHz	33-66 MHz
Maximum Data Rate/Node	212 Mbps*	0.500 Mbps	400 Mbps	1056-4224 Mbps
Maximum Data Rate/Network	27136 Mbps*	0.500 Mbps	400 Mbps	1056-4224 Mbps
Maximum Nodes	256 (Unlimited)*	32	63	8 to 4
Isolation between Nodes	LVDS	Transformer	Galvanic	Resistor
Node Redundancy	Full Port	PHY Only	PHY Only	Device

Table 1: Standards Comparisons

2. SpaceWire Core Design and ASIC Integration

In March 2003, a joint development effort began between BAE Systems and NASA Goddard Flight Center (GSFC) to join the GSFC SpaceWire IP with the BAE Systems core-based ASIC technologies[1]. The GSFC design was first realized as an FPGA design and implemented in the NASA Swift mission and then upgraded for the James Webb Space Telescope mission. This design was captured in VHDL and converted to a core for BAE Systems R25 CMOS Technology. The core contains four SpaceWire ports and two external ports and a six port routing switch. LVDS physical drivers and receivers are built into this core and mapped onto BAE Systems LVDS I/O circuits in the R25 technology. These LVDS circuits are capable of 600 MHz operation, however the combined SpaceWire circuit and link budget supports full protocol speeds to 264 MHz. 32-bit wide, 64 deep FIFOs are utilized for each transmit and receive function.

Figure 2 below shows the internal blocks of SpaceWire ASIC.

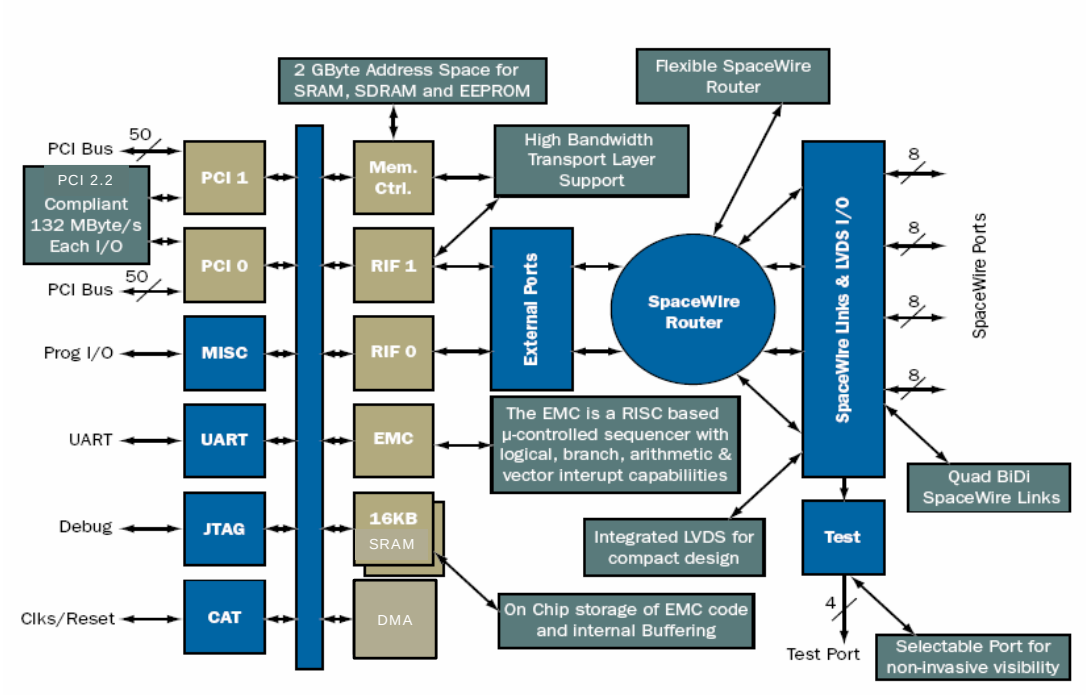


Figure 2: SpaceWire ASIC Block Diagram

The elements in blue on the right make up the SpaceWire core provided by GSFC. The external ports in the SpaceWire core provide the connection into the rest of the SpaceWire ASIC. These are connected by two Router Interfaces (RIF) into the cross-bar switch connection fabric within

the ASIC. This connection medium is termed the On Chip Bus (OCB) and is shown as the blue bar between all the square cores in the figure. Each RIF contains DMA controllers in each direction and may connect to any other core on the OCB. The other cores shown are all reused from other BAE Systems programs and include the Peripheral Component Interface (PCI), Miscellaneous core containing programmable I/O discretes (MISC), Universal Asynchronous Receiver Transmitter (UART) I/F for debug and low speed data transfers, Joint Test Action Group (JTAG) Interface, Direct Memory Access (DMA) usable by any other intelligent core for independent transfers, a Memory Controller that may access external SRAM, SDRAM or EEPROM (Mem Ctrl.), an Embedded Microcontroller (EMC), two 16 KB SRAM cores for buffering data from any core or interface on the OCB or storing programs for the EMC and the Clock and Timer (CAT) core. Only the CAT and MISC core were changed for the SpaceWire ASIC design, maximizing the reuse and minimizing the design risk.

The OCB is made up of 32 bit and 64 bit connections. All of the lighter (tan) blocks shown in the figure above are 64-bit and have connectivity to all other blocks. The darker (blue) blocks are 32 bit, low performance and connect through a single 64 to 32 bit bridge to the rest of the OCB 64 bit fabric.

3. SPACEWIRE ASIC IMPLEMENTATION

The SpaceWire ASIC is implemented in BAE Systems R25 Technology[1]. It is packaged in a 624 pin 32.5mm Ceramic Column Grid Array (CCGA) package. It operates with a core voltage of 2.5V and an I/O voltage of 3.3V. The design uses 423 of the 512 available signal pins. The ASIC utilizes five sets of clocks that may be similarly derived or unique – one drives the ASIC system clock, one drives the internal logic of the ASIC, one is used for the PLLs that set the speed for the SpaceWire ports and two are used for the PCI cores. The PLL may be user selectable to 1x, 1.5x, 2x, 2.5x, 3x, or 3.5x by configuring the clock and timing core. This is typically done with EMC code. The clocks so generated may then be divided by each port for individual port transmission speeds and provide a significant flexibility in configuring and using each port to best match its data bandwidth needs.

Any use of the SpaceWire ASIC includes analyzing the paths through the ASIC. Because of the cross-bar nature of the OCB, multiple transfers may be going simultaneously as long as they do not collide at a destination block. Thus, data may be moving through two SpaceWire ports into/out of external memory and onto/off of a PCI Bus at the same time that the EMC is executing a program and working out of internal memory, the DMA controller is moving data from JTAG onto the other PCI interface, and data is moving through the router between the remaining two SpaceWire ports. Such a maximum configuration is shown in Figure 3. A more typical operation would have some conflicts and the key analysis is latency and waiting for an interface or memory to become unblocked. External memory is especially useful for buffering data and then processing or moving it out on an interface once it has become available.

The EMC[2] is a special block and one that provides a significant processing capability for the SpaceWire ASIC. The third generation of this BAE Systems-designed core is used in the SpaceWire ASIC. It is able to execute instructions at the internal ASIC speed and includes a small instruction cache for tight loop high performance. The EMC is able to access any core on the OCB and thus can talk on any interface. Its instructions are typically stored in a non-volatile memory such as EEPROM or BAE Systems' new chalcogenide-based C-RAM and then copied into the internal SRAM on the ASIC for execution. An assembler, linker, C Compiler and simulator are all available for EMC code development.

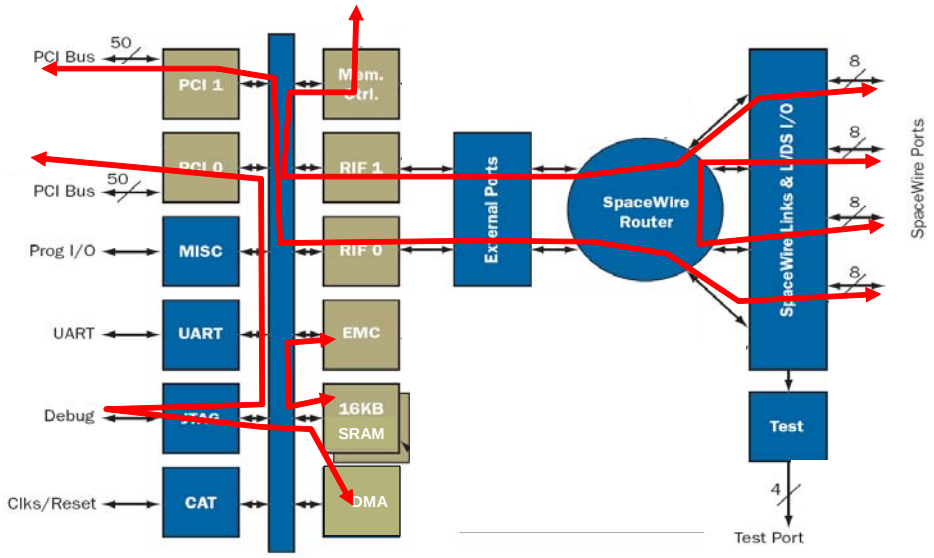


Figure 3: Simultaneous Paths Through the SpaceWire ASIC

The EMC uses a straightforward instruction set and has an execution structure shown below which is simple and elegant compared to other more complex microcontroller cores. Using a 66 MHz clock, it can execute at around 10 Dhrystone MIPS. This provides a significant adjunct processing capability and has been used for startup and configuration as well as post processing of packets sent to the SpaceWire ASIC. The EMC block diagram is shown in Figure 4.

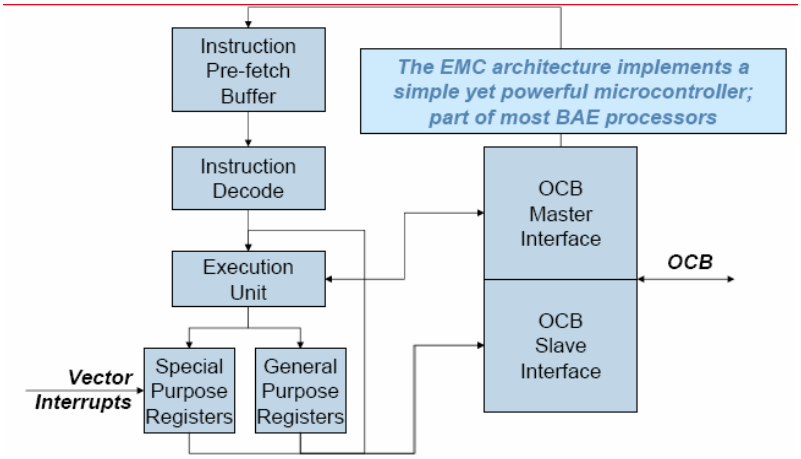


Figure 4: Embedded Microcontroller Core Block Diagram

The SpaceWire ASIC is built out of R25 Technology and is thus power efficient for this technology. It is radiation hardened to 200 Krads(Si) total ionizing dose, has an single event upset rate of less than 1E-9 errors/bit-day, and is latch-up immune. The SpaceWire ASIC has been measured to draw between 1 and 2.5 Watts in use to date at speeds up to 132 MHz. It is expected that if all four ports are being used as wells as most interfaces at full speeds, the typical power would peak around 4 watts.

4. SpaceWire ASIC Usage and Configurations

The SpaceWire ASIC has been used on two different boards for the NASA Lunar Reconnaissance Orbiter mission due to be launched in 2008. Both are single board computers

and include BAE Systems RAD750 processor, memory, other interfaces and functions. Four ports are brought out from each and used for connectivity in the system. One is built in a standard 6U-220 CompactPCI form factor and the other to a more custom slice form factor. The LRO single board computer is described in much more detail in a separate paper[3]. Both types of flight boards have been space qualified and delivered for integration.

The SpaceWire ASIC has also been delivered to other companies of use on the Geostationary Operational Environmental Satellite (GOES-R) and is available for use in other spaceborne applications as a standard product. In another application, the SpaceWire ASIC has been used as a standalone control chip for the Universal FPGA Support Device (UFSD) Test Board[4], a 6U-220 CompactPCI reconfigurable computing board utilizing Xilinx RAM-based FPGAs for space. One of the PCI Busses is brought to the backplane for connection to other CompactPCI boards while the second PCI bus is connected to one or more FPGAs on the board. The EMC in the SpaceWire ASIC is the master microcontroller on the board and is used to setup and configure the board. A second EMC is contained in the FPGA Resource Enabling Device (FRED) ASIC that manages the array of FPGA devices, the FPGA Resource Enabling Device. Both utilize BAE Systems new C-RAM non-volatile memory on the board to save program and configuration information. The SpaceWire ASIC provides two levels of system interfaces for the board – CompactPCI and four SpaceWire ports.

BAE Systems is developing a customer evaluation board based on these various board designs. A block diagram is shown below in Figure 5. At its center is the SpaceWire ASIC with onboard EEPROM to store configuration information and code for the ASIC's EMC and SRAM for buffering messages as well as scratchpad memory. Only one PCI interface will be utilized and this will form an external CompactPCI interface for the card. The card will run from 5V and 3.3V supplies on the backplane and develop the 2.5V on the card. It will have options to use an onboard oscillator for its clock sources or an external clock source from the backplane. Finally, the JTAG, UART, various discretes and a SpaceWire sniffing port will be available on a test connector.

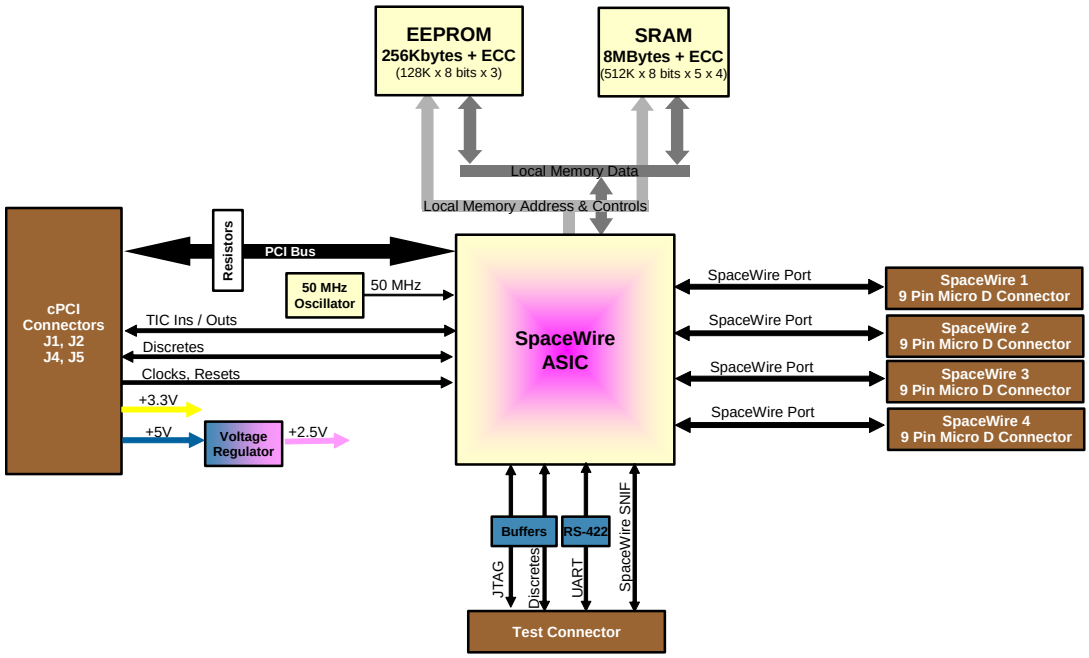


Figure 5: Four Port CompactPCI 6U SpaceWire Evaluation Board Block Diagram

The board will be implemented on a 6U-160 Compact PCI form factor. Each of the SpaceWire connectors will use the standard 9 pin micro-D connectors. A mock-up of this board is shown in Figure 6 below. Since there is so much extra space on the board, it will be very easy to layout and produce. For an actual space product, it is expected that more optional memory or other functions will take advantage and fill the board.

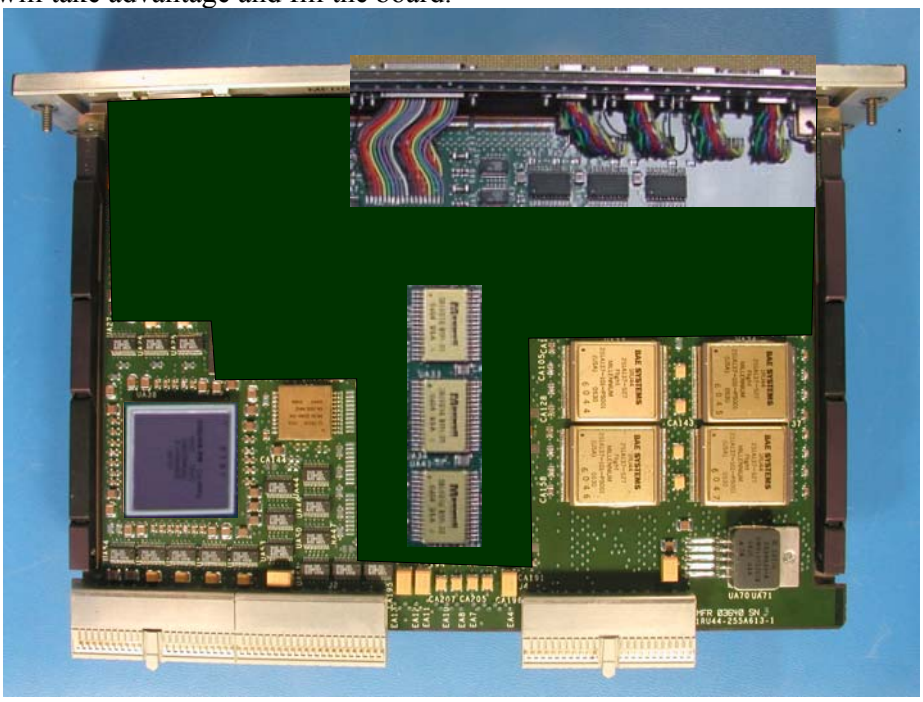


Figure 6: SpaceWire Evaluation Board Mock-up

The SpaceWire ASIC has been successfully integrated into the boards described above. Due to the high amount of reuse from both GSFC and BAE Systems, very few design problems were expected or encountered and all errata have workarounds. Potential users may design in this stable device or board into their products. The SpaceWire core elements are part of two new ASICs under development at BAE Systems, the RAD6000MC[4][5] and next generation Power PCI bridge ASIC. Both of these will provide four SpaceWire ports and appropriate internal routers to other assets within the device.

As the topologies shown earlier illustrate, there is a need for larger numbers of ports in more complex systems that require any to any switching. The dual PCI busses allow up to 8 SpaceWire ASICs to be placed on a single assembly and provide up to 32 ports with the PCI Busses used for passing data between ports not on the same device. However, these PCI busses may become bottlenecks in a maximum performance case and add latency to the routing function since each is a shared bus. For maximum performance, designs may use some of the SpaceWire ports for connections between external ports and group SpaceWire ASICs onto groups of PCI Busses. An example of such an implementation is shown below in Figure 7 as a 10 port SpaceWire Router using four SpaceWire ASICs.

There are actually three PCI busses shown. One PCI Bus is used only to connect the router to the external world through SpaceWire ASIC 1, probably through a connector. A common PCI bus is routed between all four SpaceWire ASICs and a third PCI Bus is routed between SpaceWire ASICs 2, 3 and 4. SpaceWire ASIC 1 utilizes three of its ports to connect directly to SpaceWire ASICs 2, 3 and 4, providing an alternate routing path between ports not requiring translation to PCI and back. This approach would provide at least 100% more routing performance when compared to a three ASIC solution providing 12 ports. Additionally, BAE

Systems also expects to expand its ASIC solution to between 8 and 24 ports directly with appropriate internal routers as customer needs dictate.

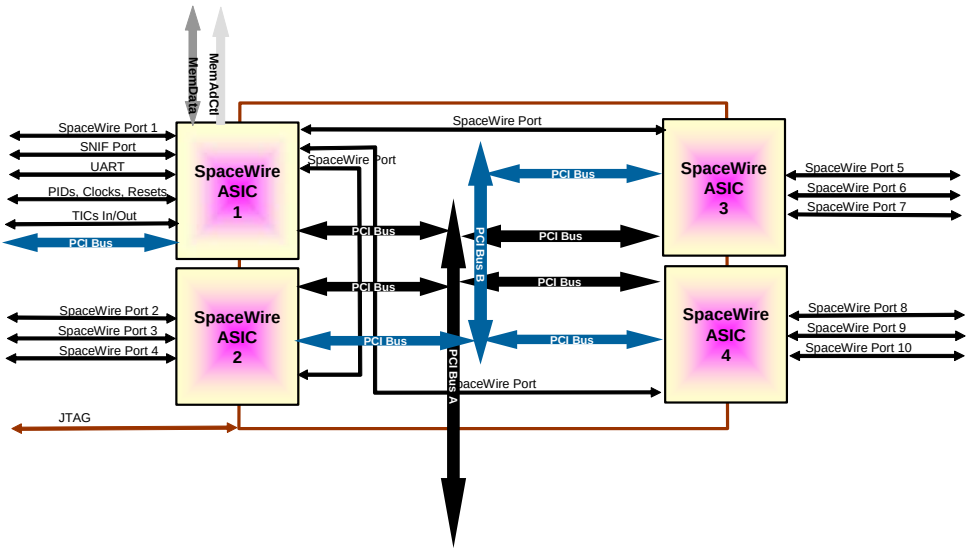


Figure 7: Ten Port Highest Performance SpaceWire Router

To support the SpaceWire ASIC, BAE Systems has developed application programming interfaces (API) for VxWorks. These may be ported to other operating systems. We have also created startup code (SUROM) that configures the SpaceWire ASIC and simple communication code for using the EMC after startup as data is flowing through the ASIC - both may be used as a basis for future applications.

SpaceWire has been identified as one of the key interfaces for the Space Avionics Plug and Play (SPA) standards for responsive space and standardization activities are underway to add appropriate layers and capabilities to the standard to support self discovery, configuration and utilization in such systems. It is very important that existing devices such as the SpaceWire ASIC are able to participate in such standards and BAE Systems and GSFC are part of these standardization efforts. The UFSD Test Board mentioned earlier is being utilized as a potential test vehicle for testing these standards as they become defined.

5. SpaceWire Roadmap

Throughout this paper we have discussed various BAE Systems products for SpaceWire focused around the SpaceWire ASIC. A product roadmap is shown below in Figure 8. At the top is the evaluation board. This is followed by the two LRO SpaceWire RAD750 boards. Next, we show how high speed multiple gigabit per second SERDES cores may be utilized for a higher speed version of SpaceWire followed by the RAD6000MC microcontroller. A potential 24 port ASIC and the next generation Power PCI Bridge round out new offerings while the actual current SpaceWire ASIC is shown on the bottom line. As this shows, BAE Systems has both current products and plans for future improvements to make the use of SpaceWire more cost effective and higher performance.

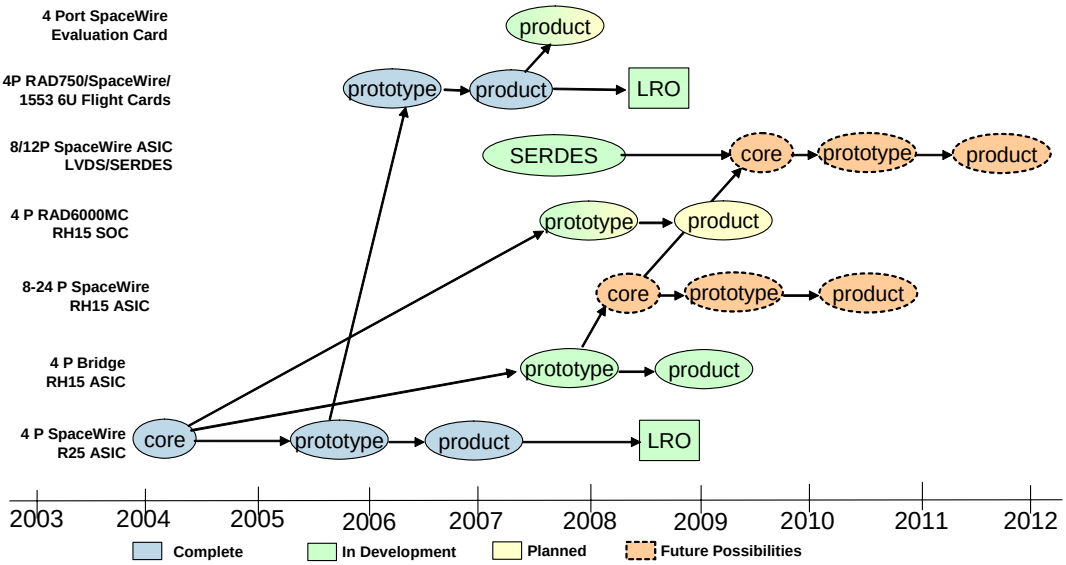


Figure 8: SpaceWire Product Roadmap

6. SUMMARY

BAE Systems along with GSFC has developed a high performance SpaceWire ASIC and successfully applied and qualified it initially to GOES-R and LRO missions and made it part of its future high performance micro-controllers, processors and reconfigurable systems. This device supports multiple topologies and form factors and is cost and power effective for high performance and scalable applications. BAE Systems is developing an evaluation board that will make it easier for its customers to utilize this state of the art Spacecraft product. It is positioned for use in upcoming plug and play and other space applications and is fully supported by support and development software and tools.

7. REFERENCES

[1] Berger, Richard W. et. al., "A Radiation Hardened SpaceWire ASIC and Roadmap", *2006 Military Applications of Programmable Logic Devices (MAPLD) Conference Proceedings*, Washington, DC, September, 2006.

[2] Marshall, Joseph R. & Robertson, Jeffrey, "An Embedded Microcontroller for Spacecraft Applications", *IEEE Aerospace Conference 2006 Proceedings*, Big Sky, Montana, March 2006.

[3] Berger, Richard W. et. al., "RAD750 SpaceWire-Enabled Flight Computer for Lunar Reconnaissance Orbiter", *2007 International SpaceWire Conference*, Dundee, Scotland, September 2007.

[4] Marshall, Joseph R. et. al., "Reconfigurable and Processing Building Blocks in Responsive Space", *2007 Infotech@Aerospace Conference Proceedings*, Rohnert Park, California, May 2007.

[5] Berger, Richard W. et. al., "A System-On-Chip Radiation Hardened Microcontroller ASIC With Embedded SpaceWire Router", *2007 International SpaceWire Conference*, Dundee, Scotland, September 2007.

Time access service for OS Linux with SpaceWire (MCB-01.2.Fpga, MCB-01.2.Asic) bridges support

Session: SpaceWire Components

Short paper

Razzhivin Dmitriy

Actually organization “The Consultative Committee for Space Data System” (CCSDS) develops standard “Spacecraft onboard interface services – time access service” (872.0-R-0.3).

Time access service (TAS) – this is the standard, that allows for many hosts to work with similar time. This makes programs run synchronously and planned. This is important for real-time onboard system.

Some primitive functions for work with time access service are defined in the standard:

1. TAS_TIME.request;
2. TAS_ALARM.request;
3. TAS_CANCEL_ALARM.request;
4. TAS_METRONOME.request;
5. TAS_CANCEL_METRONOME.request;
6. TAS_TIME.indicaion.

Our company MiT developed PCB SpaceWire MC-24EM, containing processor MC-24 and bridge MCB-01.2.Fpga. Linux OS kernel, including driver for bridge MCB-01.2.Fpga, is written to SDRAM of PCB.

Support of time access service is divided into two levels:

1. Kernel level. At this level exchange of timeCodes would take place at kernel space of all hosts of network. User space programs would work with system time of Linux OS.
2. User level. At this level accsess to timeCodes would be provided for user programs.

The following facts are advantages for the first method:

1. Exact time would be set for entire OS and all programs, running at in, but not only for one of them.

2. Information about time would be exchanged in kernel space of host of network only, transferring of this information to user space isn't needed. This will decrease time delays, needed for setting similar time in network.

Flexibility for user programs is the advantage of the second method. This may be useful if it is not enough for programs to work with system time of OS.

Additional Papers

TOWARDS THE DEFINITION OF QUALITY OF SERVICE CLASSES FOR SPACEWIRE-BASED MESSAGE PASSING

David Jameux, Albert Florit Ferrer

ESA/ESTEC, Keplerlaan, 1 / 2201 AZ Noordwijk ZH / The Netherlands

E-mail: david.jameux@esa.int, albert.ferrer.florit @esa.int

ABSTRACT

SpaceWire networks are now widely used in spacecraft and beyond but the implementations of SpaceWire-based communications are as many as the projects using SpaceWire because of a lack of standard message passing protocol for SpaceWire. In order to respond to the increasing pressure in this direction, ESA came up with consolidated ideas about the classes of Quality of Service that are required for any message passing protocol for SpaceWire to be useful an adopted by users and developers. This paper presents and justifies these classes of Quality of Service.

In this paper, we recall the general properties of digital communication protocols and select a set of these properties relevant to SpaceWire-based embedded systems in general and to space applications in particular, in the frame of the 'Single Fault' hypothesis. This leads to the definition of four classes of Quality of Service, two non real-time (minimum service and transport-like class) and two real-time (FTRT and maximum service class). We then provide some hints for the implementation of these QoS classes in the frame of SpaceWire networks. While the non real-time classes can clearly be implemented within the frame of the current SpaceWire standard, it appears that implementing real-time classes requires significant additional capability be embedded in some SpaceWire routers and nodes.

COMMON DEFINITIONS RELATED TO QOS

Quality of service, or QoS, in the field of telephony, was defined in the ITU standard X.902 as "a set of quality requirements on the collective behaviour of one or more objects". In the field of packet-switched networks and computer networking, the traffic engineering term Quality of Service refers to resource reservation control mechanisms. Quality of Service can provide different priority to different users or data flows, or guarantee a certain level of performance to a data flow.

The needs of a data flow can be characterized by four primary parameters: reliability, delay, jitter, and bandwidth. Together these determine the QoS (Quality of Service) the flow requires [11]

End-to-End QoS Levels: Service levels refer to the actual end-to-end QoS capabilities, meaning the capability of a network to deliver service needed by specific network traffic from end to end or edge to edge. Three basic levels can be provided across a heterogeneous network [12]:

- Best-effort service (Also known as lack of QoS): Best-effort service is basic connectivity with no guarantees. All users obtain unspecified variable bit rate and delivery time, depending on the current traffic load.

- Differentiated service (also called soft QoS): Some traffic is treated better than the rest (faster handling, more average bandwidth, and lower average loss rate). This is a statistical preference, not a hard and fast guarantee.

- Guaranteed service (also called hard QoS): This is an absolute reservation of network resources for specific traffic.

INTRODUCTION: CONTEXT AND FAILURE MODE

The purpose of this paper is to propose a set of classes of Quality of Service for communications within SpaceWire networks after discussing the needs of distributed data systems based on SpaceWire in the range of best-effort to mission-critical communications. Message passing being the most generic form of communication, it will be our focus here, and hereafter the term 'communication' will in fact refer to message passing.

Obviously, since fault tolerance will be one of the issues discussed in this paper, the domain of events that can be tolerated in the frame of mission-critical communications must be defined. In the context this paper, we consider the ***'Single Fault' hypothesis***: any (non-perfect) component in the system can fail, but no two independent faults will occur within a certain amount of time; this amount of time is sufficient to fully recover from the fault, e.g. by repair or reconfiguration. The expected behaviour for fault tolerance is ***fail-operational***, which means that communications shall not be affected in any way by a (single) failure.

Safety-critical communications are not addressed here as they require considering much more complex failure modes than the 'Single Fault' hypothesis.

GENERAL PROPERTIES OF COMMUNICATION PROTOCOLS

A system is considered ***dependable*** if it is Reliable, Available, and Maintainable (RAM).

A system is considered ***reliable*** if it provides for continuity of correct service. In the frame of the 'Single Fault' hypothesis, we call reliability of a communication protocol its capability to deliver messages from a sender to a group of receivers even in case of failure in this communication.

A system is considered ***available*** if it is always ready for correct service. In the frame of the 'Single Fault' hypothesis, we call availability of a communication protocol its capability to deliver messages from a sender to a group of receivers even in case of failure in a previous communication.

A system is considered ***maintainable*** if it is easy to repair or upgrade. In the case of a communication system, maintainability is not provided by the communication protocol but by the system architecture or by some network management protocol. We will therefore not consider maintainability in the definition of communication QoS classes.

In addition to dependability issues, a system is considered ***safe*** if no error, fault or failure can lead to catastrophic consequences on the user(s) and the environment. Safety cannot be ensured at the level of a communication sub-system, it is a matter of system architecture. We will therefore not consider safety in the definition of communication QoS classes.

Also in addition to dependability issues, a system is considered ***secure*** if it is available for authorized users only and confidential. Security is not yet an issue on board spacecraft. We will therefore not consider security in the definition of communication QoS classes. (it could be added later on).

At last, a system is considered ***real-time*** if it guarantees that messages from a sender to a group of receivers will be delivered before given deadlines whether other communications are competing for network resources or not, based on the hypothesis that deadlines are scheduled in such way that they can be met if no other communication is competing for network resources.

PROPERTIES OF SPACEWIRE-BASED EMBEDDED COMMUNICATION PROTOCOLS

The three parameters that we want to consider for the definition of classes of Quality of Service for SpaceWire communications (reliability, availability and real-time) have different properties so we now discuss each of them separately.

Availability

We can identify three causes that can prevent a SpaceWire network from being available for a given packet to be sent.

The first cause is obviously the case of one of the links along the SpaceWire path not being in the Running state because it is reconnecting after a link failure. Since the link reconnection process is very fast in the SpaceWire protocol, we consider that it is not worth tackling this cause of unavailability because the cost of potential solutions would exceed by far the benefits which can be achieved.

The second cause is a hardware failure either in one of the SpaceWire interfaces (within the sender or the receiver node) or in one of the routers along the SpaceWire path. Such a failure is highly unlikely, due to the simplicity of the hardware and the efforts put into its protection against radiations and its validation. We therefore again consider that it is not worth tackling this cause of unavailability because the cost of potential solutions would exceed by far the benefits which can be achieved.

The third cause is the case of one of the links along the SpaceWire path being blocked by another packet which is not flowing because its receiver node does not pump the packet out of the network (e.g. the receiver's incoming buffer is full; or the receiving application is busy with other tasks than packet reception). This problem opens the door to dramatic reduction of the availability of the network. Besides, it may impact any kind of communication over SpaceWire links, regardless of the QoS class to which they belong. At last, this problem can be solved easily with classical end-to-end ***flow control*** techniques. We therefore believe that end-to-end flow control should be part of the features of any QoS class for SpaceWire communications.

Reliability

Within the frame of the 'Single Fault' hypothesis, a communication protocol can be either fully or partially reliable.

A fully reliable communication protocol guarantees the transfer of a message from source S to destination D, even in the case of a (single) failure. We then call this protocol ***fault-tolerant***. This is e.g. the case when two copies of the same message are sent simultaneously from S and follow different and independent paths to reach D (redundant sub-network).

There are means to increase the reliability of a communication protocol without ensuring full reliability. 'Retry' mechanisms are examples of these. When two copies of the same message are sent from the source S to the destination D following the same path but at different times, and under the same network conditions, the likelihood to see the packet transfer prevented by a link failure is divided by two. In the context of this paper, we will call ***reliability-improved*** a

communication protocol which foresees some mechanism to improve the statistical reliability of the communications.

Timeliness

In the absence of failure (the 'Single Fault' hypothesis being covered by the reliability property), a communication protocol can be either fully real-time (also called "hard real-time") or 'partially' reliable (also called "soft real-time").

In the context this paper, we call ***real-time*** a communication protocol which guarantees that a message posted by a source S will reach its destination D before a given deadline. This is the case e.g. on a communication bus when a time slot is dedicated to the transfer of this message. In a network such as SpaceWire, this is the case e.g. when a time slot is dedicated to the transfer of this message on a given path from S to D and when the travel time of the message is bounded.

There are means to increase the timeliness of a communication protocol without ensuring full respect of delays. 'Priority' mechanisms are examples of these. When messages with different priorities have to share a common path in the network, the message with higher priority is likely to make its way faster than the one with lower priority. In the context of this paper, we will call ***prioritised*** a communication protocol which foresees some mechanism to improve the statistical timeliness of the communications.

Positive acknowledgment

Positive acknowledgment mechanism (the receiver node acknowledges to the sender node the reception of the packet) is not a QoS property as such. It is rather an additional service with respect to message passing. But, as it is usually considered very useful for communication protocol standardisation, we consider it in this paper and artificially include it in the QoS properties. For example, since reliability-improvement techniques do not guarantee packet transfers, they are really useful if accompanied with a positive acknowledgement. Indeed, such feature provides support for higher reliability to be implemented at the upper level (application or higher level protocol).

In the context of this paper, we will call ***acknowledged*** a communication protocol which foresees positive acknowledgement on reception of packets. Additional negative acknowledgement can improve the quality of the communication by letting the source S know that, e.g., the message reached the destination D in due time but was corrupted. This potential additional feature could be discussed later.

DEFINITION OF QOS CLASSES FOR SPACEWIRE-BASED NETWORKS

We have now defined four properties for the communications within SpaceWire-based networks. One of them, availability (through flow control techniques) is considered mandatory and will therefore be part of each and every QoS class. The three remaining can be combined. Among these three properties, one is binary (acknowledged or not) while the two others, namely reliability and timeliness, can also be implemented partially.

As a result, the eighteen possible combinations are the following:

fault-tolerant and real-time: This is nearly the most constrained QoS class we can define. We call this class hereafter ***FTRT service***. It is obviously required for mission-critical control, be it spacecraft avionics or space robotics and is enough within the frame of the 'Single Fault' hypothesis.

fault-tolerant, real-time, and acknowledged: This is the most constrained QoS class we can define. We call this class hereafter ***maximum service***. It would be the best solution for mission-critical control, be it spacecraft avionics or space robotics. It even slightly exceeds the 'Single Fault' hypothesis since the positive acknowledgment would provide support for higher reliability to be implemented at the upper level (application or higher level protocol) in the case of multiple failures (fault-tolerance covering the single fault situation). However, it might be difficult to implement since some acknowledgment mechanisms (e.g. sending a message back) are hindering the performance which is usually expected from real-time communications.

reliability-improved and real-time: This QoS class would be a compromise between the 'real-time and flow control' class (see below) and the FTRT or 'maximum service' class and would serve real-time applications which are not mission-critical but still somehow important. But this type of applications (typically multimedia streaming) has little relevance in the space business. Besides, it would probably be difficult to implement since most common reliability-improvement mechanisms ('retry') are hindering the performance which is usually expected from real-time communications.

reliability-improved, real-time, and acknowledged: Similarly, this QoS class would serve a type of applications (typically multimedia streaming) that has little relevance in the space business and it would probably be difficult to implement since most common acknowledgment mechanisms (sending a message back) would hinder even more the performance which is usually expected from real-time communications.

fault-tolerance: This QoS class would ensure transmission of information but with no guarantee on the delay. This is a theoretical case only because, in practice, there is little difference between a message reaching its destination very late and not reaching it at all. Therefore, this class does not provide much more than the less constrained class 'reliability-improved and flow control' (see below) except lower statistical figures of transmission failure.

fault-tolerant and prioritised: idem.

fault-tolerance and acknowledged: Similarly, this class does not provide more than the less constrained class 'transport-like' (see below) except lower statistical figures of transmission failure.

fault-tolerant, prioritised, and acknowledged: idem.

reliability-improved: This QoS class would be a compromise between the 'minimum service' class (see below) and the 'transport-like' class (see below). But it is not recognised as useful, even for non space applications.

reliability-improved and prioritised: idem.

reliability-improved and acknowledged: This QoS class corresponds to transport protocol layers as they are implemented in non space communication systems. We therefore call it transport-like service. There is no current application of this QoS class in space but it will probably be necessary e.g. to initiate, terminate, and manage sessions if some session-based on-board application is to be developed in the future.

reliability-improved, prioritised, and acknowledged: This QoS class is an extension of the transport-like class that allows prioritising transport messages. We therefore call it ***prioritised transport-like service***. If the implementation of the prioritisation capability is not too costly,

this is certainly preferable to the simple transport-like class since it provides more flexibility for future on-board applications.

real-time: This QoS class would serve non mission-critical real-time applications (typically multimedia streaming) which have little relevance in the space business.

real-time and acknowledged: Similarly, this QoS class would serve non mission-critical real-time applications (typically multimedia streaming) which have little relevance in the space business. Besides, it would be difficult to implement since most acknowledgment mechanisms (sending a message back) are hindering the performance which is usually expected from real-time communications.

no guarantee: This QoS class does not provide anything else than the minimum feature required (availability through flow control) for embedded communications. We therefore call it minimum service.

prioritised: This QoS class is an extension of the minimum service class that allows prioritising transport messages. We therefore call it ***prioritised minimum service***. If the implementation of the prioritisation capability is not too costly, this is certainly preferable to the simple minimum service class since it provides more flexibility for on-board applications.

acknowledged: This QoS class could be a 'light' transport-like class. But there is a general agreement on the fact that, if a transport layer does not guarantee any timeliness property (which is the case of most common transport protocol layers), then the cost of reliability improvement (mostly through 'retry' mechanisms) is low with respect to the expected benefits. A 'real' transport layer is then preferred to a 'light' one.

prioritised and acknowledged: idem.

HINTS FOR IMPLEMENTATION

We have identified three interesting classes of Quality of Service for the communications within SpaceWire-based networks (minimum service, transport-like, and FTRT), and potentially a fourth one (maximum service) if implementation allows. We can now discuss some implementation issues. The FTRT and maximum service QoS classes are certainly the most demanding and therefore would drive any implementation of the other two classes identified earlier.

FTRT/maximum service

The real-time property of the FTRT service can be achieved by reserving an entire SpaceWire path during a given time slot for a given message (one or more packets) to be transmitted from a source node to a destination node, similarly to TTP™/C or FlexRay™. This implies that routers (and potentially the sender and receiver nodes) involved in FTRT or maximum service traffic must include some '***network guardian***' functionality. The necessary time synchronisation can be achieved either through multi-clock synchronisation like in TTP™/C or using SpaceWire time codes.

If such an implementation of FTRT or maximum service classes is chosen, flow control is implicitly provided as long as the receiving (as well as the sending) SpaceWire network interfaces are independent of the application they connect to the network, like in TTP™/C.

Like in TTP™/C, fault-tolerance can be implemented through the duplication of the network links and routers involved in FTRT or maximum service traffic. This duplication is anyway a reality for critical nodes of on-board SpaceWire networks (for FDIR purposes).

By attaching to each FTRT message a CRC code and a list of the last FTRT nodes that have transmitted messages in due time (like C-state frames in TTP™/C), implicit acknowledgment is provided, allowing to upgrade from FTRT class to maximum service class at very little cost.

Moreover, the resulting QoS class would provide the applications with membership knowledge (in addition to the maximum service), i.e. the capability for each node involved in maximum service traffic to know which other nodes are 'alive' and which ones are 'down'.

Prioritised transport-like service

The flow control required by the transport-like QoS class can be implemented through classical flow control mechanisms such as the exchange of 'credit' information, i.e. the amount of data that the receiving 'buffer' is able to store.

Similarly, the reliability improvement and prioritisation required by the transport-like QoS class can be implemented through classical 'retry' and 'priority' mechanisms.

At last, the positive acknowledgment required by the transport-like QoS class can be implemented through classical acknowledgment mechanisms such as the destination node sending an acknowledgment message back to the source node upon successful reception of a message.

Prioritised minimum service

The minimum service QoS class is a subset of the prioritised transport-like class. Its implementation is therefore part of the implementation of the transport-like class (see previous paragraph).

Mixing real-time and non real-time traffic.

The main implementation challenge is to find a technical solution allowing communications belonging to the two groups of QoS classes (FTRT/maximum service vs transport-like/minimum service) to share a SpaceWire network. This can be achieved by allowing the 'network guardians' in the routers (and potentially the sender and receiver nodes) involved in FTRT or maximum service traffic to pre-empt a SpaceWire link over a non real-time packet currently flowing in the link in order to route a real-time packet.

CONCLUSION

In this paper, we have recalled the general properties of digital communication protocols and selected a set of these properties relevant to SpaceWire-based embedded systems in general and to space applications in particular, in the frame of the 'Single Fault' hypothesis. This lead to the definition of four classes of Quality of Service, two non real-time (prioritised minimum service and prioritised transport-like service) and two real-time (FTRT service and maximum service). We have then provided some hints for the implementation of these QoS classes in the frame of SpaceWire networks. While the non real-time classes can clearly be implemented within the frame of the current SpaceWire standard, it appears that implementing real-time classes requires significant additional capability be embedded in some SpaceWire routers and nodes.

REFERENCES

[1] ECSS E50 12A, “SpaceWire. Links, nodes, routers and networks”, 24 January 2003

[2] ECSS E50 12 Part 2 Draft B, “Protocol Identification”, 20 February 2005

[3] ECSS E50 11 Draft E, “Remote Memory Access Protocol”, 21 December 2005

[4] “The Operation and Uses of the SpaceWire Time-Code”, International SpaceWire Seminar, 2003, Steve Parkes, Space Systems Research Group, University of Dundee

[5] “Spacecraft Onboard Interface Services”, Draft Report Concerning Space Data System Standards, CCSDS 850.0-G-0.1. Draft Green Book. Issue 0.1., CCSDS, February 2007.

[6] “Spacecraft Onboard Interface Services—Subnetwork Packet Service", Proposed Draft Recommended Standard, CCSDS 851.0-R-0 Draft Red Book. Issue 0, CCSDS, March 2007.

[7] “Distributed Interrupts in SpaceWire networks”, December 2006, UoStP, Y. Sheynin,

[8] “TCONS Quality of Service White Paper”, 2 September 2005, Steve Parkes, Space Systems Research Group, University of Dundee

[9] SpaceWire Router SpW-10X Data Sheet, 18th August 2006, Chris McClements & Steve Parkes, University of Dundee.

[10] Time-Triggered Architecture (TTA), 25th October 2006, H. Kopetz.

[11] Computer Networks, Fourth Edition, Andrew S. Tanenbaum.

[12] Cisco Systems. Cisco IOS 12.0 Quality of Service. Indianapolis: Cisco Press, 1999.

ISBN: 978-0-9557196-0-8